



Opera Web 标准课程

目录

请注意，迄今为止本课程下的 39 篇文章已经发布，还有 10 多篇文章也将尽快发布，组成完整的课程。

开始篇

1. 序言，作者 Chris Mills，就是您现在正读的这篇。

Web 标准导论

2. 互联网和Web的历史，以及Web标准的演化，作者Mark Norman Francis
3. 互联网是如何运转的？作者Jonathan Lane
4. Web 标准所包括的模块——HTML、CSS 和 JavaScript，作者 Jonathan Lane
5. 梦想虽然很美，但实际上到底怎么样？作者Jonathan Lane

Web 设计概念

这部分并不涉及代码或标记的细节知识，而是在你开始创建任何图形或代码前，帮助你了解 Web 设计过程，以及 Web 设计的一些基本概念，如信息架构、导航、可用性等。

6. 信息架构——规划一个Web站点，作者Jonathan Lane
7. 一个好的网页需要什么？作者Mark Norman Francis
8. 色彩理论，作者Linda Goin
9. 建立站点的线框图，作者Linda Goin
10. 配色方案和设计样板，作者Linda Goin
11. 网页排版，作者Paul Haine

HTML 基础

12. HTML 基础知识，作者Mark Norman Francis
13. HTML 的 <head> 元素，作者Christian Heilmann
14. 为你的 HTML 文档选择适当的文档类型，作者Roger Johansson

HTML 正文

15. 在 HTML 中标记文本内容，作者Mark Norman Francis
16. HTML 列表，作者Ben Buchanan
17. HTML中的图像，作者Christian Heilmann

18. HTML链接——让我们建一张网吧！作者Christian Heilmann
19. HTML表格，作者Jen Hanen
20. HTML表单基础知识，作者Jen Hanen
21. 罕为人知的语义元素，作者Mark Norman Francis
22. 通用容器——div 和span元素，作者Mark Norman Francis
23. 使用导航菜单创建多页面，作者Christian Heilmann
24. 检验你的HTML，作者Mark Norman Francis

可访问性

25. 可访问性基础知识，作者Tom Hughes-Croucher
26. 可访问性测试，作者Benjamin Hawkes-Lewis

CSS 样式

27. CSS 基础知识，作者Christian Heilmann
28. 继承和层叠，作者Tommy Olsson
29. 使用CSS设置文本样式，作者Ben Henick
30. CSS布局模型——boxes、borderes、margins、padding，作者Ben Henick
31. CSS背景图片，作者Nicole Sullivan
32. 样式列表和链接，作者Ben Buchanan
33. 样式表格，作者Ben Buchanan
34. 样式表单，作者Ben Henick
35. 浮动及清除，作者Tommy Olsson
36. CSS的静态和相对定位，作者Tommy Olsson
37. CSS 的绝对和固定定位，作者Tommy Olsson

高级 CSS 研究

38. 页首、页脚、列和模板，作者 Ben Henick

JavaScript 核心技巧

39. 程序设计——真正的基础！作者 Christian Heilmann
40. 你可以用 JavaScript 做什么？，作者 Christian Heilmann
41. 初识 JavaScript，作者 Christian Heilmann

42. [JavaScript 最佳实践](#), 作者 Christian Heilmann (未完成)
43. [非干扰性 JavaScript 的原则](#), 作者 by PPK
44. [JavaScript 函数](#), 作者 Mike West
45. [JavaScript 中的对象](#), 作者 Mike West
46. [DOM 之旅](#), 作者 Mike West
47. [HTML 的创建和修改](#), 作者 Stuart Langridge
48. [动态样式——用 JavaScript 操作 CSS](#), 作者 Greg Schechter
49. [用 JavaScript 处理事件](#), 作者 Robert Nyman
50. [JavaScript 动画](#), 作者 Stuart Langridge
51. [平稳退化对渐进增强](#), 作者 Christian Heilmann

补充文章

- [将你的内容发布到网上](#), 作者 Craig Grannell.
- [有关文档中 <head> 元素的更多知识](#), 作者 Chris Heilmann.
- [补充材料: 用于排版的通用HTML实体](#), 作者 Ben Henick.
- [Opera公司Web标准课程词汇表](#), 由多人撰写。本目录尚不完整, 将随时添加。

Opera Web 标准课程

Posted 12/09/2008 - 18:00 by Csineneo

- 网络标准教程

作者: Chris Mills · 2008 年 7 月 8 日

- 下一篇: 互联网和Web的历史, 以及Web标准的演化
- 目录

序言

长久以来, 我一直有一个梦想。在过去的 8、9 年间, 我主要从事教育工作, 包括委托和编辑技术书籍, 来帮助人们使用技术建立酷炫的东西, 为我所工作过的多家公司培训新员工, 以及编辑和撰写指南性的文章, 帮助人们使用 Opera 的软件。我也很着迷于 Web, 并且是开放的 Web 标准的坚定信徒。我希望通过我所从事的教育和培训工作, 教会人们如何合作, 如何相互尊重, 教会他们如何制作可跨平台、跨浏览器、跨设备访问的 Web 站点 (甚至残疾人也能无障碍地访问), 为改造 Web 世界尽我的一点绵薄之力。要实现这个目标, Web 标准是关键。因此我决定将我的时间和精力集中用于推广 Web 标准的应用, 这是我长久以来一直有的想法, 但终于在 Opera 公司里实现了, 为此我要感谢我的上司聘用我专职从事这项工作, 我的一个梦想终于成为了现实。

在这篇文章里, 我要向大家介绍我和其他很多人花费数月时间开发的一个课程——**Web标准课程**, 该课程旨在向大家提供 Web 设计和开发的坚实基础, 无论读者是谁, 此教程完全免费、可访问, 并且不需要预备知识。当然, 我主要还是希望在大学里推广本课程, 因为我发现很多大学都缺乏好的关于 Web 标准的课程。我听说很多大学生不愿意花时间去学习关于 Web 标准的课程, 因为评分办法已很过时了。我还听说一些公司在面试申请 Web 相关职位的大学毕业生时, 发现这些大学毕业生根本不了解实际的 Web 开发是如何进行的, 而感到非常失望。而如果你以一种理性的风格, 在一所先进的大学里教 Web 标准的话, 那我要向你致敬, 请与我联系!

这篇文章的主要内容包括:

- 为什么要使用 Web 标准? 在这一部分, 我要简单论述使用 Web 标准的优点, 为什么很多时候 Web 标准未得到采用, 以及如何通过学习我们的课程来解决这些问题。
- 本课程的组织结构。这一部分概述课程内容的结构, 并讨论授课老师如何有效地使用和讲述课程材料。

- 本课程的适用人群。当我说“任何人”都可以学习本课程时，这里的“任何人”其确切意义是什么？
- 课程的目录。如果你想直接就开始学习的话，可以跳过这部分。
- 致谢
- [与我联系](#)

为什么要使用 Web 标准？

为什么你一定要在进行 Web 设计和开发时使用 Web 标准，这将在第 4 篇文章中详细阐述。但我在这儿先大概讲一下，让大家有个基本的了解。使用 Web 标准，有以下优点：

1. **提高代码的利用效率：**当你学习完本课程后，你会发现 Web 标准使用的最佳习惯主要是关于代码重用的。通过重用代码，你可以将 HTML 内容和样式信息（CSS）及行为信息（JavaScript）相分离，减小网页体积，而且只需要写一次代码，以后在需要的地方再重用即可。
2. **易于维护：**这一点与上面所说的最后一点密切相关。如果你可以只写一次 HTML 代码，然后在需要应用样式和行为的地方，使用类和函数。而以后，在你需要做出改动的时候，你可以只在一个地方修改，然后此修改就会自动应用到整个 Web 站点，而不再需要一处一处地修改。
3. **提高可访问性：**以下的两点密切相关。Web 上的一个大问题就是，让 Web 站点可以被每个人访问，而不论他们身处何种环境。这包括要能让残疾人，包括盲人、视觉受损的人、和运动功能有障碍的人（如运动受限制，不能自如地用双手或根本不能用双手的人）等）也能访问网站。通过使用 Web 标准和最佳的习惯，你可以使你的 Web 站点能被尽可能多的人访问。
4. **设备兼容性：**此处的兼容性，指的是不仅要确保你的 Web 站点可以跨平台（如 Windows、Mac、Linux），而且要能在其它的浏览设备上工作，比如现在用的手机、电视、游戏控制台等。这些装置在屏幕大小、处理能力、控制机制等方面都有一些局限性。不过你放心，通过使用 Web 标准和最佳的习惯，可以让你的 Web 站点能在绝大多数装置上显示。要知道全球手机的数量多于个人电脑的数量，而且很多手机都是可以上网的，你或你的客户怎能不去占领这个大市场呢。要想了解更多有关移动 Web 开发的信息，请访问 dev.opera.com。
5. **网络爬虫/搜索引擎：**在这里，我指的是所谓的“搜索引擎优化”，即让你的 Web 站点能尽可能多地被网络爬虫爬到，并且被索引，从而提高你的 Web 站点在 Google 等搜索引擎上的排名。这是一项专门的学问（参考关于搜索引擎优化的文章，如《更适用于搜索引擎优化的智能站点结构》、《HTML语义和搜索引擎优化》等）。同样地，通过使用 Web 标准，你就

可以让你的站点在Google、Yahoo! 等搜索引擎上尽量靠前地显示，这必将大大地促进你的业务。

尽管使用 Web 标准有以上诸多优点，但多数 Web 站点至今还尚未遵循 Web 标准，而且时至今日很多 Web 开发人员都还在使用过时的、不好的习惯。你会问“为什么会出现这种情况呢？”。原因当然有很多，这包括缺乏教育和培训、公司政策、不用学习标准也能领到薪水、学习太难、浏览器所支持的标准等。以下我将逐一详细分析这些原因，并批驳那些不采用或学习 Web 标准的借口。

1. **缺乏培训：**这确实是一个问题，这也是我们开发本课程的主要原因之一。很多大学在它们的 Web 相关课程中都不教授 Web 标准，而且很多课程讲述的内容也已经过时，由于官僚主义，这种情况还很难改变。培训课程和书籍确实也花费昂贵，但现在我们已开发出这个免费的课程，并在大学里推广。“缺乏教育和培训”再也不成其为不学习 Web 标准的借口了。
2. **公司政策：**无可否认，至今一些公司/机构依然维持其老式过时的 Web 站点，也许还制定有政策强迫其员工使用过时的浏览器。但现在有了我们这个免费提供的课程，情况应该能得到改观了。将 Web 站点进行升级，以达到当前的标准，这将促进公司升级它们所使用的浏览器，这是因为用过时的浏览器浏览升级后的 Web 站点效果不佳（尽管还是可以浏览）。公司还应该鼓励它们的客户也升级浏览器。这样做是有充分理有的，如上所述，使用了 Web 标准的站点，将有更靠前的搜索引擎排名，并能为更多的人，包括残疾人和使用电脑以外装置上网的人所访问，公司怎能忽视这么大的一个潜在客户群呢？
3. **“我不需要学习 Web 标准”：**我知道一些 Web 开发人员会这样说：“我是在使用过时的方法开发 Web 站点，但我照样领到工资，那我为什么还要去费时费力地学习这个新东西呢？”我在以上部分已经讲过，使用 Web 标准可以提高代码的效率，使得写代码变得更为容易，并使 Web 站点更易于维护。此外，还可以使你能写出可以在电脑以外的设备上显示的代码，这不很好吗？学习 Web 标准，可以提升你的专业技能，并使你能挣到更多的钱。目前很多公司都需要懂 Web 标准的专业人员。
4. **“它太难学了”：**废话。在学习完本课程后，不论你是 Web 开发/设计方面的新手，还是需要进修的 Web 从业人员，都会发现要掌握使用 Web 标准的基本知识，其实是非常容易的。学习使用 Web 标准并不比使用老式过时的 Web 开发/设计方法更难，而且还能带来那么多益处。
5. **浏览器所支持的标准：**过去有很多不同的浏览器支持标准，这使得让 Web 站点在不同的浏览器下都能正常显示，变成了一场恶梦。但那已是过去的事了，当今的浏览器都支持应

有的 Web 标准。一些老式浏览器有时仍需要特别的支持标准，但通过使用当今最好的习惯，你可以确保那些老式浏览器的用户仍然可以获得不错的用户体验。

因此像你们已经了解到的一样，真没有任何借口在从事 Web 开发工作时不采用 Web 标准。从一个初学者的观点来看，学习本课程至少可以让你一开始就学习最好的习惯，而不需要还要费力地去抛弃那些过去不好的习惯。

我们一直以不屑的口吻谈论那些过去不好的习惯，仿佛它们已是毫无用处的老古董了。的确，在本课程中我们不会教授这些不好的习惯，因为我们认为没这个必要。我们认为学员一开始就应选择正确的出发点。也许你们想知道这些过去不好的习惯究竟是怎么样的，下面我就简要介绍一下。

在过去，人们在制作 Web 站点时，习惯于这样做：把整个页面通过一些巨大的表格进行布局，使用不同的表格单元格来定位图像、文本等（不是真正的表格，而是向页面添加冗余的标记）。他们习惯于使用不可见的空白 GIF 来调整页面元素的定位（不是真正的 GIF 图像，而是向页面添加冗余的标记）。他们习惯于用 JavaScript 来生成那些飞舞的菜单（这对于那些已经在浏览器中禁用 JavaScript 的人来说，一点好处也没有，而对那些因视觉障碍而使用屏幕阅读器的人来说，也会被这些 JavaScript 搞晕）。或者，写一些仅能运行在一种浏览器上的 JavaScript（但对于那些使用其他浏览器的人，该怎么办呢？）。他们习惯于在 HTML 的``元素中，直接插入样式信息（但这将增加维护的难度，并且会使页面中增添多余的标记）。此外还有很多不好的习惯，最糟糕的是，虽然我上面说的是“过去的作法”，但现在有很多人还像以前一样做。

Web 开发本身就是一件杂乱无章的工作，而如果再没有好的开发习惯，它就难上加难了。本课程所描述的 Web 标准和最佳习惯，将是你从事 Web 开发的最佳途径。

课程结构

本课程由很多文章构成。在基础课程完成时，将有 50 多篇文章，每篇文章的长度都为几千个单词。每篇文章的内容都集中在一个具体的主题上，一般将包括以下内容：该主题的背景知识、基本理论、实际例子、漫游教程，以及练习题。

此外，我们在未来还将推出全套教程，内容将覆盖构建 Web 站点的全过程。

教授本课程适当的方式为：先确定你有多少堂课可以用于将本课程教完，然后除以文章的数量。在每堂课开始前，要求学员先通读与该堂课相对应的文章，然后在课堂上讨论实际的例子，并要求学员在课后做练习题。我认为只要让学员在课前通读了文章，讲述每篇文章所包含的基本理念只需要 1 个小时就足够了。大致说来，本课程需要 50 小时的授课时间，另需要 50 个小时的背景阅读时间。

很显然，你需要仔细想想教授完本课程需要多长时间，每堂课又需要具体讲述些什么。可以通过试讲摸索出路子。

授课对象

这是一个有关 Web 标准的课程，由多篇文章组成，其主要授课对象为任何想从零开始学习基于 Web 标准的 Web（网页）设计的人。本课程的目标为：使一开始仅懂如何浏览网页的人，通过本课程的学习，熟练掌握 CSS 样式和 HTML 语言，并了解 JavaScript 的基本知识及实际应用。通过学习，将可以使学员获得进入工作市场所需要的足够知识（当然工作经验是无法教授的，需要自己积累）。

在我看来，授课对象是希望以“正确的方式”学习 Web 设计的任何人，包括：

1. **大学/学院学生和老师：**正如我已经指出的那样，老师既可以利用本课程的全套文章创建自己的课程，并向学生讲授；也可以将其中部分文章作为你自己课程的补充。而任何已经学过一些 Web 相关课程的学生，应当使用本课程的材料补充自己的知识，并劝说自己的老师也这样做。我建议老师也把这些材料通读一遍，以确保所教授的课程中含盖的技术是当前的最佳规范。
2. **大学预科/大学年龄的学生：**尽管本课程主要是面向成人的，但中学生还是可以从学习本课程中受益，可以尝试学习本课程。
3. **在职的 Web 设计者和开发者：**有很多在职的 Web 设计者和开发者，尚未在工作中使用 Web 标准和最佳习惯，或是不能很容易就找到可用的参考资料，或不习惯更新自己的知识。我力劝前一类人一定学习一下本课程，以了解在工作中采用 Web 标准是多么有价值且又是多么容易。而对后一类人来说，我确信你们学习本课程也会有很大收获，可以更新自己的知识和技能，温习容易遗忘的知识，并使你获得足够的知识储备，可用于说服你的雇主和客户像 Web 站点的可访问性这类要素有多么重要。
4. **公司内部的培训人员：**本课程是以低成本对员工进行培训的理想方式。
5. **其他个人：**如果你是想学习 Web 设计和开发的个人，本课程也是你以低成本接受教育和培训的理想方式。

本课程是依据知识共享许可协议而发行的，任何想使用的人都可以免费使用，只要承认本课程归属于我们即可。

目录

请注意，迄今为止本课程下的 39 篇文章已经发布，还有 10 多篇文章也将尽快发布，组成完整的课程。

开始篇

1. 序言，作者 Chris Mills，就是您现在正读的这篇。

Web 标准导论

2. 互联网和Web的历史，以及Web标准的演化，作者Mark Norman Francis
3. 互联网是如何运转的？作者Jonathan Lane
4. Web 标准所包括的模块——HTML、CSS 和 JavaScript，作者 Jonathan Lane
5. 梦想虽然很美，但实际上到底怎么样？作者Jonathan Lane

Web 设计概念

这部分并不涉及代码或标记的细节知识，而是在你开始创建任何图形或代码前，帮助你了解 Web 设计过程，以及 Web 设计的一些基本概念，如信息架构、导航、可用性等。

6. 信息架构——规划一个Web站点，作者Jonathan Lane
7. 一个好的网页需要什么？作者Mark Norman Francis
8. 色彩理论，作者Linda Goin
9. 建立站点的线框图，作者Linda Goin
10. 配色方案和设计样板，作者Linda Goin
11. 网页排版，作者Paul Haine

HTML 基础

12. HTML 基础知识，作者Mark Norman Francis
13. HTML 的 <head> 元素，作者Christian Heilmann
14. 为你的 HTML 文档选择适当的文档类型，作者Roger Johansson

HTML 正文

15. 在 HTML 中标记文本内容，作者Mark Norman Francis
16. HTML 列表，作者Ben Buchanan
17. HTML中的图像，作者Christian Heilmann
18. HTML链接——让我们建一张网吧！作者Christian Heilmann
19. HTML表格，作者Jen Hanen
20. HTML表单基础知识，作者Jen Hanen

21. 罕为人知的语义元素，作者Mark Norman Francis
22. 通用容器——div 和span元素，作者Mark Norman Francis
23. 使用导航菜单创建多页面，作者Christian Heilmann
24. 检验你的HTML，作者Mark Norman Francis

可访问性

25. 可访问性基础知识，作者Tom Hughes-Croucher
26. 可访问性测试，作者Benjamin Hawkes-Lewis

CSS 样式

27. CSS 基础知识，作者Christian Heilmann
28. 继承和层叠，作者Tommy Olsson
29. 使用CSS设置文本样式，作者Ben Henick
30. CSS布局模型——boxes、borderes、margins、padding，作者Ben Henick
31. CSS背景图片，作者Nicole Sullivan
32. 样式列表和链接，作者Ben Buchanan
33. 样式表格，作者Ben Buchanan
34. 样式表单，作者Ben Henick
35. 浮动及清除，作者Tommy Olsson
36. CSS的静态和相对定位，作者Tommy Olsson
37. CSS 的绝对和固定定位，作者Tommy Olsson

高级 CSS 研究

38. 页首、页脚、列和模板，作者 Ben Henick

JavaScript 核心技巧

39. 程序设计——真正的基础！作者 Christian Heilmann
40. 你可以用 JavaScript 做什么？，作者 Christian Heilmann
41. 初识 JavaScript，作者 Christian Heilmann
42. JavaScript 最佳实践，作者 Christian Heilmann (未完成)
43. 非干扰性 JavaScript 的原则，作者 by PPK
44. JavaScript 函数，作者 Mike West

45. [JavaScript 中的对象](#), 作者 Mike West
46. [DOM 之旅](#), 作者 Mike West
47. [HTML 的创建和修改](#), 作者 Stuart Langridge
48. [动态样式——用 JavaScript 操作 CSS](#), 作者 Greg Schechter
49. [用 JavaScript 处理事件](#), 作者 Robert Nyman
50. [JavaScript 动画](#), 作者 Stuart Langridge
51. [平稳退化对渐进增强](#), 作者 Christian Heilmann

补充文章

- [将你的内容发布到网上](#), 作者 Craig Grannell.
- [有关文档中 <head> 元素的更多知识](#), 作者 Chris Heilmann.
- [补充材料: 用于排版的通用HTML实体](#), 作者 Ben Henick.
- [Opera公司Web标准课程词汇表](#), 由多人撰写。本目录尚不完整, 将随时添加。

致谢

在开发本课程的过程中, 有非常多的人曾给予我帮助, 也许无法一一列出他们的名字, 但我希望在这里能已列出他们每个人的名字。他们都是一些杰出的人, 我建议你们去听他们的讲座、买他们写的书、阅读他们的博客, 或者以其他方式给予他们支持。我向你们致以感谢和敬意。我要感谢:

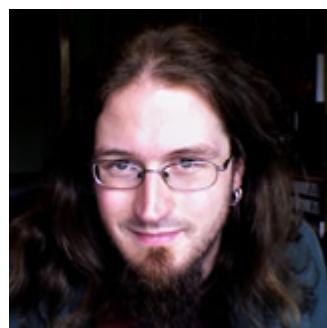
1. **文章的作者:** 非常感谢 Ben Buchanan, Tom Hughes-Croucher, Mark Norman “Norm” Francis, Linda Goin, Paul Haine, Jen Hanen, Benjamin Hawkes-Lewis, Ben Henick, Christian Heilmann, Roger Johansson, Peter-Paul Koch, Jonathan Lane, Tommy Olsson, Nicole Sullivan, Mike West。没有你们的帮助, 本课程是不可能被编写出来的。
2. **Opera 的同事:** 包括 Jan Standal, David Storey, 我手下团队的其他成员, 以及其他所有帮助我开发本课程的同事。
3. **机构:** 感谢 Yahoo 公司的每一位员工 (文章作者, 以及 Sophie Major, 她在组织和推广方面给了我很大帮助)。感谢 WaSP 公司(特别是 Gareth Rushgrove, Stephanie Troeth 和 Aarron Walter), Britpack 网站, Geekup 网页设计师联盟, 以及关注本课程的所有大学。
4. **个人:** 衷心感谢以下人士——Craig Saila, Sara Dodd, John Allsopp, Roan Lavery, Bruce Lawson, Alan White。如果我遗漏了谁, 请原谅我。
5. **读者:** 感谢你们对以正确的方式创建 Web 站点感兴趣, 并抽出时间阅读本课程的材料。

与我联系

我始终希望尽可能多的人能给我提供意见和建议，以让我能改进本课程。如果你对如何改进本课程有任何建议，或是有任何评论，或是希望某个地方推广本课程，都请与我联系。我的 E-mail 是：cmills@opera.com。 你也可以点击每篇文章页首的链接（“讨论本篇文章”），发表你对文章的评论。为参与讨论，你需要一个 [my.opera](#) 账户。

- [下一篇：互联网和Web的历史，以及Web标准的演化](#)
- [目录](#)

作者简介



Chris Mills是Opera公司的开发人员关系部经理，他在 dev.opera.com 和 labs.opera.com 这两个站点上编辑和发表文章，负责与社区社会团体联系，以提高Opera公司的知名度并收集反馈信息，并负责推广Opera公司的软件。同时，他也是Web标准课程的发起人和编辑。

在工作之余，Chris Mills是个狂热的音乐爱好者，喜欢演奏和欣赏各种类型的音乐，如重金属音乐、民谣、朋克、电子音乐等。目前他最喜欢的乐队是英国的重金属乐队 **Conquest of Steel**（征服钢铁）。

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

2. 互联网和 Web 的历史，以及 Web 标准的演化

Posted 12/10/2008 - 12:37 by Lewis

- 网络标准教程

作者: mnfrancis · 2008 年 7 月 8 日

- 上一篇: [Opera Web 标准课程](#)
- 下一篇: [互联网是如何工作的?](#)
- [目录](#)

序言

“请问陛下，我们该从哪里出发呢？”

“从起点出发”，国王严肃地回答说：“一直前行，直到终点再停下。”

—刘易斯•卡罗尔《爱丽丝漫游奇境》

我们做任何事都有一个起点，我们这个课程就从回顾历史开始吧。在以下部分，我将简要回顾互联网、万维网（World Wide Web），以及Web标准（我们课程的主题）的历史和演化过程。我认为了解互联网和Web标准的演化是很有用的，也很有趣。我的介绍很简要，不是长篇大论，可帮助你快速入门。如果我提到的任何术语你还不熟悉的话，请不用担心。如果这些术语是学习 Web 开发过程中很重要的术语，则会在后面的专题文章中给出定义的，此外你也可用 Google 查询它们的含义。如果你已经很熟悉互联网和万维网的历史，可以直接跳到 Web 标准的部分。本篇文章的内容目录如下：

- 互联网的起源
- 万维网的诞生
 - 浏览器之争
- Web标准的诞生
 - W3C（万维网联盟）的成立
 - Web标准组织
 - Web标准的兴起
- 总结
- 延伸阅读

- 练习题

互联网的起源

1957 年 10 月 4 日，一个改变世界的事件发生了，那就是苏联成功地将第一颗人造卫星（名为 **Sputnik 1 号**）送入了地球轨道。这件事震惊了世界，尤其是让美国感到十分震惊，因为美国也正在进行发射人造卫星的计划，但尚未完成。

受到苏联首先成功发射人造卫星的刺激，美国政府认为有必要成立一个研究和开发新的尖端科技的机构，于是就成立了国防部高级研究计划局（ARPA）。迄今为止，国防部高级研究计划局最有名的（无疑是得到最广泛应用的）研究计划也许就是建立互联网了。

1960 年，心理学家和计算机科学家 **Joseph Licklider** 发表了一篇名为《人机共生》的论文，在这篇论文中首次提出将计算机联网，以用于信息的储存和检索的构想。1962 年，时任国防部高级研究计划局信息处理办公室负责人的 **Joseph Licklider**，建立了一个推进计算机研究的小组，中心工作就是围绕他 1960 年提出的将计算机联网的构想开展研究。

1967 年 10 月，研究小组提出了构建这样一个计算机网络，被称为“阿帕网”（ARPANET）的计划，随后在 1969 年 12 月，首个连接四个节点的计算机网络建立并投入运行。建立一个计算机网络面临的关键问题是如何将分散的计算机系统连接起来，又不因不间断连接而耗尽网络资源。为解除这一问题，发明出了一种叫做“分组交换（包交换）”的技术，即把数据请求拆分为小的“信息包”，这些信息包可以得到快速地处理，且不阻塞与其他方的通信，目前的互联网依然在使用这一技术原理。

分组交换技术自问世后得到了广泛的应用，一些在阿帕网之后新构建的网络都采用了分组交换技术。例如，英国第一个大学间的网络 **JANET**，及美国的公共网络 **CompuServe**，都是建立在 **X.25 分组交换技术**之上，此技术由国际电信联盟开发。前者使英国的大学之间可以收发文档和电子邮件，而后者做为一家商业机构，允许小公司和个人访问分时共享的计算机资源，并且后来被接入互联网。这些网络尽管有很多连接，但实质上都还是专用网络，而不是我们今天使用的互联网。

当人们试图让所有这些分散的网络都能相互通信时，存在多种不同的网络协议就成为了一个问题。不过问题的解决方案很快就找到了，**Robert Kahn** 在从事国防部高级研究计划局的一个卫星分组网络研究项目时，开始为更开放的网络架构定义一些规则，以取代阿帕网当时使用的网络协议。随后斯坦福大学的 **Vinton Cerf** 加入了他的研究项目。他们两人创建了一个使用新的标准弥合不同网络协议间的差异的新系统。在 1974 年 12 月发表的草拟规范中，他们两人将这称为“互联网传输控制程序”。

这个传输控制规范降低了网络所起的作用，而把维持传输完整性的职责交给了主机，其最终结果就是使将几乎所有网络都连接起来这一工作成为可能。国防部高级研究计划局资助了相关软件的开发，然后在 1977 年，成功进行了三种不同的网络通信的演示。到 1981 年，传输控制规范最终确定和发布，并得到采用。1982 年，在美国以外的阿帕网连接都被转变使用新的 TCP/IP 协议，互联网诞生了。

万维网的诞生

Gopher 是 20 世纪 90 年代初期常用的信息检索系统，它提供了一种向文件、计算机资源和其他菜单添加链接菜单的方法。这些菜单可越过本机的界限，并使用互联网从其他计算机系统抓取菜单。**Gopher**深受那些希望共享全校信息的大学和希望集中储存和管理文件的大型组织的欢迎。

Gopher 是由明尼苏达大学首创的。1993 年 2 月，明尼苏达大学宣布将针对使用 **Gopher** 服务器收取许可费，这样很多组织就开始寻找替代 **Gopher** 的系统。

设在瑞士的欧洲核研究理事会(CERN)就有这样一个替代系统。该组织的 Tim Berners-Lee 当时正在开发一个信息管理系统，在该系统中，文本可包含指向其他文件的链接，这样读者就可以在文档间快速跳转。他创建了一个发布这种被称为“超文本”格式文档的服务器，以及阅读这些超文本文档的程序，他称它们为“万维网”(WorldWideWeb)。这个软件于 1991 年首次发布，不过在经历了两个事件之后，它才得到极其广泛的普及，并最终取代了 **Gopher**。

1993 年 4 月 13 日，欧洲核研究理事会公开了万维网的源代码，这样任何人都可以使用或改进这个软件，且不需要付费。

同一年的晚些时候，超级计算机应用国家中心(NCSA)发布了一个集成 Web 浏览器和 **Gopher** 客户端的程序，被称为 **Mosaic**。最初这个程序只能在 Unix 机器上以源代码的方式获得，不过到了 1993 年的 12 月，**Mosaic** 发布了可在苹果机和 Windows 计算机上安装和运行的新版本。选用 **Mosaic** 浏览器的用户激增，同时上网的人也越来越多。

可用的 Web 浏览器数量戏剧性地增长，很多浏览器都是由大学和公司开发出来的，例如，一家挪威的通信公司——Telenor，就于 1994 年开发出了第一个版本的 Opera 浏览器。

浏览器之争

Web 的普及带来了商业利益，Marc Andreessen 离开了超级计算机应用国家中心，和 Jim Clark 一道创立了 **Mosaic** 通信公司，后来改名为网景(Netscape)通信公司。他们开发出 **Netscape Navigator** 浏览器，并于 1994 年 12 月发布了该浏览器的 1.0 版本。

Spyglass 有限公司是 NCSA 的下属商业部门，它将 **Mosaic** 浏览器技术许可给微软使用，**Mosaic** 浏览器技术构成了微软 Internet Explorer 浏览器的基础。1995 年 8 月，微软发布了

Internet Explorer 浏览器的 1.0 版本。

此后微软和网景都不断推出各自浏览器的升级版本，都想在浏览器支持的功能方面赢得竞争优势，以吸引 Web 开发人员。这被称为“浏览器之争”。同一时期，Opera 浏览器占有的市场份额虽然还很小，但已有一批忠实的用户。同时，Opera 公司也在尽力创新和支持 Web 标准。

Web 标准的诞生

在浏览器之争中，微软和网景都将重点放在实施新功能上，而不是放在解决他们各自的浏览器已经支持的功能所出现的问题上。他们还着重于向各自的浏览器增添专有功能，并创建与其他浏览器已有的功能相直接竞争，但不具有兼容性的功能。

面临这种浏览器之争造成的混乱，Web 开发人员在构建 Web 站点时，不得不采取一些权宜之计，一些时候是分别针对这两个主流浏览器开发两个版本的 Web 站点，而有时干脆就选择只支持一个浏览器，使使用其他浏览器的用户无法正常访问 Web 站点。这是一种可怕的工作方式，相应地，Web 开发人员也开始寻求解决的办法。

W3C（万维网联盟）的成立

1994 年，在欧洲核研究理事会、国防部高级研究计划局和欧洲委员会的支持下，Tim Berners-Lee 在麻省理工学院创立了 W3C（万维网联盟）。W3C 的目标是：规范用于创建 Web 站点和 Web 页的协议和技术，以使 Web 站点和 Web 页的内容能为全球尽可能多的人访问。

此后的数年内，W3C 发布了多份规范文件（称为“建议”），包括 HTML 4.0、PNG 图像格式、CSS 样式表的 1.0 和 2.0 版本。

但是 W3C 并未强制执行其提出的建议，生产厂商只有在希望在产品上标注“符合 W3C 规范”的情况下，才必须遵守 W3C 制定的规范。这不是一个可以吸引消费者的产品特色，因为几乎所有的互联网用户都不知道，而且可能也不关心 W3C 是个什么样的组织。浏览器之争继续激烈地进行着。

Web 标准组织

1998 年，浏览器市场被当时的两大主流浏览器 Internet Explorer 4 和 Netscape Navigator 4 所占据。微软发布了 Internet Explorer 5 的测试版本，该测试版本的 Internet Explorer 5 执行一种新的、专有的动态 HTML，这意味着专业的 Web 开发人员需要知道 5 种不同的写 JavaScript 的方法。

在这种情况下，一个由专业 Web 开发人员和设计师组成的团体应运而生，这个团体称自己为 Web 标准组织（WaSP），他们的想法是通过呼吁将 W3C 制定的规范称为标准而不是建议，可能可以说服微软和网景支持他们。

Web 标准组织为推广他们的行动呼吁，在早期阶段采用了一种传统的广告技术——“路障策略”（**roadblock**）。所谓“路障策略”，是指一家公司在同一时段，在所有的电视频道上都播出一个同样的广告，这样观众无论将电视调到哪一个频道，得到的都是完全相同的广告讯息。借鉴这个策略，**Web** 标准组织同时在很多 **Web** 开发专业站点（包括 [builder.com](#)），连线杂志网络版（[Wired online](#)），以及一些受欢迎的邮件列表上，发布同一篇文章。

Web 标准组织使用的另一个宣传技术是嘲笑那些将要加入 **W3C**（及其他标准制定组织）的公司，不过后来就更侧重于创造新特性，而不再侧重于推广那些他们认为一开始就必须使用的基本方法。

Web 标准组织的所作所为似乎更偏重于批评，但他们并不只是坐在那里批评人，他们也向人们提供帮助。**Web** 标准组织的 7 位成员成立了一个“**CSS 武士团**”（**CSS Samurai**），明确指出 **Opera** 浏览器和 **Internet Explorer** 浏览器在支持 **CSS** 方面存在的十大问题（**Opera** 公司解决了这些问题，但微软并未解决）。

Web 标准的兴起

2000 年，微软发布了 **Internet Explorer 5** 浏览器的苹果机版本，这是一个非常重要的里程碑事件，**Internet Explorer** 浏览器成为 **Mac OS** 操作系统下的默认浏览器，并在一定程度上支持了 **W3C** 建议。这一事件，以及 **Opera** 浏览器当时已可以很好地支持 **CSS** 和 **HTML** 这一事实，推动了 **Web** 标准的使用，**Web** 开发人员和设计师第一次感到使用 **Web** 标准设计站点是轻松的工作。

Web 标准组织劝说网景公司在其浏览器完全支持 **Web** 标准前，不发布 **Netscape Navigator** 浏览器的 5.0 版本（这项工作为当前非常流行的浏览器 **Firefox** 的推出奠定了基础）。**Web** 标准组织还创建了一个“**Dreamweaver** 专责小组”，促请 **Macromedia** 公司改进其流行的网页制作工具 **Dreamweaver**，以鼓励和支持创建符合 **Web** 标准的站点。

很受欢迎的 **Web** 开发站点“**A List Apart**”，于 2001 年初进行了重新设计，在一篇文章中他们阐述了为什么要重新设计 **Web** 站点：

“在 6 个月内，或 1 年内，最多两年内，所有的站点在设计时都将使用这些 **Web** 标准。[中间略过……] 我们要么不思进取，听任自己落伍，要么现在就开始学习基于 **Web** 标准的技术。”

这有些过于乐观了。即使是到了 2008 年，也不是所有站点都是使用 **Web** 标准而构建的。不过发展趋势是可以让人乐观的，旧式浏览器所占的市场份额已经下降。两个著名的 **Web** 站点已使用 **Web** 标准对其站点进行了重新设计，它们就是连线杂志网络版（于 2002 年重新设计），和 **ESPN** 电视网的站点（于 2003 年重新设计），它们已成为支持 **Web** 标准和新技术的先锋。

同样也是在 2003 年，Dave Shea 推出了一个被称作“CSS Zen Garden”（CSS 禅意花园）的站点，这个站点用实际例子证明仅通过改变页面的样式，而内容保持不变，就可以实现整个设计的改变，这对 Web 专业人士产生了莫大的影响。

从那时起，在专业 Web 开发圈子里，Web 标准就已成为必须遵守的标准。在本课程中，我们将向大家全面介绍 Web 标准，向大家提供 Web 设计和 Web 开发的牢固基础，使你们能开发出符合 Web 标准的、可访问性好的，且代码精简高效的 Web 站点，如同大公司的站点一样。

总结

在这篇文章里，我简述了现代互联网是如何诞生的，即最初它实际上是美苏太空军备竞赛的产物；Tim Berners-Lee 是如何为一代人定义超文本的，以及微软和网景的商业之争，是如何导致 Web 开发人员群体发起推广 Web 标准的行动的。“Web 标准”这一术语现在已广泛地在 Web 专业人士圈子里使用，连 W3C 在其站点也开始使用这一术语。我们要向你们教授的正是 Web 标准，即建立 Web 站点的标准方式。

延伸阅读

如果你想了解更多的知识，可以访问以下一些站点：

- 互联网的历史（维基百科[wikipedia](#)）
- 万维网的历史（维基百科[wikipedia](#)）
- W3C的历史
- Web标准组织，及其历史
- A List Apart站点
- CSS Zen Garden站点

练习题

- Windows 平台、Mac OS X 平台、Linux 平台各自的用户，目前可用的互联网浏览器有哪些？
- 每种浏览器的用户比例分别是多少？
- 移动设备访问网页使用哪些浏览器？
- W3C 已发布了多少个“Web 标准”，其中有哪些目前得到浏览器开发商的广泛支持？
- 上一篇：[Opera Web 标准课程](#)
- 下一篇：[互联网是如何工作的？](#)
- 目录

作者简介



Mark Norman Francis 早在万维网诞生前，就在从事互联网领域的工作了，一直持续到现在。目前他是全球最大网站 Yahoo! 的前端设计师，负责制定 Web 开发的最佳习惯、代码标准和质量标准。

在加入 Yahoo! 前，他先后在 Formula One Management (F-1 管理公司)、Purple Interactive (紫色互动公司)、伦敦城市大学从事过多种工作，包括 Web 开发、后端 CGI 编程和系统架构等。他的博客地址为：<http://marknormanfrancis.com/>。

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

3. 互联网是如何运转的?

Posted 12/10/2008 - 15:29 by Lewis

- 网络标准教程

作者: Jonathan Lane · 2008 年 7 月 8 日

- 上一篇: 互联网和Web的历史, 以及Web标准的演化
- 下一篇: Web 标准所包括的模块——HTML、CSS 和 JavaScript
- 目录

序言

偶尔你们会有机会了解一件事情的来龙去脉和其中的内幕, 今天你就很幸运, 因为我将带你领略当今最热门, 同时也可能是你已经熟悉了的技术: 万维网。

本篇文章讲述万维网的基本技术, 包括:

- 超文本标记语言 (HTML)
- 超文本传输协议 (HTTP)
- 域名解析系统 (DNS)
- Web 服务器和 Web 浏览器
- 静态和动态内容

以上这些内容都是一些最基本的知识, 尽管它们对你构建一个更好的 Web 站点没有太大帮助, 但却有助于你使用专业的语言向客户和其他人讲述 Web。正如电影《音乐之声》里那个睿智的保姆所说的: “当我们学习阅读时, 从 ABC 开始; 而当学习唱歌时, 从 Do Re Mi 开始”。在本篇文章中, 我将讲述计算机是如何使用 HTTP 和 TCP/IP 协议在互联网上通信的, 然后再讲述创建网页需要的各种不同语言。本章的内容如下:

- 计算机如何通过互联网通信?
 - 剖析请求/响应周期
- 内容的类型
 - 纯文本
 - Web标准
 - 动态网页

- 需要其它程序或插件的格式
- 静态网站与动态网站的比较
- 总结
- 练习题

计算机如何通过互联网通信？

值得庆幸的是，我们使计算机易于使用。在万维网上，绝大多数网页都是使用同一种语言，即 **HTML**（超文本标记语言）编写的，并且使用一种通用的协议，即 **HTTP**（超文本传输协议）进行传输的。**HTTP** 是常用的互联网规范，它允许一台 **Window** 计算机和正在运行着最新版本 **Linux** 操作系统的计算机实现无障碍的通信。通过使用 **Web** 浏览器（它是一种解释 **HTTP**，并将 **HTML** 渲染为人类可读格式的软件），你就可以在任意地点、任意设备上（包括手机、**PDA**、游戏机等），阅读在任意计算机上以 **HTML** 创作的网页。

尽管编写网页的语言相同，但接入互联网的各种设备要能和网络相互通信，却需要有一些规则。这和在教室里学习，问问题要举手的规矩一样，**HTTP** 为互联网确立了这些基本规则。由于有了 **HTTP**，一台客户计算机（如你的计算机）就知道，它就是那台向**服务器**发出某个网页访问请求的计算机。服务器就是存放 **Web** 站点的计算机。当你在浏览器中键入一个网址，服务器就会收到你的请求，找到你想访问的网页，然后将该网页发回到你的计算机，这时网页就会在你的浏览器上显示出来了。

剖析请求/响应周期

以上我已讲述了使计算机能在互联网上通信所需的几个部分，现在我再详细谈谈 **HTTP** 的请求/响应周期。为能更有效地掌握一些重要的概念，请遵照以下步骤动手操作一下。

1. 每一次请求/响应，都从在 **Web** 浏览器的地址栏上键入统一资源定位符（**URL**）开始，比如 <http://dev.opera.com>。请打开浏览器，键入一个 **URL**。

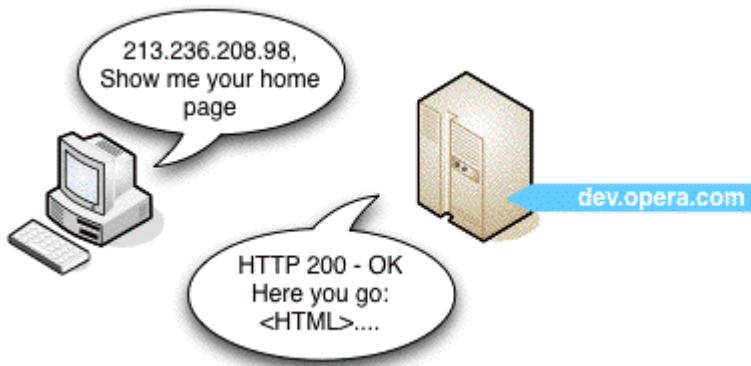
也许你们还不知道，**Web** 浏览器实际上并不是使用 **URL** 向服务器请求 **Web** 站点的，而是使用**互联网协议**或称**IP 地址**（它们和电话号码或邮政编码相似，是用来识别服务器的）。举例来说，<http://dev.opera.com> 这个站点的 IP 地址就是 213.236.208.98。

2. 请打开一个新的浏览器标签页或窗口，键入 <http://www.apple.com>，然后按回车键；然后再键入 <http://17.149.160.10/> 并按回车键，你会发现到的是同一个**Web**站点。试着在地址栏键入 <http://213.236.208.98> 并按回车键，这其实也是指向一个服务器地址，但你的浏览器上会显示“禁止访问”错误信息（错误 403），这是因为你未被允许访问这个服务器的真实根目录。

<http://www.apple.com> 基本上等于是 <http://17.149.160.10/> 的别名，但为什么要使用这个别名呢？这是因为人们更容易记住文字，而不是一长串数字。将字母组成的域名解析为数字表示的IP地址，是通过域名解析系统（DNS）实现的。DNS实际上是所有连接到互联网上计算机的自动目录。当你在浏览器的地址栏键入 <http://dev.opera.com> 并按回车键后，该网址就被发送到一个域名服务器上，由这个服务器去找到相对应的IP地址。由于连接到互联网的计算机数量非常庞大，不是每一台DNS服务器都有一个所有在线机器的列表，因此就有这样一个系统，确保你的请求可以发往正确的服务器，以完成你的请求。

因此DNS系统就查找 www.opera.com 站点，找到所请求页面的IP地址 17.149.160.10，然后把这个IP地址发送回你的Web浏览器。

你的计算机向位于指定 IP 地址的计算机发出请求，等待得到响应。如果一切顺利的话，服务器会向客户端发出一条消息说“一切都正常”（见图 1），然后将网页发送到客户端。这类消息被包含在 **HTTP 报头** 中。



图一：在这种情况下，一切都正常，服务器传输回正确的网页。

如果出现了问题，例如你键入的 URL 不正确，则会传输回 **HTTP 错误信息**，并显示在你的浏览器上，错误 404“网页未找到”，就是你可能遇到的常见错误信息。

3. 试着键入 <http://dev.opera.com/joniscool.html>，这个网页不存在，因此你的浏览器上就会显示服务器传输回的错误 404 信息。试试再键入几个不同的 Web 站点上不存在的网页的网址，你会发现传输回来的 404 错误页面各有不同。这是因为一些 Web 开发员只是让 Web 服务器传输回默认的错误页面，而另一些 Web 开发员则编写了自定义的错误页面。这是一项高级技术，未包含在本课程的范围内，不过我们希望在即将发布的一篇文章中包含这一内容。

最后，说一下有关 URL 的事情，通常你键入的指向某个站点的第一个 URL，在其末尾是没有真正的文件名的（如 <http://www.mysite.com/>），随后你访问的页面，在其 URL 的末尾有

些有文件名，有些则没有。其实你访问的都是真正的文件，只是有时 Web 开发员会通过在 Web 服务器上的设置，让服务器不在 URL 中显示文件名，这样做通常可以让人们更容易记住 URL，使访问站点的用户获得更好的体验。这是一项高级技术，也未包含在本课程的范围内。我们将在后面的文章中，讲述向服务器上载文件以及文件/文件夹目录结构的内容。

内容的类型

以上我已为大家讲述了 HTTP 请求/响应，现在我将讲述你在互联网上可以看到的不同内容。我把内容划分为四大类：纯文本、Web 标准、动态网页，及需要其它程序或插件的格式。

纯文本

在互联网发展的早期阶段，那时还没有任何 Web 标准或插件，互联网上主要就是图像和纯文本（扩展名为.txt 的文件）。当一个纯文本文件被放到互联网上，浏览器只是按原样显示，不进行任何处理。现在大学的站点上，你常常都还能看到纯文本文件。

Web 标准

构成万维网的基石就是 3 个主要的 Web 标准：HTML（或 XHTML——可扩展超文本标记语言，在本文中我将交替使用 HTML 和 XHTML）、CSS、JavaScript。

对于沟通这个用途来说，超文本标记语言是个恰如其分的名字。HTML 用于将文档划分为不同的部分，规定文档的内容和结构，并定义每部分的含义（它包含你在网站看到的所有文本等内容），同时，它使用元素来标识页面中的不同部分。

层叠样式表（CSS）使你可以完全控制一个 HTML 元素如何被显示。CSS 很简单，例如，使用样式表声明，可将所有段落改为双倍行距（`line-height (行高) : 2em;`），或让所有二级标题变为绿色（`color: green;`）。将页面结构从页面格式中分离出来，会带来非常多的好处，在下一篇文中我们将详细地讲述。为示范将 HTML 和 CSS 结合起来使用所带来的效果，请看图 2，在图 2 的左边显示的是一些纯 HTML，未添加任何格式设定；而右边显示的是应用了一些 CSS 样式后完全相同的 HTML。

Example page to show CSS styling

Web browsers will apply some basic formatting to an HTML document without any style declarations.

CSS ability

Cascading Style Sheets allow you to control the appearance of any element within your HTML document. You can, for example, change font sizes, colors, backgrounds, add borders or spacing in and around elements.

EXAMPLE PAGE TO SHOW CSS STYLING

Web browsers will apply some basic formatting to an HTML document without any style declarations.

CSS ability

Cascading Style Sheets allow you to control the appearance of any element within your HTML document. You can, for example, change font sizes, colors, backgrounds, add borders or spacing in and around elements.

图2：左边是纯HTML，右边是应用了CSS的HTML。

最后，JavaScript 为你的 Web 站点提供了动态功能。你可以写一些将在客户计算机上运行的 JavaScript 小程序，而不用在服务器上安装任何特别的软件。JavaScript 能让你可以向你的 Web 站点增添一些基本的功能和交互，但它也有局限性，因此我们下面就要谈谈服务器端的编程语言和动态网页。

动态网页

有时当你在浏览互联网时，你会看到一些网页的扩展名不是 .html ，而可能是.php、.asp、.aspx、.jsp，或其它奇怪的扩展名。它们都是动态网页技术的例子，动态网页技术可用于创建具有动态代码部分的网页，代码部分将根据从数据库、表格、或其它数据源之中输入的数值，显示不同的结果。我们将在下面的部分比较一下静态网页和动态网页。

需要其它程序或插件的格式

由于 Web 浏览器只设计用来解析和显示一些特定的技术，如 Web 标准等，因此如果你请求的地址指向了一个复杂的文件格式，或是包含了某个需要特定插件的网页，那么插件会被要求下载到你的计算机。而如果你的浏览器已安装了该插件，那么此网页会被所需插件所支持并打开。例如：

1. 如果你遇到 Word 文档、Excel 文件、PDF、压缩文件（例如 ZIP 或 SIT 文件）、复杂的图像文件（如 Photoshop PSD），或浏览器不认识的其它复杂文件，浏览器通常会问你是否想下载或是打开文件。这两种方式效果其实是一样的，只是后一种方式，将使浏览器先下载该文件，然后由可以打开该文件的应用程序（如果已安装了的话）打开文件。
2. 如果你遇到的页面含有 Flash 电影、MP3 或其它格式的音频文件、MPEG 或其它格式的视频文件，浏览器将用已安装的插件来播放它们。如果所需的插件并未安装，则浏览器

要么显示一个用于安装所需插件的链接，要么将文件下载下来，再通过桌面应用程序来打开文件。

当然，这里也存有一些灰色地带，如 **SVG**（可伸缩矢量图）是一个 Web 标准，可以在一些浏览器中直接打开，如 **Opera** 浏览器。但不能在另外一些浏览器（如 **Internet Explorer**）中直接打开，因为 **IE** 需要一个插件来解析 **SVG**。许多的浏览器都将匹配预装插件，因此你也许不会注意到哪些内容是通过插件显示的，而哪些内容是通过浏览器直接被显示的。

静态网站与动态网站的比较

什么是静态网站和动态网站，二者之间有什么差别？与一盒巧克力相似，东西都在里面。

静态 Web 站点是指这样一种网站，其中的内容、HTML 和图形始终都是静态的，任何访问者看到的都是一样的页面，除非创建网站的人决定手动更改存在服务器上的网站副本。我们在本文中讲述的主要是这类静态 Web 站点。

而在一个动态 Web 站点上，在服务器上的内容是一样的，但除了包含有 HTML 外，网站还包含有动态代码，可以显示不同的数据，取决于你向网站输入的信息。让我们看一个例子吧，通过你的 Web 浏览器访问 www.amazon.com，搜索 5 种不同的产品。亚马逊网站实际上并未将 5 个不同的网页发送给你，而是将一个相同的网页发送给你了 5 次，但每次发送给你的网页都带有加入的动态信息。这些不同的信息储存在一个数据库中，在请求时提取相关信息并发送给 Web 服务器，以插入动态网页。

需要注意的另一件事情是，为创建动态 Web 站点，必须在服务器上安装特别的软件。一方面普通的静态 HTML 文件将以.html 这一扩展名保存；而那些除了包含 HTML 内容，还包含有特别动态代码的文件，将以特殊的扩展名进行保存（如 PHP 文件通常就具有.php 这一文件扩展名）。通过这些特殊的扩展名，Web 服务器得知，它们在被发送到客户端之前，需要进行额外的处理（例如从数据库中插入数据等）。

现在有很多动态编程语言可供选择，比如我上面提到的 PHP，另外比如 Python、Ruby on Rails、ASP.NET、Coldfusion 等。实际上所有这些动态编程语言基本上都具有相同的能力，例如与数据库对话、校验输入表格的信息等，不过它们还是有轻微的差异，有各自的优点和缺点。要问哪种最好，还要哪个最适合你。

在本课程中，我们将不再进一步讲述动态编程语言，不过如果你希望深入了解动态编程语言，我在下面列出了一些可对你有帮助的资源：

- Rails: Fernandez, Obie. (2007), *The Rails Way*. Addison-Wesley Professional Ruby Series.
- [Rails screencasts](#)
- PHP: Powers, David (2006), *PHP Solutions: Dynamic web development made easy*, friends of ED.
- [PHP Online documentation](#)
- ASP.NET: Lorenz, Patrick. (2003). *ASP.NET 2.0 Revealed*. Apress.
- [ASP.NET: online ASP.NET documentation and tutorials](#).

总结

在本篇文章中，我向大家讲述了互联网是如何工作的。虽然它只是就本课程包含的很多主题泛泛地讲述了一番，但还有很有用的，因为通过将所有这些主题大致梳理一遍，可以看到它们之间是如何联系在一起的，又是如何共同工作的。关于 HTML、CSS 和 JavaScript，还有很多实际的语法要学习。在下一篇文章中，我们将重点讲述用于 Web 开发的“Web 标准”所包括的几部分，即 HTML、CSS 和 JavaScript，并讲述一下网页代码。

练习题

- 给出 HTML 和 HTTP 的简明定义，并说明二者之间的差别。
- 说明 Web 浏览器的功能。
- 花 5—10 分钟上网，试着找出一些不同类型的内容，如纯文本、图像、HTML、动态网页（如 PHP 和 .Net 网页--.aspx）、PDF、Word 文件、Flash 电影等。访问其中一些内容，并思考你的计算机是如何显示这些内容的。
- 静态网页和动态网页的差别是什么？
- 找一个 HTTP 错误代码列表，列出其中 5 种，并解释每个错误代码的意思。
- 上一篇：互联网和 Web 的历史，以及 Web 标准的演化
- 下一篇：Web 标准所包括的模块——HTML、CSS 和 JavaScript
- [目录](#)

作者简介



Jonathan Lane 是 [Industry Interactive](#)（工业互动）公司的总裁，该公司是一家从事Web开发及Web应用程序开发的公司，位于加拿大不列颠哥伦比亚省梅恩岛。他曾在[Lethbridge](#)大学的课程再开发中心工作过多年，担任该中心Web项目的协调人。

他的博客地址为：<http://www.flyingtroll.com/>。目前他正在开发 [Mailmanagr](#) 软件—[Basecamp](#) 项目管理应用程序的一个 e-mail 界面。

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

4. Web 标准所包括的模块——HTML、CSS 和 JavaScript

Posted 12/10/2008 - 17:32 by Lewis

作者: Jonathan Lane · 2008 年 7 月 8 日

- 上一篇: 互联网是如何运转的?
- 下一篇: 梦想虽然很美, 但实际上到底怎么样?
- 目录

序言

在上一篇文章, 我们已粗略了解了网页的三大构造块: HTML (或 XHTML), CSS 和 JavaScript。现在我们再深入了解和考察这三大构造块, 了解和考察它们各自的功能及它们之间如何相互作用而创建出 Web 站点的。本篇文章的内容如下:

- 为什么要将HTML、CSS、JavaScript这三者分离?
- 标记, 每个网页的基础
 - 什么是XHTML?
 - 什么是校验?
- 使用CSS添加样式
- 使用JavaScript向网页中添加行为
- 一个示范网页
 - index.html
 - styles.css
- 总结
- 练习题

为什么要将HTML、CSS、JavaScript这三者分离?

这是人们通常问的关于 Web 标准的第一个问题。既然可以仅使用 HTML 就完成网页的内容编写、样式设定 (使用 HTML 的 Font 元素) 和页面布局 (使用 HTML 表格), 那为什么还必须使用 XHTML/CSS (层叠样式表) 呢? 在过去, 人们在设计网页时常使用一些不好的方法如用 HTML 表格来进行页面布局等, 就是现在都还有很多人仍在使用这些不好的方法 (但你们绝对不应该这样做), 这也就是我们要推出本课程的原因之一。在本课程中, 我们不会讲述这些过时的方法。以下是一些你需要抛弃那些过时的方法, 取而代之以使用 CSS 和 HTML 的有力理由:

1. **代码的效率:** 你放在服务器上的文件越大，下载文件需要的时间就越长，一些上网的人可能支付的费用也就越高(现在一些人仍然是按下载的megabyte(兆字节)数支付网费的)。因此你希望避免出现这种情况：在每一个HTML文件中都包含大量的样式和页面布局信息，从而浪费带宽。你可以选择另一种好得多的方式，即在HTML文件中使用最精简的代码，而把样式和页面布局信息包含进CSS文件中。你们可以在网页设计站点[A List Apart Slashdot](#)上的一篇文章（[the A List Apart Slashdot rewrite article](#)）中找到使用这一方法的一个实例，在该篇文章中，作者以一个非常受欢迎的Web站点为例，用XHTML/CSS重写了该站点的网页代码。
2. **易于维护:** 接着上一点讲，如果页面的样式和布局信息保存在单独的CSS文件中，那么你在想改变站点的外观时，仅需要在单独的CSS文件中做出更改即可。难道你会更愿意在站点的每一个网页中一处一处地去更改样式和布局信息吗？我想不会。
3. **可访问性:** 上网用户中那些视力受损的人，可以使用一种叫“屏幕阅读器”的特殊软件，通过耳朵听而不是眼睛看来访问网页，也就是说，屏幕阅读器将网页的内容读给他们听。此外，如果语义化的网页编写得好，也有助于有运动障碍的人使用语音输入软件无障碍地访问网页。与使用屏幕阅读器的用户用键盘命令来导航标题、链接、表单等类似，使用语音输入软件的用户用语音命令来导航标题、链接、表单等。在这些情况下，以语义化的HTML（结构和表现相分离的HTML）编写的网页文件，就可以让这些用户更容易导航，且网页文件中的信息也更有可能被这些用户找到。换句话说，访问网页“内容”的速度越快则越好。由于屏幕阅读器无法读取闭锁在图像中的文本，也会被一些所使用的JavaScript搞糊涂，因此你务必确保网页的关键内容可以被每一个人访问。
4. **设备兼容性:** 由于XHTML页面只是纯标记，没有附加样式信息，它就可以针对具有不同特点（如屏幕尺寸等）的设备而被重新格式化，只需要应用一种另外的样式表即可，具体做法有好几种（请访问[dev.opera.com](#)上关于移动Web开发的文章（[mobile articles on dev.opera.com](#)），了解具体的做法）。同时，CSS本身也可以让你为不同的呈现方式和媒体类型（如在屏幕上阅读网页，打印网页，在移动设备上阅读网页等）规定不同的样式表。
5. **网络爬虫/搜索引擎:** 你肯定希望用户通过Google等搜索引擎，能更容易地找到你的网页。搜索引擎使用一种专门的软件，即所谓“爬虫”，通读你的网页。如果由于你在编写网页时的失误，如未将标题按标题来定义等，爬虫就难于发现你的网页的内容，或是会错误地解释哪些才是重要的内容，那么你的网页在搜索结果中的排名就会大受影响。
6. **这样做是良好的习惯:** 这听起来有点像只是我个人的意见，但实际上你请教任何通晓Web标准的专业Web开发员或设计师，他们都会告诉你，将内容、样式和行为这三者分离，是开发Web应用程序的最佳方式。

标记，每个网页的基础

HTML 和 XHTML 是由包含属性（attribute）的元素（element）组成的标记语言（有些是可选的，有些则是强制性的）。这些元素用于标记文档中各种不同类型的内容，并规定每部分的内容在 Web 浏览器中怎样被渲染（例如标题、段落、表格、符号列表等）。

正如你们预想的那样，元素定义内容类型，而属性定义这些元素的附加信息，例如识别元素的 ID，或一个链接指向的位置。你必须牢记一点，标记应该尽可能地语义化，即应该尽可能精确地描述内容的功能。图 1 就是一个典型的(X)HTML 元素的剖析图。

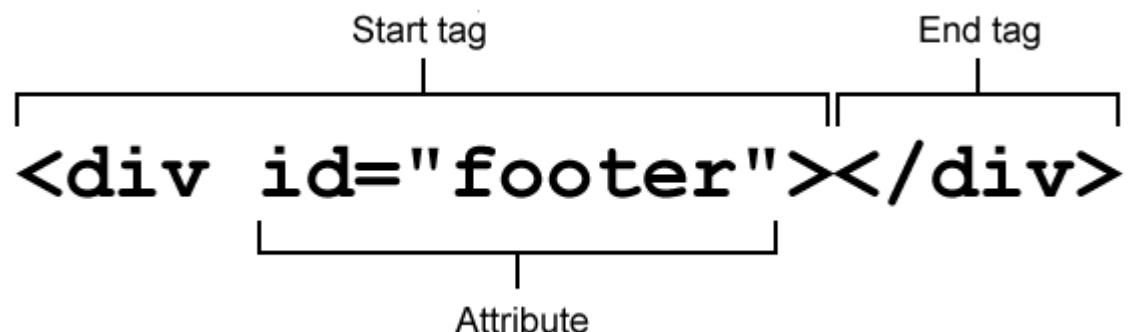


图 1：(X)HTML 元素的剖析图。阅读图 1 的说明

HTML 和 XHTML 的差别究竟在哪里呢？

什么是 XHTML？

XHTML（可扩展超文本标记语言）中“X”的含义是“可扩展的”。初学者最爱问的一个问题是：“我是该用 HTML 呢，还是 XHTML？两者之间到底有什么差别呢？”其实它们的功能是大体一样的，两者最大的差别是在结构上。见表 1 列出的 HTML 和 XHTML 的主要差别。

HTML	XHTML
元素和属性是大小写不敏感的， <code><h1></code> 等同于 <code><H1></code> 。	元素和属性是大小写敏感的，都以小写形式书写。
一些元素不需要结束标签（例如 <code>paragraphs</code> , <code><p></code> ），而另一些元素（被称为空元素）禁止用结束标签（例如 <code>images</code> , <code></code> ）。	<p>所有元素都必须有结束标签（例如 <code><p>A paragraph</p></code>）。空元素的格式是，在起始标签中包含一个斜杠（例如，<code><hr></hr></code> and <code><hr /></code>是等同的）。</p> <p>如果你将XHTML文件用作<code>text/html</code>，则必须对所有被定义为“空”的元素使用简写语法，并在<code>/></code>前加上一个空格。对任何未被定义为“空”的元素，即使其中无任何内容，也必须使用完整格式的标记（有单独的起始标签和结束标签）。</p>
一些属性值可不在引号中。	属性值必须要在引号中。
一些属性可简写（如 <code><option selected></code> ）。	所有属性都必须使用完整的属性格式（如 <code><option selected="selected"></code> ）。
服务器须以 <code>text/html</code> 这种媒体类型将 HTML 发送到客户端。	XHTML须使用 <code>application/xhtml+xml</code> 这种媒体类型，但也可使用 <code>application/xml</code> , <code>text/xml</code> 或 <code>text/html</code> 。如果使用 <code>text/html</code> ，则必须遵循 HTML 兼容性指导原则，因为这样浏览器将把其当作HTML处理（并使用错误恢复来解释语言之间的差别）。

表 1: HTML 和 XHTML 的差别。

正如我已经说过的那样，你不用太担心写的是HTML还是XHTML。你只要严格遵照本课程所有文章给出的建议去做，并使用一种HTML文档类型（关于文档类型，参见本课程第 14 篇文章），你就不会犯什么大错。

什么是校验？

由于HTML和XHTML是制订的标准（就这点而论，CSS也是制订的标准），万维网联盟（W3C）创制了一个被称为“校验器”（validator）的好工具，它可以自动检查你制作的网页，并指出你所写的代码可能有的问题/错误（如遗漏结束标签或遗漏属性的引号等）。HTML校验器可在线获取（网址：<http://validator.w3.org/>），它将自动侦测你使用的是HTML还是XHTML，以及你使用的是哪一种文档类型。如果你想检查你的CSS，适用的校验器也可在线获取（网址：<http://jigsaw.w3.org/css-validator/>）。

欲了解关于校验更多的信息，见本课程第 24 篇文章（尚未发布）。欲了解关于文档类型更多的信息，见 第 14 篇文章。

使用 CSS 添加样式

层叠样式表（CSS）让你可以很好地控制文档的格式设定和布局。使用 CSS，你可以改变或添加颜色、背景、字体大小、样式，甚至可以在网页内不同的位置对元素进行定位。使用 CSS 来应用样式，主要有三种方式：一、重定义一个元素；二、对一个 ID 应用一种样式；三、对一个类应用一种样式。下面我将分别讲述一下这三种方式：

1. 重定义一个元素。通过定义向元素添加样式的规则，你可以改变任何(X)HTML 元素的显示方式。如果你希望文档中所有段落都是双倍行距，并且是绿色的，你可以在 CSS 中添加如下所示的样式表声明：

```
p {  
    line-height: 2;  
    color: green;  
}
```

现在任何包围在<p></p>标签之间的内容，其行高都将增加一倍，并将变为绿色。

2. 定义一个 ID。你可以赋予一个元素以一个 id 属性，以在一个网页中识别该元素（在一个网页中，每一个 ID 只能使用一次）。例如，`id="navigation_menu"`。这样你就可以很好地控制页面的格式设定。例如，如果你希望某一段落是双倍行距，并以绿色文本强调显示，则可以赋予它以一个 ID：

```
<p id="highlight">Paragraph content</p>
```

然后对它应用一种 CSS 规则，如下所示：

```
#highlight {  
    line-height: 2;  
    color: green;  
}
```

这样做，该 CSS 规则仅会应用于网页内那个其 id 属性为 `highlight` 的段落（#是 CSS 惯例中用于指明 ID 的符号）。

3. 定义一个类。类与 ID 类似，它与 ID 的区别仅在于在每个网页你可以有多个相同的类。接着我们那个双倍行距的例子讲，如果你希望一个网页中的头两个段落为双倍行距且突出显示，你可以向它们添加类，如下所示：

```
<p class="highlight">Paragraph content</p>
<p class="highlight">The content of the second paragraph</p>
```

然后对它们应用一种 CSS 规则，如下所示：

```
.highlight {
    line-height: 2;
    color: green;
}
```

在以上例子中，highlight 是一个类，而不是一个 ID(.是 CSS 惯例中用于指明“类”的符号)。

以下的那个示范网页，将让你了解更多关于 CSS 如何样式化 HTML 的知识。我们将在即将发布的第 22 篇文章中更详细地讲述 CSS。

使用 JavaScript 向网页中添加行为

最后再来看 JavaScript，它是一种用于向网页中添加行为的脚本语言，可以用于检验你输入某一个表单里的数据的有效性（告诉你其格式正确与否），提供拖放功能，改变漂浮广告的样式，使页面元素如菜单等动起来，处理按钮功能等等。最新的 JavaScript 是这样工作的：找到一个目标 HTML 元素，然后对该元素进行一些处理。这和应用 CSS 差不多，不过两者的运行方式、语法等，则有相当大的差别。

JavaScript 是比 HTML 和 CSS 更为复杂和庞大的主题，在目前这个学习阶段，我们还是尽量不去学习太复杂高深的主题，因此在以下这个示范网页中，我们不讨论 JavaScript。实际上你们要在本课程的后期阶段才会学习 JavaScript。

一个示范网页

在本篇文章中，有很多细节知识我并未讲述，不过请放心，在这个网页设计教程中，我们以后会一一讲述这些知识的。现在我将给你们看一个真实的示范网页，以让你们大致了解在以后的课程中将要学到些什么知识，增加学习兴趣。

以下这个网页是一个参考网页，你们可以把它用作在你们撰写的文章（如一篇关于 Web 开发团队的团体动力学的论文，或是一篇关于美国人使用宽带互联网情况的报告）末尾罗列参考文献的参考。请注意一点，如果你是一个严守学术写作的标准的人，我会告诉你这个示例使用的是 APA（美国心理学协会）的文献引用标准。[点击这里](#)，[下载代码](#)。

index.html

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

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

    <title>References</title>
    <style type="text/css">

        @import url("styles.css");
    </style>
</head>

<body>
    <div id="bggraphic"></div>
    <div id="header">
        <h1>References</h1>
    </div>
    <div id="references">
        <cite class="article">Adams, J. R. (2008). The Benefits of Valid Markup: A Post-Modernistic Approach to Developing Web Sites. <em>The Journal of Awesome Web Standards, 15:7,</em> 57-62.</cite>
        <cite class="book">Baker, S. (2006). <em>Validate Your Pages.... Or Else!..</em> Detroit, MI: Are you out of your mind publishers.</cite>

        <cite class="article">Lane, J. C. (2007). Dude, HTML 4, that's like so 2000. <em>The Journal that Publishes Genius, 1:2, </em> 12-34.</cite>
        <cite class="website">Smith, J. Q. (2005). <em>Web Standards and You.</em> Retrieved May 3, 2007 from Web standards and you.</cite>
    </div>
    <div id="footer">
        <p>The content of this page is copyright © 2007 <a href="mailto:jonathanlane@gmail.com">J. Lane</a></p>
    </div>
</body>
</html>
```

我将不会逐行剖析这个文件，以后课程中的很多示例也会这样处理的。不过要点我还是会给你们指出来的，以下就是一些要点：

第 1 行，是所谓文档类型声明，或简称为doctype。在本示例中，这个网页的文档类型是 **XHTML 1.0 Transitional**。文档类型规定你所写的标记必须遵循的一套规则，并可进行校验。关于文档类型的更多信息，见 第 14 篇文章。

第 5—7 行，将一个 CSS 文件导入这个网页，也就是说，该 CSS 文件中包含的 CSS 将应用于该网页中的各个元素。在下一节，你们可以看到对该网页进行格式设定的 CSS 文件的内容。

我已为每一种类型的参考文献指定了不同的“类”，这样我就可以对每一种类型的参考文献应用一种样式，例如，在每一个参考文献的右边我都添加上了不同的颜色，以便于浏览该参考文献列表。

现在来看看样式化 HTML 的 CSS。

styles.css

```
body {  
    background: #fff url('images/gradbg.jpg') top left repeat-x;  
    color: #000;  
    margin: 0;  
    padding: 0;  
    border: 0;  
    font-family: Verdana, Arial, sans-serif; font-size: 1em;  
}  
  
div {  
    width: 800px;  
    margin: 0 auto;  
}  
  
#bggraphic {  
    background: url('images/pen.png') top left no-repeat;  
    height: 278px;  
    width: 362px;  
    position: absolute;  
    left: 50%;  
    z-index: 100;  
}  
  
h1 {  
    text-align: center;  
    text-transform: uppercase;
```

```
font-size: 1.5em;
margin-bottom: 30px;
background: url('images/headbg.png') top left repeat;
padding: 10px 0;
}

#references cite {
margin: 1em 0 0 3em;
text-indent: -3em;
display: block;
font-style: normal;
padding-right: 3px;
}

.website {
border-right: 5px solid blue;
}

.book {
border-right: 5px solid red;
}

.article {
border-right: 5px solid green;
}

#footer {
font-size: 0.5em;
border-top: 1px solid #000;
margin-top: 20px;
}

#footer a {
color: #000;
text-decoration: none;
}

#footer a:hover{
text-decoration: underline;
}
```

在设置这个网页的样式时，我略微有点渲染过度，比如添加了一些嵌套背景效果，以向你们展示使用 **CSS** 可以达到的一些效果。

第 1 行为文档设定一些默认样式，如文本和背景颜色、文本周围加入的边框的宽度等。有些人不愿花时间去设定这些默认样式。确实即使你不设定默认样式，现在多数浏览器都将应用它们自己的默认样式，但自己设定一些默认样式是一个好的习惯，因为这样做可以让你更好地控制你的 Web 站点在不同的浏览器中如何被显示。

在下一行中，我把网页设定为 800px（像素）宽（尽管我也可以在此规定一个页面根据浏览器窗口的大小而扩大/缩小的百分比）。我在此处使用这种边距（margin）设置，是为了确保页面内容在窗口的中心显示。

下面让我们转向看这个网页的背景图像（通过使用background: url 这一背景声明而实现的）。我在这个网页上添加了 3 个背景元素：第 1 个是横跨页面顶部的斜面，呈现出好看的青色；第 2，我使用了一个半透明的钢笔PNG图像，以和它之上的文本形成足够的对比，并将斜面衬托得更好看一些（半透明的PNG图像在Internet Explorer 6 或更低的版本中无法正常显示，但在每一种最新式的浏览器中都能很好地显示，关于如何解决IE6 无法正常显示半透明PNG图像这一问题，见 Dean Edward's IE-fixing JavaScript （Dean Edward 的文章，《用于修补 IE 的 JavaScript》），该文章还讨论了如何处理IE6 在CSS支持方面存在的问题。第 3，使用了另一个半透明的PNG图像作为页面标题的背景，它增强了标题部分的对比度，并产生一种嵌套的效果。

示范网页的显示效果如图 2 所示。

REFERENCES

- Adams, J. R. (2008). *The Benefits of Valid Markup: A Post-Modernistic Approach to Developing Web Sites*. *The Journal of Awesome Web Standards*, 15:7, 57-62.
- Baker, S. (2006). *Validate Your Pages.... Or Else!*. Detroit, MI: Are you out of your mind publishers.
- Lane, J. C. (2007). Dude, HTML 4, that's like so 2000. *The Journal that Publishes Genius*, 1:2, 12-34.
- Smith, J. Q. (2005). *Web Standards and You*. Retrieved May 3, 2007 from <http://www.webstandardsandyou.com/>.

图2：应用样式后示范网页的显示效果

总结

XHTML, CSS 和 JavaScript 并不神秘，它们都是 Web 演进的自然产物。如果你已具有一些 HTML 方面的知识，完全不用丢弃。你所具有的所有 HTML 知识，依然很有用，你所需要做的只是以与过去不同的方式来处理一些事情，并在写标记时更仔细一些而已。

使用 Web 标准进行 Web 开发，除可以让你把 Web 开发工作做得更好以外，本身也是很意义的。一些人不愿意使用 Web 标准进行 Web 开发，其理由是：这样做很耗时，而且必须费很大力气进行能跨浏览器显示的页面布局。对此我们可以提出这样的反驳：使用非基于 Web 标准的方法进行页面布局，就能让网页能跨设备并在未来更高版本的浏览器中正常显示吗？你怎么就能肯定你所写的非基于 Web 标准的标记，可以在 Opera 12.0, Firefox 5.0 和 IE 10.0 上都能显示？除非你遵循一些规则，否则是不可能做到的。

练习题

- 类和 ID 的差别是什么？
- XHTML, CSS 和 JavaScript，在 Web 站点中所起的作用分别是什么？
- 以示范网页的 index.html 文件为素材，单独使用 CSS 改变页面的格式设定(我建议使用文本编辑器如 Notepa 或 Text Wrangler 编辑文件的代码)。不要对 HTML 进行任何更改。
 - 为不同类型的参考文献添加图标（即为文章、书籍和互联网资源添加不同的图标）。先创作图标，然后单独使用 CSS，使图标显示在不同类型的参考文献旁边。
 - 隐藏页面底部的版权信息。
 - 改变标题的外观，使之更醒目。
- 为使示范网页能在手机浏览器上正常显示，你能做些什么工作？
- 上一篇：互联网是如何运转的？
- 下一篇：梦想虽然很美，但实际上到底怎么样？
- 目录

作者简介



Jonathan Lane 是 **Industry Interactive**（工业互动）公司的总裁，该公司是一家从事Web开发及Web应用程序开发的公司，位于加拿大不列颠哥伦比亚省梅恩岛。他曾在**Lethbridge**大学的课程再开发中心工作过多年，担任该中心Web项目的协调人。

他的博客地址为：<http://www.flyingtroll.com/>。目前他正在开发 **Mailmanagr** 软件—**Basecamp** 项目管理应用程序的一个 e-mail 界面。

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

5. 梦想虽然很美，但实际上到底怎么样？

Posted 12/10/2008 - 17:41 by Lewis

- 网络标准教程

作者: Jonathan Lane · 2008 年 7 月 8 日

- 上一篇: Web 标准所包括的模块——HTML、CSS 和 JavaScript
- 下一篇: 信息架构——规划一个Web站点
- 目录

序言

到目前为止，我已向你们介绍了 Web 标准的美好构想。使用 Web 标准，可以实现所有类型浏览器之间的可互通性（interoperability），使 Web 程序能在每一种操作系统上运行，甚至可以让网页在每一种电子设备上显示。但这已真正成为现实了吗？所有的 Web 浏览器都百分之百地符合 Web 标准吗？所有的 Web 开发员都适当地使用了 Web 标准吗？Web 开发员是否是在使用 Web 标准制作了一个网页后，就确信网页能跨浏览器、跨平台、跨设备地访问，不再需要做后续工作了呢？

对最后一个回答很简单：不是。尽管这是一种理想的状况，但离现实还很遥远。在本篇文章中，我将讲述以下内容：

- 你如何检查Web站点是否符合Web标准？
- Web站点是否符合Web标准的现状
 - 亚马逊网站：它符合Web标准吗？
 - 新闻网站CNN：它符合Web标准吗？
 - 苹果公司：在创意设计方面出类拔萃，那校验呢？
 - 是否符合Web标准的小调查
- 为什么缺乏符合Web标准的Web站点？
 - 教育
 - 商业原因
- 总结
- 延伸阅读
- 练习题

你如何检查 Web 站点是否符合 Web 标准?

在进行下一步的学习前，你可能问这样一个问题：“我怎么知道一个 Web 站点是否使用了 Web 标准呢？”它看起来和其他 Web 站点是不是就不一样？

答案：是，同时又不是。符合 Web 标准的 Web 站点，如果是使用恰当的方式开发的，其外观和那些使用一切都混杂在一起的标记所编写的 Web 站点应该是没有差别的。但是 Web 站点的源代码则会呈现出巨大的差异（要查看网页的源代码，可以点击右键，选择“源代码”或“查看源代码”选项，依据不同的浏览器而定）。一个符合 Web 标准的 Web 站点是以精简、清洁的标记代码写成的，在页面本身中只有很少或没有嵌入的格式代码。仅看一下 Web 站点的外观，你可能难于发现符合与不符合 Web 标准的 Web 站点之间的差别，但请相信我，那些使用屏幕阅读器上网的视力受损人士和搜索引擎是会发现两者之间的明显差别的。在上一篇文章中，我已讲述了使用 Web 标准所具有的很多优点。

要检查 Web 站点是否符合 Web 标准，最简单的方式是使用一个可在线获取的被称为“校验器”的便捷工具。万维网联盟（W3C）免费在线提供“校验器”，网址为 <http://validator.w3.org/>，如图 1 所示。你可以（也必须）使用这个工具来检查你正在开发的 Web 站点是否存在 HTML/XHTML 错误。同时，你也可以使用可在线获取的 CSS 校验器检验 CSS，网址为 <http://jigsaw.w3.org/css-validator/>。请点击这些链接，随意检查几个你最喜爱的 Web 站点。



Markup Validation Service

Check the markup (HTML, XHTML, ...) of Web documents

Jump To: Congratulations · Icons

This Page Is Valid XHTML 1.0 Strict!

Result: Passed validation

Address :

Encoding : utf-8

Doctype : XHTML 1.0 Strict

Root Element: html

Root Namespace: <http://www.w3.org/1999/xhtml>

Options

- Show Source Show Outline List Messages Sequentially Group Error Messages by type
 Validate error pages Verbose Output Clean up Markup with HTML Tidy

[Help](#) on the options is available.

Revalidate

图 1：万维网联盟提供的标记校验服务，可检查你编写的网页，指出标记可能存在的任何错误。

校验你编写的网页只是完成了工作的一半。我们如何检查浏览器是否符合Web标准呢？

Web标准组织已开发出一系列检查浏览器是否符合Web标准的测试，被称为“Acid 测试”，即使用一些复杂的HTML和CSS规则（加上其他一些标记和代码），检查某一浏览器是否能正确地渲染测试屏幕。最新版本的“Acid测试”，即Acid3，尚在开发中。想了解“Acid测试”更多的信息，可以访问 <http://www.acidtests.org/>。你同时还可以到该站点的测试页面，测试你的浏览器是否符合Web标准。

Web 站点是否符合 Web 标准的现状

主要的 Web 站点正在使用 Web 标准，还是依然在沿用那些过时的方法和习惯呢？让我们考察几家大公司的 Web 站点，看看使用万维网联盟的标记校验服务对它们进行校验后给出的结

果。你也许会感到惊异，原来有那么多大网站未能通过 Web 标准校验测试。不过你不要因此就感到沮丧，你是可以做得更好的，是可以使你的站点通过 Web 标准校验测试的。当你阅读以下的报告时，记住一点，那就是一般说来，一个 Web 站点越大、越复杂（其他一些因素如所使用的技术等，也有影响），就越难通过 Web 标准校验测试。

亚马逊网站：它符合 Web 标准吗？

如果你在网上购过物的话，很可能访问过 [Amazon.com](http://www.amazon.com)（或是它的某一个特定国家网站）。
亚马逊网站是一个巨型网上商店，出售很多种类的商品，从书籍、CD 到食品杂货等等。
[Amazon.com](http://www.amazon.com) 通过 Web 标准校验测试了吗？答案请看图 2。

This page is not Valid (no Doctype found)!

Result:	Failed validation, 1500 Errors
Address :	<input type="text" value="http://www.amazon.com/"/>
Encoding :	iso-8859-1 <input type="button" value="(detect automatically)"/>
Doctype :	(no Doctype found) <input type="button" value="(detect automatically)"/>
Root Element:	html

Options

Show Source Show Outline List Messages Sequentially Group Error Messages by type
 Validate error pages Verbose Output Clean up Markup with HTML Tidy

[Help on the options is available.](#)

图 2：Amazon.com 完全未能通过 Web 标准校验测试。不仅网页中存在不合法的标记，甚至未声明网页的文档类型（即未声明所使用的是哪一种版本的 HTML/XHTML）。

要符合 Web 标准，亚马逊网站还有一段路要走。我并不了解亚马逊网站的内情，不过我可以推测一下出现这种情况的原因。我估计由于亚马逊网站建站已经很久了，他们可能一直在使用同一种软件进行站点开发。Web 标准在 21 世纪初期才成为业界焦点，而亚马逊网站的在线销售系统很可能是在 Web 标准发布前或能被支持前就已经开发出来的。我不是很确定，但我推测亚马逊网站出现的问题应该就是一直沿用 Web 标准发布前的方法在进行 Web 开发和设计。

新闻网站 CNN：它符合 Web 标准吗？

新闻网站的情况又是怎么样的呢？CNN.com 是全球最大的媒体网站之一。通过整合全球

媒体资源，CNN实时报道全球各地的新闻，他们应该有一个专业的Web开发和设计团队，确保网站是以符合Web标准的有效标记制作的。果真是这样的吗？请看图 3。

✖ Line 569, Column 23: required attribute "ACTION" not specified.

```
<form class="cnnHidden" >
```

The attribute given above is required for an element that you've used, but you have omitted it. For instance, in most HTML and XHTML document types the "type" attribute is required on the "script" element and the "alt" attribute is required for the "img" element.

Typical values for type are type="text/css" for <style> and type="text/javascript" for <script>.

✖ Line 610, Column 1472: end tag for "UL" which is not finished.

```
...div><div class="cnnSvcsBull"><ul></ul > </div><div class="cnnSvcsMore"><a href=
```

Most likely, you nested tags and closed them in the wrong order. For example <p>...</p> is not acceptable, as must be closed before <p>. Acceptable nesting is: <p>...</p>

Another possibility is that you used an element which requires a child element that you did not include. Hence the parent element is "not finished", not complete. For instance, in HTML the <head> element must contain a <title> child element, lists (ul, ol, dl) require list items (li, or dt, dd), and so on.

✖ Line 626, Column 13: end tag for "DIV" omitted, but its declaration does not permit this.

```
</div></t > <td class="cnnPLDivCell">。目前他正在开发 [Mailmanagr](#) 软件—[Basecamp](#) 项目管理应用程序的一个 e-mail 界面。

---

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 6. 信息架构——规划一个 Web 站点

Posted 12/10/2008 - 17:43 by Lewis

作者: Jonathan Lane · 2008 年 7 月 8 日

- 上一篇: 梦想虽然很美, 但实际上到底怎么样?
- 下一篇: 一个好的网页需要什么?
- 目录

### 序言

一般说来, 规划一个 Web 站点 (或任何项目) 都是一件有点麻烦的工作。对于一个 Web 站点应该怎样构建, 每个人都有自己的意见, 而且这些意见之间常存在冲突。在构建任何 Web 站点时, 你的首要目标都应该是构建一个对将要使用站点的用户有用的 Web 站点。也就是说, 如果你要构建的是针对某一特定人群的 Web 站点, 只有这个人群的意见才是重要的。而其他如你的老板的意见, 拥有软件工程博士学位的人的意见, 甚至你的个人偏好, 都不是真的重要。

在本篇文章中, 我将讲述规划一个 Web 站点的前期阶段, 以及一门一般被称为“信息架构” (其简写为 IA) 的学科。主要内容包括你的目标受众是哪些人, 他们需要从一个 Web 站点获取什么样的信息和服务, 以及你应该怎样构建一个站点, 以向他们提供这些信息和服务。我将讲述需要放置在 Web 站点上的信息, 如何拆分信息, 以及这些被拆分的信息如何相互联系。内容分节如下:

- 你需要规划你要构建的Web站点
  - “The Dung Beatles”乐队简介
  - 画出网站地图
  - 为网页命名
  - 添加一些细节
- 总结
- 练习题

### 你需要规划你要构建的 Web 站点

你们可能遇到这样的 Web 项目, 不进行任何前期构想和规划, 就直接开始让你投入工作, 不过这只是例外, 而不是常态。以下我们将以一个被称为“The Dung Beatles”的虚构乐队为例, 看看我们怎么帮助这个乐队规划其 Web 站点。我们将先大致了解一下这个乐队, 找出他们建站

希望达到的目标，以及他们希望在其 Web 站点上展示那些信息。然后我们就开始为该乐团的信息构造一个结构框架。

### **“The Dung Beatles”乐队简介**

The Dung Beatles 乐队(简称 TDB 乐队)遇到了一个问题。他们是萨斯喀彻温省摩斯乔市最当红的模仿 Beatles (披头士乐队) 的乐队，但他们还需要在今年夏天即将举行的北美巡演中提高乐队的知名度。他们计划在加拿大和美国各地巡演，但他们在本城之外就几乎无人知晓了。如果有某种方式，让他们花费较少的金钱就可以为众多的 Beatles 迷知晓，那就好了。

TDB 乐队很幸运，有一种叫做万维网的技术可以帮助他们实现这个目标，因此他们迅速决定建立一个自己的网站。TDB 乐队需要通过自己的网站推广他们的北美巡演，建立起一个粉丝群，并提高乐队的知名度。你们将针对他们的想法，看是否能为他们的网站制订一份规划。

你安排了一次与这个新客户的面谈，以具体了解他们的需求并确定建站的时间要求和费用。在面谈开始时，你首先提议讨论网站的目标和目的，以了解这个乐队希望通过这个网站实现什么样的目标。

TDB 乐队开始谈论他们将进行的北美巡演，以及他们希望向每一个巡演地点的所有 Beatles 迷传达什么样的信息。现在已经是 2 月了，他们计划在 5 个月内就启动他们的巡演。

在此打住一下！一个网站本身是不会自动产生访问流量和宣传自己的。通过和乐队的交谈，你已知道该网站的主要目标是为 TDB 乐队的粉丝提供一个网上家园，通过该网站他们就能够了解乐队最新的新闻、巡演的日期和地点。通过这些粉丝的传播，以及其他一些广告宣传手段，新用户将访问这个网站，在网站上下载乐队的单曲、观看乐队的照片（化妆照），以及确认他们将在什么时间、什么地点观看到乐队的现场演出。

乐队的负责人 Raul McCoffee 指出，如果能通过销售一些乐队的 CD 和其他商品为巡演筹集一些额外的资金的话，那就太好不过了。你已经大致了解了乐队和其建站目标，快速勾画出访问网站的人可能的需求。这还只是初步的创意阶段，基本上还没有涉及到规划网站的结构。

可以把将要访问乐队网站的人分为两大类：TDB 乐队已有的粉丝；还不太确定是否会成为乐队粉丝的人。你决定以不同的方式满足这两大类人群的需求：对潜在粉丝，是向他们“推销”乐队；对已有的粉丝，是强化他们对乐队的“沉迷”。这两大类人群各自需要在网站上寻找哪些信息？

图 1 就勾画出网站访问者需要哪些信息的轮廓，基于这个轮廓，你就可以规划网站需要哪些页面，以及这些页面如何相互链接起来。

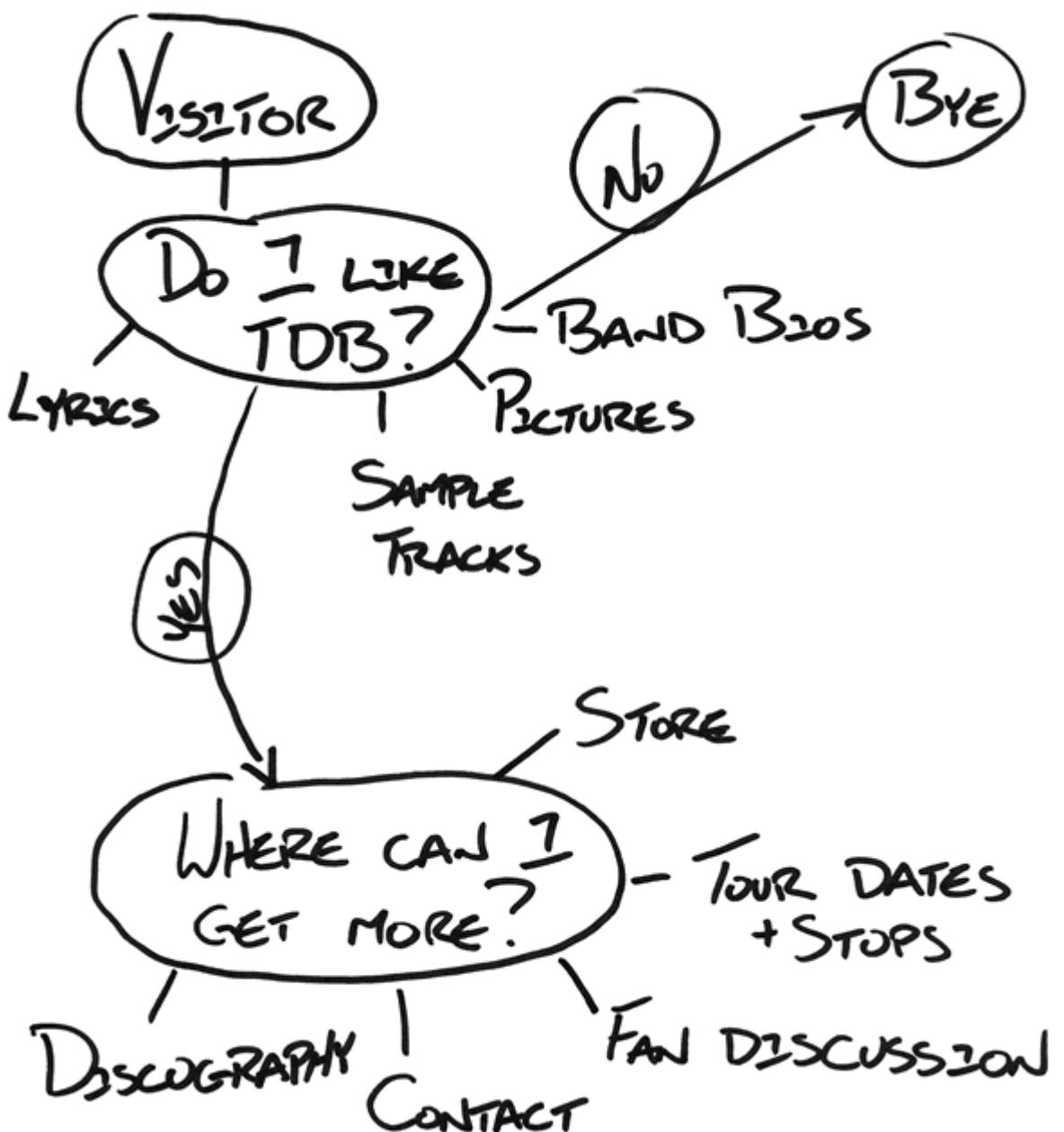


图 1：网站访问者需要哪些信息

你确定好建站预算，并同意在一个月内推出网站。你答应在一两天后回访乐队，向他们提交建站工作的计划。

#### 画出网站地图

在这个阶段，很多人会提出一个网站地图，它看起来与组织结构图相似。网站地图通常显示网站每个页面的名称，以及它们在网站总体结构中是如何相互链接起来的。就我个人来说，我会在网站地图中提供稍微详细一些的信息，包括每个页面的目的和内容。例如，一个页面可能被标注为“主页”，但“主页”到底是什么呢？是仅打出“欢迎访问我们的网站”字样的页面呢，还是一个

内容更为丰富，包含新闻条目和吸引人的图片的页面呢？请花费几分钟时间，思考以上那个信息轮廓图将具体转化为哪些页面，每个页面要包含哪些内容。请动手画出网站地图。

现在我们来看一个网站组织结构图的示例。图 2 是一个我根据和乐队的讨论结果，画出的一个乐队网站的组织结构图：

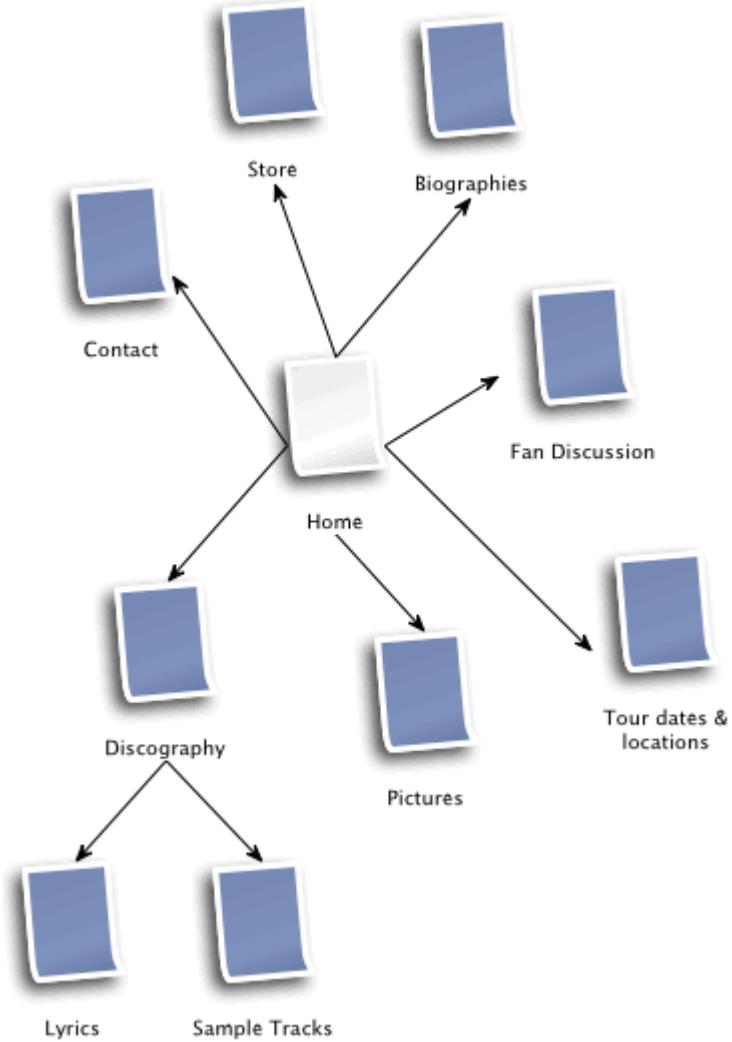


图2：初步的网站地图。图2 的图片说明

这个图已确定无疑地包含了所需要的所有页面，但尚未将页面分类。它只是列出了一大堆页面，其名称尚未经仔细推敲。下一步的工作就是将页面分成几个大类，分类的结果如图 3 所示：

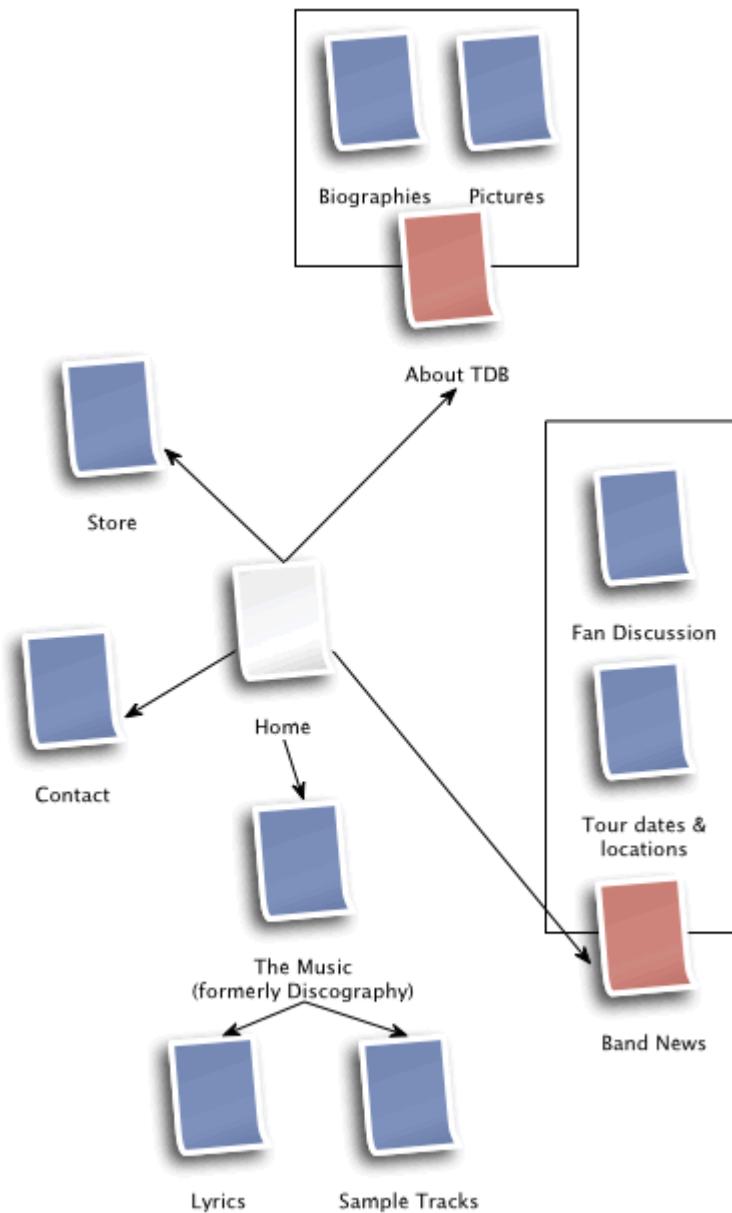


图3：修订后的网站结构图。图3的图片说明

在修订后的网站结构图中，我做了一些改进工作。“乐队新闻”页面让 TDB 乐队可发布想和其粉丝分享的任何信息。即使在他们的夏季巡演结束后，“巡演日期和地点”页面已不再相关，他们还是可以在“乐队新闻”页面发布信息。这里采用了 **blog**（博客）形式，使乐队的粉丝可以在此发表各种评论，有助于建立起一个以 TDB 乐队为主题的网络社区。新闻和巡演事件可能是最热门的讨论话题，因此就把这两个主题划分在同一个组。此外，“**News**（新闻）”是一个更简洁、更具有概括性的词，有助于浏览页面寻找信息的人，更快地找到页面。

新的“关于 The Dung Beatles 乐队”页面，包括进了每位乐队成员的传记以及他们的图片。访问这个页面，就可以点击访问每位乐队成员的传记。正如我们以上说的那样，页面标题最好使用更简洁、更具有概括性的词，而“**About**（关于）”这个词就是很多网站都经常使用的一个页面

标题词。网站访问者不论是想了解一家公司，一种产品或服务，还是某一个人，通常都会在“About (关于)”页面中寻找相关信息。

最后，“Discography（唱片分类目录）”是一个专业术语，了解这个词确切含义的人，可能要少于了解“音乐”这个词含义的人。此外，“Discography（唱片分类目录）”这个页面可以包含一些额外的内容，如某一首歌曲的灵感来源、历史等。在讲述了一些关于如何简单明确地为页面命名后，我们将学习如何为每个页面添加一些更详细的信息。

### 为网页命名

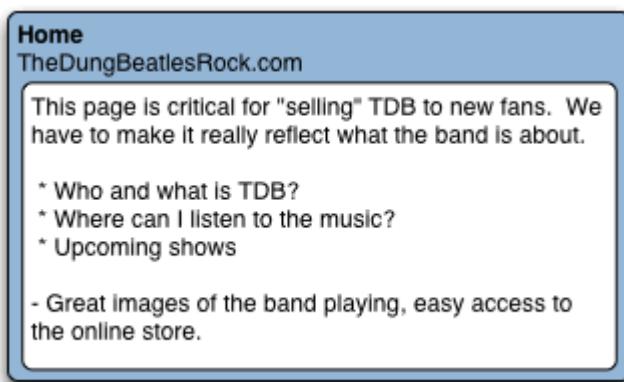
在设计网站的过程中，为网页命名是最重要的步骤之一。这不仅是因为通过网页名称，访问者可以顺畅地访问网站，而且是因为网页的名称是决定使用搜索引擎找到网站的难易度的重要因素（在本课程中，我们会多次提到搜索引擎优化）。

一般而言，在确定一个网页的重要程度时，搜索引擎查看包括在一个网页中的文本、该网页的 URL 地址，以及指向那个网页的链接。为你的网页赋予简单明确的名称和 URL 地址，有助于搜索引擎准确地描述网页。

我举一个例子。假设你们是一家汽车公司，有一款叫做“The Speedster”（速行者）的车。你已建立了一个汽车推广网站，其中有一个网页用于列出该车型的现有车型特点。你如何命名这个网页呢，是“车型特点”、“现有车型特点”、“Speedster 车的特点”，或是“车型特色”呢？我认为“Speedster 车的特点”是最恰当的名称。它指明了该网页包含的内容，这样可以将改标题放在页面上方最显眼的位置（便于搜索引擎将之编入索引），甚至可以使网页内容和 URL 地址更匹配（如使用这样一个 URL 地址：“www.autocompany.com/speedster/speedster-features/”）。

### 添加一些细节

在这个阶段，你没有必要确定每一个细节，但你至少需要给出每一个页面的简要描述。在你确定了网站的结构后，给出所需网页的数量，以及每个网页的简要描述。图 4 就是我对乐队网站主页的简要描述（在课后的练习题中，有一道题就是参照图 4，给出其他页面的简要描述）。



#### 图4：主页的页面详细描述。图片4的说明

你只需要简要地描述页面的主旨即可，不必详细描述页面的功能，你制作网页所使用的技术，以及页面布局和设计。你这样做的目的是向客户传达你的网页构想，使你能全面思考每一个要点。

在这个阶段，你有可能发现你规划了太多的网页，不能为每个网页找到相应的内容。在创建网页之间的层级时，你有可能被搞得心烦意乱。例如，如果乐队成员只是想发布一段关于他们自己的介绍，就不需要为每位乐队成员都创建单独的“传记”网页，而可以把每个成员非常简短的个人介绍集中到一个网页上。

#### 总结

在本篇文章中，我们讲述了如何规划一个 Web 站点的总体结构。在下一篇文章中，你们将学习从网页层次看，要构建出一个好的 Web 站点，需要包含哪些要素，以及这些要素的具体配置。**8、9、10** 三篇文章，将讲述网页的视觉设计。因此，在构建 Web 站点时，请遵照以下三个步骤（在每一个阶段都与客户沟通，确认他们对每个阶段的工作都满意）：

1. 首先，确定 Web 站点的内容，并确定如何将这些内容组织进网页中。
2. 其次，确定 Web 站点各个需要实际使用的功能。
3. 最后，在开始编写 Web 站点的代码前，先要进行视觉设计，如确定页面布局、配色方案等。

#### 练习题

- 参照图 1，编制一个汽车网站（选择任何一种汽车，现实的或想象的）的信息轮廓图。
- 网站访问者想要了解哪些信息？
- 在现有的汽车网站中，你认为哪些信息栏目是不可或缺的，哪些又是显得多余的？
- 整理信息轮廓图，思考什么样的页面分组才合适。
- 在规划一个网站时，研究同类网站有时也是很有用的。搜索一个模仿型乐队的网站，查看网站提供的信息和栏目。我们的网站遗漏了什么吗？
- 参照图 4，为我已提到的乐队网站的其他页面编制同样的页面详细描述图。
- 上一篇：梦想虽然很美，但实际上到底怎么样？
- 下一篇：一个好的网页需要什么？
- 目录

## 作者简介



Jonathan Lane 是 [Industry Interactive](#)（工业互动）公司的总裁，该公司是一家从事Web开发及Web应用程序开发的公司，位于加拿大不列颠哥伦比亚省梅恩岛。他曾在[Lethbridge](#)大学的课程再开发中心工作过多年，担任该中心Web项目的协调人。

他的博客地址为：<http://www.flyingtroll.com/>。目前他正在开发 [Mailmanagr](#) 软件—[Basecamp](#) 项目管理应用程序的一个 e-mail 界面。

---

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 7. 一个好的网页需要什么

Posted 12/10/2008 - 17:44 by Lewis

作者: mnfrancis · 2008 年 7 月 8 日

- 上一篇: 信息架构——规划一个Web站点
- 下一篇: 色彩理论
- 目录

### 序言

接着上一篇文章，在本篇文章中，我们将考察The Dung Beatles乐队网站的内容，以了解好的Web站点和网页需要包含些什么要素。

你们尚不需要去研究网站和网页的基础代码，而只需考察不同的网页，思考其中应该包括哪些要素，并思考一些关键性的问题，如一致性、可用性和可访问性。本篇文章的目录如下：

- 主页
  - 对我们的网站来说，这意味着什么？
- 导航
- 网站上的其他一些通用元素
- 上下文关系非常重要
  - 相关内容
  - 标题
- 可用性
- 可访问性
- 总结
- 练习题

### 主页

很多人倾向于认为“让我们从多数用户首先访问的页面—主页开始，这符合逻辑吧？”

这听起来符合逻辑，但其实并不是这样的。过分着重于主页，是一个人们常犯的错误。网站的主页常成为一个大杂烩，试图概括网站的所有内容，无所不包。

让我们看一个此类主页的实例吧，那就是 MSN 网站的主页（见图 1）。这个主页罗列了过多的内容和链接，从旅游到电视，从交友约会到指导，从小配件到绿化产品的信息，十分庞杂，

都试图引起你的注意。

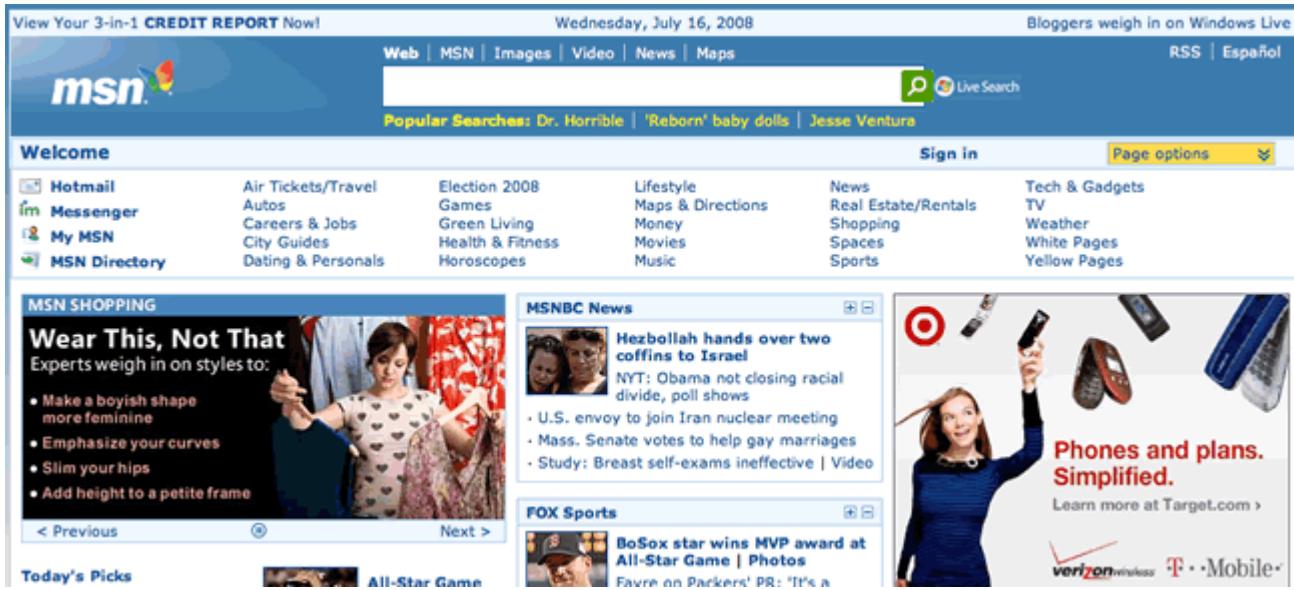


图 1：MSN 网站的主页——罗列的链接太多！

这种“将一切想得到的东西都放在上面”的主页，可能适合那些大型网站，但如果我们这个乐队网站的主页也这样的话，无疑是不恰当的，会流失很多本来可以吸引到的用户。

还有一个普遍的误解就是，主页一定是访问网站的人所看到的第一个页面。如果这些网站访问者已了解到乐队的网址，或是从广告传单、招贴画或是乐队徽章上看到乐队的网址，然后在浏览器中键入乐队的网址，确实有可能首先看到的就是网站的首页。

但是更可能出现的情况是，网站访问者是基于搜索结果来访问网站的。如果他们搜索乐队的名称，很可能（但不一定）看到的最靠前的搜索结果是乐队网站的主页。不过在其他情况下，例如他们搜索“模仿 Beatles 乐队的音乐会”，看到的第一个搜索结果可能是“巡演日期”网页；再比如他们搜索“摩斯乔市的乐队”，看到的第一个搜索结果就可能是“关于 TDB 乐队”网页，这是因为该网页提到了乐队是来自摩斯乔市的，而主页里就没有提到这一点。

《纽约时报》的网站在一篇关于决定停止向访问旧内容的用户收费的文章中提到，他们网站的访问者的行为已近发生改变，到底是什么样的改变呢，文中写道：

...越来越多的读者都是通过搜索引擎和其他网站上的链接来访问我们网站的，而不再是直接访问 NYTimes.com。以这种间接方式访问网站的读者，就无法访问那些需要付费才能看到的文章，他们与那些直接访问网站的忠实用户相比，愿意支付订阅费的可能性就要小一些。取消对访问旧内容的收费，是一个让用户可以访问更多的网页并提高网站广告收入的机会。

对我们的网站来说，这意味着什么？

这意味着你需要将内容进行分割，放在单个的网页内。你应该思考网站的访问者将如何找到他们真正在寻找的内容和信息，或者说，一旦他们开始在网站漫游，他们想访问的下一个网页是什么。

尽管很多人都试图在主页上放置过多的内容，但实际上更好的做法是把主页用作突出显示网站其他网页的内容及导向访问这些网页的一个页面。将主页和网站其他页面一样处理，并赋予它一个确定的目的(即显示更新，提供一个网站概观，仅简要介绍乐队，让访问者继续访问其他页面，等等)。主页还需要有指向其他页面的导航栏，并显示网站的品牌。

下面我们将更深入地学习这些内容。

## 导航

如何对一个网站进行导航，是网站设计中最关键的因素之一（甚至可能是最关键的）。你应当确定网站的主要栏目页，并在主导航栏中显示。

关于网站导航同样存在一个普遍的错误观念（你们可能已听说过），那就是让访问者在获取信息时不要超过三次点击。正是这种错误观念的广泛传播，使得一些网站上出现最糟糕和最复杂的导航。作为实例，你们可以去看看很多购物或价格比较网站，他们总是倾向于在页面上放置尽可能多的链接，试图使用户在购买什么之前，尽量减少点击次数，以免他们离开并去访问竞争对手的网站。但这种做法导致的结果很可能就是：罗列的信息过多，反倒使用户不能有效地获取和使用这些信息。太多的选择和太少的选择一样都会让人无所适从。

其实只要有从一个链接通向下一个页面的明确路径，显示用户正在通向最终要访问的页面的正常过程之中，用户是会继续深入访问网站的。

基于上一篇文章中讲述的信息架构，TDB 乐队网站的主导航栏应当包括指向以下部分/页面的链接，“**Store**（商店）”页面、“**About**（关于我们）”页面、“**Contact**（与我们联系）”页面、“**The Music**（音乐）”页面、“**Band News**（乐队新闻）”页面，以及一个“返回主页”的链接，并不需要包含指向如“**Tour Dates**（巡演日期）”、“**Lyrics**（歌词）”等页面的链接。指向这些页面的链接应当仅放在相应的网页内，也就是说，任何需要从某一个歌词页面直接跳转到“**Tour Dates**（巡演日期）”页面的访问者，都将可以被导航到“**Band News**（乐队新闻）”页面，然后再链接到“**Tour Dates**（巡演日期）”页面。

要想创建出成功的网站导航栏，一个最关键的因素是“一致性”。看下面图 2 所示的页面，在页面上方的导航栏中显示了一些链接如指向“**Home**（主页）”，“**Articles**（文章）”，“**Forums**（论坛）”等页面的链接。在 **dev.opera** 这个站点的其他页面，显示的导航栏都是一样的。导航指向显示你目前在网站的哪个位置，并提供指向该栏目内具体内容的链接。举例来说，点击导航栏上的“**Articles**（文章）”标签，将带你到“文章”栏目的主页面，其中包含一些指向最近发表的文章的

链接，以及一系列指向分栏目如“Accessibility（可访问性）”、“CSS”、“Mobile（移动设备）”等的链接（见图 2）。

The screenshot shows the Dev Opera website's layout. At the top, there is a horizontal navigation bar with seven items: Home, Articles, Libraries, Forums, Tools, Authors, and Contribute. Below this, the main content area features a large graphic of various drafting tools (ruler, compass, calculator, pencil) on a blue background. To the right of the graphic, the text "Welcome to Dev Opera" is displayed in bold, followed by a descriptive paragraph about the site being a community resource for developers. Below this section, another navigation bar is shown with the same seven items. Under the "Articles" section, there is a sub-navigation menu with links for Archived, Published, and Inbox, accompanied by a user icon. The "Accessibility" section features an icon of a hand and a list of articles by Frank M. Palinkas. The "CSS" section features an icon of a smartphone with a pen and a list of articles by Craig Grannell and Christopher Schmitt.

图2: The dev.opera.com 的导航栏在网站所有页面内都保持了一致性。

#### 网站上的其他一些通用元素

除导航栏外，通常还有其他一些通用元素要在网站的各个页面内都显示。

绝大多数网站都有一些标示所有权的品牌图像、网站标识或标头。例如，Yahoo!网站中几乎每一个页面，在其左上方都有一个网站标识，其中附加有你正在访问的页面所属的大栏目的名称，如“Travel（旅游）”、“Movies（电影）”、“Autos（汽车）”等等。

页面的顶部标题部分(横跨页面顶部)可以不仅仅包含网站标识，还可以包含或附加上主导航栏。此外，加上搜索框也并不少见，这可以让用户直接搜索网站的内容和信息，而不用通过使用菜单和链接来导航。你应该在你的网站的每个页面，都包括所有或大部分这些通用元素。

页脚部分(页面最下端的部分)应包含一些额外的信息，如版权声明，以及指向有用的辅助页面(如“**About This Site** (关于本网站)”、“**Terms & Conditions** (使用条款和条件)”、“**Contact Us** (联系我们)”的链接等。

配色、页面布局、图形和图标的使用、版面设置和图像，创造出作为网站有机组成部分的一个网页的整体形象，这里“一致性”是关键。让网页的外观和布置保持一致性，有助于保持网站的一体性，并创造出一种熟悉感。这样你就知道你正在访问的页面与此前访问的该网站的网页是相互联系的，都是网站的有机组成部分，因为这些页面呈现出的视觉形象就是相互联系的。当你在设计网站时，应当牢记这点，不要让网站内的各个页面看起来就不一致。

在我们的 **TDB** 乐队网站内，页面的顶部标题部分将包括乐队的标识和名称，以强化访问者对乐队的认知度，让他们在访问各个页面时，都意识到是在阅读关于乐队的各种信息。页脚部分将包括网站及网站内歌词、图片、试听歌曲等的版权信息，以及指向“联系我们”、“预订演唱会门票”等页面的链接。

### **上下文关系非常重要**

每个页面，尽管包含所有这些通用元素，本身还是应该是独一无二的。一个好的网页应该有效地履行一个或少数几个专门的功能。

### **相关内容**

要制作出非常优秀的网页，一个关键因素就是既要将内容分割，又要让他们彼此相关。各项内容既必须分别放置在各个不同的页面(这些页面具有各不相同的 URL 地址)，又必须有逻辑性地前后联系(在网站内和页面内都是如此)，这样才能便于被找到。

以乐队将举行的演唱会信息为例，尽管可以在每个页面上都放置一个“将举行的演唱会”模块，但还是应当将该项信息放在一个独立的页面中。一个简单的，链接到“巡演日期”页面的“下一场演唱会”模块，也可以有效地宣传该项信息，而不用处处都复制信息内容(这样可能把用户和搜索引擎都搞糊涂)。

### **标题**

下一次你们读报时，请仔细看看报纸的内容和版式设计。特别注意有一些报道篇幅更长，配有突出显示的字体和图片，标题也更醒目。这样做的目的是，如果你时间很紧而只想了解重大新闻的话，就可以立即发现哪些是重大新闻。

这一原则同样适用于网页。一个页面内每部分的内容都应有一个标题，以显示这部分内容

在页面内的相对重要性（这部分内容是从属于上一部分呢，还是与之同等重要？）

举例来说，在本篇文章的这个部分有两个段落标题“相关内容”和“标题”，它们都位于“上下文关系”这个大标题之下，显示它们都是从属于大标题之下小标题。

## 可用性

可用性是指一个网站能被用户以可以预期到的合理方式所使用。

设想一下以下几种情况：在你上一个新闻门户网站阅读一篇新闻时，在阅读前必须要在该网站注册；在你在网上预订飞机票或火车票时，还需要花两分钟通过电话向订票人员解释你的行程；在你输入一个邮件地址或信用卡号码时，网站只是告诉你输入格式错误；搜索发送回的结果中没有有用的条目，或是一个网站在其首页没有一个搜索工具。

以上都是网站可用性不好的例子，这源于没有考虑网站用户的需要。而在你构建和设计网站时，如果把网站用户的需要放在中心位置，就很有可能创建出令用户满意的好网站。

创建可用性好的网站并不是一件容易的事，这方面的很多知识都只能得之于经验。你可以记下其他网站可用性不好的地方，以避免重蹈覆辙。测试网站可用性最好的办法还是观察用户的实际使用体验。一旦你创建好一个网站，请从以下多个方面观察用户的使用体验：

- 用户能找到他们寻找的网页吗？
- 对用户输入的搜索主题词，搜索工具给出了正确的结果吗？
- 图像/音频/视频能在用户使用的浏览器中正常运行吗？
- 用户是否有对可用性不满意的地方？
- 用户感到满意和高兴的地方又在哪里？

专业公司对由其承担进行的网站可用性测试会收取很高的费用，但你可以选择非正式的测试方式，如让你的朋友和家人告诉你他们使用你创建的网站的体验，这样也可以让你很好地了解到一些你尚未注意到的网站存在的问题。这就是所谓当局者迷，旁观者清的道理。

## 可访问性

对网站可访问性最常见的误解就是仅将其理解为“让盲人能访问网站”，其实可访问性是一个对更多的人都有影响的问题。

“辅助技术”这一术语用于描述任何帮助人们与其使用的计算机更为有效地互动的辅助设备或硬件。你们可能马上会想到供盲人使用的屏幕阅读器，或是供无法使用鼠标或键盘的人使用的语音输入设备。但是眼镜呢？其实对近视的人来说，所戴的眼镜也是一种“辅助技术”，但是他们中绝大部分人不会认为自己是残疾人。

意识到很多使用互联网用户可能面临的问题，对制作出好的网页是非常重要的。不要想当然地就做出很多假定，如网站用户就一定有鼠标，就一定可以看到你使用的图像，就都安装了 Flash 播放器等。

在考虑网站的可访问性时，除要考虑那些需要使用“辅助技术”的人外，还要考虑其他一些人如用手机上网的用户。现在的手机还不能很好地支持 Flash（即使有 Flash 功能），例如苹果公司的 iPhone 就不具有支持 Flash 的功能（也许以后也不会有），虽然在其他方面用 iPhone 浏览网页的效果和在苹果计算机上用桌面版本的 Safari 浏览器浏览网页的效果差不多（Opera 手机浏览器也不支持 Flash）。一些新技术如 Opera Mini 手机浏览器，可以为上网的低电量手机重写网页，使网页体积变得更小，对绝大部分用户来说，网页中的图像在手机浏览器中也会被显示得小得多，这意味着用户可能无法获取网页中任何依赖于微妙的细节的内容。

在我们这个乐队网站的例子中，你应该意识到如果使用了很多图像（乐队照片），则必须描述图像的内容。如果你在页面中使用了 Flash 音乐播放器，以让人们可以听到乐队的歌曲，你同时也应该创建直接指向歌曲的链接，以让那些未安装 Flash 播放器的人也可以以他们选择的方式听到这些歌曲。

## 总结

在本篇文章中，我讲述了你们在开始实际创作网页时，需要牢记的一些重要概念，以使网页具有更好的可用性，能为更多的人访问，并能更为顺畅地运行。在以后的课程中，我们将详细讲述所有这些在本文中已提到的重要概念。

## 练习题

- 在做本篇文章所附的练习题时，你只需上网浏览一些你最喜爱的网站，试着以从本篇文章中学到的知识检视这些网站，并回答以下这些问题：
  - 这些网站具有一致的页面顶端部分、页脚部分和导航栏吗？
  - 观察你在网站各页面浏览时，导航是如何变化的。
  - 注意去发现网站是否有让你感到不便使用或让你迷惑的地方，如果有，请提出你的解决办法。
  - 如果可能的话，请关闭你使用的浏览器支持图像显示或支持 JavaScript 的功能，或是使用手机上网访问一个网站，与你使用计算机访问同一网站的体验做一个比较。
- 上一篇：信息架构——规划一个Web站点
- 下一篇：色彩理论
- 目录

•

## 作者简介



**Mark Norman Francis** 早在万维网诞生前，就在从事互联网领域的 works 了，一直持续到现在。目前他是全球最大网站 **Yahoo!** 的前端设计师，负责制定 **Web** 开发的最佳习惯、代码标准和质量标准。

在加入 **Yahoo!** 前，他先后在 **Formula One Management** (F-1 管理公司)、**Purple Interactive** (紫色互动公司)、伦敦城市大学从事过多种工作，包括 **Web** 开发、后端 **CGI** 编程和系统架构等。他的博客地址为：<http://marknormanfrancis.com/>。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 8. 色彩理论

Posted 12/10/2008 - 17:45 by Lewis

作者: Linda · 2008 年 7 月 8 日

- 上一篇: 一个好的网页需要什么?
- 下一篇: 建立站点的线框图
- [目录](#)

### 序言

如果网页上都没有很多的色彩和图像, 那么每个 Web 站点都会成为 Jakob Nielsen 心目中理想的 Web 站点。尽管 Nielsen 的简化 Web 架构的设计理念具有强调可访问性和可用性的优点, 多数网页设计师还是向他们设计的Web站点加入很多设计元素, 以使站点具有设计师个人的风格。值得庆幸的是, 如果网页设计师掌握了正确的理念和方法的话, 是可以既加入色彩让站点具有好看的外观, 又不会损害站点的可访问性和可用性的。尽管很多网页设计师在设计供很多用户使用的网站时感到工作并不难, 但是他们在选择色彩和图像时, 可能还是会感到有些力不从心。

在本篇文章中, 我将讲述色彩理论的基础知识以及 3 种简单的配色方案, 以让你们在为 Web 站点选择色彩时能有把握。在后面的文章中, 我将讲述如何简化这些色彩选择。听着别人对你设计的 Web 站点的赞扬, 毕竟比苦心孤诣地去进行色彩选择更让你愉快。本篇文章的目录如下:

- 颜色、亮色、暗色
  - 单色方案
  - 互补色方案
  - 暖色 vs. 冷色
  - 三色方案
  - 四色方案
- 总结
- 练习题

### 颜色、亮色、暗色

颜色，或色调，一般被划分为原色、间色、复色。原色包括红色、黄色和蓝色，它们被称为三原色，这是因为其它颜色不能调配出这三种颜色。在将三原色转换为 Web 颜色时，你可以将它们表示为十六进制的颜色代码，分别为 #ff0000, #ffff00 和 #0033cc，如图 1 所示：

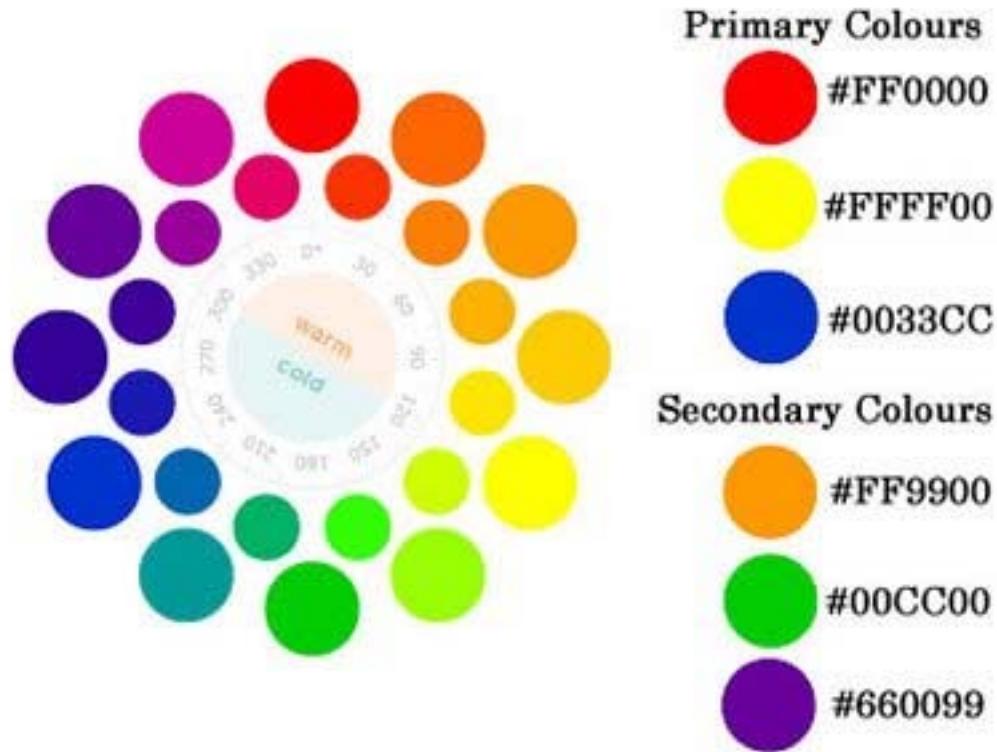


图 1：原色和间色，以及它们的十六进制颜色代码

间色指红、黄、蓝三原色中的某二种原色相互混合的颜色，包括：

- 红 + 黄 = 橙色 (#ff9900)
- 黄 + 蓝 = 绿色 (#00cc00)
- 蓝 + 红 = 紫色 (#660099)

复色由间色混合调配而成，它们位于上图色环中所示的原色和间色之间。尽管 Web 颜色与“画家”通常使用的颜色并不相同，但手头有一个色轮（色轮如图 2 所示），还是有助于你学习各种配色方案。此外，色轮还显示所有的亮色（tint）、灰色调（tone）和暗色（shade），这样你就可以认识到各种可能的颜色组合和搭配。以下列出了需要学习的几个重要术语：

- 亮色（Tint）：加入白色时显示出的颜色
- 灰色调（Tone）：加入灰色时显示出的颜色
- 暗色（Shade）：加入黑色时显示出的颜色

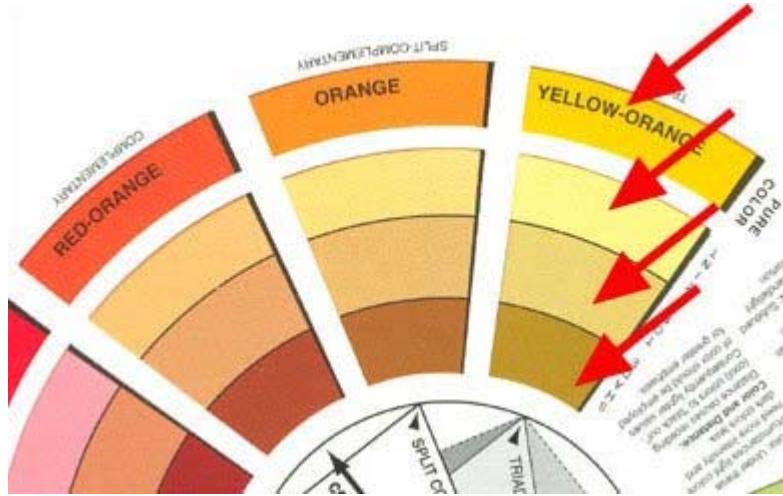


图2：色轮实样

下面为图2中箭头所指的色带的说明：

- 最外层色带：黄色和橙色调配而成的复色
- 第二色带：该复色的亮色（加入白色）
- 第三色带：颜色的灰色调（加入灰色）
- 最内层色带：字轮上的暗色（加入黑色）

正如你们从以上的色轮中看到的那样，向一种颜色加入的白色、灰色和黑色的数量是很小的，够改变初始颜色和创建出所谓“**单色方案**”即已足够。

### 单色方案

配色方案已通行很久了，因此没有必要再重新设计一个色轮。尽管 Web 颜色与印刷颜色并不相同，但概念是一样的。你只是把颜色名称转换为 16 进制的颜色代码，并使它们尽可能地完全匹配。我建议你们使用一个在线工具，即 [ColorScheme Generator II](#)（配色方案生成器II，如图3所示），它不仅可以帮助你迅速和容易地确定配色方案，甚至还可以帮助你确定你选定的颜色是否为视力差或是色盲的用户提供了足够的对比度。

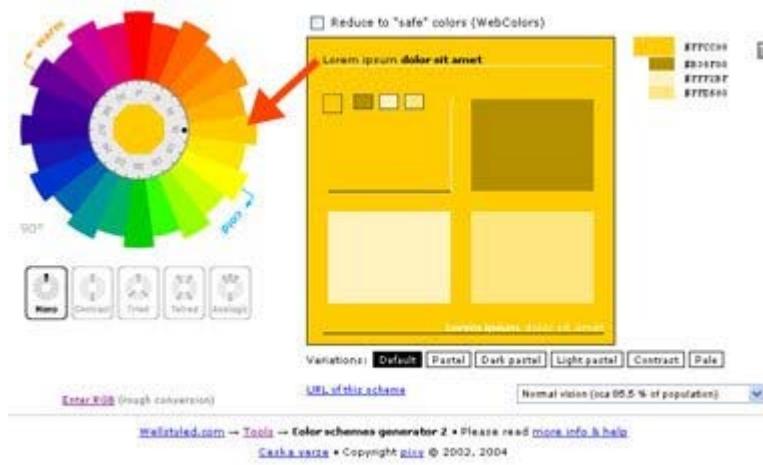


图 3：配色方案生成器 II

如果你在确定你选定的颜色是否提供了足够的对比度时需要更多的帮助，可以使用我们 Paciello 小组提供的 **Contrast Analyser**（对比度分析器）。这个工具可检查前景颜色和背景颜色之间的对比度。

为了实现在颜色生成器中生成黄橙色的亮色、灰色调和暗色，您要首先选择上图中箭头所指的颜色，然后选择色轮下方面板上的 **Mono**（单色）和图右盒子下方面板上的 **Default**（默认），并在右底部的下拉式菜单中选择 **Normal Vision**（正常视觉）选项。除非你是一个纯粹主义者，不要在颜色盒上方的“reduce to ‘safe’ colours”（降至‘安全’颜色）方框中打勾。

**备注：**“Web 安全颜色”这个术语，产生于显示器刚可以显示 256 色的那个时期，在 256 色中，只有 216 个颜色在 Windows/Mac/Unix 平台上都是一样的，因此就出现了“Web 安全颜色”这个名称。尽管一些纯粹主义者依然坚持使用“Web 安全调色板”，但现代的浏览器已经能够处理所谓“24 位色”。每个通道有 10 至 11 位的 24 位色，实际上已能生成 16,777,216 种不同的颜色。换言之，我们已经可以有把握地说已不再需要“Web 安全调色板”了。

让我们再回头来看单色方案。遵照以上所描述的步骤，生成的颜色结果如下：黄橙色 (#FFCC00)、亮色 (#FFF2BF)、灰色调 (#FFE680)、暗色 (#B38F00)。与试图通过将一个有形的色轮与一个 Web 浏览器的背光屏幕相匹配而做出的任何猜测相比，这些 16 进制的数字的可靠度要高得多。同时，就像“Mono”（单色）所建议的，将这个方案转化为一个单色方案，如图 4 所示。



图 4：一个单色方案

一个单色方案等同于一种颜色，及其所有的亮色、灰色调和暗色。尽管这个配色方案是最

容易使用的，但很多网页设计师在进行网页设计时，都不是很喜欢这种配色方案。因此，你可能希望学习其它的配色方案，为网页中的链接、图像和横幅广告添姿增彩。

### 互补色方案

下面我们学习互补色方案，即通过直接搭配色轮中相对的颜色，如图 5 所示。

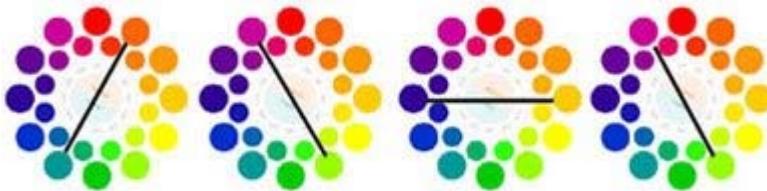


图 5：互补色方案的示例

当你选定一种颜色及其互补色时，你同时也选定了与这两种颜色相关的亮色、灰色调和暗色。这提供了更广的颜色选择范围，通过在线颜色工具可以很好地转化为互补色方案，如图 6 所示。

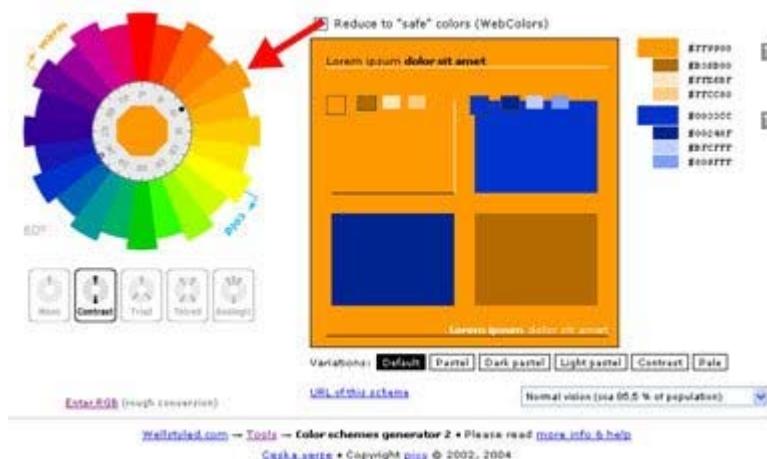


图 6：在线颜色工具生成的一个互补色方案

在上图中，我选择了橙色及其互补色蓝色。为生成这个配色方案，我选择的设置包括：色轮左下的 **Contrast**（对比色）设置，生成器下方菜单中的 **default**（默认）设置，以及 **normal vision**（正常视觉）设置。请注意，所选定的主色在色轮的内盘上以一个黑色圆点标示（在上图中和生成器站点上都是这样标示的），生成器自动为你选出该主色的互补色，以内轮缘上的一个空心圆标示。有了这些标示符号，你就可以更容易地分析你的配色方案。

这个颜色生成器让你可以容易地为链接、访问过的链接，甚至是图像选择颜色，因为它在右上角为你提供了颜色的十六进制代码。你可以混合和搭配任何纯色（顶部列出的颜色）及其亮色、灰色调或暗色，大大有助于你选择一个很好的配色方案。



图 7: The Greenpeace site (绿色和平组织的网站) —— 使用互补色方案的一个模范实例

美国绿色和平组织的网站（见图 7）是很多使用对比色配色方案的网站中的一个，在这个网站的截图上，你可以看到黄色和橙色，但主导颜色还是绿色和红色，这是两种在色轮中直接相对的颜色。使用这种互补色配色方法，你几乎从不会出错。实际上，使用“暖色”和“冷色”的搭配，可以使网站页面显得活泼，具有活力。

### 暖色 vs. 冷色

在网站中使用互补色方案效果非常好，这是因为互补色方案同时包含暖色和冷色。使用暖色和冷色提供了一种对比的效果。同时，哪些颜色是暖色，哪些颜色是冷色，是很容易记住的，你在图 8 中就可以看到（在生成器网站上也可以看到）标出的暖色和冷色：

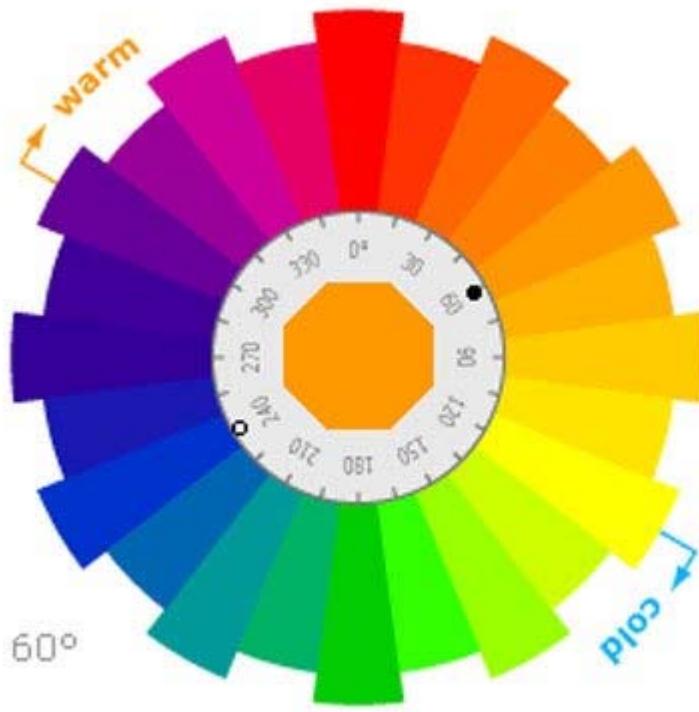


图8：暖色和冷色

暖色是那些可以让你联想起夏天、太阳或火的颜色，包括色轮中从紫色起到黄色止的多种颜色。而冷色则是那些可以让你联想起春天、冰或水的颜色，包括色轮中从黄绿色起到紫色止的多种颜色。如果你仔细观察一下色轮，就会很快发现你选择一种主色后，生成器自动选出的相对应的互补色在冷暖色方面与主色是相反的。因此，如果你选择“暖”红色，相对应的互补色就是“冷”绿色；或如果你选择“冷”绿蓝色，则相对应的互补色就是“暖”红橙色。

Ecolution 网站就是在网站中各页面都使用暖色和冷色搭配的一个实例，如图 9 所示。



图9: Ecolution 网站——使用暖色和冷色搭配的一个模范实例

Ecolution 网站在其主页上通常使用红色作为强调色，以和绿色的网站标识形成对比。他们接着将这两种对比色的亮色、灰色调和暗色在页面中各处混合搭配。某个图像中的“黑色”甚至都可以偏“暖”或偏“冷”，白色也同样如此。总体来看，这张网站截图的主色调是“暖色”，加上鲜明的纯绿色作为搭配，整体视觉效果非常好。尽管 Ecolution 网站使用的是和平组织网站一样的颜色，但由于 Ecolution 网站使用了很多灰色调和暗色，因此其外观就没有和平组织网站的外观那样“耀眼”。

你从来没有想到色彩理论会如此简单易学，对吧？那好，下面我们就学习略微高深和复杂一些的内容。

### 三色方案

三色方案是指在色彩圆环中选择一个等边三角形三个顶点上的颜色构成的配色方案，如图 10 所示：

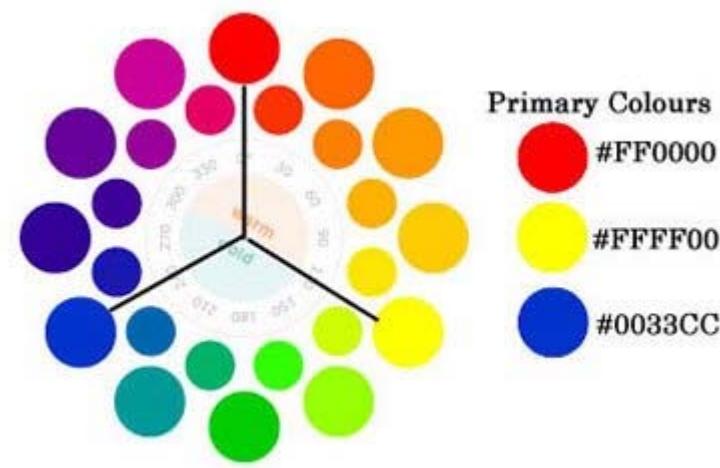


图 10：三色方案

我为这个配色方案选择了原色，原因是想显示在配色方案中可以包含这种可走向极端的配色方法。原色在色轮中所处的位置绝非偶然，因为在每个原色之间的颜色，都包含有相等数量的间色和复色。但由于原色三色方案已被过度使用，所以它看上去似乎有些过时了。你可以使用在线颜色生成器，选择其它颜色来生成一个三色方案，如图 11 所示：

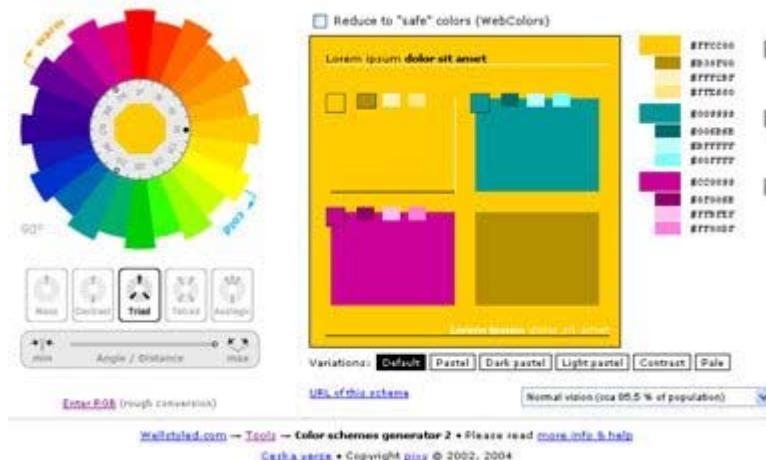


图 11：另外一种三色方案

以上所示的这个三色方案由橙黄色、蓝绿色、红紫色构建而成。我首先选择橙黄色（请注意上图中，色轮左边内圈上的那个黑色圆点），然后再选择色轮下方面板上的 **Triad**（三色）选项，生成器自动选定三色组合，包括所有的亮色、灰色调和暗色。在色轮上，伴随色以空心圆点标示，如同前面那个单色例子中互补色的标示一样。

现在，我们手上需要一个真正的色轮，因为在线生成器的结果与真正色轮的结果并不是很一致。不过当我将色轮下方的 **Angle/Distance**（角度/距离）工具拨到“max”（最大）时，在线生成器的结果与真正色轮的结果似乎就完全一致了。上图所示的结果是最接近真正的色轮生成的

结果。

三色方案也包含有暖色和冷色，但以一种暖色或冷色为主导色。一般说来，那种可以让其它颜色显暗的暖色或冷色应是你最先选择的。在我的这个三色方案中，我最先选择的是橙黄色，它是一种暖色。因此，在这个三色方案中，暖色就是主导色，其它两种颜色中有一种是冷色，以产生对比。



图 12: *Puzzle Pirates* 网站——三色方案的一个模范实例

*Puzzle Pirates* 网站（如图 12 所示），在其主页上使用了三色方案。该网站使用的是三原色（红-蓝-黄）方案，这个三原色方案非常适合一个少儿游戏网站。注意在该页面中，蓝色是主导色，红色和黄色被用作强调色，散布在页面各处。

#### 四色方案

你选择的颜色越多，则配色方案就越复杂。不过窍门是，选定一种亮色、灰色调或是暗色，并注意面板上那些，与选定颜色形成交叉的区域，而不是去混合调配纯色及它们的亮色、灰色调和暗色。这种方法很适用于四色方案这样的配色方案。这种四色方案（见图 13）与互补色方案很相似，不同之处只在于使用了两组等距的互补色。

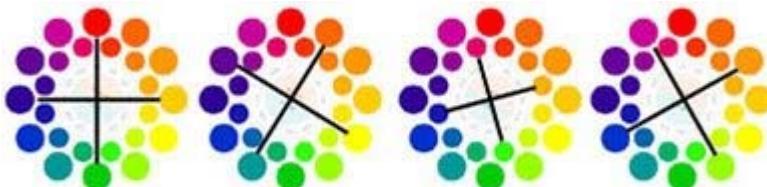


图 13: 四色方案

图 14 显示如何使用在线生成器生成一个四色方案:

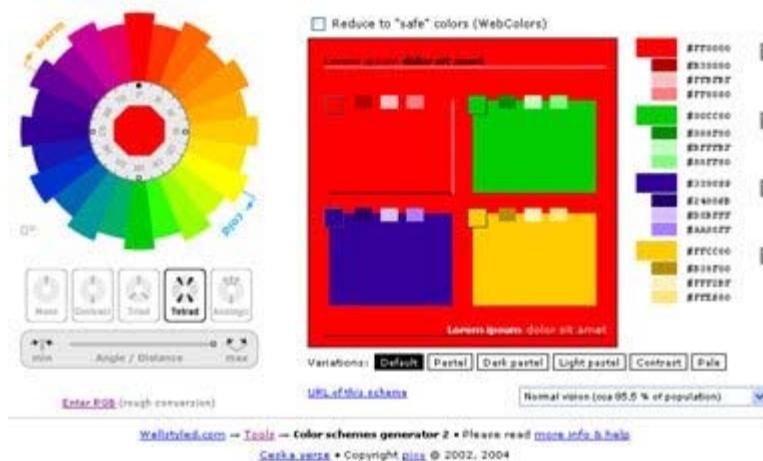


图 14: 在线生成器生成的一个四色方案

注意色轮左边红色下面的那个黑色原点，红色是我选择的第一个颜色。然后我点击色轮下方面板上的 **Tetrad**（四色）按钮，显示出来的四种颜色与我的手持式色轮显示出来的颜色又有点不同，不过当我将色轮下方的 **Angle/Distance**（角度/距离）工具拨到“max”（最大），在线生成器生成的结果与我的手持式色轮生成的结果似乎就完全一致了。以上所示的结果是最接近真正的色轮生成的结果。

这种配色方案可以变得相当复杂，因此你可能会从右栏中的颜色中挑选出所有 4 种亮色或灰色调或暗色，为此你可以点击右端的箭头。例如，图 15 就显示了一个用这个配色方案的亮色填充的方块：



图 15：四种亮色

图 16 显示了一个中间色调的四色方案实例：

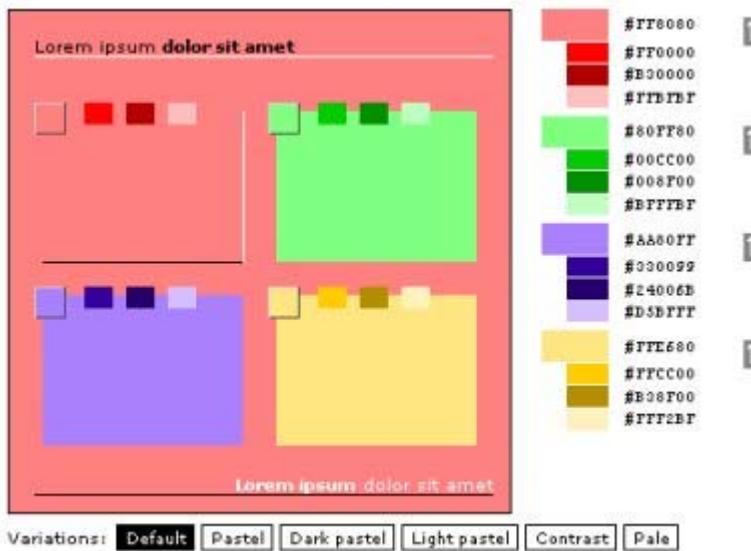


图 16：四种中间色调

如果你仔细观察上面所示的两个方块，就会发现生成器还为你提供了四个单色方案。这些单色方案同时显示在右栏内和大方块内的每个小方块内。



图 17：Jane Goodall 研究所的网站——四色方案的模范实例

Jane Goodall 研究所的网站（图 17）是少数几个成功运用四色方案的网站之一。请注意网站截图中的紫色、黄色调、高亮红色（页面下部使用了更多的红色）、绿色。紫色与在线颜色工具生成的配色方案并不完全一致，它更偏于像红紫色。不过作为一个显示你如何能同时运用色轮和在线颜色生成器为你的网站确定配色方案的例子，它的效果已经很好了。

当你在网上搜索配色和设计的创意时，请将色轮放在手边，以更多地了解网页设计师是如

何在你喜爱的 Web 站点上使用配色方案的。

## 总结

尽管色彩组合可能看起来有些复杂，但其实所有的配色方案都是遵循一定的“规则”的。了解这些指导原则，可以使我们易于理解色彩的搭配是如何为 Web 站点增添趣味和对比度的。

色轮就是为我们所用的有用工具。有了色轮和其它工具如在线颜色生成器等，就是尚缺乏经验的设计师也能容易地挑选出颜色。

本文所讲述的四种配色方案分别为：单色方案、互补色方案、三色方案、四色方案。尽管还存在其它的配色方案，但这四种配色方案是最容易理解和运用的。

## 练习题

备注：最后两个问题的答案不存在对错之分。

- 请说出三原色的名称，并解释它们为什么会被称为原色。
- 请说出三种间色的名称，以及它们是由哪些原色混合而成的。
- 描述亮色、灰色调、暗色是如何生成的。
- 什么是单色方案？
- 什么是互补色方案？
- 描述“暖色”和“冷色”。
- 什么是三色方案？你能选出适合这种方案的三种颜色吗？
- 什么是四色方案？你能选出适合这种方案的四种颜色吗？
- 哪种配色方案看起来最容易使用？
- 哪种配色方案看起来最复杂？
  
- 上一篇：一个好的网页需要什么？
- 下一篇：建立站点的线框图
- [目录](#)

## 作者简介



Linda Goin 拥有美术学士学位（主修视觉沟通，副修商学和营销学）和文学硕士学位（主修美国历史，副修宗教改革史）。她的第二个学位所学的专业似乎与她的第一个学位所学的专业相距甚远，但她已将从事网站设计 25 年来积累起来的专业知识用于考古发掘领域和物质文化的研究领域。

她的作品曾获得过很多奖项：包括获得过 15 次科罗拉多州新闻社大奖的第一名，多个美术和图像设计大奖等。《华尔街日报》、《芝加哥论坛报》、《今日心理学》、《洛杉矶时报》等报刊都曾对她进行过关于内容开发的专访。她著有多部关于网页设计、可访问性等主题的电子书，此外她还为一些搜索引擎优化专业团队代笔写作，并撰写有关个人理财的文章。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

## 9. 建立站点的线框图

Posted 12/10/2008 - 17:45 by Lewis

作者: Linda · 2008 年 7 月 8 日

- 上一篇: 色彩理论
- 下一篇: 配色方案和设计样板
- 目录

### 序言

在开始设计一个 Web 站点前，每一个 Web 设计师都应当知道和了解 Web 站点的各项参数。在本篇文章中，你们将学习开始设计商业 Web 站点时需要具有的基本知识。这些知识和信息不仅对你为其他人构建 Web 站点很有帮助，而且可以作为你构建自己的 Web 站点时适用的检查清单。这个阶段的工作，是 信息架构 的后续工作。在这个阶段，你应当了解客户希望他们的Web站点包含哪些内容，站点的结构应该是什么样的，客户公司使用哪种类型的品牌，然后基于这些了解到的信息，建立一个可视设计模型，在征得客户的认可后，再加入图形图像和配色方案。我将特别讲述以下几方面内容：

- 尽管配色和设计很重要，但你首先需要了解客户建站的目的和目标。客户建站的目的和目标，对站点的外观和总体风格有着重要影响。
- 因此在开始进行网站设计前，你需要编制一份重要事项清单，以了解客户的各项重要信息。
- 你还需要了解客户公司以前所做的营销工作，包括品牌推广工作，这些信息对网站设计很有用。
- 基于从客户那里收集到的所有信息，你将创建网站的视觉设计文件，为客户添加额外的图形图像和内容，提供可视的基础性文件。

本篇文章的内容目录如下:

- 你需要了解哪些信息
- 第一步
  - 一个虚构的示范网站
  - 标识 (Logo)
  - 页面布局

- 关于网站上的广告
- 通过校验代码和征得客户的认可，确认页面布局
- 总结
- 延伸阅读
- 练习题

### 你需要了解哪些信息

一般说来，在决定进行网站设计前，个人或企业应该已确定建站的目的和目标。尽管配色和图形图像设计很重要，但你首先应有一份建站计划，列明建站预算、目标市场、预期目标，以及完成这些工作任务所需的资源等。你需要了解这样一些信息：该网站只是向用户提供信息，还是还计划向用户销售产品或服务？该网站是将长期存续并发展，还是只是想在短期内以网站作为进入一个细分市场的营销手段（如竞选网站、针对当前的时尚而建立的营销网站等）？网站将包括博客页面、法律信息页面、图片集、电子邮件联系表单吗？网站还需要其他哪些页面？与竞争对手的网站相比较，该网站如何？

最后，还有一个很重要的问题，那就是客户公司已经建立起品牌了吗？如果尚未建立起品牌，则你在开始进行网站构建和设计前，需要特别注意这一点。也许你不具有创作公司标识、为针对某一特定市场的产品或服务进行品牌建立和推广、进行市场营销等的技能，因此如果你尚未履行过这些营销工作，你可能需要一个营销专家与你一道工作。当然如果客户公司已有一个完整的建站计划，你就必须遵循他们的指示去做，这样建好的网站才会到达他们的要求和目标，成为客户公司的重要营销工具。

尽管很多此类信息在建站任务交到设计师之前可能就已经确定了，但了解以上那些问题的答案，还是可以帮助你确定是要建立哪种类型的网站、使用什么配色方案，以及要加入哪些图形图像等。不过有一点在建站前就可以确定，那就是网站必须具有可用性和可访问性。因此，在所有情况下，写代码和设计导航都是工作重点，关于网站的可访问性，将在以后的课程中更详细地讲述（本课程中关于可访问性的专题文章将很快发布）。关于网站的可用性，可以访问 [Jakob Nielsen](#) 的主页，了解他关于可用性的一些重要观点。

关键一点是要通过分别使用 **HTML** 和 **CSS** 编写代码和进行设计，使网站易于访问。除非 **Flash** 适用于网站的一些元素，要尽量避免使用 **Flash**（为提高 **Flash** 的可访问性，已经做了很多工作，参见 [相关文章](#)。**Flash** 适合履行一些如播放视频这样的任务），并仔细思考在哪些地方需要使用 **JavaScript** 和其他一些技术。这样做，使设计师设计网站和程序员创建网站程序的工作都更为容易（如果设计师同时也是编程员，则更是如此）。同时，建立起来的网站在浏览器兼容性方面的表现也会更好。

## 第一步

为帮助你们解决这些问题，我将做一个示范，用我设计网站时使用的一套指导原则，构建一个简单的公司网站。我使用的检查清单将既包括有关公司的事项，也包括有关设计的事项。为方便起见，我假设这家公司是一家成熟的公司，已有一些现成的营销材料，公司标识、品牌等都已经有了。而如果你是从零开始工作的话，那么在开始开发公司网站前，还需要先制订出创建公司标识和品牌的计划。

作为一个 Web 设计师，我在开始为一家公司设计网站前，需要了解公司的以下信息。我想把所有重要的有关网站设计的信息都记录下来，这样我就不再需要在以后做出重大的改动。在为一家公司设计网站时，你都需要与公司所有者/决策人讨论以下这些主题，以确保你对这个网站设计项目的愿景是和他们的愿景相一致的。

1. 网站名称：网站名称是否反映了公司名称及其在网上工作？在我的这个示例中，网站的名称就是公司的名称，即“Wiki Whatevers”（无处不在的维基）。如果该公司尚未使用一个 **tag line**（宣传语）的话，可能还想创制一个宣传语。这样在网页上宣传语、公司名称和公司标识就将放在一起显示。
2. 公司标识和品牌：我想收集该公司此前已编制的所有印刷品，包括公司标识（Logo）、宣传册等，这样我就可以建立起保存信息（如电话号码和地址等）的文件夹。通过这些资料及公司设计团队已设计出的公司标识，我就可以更好地理解这家公司的整体形象、品牌和风格。如果公司尚未有标识，那么我就想聘请一个标识设计团队来为公司设计一个标识（我自己并不是一个标识设计师，因此会让别人来设计，你也可以和我一样这样做，并把产生的费用写入账单中）。
3. 网站的域名：在确定网站名称后，我想知道相应的域名是否可用。域名是一个网站的网上地址，用户在浏览器的地址栏键入域名即可达到网站。域名同时也用作从外部资源访问网站的链接。域名可选择任何一种 顶级域名，如“.com,” “.org,” 等。尽管设计师通常并不负责进行域名注册，但了解域名是否已选定和注册，对设计工作还是很有帮助的。在一些情况下，由于域名不可用，我不得不改变网站的名称和网站的一些内容。出现这个问题，将使客户发生更高的费用，而如果一开始就选定域名，就可以避免出现这种情况。
4. 研究竞争对手的网站：了解竞争对手的网站包含哪些图形图像和内容，对你设计网站是很有用的，这样你就能设计出至少和该公司主要竞争对手的网站一样好或更好的网站。
5. 信息架构：网站需要有一个购物车系统或博客系统吗？网站所有者有什么样的网站扩展计划？采用什么结构将各个页面链接在一起？这些都是很重要的问题，你需要将它们纳入网站设计和导航设计之中。你需要知道网站在未来将如何扩展，这也将决定你怎样构建网站。

6. 网站内容：网站内容已编写出来了吗？如果已经编写出来，你会希望立即就获得这些内容，以帮助你确定导航、排版设计和布局。将内容分类是确定导航的最佳方式。此外，内容也可以帮助你确定网站的外观和总体风格。因此如果网站内容尚未编写出来，最好还是不急于开始设计工作。要确保内容的相关性，并规划好内容的更新，因为网站访问者最终是为了获取内容才一次一次访问网站的。
7. 研究虚拟主机：尽管客户可能中意某一个虚拟主机提供商，但由于不是所有的虚拟主机提供商都提供一样的技术支持，因此你可能还是需要了解其他一些虚拟主机提供商的情况。虚拟主机提供商提供网站寄存服务，其中一些虚拟主机提供商提供访问数据库的服务（你设计的网站的博客系统或购物车系统可能需要数据库支持）。一些虚拟主机会限制访问网站的用户人数，如果访问的网站用户越来越多，这就可能造成问题。你可以访问虚拟主机数据库（[WHDb](#)），获取载明有很多虚拟主机提供商的名称及其提供的服务、技术能力等的列表。在你开始进行网站设计前，请确认客户已购买了虚拟主机空间，这样你就能知道你的设计参数。
8. 规划用户如何离开网站：这意味着你和你的客户控制用户将如何离开网站。用户在访问网站后，最终还是要离开的，因此为什么不通过可产生收益的广告放置或通过链接交换来事先规划用户的离开呢？做这种规划，并且通过向网站用户提供某种服务，可以为网站增加金钱上的收益。
9. 任务截止日期：确定网站什么时间“上线”。一般说来，只要客户已准备好内容，对你的配色和布局设计样本表示满意，且不需要复杂的编程，那么完成一个小的网站设计和构建项目（如我示例的这个项目），8周时间就已足够。

一旦你已获得了这些基础信息之后，你就可以开始着手工作，阅读内容，规划导航，并决定如何对网站进行搜索引擎优化（SEO）。尽管你可能并不负责从事搜索引擎优化的工作，但你可以和 SEO 专家紧密合作，确定如何最好地使用 网站的内容和代码，通过内容和大标题、小标题中的关键词，使网站的流量增加。

正如你不会在建筑师编制出蓝图前就去为新家挑选地毯或沙发一样，你也不会在规划好网站的架构前就去进行网站的视觉设计。在这个最初的规划阶段就规划导航和 搜索引擎优化可以使你在后续工作中少走弯路并节省时间。在你准备好开始进行网站的视觉设计时，你已经熟悉网站的内容和架构，这样配色和图形图像设计工作就容易做得多。

### 一个虚构的示范网站

这个虚构的网站是一家向 [wikis](#)（维基网站）提供开源代码的网站，每周至少要发布三次新的开源代码。由于开源代码可以免费使用和修改，网站所有人希望通过捐赠、广告放置，以及其程序员提供的额外服务来为网站产生收入。该网站的名称为“[Wiki Whatevers](#)”（无处不在的维

基), 域名也已选定。网站的内容已编制出来, 包括尚需分类的代码段、文章, 以及参加这个项目的程序员的个人传记。寄存网站的虚拟主机提供可用的 MySQL 数据库, 且不限制流量。现在需要列出网站将要使用的各项要素:

1. 使用公司已有的标识 (Logo)。我想将客户公司的 Logo 数字化, 并在网站的各个页面都使用这个 Logo。为此我需要一台扫描仪, 将 Logo 扫描入一个图形程序(如 Photoshop 或 Gimp)。我将在确定页面布局后, 再确定 Logo 的尺寸。我将把 Logo 图像设为 72dpi, 以使 Logo 图像能被更快地下载。
2. 我想在介绍公司员工的页面 (或“About”——关于我们页面) 使用程序员的照片, 因此我就需要他们的数码照片。他们可以提供普通照片供扫描成数码图片, 或直接提供数码图片。如果他们提供的都是数码图片, 我需要其解析度比我最终使用的图片的解析度高, 因此 300 dpi 的图片就很合适, 或者提供全尺寸的图片也可以, 我以后可以根据情况缩小图片的尺寸。
3. 由于客户已有足够的内容让博客在未来几个月内都保持活跃, 客户决定使用一个博客系统。值得庆幸的是, 客户选定的虚拟主机提供商熟悉博客系统, 有能力处理数据库和很大的网络流量 (包括高峰流量)。这个虚拟主机提供商还提供网站扩展的多个解决方案, 如果客户希望扩展其网站的话, 可以很容易地实现。如果虚拟主机的正常运行时间可以得到保证的话, 客户会在网站整个成长期内都将网站寄存在这个虚拟主机。能持续数年都将网站寄存在同一个运行状况良好的虚拟主机上, 会省去很多麻烦。
4. 使用 FTP(有多种 FTP 软件可供选择, 如开源 FTP 软件 Filezilla, 或适用于 Firefox 的 fireftp, 或专有的FTP客户端软件如 CuteFTP), 我将上传一个宣示网站即将上线的静态网页。“网站建设中”是一个应避免使用的短语, 这是因为访问者如果不知道网站的上线日期, 可能就不会再回来访问网站了。更适宜的做法是, 先上传一个网页, 在其上载明公司的名称、网站将提供些什么内容、网站的上线日期, 以及联系信息 (留一个电子邮件地址就很合适, 如果公司是一家传统行业公司, 也可以同时留下地址和电话) 等。此外, 如果能使用一个可以向个人发出网站上线通知的电子邮件表单, 那就更好了。先上传这样一个网页, 可使客户在其网站正式上线前, 就有可能建立起潜在顾客群。
5. 使用从客户那里收到的网站内容/结构信息, 以及确定客户希望在每一个页面都预留广告空间。我将设计网站的架构并规划导航和文本链接。此外我还将运用这个设计样本, 规划用于对网站进行搜索引擎优化的关键词。
6. 为 Logo 配色。我将选定两个或三个配色方案提交给客户批准。
7. 然后, 我会上图库网站如 iStock 或 Comstock, 选择其他图片或插图。不过要注意一点, 需要多上几个图库网站, 选择最优惠的交易条件和价格。使用图库网站提供的图片, 费

用并不昂贵，且避免让你自己去处理让人头疼的 版权问题。此外，我还需要客户公司已经创作好的或将要创作的所有图片，以在网站各页面中使用（包括提供代码的页面、文章页面、博客页面等）。

**备注：**以上第 6 和第 7 阶段的工作，将在下一篇文章中详细讲述。请一定记住，在开始在为各个页面配色和添加图形图像前，需要先征得客户对网站视觉布局样板的批准。

### 标识 (Logo)

Logo 是公司品牌的视觉展示。由于 Logo 要在长期内代表公司，多数公司都会精心设计他们的 Logo，不过也有一些客户对 Logo 这个代表公司的标识不太在意。根据我自己的经验，如果一家公司尚未花费时间和金钱设计出一个专业的 Logo，则通常永远不会花这笔钱——无论你有多么具有说服力的论据要他们这样做。

Wiki Whatevers 公司的所有权人，都在乔治亚理工学院就读过，所以在设计公司 Logo 时，就选用了他们母校校标的颜色——金色和黑色。这个 Logo 设计得很简洁，至少让你在配色和进行页面布局时会容易一些。公司的 Logo 如图 1 所示：



图 1：Wiki Whatevers 公司的 Logo

我在此使用的是扫描的公司 Logo，意图在于在公司网站上使用和他们在印刷品上所使用的一样的 Logo。印刷色为 CMYK（青、品红、黄、黑四色），与由 RGB（红、绿、蓝三色）组成的 Web 颜色并不匹配。因此我需要做一些配色工作，以使网站上 Logo 的颜色与公司印刷品上 Logo 的颜色尽可能地匹配。为实现这个目标，有以下四种方式可供选择：

1. 联系 Wiki Whatevers 公司的印刷商，询问他们在此前是用什么颜色将公司的 Logo 印刷在公司的印刷品上的。通常情况下，印刷商会使用 Pantone 颜色。Pantone 公司还提供可帮助设计师将印刷色与 Web 颜色相匹配的工具。印刷商那里可能已有 Pantone 配色系统，因此他们就可以帮助你将印刷色与 Web 颜色相匹配，不用再花费额外的钱去购买 Pantone 公司提供的其他工具。

2. 由于这些 **Wiki Whatevers** 公司的所有权人在 **Logo** 中使用的是乔治亚理工学院校标的颜色，我可以去访问乔治亚理工学院的网站，看是否能找到相匹配的颜色。你可以 使用一个图形程序从网站截图上提取一种颜色，将图像导入一个图形程序，使用取色工具或其他工具来配色。
3. 目视印刷品上 **Logo** 的颜色和 **Web** 颜色，使它们尽可能地匹配。在一些情况下，它们之间的差别可能相当大；而在另外一些情况下，它们之间会非常匹配，甚至不用再做修改。
4. 用一个接受 **CMYK** 颜色的扫描程序扫描印刷的 **Logo** 图像，并使用 **Photoshop** 软件的 **Pantone Colour Swatches**(Pantone 颜色样本)使印刷色与 **Web** 颜色尽可能地匹配。不过使用这种办法，你的扫描仪必须要能接受 **CMYK** 颜色，且你必须要有 **Photoshop** 软件。

就我而言，我从 乔治亚理工学院运动队网站上的吉祥物图案中抓取到了最匹配的金色。这里金色的颜色代码为 **#eab200**，黑色的颜色代码为 **#000000**。背景色为深绿蓝色（颜色代码 **#002123**），用于 **Logo** 的下拉式阴影。还有什么比从这个大黄蜂吉祥物图案（见图 2）中抓取相匹配的颜色更容易呢？



图2：用于为 **Logo** 配色的乔治亚理工学院吉祥物图案的一部分。

**附注：**你很少会遇到有公司没有在名片、信头等上面使用其在网上的公司 **Logo** 或品牌标识的。不过很多此类公司似乎都接受网站上呈现出来的公司 **Logo** 等的颜色，而不是去对颜色进行更改，以与印刷品上的颜色相匹配。因此，不要总是信赖一家公司网站上的 **Web** 颜色，尤其是 **Web** 颜色与该公司印刷品（如宣传册或信头）上的颜色并不匹配时。更适宜的做法是，询问公司偏向于选择哪种颜色——也许他们一开始并没有注意到 **Web** 颜色和印刷色是有差别的。

## 页面布局

为简便起见，以下我将只示范一种页面布局。我选择的是这样一种博客页面布局：把更新频率最高的部分放在页面主体区域的上部，把导航栏放在页面顶部和包含最近更新内容的那部分

页面主体之间，便于访问者点击，同时把过去的帖子放在页面的下端（**below the fold**，即除非用户向下滚动屏幕，否则无法看到）。“**below the fold**”（字面意思是对折线以下—译注）这个术语起源于报业，即当报纸被放在售报台上销售时，读者将只看到报纸对折线以上（“**above the fold**”）的部分，因此这部分在对折线以上的文字和图片是很重要的，因为它能促使读者购买整份报纸。

这一“将重要部分放在页面上端”的理论同样适用于网站设计。用户进入网站后不向下滚动屏幕就能看到的页面部分即是“**above the fold**”部分，而用户需向下滚动屏幕才能看到的页面部分即是“**below the fold**”部分。因此，一个页面布局的诀窍就是：要让访问者无论是在任何分辨率的显示器上首先看到的内容和图片都能引起他们的注意（这也是你需要在多种不同的显示器/屏幕分辨率下测试你的网站的原因之一）。图 3 为 **Wiki Whatevers** 网站的博客页面布局草图：

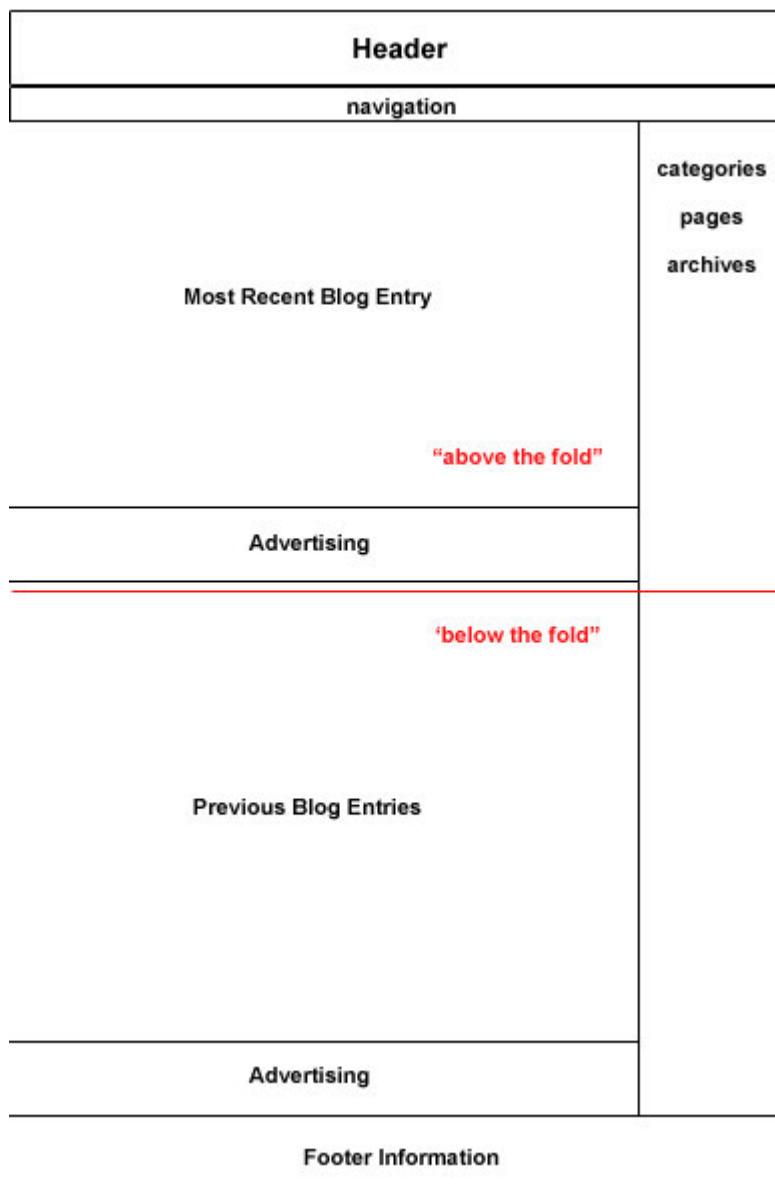


图 3: *Wiki Whatevers* 网站的博客页面布局草图（线框图）。

网站各个页面都将沿用这一页面布局，不过用于列出无图像的文章列表和博客条目列表的存档页面，页面布局可做适当修改。之所以要保持页面布局的一致性，其原因是不让网站访问者感到困惑。一旦用户“学会”如何使用一个网站，一般情况下都不喜欢看到各个页面的布局有很大变化。以下是这个页面设计包含的主要部分：

1. **页面顶部:** 页面顶部标题部分较小，我不希望让网站 **Logo** 占据太多的页面空间。尽管 **Logo** 较小，但 **Logo** 中的颜色将是网站的整体配色方案中的一个重要部分。页面顶部标题部分位于页面顶端，这是一种传统型的布局。**Logo** 将与博客主页相链接，这样便于用户点击 **Logo** 就返回主页，因为很多用户在访问任何网站时，都已习惯通过点击 **Logo** 返回主页。
2. **导航栏:** 导航栏放置在页面顶部标题部分之下，醒目且便于使用。在页面底端部分也将放置一个同样的导航栏。我这样做是遵循一种传统做法，即为那些不选择在浏览器中显示图像的用户，在页面顶部和底端都放置一个纯文本的导航栏。由于我还不确定是否要在页面顶部的导航栏中使用图像，因此我决定索性在页面其它部分（一般是底端）放置一个纯文本的导航栏。这种纯文本的导航栏，可帮助那些使用屏幕阅读器的盲人用户更容易地访问网页。至于你把纯文本的导航栏放置在页面的顶部还是底端，这并不重要，因为盲人用户和视力正常的用户一样，都可以既自上而下，也可以自下而上地快速浏览网页。是否在一个页面内的两个不同的位置都摆放导航栏，这取决于设计师和其客户的决定。如果你使用包含图像的导航栏且不再另外放置一个纯文本的导航栏，那么一定记住要为导航栏中的图像加入描述性的 **alt** 属性。这样，那些使用屏幕阅读器的用户和不选择在浏览器中显示图像的用户就都还是能知道那些图像指的是什么。请阅读本课程第 17 篇文章的相关部分，更多地了解 正确使用 **alt** 属性。
3. **最新发布的博客条目:** 最新的博客条目值得突出显示，将这个部分放在页面上端（“**above the fold**”）的显著位置，对客户和读者来说都是有益的。这样用户一进入网站，首先映入眼帘的就将是这个部分。不过将这个部分放置在这个显著的位置，意味着客户将需要对这个部分持续地进行更新，否则将面临失去回头访问者的风险，这是因为如果一个博客没有新内容在其上发布的话，人们就不大可能定期来访问这个博客。
4. **以前发布的博客条目:** 以前发布的博客条目放置在这个部分，放置 3—5 个条目应该就足以让访问者初步了解他们以下将可能从网站上看到些什么内容。这里加入图像可能会使页面更好看一些，但并非一定要加入图像，这是因为这个部分位于页面下端（**below the fold**），不是很需要吸引访问者的眼球。是否使用图像，主要可考虑两点：下载图像所需的时间会不

会成为一个问题；以及以前发布的文章旁是否真的需要配上图像，以吸引访问者点解阅读整篇文章/博客条目。

5. **右栏：**这是显示按类别列出的博客条目列表、存档文件，以及其他类型的站点内容，以供访问者点击访问的部分。其他页面如“**about the company**”（关于本公司）页面，也一般有这种右栏，列出网站网站索引和联系信息。你决定如何在这种边栏中列出这些条目是很重要的决定，因为博客将建立在你创建的分类、你构建的页面及存档资料之上。当你在扩充网站时，这些列表将变得更长，因此在我们这个示范网站中，访问者在页面上端（“**above the fold**”）右栏中可能看到的只是 **categories**（分类列表）。客户可能会认为“**pages**”（其他网页）条目比 **categories** 条目更重要，在这种情况下，右栏中排列的顺序就可改为“**pages**”在前，**categories** 在后。作为边注，这个条目列表不列入可包括入一个边框或边栏的所有东西。一些客户可能会感觉页面有两个边栏更好，这样博客页面就会变成三栏的，而不是上图所示的两栏页面。
6. **页面底端信息：**页面底端信息是非常重要的，它向访问者提供可一眼就看到的公司及其网站的背景信息。适于在页面底端列出的信息可包括：公司名称、公司 **Logo**、公司地址、电子邮件地址、链接（指向联系表单、隐私权声明、免责声明、法律信息等的链接）、新闻摘要等。正如前面已经提到的，你也可以在页面底端放置纯文本的导航栏。
7. **广告：**在这个页面布局中，广告设计为横条广告，分别放置在“最新发布的博客条目”部分和“以前发布的博客条目”部分之下。这便于客户灵活选择做文字广告或是网幅图像广告。这种类型的页面布局，仅在页面上端（“**above the fold**”）和下端（“**below the fold**”）各放置一个广告。对多数网站来说，页面上有两处广告也已足够了。此外，这种页面布局是把广告放在网站内容部分的下方，把广告降到次要位置。

这种页面布局可以使访问者不滚动屏幕，即可快速地从页面主体区域转到导航栏，此外通过显示指向网站分类列表的链接，也可使访问者看到网站可能包含的其他主题。使用这种页面布局，即使访问者不滚动屏幕到页面下端（“**below the fold**”），也已经能看到显示在页面上端（“**above the fold**”）的主要元素了。

#### 关于网站上的广告

如果网站上的广告是与网站内容相关的，那么在网站上做广告对客户来说是有利可图的，而对网站访问者来说也是一种服务。换言之，如果网站上的内容是关于花卉的，则网站上的广告可能包括景观服务、宴会服务（这与插花相关）等。因此，对这个提供开源代码资料的网站来说，你可能会去寻找与开源代码业务有关的广告客户在上面打广告。访问 **Google Adsense**，可在这一方面给你帮助。在你的网站流量大到足以吸引其他广告客户到网站来做广告之前，此用这种类型

的广告方式是最好的选择。不过在接受广告前，始终要考虑搜索引擎优化（SEO）这个要点，因为一些广告可能对客户网站在搜索引擎页面中的排名位置有负面影响。以下是一些很有用的 SEO 资源：

- [Intelligent site structure for better SEO!](#) (搜索引擎优化与智能站点机构)，作者 Joost de Valk.
- [Semantic HTML and search engine optimization](#) (语义化的HTML和搜索引擎优化)，作者 Joost de Valk.
- [How Affiliate Programs Can Affect Search Rankings](#) (分销联盟计划如何可能影响到搜索排名)，作者 Fredrick Marckini.
- [New Report Explores how PPC Rank Affects Traffic](#) (关于每次点击付费排名如何影响网站流量的研究报告)，作者 Jennifer Laycock.

**附注：**作为一个设计师，除在为自己设计一个网站时外，你一般不会负责网站的广告事务，不过如果你计划在将来与广告或设计机构合作，你可能会希望多了解一些广告业务。作为一个设计师，你在哪些要素造就一个成功的网站这方面的知识越多，就越有可能取得事业的成功。如果可能的话，尽量多学习营销知识（为你自己，也为你的客户）和了解搜索引擎优化策略。

#### 通过校验代码和征得客户的认可，确认页面布局

在使用代码执行这个页面布局前，我想让客户确认这个页面布局（或线框图）。我在说服客户某一个页面布局比其他页面布局要好时，使用了这样一种策略，那就是提醒他们在额外的页面布局上写代码要多产生费用。这会促使客户选定一种页面布局，而代码可以在以后进行调整，以做出一些结构方面的改动。

在选定页面布局后，下一步的工作就是书写代码并校验代码。我使用万维网联盟的 [标记校验服务](#) 和 [CSS 校验服务](#)，来确认用于构建页面布局的 HTML 和 CSS 没有错误。你可以直接从你的计算机将文件上载到万维网联盟的校验服务站点，而并不需要将文件上载到客户的站点上去测试。这种测试让设计师和/或程序员可以发现在前端的任何错误，这样就可以在向页面添加图像图片、广告和其他东西前，更正这些错误。

可访问性也是一个重大的问题，你需要确认网站能为残疾人（如盲人和有运动障碍的人）所访问。这要比较校验你写的 CSS 和 HTML 难一些。尽管有一些现成的检测工具如 [TAWDIS](#) 等可用，但你最好还是让真实的用户来检测你的网站的可访问性，并就网站的可访问性进行定性分析。尽管它可以就一些具体细节正确与否给出一些提示，检测工具是不能够做出网站是否具有可访问性的结论的，而且有时检测工具也会出错。在本课程后续发布的文章中，还将详细讲述可访

问性，请继续学习吧。

你还应该在不同的可用浏览器中测试你的页面布局，这样你就能确保你的网站能为尽可能多的上网用户正常看到。你可以在 Mac, Windows, Linux 和手机平台上做这种测试——这些平台上都安装有不同的浏览器；或是使用仿真程序如 VMWare Fusion 等，在一台计算机上仿真多个系统，不过这很劳神费时。另外一种可选办法则是使用浏览器截屏服务如 BrowserCam 等，这种浏览器截屏服务快捷、方便，涵盖很多种不同的浏览器（包括一些相当老的浏览器）。这些浏览器截屏网站提供 24 小时的免费试用，你可以去看看他们的服务是否适合你的需要。在免费试用期后，付费使用这种服务也是值得的，尤其是如果你要设计很多网站，并要在很多种不同的浏览器上测试网站的话。

最后，你最好与客户沟通，让他们知道页面布局的代码已经写好且已通过校验；此外，你还应该让客户知道为让网站在不同的浏览器上都正常显示，需要对线框图做出多少处改动。只有在代码已写好并通过校验，且征得客户的认可后，你才能开始向页面添加颜色、图片图像，以及其他代码（如广告代码等）。尽管这种工作可能有些乏味，但确认所有前期工作都已通过校验并获得客户的认可，这是必要的。否则，当你在后续阶段发现代码问题和浏览器兼容性问题时，你将需要很大力气去解决这些问题，造成工作强度增加。此外，当客户想审查网站的实际架构时，将你手头编制好的页面布局图或线框图交给他们，会让他们高兴。

一旦你完成了这个前期阶段的工作，你就可以开始进行格式化文本、添加颜色、图形图像等工作。至于具体该如何做，我将在下一篇文章中讲述。

## 总结

由于网站设计涉及多方面的工作，Web 设计师通常需要多方面的知识和技能。你设计的网站是会不断扩展呢，还是始终保持原样不变？在网站不断扩展时，虚拟主机提供商能始终提供高质量的服务吗？客户需要将网站从一个虚拟主机迁移到另一个虚拟主机，以适应网站扩展的需要吗？如果设计师无法一个人完成全部设计工作，他有一个能给予他帮助的团队可用吗？

因此，则配色和添加图形图像前，设计师需要先确定网站的基础结构。网站设计受到委托建站的公司所从事的业务的影响。同时，在规划阶段，就可以将一些以后可能出现的潜在问题加以解决。这种解决潜在问题的能力是职业设计师应具有的能力。

一旦基础已经确定，且网站架构图和线框图也已编制完成，设计师就可以开始实施配色方案，以完成整个网站的设计，最终提交给客户批准。

## 延伸阅读

以下是其他一些提供设计检查清单的网站：

- [Dive In Designs checklist](#)
- [Net Mechanic's Web Usability Checklist](#)
- [Max Design checklist](#)
- [Usability First's Checklist](#)
- [David Skyrme and Associates' Checklist](#)
- [SCORE's Web site design checklist](#)

## 练习题

- 在你开始进行网页设计前，需要已了解到哪些信息？
  - 为什么你必须将你计划在网页上使用的各个要素都列出来？
  - 研究虚拟主机提供商为什么很重要？
  - Web 设计师可能需要做多方面的工作。如果你对 Logo 设计一无所知，而客户又要求你设计一个 Logo，你将怎么处理？
  - 给出为什么需要研究竞争对手网站的两个有力理由。
  - 什么是 CMYK 颜色，以及这些字母都指什么颜色？
  - 给出至少两种将 CMYK 颜色转换为相匹配的 RGB 颜色的方法。
  - 给出为什么设计师在网页的页面布局中至少需要在一处放置一个纯文本的导航栏的理由。
  - 为什么设计师要在整个网站内都保持一致的页面布局？
  - 给出为什么要在设计的早期阶段进行代码校验的一个理由。
- 
- [上一篇：色彩理论](#)
  - [下一篇：配色方案和设计样板](#)
  - [目录](#)

## 作者简介



Linda Goin 拥有美术学士学位（主修视觉沟通，副修商学和营销学）和文学硕士学位（主修美国历史，副修宗教改革史）。她的第二个学位所学的专业似乎与她的第一个学位所学的专业相距甚远，但她已将从事网站设计 25 年来积累起来的专业知识用于考古发掘领域和物质文化的研究领域。

她的作品曾获得过很多奖项：包括获得过 15 次科罗拉多州新闻社大奖的第一名，多个美术和图像设计大奖等。《华尔街日报》、《芝加哥论坛报》、《今日心理学》、《洛杉矶时报》等报刊都曾对她进行过关于内容开发的专访。她著有多部关于网页设计、可访问性等主题的电子书，此外她还为一些搜索引擎优化专业团队代笔写作，并撰写有关个人理财的文章。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

## 10. 配色方案和设计样板

Posted 12/10/2008 - 17:46 by Lewis

作者: Linda · 2008 年 7 月 8 日

- 上一篇: 建立站点的线框图
- 下一篇: 网页排版
- 目录

### 序言

Web 设计师在把网站架构图或线框图提交给客户批准后, 下一步的工作是通过配色和添加图形图像, 确定网站的外观和整体风格。在本篇文章中, 我将示范我如何尽可能简单高效地完成这一阶段的工作。我信奉“简约”的设计理念, 原因有两点: 首先, 生活已经够复杂了, 不该再让它更复杂; 其次, 简约的设计也有助于提高网站的可用性和可访问性。在本篇文章中, 你们将学习以下内容:

- 确定大标题、小标题、正文部分的字体, 以及其他排版要点。
- 为网站规划不同的色彩组合。
- 为客户构建一个设计样板, 以确定配色和图形图像。
- 在网站上线前, 如何测试网站。

本篇文章的内容目录如下:

- 第一步: 考虑排版
  - 关于字体
  - 关于可读性和易读性
- 第二步: 排版
  - 注意对齐方式
- 第三步: 配色
- 第四步: 测试
- 总结
- 练习题

### 第一步: 考虑排版

在下一篇关于网页排版的文章中，你们将学到更多的有关网页排版的技术知识。这里先了解一些关于排版的知识，对你们还是很有帮助的。

**Fonts**（字体），又被称为“**typefaces**”，用于显示文本、数字、文字及其他符号。这些符号、字母和数字依据字体族、风格（如斜体、正常、倾斜等）、变体（如正常或小号大写字母等）、粗细（如加粗等）、字形（高或宽收缩或扩大）、字号（磅数或高或宽的像素）等分类。排版涉及确定文本的外观及摆置，即确定字符呈现出来的外观及它们如何被摆置在页面内（如栏、段落、对齐等等）。控制网页排版最有效的方式是使用层叠样式表（**CSS**）。

完成 Web 设计的第一步工作，就包括确定在网站每个页面都将使用哪些 **fonts**（字体）。已有很多研究结果表明，在一个网站内使用很多种字体，可能会把访问者搞糊涂。另一方面，如果整个网站只使用一种字体，似乎也显得呆板乏味。

我的建议是，为大标题和小标题选择一种字体，而为正文部分选择另一种字体，尤其是如果计划在网站上加入广告的话，更是应当如此。仅选用少数几种字体，既可以使各页面保持一致性，同时也可以使访问者在浏览网站的各个页面时，能一眼就看出标题和正文部分在字体上的区别。广告客户在网站上发布的广告，将增添字体的多样性。你不会事先知道广告客户将可能在网幅图像广告或文字广告中使用什么字体（一种或多种）。就我个人的习惯而言，我通常为正文部分选择 **Verdana** 字体，为标题选择 **Times Roman** 字体或 **Georgia** 字体。**Times Roman** 字体和 **Georgia** 字体都是衬线字体，而 **Verdana** 字体是无衬线字体。正如你们马上就要看到的，我决定为示范网站页面的正文部分选择 **Verdana** 字体，不过由于网站 **Logo** 选用的字体是 **Arial Black**，因此我为所有标题选用的还是无衬线字体。有时候你需要打破自己订下的规则，在这个示范网站中，我为正文和标题都选用的是无衬线字体，就是一个打破自己订下的规则的例子。首先还是让我给你们讲述不同字体之间的差别，以及我为什么只选用两三种字体的原因。

## 关于字体

字体的类型主要有四种，分别为：

1. **衬线字体（Serif）**: 指在字的笔画末端的地方有额外的装饰，或是有真实的衬线（包括粗截线）的字体。由于衬线字体易于在印刷品上阅读，因此在传统印刷中，衬线字体一直用于正文印刷。但是网页不同于印刷品，一些关于 字体可读性的 研究显示，一些无衬线字体（如图 2 所示的**Verdana**字体）用于网页的正文中时更易于阅读。

## Times New Roman, regular, 18 point

图 1: 衬线字体的一个示例，**Times New Roman** 字体，正常（非粗体或斜体），大小为 18 磅

2. 无衬线字体 (**sans-serif**): 指在字的笔画末端的地方没有任何额外的装饰或衬线的字体。尽管一些作者撰文指出关于字体可读性的研究是有错误的，但我注意到发表这些文章的网站，其正文部分就是使用的无衬线字体。就是那些发表主张衬线字体更易于阅读的网站，其正文部分依然使用的是无衬线字体。由于在网页的正文部分使用无衬线字体已成为惯例，我决定从众，使用如图 2 所示的无衬线字体。

**Verdana, regular, 18 point**

图 2: 无衬线字体的一个示例，Verdana 字体，正常，大小为 18 磅

3. 手写或草写字体 (**Script or Cursive**): 手写或草写字体通常看起来像是用手写笔或毛笔写成的，而不像是打印出来的，如图 3 所示。这类字体可包括那些即使不是草体，但是看起来像是手写体的字体。应避免在网页中，尤其是正文部分使用手写或草写字体，主要是因为它们不易阅读（回想一下你婶婶给你写的手写体信件吧，或是你在博物馆曾看过的 12 世纪的手抄本，想想它们有多难阅读）。此外，不是所有的浏览器都显示一样的字体，因此如果你在网页中使用手写或草写字体，在其他人的浏览器中可能被显示为一种衬线字体。

**Staccato222BT, regular, 18 point**

图 3: 草写字体的一个示例，Staccato 字体，大小为 18 磅

4. 特殊字体，包括等宽字体 (**Specialty, including monospace**): 确定等宽字体的唯一标准是所有字符的宽度都一样，为固定宽度，和打印页面看起来类似。其他一些特殊字体外观可能很有吸引力，如以下图 4 所示的那种字体，不过它们仅用于装饰目的。网站上有时会使用等宽字体，尤其是在显示程序代码时。由于等宽字体可以清楚地显示代码中所使用的每个字母和符号，因此常用于显示程序代码。

**JOKEWOOD, REGULAR, 18 POINT**

图 4: 特殊手写字体的一个示例，Jokewood 字体，大小为 18 磅

仔细观察以上图 1 至图 4 所显示的 4 种字体的示例，就会发现即使每种字体的磅数都一样，也不是一样大。字体的磅数指的是字母的高度，在磅数都为 18 的字体中，一些字体会比另一些更大。此外不同的字体之间还存在其他差异，如字母和词之间的间距等；还有一些字体，如 Jokewood 字体等，没有小写字体。不过尽管一些手写字体如 Jokewood 字体、 Staccato 字体，因易读性不好一般不在正文部分中使用，它们还是可以在标题、广告等中小范围地使用。

这里有一点值得引起注意，那就是由于不同的浏览器基本上还是不兼容，因此这些字体可能不是在所有浏览器中都显示同样的样子。你们可能会想起我在上面说过的话“不是所有的浏览

器都显示一样的字体”。造成出现这个问题的原因是不是所有的操作系统都支持一样的字体；或者它们可能支持一样的字体，但字体的变体、粗细及其他要素在不同的浏览器中显示时可能有差异。由于这个原因，你可以选择一种 **generic font**（通用字体），或就选择一种“衬线字体”或“无衬线字体”来显示你的网页的文字排版。或者你也可以同时既使用所选定的字体的通用名称和专有名称，以期达到最佳效果，毕竟有时候用户也能更改你选定的字体或其显示方式。

你要使用能在所有浏览器中都显示为一样的特别字体（具有特别的风格、变体、粗细或字形），唯一的办法是使用一个图形程序创作那种字体的图形。但出于很多原因，并不建议你这样做。这些原因包括：屏幕阅读器不能阅读隐藏在图形中的文字；难于缩放（不是所有的浏览器都可以全页面缩放）；难于维护（你每次想更改文字的时候，都不得不重新创作图形字体）。相信我，不要这样做。

也正是出于这个原因，作为一个 Web 设计师，你必须学会去接受 Web 基本上是可变性很高的格式这一事实。在设计网站时，很多事情并不是你能完全控制的。你能在一定程度上控制文字排版，但其前提条件是你要采取尽可能简约的方式排版。这也就是这么多年来，我一直在网页正文内容中使用 **Verdana** 字体，在标题中使用 **Times Roman** 字体或 **Georgia** 字体的原因所在。

字体设计师和程序员一直在努力寻找让字体更易读且更美观的方式。因此，你们不必把我说的话当作金科玉律，还是可以尝试一些运用字体的新方式，只要你认为它们有好的效果的话。你只需在多种浏览器中测试你的网页（这一点我将在以下部分讲述），就可以很快发现所运用的字体方案是否成功。

### 关于可读性和易读性

当你将你的网页设计提交给客户看时，客户通常并不知道衬线字体和无衬线字体之间的差别。客户所知道的只是他是否能顺畅地阅读网页上的内容。因此，说到底易读性才是关键。为此，你需要确保：

1. **你使用的字体足够大，能在多种不同的浏览器分辨率下能为用户顺畅阅读。** 尽管用户在一些浏览器中（如 **Opera** 浏览器）可以改变字体大小，你还是应确保所使用的字体大小能在不同的浏览器分辨率下可调整。要达到这一效果，一个可用的方法是使用 **CSS**，以百分比或 **ems** 设定字体大小，而不是用像素高度来设定字体大小。
2. **你为背景和正文部分提供了足够的对比。** 在背景色为黑色的正文部分中使用白色或亮色字体，可能会让一些用户的眼睛感到疲劳，不过如果你需要使用这种版面效果，可以使用另外一种样式表，将正文部分的背景色设为亮色，并将正文部分的字体设为暗色。

3. **标题部分与正文部分呈现出差别。**为大块的正文部分加上标题和小标题，或者编号（如我在这部分这样），便于用户快速浏览整个页面，发现对他们来说是重要的内容。用图像将页面分隔为几部分也是可行的做法，不过你要确定这些图像与页面内的内容是相关的。否则，你就只是在浪费带宽。
4. **避免在使用流动布局时让正文的宽度延伸至整个屏幕的宽度。**你可以试试在一个大屏幕显示器上阅读一个宽度延伸至整个屏幕宽度的段落，你很快就会感到疲劳，因为你的眼睛必须不停地从屏幕一端移动到另一端。你可以去看看这个 [关于易读性的网页](#)，它用例图阐述了易读的正文宽度最好是多宽，这是迄今为止我看到过的关于这个问题最好的阐述（例图见以下图 5）。这个网页还详细说明了人们是如何阅读一个页面的，不论是 Web 页面还是印刷页面。以下那个示例图是从一个 24 英寸的屏幕（分辨率为 1920 x 1200）捕捉的页面截图。
- 将该示例图与你点击上面那个链接时在你的屏幕上显示的图像加以对比。然后查看你的显示器的分辨率，观察这两个图像的差别。有时使用固定宽度布局是很好的选择，因为它定义了正文部分的宽度参数，便于用户阅读。不必担心正文周围的那些空白（如下图所示）。只要设置好适当的背景（不分散用户对设计或正文的注意力），那些使用大屏幕显示器的用户一样会获得满意的用户体验。

一般认为，正文每行包含 40–60 个字是比较合适的，会因为一些因素如字体大小、目标受众等而有所变化。



图 5：适宜的行宽的一个示例，在一个大屏幕显示器上的显示效果。

最好，一定记住检查正本部分和标题是否存在文字排版错误和拼写错误。我在本篇文章中提供了一些网页的链接，我在其中约一半的网页中都至少发现了一处文字排版错误和几处拼写错误。尽管人都会犯错，但像拼写错误、排版错误（如文字间距不够等）这样一些低级错误，是应该避免的，它们会损害到公司或个人的可信度。

## 第二步：排版

在你选定了将在网站中使用的字体后，你需要在页面布局中对大标题、小标题、正文等进行排版。在上一篇文章中，我引入了一家虚构的“Wiki Whatevers”公司，该公司希望在其网站上与用户共享作为开源代码的代码段。我设计好了网站架构供该公司批准，他们很认可我提交的网站架构图，这让我很高兴。我未在线框图内放入任何图形或图像（包括公司 Logo），以避免让客户对后续设计产生先入为主的概念。这个线框图如以下图 6 所示：

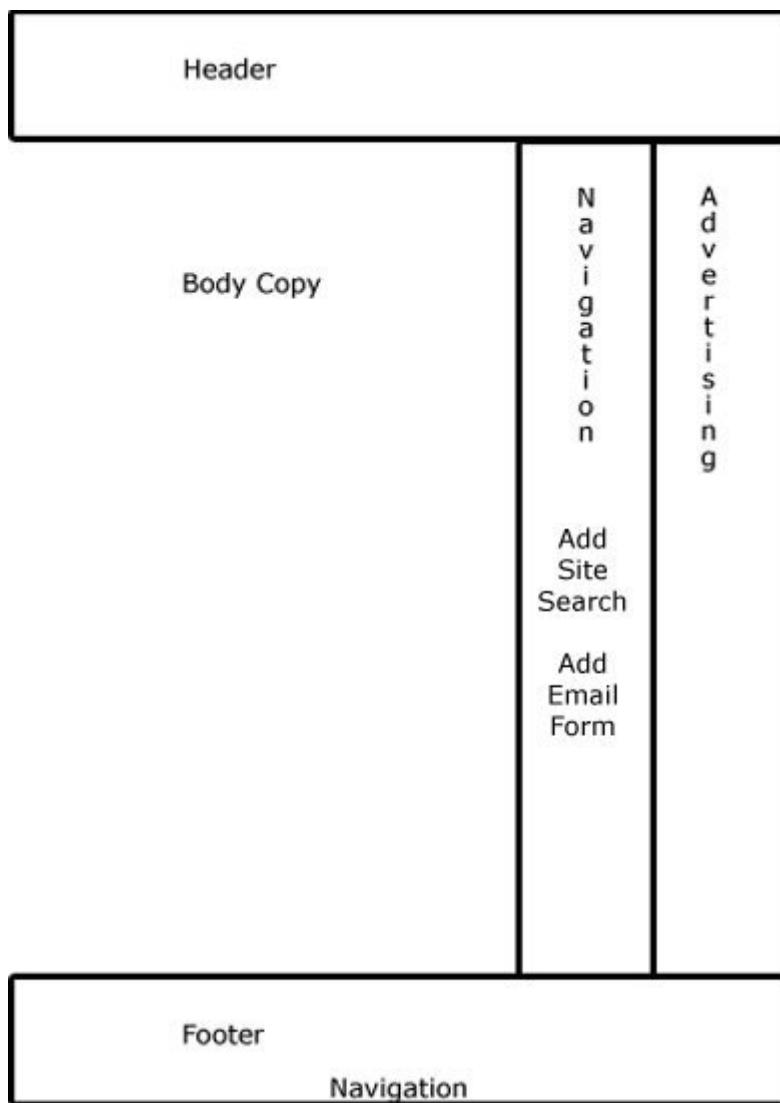


图 6：Wiki Whatevers 公司网站的线框图

现在我想在这个线框图中加入公司 Logo 和公司提供的一些正文部分的内容，以确定它们在页面布局中的显示效果。我在此时加入公司 Logo 和一些正文部分的内容，还有另外一个原因，那就是排版可以为我将要进行的配色工作提供帮助。网页上的正文部分、标题和其他一些版面设置，本身即可以为网站加入“颜色”。你可以将图 7 和图 6 相对照，看看二者之间的差别：

**WW Wiki Whatevers**  
Open Source Wikis

---

Home Categories Pages Archives

## The latest Wiki

Lore ipsum dolor sit amet, consectetur adipiscing elit. Quisque enim tellus, dictum vitae, accumsan sed, aliquam id, ipsum. Donec egestas erat et tortor. Vestibulum sit amet massa ut metus mattis elementum. Vestibulum scelerisque neque id tellus. Morbi posuere malesuada justo. Etiam ultrices convallis lorem. Nullam condimentum dignissim lorem. Cras adipiscing tellus ut ante.

Cras at tellus. Praesent eget arcu. Sed ut magna eu nunc sodales molestie. Maecenas odio. Lore ipsum dolor sit amet, consectetur adipiscing elit. Quisque enim tellus, dictum vitae, accumsan sed, aliquam id, ipsum. Donec egestas erat et tortor. Vestibulum sit amet massa ut metus mattis elementum. Vestibulum scelerisque neque id tellus. Morbi posuere malesuada justo. Etiam ultrices convallis lorem. Nullam condimentum dignissim lorem. Cras adipiscing tellus.

Lore ipsum dolor sit amet, consectetur adipiscing elit. Quisque enim tellus, dictum vitae, accumsan sed, aliquam id, ipsum. Donec egestas erat et tortor. Vestibulum sit amet massa ut metus mattis elementum. Vestibulum scelerisque neque id tellus. Morbi posuere malesuada justo. Etiam ultrices convallis lorem.

|                                                                              |                                                                              |                                                                              |
|------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesq. | Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesq. | Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesq. |
|------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------|

**Previous Blog Entry**

Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesque venenatis sapien. Morbi eu massa sed dolor pulvinar var. [>>>](#)

**Previous Blog Entry**

Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesque venenatis sapien. Morbi eu massa sed dolor pulvinar var. [>>>](#)

**Previous Blog Entry**

Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesque venenatis sapien. Morbi eu massa sed dolor pulvinar var. [>>>](#)

|                                                                              |                                                                              |                                                                              |
|------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesq. | Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesq. | Lore ipsum dolor sit amet, consectetur adipiscing elit. Maecenas pellentesq. |
|------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------|

**Wiki Whatevers**

141 Open Source Drive, Sarasota, Florida

**Sign Up!**  
Receive our news  
as it happens!

**Categories**

- \* [Lorem ipsum](#)
- \* [Curabitur](#)
- \* [Nullam](#)
- \* [Morbi](#)
- \* [Morb](#)
- \* [Sed](#)
- \* [Aenean](#)
- \* [Nam](#)

**Pages**

- \* [Lorem ipsum](#)
- \* [Curabitur](#)
- \* [Nullam](#)
- \* [Morbi](#)
- \* [Morb](#)
- \* [Sed](#)
- \* [Aenean](#)
- \* [Nam](#)

**Archives**

- \* [Lorem ipsum](#)
- \* [Curabitur](#)
- \* [Nullam](#)
- \* [Morbi](#)
- \* [Morb](#)
- \* [Sed](#)
- \* [Aenean](#)
- \* [Nam](#)

图7：加入了Logo和正文部分的网站线框图。

以上所示的样本稿为网站加入了“颜色”，虽然唯一加入的颜色只是Logo的颜色。样本稿不仅仅为网页加入了内容，而且也加入了黑色、白色和灰色的亮色、灰色调和暗色。我仅仅加入了网站不可或缺的那些要素，包括Logo、网站名称（也是公司名称）、宣传语（即Open Source Wikis，开源Wikis）、链接、订阅新闻组的注册栏或中栏中的Feed订阅、页面底端关于公司的

信息、纯文本导航栏、广告样稿。由于搜索框较大，无法仅放置在一栏，因此我把它放置在了页面顶部。

虽然我说我使用了正文稿，实际上我使用的是 [Lorem Ipsum](#) 生成器网站提供的模拟正文稿。当你在设计网站时，如果尚未能得到真实的正文稿，可以使用这种模拟正文稿作为代替。

### 注意对齐方式

在这部分，我将示范我是如何决定网页上各个页面元素的对齐方式的。对齐指页面元素如何摆置在页面的特定区域。你可以选择左对齐方式，把不整齐的地方都放在右边缘，这是一种传统的页面布局。你还可以选择居中对齐方式、两端对齐方式（页面左右两边都是整齐划一的），或右对齐方式，把不整齐的地方都放在左边缘。

我选择使用传统的左对齐方式，即所有的页面元素都整齐一致并且靠左对齐。不过你们可能已经注意到了，相对于其小标题，正文部分向右缩进了一些。我之所以要选择这种对齐方式，是因为 Logo 的原因。请看图 8，这个图显示了我为何要采用这种对齐方式：



图 8: Logo 和正文部分之间的对齐方式示意图

由于这个 Logo 并不是那种获得过设计大奖的 Logo，因此我希望它显示得不要那么大。此外，这个 Logo 使用的 Arial Black 字体使得它显得较大，与页面其他元素不太成比例。尽管我已将 Logo 的尺寸大大压缩，但我还是希望它的大小要与旁边在 Logo 高度内显示的公司名称和标签行（“Open Source Wikis”）相称。你可以从上图中的红色箭头中看到，公司名称是与倾斜的 Logo 的顶端对齐的，而标签行是与 Logo 中黑色“W”的底端对齐的。（请同时回头看看图 7，从 Logo 中第一个“W”往下看，你会发现“The latest Wiki”这个标题是与第一个“W”的左端点对齐的。由于 Logo 是倾斜的，其最低点可指向位于导航栏之下的标题。实际上由于标题的字体

大小与公司名称的字体大小一样，导航栏看起来似乎处于次要位置。)

这时我发现我为博客条目标题选用的 **Georgia** 字体在这里似乎有点不协调，因此我决定将标题的字体改为 **Arial Black**，这是一种无衬线字体，与我在正文部分中使用的 **Verdana** 字体有些不同，但差别也不是很大，不会造成混乱。

使用 **CSS** 是设置页面内各元素对齐方式简便有效的方法。尽管一些浏览器不一定能完全按你设置的对齐方式显示网页，但在很多情况下使用页面布局中精确的点来对齐页面元素是可行的。例如你可以将搜索框与标签行的底端对齐，或是将该搜索框的右边与位于其下的表单的右边对齐，如同我在以上的示例页面所设置的对齐方式一样。

我本可以将正文部分与 **Logo** 的左上角及标题对齐的，不过我选择了将正文部分向右缩进了一些，这样可以让用户能快速浏览标题，选择阅读他们认为是重要的文稿。这也就是说，网页是各不相同的，如果选用另外一种 **Logo** 设计，可能就需要选用完全不同的对齐方式。关键之处在于要让页面上的重要元素对齐，整齐有序地排列和显示。访问网页的人不会注意到你曾为对齐而煞费苦心（客户一般也不会），但是如果你未花费时间将页面上的重要元素对齐，有人就可能会评论说“这个页面总有什么地方不对劲”。

将正文部分向里少量缩进，还产生出所谓“留白”，在正文或其他任何页面元素周围形成一处空白，这样就能让访问者容易地分辨出每一篇正文文稿。正文部分左边的留白通常应设置为与正文部分（或图像）右边的留白一样，这样才能创造出一种平衡感。设置留白，可以让页面不致显得太拥挤，而是疏落有致。

下一次当你在网上冲浪时，请仔细看看你访问的网页，注意哪些是你觉得设计得很好或感觉还可以的网页，观察这些网页中正文部分、标题、**Logo** 及其他页面元素的对齐方式。它们都对齐了吗？还是显得参差不齐？设计师要花时间去注意这些微小的细节，这也许使得网页设计显得没有内容开发那么重要，但是好的设计确实又必须注意这些微小的细节。

此外，网页在不同的浏览器中显示时，也可能会遇到对齐方面的问题。文本可能在 **Safari** 浏览器中显示为完全对齐，而在 **Opera** 浏览器或 **Firefox** 浏览器中又显示为已完全对齐。我将在讲述了为网页配色后，再告诉大家如何解决这个问题。

对我为什么没有为小标题和公司名称也选择 **Arial Black** 字体，你们当中有些人可能会感到不解。我为小标题和公司名称选择的是 **Times New Roman** 字体，因为这种衬线字体和其他部分选用的无衬线字体形成了对比，为页面添加了趣味，这是一条设计原则。如果页面内所有标题都使用 **Arial Black** 字体，会让页面显得呆板乏味。

### 第三步：配色

在我为客户准备网站样板时，我通常都是在线框图已编制完成后才向客户提交网站样板。如果可能的话，我会选择使用代码而不是图像来编制线框图。这样我就可以把页面元素如 Logo、正文部分，甚至广告样本都包括进线框图中，向客户展示在最终的页面中各个元素将是如何显示的。有了这个完整的页面布局图，客户就会很清楚地知道当所有这些页面元素放置到位后页面会是什么样子，然后就可以决定哪些元素可以增加或移除。此外，当我能在计算机上向客户展示和以后在网上显示时一样的网页之时，客户就能直观地看到网页在以后上线后将呈现出来的样子。

在向客户提交的设计样板中，应添加入颜色。之所以要这样做是因为不同的配色方案可以完全改变一个网站的整体观感，即使所有页面元素都已放置到位。此外，由于太多的颜色样本可能会让客户无所适从，我选择向客户提交尽可能少的颜色样本。在本示例中，由于客户的建站预算很有限，我劝说他们只选择一种配色方案，作为向网页添加颜色的样本。

在前面第 8 篇文章中介绍配色方案生成器<sup>II</sup>时，我没有提到你可以将 16 进制的颜色代码输入这个工具，从一种特定的颜色生成一个配色方案。在色轮的正下方，你们可以看到一个“Enter RGB”（输入 RGB 颜色）的链接。在本示例中，由于 Logo 中的金色是最强烈的颜色，因此我输入金色的 16 进制颜色代码 #eab304，以看看生成的配色方案会是什么样子。生成的单色方案看起来有些乏味，而对比色方案看起来就很不错。这个互补色方案中包含金色的对比色蓝紫色，可供我选用，实际上 Logo 背后的阴影也是浅蓝色的，如图 9 所示：

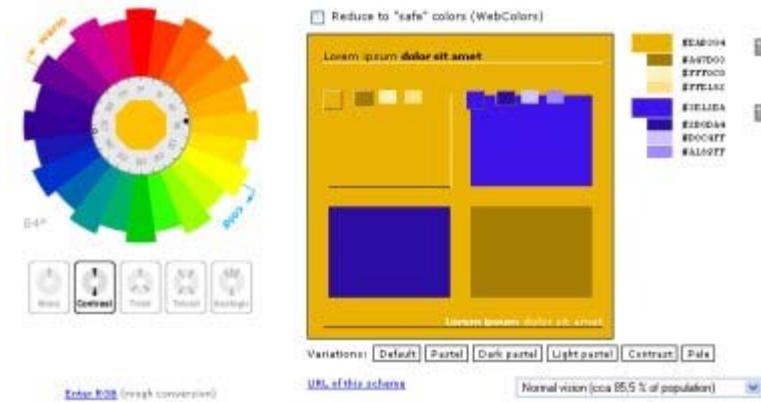


图 9：基于 16 进制颜色代码 #eab304 的对比色方案

根据上图显示的颜色，我决定将 Logo 中的主色金色用作顶部导航栏的背景色。我为有下划线的链接选用颜色代码为 #2b0da4 的暗蓝色（几乎就是蓝紫色），并选用深蓝色作为广告背景色。在下面的图 10 中，你们可以看到加入颜色后页面外观发生的变化：

The screenshot shows a website with a dark blue header and footer. The header features a yellow 'WW' logo and the text 'Wiki Whatevers' and 'Open Source Wikis'. A search bar and a 'GO!' button are on the right. Below the header is a yellow navigation bar with links: Home, Categories, Pages, Archives. The main content area has a white background with black text. It includes a section titled 'The Latest Wiki' with three paragraphs of placeholder text (Lorem ipsum). Below this are three columns of placeholder text. To the right is a sidebar with a 'Sign Up!' form, a 'Categories' list (with links to 'Lorem ipsum', 'Curabitur', etc.), a 'Pages' list (with links to 'Lorem ipsum', 'Curabitur', etc.), and an 'Archives' list (with links to 'Lorem ipsum', 'Curabitur', etc.).

图 10：应用对比色配色方案后的页面外观

在上图中你们可以看到现在选用的颜色还是太深太暗了。因此，我将导航栏中颜色的暗度调低至 75%，将广告栏中颜色的暗度调低至 20%。在以下的图 11 中，你们可以看到页面中的颜色立即就发生了变化：

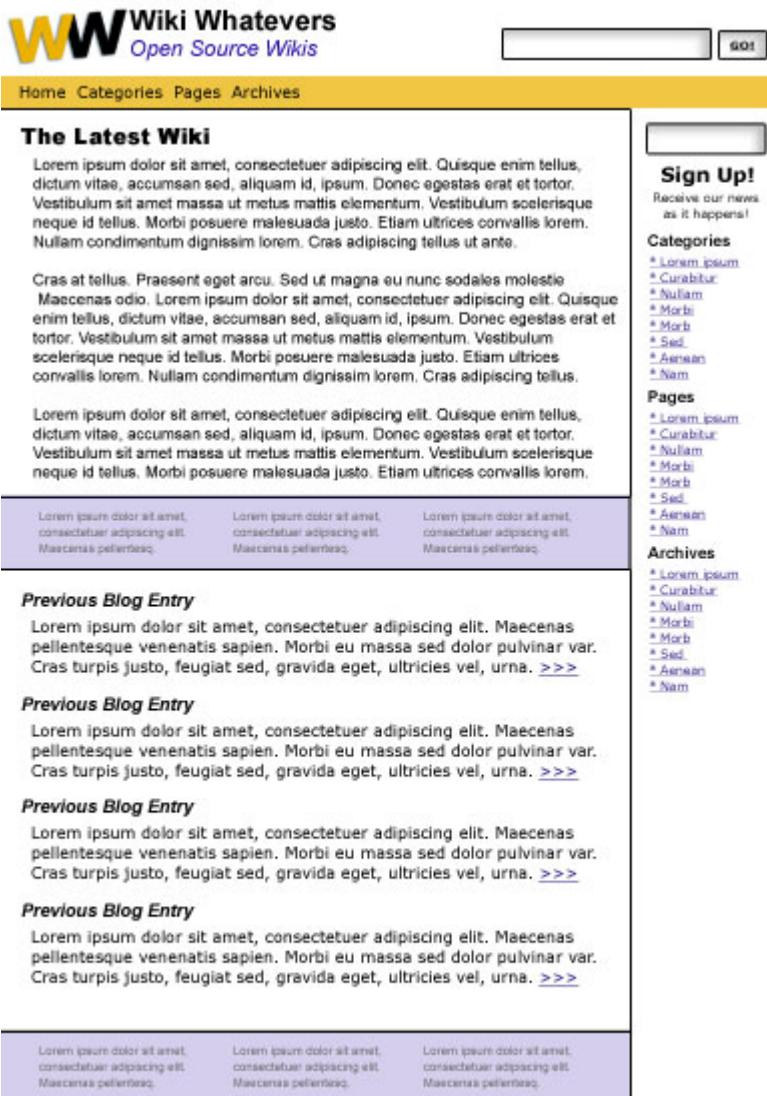


图 11：颜色暗度调低后的页面外观

将导航栏中金色的暗度调低，使它与 Logo 中的金色显得更一致。将广告栏中背景色的暗度调低，使它与链接的颜色显得更一致。由于广告中包含链接，这种让广告栏背景色与链接的颜色更匹配的做法是适当的。加入广告栏的背景色是很好的做法。如果你使用像 Google Adsense 这样的广告服务，你就会知道 Google 一般会要求你让广告在页面中更显眼，而在广告栏中加入背景色就有助于到达这个要求。我还为标签行选用了颜色代码为 #2B0DA4 的深对比色，这样可以让蓝色这种对比色在页面顶部和底部都有显示。

尽管看起来这个页面的设计和配色并不难，但我还是很花费了一些时间，多次配置背景色（使用对比色方案中的颜色）、标题颜色，并改变广告栏的设计。每次改动时，颜色看起来都似乎不够简约，因此我最后决定减少颜色的使用，为标签行以外的文字都选用黑色。尽管我可以设

置让已访问过的链接改变颜色，但我认为最佳的选择还是仅使用两种颜色，这样适宜插入彩色图片，不致造成色彩的混乱。

另一方面，你也可以看到预先建立好线框图能怎样简化你的设计工作。一旦你手上有了网站线框图或架构图，向页面中添加颜色的工作基本上就像是在确定的界限内添加颜色。如果你坚持使用你的页面布局，那你就可以使用该页面布局来决定颜色选择。此外，如果你采用简约的设计风格，设计出来的网站不仅更为简洁典雅，还具有更好的可用性和可访问性。

让这个页面布局保持简约的最后一个有力的理由是，在下级页面中将显示代码段，我将遵照最佳习惯为这些代码段选用等宽字体。这也是我为什么在这个页面中只选用两种风格相似的无衬线字体的原因，在下级页面中将出现的等宽字体，以及广告中可能出现的任何其他字体，已经使该网站有了足够丰富的字体。请记住一定为大标题和小标题使用标题元素（`h1`, `h2`, `h3`, 等），而不是为它们加粗（`strong`）或使用斜体字（`em`）。使用标题元素可以让你的网站具有更好的可访问性。你可以在样式表（CSS）中改变这些标题将如何显示。

关于以上这个页面，有几点需要注意：

- 我为放置在页面顶部的公司名称选用的是黑色，因为这与旁边的 `Logo` 中的黑色相衬托和呼应。
- 在中栏的顶部我使用了居中对齐方式，以突出显示注册框。由于用于订阅新闻组的注册框的宽度与中栏的宽度一致，采用居中对齐方式就创造出一种平衡感，使文字看起来“属于”注册框所创造出来的形状的一部分。
- 我为在页面底端显示的公司地址使用了居中对齐方式，这可能看起来有点不平衡，但我希望地址成为衬托正文部分的区域的一部分。随着网站内容的增多，右栏中会有更多的链接，可能还会有一些图像，我希望这个右栏区域与主正文部分完全分离。这种分离会提示访问者右栏是他们需要去找寻更多的信息的区域。

最后，作为点睛之笔，我将为页面加入图像。除非客户已有要使用的图像，你自己选择的图像必须与页面的配色方案相协调。换句话说，你要寻找能反映你为页面选定的颜色的适当图像。在本示例中，我选择为页面加入一张“极客”（geek）的图片，为页面增添一点幽默元素。我之所以选择这张图片，原因之一是图片中那个男子直视访问网站的人，这意味着访问者一进入这个页面，就会被这张图片吸引住。由于这张图片如此重要，我很高兴地发现图片上这个男子所穿的衬衫的颜色看起来与页面的两种主色（金色和蓝色）接近，而且他戴的黑色眼镜也呼应了网站标题的深黑色色调。我还在 `Photoshop` 中对这张图片进行了处理，在其阴影部分加入了蓝色，并在其强光部分加入了黄色，以使该图片与页面整个的配色方案更为协调。

我还加入了包含博客已贴标签的标签行和日期/时间标记，这样访问者就可以知道博客条目是什么时候发布的。如你们在下图中看到的那样，所有这些加入的元素都使得页面变得更为复杂，这也是你为什么要使重要和关键的信息以尽可能简约的方式呈现出来的另一个原因。页面的最终显示效果，见图 12：

图 12：最后的设计稿，供客户检查和批准

Web 设计不同于印刷设计，它是可以不停改动的，而印刷设计则不行。因此，随着公司的发展，网站是可以多次改版的。你可以修正错误，修改配色方案等。尽管如此，最好的选择还是在一开始就争取让网站以最佳的形象呈现在网上，这样既有利于维护你作为设计师的声誉，同时也利于维护客户的声誉。

在我构建起更多的网页后，我会执行最后的步骤即测试，然后再将定稿交给客户批准。

## 第四步：测试

测试网页意味著设计师将使用各种工具细心地检查所有网页，找出可能有的各种错误，以在网站上线前就修正这些错误。你需要进行多种测试，包括：

1. **测试排版和链接：**你可以请你的朋友、职业编辑检查网页中是否存在排版和拼写错误，同时也请他们检查网页中的链接，确保不存在无效链接。这里要提醒一点，大多数客户是不愿看到网站在上线前，就被外人看到的。在这种情况下，你需要预先就在建站预算中包括进聘请职业编辑来校对网页的费用，并要求所聘请的职业编辑签署保密协议（NDA），从而有责任对网站内容严加保密。
2. **测试代码的有效性：**在你每一次向网页中加入新代码时，都需使用万维网联盟（W3C）的“校验器”检查 HTML 和 CSS 代码。在执行下一个步骤前，这是必须先执行的步骤，这是因为一个代码错误就可能使整个网页不能在各种浏览器中正常显示。至于广告代码，你不要试图去改动广告代码，尽管它可能不能通过校验。多数设计师都已知道他们必须接受广告代码就是那个样子的事实，基本上没有什么办法去改动它。令人感到欣慰的是，绝大多数广告代码是不会影响到你下一步的页面布局的。
3. **测试浏览器兼容性：**你也许会问，要测试网站的浏览器兼容性，是不是就需要购置多台有不同尺寸显示器的计算机，并在各台计算机上安装不同的操作系统呢？答案是你不需要。没有几个设计师负担得起如此高昂的费用，因此他们使用 几种选项 来测试网站的浏览器兼容性。这些选项包括：把一个操作系统的几种版本下载到一台计算机上，寻求朋友或论坛的帮助，以及使用截屏网站提供的服务。这些截屏网站可提供某一时间一个网页在多台不同的计算机上显示画面的截屏。使用这类截屏服务，你就可以发现你在网页上使用的字体是否在某一分辨率设置下显得过大，而在另一种分辨率设置下又显得过小；或者发现所使用的页面布局是否会迫使一个用户要水平滚动屏幕才能看全，或者是否在某一种浏览器中页面的某一栏完全看不到。我通常使用截屏服务来测试网站的浏览器兼容性，因为前两个测试方法的效果都不是那么好。此外，使用截屏服务进行测试，还可以使客户的网站内容不外泄。互联网上有多个截屏网站，不过一些截屏网站响应时间太慢（排队等待测试的用户太多），而且可提供的测试环境的数量也相当有限。因此，我花了一小笔钱，使用 BrowserCam 提供的截屏服务，这笔钱还是很值得的。这个网站还提供免费的试用服务，因此你可以先免费试用一下他们提供的截屏服务，了解其功能和特性。
4. **测试可用性和可访问性：**要测试网站的可访问性，有多种 在线工具 可用。其中一些需要使用声音，这样你就能听到一个语音机是如何“朗读”你的网页的。另外一些测试结果可能要求你更改一处代码，或更改页面上的颜色，为低视力的用户提供更强的对比度等。至于要

测试可用性，也有一些在线工具和检查清单可用，它们可以帮助你确认你设计的网站易于为最大多数的用户使用。

这些测试是烦人的工作。在测试过程中，你可能会发现你设计得很好的网页在一些浏览器中显示出来的样子也确实很好看，但在另一些浏览器中却可能显得一团糟。不过你们始终要记住，最重要的还是内容，只要内容在所有浏览器中都能被看到且易读，那么一些设计细节在某一浏览器中显示出来的效果不好就不是那么重要。其实只要绝大多数网站的用户能无障碍地访问你设计的网站，获取所需要的信息和材料，那你就已经实现了一个被很多只在意网页设计是否能获得设计大奖的设计师所忽视的一个重要目标。

## 总结

尽管设计师都很重视配色和页面布局，但同样也需要处理好其他一些设计元素。处理好排版、图像，以及客户的广告需要，都是网站设计中重要的部分。要求设计师既满足客户的要求，又让网站保持好的可用性、可访问性和易读性，同时还要创作出非常棒的设计，有时候这会显得像是过高的要求。

更让人感到沮丧的是，不同的浏览器之间还缺乏兼容性。尽管过去十年间在让不同的浏览器更兼容方面取得了一定的进展，但你必须明白一点，离网页在所有浏览器中都一样显示的理想状态还很遥远。此外，你还应当认识到，用户可以在一些浏览器中很容易地改变网站的外观，比如可以不显示网页中的图像，可以更改文本和背景的颜色，可以关闭 **Javascript** 支持功能等。

另一方面，让不同的浏览器更兼容的努力，以及 **Web** 功能性的不断改进，对当今的 **Web** 设计师也构成了激动人心的挑战。想想吧，家庭计算机大规模进入零售市场也才不到 30 年时间，在未来 10 年，对一个希望始终更上所有变化的设计师来说，会面临多少机遇和挑战。

## 练习题

- 四种主要字体是什么？
- 哪种字体最适合在正文部分使用？为什么？
- 要让正文部分和其背景之间保持足够的对比度，这一点为什么重要？
- 给出至少两种将布满正文内容的网页分割开来的方式。
- 给出为什么在向页面加入图像前，先要排版的一个理由。
- 说出四种类型的对齐方式。
- 说明对齐如何能让网页显得“更清爽”。
- 什么是保密协议，以及你该在什么情况下使用这种法律文件？
- 检查网页上的拼写为什么很重要？

- 给出在网站上线前对网站进行测试的**4**种方式。

- 上一篇：建立站点的线框图

- 下一篇：网页排版

- 目录

## 作者简介



Linda Goin 拥有美术学士学位（主修视觉沟通，副修商学和营销学）和文学硕士学位（主修美国历史，副修宗教改革史）。她的第二个学位所学的专业似乎与她的第一个学位所学的专业相距甚远，但她已将从事网站设计 25 年来积累起来的专业知识用于考古发掘领域和物质文化的研究领域。

她的作品曾获得过很多奖项：包括获得过 15 次科罗拉多州新闻社大奖的第一名，多个美术和图像设计大奖等。《华尔街日报》、《芝加哥论坛报》、《今日心理学》、《洛杉矶时报》等报刊都曾对她进行过关于内容开发的专访。她著有多部关于网页设计、可访问性等主题的电子书，此外她还为一些搜索引擎优化专业团队代笔写作，并撰写有关个人理财的文章。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

# 11. 网页排版

Posted 12/10/2008 - 17:47 by Lewis

作者: paul-haine · 2008 年 7 月 8 日

- 上一篇: 配色方案和设计样板
- 下一篇: HTML 基础知识
- 目录

## 序言

什么是排版? 简单地说, 排版指文本的样式、设计和摆置, 这是一个从传统的印刷设计中借用来的概念。知道你**能**通过排版做什么与知道你**不能**通过排版做什么一样重要。在万维网上, 排版通常很少引起人们的注意, 同时也存在一些技术限制, 可能使得网页排版比印刷排版更为困难。虽然如此, 有了一些可用的工具, 你还是能排出各种样式美观、独具风格的网页版面的。

在本篇文章中, 我将讲述为什么相对于印刷排版, 网页排版是受到限制的, 并告诉大家一些网页排版的诀窍, 同时以一个示范网页演示如何运用其中一些诀窍。如果你还不理解 CSS 和 HTML 代码, 也不必担心, 现在你仅需要考虑设计。在你阅读本篇文章时, 最好在身旁放置一支笔(钢笔或铅笔都可以)和一张纸, 自己动手勾画文本布局。本篇文章的内容目录如下:

- 网页排版受到的限制
  - 可供选择的字体较少
  - 连字符连接
  - 字距调整
  - 缺乏控制
- 如何进行网页排版?
- 网页排版的一些诀窍
  - 选择字体范围
  - 行长
  - 行高
  - 首字母下沉
  - 小型大写字母
  - 悬挂标点
  - 排版正确的标点和其它HTML实体

- 重要引述
- 总结
- 练习题

### 网页排版受到的限制

印刷设计师在排版时，可供其使用的排版选项是非常多的，包括有很多字体可供选择，可供选择的定位文本的方式也很多。相比之下，网页排版就要受到很多限制了，这是因为我们在使用字体和文本定位等进行网页的排版设计时，必须考虑到排版设计要能在访问网站的用户的计算机上正常显示，这绝不像仅为自己设计那样简单。

网页排版受到的限制包括：

- 可供选择的字体较少
- 没有连字符连接，这使得让在狭窄的栏里的文字两端对齐时会显得很难看
- 很难控制字距调整
- 缺乏对排好的版面将如何显示的控制，设计师必须考虑在很多不同的屏幕尺寸、分辨率和环境下排好版的页面将如何显示

以下我们就来详细看看这些限制因素。

#### 可供选择的字体较少

在格式化文本时，你通常遇到的第一个问题就是可供选择的字体较少。虽然你可以在 CSS 中指定你喜欢的任何字体，但只有在网站访问者的计算机中也安装有该种字体的情况下，他们才能看到文本是以该种字体显示的。否则的话，他们使用的浏览器会用你在 CSS 中指定的替代字体来显示，或是用浏览器的默认字体（通常是 Times New Roman 字体）来显示。因此，也许你喜欢网页的所有正文内容都以你指定的字体如 Trump Medieval 字体或 Avant Garde 字体来显示，但除非你的目标受众和你的偏好很一致，他们是不太可能从这种字体设置中受益的。正是因为这个原因，多数 Web 设计师都仅使用一些最为常见的字体，通常包括：

- Andale Mono 字体
- Times New Roman 字体
- Georgia 字体
- Verdana 字体
- Arial/Arial Black 字体
- Courier/Courier New 字体

- Trebuchet MS 字体
- Comic Sans 字体（很多人认为这种字体难看，因此除偶尔可在面向儿童的网站中使用这种字体外，要避免使用这种字体）
- Impact 字体

这些字体的外观如图 1 所示：

Andale Mono  
Times New Roman  
Georgia  
Verdana  
Arial  
Courier New  
Trebuchet MS  
Comic Sans  
**Impact**

图 1：能在所有系统上正常显示的一些最为常见的字体

为文本选定以上这些最为常见的字体中的任何一种，意味着你选定的字体很可能是绝大多数用户也都已安装的字体。微软在其 Windows Vista 和 Windows XP 系统中引入了 6 种新字体，名称都是以字母 C 开头的，即 Cambria 字体、Calibri 字体、Candara 字体、Consolas 字体、Constantia 字体，和 Corbel 字体。但我建议你们最好还是不要使用这 6 种字体，原因是

在 Mac 平台或 Linux 平台上它们似乎还不可用。

因此与有上千种字体可用的印刷设计师相比，Web 设计师可放心选择的字体只有十几种。不过这是一个重大的限制因素吗？应该不是。排版远不止是选择有吸引力的字体那么简单，它还涉及设置行长、字母间距、空白等诸多工作。不要忘了，那些生活在电子字体出现以前时代的排印工人也面临一样的限制的。

### 连字符连接

当你想对齐在其容器里的文本时，有四种对齐方式可供选择：左对齐、右对齐、居中对齐、两端对齐。两端对齐的文本，段落的左右两端都与容器的垂直边对齐，看起来要比段落一端参差不齐的文本好看些，你在很多杂志和书籍上都可以看到这种两端对齐的排版方式。但是在网页上，由于缺乏自动连字符连接，采用两端对齐就可能出现问题。为使文本显示为两端对齐，浏览器所能做的只是调整单词之间的字距，这可能导致文本中出现大量的空白—当文本块内的行距过短，没有足够的空间来很精细地调整单词之间的字距时，通常就会出现这种情况，如图 2 所示。

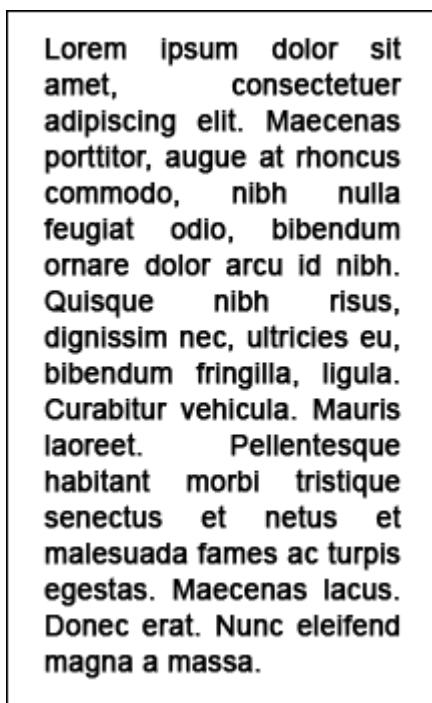


图2：大量空白使两端对齐的文本块显得很不好看

从这个截屏图中你们可以看到，缺乏连字符将单词自然地分割开来，造成一些单词之间出现过大的空白。为避免出现这种情况，网页上多数地方的文本都应使用左对齐的方式。

### 字距调整

字距调整指在使用比例字体（如 Times New Roman 字体，字符之间的间距各不相同）时，调整字母之间的间距，而使用等宽字体（如 Courier 字体，字符之间的间距都是相同的）时，不需要调整字母之间的间距。在印刷排版中，字距调整用于紧缩一些字母之间的间距，如 W 之

后是 A 时，以使它们排列自然，这样可以使文字排版显得更专业。多数专业字体都带有内建的字距调整指示，向排版技师提供字距信息。请看图 3，它显示了字距调整的一个示例。

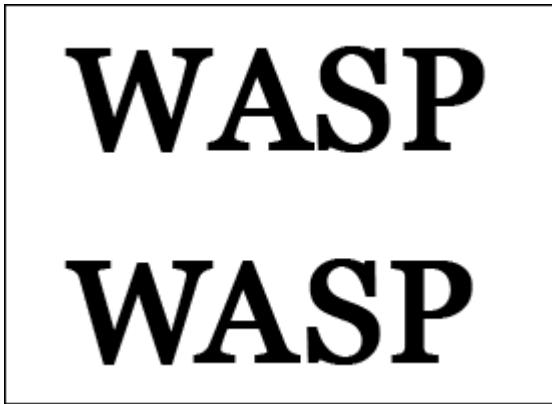


图 3：字距调整无疑改善了文字的外观

在以上这个截屏图中，第一个单词没有进行字距调整，而在第二个单词中，W 和 A 之间的间距缩小了，S 和 P 之间的间距又略微增大了一点。

而在网页上，要对特定单词进行如此精确的字距调整尚无法做到。要调整字距，我们只能使用 tracking（全篇字距调整），这个词在印刷业指同时调整整篇文稿中所有字符之间的间距，也就是说，你虽然可以缩小 W 和 A 之间的间距，但这种缩小对其它所有字母之间的间距同样有影响。在网页上，tracking 更为常用的名称是 letter spacing（字母间距），可用 CSS 加以控制，如图 4 所示。



图 4：在网页上对特定单词进行精确的字距调整尚无法做到，目前能做到的只是全篇字距调整

在以上这个截屏图中，每个字符之间的间距都相同地增大了一点。虽然这样使 A 和 S 不至显得靠得过近，但 W 和 A 之间的间距却又显得过大了。正是由于这个原因，CSS 的字母间距调整属性是很难有效使用的属性，最好谨慎地使用。

#### 缺乏控制

以上我们一直在说印刷业如何如何，这里你需要牢记非常重要的一点，那就是网页不同于印刷品。这也就是说，印刷设计师不用担心读者会自己调整文字的大小，或未安装某些字体，或未开启使字体的边缘平滑显示的功能，而我们这些网页设计师就会担心这些问题。很多网页设计师常试图迫使访问网页的人接受某一特别的设计，如固定文字大小，或将文本放在固定宽度和高度的容器内，甚至用图像替代整块文本等。

其实这种缺乏对排好的版面将如何显示的控制，不必成为一个问题，只要你认识到人们是希望在不同的设备上以不同的方式阅读你的网页上的内容。你不应为他们访问网页设置任何障碍，而应使他们能尽可能容易地阅读网页上的内容。他们也许想在坐通勤车回家的途中在移动设备上阅读网页的内容；他们也许会选择将网页所有内容都打印出来，在纸上而不是在电脑屏幕上阅读这些内容；他们也有可能是视力受损的人，需要将字体扩大一些。你在格式化网页上的文本时，实质上是在向所有这些不同的浏览设备提供你希望这些文本如何被显示的指导。当然这些浏览设备不一定就按你希望的样子显示网页，但这也不是什么问题。这里真正重要的是你不要试图把你的设计强加给所有受众。

### 如何进行网页排版？

应使用 CSS 控制网页的排版。CSS 的网页排版功能十分强大，使用 CSS，你不仅可以控制文字的大小、颜色和字体，还可以控制行高、字母间距、大小写（全部大写、首字大写、小型大写或全部都小写等），甚至还可以控制文本的第一个字或第一行的样式。

通过格式化文本块中包含的文字，你还能控制文本的对齐方式和行长。不仅如此，你只需要在某一位置创建你的样式规则（即样式表），就可以让这些样式规则应用于整个网站的所有文本（当然你也可以为特定的段落或网页的某些区域创建特别的样式表）。此外，任何时候你发现需要让网站上的文字增大，或是需要改变正文的字体，你只需更改样式表中的“值”即可。

### 网页排版的一些诀窍

以下是网页排版的一些诀窍：

#### 选择字体范围

在规定你希望显示的字体时，指定一些备用的字体，这是一个良好的习惯。因此，与只指定使用“Georgia 字体”相比，更好的做法是指定“Georgia, Cambria, "Times New Roman", Times, serif”。这样浏览器在显示字体时，会首先试着使用第一选项 Georgia 字体，如果访问者的计算机上未安装 Georgia 字体，则试着使用 Cambria 字体，如果也未安装，则试着使用 Times New Roman 字体，然后是 Times 字体。如果以上字体都未安装，最后就会使用被操作系统定义为“serif”（衬线字体）的任何一种字体。

#### 行长

为增强易读性，文本块内每行文字的平均长度应为每行 40—60 个字符，当然具体还要根据不同的受众对象而有变化（儿童喜欢每行短一些，而成人喜欢每行长一些）。适宜的每行长度如图 5 所示：

**Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor, augue at rhoncus commodo, nibh nulla feugiat odio, bibendum ornare dolor arcu id nibh. Quisque nibh risus, dignissim nec, ultricies eu, bibendum fringilla, ligula. Curabitur vehicula. Mauris laoreet. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Maecenas lacus. Donec erat. Nunc eleifend magna a massa.**

图 5：每行 60 个字符—适宜的每行长度

在以上这个截屏图中，每行有约 60 个字符。如果每行内的字符数超过这一数字，阅读者就可能不得不开始移动他们的眼睛或甚至是头部来看全每行文字，这样可能会让阅读者的眼睛更疲劳，同时也降低了文本的易读性。

### 行高

行高指文本行的基线间的距离。为使文本更易读，你可以将行高设置为比浏览器的默认行高更高一点（这样还可以使手写或草写字体的字符有更大的间隔）。请看图 6 中两段不同行高的文字之间的差别：

**Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor, augue at rhoncus commodo, nibh nulla feugiat odio, bibendum ornare dolor arcu id nibh. Quisque nibh risus, dignissim nec, ultricies eu, bibendum fringilla, ligula.**

**Curabitur vehicula. Mauris laoreet. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Maecenas lacus. Donec erat. Nunc eleifend magna a massa.**

图 6：不同的行高可以让文本的外观呈现出很大的差别

在以上这个截屏图中，第一段文字的行高为默认行高，给人的感觉是有点密密麻麻的。而第二段文字的行高则增加了一点，这样文字排列得要舒展一些，更为易读。不过你也要注意，行高设置得过大，同样也会让文本难于阅读的。

### 首字母下沉

在 CSS 中，通过为 `first-letter` 伪元素设定 `p:first-letter { }` 这样的样式规则，你可以让一行文字的首字母的样式与其它字母的样式不同，这种样式设置通常被称为首字母下沉，即让首字母下沉 3—4 行，如图 7 所示。

**L**orem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor, augue at rhoncus commodo, nibh nulla feugiat odio, bibendum ornare dolor arcu id nibh. Quisque nibh risus, dignissim nec, ultricies eu, bibendum fringilla, ligula.

图 7：首字母下沉示例

#### 小型大写字母

有时候我们需要使用字体的小型大写字母变体，即单词的每个字母都大写，但大小和单词的小写体差不多。当你想将一些词大写但又不想让其过于醒目时，就可以使用小型大写字母（如用于缩写词等）。即使操作系统内没有安装某一特定字体的小型大写字母变体，这也不是什么问题，因为浏览器会生成该字体的小型大写字母变体（即把该字体的正常大写变体缩小约 30%）。

图 8 为小型大写字母的实例。

**ABANDON HOPE  
ALL YE WHO ENTER HERE**

图 8：小型大写字母的实例

#### 悬挂标点

如果网页中文本的句子以引号开头，你可以使用悬挂标点。具体做法是使用 `text-indent` 这个 CSS 属性，并为其加上负数值，`ems` (-10em)，`points` 磅数 (-10pt)，`pixels` 像素 (-10px)，`percent` 百分比 (-10%) 都可。这样引号的边缘就悬挂在文本左边距以外，使文字在视觉上呈现左对齐状态，如图 9 所示：。

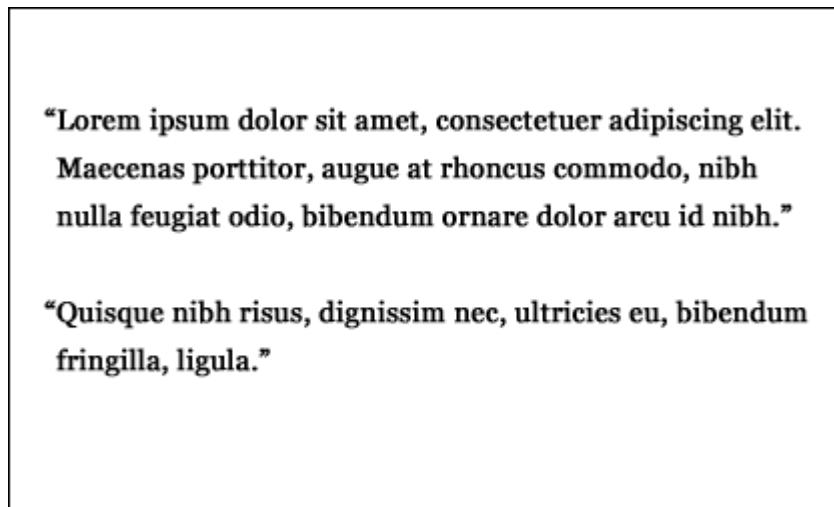


图 9：悬挂标点

#### 排版正确的标点和其它 HTML 实体

你可以通过使用多种可用的用于排版的 HTML 实体，如“智能”或“卷曲”的引号，和 en—、em—破折号等，让你的文本看起来更专业、更雅致。许多博客软件和字处理软件在你键入文本时都会自动为你做这种调整，比如将你一直习惯使用的直引号改为排版正确的卷曲引号，将你使用的不规范的破折号改为 en—破折号和 em—破折号等。图 10 为排版正确的标点符号的示例。

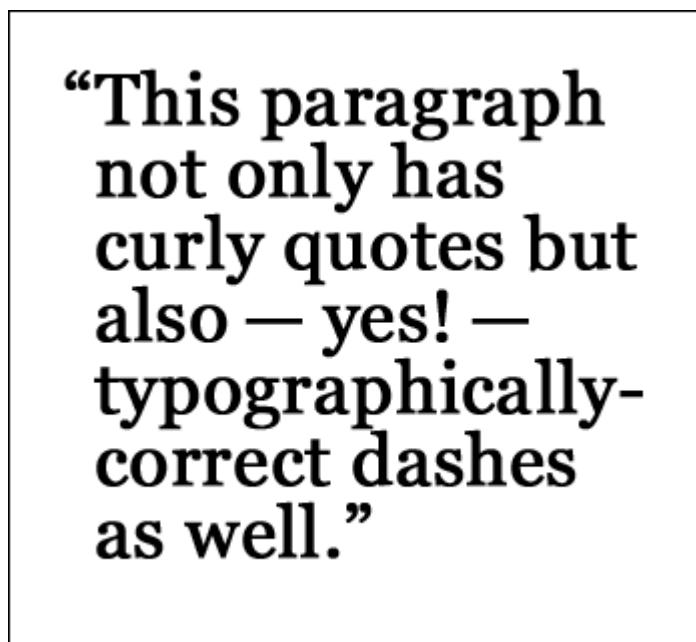


图 10：排版正确的标点符号

在你的文本中使用了智能标点后，文本会显得更专业、更雅致，换句话说，更像杂志和书籍的版面。不过有一点你要注意，对那些使用较老式的显示器或未开启使字体的边缘平滑显示的功能的用户来说，这种智能标点可能会显得像素较粗，因此要谨慎使用。

为插入一些键盘上没有的特殊字符，你可以将一些特别的 HTML 实体插入文本中，以生成这些特殊字符。图 11 中显示了多种此类 HTML 实体：

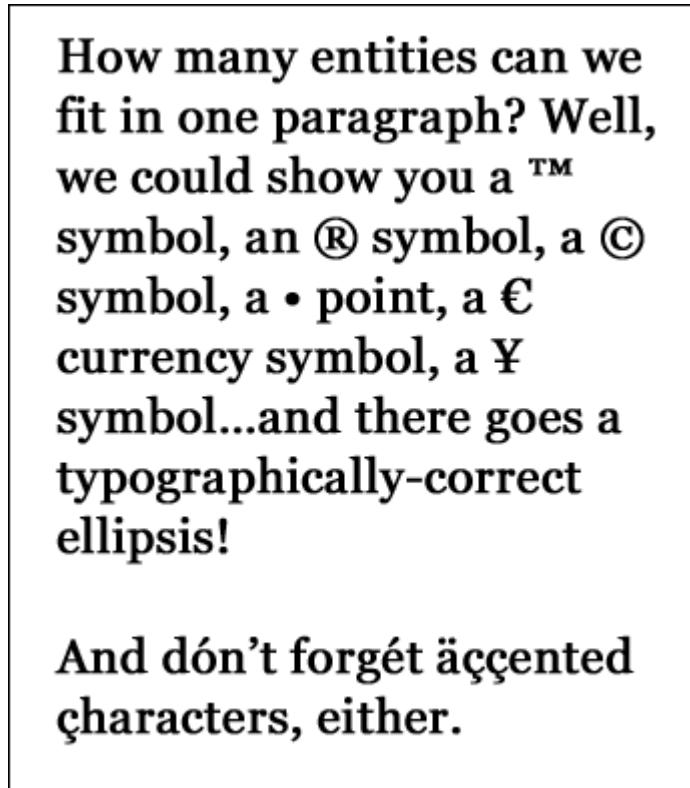


图 11: HTML 实体

你可以自己键入这些 HTML 实体，不过有很多内容管理软件可以为你转换或插入这些 HTML 实体。

### **重要引述**

“重要引述”（pull-quote）指你的文本中的一小段文字，在页面其它地方以更大的字号（有时还用不同的字体）显示，以引起阅读者的注意。在你读过的几乎每本杂志上你可能都已看到过这种“重要引述”，它是有效地分隔你的文本和让重要的引语和短句突出显示的有效方式。在网页上摆置“重要引述”也很容易，只需使用一些简单的标记和样式即可。你可以为“重要引述”设置更大的字号，也许还可以使用不同的字体，然后将其定位，把文本包裹进去，这样就行了。此外，还有更先进的方法可用，即用 JavaScript 挑选出选定的文本，然后自动创建出“重要引述”，这样你就不必在你的标记中两次书写同一个文本。

### **总结**

本篇讲述的主题是网页排版。你们在读完这篇文章后，就能知道网页上的文本不是单调到只使用 Verdana, small, #333333 这样一种样式，是有很多排版技巧和诀窍可用的，它们可以帮助你排出雅致美观、独具风格的文本。对绝大多数网站来说，人们去访问它们就是为了去阅读

网站上的内容，因此作为一个网页设计师，你应该通过专业的网页排版，让人们能轻松愉悦地阅读网页上的内容。

### 练习题

- Kerning（字距调整）和 Tracking（全篇字距调整）有什么差别？哪一种方式是网页设计师可用的？
- 你如何避免网页文本中出现大量空白？
- 说出通过 CSS 可设置的 4 种不同的大写样式。
- 正文内容适宜的行长是多少？哪些因素对行长有影响？
- 衬线字体和无衬线字体的差别何在？分别说出一种衬线字体和无衬线字体的具体字体名称。
- 悬挂标点和普通标点的差别何在？
- 如果你想向网页文本中插入版权符号，你可以使用一个 HTML 实体。请上网看看你是否能找到其它所有的 HTML 实体。实际上有约 250 种 HTML 实体！
  
- 上一篇：配色方案和设计样板
- 下一篇：HTML 基础知识
- 目录

## 作者简介

在刚成年的时候，Paul Haine 离开了偏远、阴冷的萨默塞特郡（英国的一个郡），但他没想到的是此后 6 年，他都还是需要在同样偏远、阴冷的肯特郡读书，主修历史，同时在业余时间学习Web标准。在牛津大学待了两年并写出 [HTML Mastery](#) 后，他搬到了伦敦著名的伊斯林顿区居住，担任 [The Guardian](#)《卫报》的客户端开发员。

Paul 的个人网站，<http://joeblade.com>，已经无法访问，也许已经关闭了。

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 12. HTML 基础知识

Posted 12/10/2008 - 17:47 by Lewis

作者: mnfrancis · 2008 年 7 月 8 日

- [上一篇: 网页排版](#)
- [下一篇: HTML 的 <head> 元素](#)
- [目录](#)

### 序言

在本篇文章中，你们将学习 HTML 基础知识，包括 HTML 的定义、功能，它的简要发展史，以及 HTML 文档的结构是什么样的。后面的几篇文章将详细讲述 HTML 的各个组成部分。本篇文章的内容目录如下：

- [HTML 是什么](#)
- [HTML 是什么样子的](#)
- [HTML 的发展史](#)
- [HTML 文档的结构](#)
- [HTML 元素的语法](#)
- [块级元素和内联元素](#)
- [字符引用](#)
- [总结](#)

### 什么是 HTML

绝大多数读写文件的桌面应用程序都使用某一种特定的文件格式。例如，Microsoft Word 用于读写文件扩展名为“.doc”的文件，而 Microsoft Excel 用于读写文件扩展名为“.xls”的文件。这些文件包含如何在你下次打开文档时重建文档的指令，文档的内容是什么，以及文档的“元数据”，如文档的作者、最后修改日期等，甚至还可以包括已做的改动的列表，让你可以查看文档的不同版本。

HTML（“超文本标记语言”）是一种描述 Web 文档的内容的语言。HTML 使用特殊的语法来包含标记（被称为“元素”）——文档中的文本都被包裹在标记中，来指示用户代理（user agents）应该如何解释文档的各个部分。

注意，我们在这里使用了一个技术术语“用户代理”，而不是“Web 浏览器”。所谓“用户代理”

是指用于代表用户访问网页的任何软件。这里需要指出的是，“用户代理”和“Web 浏览器”之间有一个重大的区别：即所有类型的桌面浏览器（如 Internet Explorer, Opera, Firefox, Safari 等）以及其它浏览器（如 Wii 互联网频道、手机浏览器如 Opera Mini、iPhone 上的 WebKit 浏览器等）都是“用户代理”，但并不是所有的“用户代理”都是浏览器软件。Google 和 Yahoo! 用于为其搜索引擎编制网页索引的自动程序也是“用户代理”，但没有人在直接控制这些自动程序。

## HTML 是什么样子的

HTML 就是内容和其一般含义的纯文本形式表现。例如，标题为“The Purpose of HTML”的 HTML 代码看起来是这样的：

```
<h2 id="htmllooks">What HTML looks like</h2>
```

“<h2>”部分是一个标记（marker），我们称之为一个“标签”（tag），其含义是“这之后的文本应被视为一个二级标题”。“</h2>”是一个指明二级标题在哪里结束的标签，我们称之为一个“结束标签”。起始标签、结束标签，以及它们之间的所有文本，被称为一个“元素”（element）。许多人在使用“元素”和“标签”这两个术语时，将二者视为可互换的意思完全相同的术语，严格说来这是不正确的。`id="htmllooks"` 是一个属性（attribute）。在后面我还将更为详细地讲述这些术语。

在绝大多数浏览器中，都有一个“源代码”或“查看源代码”选项，一般在“查看”菜单下。如果你使用的浏览器有这个选项，请现在就选定它，查看本页面的 HTML 源代码。

## HTML 的发展史

在 互联网和 Web 的历史这篇文章中，你们已经了解到一些关于现代 Web 发展史的知识。当 Tim Berners-Lee 发明万维网时，他创制了第一个 Web 服务器、Web 浏览器和第一个版本的 HTML。

尽管自其诞生以来，HTML 已有了相当大的变化，但我们目前使用的 HTML 的很多内容都可以在 Tim Berners-Lee 写的第一个 HTML 文档中找到，而且在最初的那个“HTML 标签”文件所描述的“标签”中，有超过一半以上的“标签”现在依然存在。

随着越来越多的人开始编写网页，以及更多的浏览器的不断出现，更多的特性和功能被加入 HTML。许多新特性和功能得到普遍采用（如使用 `img` 元素向文档中插入图像，这最先是在 NCSA Mosaic 浏览器中得到使用的）。有些特性和功能则具有专有性，只在一两种浏览器中得到使用。在这种情况下，要求对 HTML 进行“标准化”的呼声日益高涨，这样其它 Web 浏览器软件的作者就能有一个明确描述 HTML 是什么样子的文件（被称为“规范”），可以判断是否遗漏执行了 HTML 的某些部分。

互联网工程工作组 (IETF, 一个负责制订互联网标准的组织)于 1993 年发布了一个 HTML 草案, 它未能成为一个标准, 并在 1994 年就过期了, 但它促使 IETF 创建了一个研究 HTML 标准化的工作小组。

1995 年, 基于 1993 年那个 HTML 草案的“HTML 2.0”编写完成。同时, 另一个被称为“HTML+”的方案也由 Dave Raggett 编写完成, 它被用作浏览器执行的很多新的 HTML 元素的标准 (如向文档中插入图像的方法等, 向文档中插入图像最先是在 NCSA Mosaic 浏览器中得到使用的)。

1995 下半年, HTML 3.0 的草案开始编写, 但由于缺乏浏览器制造商对工作方向的支持, 制订 HTML 3.0 的工作停顿了下来。HTML 3.2 放弃了 HTML 3.0 中的很多新特性和功能, 转而采用了随后流行的 Mosaic 浏览器和 Netscape Navigator 浏览器创造的很多新特性和功能。

1997 年, 万维网联盟 (W3C) 发布了 HTML 4.0, 将其作为建议使用的规范。HTML 4.0 采用了更多的浏览器特定扩展, 同时通过将一些 HTML 元素标注为不建议使用的元素 (即这些元素已过时, 将不再出现在以后更高版本的 HTML 规范中), 万维网联盟也试图净化 HTML, 使之更为合理。万维网联盟希望通过 HTML 4.0 的发布, 促使人们在文档中使用更好的、更为语义化的 HTML (详细信息请参见 Web 标准所包括的模块这篇文章)。

1999 年, 万维网联盟发布了 HTML 4.01, 并在 2001 年做了一些勘误。这是最近版本的 HTML 规范, 不过目前 HTML 5 已正在起草中。

2000 年, 万维网联盟还发布了 XHTML 1.0 规范, 即把 HTML 重构为有效的 XML 文件。

## HTML 文档的结构

以下是一个最简单的有效 HTML 文档示例:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>Example page</title>
</head>
<body>
<h1>Hello world</h1>
</body>
</html>
```

在以上这个 **HTML** 文档中，第一行是文档类型元素，或被称为 **doctype**（文档类型，将在 [为你的 HTML 文档选择适当的文档类型这篇文章中](#)详细讲述）。它描述所使用的是哪一种类型的 **HTML**，这样“用户代理”就能确定该如何解释这个文档，并确定它是否遵守所声明将要遵守的规则。

在文档类型元素之后，你可以看到 **html** 元素的起始标签，它包裹了整个文档。最后是 **html** 的结束标签，在任何一个 **HTML** 文档中，它都位于结尾处。

在 **html** 元素之中，有一个 **head** 元素，它包裹的是关于文档的信息（元数据）。关于 **head** 元素，将在 [下一篇篇文章中](#)详细讲述。在 **head** 元素之中，有一个 **title** 元素，它定义将被显示在标题栏中的“**Example page**”标题。

在 **head** 元素之后，有一个 **body** 元素，它包裹的是本页面的实际内容，在本示例中，就只有一个一级标题（**h1**）元素，其中包含“Hello world.”这个文本。这就是我们这个 **HTML** 文档的全貌。

正如你们看到的那样，元素中常常包含其它元素。文档主体中通常都包含有很多嵌套元素。页面分区（**divisions**）创造出文档的整体结构，其中将包含小分区（**subdivision**）。它们将包含标题、段落、列表等等。段落又可包含链接到其它文档的元素、引用、强调等。在以后的课程中，你们将更多地了解这些元素。

## HTML 元素的语法

正如你们已看到的那样，**HTML** 中的一个基本元素由两个包裹一块文本的标记组成。不过有些元素并不包裹文本，同时几乎在所有情况下，元素中都可包含子元素（如在以上那个示例中，**html** 元素包含 **head** 和 **body** 元素）。

元素也可以有属性，属性可修饰元素的行为并引入额外的含义。

```
<div id="masthead">

 <h1>The Basics of

 <abbr title="Hypertext Markup Language">HTML</abbr>

 </h1>

</div>
```

在这个示例中，一个 **div** 元素（页面分区，一种将文档划分为逻辑块的方式）被添加了一个 **id** 属性，而属性又被赋予了一个“值”（**value**），即 **masthead**。这个 **div** 元素包含一个 **h1** 元素（一级标题，或最重要的标题），该 **h1** 元素中又包含一些文本。这些文本的一部分被包裹在一个 **abbr** 元素（用于定义缩略语）中，该元素的属性为 **title**，属性的值为 **Hypertext**

Markup Language。

**HTML** 中的很多属性都通用于所有元素，不过有些属性仅适用于一种或几种特定的元素。所有属性的格式都是 `keyword="value"` 这种格式。“值”必须位于单引号或双引号之间（一些情况下可不用引号，但这不是一个好的习惯，因此你应当把“值”始终放在单引号或双引号之间）。

在 **HTML** 规范中，定义了绝大多数的属性及它们可能有的“值”。你如果自己编造属性的话，会使 **HTML** 无效，因为这样会让“用户代理”无法完全正确地解释网页。唯一的例外是 `id` 和 `class` 属性，它们的“值”是完全由你控制的，这是因为它们的功能就是让你为文档添加你自己的含义和语义。

包含在另一个元素中的一个元素被称为另一个元素的“子元素”。在以上那个示例中，`abbr` 就是 `h1` 的子元素，而 `h1` 又是 `div` 的子元素。相反地，`div` 被称为 `h1` 的“父元素”。父元素 / 子元素概念是很重要的概念，它构成了 **CSS** 的基础，并在 **JavaScript** 中大量使用。

## 块级元素和内联元素

**HTML** 中的元素可分为两大类，即块级元素和内联元素，对应于这些元素所代表的内容和结构的类型。

块级元素指更高层级的元素，通常用于显示文档的结构。通俗地说，如果元素是块级的，那么它总是在新行上显示。一些常见的块级元素包括段落、列表项目、标题和表格。

内联元素指那些包含在块级结构元素只中的元素，只包裹文档内容的一小部分，而不是包裹文档内容的整个段落和编组。内联元素不在文档的新行上显示，而是在文本的一个段落（当前行）中显示。一些常见的内联元素包括超文本链接、突出显示的单词或短句，以及简短的引述。

## 字符引用

最后我们来看看如何在 **HTML** 文档中插入特殊字符。在 **HTML** 中`<`，`>`，和 `&` 这三个字符都是特殊的字符，它们用于表示 **HTML** 的起始和结束，而不是代表小于符号、大于符号和 `&` 符号。

网页作者最先可能犯的错误之一就是在 **HTML** 文档中使用 `&`，然后就出现意想不到的错误。例如，写入“Imperial units measure weight in stones&pounds”（意为，英国的重量单位为英石和磅），“stones&pounds”在一些浏览器中可能被显示为“...stones£s”。

这个问题的原因是，在 **HTML** 中“`&pound;`”这个字符串实际上是一个字符引用。字符引用适用于以下情况，试图在文档中包含某些无法或难于使用键盘输入的字符，或者是在一些特殊的文件编码中，保存并表示一些不属于自己编码集的字符。

`and` 符号（`&`）表示字符引用开始，半角分号（`;`）表示字符引用结束。不过很多“用户代理”

都可以容忍一些 HTML 错误，如遗漏分号等，这样就会把“&pound;”视为字符引用处理。需要说明的是，字符引用可以为数字形式（数字字符引用——**numeric references**），也可以为简写词形式（实体引用——**entity references**）。

在 HTML 文档中输入真正的 & 符号，可以采用字符实体引用，即输入“&amp;”，或采用数字字符引用，即输入“&#38;”，你可以在 [evolt.org](http://evolt.org) 这个网站上找到完整的字符引用表。

## 总结

在本篇文章中，你们学习了 HTML 的基础知识，了解到了 HTML 文档结构的一些初步知识。在后面的文章中，我们将接着深入学习 HTML 文档的 `<head>` 元素和 `<body>` 元素。

- 上一篇：网页排版
- 下一篇：HTML 的 `<head>` 元素
- 目录

## 作者简介



Mark Norman Francis 早在万维网诞生前，就在从事互联网领域的工作了，一直持续到现在。目前他是全球最大网站 **Yahoo!** 的前端设计师，负责制定 Web 开发的最佳习惯、代码标准和质量标准。

在加入 **Yahoo!** 前，他先后在 **Formula One Management** (F-1 管理公司)、**Purple Interactive** (紫色互动公司)、伦敦城市大学从事过多种工作，包括 Web 开发、后端 CGI 编程和系统架构等。他的博客地址为：<http://marknormanfrancis.com/>。

---

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 13. HTML 的 `<head>` 元素

Posted 12/10/2008 - 17:47 by Lewis

作者: Christian Heilmann · 2008 年 7 月 8 日

- 上一篇: [HTML 基础知识](#)
- 下一篇: [为你的 HTML 文档选择适当的文档类型](#)
- 目录

### 序言

在本篇文章中, 我要专门讲述 HTML 文档中一个未引起人们应有重视的部分, 即 `head` 元素中的标记。在学习完本篇文章后, 你们将更多地了解本章节中的各个不同部分, 以及它们各自的功能, 包括文档类型、`title` 元素、关键词及描述 (使用 `meta` 元素处理)。你们还将了解到 JavaScript 和 CSS 样式, 以及不要在 `head` 中写入什么。你们可以 [点击下载一些示范文件](#), 我在本篇文章中将不时提到这些示范文件。在完成本篇文章的学习后, 你可以随意修改这些示范文件。请一定从头到尾地学完这篇文章, 它能让你学会处理 HTML `head` 的最佳习惯。本文章各个部分分别讲述一个主题, 然后在文章最后给出了总结性的结论, 以让你重新思考前面各个部分给出的建议。本篇文章的内容目录如下:

- [头部信息包含些什么信息?](#)
- [设置文档的首选语言](#)
- [通过文档的标题来判断文档](#)
- [添加关键词和描述](#)
- [添加样式](#)
- [使用 JavaScript 添加动态功能和特性](#)
- [注意! 使用内联 CSS 和 JavaScript 并非很好的做法!](#)
- [总结](#)
- [练习题](#)

### [头部信息包含些什么信息?](#)

在前面的课程中, 你们已经了解到一个有效的 HTML 文档需要有一个文档类型 (`doctype`), 文档类型说明文档使用的是什么类型的 HTML, 并指示浏览器相应地显示 HTML 文档。Roger 将在第 14 篇文章中详细地讲述文档类型, 简而言之, 文档类型规定 HTML 文档需要有一个其中包含有 `head` 和 `body` 元素的 `html` 元素。`body` 部分包含文档的全部内容,

因此将是你编写文档时在上面所花时间最多的部分。相比之下，`head` 部分的作用似乎就不那么重要，因为除了 `title` 元素，其它写入 `head` 部分的信息，是网站访问者在浏览器中不能直接看到的。`head` 部分是向浏览器发出的绝大多数指示开始的地方，也是你储存有关文档的附加信息（所谓 `meta` 元信息）的地方。

### 设置文档的首选语言

`HTML` 文档的这条信息会在 `head` 元素的父元素 `html` 元素上继续。我们这里是在 `head` 元素中定义文档的首选自然语言。这里的“自然语言”指人类语言，例如法语、泰语或英语等。设置文档的首选语言，可以帮助屏幕阅读器阅读网页，这是因为比如说“six”（六）在英语和法语中的读音差别就非常大；这还可以帮助搜索引擎。因此定义文档的首选语言是一个好的做法，在你编写面向国际受众的网页时就更是如此了，尽管现在在源代码中定义了文档首选语言的网页还不是很多。以下是在 `HTML` 文档中定义文档首选语言的示例：

```
<html lang="en-GB">
 ...
</html>
```

请注意，你还可以在其它元素中使用 `lang` 属性，为文档的小分区设置语言，例如 `<span lang="fr">Bonjour</span>`。

你使用哪些属性来设置语言，取决于你的文档的 `DOCTYPE`。万维网联盟（W3C）发布的文件中写道：

对 `HTML` 文档，仅使用 `lang` 属性。对用作 `text/html` 的 `XHTML 1.0` 文档，使用 `lang` 和 `xml:lang` 属性。对用作 `XML` 的 `xHTML` 文档，仅使用 `xml:lang` 属性。

语言代码可以是两个字母的代码，如 `English`（英语）的代码是 `en`，也可以是四个字母的代码，如 `American English`（美国英语）的代码是 `en-us`，还可以是一些不常用的代码。两个字母的语言代码由 `ISO 639-1` 定义，不过根据目前的最佳习惯，你应当 使用 `IANA subtag registry` 来定义你的语言代码。

### 通过文档的标题来判断文档

`title` 元素是 `head` 中最重要的元素之一。包含在 `title` 元素中的文本，在所有“用户代理/浏览器”的标题栏上都要显示（浏览器标题栏位于浏览器的最上方）。显示在标题栏上的标题，将是用户在访问网站时看到的第一条内容，因此它是非常重要的。此外，对“辅助技术”如屏幕阅读器（一种为视力受损的用户朗读网页的软件）等而言，标题栏上的标题是显示文档主要内容的提示，对很多搜索引擎来说，也同样是如此。因此，你可能会发现，当你为你的 `Web` 文档选用了一个包含关键词且人类易读的好标题时，它的访问量会激增。请用浏览器打开以下这个

HTML 文档（你下载的 zip 文件中的 headexample.html）。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<title>I am a title example</title>

</head>

<body>

</body>

</html>
```

你会看到包含在 `title` 元素中的文本，被显示在位于浏览器导航栏上方的标题栏上，如图 1 所示。

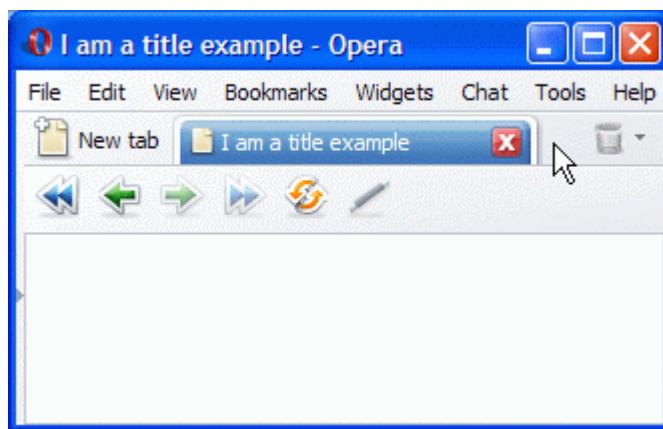


图 1：浏览器标题栏上显示的一个标题

网上有很多关于如何写出好的文档标题的教程，它们中的多数都与搜索引擎优化（SEO）有关。不过有一点要注意，不要为了想试图让搜索引擎在很多搜索结果上都显示你的 Web 文档，就在标题栏上罗列过多的词句。适当的做法是写一个能准确反映文档主要内容的精练的标题栏标题，例如“**Breeding Dogs—Tips about Alsatians**”（意为，狗的饲养——饲养阿尔萨斯狼狗的提示）这样一个标题栏标题，就远比“Dogs, Alsatian, Breeding, Dog, Tips, Free, Pet.”（意为，狗、阿尔萨斯狼狗、饲养、狗、提示、免费、宠物）这个标题栏标题更能为用户所接受和易读。

### 添加关键词和描述

下一步你要做的是添加关键词和描述，第一眼看来这似乎是多余的，因为访问者在浏览器中是不能直接看到这些信息的。关键词和描述都添加在 `head` 内的 `meta` 元素之中，如以下这

个选自 Yahoo! 欧洲网站的 HTML 文档 (`headwithmeta.html`) 所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<title>Yahoo! UK & Ireland Eurosport—Sports News | Live Scores | Sport</title>
<meta name="description" content="Latest sports news and live scores from
Yahoo! Eurosport UK. Complete sport coverage with Football results, Cricket
scores, F1, Golf, Rugby, Tennis and more.">
<meta name="keywords" content="eurosport, sports, sport, sports news, live
scores, football, cricket, f1, golf, rugby, tennis, uk, yahoo">
</head>
<body>
</body>
</html>
```

如果你用一个浏览器打开这个文档，在文档的 `body` 中是看不到任何东西的。但是，如果你把这个文档放在网上，搜索引擎又将之编入索引，则你在 `meta` 元素之中所写入的描述 (`description`) 将显示为搜索结果中链接的下方所显示的文本，如图 2 所示。

[Yahoo! UK & Ireland Eurosport - Sports News | Live Scores | Sport](#)  
Latest sports news and live scores from Yahoo! Eurosport UK. Complete sport coverage with  
Football results, Cricket scores, F1, Golf, Rugby, ...  
[uk.eurosport.yahoo.com/](http://uk.eurosport.yahoo.com/) - 35k - [Cached](#) - [Similar pages](#) - [Note this](#)

图 2：搜索引擎的搜索结果页面中显示的描述

这些信息可能就是网站的潜在访问者正在找寻的，所以才会去点击搜索结果页面显示的链接。“描述”还有另外一个作用，即有些浏览器会在用户将 Web 文档加入“收藏夹”时显示作为附加信息的“描述”，如图 3 所示：

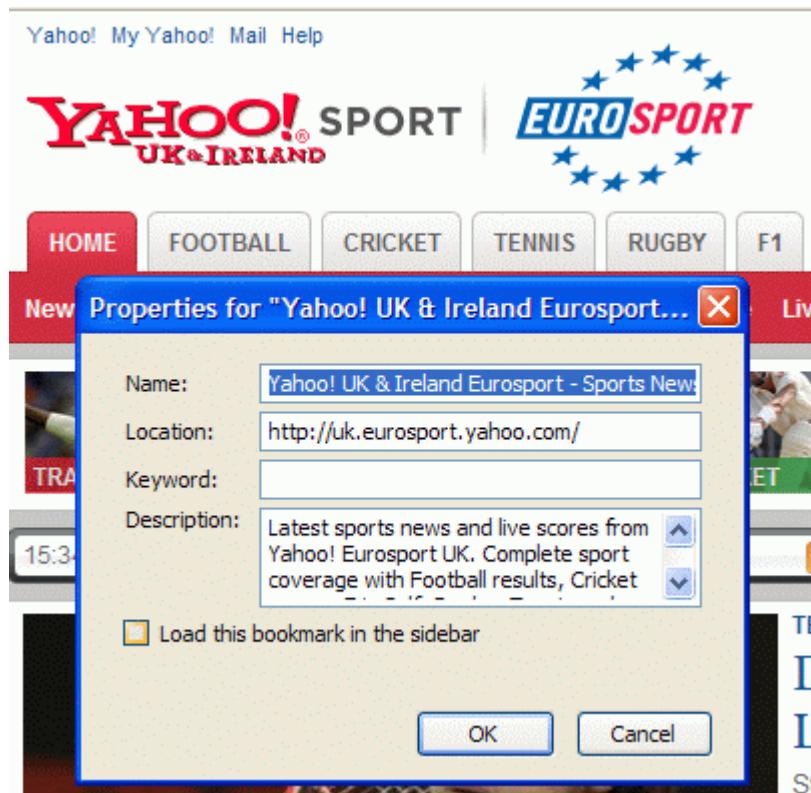


图3：有些浏览器会在用户将 Web 文档加入“收藏夹”时显示描述

因此，尽管在文档中加入元描述没有立竿见影的好处，但在让你的 Web 文档能为更多人访问方面，它们还是发挥着重要作用。同样地，加入关键词也可帮助你的 Web 文档能为更多的人访问。

尽管由于多年来垃圾邮件发送人的滥用，使得搜索引擎不再对关键词很看重，但在以下情况下，即如果你想不必浏览和分析内容就为很多文档快速编制索引的话，它们依然是一种很好的工具。例如，你可以在一个内容管理系统中使用 `meta` 关键词，通过撰写一个脚本为这些关键词编制索引，这样可以让你的搜索引擎运行的速度快很多。这就提供了一种可不必分析文档的内容就找到文档的不错的方式。通过在一个 `meta` 元素中加入一些关键词，这样如果你想在未来创建一个类似的文档索引的话，就可以让站内搜索变得更为智能化、更快速。你可以把关键词视为你在一本砖头厚的大书中留下的小书签，这样你就能更容易地立即找到某个特定章节，而不必一章一章地去费力找了。

### 添加样式

下一个你可以向 HTML 文档的 `head` 添加的是在层叠样式表 (CSS) 中定义的样式规则。你可以通过使用一个 `style` 元素，把那些样式规则直接嵌入 `head`，例如 (`headinlinestyles.html`)：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>

<head>

<title>Breeding Dogs—Tips about Alsatians</title>

<meta name="description" content="How to breed Alsatians, tips on proper breeding and information about common issues with this breed.">

<meta name="keywords" content="Dogs, Alsatian, Breeding, Dog, Tips, Free, Pet">

<style type="text/css">

body{

background:#000;

color:#ccc;

font-family: helvetica, arial, sans-serif;

}

</style>

</head>

<body>

<p>Test!</p>

</body>

</html>
```

如果你用一个浏览器把以上这个 HTML 文档打开，看到的将是，在黑色的背景上显示的“Test!”这个词，它的颜色是灰色的，字体是 Helvetica 字体或 Arial 字体（具体显示是哪一种字体，要依你安装的操作系统而定）。style 元素还可以包含另外一种被称为 media（媒介）的属性，它定义哪种媒介将使用这些样式，即你在计算机屏幕或手持设备上阅读文档时，或在将文档打印出来时，希望使用这些样式吗？有多种媒介类型可供选择，以下是其中一些最有用的媒介：

- 屏幕—在显示器上显示文档
- 打印—定义文档的打印输出稿的外观
- 手持设备—定义文档在移动设备及其它手持设备上显示时外观和样式
- 投影机—适用于以 HTML 格式保存的演示文稿，例如 **Opera Show** 软件就支持这种以 HTML 格式保存的演示文稿

举例来说，如果你为了让网页更适宜打印而想除去它在屏幕上显示出来的色彩，并为网页

上的文本使用更大的字号，则可以在第一个样式块下方添加另一个样式块，赋予它一个属性值为 print 的 media 属性，如以下所示（在 `headinlinestylesmedia.html` 中检查全部代码）：

```
<style type="text/css" media="print">

body{

background:#fff;

color:#000;

font-family: helvetica, arial, sans-serif;

font-size:300%;

}

</style>
```

现在当你准备打印这个网页时，浏览器就知道使用打印样式表而不是屏幕样式表来打印网页。请在浏览器中打开 `headinlinestylesmedia.html`，然后选择打印预览，你看到的将是图 4 中显示出来的样子：

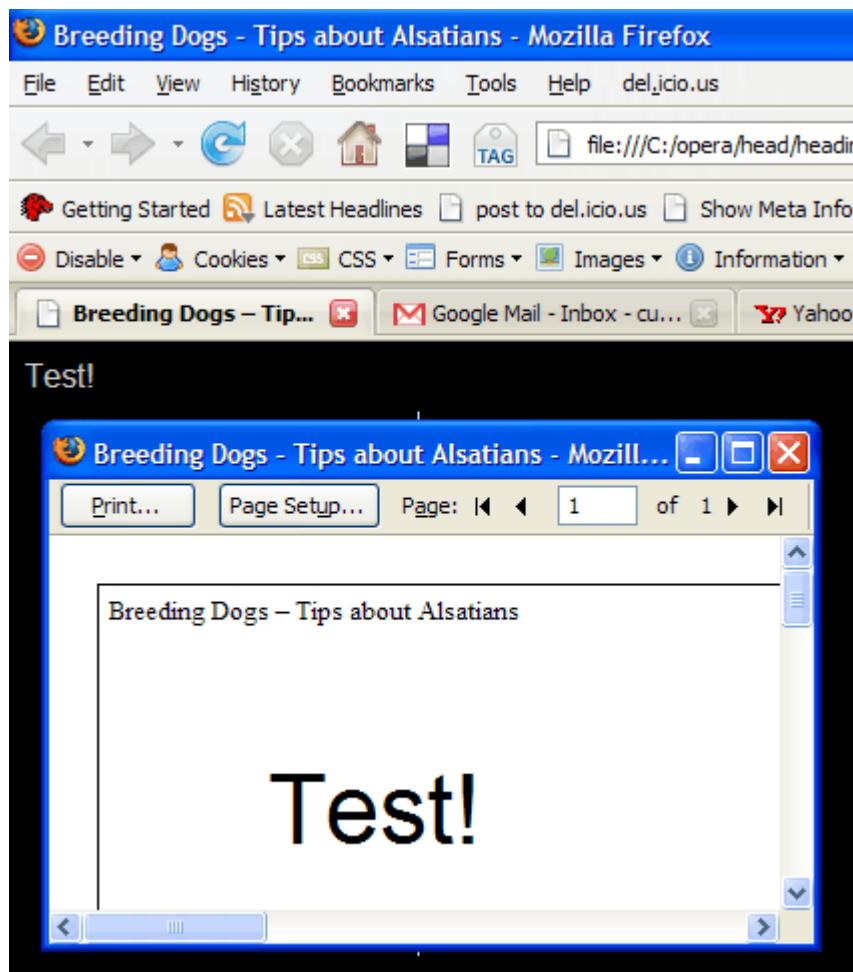


图4: 一个屏幕和一个打印样式表

## 使用 JavaScript 添加动态功能和特性

你还可以向文档的 `head` 添加可由浏览器执行的用 `JavaScript` 编写的脚本，即所谓“客户端脚本”。正如你们 第4篇文章 中已了解到的，`JavaScript` 向静态的 `HTML` 文档添加动态行为，如动画效果、表单数据校验，以及用户执行某些行动时触发的其它一些动态效果等。

你使用 `script` 标签向文档中添加 `JavaScript`。当浏览器遇到文档中的一个 `JavaScript` 时，它会试着去执行其中的代码，并暂停解析文档的剩余部分。这意味着如果你想保证你添加的 `JavaScript` 在主文档载入前就可用，就需要将 `JavaScript` 添加进 `head` 之中。例如，你可以使用以下的脚本（`headscript.html`），提醒访问者如果点击某一个特别的链接，将被带到另一个服务器：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<title>Breeding Dogs—Tips about Alsatians</title>
<meta name="description" content="How to breed Alsatians, tips on proper breeding and information about common issues with this breed.">
<meta name="keywords" content="Dogs, Alsatian, Breeding, Dog, Tips, Free, Pet">
<style type="text/css" media="screen">
body{
background:#000;
color:#ccc;
font-family: helvetica, arial, sans-serif;
}
a {color:#fff}
</style>
<style type="text/css" media="print">
body{
background:#fff;
color:#000;
font-family: helvetica, arial, sans-serif;
}
```

```

 font-size:300%;

 }

</style>

<script>

function leave() {

 return confirm("This will take you to another site,\n are you sure you
want to go?")

}

</script>

</head>

<body>

Test!

The Daily Puppy

</body>

</html>

```

如果你用浏览器打开这个示范网页并点击所显示的链接，则会弹出一个对话框，请你确认是否要去另一个网站。这只是随便选用的一个脚本撰写例子，这种脚本撰写方式远不是目前的最佳习惯。在本课程的后面部分，将深入讲述使用 **JavaScript** 的最佳习惯和传授 **JavaScript** 技术，因此现在你不必担心还不太懂 **JavaScript**。

#### **注意！使用内联 CSS 和 JavaScript 并非很好的做法！**

我知道以上这些词很强烈，其实我只是想让你们记住非常重要的一点，那就是在你构建网站时，最好的办法是尽可能地重用代码。将全网站通用的样式和脚本添加进每一个页面不仅没有多大意义，相反还使维护整个网站的工作变得更为困难，并使各个文档的体积不必要地膨胀。

一个比这好得多的做法则是将你的样式和脚本都放在单独的外部文件内，在需要时再导入你的 **HTML** 文件，这样如果需要对样式和脚本做出改动，你仅需要在一个地方修改样式和脚本即可。为此，对 **JavaScript**，可使用 **script** 元素，但其中却并不包含脚本，而是使用 **src** 属性与一个外部文件链接，如以下代码所示（**externaljs.html**）：

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

```

```

<html>

<head>

<title>Breeding Dogs—Tips about Alsatians</title>

<meta name="description" content="How to breed Alsatians, tips on proper breeding and information about common issues with this breed.">

<meta name="keywords" content="Dogs, Alsatian, Breeding, Dog, Tips, Free, Pet">

<style type="text/css" media="screen">

body{
 background:#000;
 color:#ccc;
 font-family: helvetica, arial, sans-serif;
}

a {color:#fff}

</style>

<style type="text/css" media="print">

body{
 background:#fff;
 color:#000;
 font-family: helvetica, arial, sans-serif;
 font-size:300%;
}

</style>

<script src="leaving.js"></script>

</head>

<body>

Test!

The Daily Puppy

</body>

</html>

```

处理 CSS 就要难一些了。style 元素并没有 src 属性，因此你需要使用 link 元素，

该元素有一个 `href` 属性（用于指定要导入的一个外部 CSS 文件），以及一个 `media` 属性（用于定义这些样式是否应该用于屏幕、打印文档等）。通过将 CSS 和 JavaScript 放在单独的外部文件中，你可以大大缩短文档 `head` 部分的长度，如以下所示（[externalall.html](#)）：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<title>Breeding Dogs—Tips about Alsatians</title>
<meta name="description" content="How to breed Alsatians, tips on proper breeding and information about common issues with this breed.">
<meta name="keywords" content="Dogs, Alsatian, Breeding, Dog, Tips, Free, Pet">
<link rel="stylesheet" type="text/css" media="screen" href="styles.css">
<link rel="stylesheet" type="text/css" media="print" href="printstyles.css">
<script src="leaving.js"></script>
</head>
<body>
Test!
The Daily Puppy
</body>
</html>
```

将 CSS 和 JavaScript 放在单独的文件中，还具有以下优点：

1. 可以使访问者在访问你的网站时更快且更容易，这是因为虽然还需要下载一些额外的文件，但在每个页面上就不需要再重复下载样式和脚本信息了（它们都放在单独的脚本/样式文件中，因此只需要下载一次），这样每个页面的下载时间就都会缩短。此外，单独的 CSS 文件和 JavaScript 文件将存入缓存，这样在你下一次访问这个网站时，因为它们都已在缓存中，就不需要再下载一次了。
2. 使网站易于维护。由于整个网站（上面可能有数以千计的文档）的样式和脚本都放在单独的文件中，这样在你需要对样式或脚本做出某些改动时，只需在单独的样式文件或脚本文件中改动即可，不再需要在上千个文件中一处一处地改动。

## 总结

在本篇文章中，你们已了解到 **HTML** 文档的头部分中可包含的各个不同的元素，它们分别是：

- `title` 元素，用于介绍文档。
- `meta` 元素，包含文档内容的描述以及可让对内容编制索引的工作变得更容易的关键词。
- `link` 元素，指向外部的 **CSS** 文件。
- `script` 元素，指向外部的 **JavaScript** 文件。

确保在你的 **HTML** 文档中以上所有元素都正确和有效，可以使你的文档能被访问者快速和容易地找到和解读。

## 练习题

- 为什么“描述”并不直接在屏幕上显示，还是需要在 `meta` 元素中加入“描述”？
  - 将 **JavaScript** 添加进 **HTML** 文档的 `head` 部分，而不是 `body` 部分，这样做有什么优点？
  - 你如何能从浏览器缓存受益，以及你需要做些什么来利用浏览器缓存？
  - 既然搜索引擎对 **Web** 文档的标题栏标题很重视，是不是罗列一长串相关的关键词来作为标题栏标题就很有用呢？这样做有什么负面效果？
  - 由于标题栏标题的显示可能有点沉闷，使用 `b` 元素让其中一些词加粗显示，这是否是一个好的主意？是否可能做到？
- 
- 上一篇：[HTML 基础知识](#)
  - 下一篇：[为你的 HTML 文档选择适当的文档类型](#)
  - 目录

## 作者简介



照片来源: [Bluesmoon](#)

Chris Heilmann 从事 Web 开发工作已达 10 年，之前他曾从事电台新闻工作。目前他在 Yahoo! 英国工作，担任培训师和首席开发员，并负责监督面向欧洲和亚洲的前端页面的代码质量。

Chris 的博客地址是 [Wait till I come](#)。他用“codepo8”这个网名，出现在很多社交网络网站上。

---

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 14. 为你的 HTML 文档选择适当的文档类型

Posted 12/10/2008 - 17:48 by Lewis

作者: [Roger Johansson](#) · 2008 年 7 月 8 日

- [上一篇: HTML 的 <head> 元素](#)
- [下一篇: 在 HTML 中标记文本内容](#)
- [目录](#)

### 序言

第 13 篇文章剖析了 HTML 文档的 head 部分, 简要讲述了在 head 部分可包含的各种不同的元素, 以及它们各自的功能。在本篇文章中, 我将详细讲述 HTML 文档的文档类型 (doctype), 内容包括: 文档类型所起的作用; 它如何帮助你校验你的 HTML 文档; 如何为你的文档选择一个正确的文档类型, 以及 XML 声明 (你一般不需要使用这种 XML 声明, 但偶尔还是会遇到它)。

- [首先就需要声明文档类型](#)
- [文档类型转换和显示模式](#)
- [校验](#)
- [选择文档类型](#)

- XML 声明
- 总结
- 练习题
- 延伸阅读

### **首先就需要声明文档类型**

在你创建的任何 HTML 文档的开头部分，都应该首先声明文档类型定义（DTD）。如果此前你还从未听说过文档类型定义这个术语，也不用担心。为简便起见，文档类型定义通常被简称为“doctype”（文档类型），在本篇文章的以下部分我都使用“doctype”这个术语。

你也许还不清楚“DTD”或 `doctype` 究竟指什么。所谓 DTD，是“Document Type Definition”（文档类型定义）的缩写，它定义 HTML 文档中所使用的元素和属性遵守哪一种版本的 HTML 规范。尽管目前存在多个 HTML 规范，你也不必为此感到无所适从，你需要做的只是选定其中一个。

文档类型主要用于不同软件的以下两种情况：

1. Web 浏览器使用文档类型来确定它该使用什么显示模式来显示 HTML 文档（关于显示模式，在后面还将更详细地讲述）。
2. 标记校验器将检查文档类型以确定该使用什么规则来校验文档（在后面还将更详细地讲述）。

以上这两点都将对你编写的 HTML 文档有影响，我将在本篇文章的后面部分更详细地讲述。

以下是一个示例：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
```

你们可能还看不懂这段文档类型声明代码，因此下面我将简要地向你们解释这段代码的含义。如果你们想详细了解其中每个字符究竟指什么，请阅读 **!DOCTYPE** 这篇文章。

在以上这个文档类型声明中，最重要的部分是放在引号里的两个字符串。第一个字符串即 “`//W3C//DTD HTML 4.01//EN`”，指明这是一个万维网（W3C）联盟发布的 DTD（文档类型定义）文档，DTD 描述的是 HTML 4.01 规范，DTD 中所使用的语言为英语。

第二个字符串，即 `http://www.w3.org/TR/html4/strict.dtd`，是一个指向这种文件类

型所用的 DTD 文档的 URL 地址。

虽然文档类型看起来可能有点奇怪，但 HTML 和 XHTML 规范都要求文档必须声明其文档类型。如果你未在文档中声明文档类型，则你在使用 W3C 标记校验器或其它用于检查 HTML 文档错误的工具检查你的 HTML 文档的语法时，就会被显示文档存在错误。一些 Web 浏览器甚至内建有这种检查功能；通过安装一个扩展件，其它浏览器也可以具有这种检查功能。

### **文档类型转换和显示模式**

即使你未在 HTML 文档中声明文档类型，浏览器还是要处理和显示该文档。浏览器需要试着去渲染不论是多么奇怪的 Web 文档。尽管如此，由于存在所谓“文档类型转换”，如果你未在 HTML 文档中声明文档类型，则其在浏览器中实际显示出来的样子就可能不是你希望它显示出来的样子。

2000 年以后发布的多数浏览器，都首先要查看所遇到的任何 HTML 文档的文档类型，并使用文档类型来确定编写 HTML 文档的人是否已根据 Web 标准适当地使用了 HTML 和 CSS。

如果浏览器发现文档中声明的文档类型是以适当的代码写入的，则会使用所谓“标准模式”来显示网页。在“标准模式”下，浏览器一般会根据 CSS 规范来显示网页，也就是说，浏览器信任编写网页的人，会按编写网页的人希望的样子显示网页。

另一方面，如果浏览器发现文档中声明的文档类型已过时或不完善，则会使用所谓“怪异模式（Quirks mode）”来显示网页，以向后兼容老习惯和老浏览器。在“怪异模式”下，浏览器会假定 HTML 文档是已过时的老式文档或未按 Web 标准编写的文档，这意味着网页虽然也会在浏览器中显示，但浏览器在渲染网页时要做很多处理，因此网页在浏览器中实际显示出来的样子可能就有些怪异或不好看，而不是你希望它显示出来的样子。

“标准模式”和“怪异模式”之间的差异，主要与 CSS 如何在浏览器中显示有关，基本上与浏览器如何处理 HTML 无关。作为一个 Web 设计师或开发员，你需要确保所有的浏览器都将以“标准模式”显示你设计或开发的网站，以实现具有一致性的显示效果，因此你应当遵循 Web 标准并使用适当的文档类型。

### **校验**

我在前面已提到，HTML 校验器也会使用文档类型，关于 HTML 校验器在本课程后面的文章中还会详细讲述。现在你仅需知道 HTML 校验器是用于检查你的 HTML 文档的语法是否有任何错误的工具就够了。校验器会查看你所使用的文档类型，因而来确定该采用哪些规则来检查你的 HTML 文档。这有点类似于告诉拼写检查程序一个文档是用什么语言写成的，不然拼写检查程序就不知道该使用什么拼写和语法规则来检查你的文档。

## 选择文档类型

现在你们已经知道需要在 **HTML** 文档中插入文档类型以及它在文档中所起的作用了，你们也许会问，我怎么知道该选择哪一种文档类型呢？首先你要明白一点，文档类型不是只有一种，而是有很多种。如果你紧跟前沿技术，甚至可以创造出你自己的文档类型。不过为简明起见，我在这里就不向你们一一介绍很多种不同的文档类型了，而只推介两种文件类型。

如果你的文档是 **HTML** 文档，使用以下这种文档类型：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
```

如果你的文档是 **XHTML** 文档，使用以下这种文档类型：

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

附注：“真正的”**XHTML** 应当以 **XML** 格式交付给浏览器解析，不过何时和如何该这样做，以及这样做的意义何在，已超出本篇文章的讲述范围了。

使用这两种文档类型，都可以确保浏览器会以“标准模式”处理和显示你的文档。使用这两种文档类型，最明显的效果可能就是当你用 **CSS** 对文档进行样式化时，在各种浏览器中会得到更具有致性的显示效果。至于其它一些可供你使用的文档类型，你可以查阅万维网联盟（W3C）发布的，在你的 **Web** 文档中建议使用的 **DTD** 列表。

你们可能注意到了，我推荐你们使用的这两种文档类型都是所谓“严格型”（**strict**）的文档类型。尽管“严格型”这个词听起来有点严厉，其实不是这样的。

**HTML** 文档和 **XHTML** 文档都存在所谓“严格型”和“过渡型”（**transitional**）的文档类型。在我们这个示例中，“严格型”的文档类型指与“过渡型”文档类型相比，其中应尽可能少地包含表现性（**presentational**）的标记。实际上由于你应该使用 **HTML** 来定义文档的结构和含义，并使用 **CSS** 来确定文档如何被呈现，“严格型”的 **HTML** 文档和 **XHTML** 文档中都不应该再有表现性的标记。使用“严格型”的文档类型，将有助于实现这种结构和表现的分离，因为标记校验器会提醒你在你的 **HTML** 代码中是否混入了任何表现性的元素或属性。

## XML 声明

前面我已经说过，在你的 **HTML** 文档的开头部分，需要首先声明文档类型。不过这是一种略微有点简单化的说法，因为 **XML** 声明可以位于文档类型之前。

在一些 **XHTML** 文档中，你可能看到在文档类型声明之前，有类似于下面这样的代码段：

```
<?xml version="1.0" encoding="UTF-8"?>
```

这被称为 **XML** 声明，如果要使用 **XML** 声明，则该声明需要位于文档类型之前。

**Internet Explorer 6.0** 在处理 **XML** 声明时有一个问题，即会用“怪异模式”来显示使用了 **XML** 声明的 **Web** 文档，这当然是文档编写者不希望看到的。

不过值得庆幸的是，除非你真的以 **XML** 格式将 **XHTML** 文档发送给浏览器（参见以上讲述 **XHTML** 部分的附注），并使用 **UTF-8** 字符编码以外的字符编码，且你使用的服务器不发送确定字符编码的 **HTTP** 消息头，你是不需要使用 **XML** 声明的。

由于以上提到的那三种情况同时发生可能性是很小的，因此要解决 **Internet Explorer** 处理 **XML** 声明出现的问题，最简单的办法就是不在文档中添加 **XML** 声明，不过千万不要忘了声明文档类型。

## 总结

在你创建的所有 **HTML** 文档的开头部分，都应该首先声明文档类型，这是你必须牢记的。声明了文档类型，标记校验器就可以知道文档使用的是哪一个版本的 **HTML** 规范，从而能正确地向你报告文档中是否存在任何错误。此外，这也确保所有新式浏览器都以“标准模式”显示你的文档，从而在你用 **CSS** 对文档进行样式化时，在各种浏览器中会得到更具一致性的显示效果。

## 练习题

- 在 **HTML** 文档中都应首先声明文档类型，这样做有两个主要目的，它们分别是什么？
- 与使用“过渡性”文档类型相比，使用“严格型”文档类型有哪些优越之处？
- 为什么使用 **XML** 声明可能造成出现问题？
- 在本篇文章中，我未提及“框架集”(**frameset**) 文档类型，查查它是用来做什么的，并且它为什么不应该被使用。

## 延伸阅读

- [Don't forget to add a doctype](#)
- [Recommended DTDs to use in your Web document.](#)
- [Fix Your Site With the Right DOCTYPE!](#)
- [Activating the Right Layout Mode Using the Doctype Declaration](#)
- [The Opera 9 DOCTYPE Switches](#)
- [Quirks mode and strict mode](#)

- [Transitional vs. Strict Markup](#)
- 上一篇: [HTML 的 <head> 元素](#)
- 下一篇: [在 HTML 中标记文本内容](#)
- [目录](#)

## 作者简介



Roger Johansson 是一名 Web 专业人员，也是 Web 标准、可访问性、可用性的坚定支持者。目前他在瑞典的 Web 咨询公司 NetRelations 公司从事开发 Web 站点的工作，晚上和周末则为他的个人网站 [456 Berea Street](#) 和 [Kaffesnobben](#) 撰写文章。

在未在电脑前工作时，Roger 常常去花园做些园艺活，或者是去野外钓鱼。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

## 15. 在 HTML 中标记文本内容

Posted 12/10/2008 - 17:48 by Lewis

作者: mnfrancis · 2008 年 7 月 8 日

- 上一篇: 为你的 HTML 文档选择适当的文档类型
- 下一篇: HTML 列表
- 目录

### 序言

在本篇文章中，我将向你们讲述以下方面的基础知识：使用 HTML 来描述文档主体内的内容含义。

我们将考察一些常用结构元素如标题、段落，以及嵌入式引用和代码。之后我们将考察内联内容，如简短引用和强调，最后再大致看看已经过时的表现性（presentational）内容。本篇文章的内容目录如下：

- 空格——最后的边界
- 块级元素
  - 页面部分的标题
  - 一般段落
  - 引用其它来源

- 预格式化的文本
- 内联元素
  - 简短引用
  - 强调
  - 斜体文本
- 表现性元素——绝不要使用
- 总结

**附注:** 在每个代码示例后，都有一个“观看实例”的链接，点击该链接将看到源代码在浏览器中实际渲染的效果。这样你们就既可以看到源代码，又可以看到源代码在浏览器中实际渲染的效果了。

### 空格——最后的边界

在开始讲述文本之前，我想先讲述一个重要内容，那就是空格，确切地说，是词语之间的空格。在写 **HTML** 时，源文件中包含被称为“空白（white space）”的用于分隔开文本的字符。一个空格字符（如你用键盘上的空格键打出的空格字符）是最常见的“空白字符”，不过“空白字符”还可包括如文档中两个单独的行之间（被称为回车或新行）的制表符和换行符等。

在 **HTML** 中，多个空白字符（几乎）总是被作为一个空格处理。例如：

```
<h3>In the
beginning</h3>
```

[观看实例](#)

在被浏览器解读时，等同于：

```
<h3>In the beginning</h3>
```

唯一的例外出现在 **pre** 元素中，这在本文的后面部分将详细讲述。

在 **HTML** 中，多个空格被作为一个空格处理，这可能会是第一次编写 **HTML** 文档的新手搞不清楚的地方，这些新手会为了达到所希望的缩进而试图在文本中加入多余的空格，或试图在句子之间的句号后面加入更多的间距并在两个段落间引入更多的垂直空格。这里你们要记住一点，那就是网页的外观应由 **CSS** 样式表控制，而不由 **HTML** 控制，这在以后的课程中将详细讲述。

### 块级元素

在这个部分，我将讲述用于格式化文本的一些常用 块级元素的语法和用法。

## 页面部分的标题

一旦页面被划分为逻辑区段，则每个区段都应有适当的标题。关于这一点，在“一个好的网页需要什么”这篇文章中有进一步的讨论。

HTML 定义有 6 个层次的标题，即 h1, h2, h3, h4, h5, 和 h6。一般而言，h1 将是整个网页的主标题，然后用 h2 表示区段标题，用 h3 表示区段中的小节标题，依次类推。

使用标题层次来描述文档的区段、分区段、分区段下的分区段，这是很重要的，因为这样可以使屏幕阅读器更容易读懂文档，并便于进行自动处理（如 Google 的网页索引抓取程序）。

以本文档为一个模板，标题结构的示例如下：

```
<h1>Marking up textual content in HTML</h1>
<h2>Introduction</h2>
<h2>Space—the final frontier</h2>
<h2>Block level elements</h2>
<h3>Page section headings</h3>
<h3>Generic paragraphs</h3>
<h3>Quoting other sources</h3>
<h3>Preformatted text</h3>
<h2>Inline elements</h2>

[...and so on...]
```

## 观看实例

### 一般段落

段落是绝大多数文档的构造块。在 HTML 中，一个段落由 p 元素表示，该元素无特定的属性。例如：

```
<p>This is a very short paragraph. It only has two sentences.</p>
```

## 观看实例

许多文章和书籍中的一个段落可只包含一个句子。尽管在书写的文章中，“段落”一词的含义是相当清楚的，但在网页上，一些短得多的文本常常也被包裹在段落元素中，这是因为编写网页的人认为这样做比使用一个 div 元素更为“语义化”（我们将在后面的一篇被称为“通用容器”的文章中再讲述这方面的内容）。

一个段落包含一个或多个句子，如同报纸和书籍中的段落一样。在网页上，最好将这种意义上的“段落”用段落元素表示，而不用段落元素来表示页面内那些不成句子的任何文本。那些不成句子的由几个词组成的文本，不应被标记为一个段落。

## 引用其它来源

很多文章、博客文章和参考文档都常常引用其它文章的全部或部分。在 HTML 中，这种较长的引用（如引用完整的句子、段落、列表等），是使用 `blockquote` 元素而被标记的。

一个 `blockquote` 元素不能包含文本，而是必须在其中有另外一个块级元素。你应当使用与被引用的原文档中所使用的块级元素一样的块级元素，例如，如果你引用的是文本的一个段落，则应使用段落元素；如果你引用的是项目列表，则应使用列表元素，以此类推。

如果引用的内容来自另一个网页，你可以使用 `cite` 属性，来指明引用的内容是来自另一个网页，如下所示的那样：

```
<p>HTML 4.01 is the only version of HTML that you should use
when creating a new web page, as, according to the
specification:</p>

<blockquote cite="http://www.w3.org/TR/html401/">

<p>This document obsoletes previous versions of HTML 4.0,
although W3C will continue to make those specifications and
their DTDs available at the W3C Web site.</p>

</blockquote>
```

## 观看实例

如果引语来自小说、杂志或其它形式的离线内容，则不需使用 `cite` 属性。

## 预格式化的文本

任何预格式化的文本和其中的空白字符需要按原样保留的文本都应当使用 `pre` 元素而被标记。

在绝大多数浏览器中，被标记为预格式化的文本将以其在源文件中的原样显示，有时候使用固定宽度（等宽）的字体，给人以文本是用打字机打出来的感觉。为预格式化的文本使用固定宽度的字体，是程序员的习惯做法。

以下这个示例是一个用 `perl` 编程语言写成的一个代码段：

```
<pre><code class="language-perl">

read in the named file in its entirety

sub slurp {

 my $filename = shift;
```

```
my $file = new FileHandle $filename;

if (defined $file) {

 local $/;
 return <$file>;
}

return undef;
};

<code></pre>
```

## 观看实例

关于 `code` 元素的使用，将在后面“罕为人知的语义元素”这篇文章中详细讲述。

## 内联元素

在这个部分，我将讲述用于格式化文本的一些常用 内联元素的语法和用法。

### 简短引用

用于正常的句子或段落之中的简短引用被包含在 `q` 元素之中。与 `blockquote` 元素相似，`q` 元素可包含一个 `cite` 属性，用于指明引用的内容来自互联网上的哪个网页。

一般而言，在被显示时，简短引用都应在引号之内。根据 `HTML` 规范，引号应由“用户代理”插入，这样它们就能被正确地嵌套，并能显示文档中所使用的语言。可用 `CSS` 来控制所使用的引号，这将在后面“样式化文本”的文章中详细讲述。

以下是一个 `q` 元素的示例：

```
<p>This did not end well for me. Oh well,
<q lang="fr">c'est la vie</q> as the French say.</p>
```

## 观看实例

### 强调

`HTML` 中包含有两种指明文档中的文本是需要向用户强调显示的文本（如提示出现错误的消息、警示或通知等）的方法。对浏览器来说，这通常意味着为需要强调显示的文本应用不同的颜色、字体，或用粗体或斜体显示文本。对使用屏幕阅读器的用户来说，可能会以不同的声音或其它听觉效果而知晓这些强调文本。

对需要强调的文本，你使用 `em` 元素，如以下所示的那样：

```
<p>Please note: the kettle is to be unplugged at night.</p>
```

## 观看实例

如果整个句子需要强调，且该句子中还有某一处需要进一步地强调，则你可使用 `strong` 元素来指明需要比一般强调内容更为明显的强调内容，如以下所示的那样：

```
<p>Please note: the kettle must be unplugged every
evening, otherwise it will explode -
killing us all.</p>
```

## 观看实例

### 斜体文本

一般认为，“斜体”不描述文本的含义，因此 `i` 元素就不应该在 `HTML` 中使用（也就是说，`i` 元素和后面要讲的其它一些表现性元素很类似。

其实不能一概而论，在某些情况下，在 `HTML` 中将文本内容描述为“斜体文本”是完全正确的。一些词句如船名、电视连续剧、电影和书籍的名称、一些技术术语或其它分类学名称，最好被描述为“斜体文本”，而不必为它们创建一些仅供它们使用的元素。

有观点认为，以斜体显示一小段文本指明了该小段文本是特别的，而它特别在哪里，则由上下文关系指明。实际上，在目前正在起草的 `HTML 5` 规范中就反映出了这种观点：

`i` 元素表示一小段语态和语气有所不同的文本。只有在没有更适宜的其它元素可用的情况下，才使用 `i` 元素。

由于 `i` 元素可被 `CSS` 重新样式化为非斜体显示，在以上这段引语中，“斜体”基本上是指“文本有点不同”。就我个人的观点而言，我认为这是不能接受的，不过以这样的方式使用“斜体”这个词已有很多的先例。

### 表现性元素——绝不要使用

`HTML` 规范中包含一些被广泛地称为“表现性元素”的一些元素，它们之所以被称为“表现性元素”，是因为它们仅规定包含在它们之中的内容的外观样式，而不是含义。

这些“表现性元素”中的一些，已在 `HTML` 规范中被注明为“不赞成使用的元素”。这就是说，它们已被能到达同样的样式化文本效果的新方法所取代。

在此我将简要介绍一下这些“表现性元素”，不过你们要注意，这只是在回顾以前的 `HTML` 编写方法，你们现在写 `HTML` 文档时绝不应再使用这些“表现性元素”。样式化文本应采用另外的方法，这将在以后的两篇文章即“用 `CSS` 进行文本样式化”和“罕为人知的语义元素”中详细讲

述。

`font face="..." size="..."`

包含在该元素中的文本应由浏览器以与默认字体不同的一种字体渲染。正确的做法是使用 **CSS** 来设置字体。

**b**

包含在该元素之中的文本为粗体，基本上就意味着文本是被强调的文本，因此你们应当使用前面提到的 `em` 元素或 `strong` 元素代替它。

**s 和 strike**

包含在该元素之中的文本为有删除线的文本，如果这只是为了实现一种表现性效果，应使用 **CSS** 来设置。而在文本确实是被标示为已被删除或不需要的文本时，则应使用 `del` 元素来标记文本，这在后面的文章中将详细讲述。

**u**

包含在该元素之中的文本为有下划线的文本，这基本是一种视觉效果，因此正确的做法是使用 **CSS** 来设置有下划线的文本。

**tt**

包含在该元素之中的文本以“打字体”或等宽字体显示，这应该通过使用 **CSS** 来设置，或者使用更为恰当的语义化的元素如 `pre` 元素。

**big 和 small**

包含在该元素之中的文本被以大号字体或小号字体显示，这应该通过使用 **CSS** 来设置。

## 总结

在本篇文章中，我讲述了一些最常用的标记文本内容的 **HTML** 元素。在下一篇文章中，你们将学习如何标记另外一种类型的内容，即项目列表。

- 上一篇：为你的 **HTML** 文档选择适当的文档类型
- 下一篇：**HTML** 列表
- 目录

## 作者简介



Mark Norman Francis 早在万维网诞生前，就在从事互联网领域的作了，一直持续到现在。目前他是全球最大网站 **Yahoo!** 的前端设计师，负责制定 **Web** 开发的最佳习惯、代码标准和质量标准。

在加入 **Yahoo!** 前，他先后在 **Formula One Management** (F-1 管理公司)、**Purple Interactive** (紫色互动公司)、伦敦城市大学从事过多种工作，包括 **Web** 开发、后端 **CGI** 编程和系统架构等。他的博客地址为：<http://marknormanfrancis.com/>。

---

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

## 16. HTML 列表

Posted 12/13/2008 - 12:37 by Lewis

- 网络标准教程

作者: Ben Buchanan · 2008 年 7 月 8 日

- 上一篇: 在 HTML 中标记文本内容
- 下一篇: HTML 中的图像
- 目录

### 序言

列表用于将相关联的信息集合在一起，这样这些相关联的信息就被紧密地相互联系在一起，便于人们阅读。在现代 Web 开发中，列表是广泛使用的元素，频繁地用于导航和一般内容中。

从文档结构上来看，因为使用列表有助于创建出结构良好，更容易访问且易于维护的 Web 文档，因此使用列表是很好的选择。此外列表还为你提供了可附加 CSS 样式的额外元素，有助于应用各种样式。

在本篇文章中，我将讲述以下内容：HTML 中可用的列表的不同类型，应当在什么时候及如何使用列表，以及如何应用一些基本的样式。以下为本篇文章的内容目录（其实，它就是列表！）：

- 三种列表类型
  - 无序列表
    - 无序列表的标记

- 有序列表
  - 有序列表的标记
  - 不以“1”作为有序列表的起始编号
- 定义列表
- 选择列表类型
- HTML 列表和文本之间的差别
- 嵌套列表
- 一个按步骤编写网页的例子
  - 主页的标记
  - 添加一些样式
  - 食谱页面
  - 食谱页面的标记
  - 对食谱页面进行样式化
- 总结
- 延伸阅读
- 练习题

### 三种列表类型

在 HTML 中，有三种列表类型：

- **无序列表**——用于将一组相关的列表项目排列在一起，列表中的项目没有特别的先后顺序。
- **有序列表**——用于将一组相关的列表项目排列在一起，列表中的项目有特别的先后顺序。
- **定义列表**——用于显示名称及其对应的“值”（value），如术语及其定义，或时间及相对应的事件等。

以上每一种类型的列表都有特定的用途和含义，不是可以互换使用的。

### **无序列表**

无序列表，或称符号列表，在列表中的项目可以以任何顺序排列时使用。购物清单就是无序列表的一个例子：

- 牛奶

- 面包
- 黄油
- 咖啡豆

这些项目都是该购物清单的一部分，不过这些项目是可以以任何顺序排列的，如下所示：

- 面包
- 咖啡豆
- 牛奶
- 黄油

你可以使用 **CSS** 将项目符号改为多种默认样式中的一种，或使用你自己选定的图像作为项目符号，甚至可以在显示列表时不使用项目符号，在本篇文章的后面部分我将大致讲述一下具体的做法，更详细的内容将在后面的一篇文章中讲述。

### 无序列表的标记

无序列表使用一组 `<ul></ul>` 标签，标签中包裹有很多组 `<li></li>`：

```

 面包
 咖啡豆
 牛奶
 黄油

```

### 有序列表

有序列表，或称编号列表，在列表中的项目需要以特定的先后顺序排列时使用。食谱就是有序列表的一个例子，其中列出的烹饪步骤是有严格先后顺序的：

1. 准备好原料
2. 将原料混合在一起
3. 将混合好的原料放入烘烤盘
4. 在烤箱中烘烤 1 小时
5. 从烤箱中取出
6. 放置 10 分钟
7. 端上餐桌

如果将该列表中的项目以不同的顺序排列，这个食谱就混乱了，不再有用：

1. 准备好原料
2. 在烤箱中烘烤 1 小时
3. 从烤箱中取出
4. 端上餐桌
5. 将混合好的原料放入烘烤盘
6. 放置 10 分钟
7. 将原料混合在一起

在显示有序列表时，可以用多种数字或字母作为项目编号。在多数浏览器中，默认的有序列表中的项目编号为十进制的数字，不过还有更多的可选项目编号：

- 字母
  - 小写的 ascii 字母 (a, b, c...)
  - 大写的 ascii 字母 (A, B, C...)
  - 小写的古典希腊字母 (α, β, γ...)
- 数字
  - 十进制数字 (1, 2, 3...)
  - 前面加有 0 的十进制数字 (01, 02, 03...)
  - 小写的罗马数字 (i, ii, iii...)
  - 大写的罗马数字 (I, II, III...)
  - 传统的格鲁吉亚数字编号 (an, ban, gan...)
  - 传统的亚美尼亚数字编号 (mek, yerku, yerek...)

同样地，你也可以使用 CSS 来改变有序列表的样式。

## 有序列表的标记

有序列表使用一组 `<ol></ol>` 标签，标签中包裹有很多组 `<li></li>`：

```

 准备好原料
 将原料混合在一起
 将混合好的原料放入烘烤盘
 在烤箱中烘烤 1 小时
 从烤箱中取出
```

```
放置 10 分钟
端上餐桌

```

## 不以“1”作为有序列表的起始编号

通过使用带有一个数字属性值的 `start` 属性，可以让有序列表不以“1”（或 `i`, `I` 等）为起始编号，即使你已使用 CSS 的 `list-style-type` 属性，将有序列表中的编号改为字母或罗马数字也同样可以达到这个效果。如果你有想在一个有序列表中加入一些注释或其它相关信息，就可使用这种方法。以上面那个食谱为例，我们可以将其改为：

```

 准备好原料
 将原料混合在一起
 将混合好的原料放入烘烤盘

<p class="article_note">在你将原料放入烘烤盘之前，请先把烤箱预热到 180 摄氏度或 350 华氏度。</p>

<ol start="4">
 在烤箱中烘烤 1 小时
 从烤箱中取出
 放置 10 分钟
 端上餐桌

```

现在食谱就变成如下的样子：

1. 准备好原料
2. 将原料混合在一起
3. 将混合好的原料放入烘烤盘

在你将原料放入烘烤盘之前，请先把烤箱预热到 180 摄氏度或 350 华氏度。

4. 在烤箱中烘烤 1 小时
5. 从烤箱中取出
6. 放置 10 分钟
7. 端上餐桌

附注：在最近版本的 HTML 规范中，这个 `start` 属性已被标注为不赞成使用的属性，这意味着你如果将你的文档类型定义为“严格型”，在其中使用这个属性的话，万维网联盟（W3C）的标记校验器就会提示你文档中存在 HTML 错误。这显得有点奇怪，因为这个属性其实很有用的，而且 CSS 中也没有相应的属性。这说明对 HTML 进行校验是个追求的理想目标，但并不总是完全绝对的和最终的。此外，在正在起草中的 HTML5 规范中，`start` 属性已不再被标注为不赞成使用的属性（参见 [HTML5 和 HTML4 之间的差异这篇文章](#)）。如果你想在文档类型被定义为“严格型”的 HTML 文档中使用这个 `start` 属性，且需完全通过校验，那你可以使用 CSS 计数器来作为代替。

## 定义列表

定义列表将列表中的项目与其定义配对显示。例如，如果你想为购物清单中的项目给出定义，你可以像以下显示的那样做：

牛奶

一种白色的液体乳制品。

面包

一种用面粉或玉米粉烘焙而成的食品。

黄油

一种黄色的固体乳制品。

咖啡豆

咖啡树果实中的种子。

每个术语及其定义是一个定义组（或称“名称—值”组）。你可以在定义列表中列出任意数量的定义组，但每个定义组中必须至少有一个术语及一个定义。也就是说，术语必须要有至少一个相对应的定义，定义必须要有至少一个相对应的术语。

你可以将多个术语与一个定义相配对，也可以将多个定义与一个术语相配对。例如，“咖啡”这个术语可以有几种含义，你可以将这几种含义都显示在“咖啡”这个术语项下：

咖啡

一种由炒熟的咖啡豆制成的饮料

一杯咖啡

供应咖啡的一种社交聚会

咖啡色

另一方面，你也可以将多个术语与一个定义相配对。这在显示具有相同含义的一个术语的几个变体时很有用：

苏打水

汽水

充气饮料

可乐

一种甜的碳酸饮料。

定义列表与其它类型的列表不同，它用定义术语和定义描述来替代列表项目。

因此，定义列表使用一组 `<dl></dl>` 元素，元素中包裹有很多组 `<dt></dt>` 和 `<dd></dd>` 标签。你必须将至少一组 `<dt></dt>` 标签与至少一组 `<dd></dd>` 相配对；在源文件中，`<dt></dt>` 标签应当始终位于 `<dd></dd>` 标签之前。

以下是一个简单的定义列表（其中一个术语只有一个定义）的代码示例：

```
<dl>
 <dt>术语</dt>
 <dd>术语的定义</dd>
 <dt>术语</dt>
 <dd>术语的定义</dd>
 <dt>术语</dt>
 <dd>术语的定义</dd>
</dl>
```

其实际渲染出来的样子如下：

术语

术语的定义

术语

术语的定义

术语

术语的定义

在以下这个示例中，我们将多个术语与一个定义相配对，并将多个定义与一个术语相配对，

代码如下：

```
<dl>
 <dt>术语</dt>
 <dd>术语的定义</dd>
 <dt>术语</dt>
 <dt>术语</dt>
 <dd>适用于以上两个术语的定义</dd>
```

```
<dt>具有以下两个定义的术语</dt>
<dd>术语的第一个定义</dd>
<dd>术语的第二个定义</dd>
</dl>
```

其实际渲染出来的样子如下：

术语

    术语的定义

术语

术语

    适用于以上两个术语的定义

具有以下两个定义的术语

    术语的第一个定义

    术语的第二个定义

一般而言，将多个术语与一个定义相配对是很少见的，不过知道可以这样做，肯定会有帮助的。

### **选择列表类型**

在决定使用哪种类型的列表时，你通常只需要问自己两个简单的问题即可：

1. 我是在定义术语吗（或是在将名称/值相配对吗）？
  - 如果是，则使用定义列表。
  - 如果不是，则不使用定义列表，然后继续问下一个问题。
2. 列表中项目排列的先后次序重要吗？
  - 如果是，则使用有序列表。
  - 如果不是，则使用无序列表。

### **HTML 列表和文本之间的差别**

你们也许还不清楚 **HTML** 列表和一些前面带有以手工方式写入的项目符号或编号的文本之间到底有什么差别。差别肯定是存在的，使用 **HTML** 列表有以下几个优点：

- 如果你必须在一个有序列表中更改列表项目的顺序，你只需在 **HTML** 中将顺序进行调整即可。而如果你是以手工方式写入项目的编号，则你不得不通篇检查并更改每一个项目的编号，想想这是多烦人的工作。

- 使用 **HTML** 列表可以让你以适当的方式对列表进行样式化。如果你只是使用混杂有项目符号或编号的文本，你会发现单个项目进行样式化是更为困难的工作。
- 使用 **HTML** 列表为内容赋予了适当的语义化的结构，并不只是创造出列表的视觉效果。这有一些显著的优点，如能让屏幕阅读器告诉那些视力受损的用户，正在朗读的内容是一个列表，而不是混杂有项目符号或编号的文本。

换句话说，**文本和列表是有差别的**。使用文本来替代列表不仅会让你做更多的工作，还可能会影响阅读你的文档的人造成阅读障碍。因此当你的文档需要一个列表时，你应当使用正确的 **HTML** 列表。

### 嵌套列表

一个列表项目可包含另一个完整的列表，这被称为“嵌套”列表。像目录（如本文开头部分列出的内容目录）这些一般就需要使用嵌套列表：

1. 第一章
  1. 第一节
  2. 第二节
  3. 第三节
2. 第二章
3. 第三章

在使用嵌套列表时，有一个关键之处必须牢记，那就是嵌套列表必须和一个特定的列表项目相联系。为在代码中反映这种联系，嵌套列表是要包含在那个列表项目之中的。以上那个目录的源代码如下：

```

 第一章

 第一节
 第二节
 第三节

 第二章
 第三章

```

请注意，嵌套列表在 `<li>` 及包含列表项目的文本（“第一章”）之后起始，在包含有列表

项目的 `</li>` 处结束。由于嵌套列表是定义网站结构的一个很好的方式，它们通常构成网站导航菜单的基础。

从理论上讲，你可以嵌套你希望的任何数量的列表，不过叠加层次过多的嵌套列表可能会造成混乱。对一个很大的列表，你最好还是将其内容拆分到数个带有标题的列表之中，甚至还可以拆分为不同的网页。

### 一个按步骤编写网页的例子

以下我们来看一个运用你们已经学到的知识按步骤编写网页的例子。请考虑以下这个情景状况：

你正在为 **HTML** 烹饪学校构建一个小网站。在网站的主页上，将要显示一个分类食谱的列表，由与食谱页面相链接的列表项目组成。每个食谱页面都会列出所需的原料、原料的说明及制作方法。三个大类的食品分别是“蛋糕”（包括“松糕”、“巧克力蛋糕”、“苹果茶蛋糕”的制作食谱）、“饼干”（包括“澳式饼干”、“果酱饼干”和“速融饼干”的制作食谱），以及“速食面包”（包括“丹波面包”和“烤饼”的制作食谱）。客户对列表中的食品大类和食谱的排列先后顺序无特别要求，仅要求能让访问网站的人明确知道：哪些列表项目是食品大类，哪些又是食谱。

现在我们就开始按步骤地构建这个网站。在本部分，我将讲述如何写 **HTML** 标记，以及如何向列表添加一些样式。关于对 **HTML** 文档进行样式化的内容，我这里不做详细地讲述，在本课程后面的文章中将详细地讲述这方面的内容。

#### 主页的标记

- 创建一个包含有文档类型、`html` 元素、`head` 元素、`body` 元素的 **HTML** 文档，将其保存为 `stepbystep-main.html`。添加主标题 (`h1`)，即“HTML 烹饪学校”，以及副标题 (`h2`)，即“食谱”：

```
<h1>HTML 烹饪学校</h1>
```

```
<h2>食谱</h2>
```

- 有三个食谱大类需要显示，而它们排列的先后顺序并不重要，因此最适于使用无序列表，请在文档中加入以下代码：

```
<h2>食谱</h2>

 蛋糕
 饼干
 速食面包
```

```

```

将 `li` 元素缩进了一点只是为了让代码更易读一些，不是一定需要缩进。

3. 现在你需要添加作为子项目的食谱，如“松糕”、“巧克力蛋糕”、“苹果茶蛋糕”都是“蛋糕”这个大类下的子项目。为此你需要在每个大类项目内创建一个嵌套列表。由于子项目排列的前后顺序也不重要，因此适于创建一个嵌套的无序列表。为简便起见，我将让你们把所有这些食谱列表项目，都链接到一个食谱页面（第 18 章将详细讲述 HTML 链接，访问这个链接，你可以读到关于“链接”的内容）：

```
<h2>食谱</h2>

蛋糕

松糕
巧克力蛋糕
苹果茶蛋糕

饼干

澳式饼干
果酱饼干
速融饼干

面包/速食面包

丹波面包
烤饼


```

### 添加一些样式

客户认可这种列表安排，但希望食谱大类前都有一个箭头，来替代项目符号。同时还希望食谱大类在页面上靠左对齐。为此你需要指定一个图像来替代项目符号，然后调整边距/填充距设置。

1. 为避免与网站中的其它列表有抵触，你应该为这个列表添加一个“类”，这样你就能在你的样式表中指定特别的上下文选择符。这个“类”被命名为“`recipe-list`”是比较合适的。

```
<h2>食谱</h2>

<ul class="recipe-list">
```

2. 现在你需要创建一个样式表，并为它添加一些样式规则。首先在你的文档的 `head` 部分加入起始和结束的 `style` 标签。
3. 现在将从列表中移除间距。由于按默认设置，一些浏览器为空间元素使用的是 `margin` 属性，而另一些浏览器为空间元素使用的是 `padding` 属性，因此你需要把这两个属性的值都设为 0。在你的 `style` 标签中加入以下 CSS 代码：

```
ul.recipe-list {
 margin-left: 0;
 padding-left: 0;
}
```

4. 然后创建一个自定义的项目符号图像，如果你喜欢的话，可以使用我创建的图像（如图 1 所示）。



图 1：自定义的项目符号图像

5. 现在你将从列表中移除项目符号，将项目符号设为列表项目的背景图像。你需要加入一些填充距，这样文字就不会被摆在背景图像的上方。为此你可以在结束 `style` 标签前加入以下 CSS 代码：

```
ul.recipe-list li {
 list-style-type: none;
 background: #fff url("example-bullet.gif") 0 0.4em no-repeat;
 padding-left: 10px;
}
```

6. 最后，你将把项目符号放回到嵌套列表中的项目之上，并将背景设为纯白色（第二条规则更为具体，因此将替代那条背景图像规则）。请记住，第一条 CSS 规则将被嵌套列表继承，因此你需要撤销所有的容器设置。在结束 `style` 标签前加入以下 CSS 代码：

```
ul.recipe-list li li {
 list-style-type: disc;
 background: #fff;
}
```

页面的最后显示效果应该如图 2 所示的那样：

# HTML 烹饪学校

## 食谱

### > 蛋糕

- [松糕](#)
- [巧克力蛋糕](#)
- [苹果茶蛋糕](#)

### > 饼干

- [澳式饼干](#)
- [果酱饼干](#)
- [速融饼干](#)

### > 面包/速食面包

- [丹波面包](#)
- [烤饼](#)

图2：完成的主页，其中包含自定义的项目符号图像

你也可以 [点击这里观看实例](#)（译注，英文原版示例）。

## 食谱页面

以下我只创建了“松糕”的食谱页面，作为一个示例。你们可以以这个示例页面为模板，自己创建其它食谱页面。客户已用文本形式向你提供了“松糕”的食谱，如下所示（译注，松糕的食谱内容未译）：

Simple Sponge Cake

Ingredients

3 eggs

100g castor sugar

85g self-raising flour

Notes on ingredients:

Caster Sugar - Finely granulated white sugar.

Self-raising flour - A pre-mixed combination of flour and leavening agents (usually salt and baking powder).

Method

1. Preheat the oven to 190°C.
2. Grease a 20cm round cake pan.
3. In a medium bowl, whip together the eggs and castor sugar until fluffy.
4. Fold in flour.
5. Pour mixture into the prepared pan.

6. Bake for 20 minutes in the preheated oven, or until the top of the cake springs back when lightly pressed.
7. Cool in the pan over a wire rack.

## 食谱页面的标记

1. 创建一个 HTML 文档，将其保存为 `stepbystep-recipe.html`。在文档中添加以下标题：

```
<h1>Simple Sponge Cake</h1>
<h2>Ingredients</h2>
<h3>Notes on ingredients</h3>
<h2>Method</h2>
```

2. 原料列表中有几个列表项目，它们排列的先后顺序不重要，因此适于使用一个无序列表。将下列代码加入你的 HTML 文档的 `body` 部分：

```
<h2>Ingredients</h2>

3 eggs
100g castor sugar
85g self-raising flour

```

3. 原料说明用于恰当地定义一些原料究竟是什么。你需要将“原料”这个术语与其定义相配对。这里适于使用一个定义列表。将下列代码加入 HTML 文档中，放在上一个步骤所加入的无序列表之下：

```
<h3>Notes on ingredients</h3>
<dl>
<dt>Castor Sugar</dt>
<dd>Finely granulated white sugar.</dd>
<dt>Self-raising flour</dt>
<dd>A pre-mixed combination of flour and leavening agents (usually salt and baking powder).</dd>
</dl>
```

4. 很明显，蛋糕的制作必须依照一个正确的先后顺序进行，因此蛋糕的制作方法必须使用一个有序列表。将下列代码加入 HTML 文档中，放在定义列表之下：

```
<h2>Method</h2>

Preheat the oven to 190°C.
Grease a 20cm round cake pan.
```

```
In a medium bowl, whip together the eggs and castor sugar until fluffy.
Fold in flour.
Pour mixture into the prepared pan.
Bake for 20 minutes in the preheated oven, or until the top of the cake springs
back when lightly pressed.
Cool in the pan over a wire rack.

```

## 对食谱页面进行样式化

客户认可这种列表安排，但希望原料的定义以粗体显示，从而能增强易读性。在你的 HTML 文档的 `head` 部分加入下列代码：

```
<style>
dt {
 font-weight: bold;
}
</style>
```

页面的最后显示效果应该如图 3 所示的那样：

# Simple Sponge Cake

## Ingredients

- 3 eggs
- 100g castor sugar
- 85g self-raising flour

## Notes on ingredients

### Castor Sugar

Finely granulated white sugar.

### Self-raising flour

A pre-mixed combination of flour and leavening agents (usually salt and baking powder).

## Method

1. Preheat the oven to 190°C.
2. Grease a 20cm round cake pan.
3. In a medium bowl, whip together the eggs and castor sugar until fluffy.
4. Fold in flour.
5. Pour mixture into the prepared pan.
6. Bake for 20 minutes in the preheated oven, or until the top of the cake springs back when lightly pressed.
7. Cool in the pan over a wire rack.

图 3：完成的食谱页面，其中原料定义以粗体字显示

你也可以 [点击这里](#) 观看页面的实际显示效果。

现在这两个页面就都已创建完成了！

## 总结

在学完本篇文章后，你应该已经清楚地了解了 HTML 中的三种列表类型。使用一个按步骤编写网页的例子，你应该已经创建出这三种类型的列表，并已学会如何为列表项目加入嵌套列表。

在你知道如何恰当地使用 HTML 列表后，你可能会发现在编写网页时随时都可能使用这些 HTML 列表。网页上很多本应放置在列表中的内容，却只是被包裹在某一个通用元素中，并被加上一些断行标签。这是一个懒惰的工作习惯，其造成的问题远比其解决的问题要多，因此你一定不要这样做！你应当始终创建语义正确的列表，以帮助人们阅读你网站上的内容。这样做是一个良好的习惯，尤其是在你以后需要维护你的网站时更是如此。

## 延伸阅读

- [A List Apart: Taming Lists](#)
- [W3C CSS2: list-style-type definition](#)

## 练习题

- 三种类型的 **HTML** 列表分别是什么？
- 每一种类型的 **HTML** 列表分别将在什么情况下使用？你将如何选择 **HTML** 列表类型？
- 你如何嵌套入列表？
- 为什么你应当使用 **CSS**, 而不是 **HTML** 来对你的列表进行样式化？
- 上一篇：在 **HTML** 中标记文本内容
- 下一篇：**HTML** 中的图像
- 目录

## 作者简介



Ben Buchanan 10 多年前就开始创作网页，那时他就读的学位课程其实并不是 IT 课程。他在公共机构（大学）和私营部门都曾工作过。他曾参与数家著名网站的改版，包括 澳大利亚人报的网站（[The Australian](#)）和前后三个版本的 格里菲斯大学的官方网站。目前他在 新闻数字媒体公司（[News Digital Media](#)）任前端架构师。他的博客地址为：[the 200ok weblog](#)。

---

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

# 17. HTML 中的图像

Posted 12/15/2008 - 16:51 by Lewis

- 网络标准教程

作者: Christian Heilmann · 2008 年 7 月 8 日

- 上一篇: [HTML 列表](#)
- 下一篇: [HTML 链接——让我们建一张网吧！](#)
- [目录](#)

## 序言

在本篇文章中，我将讲述美化 Web 页面的要素之一，即图像。在学习完本篇文章后，你将了解到如何以可访问的方式向 Web 文档中加入图像（这里的可访问指视力受损的人也可以使用你的网站中的信息），以及什么时候及如何在页面布局中使用内联图像或背景图像。你可以点击[这里](#)下载本篇文章中使用的示例图像，在本篇文章中我将引用这些图像文件。本篇文章的内容目录如下：

- 一张图片胜过千字描述，是这样的吗？
- 网页上不同类型的图像——内容图像和背景图像
- `img` 元素及其属性
  - 使用 `alt` 属性为图像提供替代文本
  - 使用 `title` 属性为图像添加附加信息
  - 使用 `longdesc` 属性为复杂图像提供一个替代文件
  - 使用 `width` 和 `height` 属性来定义图像尺寸，以便图像被更快地显示
- 内联图像的内容就到这里
- 使用 CSS 控制背景图像
  - 如何通过 CSS 来应用背景图像
- 总结
- 练习题

### 一张图片胜过千字描述，是这样的吗？

你可能很想在你的网站上使用很多图像。的确，图像能很好地让访问者直观地感受网站的

整体风格，同时图像和插图也是帮助视觉学习者更容易地理解复杂信息的好方式。

但在网站上放置很多图像，也有其缺点，那就是不是每个在网上浏览网站的人都可以看到这些图像。回想一下浏览器刚能支持网页上的图像时，许多访问网站的人为节省带宽和获得更快的网页浏览速度（那时候网速非常慢，而且上网的费用非常高，按分钟计费），都选择关闭浏览器显示图像的功能。虽然现在这种情况已不常见了，但并不是就完全不存在了，在网页中使用图像可能面临的问题包括：

- 由于显示屏幕小及下载数据的费用较高，用移动设备浏览网页的人也许依然会关闭浏览器显示图像的功能。
- 访问网站的人可能是盲人或视力受损而无法正常地看到网页上图像的人。
- 一些访问者可能是来自另一种文化背景的人，无法理解你在网页上使用的图标。
- 目前搜索引擎还仅把文本编入索引，而并不分析图像，这意味着储存在图像里的信息尚不能被搜索引擎发现和编入索引。

因此，为你的网站精心挑选图像并只在确实需要时才使用它们，这是很重要的。更为重要的是，你应确保总是为那些不能看到图像的访问者提供了替代性的选择。在“网页导航栏和菜单”上不正确地使用图标和图像，会造成很多问题，这将在后面的文章中讨论。现在，先让我们看看向 **HTML** 文档中加入图像有哪些可用技术。

### **网页上不同类型的图像——内容图像和背景图像**

向 **Web** 文档中添加图像，主要有两种方法：使用 **img** 元素来添加内容图像和使用 **CSS** 来为元素应用背景图像，具体使用哪一种方法要取决于你加入图像的目的：

- 如果图像对文档的内容而言是至关重要的，例如作者的一张照片或一个显示一些数据的图形，则该等图像应该被作为一个 **img** 元素而加入文档，同时带有适当的替代文本。
- 如果图像是仅起美化和装饰网页作用的图像，则你应该使用 **CSS** 背景图像。原因在于这些图像不必带有任何替代文本（为此类图像加入这样的替代文本“一个绿色的圆形”，对盲人访问者而言有什么用呢？），而且你在 **CSS** 中进行图像的样式化，比在 **HTML** 中这样做要多得多的选择。

### **img 元素及其属性**

通过使用 **img** 元素，可以很容易地向 **HTML** 文档中添加图像。下面的这个 **HTML** 文档（即下载的 **zip** 文件中的 **inlineimageexample.html**），就加入了一张名为 **balconyview.jpg** 的图片，可在浏览器中显示（但前提条件为在你保存 **HTML** 文档的文件夹里保存有这张图片）。

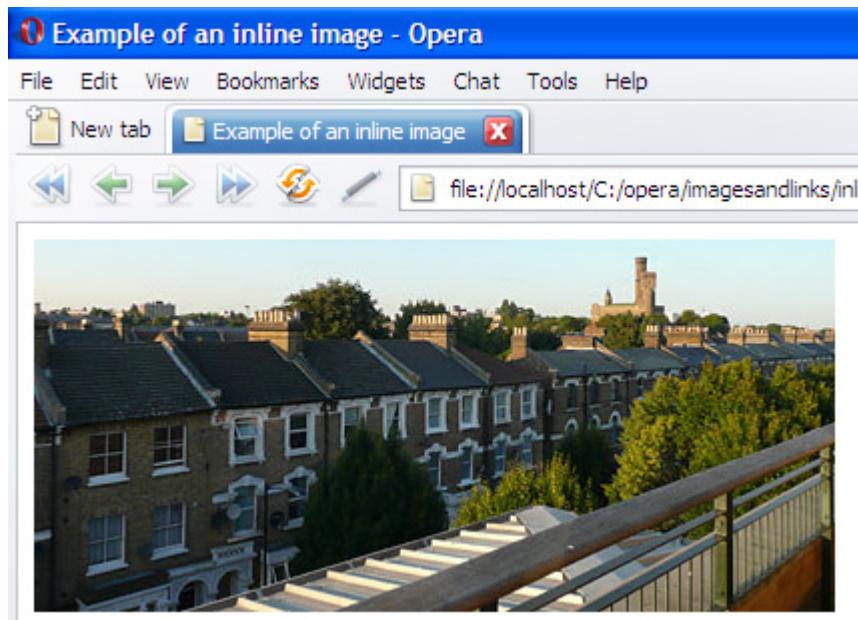
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<title>Example of an inline image</title>
</head>
<body>

</body>
</html>
```

如果你在浏览器中运行这段代码，其显示出来的样子如图 1 所示。

图 1：图片在浏览器中显示出来的样子



#### 使用 alt 属性为图像提供替代文本

以上那段 HTML 代码中插入了一个能在浏览器中正常显示的图像，但由于 img 元素都需要一个 alt 属性，因此它是一段有问题的 HTML 代码。这个属性包含在由于某些原因而使图像不可用时，所显示的替代文本。造成图像不可用的原因包括：无法找到图像、无法载入或“用户代理”（一般指浏览器）不支持图像显示。此外，对靠辅助技术为他们朗读网页内容的盲人来说，辅助技术是把 img 元素的 alt 属性中包含的替代文本读给他们听。因此，写出描述图像内容的适当替代文本并将它们放入 alt 属性中，这是很重要的。

在网上你们可以找到很多讨论“`alt` 标签”的文章。“`alt` 标签”这种说法实际上是错误的，因为它并不是一个标签（或元素），而是 `img` 元素的一个属性，它对网页的可访问性和搜索引擎优化而言都是非常重要的。

为让网页上的图像能为所有人都理解，你需要为图像添加适当的替代文本，例如，对以上那个图像就可以加上这样的替代文本：“`View from my balcony, showing a row of houses, trees and a castle`”（译注，从阳台上看到的景色，包括一排房子、绿树和一个城堡）（[inlineimageexamplealt.html](#)）：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
 <head>
 <title>Example of an inline image</title>
 </head>
 <body>

 </body>
</html>
```

`alt` 属性包含在图像不可用时应该显示的文本。在图像成功载入和显示时，`alt` 属性中包含的信息就不应显示。在这方面，**Internet Explorer** 浏览器就存在一个错误，即当你把鼠标指针悬停在一个图像上时，会显示一个工具提示。这的确是一个错误，因为它导致很多人在 `alt` 属性中加入关于图像的附加信息。如果你想为图像加入附加信息，应该使用 `title` 属性，而不是 `alt` 属性。关于 `title` 属性，我将在下面的部分讲述。

### 使用 `title` 属性为图像添加附加信息

当你把鼠标光标悬停在一个图像上时，绝大多数浏览器都会将 `img` 元素的 `title` 属性值显示为一个工具提示（如图 2 所示）。这可以帮助访问者了解图像的附加信息，但你不能假定每个访问者就一定有一个鼠标。`title` 属性虽然可以很有用，但它并不是提供关于图像的关键信息的最保险的方式。实际上 `title` 属性是为图像添加如：心情，或上下文意思等信息的好方式（[inlineimagewithtitle.html](#)）：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```

<html>
 <head>
 <title>Example of an inline image with alternative text and title</title>
 </head>
 <body>

 </body>
</html>

```

如果你在浏览器中载入以上的代码，会看到如图 2 所示的显示画面。

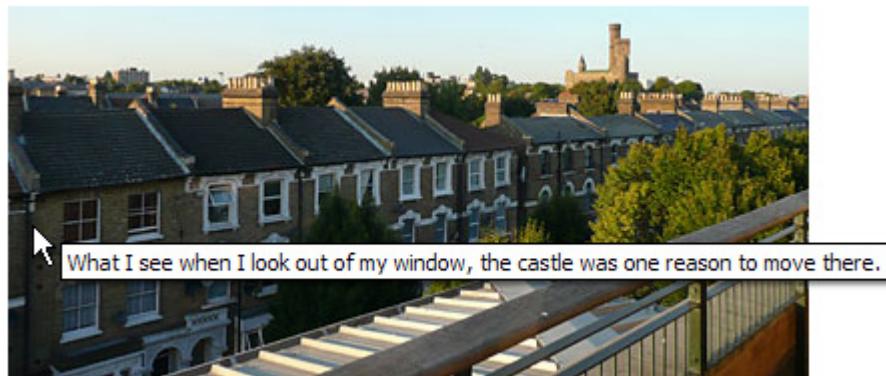


图2：在绝大多数浏览器中 `title` 属性被显示为工具提示

#### 使用 `longdesc` 属性为复杂图像提供一个替代文件

如果图像是一个非常复杂的图像，如一个图表，你可以使用 `longdesc` 属性为图像提供更长篇幅的描述，这样那些使用屏幕阅读器访问网页的人或在浏览器中关闭图像显示功能的人，也依然可以了解到图像所传达的信息。

这个属性包含一个指向包含同样信息的文档的 `URL` 地址。例如，如果你在网页上放置了一个显示一组信息的图表，你就可以使用 `longdesc` 属性，将该图表与一个数据表相链接（[inlineimagelongdesc.html](#)）：

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
 <head>

```

```

<title>Example of an inline image with longdesc</title>

</head>

<body>

</body>

</html>

```

数据文件 `fruitconsumption.html` 包含一个表述相同信息的非常简单的数据表:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<title>Fruit consumption</title>

</head>

<body>

<table summary="Fruit Consumption of under 15 year olds, March 2007">

<caption>Fruit Consumption</caption>

<thead>

<tr><th scope="col">Fruit</th><th scope="col">Amount</th></tr>

</thead>

<tbody>

<tr><td>Apples</td><td>10</td></tr>

<tr><td>Oranges</td><td>58</td></tr>

<tr><td>Pineapples</td><td>95</td></tr>

<tr><td>Bananas</td><td>30</td></tr>

<tr><td>Raisins</td><td>8</td></tr>

<tr><td>Pears</td><td>63</td></tr>

</tbody>

</table>

```

```
<p>Back to article</p>

</body>

</html>
```

这两种不同的数据表示方式的显示效果如图 3 所示。

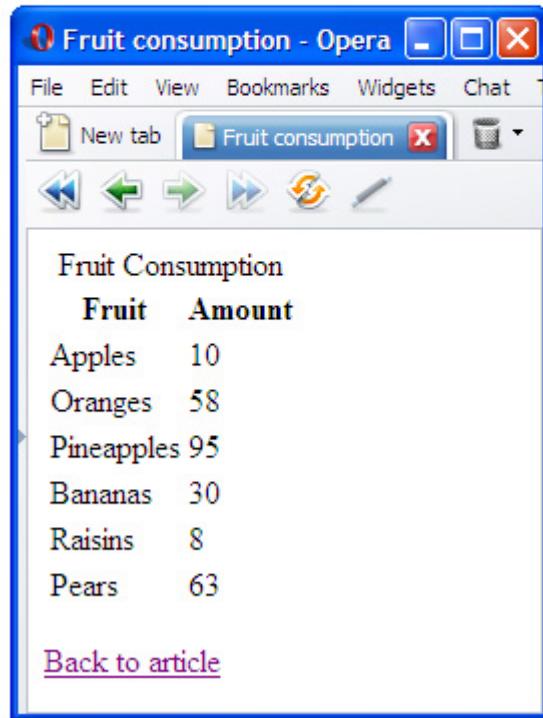
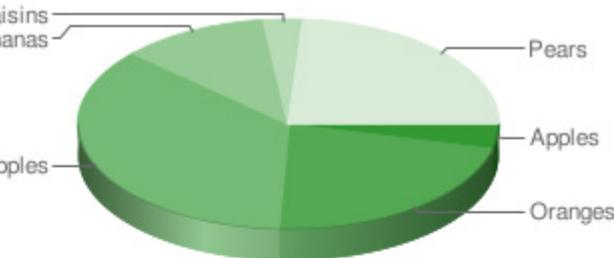


图 3：你可以使用 `longdesc` 属性将一个包含复杂数据的文档与一个图像相链接

请注意，在页面上是无法看到有一个与图像相链接的长描述文件的，但辅助技术设备是可以让其用户知道有一个可用的替代图像文件的。

使用 `width` 和 `height` 属性来定义图像尺寸，以便图像被更快地显示

当“用户代理”发现 HTML 中的一个 `img` 元素时，它就开始载入 `src` 属性所指向的图像。在默认状态下，“用户代理”并不知道图像的尺寸，因此它将仅显示混杂在一起的文本，而在图像完全载入和显示时，才将页面显示全。这样就可能使页面的载入速度变慢，并可能给访问者造成不便。为避免出现这种情况，你可以通过使用 `width` 属性和 `height` 属性为图像设置尺寸，以

告诉“用户代理”在载入图像前为图像分配适当大小的空间 ([inlineimagewithdimensions.html](#)):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<title>Example of an inline image with dimensions</title>
</head>
<body>

</body>
</html>
```

这样“用户代理”将根据图像的尺寸显示一个预留的图像框，直到图像被完全载入进这个图像框，因此就避免了不好看的页面移动。你也可以使用 `width` 和 `height` 属性来缩放图像（你可以试着将上面例子中的两个属性值分别减半并保存，然后重新载入页面），但这不是一个好的做法，因为在所有的浏览器中，图像缩放的质量并不都是平滑的。将图像缩小成缩略图，尤其不是好做法。因为使用缩略图，你不仅在物理大小上获得了更小的图像，同时其文件体积也变小了。但没有人愿意下载一个 **300KB** 的照片，而仅仅为了看本该 **5KB** 的小图像。

### 内联图像的内容就到这里

还有很多图像属性可供你使用，不过它们中的大多数已被标明为“不赞成使用”的属性，因为它们仅是定义图像摆置和对齐方式的属性。这不应使用 `HTML`，而应使用 `CSS` 来控制。有一点是你们必须牢记的，那就是图像就其默认状态来说是内联元素。这意味着它们可以在文本的字词之间出现，并非一定要在新行上显示。这样，如果你想在文本正文中加入一些小图标的话，的确很方便，但当你试图使用图像和文本来创作页面布局时，就可能会很麻烦。使用 `CSS`，你可以废除图像是内联元素这一默认性质，而让图像如块级元素一样显示（块级元素指被加入文档中时，在新行上显示的元素）。

### **使用 CSS 控制背景图像**

可以很有把握地说，从浏览器开始支持 `CSS` 起，`Web` 设计就变得更为轻松和有趣。有了 `CSS`，我们就不再需要费力地使用表格单元格来定位页面元素，及使用不间断空格 (`&nbsp;`) 来保持空格，也不再需要使用空白 `GIF`（指透明的  $1 \times 1$  像素的 `GIF` 图像，被缩放后用于创建外边距），而只需使用 `CSS` 中的填充距、外边距、尺寸及定位属性等，让 `HTML` 仅控制内容

的结构即可。

使用 CSS，还意味着你能以非常多的方式使用背景图像，以你希望的任何方式将背景图像定位在文本后面及周围，并能使用以一定方式重复的图像来创建背景。这里我将只简单地介绍一下 CSS 图像，后面的文章将详细地讲述关于 CSS 背景图像的内容。

### 如何通过 CSS 来应用背景图像

通过 CSS 来应用背景图像是很容易的。在你查看下面显示的 CSS 代码前，请在浏览器中载入 `imagesandcss.html` 这个示例文件，或查看图 4，以大致了解 CSS 中各种可能有的背景图像的属性。

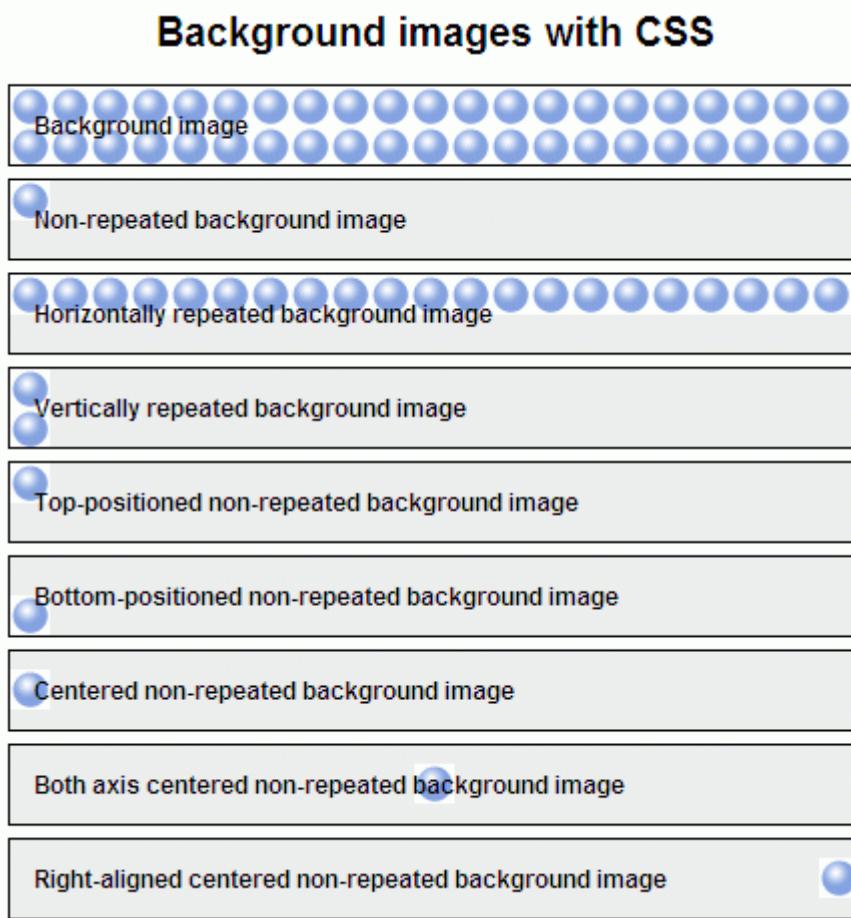


图 4: 通过 CSS 应用背景图像

上图中不同的盒子实际上是已被样式化的 `h2` 标题元素，通过 CSS 应用了一些填充距和边框属性，为显示背景图像提供足够的空间。如果你检查 HTML 源文件，你会发现每个 `h2` 元素都有一个独一无二的 `id`，这样每个 `h2` 元素就都可以应用一种不相同的 CSS 规则。以上第一个例子中的 CSS 代码如下：

```
background-image:url(ball.gif);
```

```
background-color:#eee;
```

你使用 `background-image` 选择符来添加图像，并赋予它一个带括弧的 **URL** 地址，以指定将要加入的图像。你同时应使用 `background-color` 选择符和颜色值（可以是十六进制的颜色代码、颜色名称或 **RGB** 颜色），提供一种背景色（此处我选用的是浅灰色），以作为图像不可用时的替代性的选择。

在默认状态下，背景图像将既水平重复又垂直重复，来填满整个元素空间。不过你可以使用 `background-repeat` 选择符来定义不同的重复方式：

- 背景图像完全不重复：`background-repeat: no-repeat;`
- 仅水平重复：`background-repeat: repeat-x;`
- 仅垂直重复：`background-repeat: repeat-y;`

在默认状态下，背景图像（如果不重复的话）将被定位在元素的左上角。不过你可以使用 `background-position` 来改变背景图像的位置。垂直对齐可选择的属性值包括 `top`、`center` 和 `bottom`，而水平对齐可选择的属性值包括 `left`、`center` 和 `right`。例如，要将图像定位在右下角，你需要使用 `background-position: bottom-right;`，而要将图像垂直居中，然后靠右对齐，你需要使用 `background-position: center-right;`。

通过使用 **CSS** 来控制背景图像的重复方式和位置并使用适当的图像，你可以创造出许多在 **CSS** 出现前不可能创造出的神奇效果。同时，通过将背景图像的定义放置在单独的 **CSS** 文件中，你可以仅修改数行 **CSS** 代码，就很容易地改变整个网站的外观和风格。在本教程第 30 篇文章中，将详细地讲述这方面的内容。

## 总结

本篇文章讲述了你在向 **HTML** 文档中添加图像时需要了解的知识和方法。关于使用图像和 **CSS**，还有很多可用的技巧和诀窍，不过你们现在首先需要掌握本篇文章中所讲述的应用图像的最佳习惯。在本篇文章中，我们讲述了：

- `img` 元素及其基本属性：
  - `src` 属性，用于定义图像的文件位置
  - `alt` 属性，用于显示在图像未载入或不能被看见时应该显示的替代文本
  - `title` 属性，用于显示关于图像的附加信息（有趣的，但不是关键性的信息）
  - `longdesc` 属性，用于指向一个外部文件，该文件包含复杂图像（如一个复杂的图表）中所表示的数据的文本显示

- width 和 height 属性，用于告诉浏览器图像的尺寸大小，从而让浏览器为图像分配适当大小的空间
- CSS 背景图像的基础知识
  - 什么时候该使用背景图像（一般而言，当图像不需要替代文本，仅美化和装饰页面时，即可使用背景图像）
  - 如何在 CSS 中定位和重复背景图像

## 练习题

- 使用 alt 属性为图像添加替代文本，为什么是重要的？你真的需要这样做吗？
- 如果你想把一张 1280x786 像素大小的图像显示为 40x30 像素大小的缩略图，你能在 HTML 中实现这样的效果吗？这样做是否是好的做法？
- longdesc 属性的功能是什么？浏览器是如何显示这种属性的？
- valign 和 align 属性的功能是什么？为什么我们不在这篇文章中讲述这两个属性？
- 在默认状态下，CSS 背景图像被定位在元素内的哪个位置？在默认状态下，这些 CSS 背景图像又是如何重复的？
- 上一篇：HTML 列表
- 下一篇：HTML 链接——让我们建一张网吧！
- 目录

## 作者简介



照片来源: [Bluesmoon](#)

Chris Heilmann 从事 Web 开发工作已达 10 年，之前他曾从事电台新闻工作。目前他在 Yahoo! 英国工作，担任培训师和首席开发员，并负责监督面向欧洲和亚洲的前端页面的代码质量。

Chris 的博客地址是 [Wait till I come](#)。他用“codepo8”这个网名，出现在很多社交网络网站上。

---

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

# 18. HTML 链接——让我们建一张网吧！

Posted 12/15/2008 - 16:52 by Lewis

- 网络标准教程

作者: Christian Heilmann · 2008 年 7 月 8 日

- 上一篇: HTML 中的图像
- 下一篇: HTML 表格
- 目录

## 序言

在本篇文章中，你们将学习 Web 发展史中最具有创新性的发明之一，即“链接”。有了链接，阅读文档的人可以通过点击链接跳转到另一个文档，还可以从一个服务器跳转到另一个服务器，且无需断开连接和再次连接。自链接发明以来，Web 已发生了很大的变化，但有一点依然没变，那就是：链接是 Web 体验非常重要的部分，你使用链接的方式，将使访问者易于或难于访问你的网站。

在你们学习完本篇文章后，你们将了解如何创建易于理解的，且可以在任何环境下发挥功能的链接。此外，你们还将了解链接如何影响你的网站在搜索引擎上的排名，并了解一些为链接添加文字说明的诀窍。如同前几篇文章一样，[点击这里可下载一个zip 文件](#)，其中包含几个我在本篇文章中将要引用的文件。本篇文章的内容目录如下：

- 什么是链接？
- 一个锚链接的剖析
- 链接还是目标? name 属性和 href 属性
- 不要让被链接的对象有任何的含糊
  - 使用 title 属性为链接提供附加信息
  - 链接到非 HTML 文档的资源——不要让人们猜测
  - 外部链接和内部链接的对比
- 框架和弹出窗口——不要使用
- 出站链接和入站链接的优点
- 为链接添加文字说明
- 对链接进行样式化

- 总结
- 练习题

## 什么是链接?

链接是一个网站（通常使用 **HTML** 创建，但不总是使用 **HTML** 创建）中指向其它资源的部分，这里的其它资源包括其它 **HTML** 文档、文本文件、**PDF** 等。有一些通过使用 **link** 元素创建的链接（在前几篇文章中你们已经看到过这类链接，它们被用于将 **CSS** 文件导入 **HTML** 文档）是浏览器必须自动跟踪的链接，还有一些链接则是用户可选择去激活的链接。这些链接被称为 **anchors**（锚），你可以通过使用 **a** 元素把锚加入文档。

## 一个锚链接的剖析

你可以通过在元素内加入一个 **a** 元素，将文档中的任何内联元素设置为一个锚链接。例如，在以下这个 **HTML** 文档中，*Yahoo Developer Network*（雅虎开发者网络）这段文本就被设置为一个链接（[linkexample.html](#)）。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>Link Example</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h1>A link to the YDN</h1>
<p>Yahoo Developer Network</p>
</body>
</html>
```

激活这个链接的访问者（可以通过鼠标点击，或在某些情况下使用键盘或语音来激活链接），将离开当前所在的网站而前往雅虎开发者网络站点。链接被激活后自身的外观也可以发生变化，这将在本篇文章讲述对链接进行样式化的部分讨论。

锚元素具有以下几个可用属性：

- `href` 属性：锚指向的资源（外部文件或一个锚 ID）。
- `id` 属性：如果锚被充当一个目标，而不是一个链接时，使用锚 ID。
- `title` 属性：锚所指向外部资源的附加信息。

还是让我们先考察一下锚最重要的属性，然后再讲述，你怎么做才能容易地抓牢你的访问者。

### 链接还是目标? `name` 属性和 `href` 属性

`a` 元素可发挥几种功能，但这取决于你为它赋予的属性。最常用到的属性将是 `href` 属性，它定义链接所指向的资源。同时，这个属性可包含不同的属性值：

- 在同一文件夹中的一个 URL 地址，相对于当前文件夹（例如“`..../help/help.html`”，其中连写的两个英文句点“..”，指的是“站点文件夹层级中的上一级目录”），或者相对于服务器根目录（例如“`/help/help.html`”，在 URL 地址最前面有一个“/”，指的是地址开始于页面所在计算机的根目录）
- 在另一台不同服务器上的一个 URL 地址（例如“`http://wait-till-i.com`” 或 “`ftp://ftp.opera.com/`” 或 “`http://developer.yahoo.com/yui`”）
- 前面有一个 # 符号的一个片段标识符或锚点名称（例如“`#menu`”），它指向位于同一文档中的某个目标。
- URL 地址和片段标识符的混合体，你可以将 `href` 属性指向一个后面跟有片段标识符的 URL 地址（例如 “`http://developer.yahoo.com/yui/#cheatsheets`”），从而直接链接到一个不同文件的某一个部分。

以上任何一种 `href` 属性值都将使 `a` 元素成为一个指向某处的链接。另一方面，`id` 属性将使 `a` 元素成为页面内的一个锚——别的链接指向的某种东西。这有点儿晕，因为它们都使用锚元素 (`a`)。为了便于记忆，你可以这样想：`id` 属性让一个链接成为一个锚，你可以使用这个锚与文档某一个部分相链接。以下这个示例 HTML 文档中就包含了所有不同类型的链接

(`linkexamples.html`):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
"http://www.w3.org/TR/html4/strict.dtd">

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

```
<title>Different Link Example</title>

<link rel="stylesheet" href="linkexamplestyles.css">

</head>

<body>

 <h1>Different Link examples</h1>

 <h2>Example of in-page navigation with fragment identifiers, links and anchors</h2>

 <div id="nav">

 <ul id="toc">

 Section One

 Section Two

 Section Three

 Section Four

 Section Five

 </div>

 <div id="content">

 <div>

 <h2>Section #1</h2>

 <p>Back to menu</p>

 </div>

 <div>

 <h2>Section #2</h2>

 <p>Back to menu</p>

 </div>

 <div>

 <h2>Section #3</h2>

 <p>Back to menu</p>

 </div>

 </div>


```

```

</div>

<div>

<h2>Section #4</h2>

<p>Back to menu</p>

</div>

<div>

<h2>Section #5</h2>

<p>Back to menu</p>

</div>

</div>

<h2>Some other link examples</h2>

Yahoo Developer Network

Tips on marketing yourself

Download different Opera versions

Photo of my book

</body>

</html>

```

用浏览器打开这个文档，并试着去激活里面的一些链接。你会发现激活第一个列表中的任何链接，就都将跳转到文档的相应部分。这是因为它们都是由同一个片段标识符连接起来的，例如，该列表中的第一个链接有一个属性值为 `#sec1` 的 `href` 属性，其属性值与第一个 `h2` 元素内的链接的 `ID` 值相同。这就是在你把一个文档中的两个锚元素连接起来时所全部需要做的，即如果你把其链接在 `href` 属性中时，使用相同的前面带有一个 `#` 的属性值。你可能已经发现浏

览器中的地址栏中所显示的 **URL** 地址已经改变，现在片段识别符被显示在地址的最后，这意味着访问者可以把这个部分加为书签，或将该链接用电子邮件发送给其他人，告诉他们应该访问的确切的 **URL** 地址。

不过，如果你激活任何一个“**Back to menu**”的链接，同样的功能也会出现。这是如何实现的呢？原因在于片段标识符可以是任何带有一个 **ID** 的元素。简单地说：

- 锚链接可以有一个片段标识符，如同 **ref** 的属性值。这个片段标识符必须以 **#** 起始。
- 当被激活时，这个链接将跳转到任何一个带有这个 **id** 属性值的 **HTML** 元素。页面上的每个 **ID** 都必须是唯一的。
- ID** 遵循一些命名约定。最重要的一点是，**ID** 都必须以一个字母数字字符起始，且其中不能有任何空格。

以上讲述了示例文档中链接到菜单和不同部分的链接，其它链接又是怎么样的呢？如果你一一激活这些链接，就会看到它们指向不同的目标，激活一个链接会被带到另一个网站，激活另一个链接则显示一张图片，再激活一个链接则显示另一个网页中的某个部分（通过跳转到一个特定的 **ID**）。如果这些链接你都能正常激活，那是很好的事。不过如果你或你使用的浏览器不能理解其中一些资源，那又将如何呢？

### 不要让被链接的对象有任何的含糊

关于链接，必须记住的最重要一点是：它们是你网站最基本的一部分，对于网站访问者来说，链接非常重要。他们确信点击你在网页上提供的链接，就可以找到适当的相关信息。如果由于被链接的资源不可用，或是以访问者无法打开的格式提供的资源，造成链接不能正常工作。那么，将损害你网站的可信度，这是你应该避免的。

### 使用 **title** 属性为链接提供附加信息

像几乎所有其它 **HTML** 元素一样，你也可以为 **a** 元素添加 **title** 属性，用于为链接添加附加信息。这样当访问者将鼠标悬停在链接上时，浏览器就会显示一个“工具提示”，告诉访问者该链接的附加信息。例如，你可以在 **title** 属性中加入对被链接文档的内容和地址的简短介绍（[titleexample.html](#)）：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

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

```

<title>Adding extra information with a title attribute</title>

<link rel="stylesheet" href="linkexamplestyles.css">

</head>

<body>

 <h1>Adding extra information with a title attribute</h1>

 Find more information on the Yahoo Developer Network.

</body>

</html>

```

**说明 1：**为链接添加 `title` 属性，以让在访问者将鼠标悬停在链接之上时，以工具提示的形式显示信息

不过，你不能指望访问者都有足够的耐心和良好的手眼协调能力，都能看到以工具提示形式显示的重要信息。那些完全看不见网页的视力受损的用户，就几乎肯定无法获取这些信息。尽管屏幕阅读器可以为视力受损的用户读出这种 `title` 属性的功能选项，但在默认状态下这个功能是被关闭的，这也就是你绝不应使用 `title` 属性来传达关键性信息的原因之所在。这里所称的关键性信息，可包括：

- 指向非 HTML 文档的资源，如 PDF 文件、图像、视频、声音文件或下载等的链接。
- 离开当前站点，指向另一个服务器的链接（外部链接和内部链接的对比）。
- 指向将在一个不同的框架或弹出窗口中打开的文档的链接。

### **链接到非 HTML 文档的资源——不要让人们猜测**

如果你点击一个链接，而你使用的浏览器却不知道如何处理该链接指向的资源，那可能是很恼人的。很多网站都存在这个问题，那就是在提供一些指向图像、PDF 文件和视频等的链接时，未提醒访问者“这些链接指向的是浏览器可能无法直接打开的资源”。不可忽视的是，视频常常引起浏览器崩溃。此外，被链接的资源的文件体积可能（如 20MB 大的 PDF 文件）很大，这意味着访问者可能会选择下载文件而不是在浏览器中直接打开文件，以避免消耗大量内存或干

脆就无法打开文件。

Web 有一个最为成功之处，即不会让访问者去猜测当他们在 Web 上执行某一操作时会发什么，而是明确地告诉他们执行的操作将会得到什么结果。就链接而言，你需要做的是明确地告诉访问者被链接的资源是什么。以下是一些示例（[linkingnonhtml.html](#)）：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

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

<title>Linking non-HTML resources</title>

<link rel="stylesheet" href="linkexamplestyles.css">

</head>

<body>

<h1>Linking non-HTML resources</h1>

Find more information on the Yahoo Developer Network site (external)

Download the Dom Cheatsheet \(PDF, 85KB\)

Pick and download different Opera versions from their FTP (external)

Check out a Photo of my book \(JPG, 200KB\)

</body>

</html>
```

通过提供此类关于被链接的文件及其大小的信息，你就可以让访问者自行决定如何去处理

这些链接，而不是预先就假定访问者使用的浏览器已做了一些设置或已安装了一些软件。如果你再对链接进行了适当的样式化，你还可以让链接的外观变得更为好看且一目了然。为保险起见，你还可以在页面上放置一个帮助文件，说明不同文件格式之间的差别，以及在哪里可以下载到显示或打开这些格式的文件的软件。

### 外部链接和内部链接的对比

公司的决策人都很害怕人们在访问公司的网站时过早地离开网站。这常常是很多公司不在其网站上提供第三方链接（除非第三方为通过链接而实现的访问量支付费用）的原因之所在。这是错误的做法，我将在后面部分再讨论这个问题。现在我将谈谈为把访问者留在网站上人们所采取的措施，以及这些措施如何影响到网站的成功。

### 框架和弹出窗口——不要使用

既想在页面上提供一些外部链接，又害怕因访问者点击这些外部链接而让他们离开当前访问的网站，使得一些 Web 开发人员发明了所谓框架和弹出窗口，这其实是两种损害网站可用性的不好做法。

使用 HTML 框架意味着你把在浏览器中显示的页面分隔为几个不同的文件。这样做的好处是，无论你从自己的服务器还是从第三方的服务器载入页面的某些部分时，文件看起来似乎都是一样的。但实际上框架会损害用户体验，有很多负面效果：

- 这样搜索引擎就不能将整个网页编入索引，而可能在搜索结果中，显示那些离开上下文就毫无意义网页的某些部分。
- 访问者无法将整个网页加入书签，也就是说，他们在下一次打开书签时，看到的将不是所存入的整个网页，而依然是上次所看到的框架部分。
- 那些依靠辅助技术设备上网的访问者，在框架间导航时会非常困难。
- 第三方可能不愿看到其网站在框架内显示，因此会使用“**framebreaker**”（框架破坏者）脚本。在你试图将他们的网站嵌入框架时，以一个真正的 URL 地址来替代框架。这是为了阻止不法分子诱使互联网用户，在某个看起来像银行站点的网站上，输入如信用卡号码等信息（被称为“网络钓鱼”）。

放在一个框架集里的链接，使用锚的 target 属性来定位正确的框架。框架集中的每一个框架都有一个特定的名称，激活链接就将在那个框架中打开在 href 属性中所定义的文档。如果框架集不可用时（例如，当一个访问者是通过搜索引擎找到带有链接的文件时），每个链接都会在浏览器的新窗口中打开。

在链接到第三方网站时，在新窗口打开第三方网站也是常用的方式，可通过使用一个脚本

化的弹出式窗口或使用属性值为 `_blank` 的 `target` 属性实现。由于现代的浏览器都带有弹出式窗口拦截器，现在依靠这种在新窗口中打开第三方网站的技术，已不再是安全和保险的方式。

简言之，除非你真正知道你正在做什么，在创建链接时不要使用 `target` 属性。总之，使用 `target` 属性的做法已经过了，这是因为目前多数浏览器都具有多标签浏览界面，这样用户可以在停留在你网站上的同时，在新标签页中打开第三方的网站以备在随后再阅读。在某些情况下，你可能想指明外部链接和内部链接之间的不同，但你还是始终应该让用户自行决定如何去处理这些链接。

### **出站链接和入站链接的优点**

即使第三方是你的竞争对手，你还是可以在自己的网站上创建指向这些第三方网站的链接，这样做是有一些充分理由的。

- 这样做可以让访问者信任你的网站，表明你对自己网站内容的质量有充分的信心，一点也不惧怕来自第三方网站的竞争。
- 这样做也是一个为用户提供全面服务的机会，即你可以将其它网站上的内容和文章，甚至是产品链接到你的网站上——这些内容和文章或产品是你的网站未提供的，但是是访问你网站的人可能有兴趣做进一步的了解。
- 你可以基于第三方的已发布的一篇老文章，提出问题并提供更好的或是不同的解决方案，为此就可以创建一个指向该篇文章的链接，以作为参照。

至于入站链接（即从第三方网站链接到你的网站上）的有用性，基本上是不言而喻的。在越多的高质量网站上面，有指向你的网站的链接（带有相关的关键词），则你的网站在 Google 等搜索引擎上的排名就会越靠前。不过要做到这一点，你需要先证明，你并不排斥在自己的网站上创建指向其它网站的链接。

由于与链接相关的关键词是很重要的，因此下面我们要来看一下，创建好链接的另外一个关键因素，即如何为链接添加文字说明。

### **为链接添加文字说明**

在以上关于链接到非 HTML 文档的资源的部分，我已经谈到一些关于为链接添加文字说明的内容。我们应该牢记一点，那就是链接不仅是页面正文的重要组成部分，而且也是页面内的交互性要素。

一些辅助技术设备会向访问者提供一个 Web 文档中的链接列表（而不是整个文档），以使访问者可以快速地在文档中导航，这就意味着你需要确保所创建的链接上面的文字在其所处的语境之外也有明确的含义。要做此种检查，有一个很简便的做法，那就是用 Opera 浏览器打开任

任何一个网站，然后从菜单中选择“工具>链接”，或是按下 `Ctrl + Shift + L`，就将看到一个显示文档中所有链接及它们所指向的网址的标签页。

这意味着你不仅应确保所有链接上面的文字都有明确的含义，还应确保不出现具有相同文字说明的链接却指向不同资源的情况。一个典型的错误是使用“点击这里”这类的文字链接，如“点击这里下载我们工具软件的最新版本”。与此相比，一个好得多的做法是使用指明某一链接是用来做什么的文字，如“点击这里下载我们软件的最新版本”就最好改为：“你可以点击下载我们工具软件的最新版本并试用它”。

同样地，仅在链接上写上一个“more”（更多）也不是好的做法。在很多新闻网站上，你会发现有很多这种出现在新闻标题和简短摘要后面的“more”或“full story”（详细内容）链接。要解决这个问题，可使用一个代表“more”的被链接的图像，并为图像添加特别的替代文本；或是在链接内的“more”之后添加一个间距，并使用 CSS 将其隐藏。在本课程后面关于菜单和导航的文章中，你们还将学到更多此类技巧和诀窍。

### 对链接进行样式化

到目前为止，我们尚未深入地讲述 CSS。不过你们应该记住，链接的外观也是非常重要的，有几种不同的链接状态可供选择，这些链接状态（它们与 CSS 伪选择符有关，这将在后面的课程中讲述）包括：

- `link` 的默认状态，它定义在文档的某一特定部分链接应显示为什么样子。在默认状态下，未访问的链接显示为蓝色。
- `visited`，指以前已经访问过的链接（及可能已在浏览器缓存中保存的链接）的样式。在默认状态下，已访问的链接显示为紫色。
- `hover`，指鼠标光标悬停在链接之上时，链接的样式。
- `active`，指在激活过程中（即尚在连接到另一个网站的过程之中）的链接的样式；它也是你在浏览器中返回一个页面时最后激活的那个链接的样式。

### 总结

在本篇文章中，你们学习了关于链接的很多知识，其中最重要的是链接是如何工作的及它们应该发挥的功能。作为一个 Web 开发员，你在以后的职业生涯中还会学到很多有关链接的技术和技巧。本篇文章讲述的只是有关链接的基础知识，我希望这些知识可以帮助你们在以后处理链接时总能先想一想：这样的处理方式真是必要的吗？

在本篇文章中，我讲述了：

- [a 元素及其（赞成使用的）属性的剖析](#)
- [a 元素作为链接（具有 href 属性）与它作为锚（具有 name 属性）之间的差别](#)
- 锚名称需要是唯一的
- 告诉访问者链接所指向的文件的格式及文件大小的必要性
- 如何通过使用 title 属性添加将以工具提示形式显示的信息，以及这种方式为什么不是提供关键信息的安全方式
- 外部链接（即指向第三方网站的链接）与内部链接（指向同一服务器上的文件的链接）之间的差别
- 已过时的网页编写习惯，如使用框架和弹出式窗口等，以及为什么不应使用它们
- 导出链接和导入链接的优点
- 如何为链接添加适当的文字，使其在所处的语境之外也有明确的含义；以及这样做为什么是必要的
- 对链接进行样式化的基础知识

掌握了以上这些知识，你就应该能编写出适当地链接在一起的 HTML 文档，并为设计菜单和站点导航做好准备

### 练习题

- 以下这个链接问题出在那里：`<a href="report.pdf" title="report as PDF, 2.3MB">get our latest report</a>`？
  - 链接中的 target 属性是用来做什么的？使用这个属性有什么优点吗？
  - 我已经讲述过链接和锚之间的关系。那么，有描述文档之间的关系的链接属性吗？
  - 你如何创建出这样的链接：即当访问者点击该链接的时候，会被带到页面内的某一个元素？为此你需要什么样的准备工作？
- 
- 上一篇：HTML 中的图像
  - 下一篇：HTML 表格
  - 目录

## 作者简介



照片来源: [Bluesmoon](#)

Chris Heilmann 从事 Web 开发工作已达 10 年，之前他曾从事电台新闻工作。目前他在 Yahoo! 英国工作，担任培训师和首席开发员，并负责监督面向欧洲和亚洲的前端页面的代码质量。

Chris 的博客地址是 [Wait till I come](#)。他用“codepo8”这个网名，出现在很多社交网络网站上。

---

本文采用的授权是 创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

## 19. HTML 表格

Posted 12/15/2008 - 16:52 by Lewis

- 网络标准教程

作者: Jenifer Hanen · 2008 年 7 月 8 日

- 上一篇: [HTML 链接——让我们建一张网吧!](#)
- 下一篇: [HTML 表单基础知识](#)
- [目录](#)

### 序言

你如何使用 Web 标准来组织大量的数据呢? 如果使用数量庞大的嵌套元素, 将所有的数据都有序地放入行和方框中, 一定会让你伤透脑筋。不过对此有一个好的解决办法, 那就是使用表格。

对网页设计师来说, 使用表格是将数据组织整理为表格形式的好方式。换句话说, 表格、图表及其它信息图表可以帮助你阅读和处理经过整理的大量信息, 让你可以对照和比较不同的数据。你在网上随时都可以看到各种表格和图表, 如政治选举结果的摘要或比较、体育比赛的统计、价格比较表、尺码表等。

在互联网发展的早期, 那时候 CSS 作为一种将表现从 HTML 的结构中分离出来的方法尚未得到运用, 网页设计师普遍使用表格来进行页面布局, 创建表格和方框等来安排页面内容的摆置。当然这是不正确的做法。使用表格进行页面布局, 会使页面充斥着大量的嵌套表格和无用代码, 从而导致文档体积膨胀, 难于维护且难于修改。即使是现在, 你都还能看到这类网站。不过你们一定要记住, 应该只将表格用作其本身的用途, 即把数据列入表格之中。至于页面布局, 则应使用 CSS 来控制。

在本篇文章中, 我将讲述如何恰当地使用 HTML 表格。本篇文章的内容目录如下:

- 最基本的表格
- 为表格添加一些特性
- 进一步调整表格的结构
- 使用 CSS 为表格添加样式, 使表格更好看
- 总结

- 延伸阅读
- 练习题

## 最基本的表格

我首先以语义化的 HTML 代码编写一个最基本的表格，这个用作示例的表格列入的是，北美的太平洋地区最近火山爆发的比较数据。在我还是一个小孩子时，就喜欢火山，在我们一家人夏季去探访我祖母的旅程中，曾说服我母亲带我去看了几座火山。当时我非常希望亲眼看到某一座太平洋西北部的火山的爆发，不过未能如愿。第一个表格的代码如下所示：

```
<table>

<tr>

 <td>Volcano Name</td>

 <td>Location</td>

 <td>Last Major Eruption</td>

 <td>Type of Eruption</td>

</tr>

<tr>

 <td>Mt. Lassen</td>

 <td>California</td>

 <td>1914-17</td>

 <td>Explosive Eruption</td>

</tr>

<tr>

 <td>Mt. Hood</td>

 <td>Oregon</td>

 <td>1790s</td>

 <td>Pyroclastic flows and Mudflows</td>

</tr>

<tr>

 <td>Mt .St. Helens</td>

 <td>Washington</td>

 <td>1980</td>

</tr>
```

```
<td>Explosive Eruption</td>
</tr>
</table>
```

上面的代码渲染后的样子：

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt .St. Helens	Washington	1980	Explosive Eruption

让我们一项一项地分析一下以上代码中的 HTML 标记：

- `<table></table>`: 这个 `table` 元素是必须的，用来告诉浏览器你想以表格方式排列内容。
- `<tr></tr>`: `tr` 元素定义表格的一行。这可以让浏览器将 `<tr>` 和 `</tr>` 标签之间所有的内容组织到表格的一行中。
- `<td></td>`: `td` 元素定义表格的单元格，或者行中各个单独的内容。所使用的 `td` 单元格的数量应仅为将数据放入单元格所实际需要的数量。不要使用空的 `td` 单元格来创建空格或填充距，而应该使用 `CSS` 来创建所需的空格或填充距，这不仅是将 `HTML` 文档的结构和表现相分离的好方式，同时还可以使那些靠屏幕阅读器，为他们朗读表格中内容的视力受损的用户，更容易理解表格。

请注意，这些基本元素必须像下面那样被嵌套：

```
<table>
 <tr>
 <td>content</td>
 <td>content</td>
 <td>content</td>
 </tr>
</table>
```

如果以其它顺序排列这些元素，将导致浏览器以非常奇怪的方式渲染表格，甚至无法渲染表格。

### 为表格添加一些特性

现在这个基本的表格已创建出来，接下来可以为表格添加一些复杂一点的特性。首先，我将加入标题和表头，使表格里的数据更语义化并更能为屏幕阅读器所易读。加入这些代码后，表格的标记如下所示：

```
<table>

 <caption>Recent Major Volcanic Eruptions in the Pacific Northwest</caption>

 <tr>

 <th>Volcano Name</th>
 <th>Location</th>
 <th>Last Major Eruption</th>
 <th>Type of Eruption</th>

 </tr>

 <tr>

 <td>Mt. Lassen</td>
 <td>California</td>
 <td>1914-17</td>
 <td>Explosive Eruption</td>

 </tr>

 <tr>

 <td>Mt. Hood</td>
 <td>Oregon</td>
 <td>1790s</td>
 <td>Pyroclastic flows and Mudflows</td>

 </tr>

 <tr>

 <td>Mt. St. Helens</td>
 <td>Washington</td>
 <td>1980</td>

 </tr>
```

```

<td>Explosive Eruption</td>
</tr>
</table>

```

上面的代码渲染后的样子：

## Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption

所使用的新元素包括：

- `<caption></caption>`: 使用 `caption` 元素，可让你为表格加入标题。除非你选择使用 CSS 设置不同的文本对齐方式，绝大多数浏览器会将表格标题居中显示，并将标题行的宽度渲染为和表格一样的宽度。
- `<th></th>`: `th` 元素用于表格内的表头单元格。这不仅有助于更为语义化地描述表头内的内容的功能，还有助于各种不同的浏览器和设备更为准确地渲染表头。以上这个示例中显示的是使用 `th` 元素最为简单的方式。

### 进一步调整表格的结构

作为调整这个表格的最后一个步骤，我将定义表头部分和表体部分，加入一个表脚，并定义行和列标题的范围。我还将就加入一个描述表格内容的摘要属性，最后写成的标记如下所示：

```

<table summary="a summary of recent major volcanic eruptions in the Pacific Northwest">

 <caption>Recent Major Volcanic Eruptions in the Pacific Northwest</caption>

 <thead>

 <tr>

 <th scope="col">Volcano Name</th>
 <th scope="col">Location</th>

```

Last Major Eruption	Type of Eruption
Compiled in 2008 by Ms Jen	
Mt. Lassen	California
1914-17	Explosive Eruption
Mt. Hood	Oregon
1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington
1980	Explosive Eruption

以上代码在浏览器中的样子如下：

## Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Compiled in 2008 by Ms Jen			
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption

所添加的新元素/属性如下所示：

- `thead`, `tbody` 和 `tfoot` 元素：这三个元素分别定义表格的表头、表体和表脚。除非你是在为确实是复杂，有很多数据行和数据列的表格写代码，使用这些元素可能显得多余。不过对一个复杂的表格，使用这些元素就是很恰当的，这样不仅可以按代码编写者的意图使用表格来组织内容，而且也可以让浏览器和其它设备更清楚地显示表格的结构。
- `colspan` 和 `rowspan` 属性：使用 `colspan` 属性，可创建跨度超过一列的表格单元格。在以上示例表格的表脚中，我想让一个表格单元格的跨度为整个表格的宽度，因此我就使用 `colspan` 属性将表脚的跨度定义为 4 列。另外，你也可以为表格单元格添加 `rowspan` 属性，将单元格的跨度定义为跨越 x 行，例如 `<td rowspan="3">`。
- `summary` 属性：该属性用于定义表格内容的摘要，供屏幕阅读器使用（请注意，上面表格被渲染到浏览器时，这个属性不可见）。在万维网联盟过去发布的建议规范（包括 WCAG 1.0 和 HTML 4.0）中，都规定以我上面所述的方式使用 `summary` 属性。不过在万维网联盟新近发布的规范草案中，未再提及 `summary` 属性。看来是否可继续使用 `summary` 属性尚未最后确定，但目前还是可以放心地使用的。毕竟这个 `summary` 属性具有提供网页可访问性的作用，且也不会造成 HTML 代码出现任何问题。
- `scope` 属性：你们可能已经注意到 `th` 标签中的 `scope` 属性，实际上我是使用这个属性在数据行中把火山的名称定义为标题，这种做法是完全被允许的。可在 `th` 元素中添加 `scope` 属性，以告诉屏幕阅读器 `th` 元素中包含的内容是一列或一行的标题。`scope` 属性仅在 `th` 元素中使用。

## 使用 CSS 为表格添加样式，使表格更好看

以上这些列出的元素和属性，都是创建好的 HTML 表格必需的元素和属性。现在我已创建出这个示例 HTML 表格的结构，以下我将使用 CSS，为表格添加一些简单的样式，让它显得更好看一些：

```
body {
 background: #ffffff;
 margin: 0;
 padding: 20px;
 line-height: 1.4em;
 font-family: tahoma, arial, sans-serif;
 font-size: 62.5%;
}

table {
 width: 80%;
 margin: 0;
 background: #FFFFFF;
 border: 1px solid #333333;
 border-collapse: collapse;
}

td, th {
 border-bottom: 1px solid #333333;
 padding: 6px 16px;
 text-align: left;
}

th {
 background: #EEEEEE;
}
```

```

caption {
 background: #E0E0E0;
 margin: 0;
 border: 1px solid #333333;
 border-bottom: none;
 padding: 6px 16px;
 font-weight: bold;
}

```

在加入这些 CSS 代码后，表格最后显示出来的样子如图 1 所示。你也可以 [点击这里观看](#) 表格在浏览器中显示出来的样子。

Recent Major Volcanic Eruptions in the Pacific Northwest			
Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption

Compiled in 2008 by Ms Jen

图 1：现在表格的外观好看多了

看到了吧，现在表格的外观好看多了。在深入地学习了如何使用 CSS 对表格进行样式化（后面有专文讲述个主题）之后，你可以按照你喜欢的任何方式，为表格添加样式。现在我将简要地向你们讲述以上的 CSS 代码：

- **body:** 在以上的 CSS 代码中，我加入了用于设置边距和填充距的属性（本示例中我将边距设为 0），以及设置背景色（白色）、字号和字族、行高的属性。你们可以 [点击这里](#) 下载示例表格的代码，然后试着对 CSS 文件中的属性做一些改动，看看表格的外观会发生哪些变化。
- **table:** 使用 CSS 边框声明，添加了边框。为正确地加入边框，我将 `border-collapse` 属性设置为 `collapse`，重设了表格内边框的属性值，并让 `border-bottom` 成为一条横跨整行的直线，而不是在每个单元格的末端被分割开。我将表格

的 `width` 设置为百分比宽度，具体的值为 **80%**（这样表格宽度在显示时就为屏幕宽度的 **80%**，且会随着浏览器窗口宽度的变化而相应地变化）。

- `th` 和 `td`: 在这个示例表格中，我使用 **CSS** 将文字对齐方式设为左对齐，不过你也可以将它们设为居中对齐，甚至还可以为各个 `th` 和 `td` 元素赋予类名称，然后使用 **CSS** 对每行或每列的样式进行控制（例如，如果要控制行的样式，可以为 `tr` 元素赋予一个类名称）。我还为 `th` 元素和 `td` 元素都添加了一点填充距，使每行内的文字隔得更开一些，以提高易读性。在 `th` 选择符内，我为标题设置了另一种颜色，以把标题和表格其它部分区分开来。
- `caption`: 如果你没有为 `caption` 选择符设置 **CSS** 属性，这样即使标题的 **HTML** 标记是包含在 `table` 标签之中，表格标题也不会有一个边框，其背景色也会和整个页面的背景色一样。因此，在以上这个示例中，我为表格标题设置了一个边框（无下边框，这是因为表格内的边框已提供了下边框），并设置了不同的背景色和字体（粗体），以把表格标题从表头行中分离开来。

## 总结

在本篇文章中，我已全面讲述了创建有效的 **HTML** 数据表格所需要了解的知识。以下是你需要注意特别注意的：

- 以正确的代码创建可被各种浏览器、移动电话及其它设备访问的表格，这是相当重要的。你应以尽可能地精简的 **HTML** 创建表格的结构，同时使用 **CSS** 对表格进行样式化。关于 **CSS**，在后面的课程中将详细讲述。
- 只要你以精简的代码编写表格，并使用如 `scope` 和 `summary` 这样的属性，以及 `caption` 元素，清楚地指明表格的各个部分，那么表格是能被移动设备和屏幕阅读器正常访问的。此外，为确保表格的可访问性，不要使用空单元格来创建单元格间距（应使用 **CSS** 创建单元格间距）。

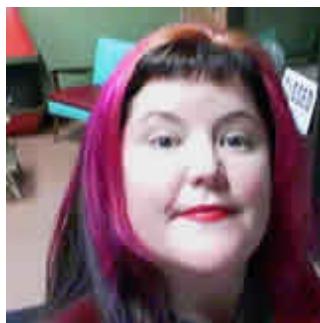
## 延伸阅读

- [W3C HTML 4 Tables Recommendation](#)
- [W3C CSS 2 tables recommendation](#)
- [Roger Johansson's "Bring on the Tables"](#)

## 练习题

- 用 3 个主要的表格元素，即 `table`, `tr` 和 `td` 元素，编写一个简单表格的代码。完成后将其保存，并在浏览器中观看。
  - 参照正文中的第二个示例，为你创建的表格添加表格标题、表头和表脚。完成后，表格在浏览器中显示出来的样子会发生哪些变化？
  - 为让你创建的表格能被屏幕阅读器和手持设备正常访问，你应该怎么做？
  - 创建一个 `CSS` 文件。尝试仅使用 `CSS`，而不使用 `HTML` 标记的属性，对你创建的表格的边框、填充距、单元格间距进行样式化，并添加背景色和设置字体。
- 
- [上一篇：HTML 链接——让我们建一张网吧！](#)
  - [下一篇：HTML 表单基础知识](#)
  - [目录](#)

## 作者简介



Jen 女士是一名专业 Web 设计师和开发人员，同时也是一名摄影师、移动博客和数码艺术的狂热爱好者。她很早就显示出具有艺术和设计的天赋。1996 年，一个因懂点计算机知识就自命不凡的人断言艺术家是学不会如何写网页代码的，为反击这种无稽之谈，她自学了 HTML，并从此深深爱上了网页设计。

Jen 女士是 Black Phoebe Designs 设计事务所(从事网页和移动博客设计)的创始人和业主。她在位于爱尔兰都柏林市的三一学院获得了计算机科学和多媒体系统硕士学位。2001—2005 年间，她在洛杉矶地区的一所大学教授网页设计课程。她参与过诺基亚公司的两个移动博客项目，即 Wasabi Lifeblog 项目（2004–2005）和 Nokia Urbanista Diaries 项目（2008）。她的设计事务所的网址为 [blackphoebe.com](http://blackphoebe.com) 和 [blackphoebe.mobi](http://blackphoebe.mobi)。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 20. HTML 表单基础知识

Posted 12/15/2008 - 16:53 by Lewis

- 网络标准教程

作者: Jenifer Hanen · 2008 年 7 月 8 日

- 上一篇: HTML 表格

- 下一篇：罕为人知的语义元素
- 目录

## 序言

每个人都曾看到过和用过表单，不过你曾写代码创建过一个表单吗？

表单是网页上用于输入信息的区域，例如向文本框中输入文字或数字，在方框中打勾，使用单选按钮选中一个选项，或从一个列表中选择一个选项等。在你按下提交按钮后，表单就被提交到网站。

表单在网上随处可见，它们被用于在登录页面输入用户名和密码，对博客进行评论，在社交网络网站填写个人信息，或在购物网站指定记账信息等。

创建表单并不难，但要创建出符合 Web 标准的表单又是怎么样的呢？如果你已经通读了本课程前面的所有文章，也许已认识到 Web 标准是规范网页编写工作的好方式。因此，创建一个符合 Web 标准的、可访问性好的表单，并不比以不符合规范的代码编写一个不符合 Web 标准的表单更费力。

在本篇文章中，我将一步一步讲述如何创建 HTML 表单，从最基本和最简单的表单到更复杂的表单。在学习完本篇文章后，你们应该就能掌握使用 HTML 创建有效的、可访问性好的表单的基础知识。本篇文章的内容目录如下：

- 第一步：基本代码
- 第二步：添加结构和行为
- 第三步：添加语义、样式和更多的结构元素
- 总结
- 延伸阅读
- 练习题

### 第一步：基本代码

我们从创建一个简单的评论表单开始，评论表单可用于让访问者对网站上的内容如文章等发表评论，还可用于让不知道你邮箱地址的访问者在表单中输入信息，与你联系。这种评论表单的代码如下：

```
<form>
Name: <input type="text" name="name" id="name" value="" />
Email: <input type="text" name="email" id="email" value="" />
```

```

Comments: <textarea name="comments" id="comments" cols="25"
rows="3"></textarea>

<input type="submit" value="submit" />

</form>

```

如果你把以上代码输入一个 HTML 文档，然后用浏览器打开这个文档，代码将被渲染为如图 1 所示的样子。

图 1：一个简单的表单示例

请自己动手将以上的代码输入你自己的样本 HTML 文档，并在浏览器中载入这个文档，或是 [点击这里](#) 观看在一个单独页面显示的表单。你们可以试着对代码做些改动，看看表单会发什么变化。

在以上这段代码中，你们看到有一个 `<form>` 开始标签和一个 `</form>` 结束标签，以及在这两个标签之间的一些信息。这个元素内包含两个用于让访问者输入名字和电子邮件地址的文本输入框，以及一个用于发表和提交评论的文本区域。

以下是对表单代码的分析：

- `<form></form>`: 这两个标签是创建一个表单所需的基本标签，每一个表单都必须以 `<form>` 标签起始，并以 `</form>` 标签结束。  
`<form>` 标签可有几个属性，这些属性将在以下第二步中再讲述。有一点你们必须注意，那就是不能在一个表单内再嵌套一个表单。
- `<input>` (如果你使用 XHTML 文档类型，则应写为 `<input />`): 这个标签定义了那些可以输入信息的区域。在以上这个示例表单中，`input` 标签定义了网站访问者可以输入名字和电子邮件地址的文本框。

每个 `input` 标签都必须有一个定义接受类型的 `type` 属性，其可能的属性值包括：`text` (文本框)，`button` (按钮)，`checkbox` (复选框)，`file` (文件)，`hidden` (隐藏字段)，`image` (图像)，`password` (密码框)，`radio` (单选按钮)，`reset` (重置按钮)，`submit` (提交按钮)。

每个 `<input>` 标签还必须有一个名字 (除了一些特殊情况，需要将 `value` 属性值始终设为

和 `type` 属性值一样，如 `type="submit"` 或 `"reset"`），做为编码人员的你就可以决定这件事。`name` 属性定义提交表单时数据字段的名称（如一个数据库，或是通过服务器端的脚本发送给网站管理员的电子邮件）。当表单被提交时，绝大多数脚本都使用 `name` 属性来将表单数据放入数据库，或放入可供人阅读的电子邮件。

因此，如果 `<input>` 元素是用于让网站访问者输入其名字的，则 `name` 属性可以为 `name="name"` 或 `name = "first name"` 等。如果 `<input>` 标签是用于输入电子邮件地址，则 `name` 属性可以为 `name="email"`。为让你更容易地创建表单，并让使用表单的人更容易使用，建议你以语义化的方式为 `<input>` 元素命名。

所谓语义化的方式，我指的是像上面那样，根据其功能为其命名。如果要输入的数据是电子邮件地址，则把其命名为 `name="email"`。如果要输入的数据是网站访问者的街道地址，则把其命名为 `name="street-address"`。使用的命名词语越准确，不仅让你编写表单代码变得更容易，而且让你以后做维护网站的工作时也更为轻松，同时还让接收表单的人或数据库更容易解释数据。我们需要多思考一会儿，选择精确的名称。

- 每个 `<input>` 标签还必须有一个 `value` 属性。`value` 属性值可以为空，即 `value=""`，这将告诉处理脚本插入网站访问者在表单框里输入的任何数据。对于复选框、单选按钮、隐藏字段、提交按钮，或其它类型的属性，你可以为其设置你希望的默认值。在其它一些情况下，如对“提交按钮”或“隐藏字段”类型，你将其值设为等于最终的输入。例如，`value="yes"` 代表“是”，提交按钮设为 `value="submit"`，重置按钮设为 `value="reset"`，隐藏的重定向字段设为 `value=http://www.opera.com` 等。

以下是一些如何使用 `value` 属性的示例：

空值属性，由用户的输入决定属性值：

- 代码为：`<input type="text" name="first-name" id="first-name" value="" />`
- 用户输入：Jenifer
- 当表单提交时，`first-name` 的值以“Jenifer”发送。

预定值：

- 代码为：`<input type="checkbox" name="mailing-list" id="mailing-list" value="yes" />`
- 用户在方框里打勾，代表他们希望加入网站的邮件列表。
- 当表单被提交时，`mailing-list` 的值以“yes”被发送。

- 在 `<input>` 元素之后，你们可以看到另一个有点不同的 `textarea` 元素。

`textarea` 元素提供了一个可输入文本的更适宜的空间。与 `<input>` 元素提供的那种一行的普通文本框不同，`textarea` 元素提供多行的可输入文本的区域，其行数可以由你定义。

注意 `cols` 和 `rows` 属性，它们是 `textarea` 元素必须拥有的属性，它们定义文本区域的行数和列数为多大，属性值的单位是字符。

- 最后，在上面的示例中，还有一个特别的属性为 `value="submit"` 的 `<input>` 元素，这个元素将在浏览器中渲染为一个提交按钮，而不是一个单行文本框。点击提交按钮，表单就会被提交到已预订指定的接收数据的地方（在以上的示例中，功能尚不完全，因此进行表单提交时将不起作用）。

## 第二步：添加结构和行为

为什么当你在上面的示例表单中填写入文本，并点击提交，还不起作用呢，而且表单的外观也不好看，只有一行呢？答案在于尚未为表单添加结构，并定义表单提交到那个地方。

现在看一下加入新代码后的表单：

```
<form id="contact-form" action="script.php" method="post">

<input type="hidden" name="redirect" value="http://www.opera.com" />

 <label for="name">Name:</label>
 <input type="text" name="name" id="name" value="" />

 <label for="email">Email:</label>
 <input type="text" name="email" id="email" value="" />

 <label for="comments">Comments:</label>
 <textarea name="comments" id="comments" cols="25" rows="3"></textarea>


```

```

 <input type="submit" value="submit" />

 <input type="reset" value="reset" />

</form>

```

当以上这段代码在浏览器中渲染时，样子如图 2 所示：

图2：新表单示例，看起来更好一些了，但尚不完善

你们可以 [点击这里](#)在独立的网页中观看这个新表单。

在以上这段代码中，我为第一步那个基本的表单添加了一些代码。以下是这些新添加代码的逐项分析：

- 在 `<form>` 标签中加入了一些新属性。我加入了 `id` 属性，这不仅为表单赋予了语义化的名称，而且为表单提供了独一无二的 ID，这样便于使用 CSS 为表单添加样式，或使用 JavaScript 为表单添加行为。在每个网页中每个 `id` 必须是唯一的，在这个示例中，我将其命名为 `contact-form`。
- 灯光、摄像、开始！当你在第一个表单中点击提交按钮时，什么也没有发生，这是由于尚未添加行为或提交方式。`method` 属性指定了数据如何发送给处理该等数据的脚本。两种最常见的提交方式是“GET”和“POST”。“GET”提交方式将把数据发送到网页的 URL 地址中（你们有时候会看到这样的 URL 地址：  
`http://www.example.com/page.php?data1=value1&data2=value2...`，这就是在使用“GET”提交方式发送数据）。除非你有特别的理由需要使用“GET”提交方式，最好不要使用这种方式发送安全信息，因为通过 URL 地址传输信息，任何人都可以看到信息。而“POST”提交方式，则是通过处理表单的脚本。无论是以电子邮件方式发送给网站管理员，或者是存储到数据库中供以后访问，都比“GET”URL 要好。“POST”提交方式更为安全，通常也是更好的表单提交方式。

如果你很在意表单中数据的安全性，例如要向一个购物网站提交信用卡号码。则你应当使用

带有安全套接层（SSL）的 https 协议。简单地说，这意味着数据将通过 https 协议而不是 http 协议传送。你可以查看一下一些购物网站或网上银行的 URL 地址，你很可能看到的是 https:// 这样的 URL 地址，而不是 http:// 这样的 URL 地址。https 协议和 http 协议二者之间的差别在于，https 连接在传输数据的速度方面要比 http 连接慢一点，但数据是加密传送，这样任何侵入数据连接的人都无法知道正在传输的数据到底是什么。你可以询问你的虚拟主机提供商，了解他们如何向你提供 https 和 SSL 服务。

- action 属性规定表单数据应该发到哪个脚本文件来进行处理。许多虚拟主机提供商都提供有发送邮件脚本，或其它可定制的表单脚本（请查阅你的虚拟主机技术文档了解相关信息）。另一方面，你也可以使用自己，或他人编写的用于处理表单数据的服务器端脚本。通常，人们使用 PHP、Perl、Ruby 等语言来编写处理表单的脚本。例如，使用这类脚本，你可以发出包含表单信息的电子邮件，或是将表单信息输入数据库，供日后使用。

为你们编写一个服务器端脚本，或是教你们如何编写服务器端脚本的代码，这已超出了本课程的范围。请询问你的虚拟主机提供商，他们可为你提供哪些服务器端脚本，或请教专业编程员。

如果你想深入了解服务器端脚本，可以访问以下列出的一些网络资源：

- Perl: <http://www.perl.com>
- PHP: <http://www.php.net>
- 处理表单的PHP技术文档: <http://uk3.php.net/manual/en/tutorial.forms.php>
- Python: <http://python.org>
- Ruby: <http://www.ruby-lang.org>
- Sendmail: <http://www.sendmail.org>
- ASP.NET: <http://www.asp.net>
- 在上面的表单代码中，新加入的第二段代码是“隐藏的”输入字段，即一个网页重定向。为实现将 HTML 标记的结构与表现、行为相分离的目标，让处理表单的脚本在表单提交时，同时也把用户重定向到一个网页，是很理想的方式。你肯定不愿意让用户在提交表单后不知道下一步究竟该怎么做，此时把用户重定向到一个包含“下一步怎么做”的链接的网页，肯定能大大改善用户体验。在我们这个示例表单中就加入了一段代码，该段代码规定在用户提交了表单后，将被重定向到 Opera 的主页。
- 为让表单的外观更好看，我把所有的表单元素都放入了一个无序列表，这样我就可以使用标记将这些元素排列好，然后使用 CSS 为表单添加样式。

一些人也许会说，你不应该使用一个无序列表来标记表单，而是应该用定义列表来标记表单。还有一些人也许会说，根本就不应该使用列表，而是应该使用 CSS 来为 `<label>` 和 `<input>` 标签添加样式。我想让你们对这两种不同的观点进行思考，就哪一种方式更为适当、更为语义化做出自己的判断。在以上这个示例表单中，我选择使用无序列表。

- 最后，我为所有表单元素添加了标识，既标明了表单元素的含义，也让各种联网设备可以访问表单。通过查看 `label` 元素的内容，你会发现通过将 `input` 和 `textarea` 的 `id` 属性值设置得和 `label` 元素的 `for` 属性值相同，可以将 `label` 绑定到相应的表单元素上。这是非常好的做法，因为这样不仅直观地指明了屏幕上每个表单字段的目的，而且还使表单域具有更为语义化的含义。例如，那些通过屏幕阅读器访问网页的视力受损的用户，现在就能知道标识对应着哪一个表单元素。`id` 还可以用于使用 CSS 对各个表单字段进行样式化。

现在你们可能会问，为什么表单元素既要有作为识别符的 `id` 属性，又要有 `name` 属性呢？

答案如下：没有指定 `name` 属性的 `input` 元素不会被提交到服务器，因此 `name` 属性是必需的。同时，由于 `id` 属性用于将表单元素与它们相对应的 `label` 元素联系起来，因此也是必需的。

在添加了以上代码后，现在这个示例表单看起来要好一些了，不过其外观还是不太好看。在为表单应用样式前，还需要对它进行一些改动。

### 第三步：添加语义、样式和更多的结构元素

现在我将把示例表单的最后版本呈现在你们面前，其代码如下：

```
<form id="contact-form" action="script.php" method="post">

<fieldset>

 <legend>Contact Us:</legend>

 <label for="name">Name:</label>

 <input type="text" name="name" id="name" value="" />

 <label for="email">Email:</label>

 <input type="text" name="email" id="email" value="" />

</fieldset>
</form>
```

```


 <label for="comments">Comments:</label>

 <textarea name="comments" id="comments" cols="25" rows="3"></textarea>

 <label for="mailing-list">Would you like to sign up for our mailing
list?</label>

 <input type="checkbox" checked="checked" id="mailing-list" value="Yes,
sign me up!" />

 <input type="submit" value="submit" />

 <input type="reset" value="reset" />

</fieldset>

</form>

```

当以上表单代码在浏览器中被渲染时，其呈现出来的样子如图 3 所示。

The screenshot shows a contact form with the following elements:

- Contact Us:** (Section title)
- Name:** (Label)
- Email:** (Label)
- Comments:** (Label)
- Would you like to sign up for our mailing list?** (Label)
- submit** **reset** (Buttons)

图 3：示例表单的最后版本

要观看这个表单在浏览器中实际显示出来的样子，可以 点击这个链接，在单独的网页中观看这个表单。

我为这个表单添加的最后两个主要元素是 `fieldset` 元素和 `legend` 元素。这两个元素都不是必须使用的元素，但较复杂的表单一般都会用到它们。

使用 `fieldset` 元素，可让你将表单组织入语义化的模块中。在较复杂的表单中，你可以为地址信息、记账信息、顾客偏好等使用不同的 `fieldsets` 元素。使用 `legend` 元素，则可让你为每个 `fieldset` 区段命名。

我还使用了一点 CSS，对结构性标记进行了样式化。我使用的是一个外部样式表，可[点击这里观看样式](#)。我使用 CSS 想完成的两个最重要的任务：第一，添加边距，以让标签和输入框排列整齐；第二，去除无序列表的项目符号。以下是那个外部样式表中的 CSS 代码：

```
#contact-form fieldset {width:40%;}

#contact-form li {margin:10px; list-style: none; }

#contact-form input {margin-left:45px; text-align: left; }

#contact-form textarea {margin-left:10px; text-align: left; }
```

这些代码的功能是什么？第一行代码为 `fieldset` 设置宽度，使之不占据整个页面。如果你不希望它有边框，可以使用 `{border: none;}` 来设置。第二行代码为 `li` 元素添加了 10 个像素的边距，以使每个列表项目间有一些间隔。第三行和第四行代码，为 `input` 和 `textarea` 元素设置了左边距，这样它们就可以排列整齐，且不使标签显得挤在一起。如果你想了解关于对表单进行样式化更多的知识，请查阅本课程后面的专题文章“对表单进行样式化”（稍后将会发布）或 [Nick Rigby](#) 的文章“更美观且可访问性好的表单”。关于边距和边框，在后面的文章中将详细讲述。

## 总结

在本篇文章中，我讲述了创建符合 Web 标准的表单的三个基本步骤。还有很多关于表单的知识我并未在本篇文章中讲述，包括快捷键、复选框、单选按钮、自定义提交、重设按钮、选择框等，这些留待你们自己去钻研。

在以上示例表单中，我加入了一个复选框，以向你们展示如何使用 `input` 元素的附加属性，来收集单行文本输入或多行文本区输入范围以外的信息。可以使用 `type` 为 `checkbox` 和 `radio` 的属性值，列出一些简短的问题及一个供阅读者选择的选项列表（复选框可选择多个选项，而单选按钮只能选择一个选项）。在调查表单中，单选按钮非常有用。

在本篇文章中未讲述的 `select` 元素，可用于下拉菜单选择（例如，国家、州或省的列表）。

## 延伸阅读

- Cameron Adams, “Accessible, stylish form layout”:  
<http://www.themaninblue.com/writing/perspective/2004/03/24/>.
- Brian Crescimanno, “Sensible Forms: A Form Usability Checklist”:  
<http://www.alistapart.com/articles/sensibleforms/>.
- Simon Willison, “Easier form validation with PHP”,  
<http://simonwillison.net/2003/Jun/17/theHolyGrail/>.
- The Spec. Not any old specification, but THE W3C  
SPEC—<http://www.w3.org/TR/html401/interact/forms.html>. If you ever have a bout of insomnia in which a warm glass of milk, counting sheep, and Ambien are not putting you to sleep, go read the whole spec for HTML 4.01 or XHTML 1.0 at the w3.org. It is cheaper and more effective than any remedy out there. [Insert name of deity here] bless the engineers of the world.
- The nice folk over at the W3.org have defined the differences between “GET” & “POST” and when to use them: <http://www.w3.org/2001/tag/doc/whenToUseGet.html>.
- Many blessings upon Mr. Rigby: <http://alistapart.com/articles/prettyaccessibleforms>.

## 练习题

自己动手编写一个联系表单的代码。

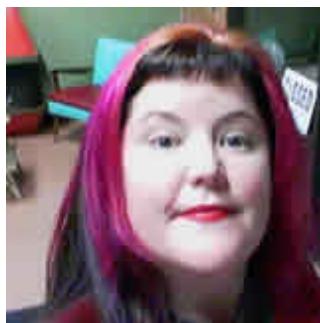
1. 创建一个要求用户提交姓名、电子邮件地址、评论的简单表单。
2. 为表单添加一个复选框，询问读者是否愿意加入邮件列表。
3. 使用 CSS 为表单添加一些样式，包括设置表单的宽度、将标签设置为左对齐、为页面设置背景色等。

提示：可以尝试添加各种表单元素和 CSS 样式，看看这个简单的联系表单会发生哪些变化。

更多的提示：如果你想进一步学习更高级的内容，去找一个脚本或使用虚拟主机提供商用于将表单发送给你的脚本。如果有什么问题，你可以请教程序员如何调试脚本。

- 上一篇：HTML 表格
- 下一篇：罕为人知的语义元素
- 目录

## 作者简介



Jen 女士是一名专业 Web 设计师和开发人员，同时也是一名摄影师、移动博客和数码艺术的狂热爱好者。她很早就显示出具有艺术和设计的天赋。1996 年，一个因懂点计算机知识就自命不凡的人断言艺术家是学不会如何写网页代码的，为反击这种无稽之谈，她自学了 HTML，并从此深深爱上了网页设计。

Jen 女士是 Black Phoebe Designs 设计事务所(从事网页和移动博客设计)的创始人和业主。她在位于爱尔兰都柏林市的三一学院获得了计算机科学和多媒体系统硕士学位。2001—2005 年间，她在洛杉矶地区的一所大学教授网页设计课程。她参与过诺基亚公司的两个移动博客项目，即 Wasabi Lifeblog 项目（2004—2005）和 Nokia Urbanista Diaries 项目（2008）。她的设计事务所的网址为 [blackphoebe.com](http://blackphoebe.com) 和 [blackphoebe.mobi](http://blackphoebe.mobi)。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 21. 罕为人知的语义元素

Posted 12/15/2008 - 16:53 by Lewis

- [网络标准教程](#)

作者: [Mark Norman Francis](#) · 2008 年 7 月 8 日

- 上一篇: [HTML 表单基础知识](#)
- 下一篇: [通用容器——div 和 span 元素](#)
- [目录](#)

### 序言

在本篇文章中，我将向你们介绍一些更隐晦、罕为人知且极少用到的 HTML 语义元素。

我将介绍有关标记程序代码、与计算机的交互、引用与缩略语，以及对文档所做的改动等内容。

在本文的结尾部分，我还将在 **HTML 5** 草案中对新定义的额外语义元素的一些提议。

- 强调显示联系信息
- 编程语言和代码
- 显示计算机交互
- 变量
- 引用
- 缩略语
- 定义实例
- 上标和下标
- 换行符
- 水平线
- 文档的改动（插入与删除）
- 一些将在未来引入的 **HTML** 元素
- 总结

**附注：**在每个代码示例后面，都有一个“查看源文件”的链接，通过点击链接你将看到源代码在浏览器中的实际渲染效果，这些效果包含在一个不同的文件中——这样你就可以一边看着源代码，一边看着它们在浏览器中是如何被渲染的实际例子。

### 强调显示联系信息

`address` 元素也许是 **HTML** 中命名最不当、最被人误解的元素。乍看之下，由于其名称为“地址”，似乎就是用来封装地址、电子邮件地址、邮政地址等内容的元素，但这并不完全对。

`address` 元素的实际含义是提供其所在网页或该网页主要部分的作者或作者们的联系信息。可采用的形式包括姓名、电子邮件地址、邮政地址，或指向另一个包含更多联系信息的页面的链接。例如：

```
<address>
 Mark Norman Francis,
 1-800-555-4865
</address>
```

[查看源文件](#)

在下面这个示例中，地址包含在页脚段落中，并链接到网站的另一个网页上。这个链接所指向的网页可包含更为详细的联系信息，如此一来就可以避免在整个网站的每个页面上都重复显示联系信息。

```
<p class="footer">© Copyright 2008</p>

<address>

Mark Norman Francis

</address>
```

[查看源文件](#)

当然，如果网站的作者不止一人，也可采用同样的方式来显示联系信息，只要把不同的作者链接到不同的联系信息页面就可以了。

使用 `address` 元素来标示其它任何类型的地址都是\*不正确\*的，以下就是一个不正确的例子：

```
<p> Our company address: </p>

<address>

 Opera Software ASA,
 Waldemar Thranes gate 98,
 NO-0175 OSLO,
 NORWAY

</address>
```

[查看源文件](#)

(当然，如果 **Opera** 公司要为本文承担集体责任的话，那么即使只有我而不是 **Opera** 公司的全体员工，是这个特别的网页的作者，以上这种做法就是正确的)。

对于一般的地址，你可以使用一种叫做“微格式”的东西来表示某一段落中包含一个地址。在 [dev.opera.com](http://dev.opera.com) 上的其它文章中有更多关于“微格式”的信息。

### 编程语言和代码

`code` 元素用于指明计算机代码或编程语言，如 **PHP**、**JavaScript**、**CSS**、**XML** 等。对于某个句子中简短的代码示例来说，你可以把代码片段包裹在 `code` 元素中，就像这样：

```
<p>It is bad form to use event handlers like
```

```
<code>onclick</code> directly in the HTML.</p>
```

[查看源文件](#)

对于占据多行的较大代码示例，你可以使用预格式化的代码块，就如 标记 **HTML** 中的文本内容一文中所描述的那样。

尽管没有定义过的方法来指明 `code` 元素中包含的是何种编程语言或代码，你也可以使用 `class` 属性来指明。通常的习惯（见 **HTML 5 草案**）是使用前缀 `language-`，后边再跟上编程语言的名称。如此以上那段代码就将变为：

```
<p>It is bad form to use event handlers like
<code class="language-javascript">onclick</code>

directly in the HTML.</p>
```

[查看源文件](#)

某些编程语言的名称难以用 `class` 来表示，如 **C# (C-Sharp)**。书写 **C#** 的正确方式可以是“`class='language-c\#'`”，但这容易引起混淆或打错。我建议在 `class` 中仅包含字母和连字符，并将编程语言的全称拼写出来，因此在本例中可用“`class='language-csharp'`”来代替。

## 显示计算机交互

`samp` 和 `kbd` 这两个元素可用于指明与某个计算机程序进行输入和输出的交互，例如：

```
<p>One common method of determining if a computer is connected
to the internet is to use the computer program <code>ping</code>
to see if a computer likely to be running is reachable.</p>
<pre><samp>kittaghy%</samp> <kbd>ping -o google.com</kbd>

<samp>PING google.com (64.233.187.99): 56 data bytes
64 bytes from 64.233.187.99: icmp_seq=0 ttl=242 time=108.995 ms

--- google.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss</pre>
```

```
round-trip min/avg/max/stddev = 108.995/108.995/108.995/0.000 ms
</samp></pre>
```

[查看源文件](#)

`samp` 元素指明了某个计算机程序的输出示例。如以上代码所示，不同类型的输出可使用 `class` 属性来标示，不过现在还没有什么广泛采用的约定来规定哪些 `class` 属性应该被使用。

`kbd` 元素指明的是用户在与计算机进行交互时的输入。尽管交互传统上是指键盘输入 (`kbd` 是 `keyboard` 的缩写)，但是其它类型的输入，如语音输入，也应用 `kbd` 元素指明。

## 变量

`var` 元素用于标示文本内容中的变量，包括代数数学表达式或程序代码中的变量。例如：

```
<p>The value of <var>x</var> in
3<var>x</var>+2=14 is 4.</p>

<pre><code class="language-perl">
my <var>$welcome</var> = "Hello world!";
</code></pre>
```

[查看源文件](#)

## 引用

`cite` 元素用于标示其邻近页面内容的来源——当需要引用某个人、某本书或其它出版物，或笼统地向人们提及另一个来源，该来源应该被包裹在 `cite` 元素中。例如：

```
<p>The saying <q>Everything should be made as simple as possible,
but not simpler</q> is often attributed to <cite>Albert
Einstein</cite>, but it is actually a paraphrasing of a quote which
is much less easy to understand.</p>
```

[查看源文件](#)

## 缩略语

`abbr` 和 `acronym` 元素用于指明哪些词语是缩略语，并提供了一种不需要地打断该文档流而对它们进行展开的方法。

作为缩略语的文字应被包裹在 `abbr` 元素中，而缩略语的全称则放在 `title` 属性中，就像这样：

```
<p>Styling is added to
<abbr title="Hypertext Markup Language">HTML</abbr> documents
using <abbr title="Cascading Style Sheets">CSS</abbr>. </p>
```

[查看源文件](#)

首字母缩写词是缩略语的一种，与一般缩略语的区别在于它被公认如同一般的单词一样，它的发音也是如此。比如说 `scuba` 就是一个首字母缩写词，它是由“`self-contained underwater breathing apparatus`”（自给式水中呼吸器）缩写而成。尽管 `HTML 4.01` 规范允许既使用 `abbr` 元素，又使用 `acronym`，但在实际使用中还是会遇到一些问题……

`Internet Explorer` 浏览器（`IE7` 及其更早的版本并未像其它浏览器一样为缩略语提供点状下划线）不认 `abbr` 元素，而只认 `acronym` 元素。不幸的是，`acronym`（首字母缩写词）只是 `abbreviation`（缩略语）的一个子集，因此使用 `acronym` 元素来标记“`HTML`”这样的缩略语是不正确的做法。

此外，由于任何首字母缩写词都是一个有效的缩略语，在 `HTML 5` 规范草案中，为了支持标准化，`acronym` 元素被舍弃了，而用 `abbr` 元素来同时表示缩略语和首字母缩写词。

因此，最好的做法是避免使用 `acronym` 元素，而在你的代码中自始至终只使用 `abbr` 元素。如果你需要为 `abbr` 元素添加一些视觉上的样式，可以在其中放入一个 `span` 元素，并代替 `abbr` 在其上添加样式，这样所有的浏览器都将应用该视觉样式。随后的“`样式化文本`”一文将详细介绍这方面的内容。

## 定义实例

关于如何恰当地使用 `dfn` 元素，存在一些易于混淆之处，`dfn` 元素在 `HTML` 规范中被描述为“所附上的术语的定义实例”，这与定义列表中使用的 `dt` 元素（`定义术语`）的概念非常相似。

两者之间的差别仅在于 `dfn` 元素中所使用的术语并非需要是某个术语和说明列表的一部分，而可以用作常规文本流的一部分，甚至可以用在会话体文章中。以下是一个使用 `dfn` 元素的示例：

```
<p><dfn>HTML</dfn>: HTML stands for "HyperText Markup Language". This is
the language used to describe the contents of web documents.</p>
```

在 `HTML` 这个术语出现后，紧接着就是这个术语的定义，这就很适合使用 `dfn` 元素。你

应当在一个页面中只使用一次 `dfn` 元素，即在术语第一次被定义时使用，由于术语本来就不需要重复定义，因此这倒也不会太麻烦。

这样使用 `dfn` 元素好倒是好，但仅举一个孤立的例子并不全面——当某个缩略语在一个网页中多次出现时也建议使用 `dfn` 元素。例如，在前面的 [HTML 基础知识](#) 一文中，`HTML` 这个术语就出现了 40 多次。如果每次都重复使用 “`<abbr title="HyperText Markup Language">HTML</abbr>`” 这段代码，将会造成带宽的浪费，在视觉上导致注意力的分散，而且使用屏幕阅读器的用户将不得不重复地听到 `HTML` 的全称，这会使他们感到厌倦。在这种情况下，本来是在第一次为用户定义 `HTML` 时加入如下代码的：

```
<p><dfn><abbr>HTML</abbr></dfn> ("HyperText Markup Language") is
a language to describe the contents of web documents.</p>
```

[查看源文件](#)

之后当 `HTML` 这个术语再次出现时，就可以简单地将其标记为 “`<abbr>HTML</abbr>`” 而后用户代理就可以让用户用某种方式去检索这个缩略语的定义实例。但不幸的是，目前还没有一种用户代理可以做到这一点，包括屏幕阅读器。因此更好的做法是同时也使用 `title` 属性来提供定义信息，如以下所示：

```
<p><dfn><abbr title="HyperText Markup Language">HTML</abbr></dfn> ("HyperText
Markup Language") is a language to describe the contents of web documents.</p>
```

[查看源文件](#)

遗憾的是，这样做又让 `HTML` 的全称出现了重复，这对使用屏幕阅读器的用户可能成为问题。不过如果省略掉全称的话，对视力正常的用户而言，会使文档的可用性降低，毕竟网站的用户群中视力正常的用户还是占大多数。

我建议某个页面中如果只有一两个术语需要定义的话，那么这种权衡是值得接受的。（如果网页中有大量术语需要定义，最好在页面中创建一个词汇表章节或另建一个页面，在这个页面中就可以使用更为严谨的定义列表标记）。如果你非常重视这个问题，可以使用以下代码：

```
<p><abbr title="HyperText Markup Language">HTML</abbr>
(<dfn>HyperText Markup Language</dfn>) is a language to
describe the contents of web documents.</p>
```

[查看源文件](#)

尽管如此，用户代理还是得提供一些方法，来将定义实例与页面内出现的被定义的术语联系在一起。虽然对于 CSS 来说，它仍然是一个可以用来进行样式的不错的钩子，但目前还没有一种浏览器能有效地支持 `dfn` 元素。以上我提出的已经是目前最好的解决办法了。

现有的 HTML 规范对 `dfn` 元素应该如何使用并没有明确的规定，已有的规定可能也不是基于该元素在实际运用中的做法——不然的话应该会有将术语和其完整描述或定义结合在一起的方法。HTML 5 规范中对 [如何使用 `dfn` 元素有详细的描述](#)，但该规范尚在草拟之中，目前还不适于将该规范用在网络上。

## 上标和下标

要将某些文本的一部分标记为上标文本或下标文本（即相对于该文本的其它部分略微升高或下降），你可以使用 `sup` 和 `sub` 元素。

为了正确使用缩略语，一些语言需要用到这些元素，此外，在标记一小段数学内容时，也可以使用这两种元素，而不必使用 MathML 元素(该元素是一种专门的，重量级的数学标记语言，用于标记重量级的数学公式)。

以下是这两种元素的代码示例：

```
<p>The chemical formula for water is H₂O, and it
is also known as hydrogen hydroxide.</p>

<p>The famous formula for mass-energy equivalence as derived
by Albert Einstein is E=mc² – energy
is equal to the mass multiplied by the speed of light
squared.</p>
```

[查看源文件](#)

## 换行符

由于 HTML 对空格的定义方式，在写入文本时如果仅按回车键，是无法控制文本的换行的（例如将一个邮政地址标记为一个段落，但在实际显示时地址的各个部分都显示在单独的一行上）。

可以使用 `br` 元素在文档中插入换行符。不过该元素应只在需要时作强制换行之用，绝不应被用于为段落之间设置更多的垂直间距——这里应该使用 CSS 更合适。

有时候使用预格式化的文本块可能比插入 `br` 元素更为方便。或者，如果某些文本的某个特殊部分需要自成一行，而这又只是样式问题，那可以将它们放在 `span` 元素中，并设置为按

块级元素显示。

举个例子，你们可以将在前面讨论 `address` 元素的部分中看到过的 **Opera** 公司的联系地址以如下代码表示：

```
<p>Our company address: </p>

<address>

 Opera Software ASA,
Waldemar Thranes gate 98,

 NO-0175 OSLO,
NORWAY

</address>
```

[查看源文件](#)

如果你是在编写 **XHTML** 文档而不是 **HTML** 文档，那么 `br` 元素就必须是自闭合的，像这样：`<br />`.

## 水平线

在 **HTML** 中可使用 `hr` 元素来创建水平线。该元素会在文档中插入一根水平线，用以表示文档不同部分的分界线。

尽管有人认为 `hr` 元素是一个不能被继承的非语义元素，而只是一个创造视觉和表象效果的元素，但在文献中使用这个元素是有一些先例的。在书籍的某一章（也可以说成是书籍的一部分）中，发生在不同时间和/或地点的场景间就常常出现水平线。此外，诗歌中也可以使用装饰性的断行将不同的小节分隔开来。

`hr` 元素是公认的在文档各部分之间制造分界线的标记，而不是一个新的头部元素。

`hr` 元素没有特别的属性，如果你对其默认样式感到不满意的话，应该使用 **CSS** 来对其进行样式化。

同样的，就如同换行符一样，如果你是在编写 **XHTML** 文档而不是 **HTML** 文档，就必须采用自闭合的形式——`<hr />`.

## 文档的改动（插入与删除）

如果某个文档在首次上线之后有所改动，你可以标注出这些改动，这样那些网站回头客或自动化程序就可以知道文档在何时做了哪些改动。

新文本（插入部分）应当被包裹在 `ins` 元素中，而已被移除的文本（删除部分）应当被包裹在 `del` 元素中。如果在文档中同一个地方既有删除又有插入，正确的做法是将被删除的文本

放在前面，然后才是新插入的文本。

`ins` 元素和 `del` 元素都可以带有两个属性，这些属性可以给文档编辑赋以更多的意义。

如果文档改动的原因在该页面上或网上其它地方已有指明，你就应该在 `cite` 属性中链接到那个文档或片段。这实际上就等于宣示：“因为这个原因才做出了这个改动。”

你还可以使用 `datetime` 属性指明改动发生的时间。该属性的值必须是一个符合 ISO 标准的时间戳，格式一般为“YYYY-MM-DD HH:MM:SS ±HH:MM”（更多信息可查阅维基百科）。

下面是一个用到了这两个属性的代码示例：

```
<p>We should only solve problems that actually arise. As
<del datetime="2008-03-25 18:26:55 Z"
cite="/changes.html#revision-4">Donald Knuth<ins
datetime="2008-03-25 18:26:55 Z"
cite="/changes.html#revision-4">C. A. R. Hoare</ins></p>

said: <q>premature optimization is the root of all
evil</q>.</p>
```

[查看源文件](#)

## 一些将在未来引入的 HTML 元素

我们在本文和其它一些文章中都已提到 **HTML 5 规范** 目前正在起草中。**HTML 5** 将是自 **HTML** 推出以来修正最大的一个新版本。通过实际研究目前网上对 **HTML** 的使用方式，而不是空想什么对人们可能有帮助，**HTML 5** 有望将目前只能算是约定的语义元素直接写入规范之中。

一些可以改善我们编写文档代码和使用文档方式的示例元素将被引入 **HTML**，其中包括：

- `header`——包含页面的顶部区域（页头），通常由 `logo` 和网页标题组成，还可以包含一个简短的“`about`”（简介）和全站导航，以及登录/退出/公司简介文档链接等。
- `footer`——包含页面的底部区域（页脚），通常由站内链接、版权声明和其它法律信息组成。
- `nav`——包含网页的主要导航链接。
- `article`——包含页面内主内容区域的部分，但不包括其它所有页面元素，如导航、页头、页脚等。

- `aside`——包含页面特定区域上的侧边栏信息，还可用于标记主要内容之中的重要引述或注释。

你们可以通过查阅 [HTML 5 规范](#)了解更多新引入的元素。

## 总结

在本篇文章中，我讲述了 `HTML` 中罕为人知、鲜为人用的一些语义元素。在下一篇文章中，我们将进一步讨论如何使用 `HTML` 中两个语义中性的元素，`div` 和 `span` 元素。

- [上一篇：HTML 表单基础知识](#)
- [下一篇：通用容器——div 和 span 元素](#)
- [目录](#)

## 作者简介



Mark Norman Francis 早在万维网诞生前，就在从事互联网领域的工作了，并一直持续到现在。目前他是全球最大网站 **Yahoo!** 的前端架构师，负责制定国际化 **web** 开发的最佳实践、代码标准和质量标准。

在加入 **Yahoo!** 前，他先后在 **Formula One Management**(F-1 管理公司)、**Purple Interactive**、伦敦城市大学从事过多种工作，包括 **web** 开发、后端 **CGI** 编程和系统架构等。他在 <http://marknormanfrancis.com/> 有一个 blog。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

## 22. 通用容器——div 和 span 元素

Posted 12/15/2008 - 16:53 by Lewis

- 网络标准教程

作者: Mark Norman Francis · 2008 年 7 月 8 日

- 上一篇: 罕为人知的语义元素
- 下一篇: 使用导航菜单创建多页面
- 目录

序言

在本篇文章中，我将向你们介绍何时以及如何使用 `HTML` 中两个不是用于描述内容的元素。`span` 和 `div` 这两个元素不会向它们所包裹的内容赋予任何的含义；反之，它们是在没有其它合适的 `HTML` 元素可用时，用于创建自定义的结构或元素分组的一种通用机制。然后就可以用 `CSS` 对这些结构或分组进行样式化，或通过 `JavaScript` 对其进行操作。尽管 `div` 元素未添加任何语义含义，但跟适当语义的 `class` 或 `id` 名称一样，`div` 元素可被认为是表示标记的结构性分区。

`div` 是“标签的最后一一种选择”，并且只能被用于没有其他合适的 `HTML` 元素的情况下，因为 `div` 对辅助技术或搜索引擎都没有任何意义可言。

本文的结构如下：

- 语义中性
- 内联元素和区块元素
- 分组内容的进一步说明
- 额外信息
- `Javascript` 和 `CSS` 的钩子
- 避免滥用 `div` 元素
- 不恰当的语意元素
  - 通用段落
  - 表现性元素
- 总结
- 练习题

## 语义中性

`HTML` 中绝大多数元素都是用于描述内容的，例如图像、列表、标题，或是辅助建立文档的，如头部、主体、链接、元信息等。不过有两个元素是没有给定的含义的。`HTML` 规范中说：

`div` 和 `span` 元素，连同 `id` 和 `class` 属性，为向文档添加结构提供了一种通用的机制。

这两个元素可被视为是 `HTML` 的“脚手架”。它们让你能对内容进行分类、在内容周围添加额外信息，并为添加样式和交互性提供钩子。不过它们并不向文档添加任何新的语义含义，无论是在它们包裹之中的内容还是它们自身。

## 内联元素和区块元素

通过前面的学习，你们已经知道 区块元素是指明文档结构的元素。`div` 元素，`division` 的缩写，是块级通用容器。它通常被用于包裹其它块级元素，将它们集合在一起（更多讨论请参见

下一部分)。它也可被用于把一串在逻辑上不适合放在其它块级元素之下的内联元素和/或文本组合在一起，不过只应作为最后的手段使用。

`span` 元素是内联通用容器。它同样可帮助指明文档结构，不过是用于集合或包裹其它内联元素和/或文本的，而不是块级元素。

初看之下，这两种不同类型元素之间的区分似乎是非常随意的。需要记住的是，差别之处在于内容的类型，以及在未添加任何样式时内容将如何显示。`div` 元素包裹的是一组块级元素——例如包裹一个标题加上一个链接列表，以创建一个导航菜单。`span` 元素包裹的是一组内联元素或(最常见)纯文本。这里“组”这个词很关键，如果 `div` 元素仅包裹一个块级元素，或 `span` 元素仅包裹一个内联元素，就没有必要使用它们。例如，请查看以下这段简单的标记中使用 `div` 和 `span` 元素的方式：

```
<body>

 <div id="mainContent">
 <h1>Title of the page</h1>
 <p>This is the first paragraph of content on my example page.</p>

 <p>This is the second paragraph of content on my example page. It is very similar to the first, but there is a special alert here
 that we want to colour and increase text size using CSS.
 It's not really standard emphasis - it's more just styling, so and are not really appropriate.</p>
 </div>
</body>
```

现在你可以使用它们的 `id` 属性来选择 `div` 和 `span` 元素中的内容，并使用 `CSS` 对它们进行专门的样式化和定位。

### 分组内容的进一步说明

通过查看互联网上很多网页的源代码，你都将看到大量的 `div` 元素以及写在 `class` 和/

或 `id` 中的一些常见含义——例如 `header`、`footer`、`content`、`sidebar`，等等。

`class` 和 `id` 的名称应该始终都是语义化的，也就是说，它们应该指内容的含义/作用，而不是仅仅指内容的视觉外观。例如 `sidebar`(边栏)和 `alertMessage`(警示消息)就是很好的 `class` 名称，而 `redLefthandColumn` (红色的左边栏) 和 `blueFlashingText` 就不是。如果以后你想将边栏的颜色改为红色，或将其位置从页面左边调整到页面右边，`redLefthandColumn` 这个 `class` 名称还有什么用呢？还有如果你想将警示消息从蓝色闪光改为绿色不闪光，`blueFlashingText` 这个 `class` 名称又有什么用呢？

这些分区在创建页面结构时提供了代码结构的可预测性，更重要的是，在后来重新查看 `HTML` 代码时，它们提供了你正位于页面哪个部分的线索。一个划分清晰的网页，其内容和意图几乎就是自文档化的。

为更清楚地说明这个道理，让我们在一个真正的网站上查看一下它的 `div` 结构——即 [dev.opera.com](http://dev.opera.com) 的主页。请注意，除了我特意包含进去的其它一些对创建页面结构来说很重要的元素，以下的示例代码中未包含任何内容。不过我主要考察的还是使用 `div` 元素来重现该网站的实际结构。在以下的代码中，请仔细阅读这些 `HTML` 注释——我加入了一些说明网页结构的 `HTML` 注释。在查看源代码的同时，请在你的浏览器新标签页或窗口中打开 [dev.opera.com](http://dev.opera.com) 的主页，这样你就可以通过参照该网站的外观来研究其结构。

```
<body>

<!-- First up we have a wrap div, which wraps the entire page, and allows more
precise control of it as a whole -->

<div id="wrap">

 <!-- This unordered list contains the list of links to all Opera's different
sites, which you can see across the very top of the page -->

 <ul id="sitenav" class="hidemobile">
 ...

 ...
 <!-- This is the search form - the search box you see at the top right of
the page -->

 <form action="/search/" method="get" id="search">
```

```
<div>

 ...

</div>

</form>

<!-- This unordered list contains the main navigation menu of the site --
the horizontal tab menu you see just below the main title graphic -->

<ul id="menu">

 ...

<!-- This nested div forms the structure of the login box, where you enter
your user name and password to log in to the site. You will only see this if
you are currently logged out. -->

<div id="loginbox">

 <div id="login">

 ...

 </div>

</div>

<!-- This series of nested divs is where the main content of the page is
contained - all of the article summaries that form the main bulk of the page
content -->

<div id="content2">

 <div id="main">

 ...

 <div class="major">

 ...

 </div>

 <div class="major">

 ...

 </div></pre>
```

```
 ...
 </div>
 </div>
 <!-- This div contains the sidebar of the page - the article categories,
the latest comments, etc -->

 ...
<div id="side">
 ...
</div>
 <!-- This div contains the page footer, which is where you'll see the
copyright message, and various links at the bottom of the page. -->
<div id="footer">
 ...
</div>

 <!-- The end of the page - this is the closing tag for the wrap div -->
</div>
</body>
```

## 额外信息

一些内容具有对用户代理和其它解析器有用的额外信息，这类额外信息需要通过一个属性来传达。`span` 元素为将此类额外信息附加到网页内容上提供了很好的方式，就如你们将在下面所看到的那样。

在一个文档中出现另一种不同的语言，就是一个好例子。例如：

```
<p><q>Plus ça change, plus c'est la même chose</q> she said.</p>
```

尽管主文档的语言为英语，但这段引用是法语。通过使用 `lang` 属性，这一点得以被指明，如下所示：

```
<p><q lang='fr'>Plus ça change, plus c'est la même chose</q> she said.</p>
```

在以上这个例子中，由于整句引语都是法语，要标记出语言的变化是很容易的，使用 `q` 元素将引语包裹起来即可。不过在有些情况下，当没有适当的语义化元素可用时，就只能采用 `span`

或 `div` 元素，例如：

```
<p>A screen reader will read the French word chat (cat) as chat (to talk informally) unless
it is properly marked up.</p>
```

在以上这个例子中，单词 `chat` 的第一个实例，作为英语文档中出现的一个法语词，必须指明其差异才不会被解释为英语单词 `chat`。这时就可以使用 `span` 元素将“`chat`”这个法语词包裹起来，因为其它 `HTML` 元素都不适于用来包裹这个法语词（我们并不想强调这个词，它既不是一句引语，也不是一段代码，等等）。同时作为一个句子内的单个词语，它属于内联级别。因此上面那段代码最好写成下面这个样子：

```
<p>A screen reader will read the French word chat (cat) as chat
(to talk informally) unless it is properly marked up.</p>
```

使用 `微格式` 标记网页内的一般数据格式时，也用到了同样的技术。关于微格式更多的信息，可在 [dev.opera.com](https://dev.opera.com/) 上关于更高级的 `HTML` 的文章中找到。

### JavaScript 和 CSS 的钩子

我已经讲过，如何使用 `div` 和 `span`，连同 `id` 和 `class` 属性，为页面内特定部分的内容提供 `CSS` 样式化和定位的钩子。在文档中使用 `JavaScript` 时，也可以这样做。

如果文档内某一给定的元素需要由 `JavaScript` 查找和处理，通常的做法是为该元素赋一个 `id` 然后使用 `getElementById` 函数来找到这个元素。关于 `JavaScript`，在本课程的后面部分将详细讲述。

### 避免滥用 `div` 元素

有一点值得引起注意，那就是在 `web` 开发人员圈子里通常被称为“`div-itis`”的对 `div` 元素的滥用现象。

通过大量的嵌套 `div` 或 `span` 元素来添加样式确实很方便，但还是应尽量避免这种诱惑。在多数情况下，为文档内现有元素附加样式或 `JavaScript` 功能是可以做到不必引入 `div` 或 `span` 元素的。作为通用容器的 `div` 或 `span` 元素应作为最后的手段使用——也就是说，最好是先尽量以语义化元素来编写网页，仅在确实需要时才加入通用容器元素。

### 不恰当的语义元素

在这个部分，我们将探讨一些使用 `HTML` 标记内容时的常见错误，如果可能的话，应尽量避免犯这些错误。

#### 通用“段落”

有时候人们倾向于将任何文本块都包裹在 `p` (段落)元素之中，这其实是不对的。正如我在

前面那篇 关于标记文本内容的文章中已指出的那样：

那些不成句子、仅由几个词组成的文本，不应被标记为一个段落。

对于那些不能被其它具有语意的 HTML 元素所涵盖的不连贯的文本内容，将它们包裹在 `div` 或 `span` (视具体情况而定)元素中，才是正确的选择。

### 表象性元素

在互联网上经常看到一种不良建议，即建议使用短的表象性元素如 `b` 或 `i` 元素来代替 `span` 元素作为通用容器，其理由无外乎以下两个原因中的一个：

- `b` 或 `i` 元素比 `span` 元素少 3 个字节，这样在 HTML 和 CSS 中都节省带宽。
- 内容所需的样式仅涉及外观，因此在这种情况下使用“表象性”元素是可以的。

第一个理由并没有错，但所节省的带宽可以忽略不计（除非你使用了惊人数量的表象性元素）。而且 web 服务器在通过互联网将文档发送到浏览器上之前，都要采用现代技术对文档进行压缩，这已让文档的体积大大缩小，远超过任何人工缩写所能达到的程度。

第二个理由则显示出他们对 HTML 语境中表象性元素的真正含义缺乏正确的理解。表象性元素描述的是它们所包含的内容所应有的外观（如 `b` 就仅仅是指“包含的文本以粗体显示”）。而不是为它们所包含的内容提供进行样式化的通用钩子。

如果一个段落中的一小段文本需要被样式化或被 JavaScript 处理，且没有适用的语义元素来包裹该文本，那么使用 `span` 元素来包裹它们就是唯一正确的选择。

### 总结

我已深入地讲述了 `span` 和 `div` 元素的各个方面——在读完这篇文章后，你们应该已经很好地了解了这两个元素的功能，并能自如地使用它们了。在后面关于 CSS 的文章中，我将更深入地讲述如何使用它们进行页面布局及其它内容。

### 练习题

- `div` 元素和 `span` 元素的区别何在？
- 给出这两种元素在网页内的两个主要用途。
- 查看你最喜爱的网站内某一个网页的源代码，并思考其页面结构。该页面是否使用了大量的 `div` 和 `span` 元素？你能发现有不正确地使用这两个元素的地方吗？如果有，该如何修正？
- 上一篇：罕为人知的语义元素

- [下一篇：使用导航菜单创建多页面](#)
- [目录](#)

## 作者简介



Mark Norman Francis 早在万维网诞生前，就在从事互联网领域的工作了，并一直持续到现在。目前他是全球最大网站 **Yahoo!** 的前端架构师，负责制定国际化 **web** 开发的最佳实践、代码标准和质量标准。

在加入 **Yahoo!** 前，他先后在 **Formula One Management**(F-1 管理公司)、**Purple Interactive**、伦敦城市大学从事过多种工作，包括 **web** 开发、后端 **CGI** 编程和系统架构等。他在 <http://marknormanfrancis.com/> 有一个 blog。

---

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

---

## 23. 使用导航菜单创建多页面

Posted 12/15/2008 - 16:54 by Lewis

- 网络标准教程

作者: Christian Heilmann · 2008 年 9 月 26 日

- 上一篇: 通用容器——div 和 span 元素
- 下一篇: 检验你的 HTML
- 目录

### 序言

在本篇文章中，我们将讨论网站导航和菜单。你们将了解到不同类型的菜单及如何在 HTML 中创建菜单。本文还将涉及导航菜单的可用性与可访问性。本文不会讲述对导航菜单进行样式化的内容，但会为本系列课程后面对应的 CSS 教程打下基础。本文中所使用的代码示

例，可 [点击这里](#) 下载本文的内容目录如下：

- **HTML 菜单工具——链接、锚记和列表**
- 对灵活性的需要
- 不同类型的导航菜单
  - 页面内导航（目录）
    - 让网站访问者感到“我在这里”
    - 每次你应给用户几个链接选项？
  - 上下文导航菜单
  - 网站地图
  - 分页
- 当列表不够用时——图像映射和表单
  - 使用图像映射设置热点
  - 使用表单节省屏幕空间并防止链接过多
- 导航菜单该放在什么位置，提供跳过导航菜单的选项
- 总结
- 练习题

### **HTML 菜单工具——链接、锚记和列表**

HTML 中常用的菜单和导航术语有几种不同的类型，它们都与 `link` 和 `a`（锚记）元素有密切的联系。概括地说：

- `link` 元素描述多个文档间的关系。例如，通过链接你可以告诉用户代理当前的文档是一个由多个文档构成的课程的一部分，以及哪个文件是课程的目录。
- 使用锚记（也称作 `a`）元素，你可以链接到另一个文档或当前文档的某个特定部分。锚记链接将由网站访问者使用任何可用的方式（鼠标、键盘、语音识别器……）激活，而不由“用户代理”自动跟踪。

如果你尚未阅读前面 [关于链接的文章](#) 和 [关于列表的文章](#)，我建议你去读一遍，为了避免重复，我将本文涉及到的一些信息放在这两篇文章中了。

锚记/链接并不会自己变成菜单——你需要对它们进行结构化和样式化，让浏览器和用户知道它们的功能是导航菜单，而不只是一组随机链接。如果网页的先后顺序不重要，你可以使用一

个无序列表，如同这个 [无序列表菜单示例](#)一样。

请注意，我为 `ul` 元素赋予了一个 `id`。这是为以后使用 `CSS` 对 `ul` 元素进行样式化，以及使用 `JavaScript` 为它添加专门的行为所提供一个钩子。使用 `id`，可以让其他技术很方便地选定 `HTML` 文档中的某个特定元素。

如果访问者遍历全部文档的先后顺序很重要，则需要使用一个有序列表。举个例子，假如你有一个由多个文档组成的在线课程，且课程是循序渐进的，那么你就可以使用一个有序列表，如同这个 [有序列表示例](#)一样。

列表是创建导航菜单非常好的方式，原因包括：

- 所有 `HTML` 都包含在一个单独的列表元素（如 `ul` 元素）中，这意味着你可以使用 `CSS` 中的层叠为各个元素设置不同的样式，而且便于用 `JavaScript` 自上而下地访问某个元素。
- 列表可以嵌套，这意味着你可以容易地创建多层次的导航结构。
- 即使没有为列表应用任何样式，浏览器对 `HTML` 的渲染也能让导航菜单一目了然，并易于让访问者知道这些链接构成一个逻辑单元，都是导航菜单的一部分。

在嵌套列表时，应当把嵌套的列表嵌 `li` 元素之中，而不是之后。这里是两个 [正确的](#)和[不正确的](#)嵌套列表代码的示例。

请注意，在浏览器中这两个列表显示出来的样子是一样的。但浏览器中显示出来的样子绝不应是衡量代码质量的标准。像上面那个不正确示例中的无效 `HTML` 结构将很难被样式化或是使用 `JavaScript` 为其添加行为，也难以转换成其他格式。嵌套的 `UL` 元素，其结构应为  
`<ul><li><ul><li></li></ul></li></ul>`，而绝不应是  
`<ul><li></li><ul><li></li></ul></ul>`。

### 对灵活性的需要

网站的导航菜单一般不会长久保持不变——随着新功能的添加和用户群的增长，网站也会有系统地逐步扩展。因此，当你在创建导航菜单时，应随着站点发展而为菜单项的添加和删除留有余地，并为导航菜单被翻译成不同的语言（因此链接的长度将发生变化）预留出空间。此外，你还可能遇到这种情况：网站的导航菜单的 `HTML` 是由服务器端语言动态地创建的，而不是由静态的 `HTML` 创建的。当然这并不意味着 `HTML` 知识就已过时；实际上它甚至变得更为重要了，因为你仍然需要为服务器端的脚本创建 `HTML` 模板。

### 不同类型的导航菜单

当你在构建不同的网站时，有多种不同类型的导航菜单需要用 `HTML` 创建。多数导航菜

单可使用列表来创建，不过有时候界面限制会让你不得不采用其他的方式来创建导航菜单（后面有详细介绍）。使用列表方式可创建的导航菜单包括：

- **页面内导航**：例如单个网页的目录，带有指向页面内不同部分的链接。
- **网站导航**：整个网站（或网站某个部分）的导航栏，带有指向网站内不同网页的链接。
- **内容上下文导航**：一张链接列表，指向与当前页面的主题密切相关的网页（同一网站上或其他网站上的网页）的链接列表。
- **网站地图**：指向网站所有不同页面的大型链接列表，并且为便于访问，它会被划分为相关的子列表。
- **分页**：指向与当前文档一起作为某个完整文档的章节或部分的其他网页的链接，例如，一篇文章的第 1 部分、第 2 部分、第 3 部分等。

### 页面内导航（目录）

在前面那篇讲述链接的文章中，我已经介绍过一部分关于页面内导航的内容，以下我将详细介绍页面内导航的功能以及如何创建好的页面内导航。

在 [页面内导航示例](#) 中，我使用了一个链接列表来指向页面内其它内容的锚记。为把锚记连接起来，你需要为导航的锚记元素赋予一个 `id` 属性以及一个 `href` 属性，该属性由一个 `#` 符号和该链接要指向的那个元素的 `id` 属性值相同的名称构成。页面的每一部分都有一个按相同方式工作的“返回导航菜单”的链接，只不过链接指向的是导航菜单本身。

从技术上讲，要让页面内导航正常工作，像上面这样做就已经够了。不过由于 [Internet Explorer](#) 浏览器存在一个令人郁闷的漏洞，你还不得不做一点工作。

你们可以按以下步骤，来对 [Internet Explorer](#) 浏览器的这个漏洞进行试验：

1. 用 [IE6](#) 或 [IE7](#) 打开这个示例文档
2. 不使用鼠标，而是使用键盘来导航文档。你可以使用 `TAB` 键从一个链接跳转到另一个链接，使用 `Enter` 键激活链接——以此跳转到链接所指向的页面部分。
3. 当你这样做的时候，似乎一切正常——浏览器往下滚动到你想要跳转到的页面部分。
4. 这时如果你再次敲击 `TAB` 键，浏览器应该将你带回到（聚焦在）你选择的页面部分之内的第一个链接，但 [IE](#) 浏览器会把你带回页面顶部的导航菜单！

造成这个问题的原因非常复杂，而且涉及到 [IE](#) 浏览器对自己的一个叫做 `hasLayout` 的特殊属性的处理方式。你可采用几种方法来触发这个漏洞，参见 [Ingo Chao](#) 那篇“关于布局处理”的出色文章。最简单的方法是使用 [CSS](#) 设置 `div` 元素的宽度，我在示例中就是这么做的。这

就是 IE 想要的——让锚记处于带有 `hasLayout` 属性的元素之内。

不得不做这样的额外工作让人有点郁闷，不过如果你想为页面不同的区段设置不同的样式的话，这对你还是有帮助的。这也有助于解决 HTML 中标题的问题：即标题中不包含自己所应用到的区段，并且假定一个标题至下一个标题之间的所有内容都属于同一个页面区段。这样不加入 `<div>` 元素就无法为页面的不同区段设置不同的样式。其他一些标记语言提倡采用一个包裹有 `<title>` 标签的 `<section>` 元素，这与可让你标记表单的某些部分，就如同 `<fieldset>` 元素和 `<legend>` 元素一样。

附注：在 Opera 浏览器中使用键盘进行页面导航的方式稍微有些不同——你可以试试在 Opera 浏览器中打开以上那个示例页面，然后按住 Shift 键并使用箭头键来导航各个链接（这种方式也适用于导航表单元素）。这被称为空间导航。

## 网站导航

网站导航栏很可能是你所需要创建的最常见的导航菜单。它是整个网站（或网站某个部分）的导航菜单，指明网站访问者可选择访问的链接，并显示网站的结构。采用列表是创建网站导航菜单最好的方式，就如你在这个 [网站导航菜单示例](#) 中所看到的那样。

这个示例中并没什么让人惊奇的东西，至少从纯 HTML 的角度来看是没有什么好惊奇的。在后面的课程中，我们将讨论使用 CSS 为这类导航菜单添加样式以及使用 JavaScript 为它们添加行为。如何在导航菜单中强调显示当前页面、如何让用户感到自己正处于一个特别的位置、并且他们正在移动位置（即使他们只是端坐在电脑前，当然如果他们使用移动设备上网则确实是在移动位置），这些是需要考虑的重点问题，下面我将讨论这些问题。

### 让网站访问者感到“我在这里”

Web 开发和导航的一条基本原则就是当前文档绝不应与自身链接，而应与导航菜单上的其他条目有明显的区别。这会使访问者愿意在网站上停留，并知道自己现在正处于网站的哪个位置上。除极少数例外的 web 应用程序、博客上的永久链接以及所谓“只有一个页面”的网站等，在 99% 的情况下，创建指向访问者正在阅读的文档的链接都是多余的，况且这么做还会把访问者搞糊涂。

在前面那篇关于链接的专题文章中，我已经讲过，链接是一项协议，也是一项义务：通过链接你向访问者提供了一种方式，使得他们能获取所需的信息，但你在创建链接时需要格外小心——如果链接没有为访问者提供他们所想要的信息，或是导致了意料之外的行为的话，你将失去自己的信誉和访问者对你的信任。例如，你提供了一个指向当前文档的链接，激活这个链接将重新载入这个文档。用户是不希望有这样的链接的——点击它有什么用呢？这只会把用户搞糊涂。

这就是为什么绝不应通过导航菜单链接到当前页面的原因之所在。你可以将它从导航菜单中完全删掉，也可以不再将其作为链接，只仅仅强调显示当前页面（例如把它包裹在 `strong` 元素之中）——这样就可以告诉用户他们正在访问当前页面，并告诉盲人访问者这是重要内容，而且是导航菜单中的当前词条——你可以 [点击这里查看强调显示当前页面的示例](#)。

### 每次你应该给用户几个链接选项？

另一个值得考虑的问题是你想给用户几个选项。你可以看到很多网站的导航菜单都试图确保网站的每一个页面都可以通过同一个导航菜单来访问。这就是脚本处理和 CSS 技巧之所在——让导航菜单的一些部分隐藏起来，直至用户选中了某些区域时才显示（有时被称为下拉菜单）。仅从技术角度上看，这是聪明的做法，但是使用这种方法存在以下一些问题：

- 不是所有访问者都能按预期的那样使用这种聪明的花招，例如，使用键盘的用户就不得不使用 TAB 键在页面上所列出的全部链接中一个一个地寻找他们想访问的某个链接。
- 使用这种方法创建导航菜单，你需要向网站内的每个文档添加许多 HTML 代码，对于许多页面来说这些代码的绝大部分都是多余的。如果我仅需通过 3 个层级的链接就能访问到一个我想阅读的文件，那么我就不需要看到链接到第 4、5、6 层级的选项。
- 如果你同时为用户提供了太多的选项，可能会让他们感到无所适从——人们都不喜欢做出决策。想一想从一个很长的餐馆菜单中选择想点的菜品可能要花多长时间，你就明白这个道理了。
- 如果页面上内容不多，却有大量的链接，那么搜索引擎就会假定该页面内没有多少有效的信息，也就不会去关注这个页面，从而导致搜索网络的时候很难找到这个页面。

总的来说，在导航菜单上放入多少个条目是完全由你自己决定——不同的设计会要求做出不同的选择——不过如果你拿不准的话，你应当试着仅在导航菜单中列入那些指向网站各主要部分的链接。因为如果需要的话，你总是可以再创建子菜单的。

### 上下文导航菜单

上下文导航菜单是基于当前文档的内容的相关链接，提供与当前正在访问页面相关的更多信息。一个经典的例子就是在新闻的底部经常可以看到的“相关文章”链接，如图 1 所示。

#### Related Articles: Party Loans Investigation

- › [Top minister denies new funding claims](#) AFP - 49 minutes ago
- › [Health secretary denies new funding claims](#) AFP - Sunday, January 27 04:47 pm
- › [Brown enjoying 'world's best job'](#) Press Assoc. - Sunday, January 27 12:17 pm
- › [Health secretary 'failed to declare donations'](#) AFP - Sunday, January 27 09:01 am
- › [PM denies 'dithering' over Hain](#) Press Assoc. - Friday, January 25 04:34 pm

#### Related Articles: Politics

- › [Top minister denies new funding claims](#) AFP - 49 minutes ago
- › [Johnson 'took third party donations'](#) Epolitix - Sunday, January 27 10:33 am
- › [Rock collapse 'has hit UK competitiveness'](#) Epolitix - Sunday, January 27 03:18 am
- › [Purnell to press private sector role](#) Epolitix - Sunday, January 27 02:09 am
- › [Ashdown out of running for Afghan role](#) Epolitix - Sunday, January 27 01:56 am

图 1：上下文导航菜单的一个例子——在新闻的底部放置的相关新闻条目的链接。

这与在软件用户界面中的上下文菜单略有不同，软件中的上下文菜单会根据不同的访问目的提供不同的选项（就如同在桌面应用程序中可以看到的右键菜单，或者按住 **Ctrl** 单击左键的弹出菜单那样，它会根据当前鼠标的位置而提供专门的选项）。

在网站上使用上下文导航菜单，是推广网站其它部分上的内容的好方式，但从 **HTML** 方面来讲，它们只是另一个链接列表。

## 网站地图

网站地图可能正是你所期望的——网站所有网页（如果网站是大型网站，则是网站的主要部分）的地图。网站地图让访问者可以大致了解网站的总体结构，并访问他们想要快速访问的页面——即使该页面在页面层级中处在很深的位置。

对于向访问者提供在网站内找不到方向时的后备选项，以及为那些匆忙的访问者提供快速的入口，网站地图和站内搜索都是非常好的方式。

从 **HTML** 代码的角度看，网站地图可以是一个大型的充满链接的嵌套列表，或者——对大型网站而言——则是网站各部分的标题（带有各部分自身层级的嵌套链接）或者是每个部分的搜索表单。

## 分页

当需要为分割成多个页面的大型文档提供导航方式时，分页菜单是必须用到的。在大型的图像库网站或搜索结果网页（如 **Google** 或 **Yahoo** 搜索）中，你们都可以看到分页菜单的例子。

分页菜单与其他类型的导航有所不同，因为它们通常是链接回同一个文档的——不过有一个带有更多信息（如从哪一个页面开始）的链接。图 2 中就是一些分页菜单的例子：



图 2：分页菜单可以让访问者在浏览大型的数据集时，知道自己正在访问的页面的位置。

分页菜单的 `HTML` 代码不是什么开创性的东西——这里你又一次提供了一个包括当前链接的链接列表（指明正在显示的是哪一个数据块或者当前页面所处的位置），其中当前链接未与自身相链接，而是被强调显示（例如通过 `strong` 元素）。

分页菜单和网站导航菜单的区别在于，分页菜单要使用大量的编程逻辑。依据当前页面在数据集中的位置，你需要显示或隐藏前一个、后一个、第一个、最后一个链接。如果确实有数量非常庞大的信息需要导航，你会希望提供到标志性页面如第 100 页、第 200 页等页面的链接，以及其他更多的选项。在这种情况下，你可能不大会像那些在 `HTML` 中的菜单一样创建硬编码菜单，而可能会选择在服务器端创建它们——但编写规则还是一样的，即当前页面不应该与自身相链接，也不应有无效链接。

### 当列表不够用时——图像映射和表单

在 99% 的情况下，使用 `HTML` 无序列表或有序列表创建导航菜单就已经足够了，尤其是因为列表的逻辑顺序和嵌套还允许使用 `CSS` 为列表设置很好看的样式。不过在另一些情况下，可能需要采用不同的设计技巧。

#### 使用图像映射设置热点

技巧之一就是客户端的图像映射。图像映射通过将图像的区域变为可以链接到不同文档的交互式区域，由此可以将一幅图像变成一个导航菜单。本部分所附的 [图像映射的示例](#)，就把一个图像变为可点击的导航菜单。你可以对这个示例进行试验，请点击上面那个链接，并点击图像中的三角形（如图 3 所示）的各个部分。

# Web Developer Skillset

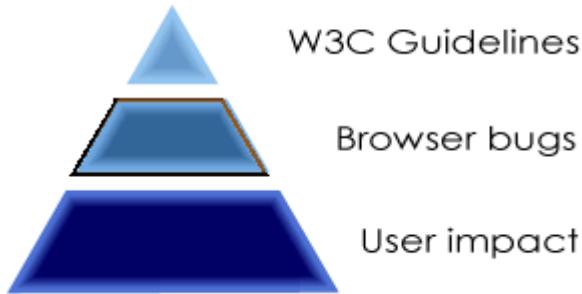


图 3：通过使用 `area` 元素定义图像映射，你可以把一个图像的区域变为交互式元素。

你可以通过将一个图像映射定义为不同的区域（也被称为热点），把任何一个图像变成一个导航菜单。当你为图像映射赋予一个 `name` 属性，并在 `img` 元素中使用 `usemap` 属性时，就把图像与映射关联了起来。请注意，这种做法很像页面内的链接，意味着你需要在 `usemap` 属性的值前加上 `#` 符号。

每个区域应该有以下几个属性：

`href`

定义该区域须链接到的 URL 地址（或同一文档内的目标）

`alt`

定义在图像无法找到或用户代理不支持图像时所显示的替代文本

`shape`

定义区域的形状。长方形可使用 `rect`，圆形可使用 `circle`，定义为多边形的不规则区域可使用 `poly`。

`coords`

定义图像中那些应成为热点的坐标——坐标值从图像的左上角开始算起，并可以像素或百分比衡量。对于长方形，你只需定义其左上角和右下角；对于圆形，你只需定义圆心和半径；对于多边形，你需要提供所有顶点的列表。

用 HTML 定义和加入图像映射是比较枯燥的，因此一些图像处理工具如 `Adobe Image Ready` 或 `Fireworks` 都提供了可视化创建图像映射的选项（它们会替你生成 HTML）  
使用表单节省屏幕空间并防止链接过多

另一种技巧是使用 `select` 元素创建一个表单。你可以把不同的页面定义为 `select` 元素内的选项，这样访问者可以选定一个选项，然后提交表单，以跳转到不同的页面。你可以 点击

此处查看实际运行的表单导航菜单的示例。

使用表单菜单最明显的特点是可以在不占用大量屏幕空间的情况下，提供大量的链接选项。浏览器会将表单菜单渲染成一行，如图 4 所示。



图 4：表单菜单在屏幕上只占据一行。

你还可以对表单菜单进一步进行处理，使用 `optgroup` 元素将菜单选项分组，如这个 `optgroup` 示例所示的那样。

在分组后，表单菜单将变为如图 5 所示的样子，其中分组的名称是不能选定的选项。

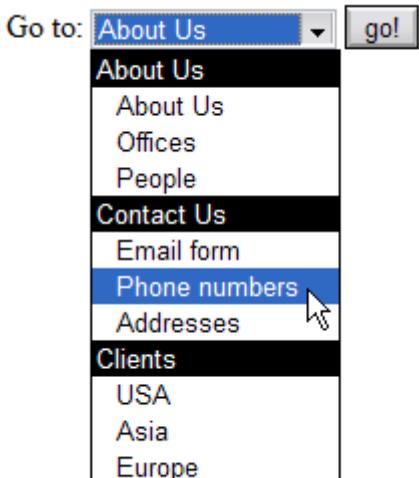


图 5：表单菜单可以将选项分组，以告诉访问者哪些选项属于同一个分组。以上为表单菜单在 Opera 9.5 浏览器中的渲染效果。

这个技巧具有只占用很小的页面空间的优点，不过也意味着你需要用一个服务器端脚本把访问者带到所选定的页面。你也可以使用 `JavaScript` 来让菜单中的链接生效，但你不能假定 `JavaScript` 就一定可用——你得确保用户在 `JavaScript` 禁用的情况下还是能够使用这个菜单。

使用表单菜单还有另一个不那么显而易见的优点，即你不再需要在同一个文档内提供太多的链接。这意味着你就不会让那些使用辅助技术上网的用户无所适从了（他们常常是只能面对一

个又长又大的列表中的链接)。同时也不会因链接到文本的比例而让文档显得像一个站点地图，从而使搜索引擎认为那些链接是没有价值的。不过很多辅助技术都可以生成一个页面链接的地图，如果重要的链接都在表单菜单中，那么使用辅助技术的用户就有可能失去发现这些链接的机会。因此，一个好的做法是既提供指向主要目标页面的锚记链接，又使用 `select` 元素创建表单菜单，以提供更多的链接选项。这样访问者将可以使用这些链接，而机器如搜索引擎机器人则不必知道这些链接的存在。

### 导航菜单该放在什么位置，提供跳过导航菜单的选项

关于 HTML 导航菜单，最后需要指出的一点是，菜单放在页面的哪个位置是相当重要的。因为访问者可能没有滚动机制或可能有视力障碍，只能依靠键盘导航来访问网站。在他们载入文档时，首先将遇到的是文档地址和标题，然后至上而下地阅读文档，在遇到每个链接时都会停下，并询问访问者是否要跟踪该链接。其他选项还包括获取一个所有链接的列表或从一个标题跳转到另一个标题。

如果导航菜单位于文档顶部，那么它将是用户首先遇到的页面元素。在能读到任何内容之前，还得一个一个地跳过 15 或 20 个链接，这会让他们感到相当郁闷。要解决这个问题，有两个变通方法：第一，你可以把导航菜单放在文档主要内容之后（如果你愿意，依然可以使用 CSS 将菜单放置在页面顶部）。第二，你可以提供一个跳过链接。跳过链接是一个放置在主菜单之前，能直接链接到内容开始之处的链接，它可以让访问者跳过菜单而直接阅读内容。你还可以在文档的末尾添加另一个“回到菜单”的链接，让访问者可以轻松地返回文档顶部。[点击这里](#) 可查看跳过链接的示例。

跳过链接不仅对残疾人有用，而且也有助于使用小屏幕移动设备上网的用户更方便地进行站点导航。

### 总结

在本篇文章中，我介绍了可能用 HTML 来编写的不同类型的导航菜单。我们讨论了：

- 为什么带有锚记的列表是定义导航菜单的最佳 HTML 构造
- 为什么说不要把导航菜单看成是一成不变的，而要为其预留改动的空间是很重要的
- 页面内导航：链接到当前文档的各个部分，并返回导航菜单
- 网站导航：提供一个显示当前站点的网页及它们的层级结构的导航菜单；我还介绍了为什么强调显示用户正在访问的当前页面是很重要的
- 上下文导航：提供链接到当前网站（或其他网站上）的相关页面的链接
- 网站地图：向那些在网站内找不着北的访问者或有特定需要的访问者提供的再定位工具

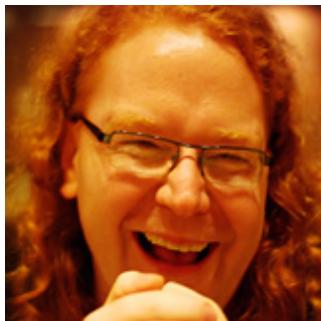
- 分页：让访问者可以导航一个被分割为多个页面的文档
- 图像映射：通过在图像上添加热点来创建图像菜单
- 表单菜单：提供很多链接选项，而且不会占用大量页面空间，也不必在页面上放置太多链接
- 跳过链接以及导航菜单的摆放位置

在本教程后面的 **CSS** 部分，我们还将回到上述某些内容，来介绍如何使用 **CSS** 让 **HTML** 结构的外观显得好看，并更明显地作为菜单展示在访问者的眼前。

### 练习题

- 为什么说将导航菜单标记为列表是一个好主意？
  - 当你在设计一个导航菜单时，你需要为未来做哪些规划？
  - 使用 `select` 元素创建表单有哪些优点，又可能导致出现哪些问题？
  - 你用 `area` 元素来定义什么，该元素的 `nohref` 属性有什么用处？（这未在本篇文章中介绍，你需要上网搜索资料并研究）
  - 跳过链接有哪些优点？
- 
- 上一篇：通用容器——`div` 和 `span` 元素
  - 下一篇：检验你的 **HTML**
  - 目录

## 作者简介



Chris Heilmann 从事 web 开发工作已达 10 年，之前他曾从事电台新闻工作。他在英国做过 Yahoo! 的培训师和高级开发工程师，同时负责监控面向欧洲和亚洲的前端代码质量。

Chris 的博客是 [Wait till I come](#)，在许多社会网络网站上都可以通过输入“codepo8”来找到他。

本文采用的授权是创作共用的“署名-非商业性使用-相同方式共享 2.5 通用许可”。

## 24. 检验你的 HTML

Posted 12/15/2008 - 16:54 by Lewis

- 网络标准教程
- 作者: [Mark Norman Francis](#) · 2008 年 9 月 26 日
- 上一篇: [使用导航菜单创建多页面](#)

- 下一篇：可访问性基础知识
- 目录

## 序言

你应该已经编写了几个 **HTML** 网页，它们在浏览器中显示出来的样子也还不错，不过其中的代码可能并不完全正确。要着手查找错误之处，并确保这些网页（以及所有你将来要编写的网页）能跨浏览器正常显示，而且没有任何差错，最好的方法是什么？

答案就是代码检验！在万维网联盟（**W3C**）及其它一些地方有很多可用的工具，这些工具可以让你对自己网站的代码进行检验。最常用的三种检验工具是：

- **W3C 标记检验器**：这个标记检验器会首先查看你提交给它进行检查的文档所使用的（X）**HTML** 文档类型，然后检查整个文档，并指出文档中哪些地方不符合该文件类型（比如 **HTML** 中存在错误的地方）。
- **W3C 链接检测器**：这个链接检测器会仔细地检查你所提交给它的整个文档，并测试文档内的所有链接，以确定不存在死链接（比如 **href** 属性的值指向的是不存在的资源）。
- **W3C CSS 检验器**：你可能已经猜到了，这个 **CSS** 检验器将遍历整份 **CSS**（或 **HTML/CSS**）文档，并检查其中的 **CSS** 是否正确地遵守了 **CSS** 规范。

在本篇文章中，我只会介绍上述的第一个检验器，即 **W3C 标记检验器**，内容包括如何使用这个标记检验器，以及如何分析该检验器提交给你的几种具有代表性的检验结果。链接检测器的工作原理和结果是显而易见的，而在阅读了本篇文章和本系列教程中后面的 **CSS** 专题文章后，**CSS** 检验器的原理和结果也就不言自明了。

本文的结构如下：

- 错误
- 什么是检验？
- 为什么要检验？
- 不同的浏览器以不同的方式解析无效 **HTML**
  - 怪异模式
- 如何检验你的网页
  - **W3C HTML 检验器**
- 总结
- 更多检验工具

- 练习题

## 错误

在计算机编程中，一般来说代码的问题包括两大类：

- 语法错误——语法错误是指那些会导致计算机无法正确地执行或编译程序的代码书写错误。
- 编程（或逻辑）错误——这类错误是指代码没有完全反映出程序员的意图。

在多数编程语言中，第一类错误是很容易被发现的——出现此类错误时，程序会拒绝执行或编译代码，直至错误被修正为止。这样就让程序员在冥思苦想“程序为什么没有按我想象中的那样运行”时，能很容易地发现和修正这类错误。

`HTML` 不是编程语言。网页中存在的语法错误一般并不会导致浏览器拒绝打开网页（`XHTML` 在这方面要比 `HTML` 严格得多——至少在特意被用作 `application/xhtml+xml` 或 `text/xml` 时是如此——此外一些文档类型不允许使用某些特定类型的 `HTML` 元素）。但这（浏览器不会拒绝打开存在语法错误的网页）也是 `HTML` 得到迅速采用和推广的最主要原因之一。

第一个 `Web` 浏览器，`WorldWideWeb`（由 `Tim Berners-Lee` 编写）同时也是一个编辑器，可以让人们在事先没有学习过 `HTML` 的情况下创建网页。这个编辑器会生成无效的 `HTML`。虽然这是可以被修正的，但它建立起了迄今所有浏览器都遵循的一个重要先例——比起向那些不懂或者理应修正网页错误的人抱怨网页中的问题，让人们能够访问网页上的内容显得更为重要。

### 什么是检验？

尽管浏览器会接受使用糟糕的代码（我们称为无效代码）所编写的网页，并通过尽量猜测网页作者的意图来尽可能好地渲染网页，但检查 `HTML` 是否编写正确仍然是可能的，而且这是不折不扣的一个好主意，我们称之为“检验”`HTML`。

检验程序将网页中的 `HTML` 代码与相对应的 文档类型的规则相比较，并告诉你这些代码是否违反了某些规则，以及在哪些地方违反了规则。

### 为什么要检验？

一些 `Web` 开发员认为，只要网页在浏览器中看起来还不错，即使其源代码没有通过检验也没关系。他们认为检验是一个理想的目标，而不是一个黑白分明的问题。

这种看法也不能说完全没有道理。`HTML` 规范本身尚不完善，而且现在看来也已经很过时了。一些存在争议的用法（如 让一个有序列表不以“1”起始）也会被检验器认定是无效的 `HTML`。

尽管如此，正如一句格言所说的：

先弄懂规则，才会知道该如何适切地打破成规。

你需要检验你所编写的 HTML，有两个非常有力的理由。

- 人无完人，同样也没有完美的代码——我们都会犯错，如果你清除了所有的代码错误，你所编写的网页的质量就会提高（如，更具有一致性）。
- 浏览器会变化。将来的浏览器在解析无效代码时，可能会更加严格，而不是更宽松。

检验可以作为你的预警系统，用于发现插入在页面内的缺陷，这些缺陷往往会以一种有趣而又无法判断的形式显露出来。当浏览器遇到无效 HTML 时，它不得不根据经验去猜测代码作者的意图——这时不同的浏览器就会得出不同的答案。

### 不同的浏览器以不同的方式解析无效 HTML

有效的 HTML 是你与浏览器制造商之间唯一的约定。HTML 规范规定了你应该如何编写 HTML，以及浏览器应该如何解析你所编写的文档。近年来，符合 Web 标准的浏览器已能实现以下这种效果：只要你编写的是有效的代码，所有的主流浏览器都会将其解析为一样的结果。不过这基本上还只是适用于 HTML 代码，而浏览器在支持其它标准时还存在着这样那样的差异。

但是如果你提交给浏览器的是一份无效代码，又会发生什么呢？答案是浏览器的错误处理程序将启动，以决定该对这段代码进行什么操作。这相当于是在说：“那么，这是一段无效的代码喽，那我们该怎样将这个网页呈现给终端用户呢？干脆就让我们这么做吧！”

听上去挺不错的，是吧？如果你编写的网页存在几个缺陷，浏览器会帮你填补缺陷？不是这样的，因为不同的浏览器的做法是不一样的。例如：

```
<p>This text should be bold</p>

<p>Should this text be bold? How does the HTML look when rendered?</p>

<p>This text should be a link</p>

<p>Should this text be a link? How does the HTML look when rendered?</p>
```

以上这段代码存在的错误有：`strong` 元素被错误地嵌套在多个区块元素中，而且锚记元素未闭合。当你在不同的浏览器中渲染这段代码时，不同的浏览器解析这段代码的方式是差别很大的：

- **Opera** 浏览器会让后面的元素成为 **bold** (粗体) 元素的子元素。
- **Firefox** 浏览器会在段落之间添加额外的 **bold** (粗体) 元素，而这在标记中实际上是没有的。
- **Internet Explorer** 浏览器会将“*This text should be a link*” (这应是一个链接) 的文本放在创建链接的锚记标签之外。

上面这个例子的原始版本出自 **Hallvard Steen** 的文章“同样的 DOM 错误，不同的浏览器解析结果”——你可以通过阅读这篇文章，以进一步了解 **HTML** 错误的处理，以及关于调试工具的一些信息。

上面这些浏览器行为都没有什么不对的；这些不同的行为都是在试图处理不正确的代码，因此都是有其道理的。你应该牢记的基本原则是：尽可能避免在页面源代码中出现无效的标记！

### 怪异模式

怪异模式也是你所应该了解的。如果浏览器遇到未声明文档类型或不正确地声明文档类型的网页，就会以怪异模式渲染网页。在怪异模式下，浏览器会尽量猜测应该用什么样的规则来检验代码，并尽力处理无效代码。有了这种模式，那些以老式方法编写的网页就依然能被显示，但你现在在编写网页时，一定要正确地声明文档类型，以免浏览器调用怪异模式来渲染网页。

### 如何检验你的网页

我们已讨论了检验 **HTML** 的理论知识，现在我们来谈谈比较容易的部分——实际的检验操作！把一个 **URL** 地址输入检验器，查看网页是否能够通过检验，这确实不难；但由于检验器所提示的错误消息有时候并不太好理解，因此要知道具体有些什么错误，该如何修正这些错误，有时就不那么容易了。下面我将讨论几个例子。

在本节中我们将考察的示例如下（你可以下载或查看该 **HTML** 文档）：

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

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

<head>
 <title>Validating your HTML</title>
</head>
<body>

 <h2>The tale of Herbet Gruel</h2>
```

<p>Welcome to my story. I am a slight whisp of a man, slender and fragile, features wrinkled and worn, eyes sunken into their sockets like rabbits cowering in their burrows. The <em>years have not been kind to me</em>, but yet I hold no regrets, as I have overcome all that sought to ail me, and have been allowed to bide my time, making mischief as I travel to and fro, 'cross the unyielding landscape of the <a href="http://outer-rim-rocks.co.uk" colspan="3">outer rim</a>. </p>

### <h3>Buster</h3>

<p>Buster is my guardian angel. Before that, he was my dog. Before that, who knows? I lost my dog many moons ago while out hunting geese in the undergrowth. A shot rang out from my rifle, and I called for Buster to collect the goose I had felled. He ran off towards where the bird had landed, but never returned. I never found his body, but I comfort myself with the thought that he did not die; rather he transcended to a higher place, and now watches over me, to ensure my well-being.

### <h3>My possessions</h3>

<p>A travelling man needs very little to accompany him on the road:</p>

<ul>

<li>My hat full of memories</li>

<li>My trusty walking cane</li>

<li>A purse that did contain gold at one time</li>

<li>A diary, from the year 1874</li>

<li>An empty glasses case</li>

<li>A newspaper, for when I need to look busy</li>

</ul>

</body>

上面这个简单的网页由 3 个标题、3 个段落、1 个超链接、和 1 个无序列表构成。它使

用的检验规则集是 **XHTML 1.0** 严格型文档类型，因此在检验代码时将依照这个规则集对其进行检验。这个文档存在一些错误，主要使用 **W3C** 的 **HTML** 检验器检验后就可以发现。

### W3C HTML 检验器

正如前面所介绍的那样，**W3C** 提供了一个在线的 **HTML** 检验器——请在这个超链接上按住 **ctrl** 并单击鼠标右键，然后选择“在新标签中打开”这个选项——这样便于在你查看示例的时候用 **tab** 键在检验器网页和本篇文章所在的网页之间进行切换。

附注：在 **Opera** 浏览器中，在网页上按住 **ctrl** 并单击鼠标右键，选择“检验”选项，可直接使用 **W3C** 的检验器检验网页。

在检验器界面的顶部，你可以看到 3 个可用的标签：

- 通过 **URI** 地址进行检验：你可以输入网上已有网页的 **URL** 地址，对网页进行检验
- 通过文件上传进行检验：你可以上传 **HTML** 文件进行检验
- 通过直接输入进行检验：你可以把 **HTML** 文档的内容粘贴到检验窗口中

不论你使用哪种方法，检验结果应该都是一样的。在我看来，要检验这个示例网页，最简单的办法是复制全部代码并粘贴到上面第三个标签所说的检验窗口中。这样你就可以得到如图 1 所示的检验结果：

The screenshot shows the W3C Markup Validation Service interface. At the top, there's a blue header bar with the W3C logo and the text "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below the header, there are two navigation links: "Jump To:" and "Validation Output". The main content area has a red header bar that reads "Errors found while checking this document as XHTML 1.0 Strict!". Underneath, a table provides details about the validation results:

Result: 17 Errors	
Source:	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html xmlns="http://www.w3.org/1999/xhtml" lang="en"><head><title>Validating your HTML</title></head><body><h2>The tale of Herbet Gruel</h2><p>Welcome to my story. I am a slight whisp of a man, slender and fragile, features wrinkled and worn, eyes sunken into their sockets like rabbits covering in their burrows. The <em>years have not been kind to me</em>, but yet I hold no regrets, as I have overcome all that sought to ail me, and have been allowed to bide my time, making mischief as I travel to and fro, 'cross the unyielding landscape of the <a href="#">
Encoding:	utf-8 (detect automatically)
Doctype:	XHTML 1.0 Strict (detect automatically)
Root Element:	html
Root Namespace:	<a href="http://www.w3.org/1999/xhtml">http://www.w3.org/1999/xhtml</a>

图 1：示例文档的检验结果——有 17 个错误！

这听起来真让人担心，尤其是当我告诉你这个文档存在的错误不是 17 个的时候！其实用不着灰心——检验器报告的错误比实际存在的错误要多一些，这是因为页面顶部的错误通常会产生级联，这会导致检验器在页面接下来的部分报告更多的错误，因为它会认为有更多的元素看起来都没有闭合或是被不正确地嵌套。你只需了解所提示的错误消息的含义，并找出标记中明显的错误即可。在下面的表 1 中，列出了我为让这个网页通过检验所修正的错误，包括我对为什么会出现该错误的分析，以及我所做的修正。

表 1：我为使该网页通过检验所修正的错误

错误提示消息	逻辑	所做的修正
第 8 行, 第 461 列: 没有 “colspan” 属性	我们知道这一行里是有一个 colspan 属性的，并且这个属性是有效的 HTML，但检验器为什么会说这里没有 “colspan” 属性呢？等等，会不会是它被用到了一个不应使用这个属性的元素上了呢？果然，“colspan” 属性被用于 a 元素——这是错误的！	从 a 元素中删除 colspan 属性。
第 13 行, 第 7 列: 文档类型不允许在此使用 “h3” 元素；遗漏了 “object”、“applet”、“map”、“iframe”、“button”、“ins”、“del” 其中之一的起始标签。<h3>My possessions</h3>	同样，乍看之下这个检验结果有点奇怪——明明 h3 元素是正确地闭合了呀，而且在这个语境下也是允许使用的。应该注意的是，通常这类错误提示消息意味着附近有个未闭合的元素…	在当前行中添加一个 p 的结束标签。
第 19 行, 第 40 列: 文档类型不允许在此使用 “li” 元素；遗漏了 “ul”、“ol”、“menu”、“dir” 其中之一的起始标签。<li>A diary, from the year 1874</li>	这个错误很容易发现——从它提示的那行代码中，一眼就能看见，li 元素的结束标签遗漏了 (/)。	为 li 元素的结束标签加上 (/)。
第 23 行, 第 9 列: 遗漏了 “html” 的结束标签，但却指定了 OMITTAG NO。</body>	这个错误也不难发现，即 html 的结束标签被遗漏了。该错误信息说明连你忘了闭合某个标签都能找出来。	为 html 元素加上结束标签。

在修正了这些错误后，检验器会提示文档已通过检验，如图 2 所示：

The screenshot shows the W3C Markup Validation Service interface. At the top, it says "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below that, a green bar says "This document was successfully checked as XHTML 1.0 Strict!". The "Result" is listed as "Passed". The "Source" code is displayed in a large text area, showing a valid XHTML 1.0 Strict document. Below the source code, there are four configuration fields: "Encoding" set to "utf-8" (with a dropdown for "detect automatically"), "Doctype" set to "XHTML 1.0 Strict" (with a dropdown for "detect automatically"), "Root Element" set to "html", and "Root Namespace" set to "http://www.w3.org/1999/xhtml".

图 2：一个所有错误已得到修正的成功信息。

检验 HTML 并不是很难的工作，你只需保持头脑冷静，并要记住自己在用什么文档类型来进行检验。下载或查看修正后的 HTML 文档。

### 总结

在读完本篇文章后，你应该就能自如地使用 W3C 的在线检验器来检验你所编写的 HTML 了。不过这还只是检验知识的冰山一角——还有很多更加复杂的检验工具可用，我在下面列出来了一些。这些工具可以帮助你检验更复杂、更大型的网页。

### 更多检验工具

- [The Opera debug menu](#)
- [General validation bookmarklet](#)
- [The Firefox web developer toolbar extension](#)
- [The IE developer toolbar](#)
- [Safari tidy](#)
- [HTML tidy](#)

### 练习题

- 浏览器解析无效的 HTML 时，会发生什么？
- 浏览器解析无效的 HTML 时，为什么会产生问题？

- 通过 HTML 4 严格型文档类型检验的文档中使用 `frameset` 会导致错误吗？
- 上一篇：使用导航菜单创建多页面
- 下一篇：可访问性基础知识
- 目录

## 作者简介



Mark Norman Francis 早在万维网诞生前，就在从事互联网领域的 works 了，一直持续到现在。目前他是全球最大网站 Yahoo! 的前端设计师，负责制定国际 Web 开发的最优方法、代码标准和质量标准。

在加入 Yahoo! 前，他先后在 Formula One Management (F-1 管理公司)、Purple Interactive、伦敦城市大学从事过多种工作，包括 Web 开发、后端 CGI 编程和系统架构等。他在 <http://marknormanfrancis.com/> 有一个博客。

## 25. 可访问性基础知识

Posted 12/15/2008 - 16:54 by Lewis

- 网络标准教程

作者: Tom Hughes-Croucher · 2008 年 9 月 26 日

- 上一篇: 检验你的 HTML
- 下一篇: 可访问性测试
- 目录

### 序言

在你创建网站的时候, 网站的可访问性——也就是使该网站对所有人都可用, 无论他们是否残疾——应当始终是你关注的焦点。虽然你可能还没有意识到这一点, 但是迄今为止本教程中出现的每个例子都体现着可访问性; 在本文中我将对可访问性进行详细地探讨, 这样你就可以充

分地理解什么是可访问性，为什么它那么重要，怎样才能确保网站是可访问的，以及用什么准则来界定网站到底是不是可访问的。本文目录如下：

- 什么是可访问性？
- 为什么可访问性很重要？
  - 可访问性的法律义务
  - 潜在市场
  - 搜索引擎
  - 道德规范与品牌
- 做设计时要时刻牢记可访问性
- 互操作性要求
- 易于访问的网页的特征
  - 语义结构
  - 替代内容
  - 定义交互
- 可访问性标准
  - Web 内容可访问性指南 1.0
  - Web 内容可访问性指南 2.0
  - Section 508 法案
  - 其它标准
- 总结
- 练习题

在对网页的可访问性进行具体的讨论之前，我们先来概括地看一下可访问性——毕竟，可访问性并不仅仅只关系到网站；它与你生活中遇到的每一次服务，每一个对象或每一项技术都有潜在的关系。

注意，这里有个需要了解的相关话题 **WAI ARIA** ——由 Web 可访问性推动组织所提出的“可访问的富互联网应用倡议”，该提议本质上是一种方法论，用于指导创建具有更好可访问性的 Ajax/JavaScript 架构的应用程序。你可以在 [dev.opera.com](http://dev.opera.com) 上找到大量关于 ARIA 的介绍性文章。

## 什么是可访问性？

现在请你四下里看一看。如果顺利的话你应该能看到其他人；如果你身边一个人也没有，

那干嘛不出去散散步呢？散步应该是很享受的事情，而且对你也有好处。你会发现周围的人没有一个是跟别人一模一样的——有人棕发，有人不是。有人碧眼，有人不是。有人戴眼镜，有人不戴。我们之中没有一个是跟其他人完全一样的。像头发和眼睛的颜色之类的差异只是装饰性的——它们并不会对我们的生活方式有太大的影响。但有的差异，比如戴眼镜，就会影响到我们了。可访问性是一件简单的事情，是一种宗旨，不过在有的国家，它也是法律的一部分。

可访问性就是对所有人一视同仁，无论他们是否有残障。

我知道这个声明可以作各种解释。大多数针对可访问性的讨论都会首先谈到残疾。这说明了残障人士应受到特殊待遇。但这并非可访问性真正的含义——事实上，它只提及可访问性的表象，只是说明了人们修建建筑、创建网站，以及进行几乎所有其他事情的传统做法。

如果你假定每个人都跟自己一样来进行设计，那么你所创建的东西将总是对某些人不合适。人们之所以认为可访问性是针对帮助残障人士而言的，是因为可访问性的改造在我们的社会中随处可见。比如说，许多建筑物开始的时候只有台阶，后来却突然修起了粗糙的斜面。然而，可访问性长期以来一直是军用设计的特色。这又是为什么呢？因为这对于生存来说常常是很关键的——在高负载的喷气式战斗机里，飞行员无法做到他们在地面上能做到的事。如果飞行器设计师没有将飞行员在高负载和低负载环境下的需求都纳入考虑范围的话，坠机事件可能就会多得多了。

那么，这对网站开发员来说意味着什么呢？简短地说，就是你必需认识到所有可能浏览你的网站的全体用户的需求。可能得等到你稍微了解了健康人群的不同层次，以及他们如何使用电脑之后，才能把这个问题更详细的答案告诉你。通过使用本教程和其它相关文献中列出的技术，你可以创建具有多种交互形式的网站。这样，无论人们属不属于下面的类型，你的网站对于他们来说也是可用的：

- 盲人或严重视觉障碍者，他们使用屏幕阅读器来听取网站，或者通过点字显示器来感知网页。
- 近视者，需要将字体大小放大到 200%。
- 患有运动性残疾，因此无法用手操作鼠标，而用点击棒来操作键盘，或通过视线点击器来操作网站的人。
- 使用跟踪球，或其它不常见的计算机控制设备的人。

你不用对上述交互模式的特定细节感到担心——稍后我们将循序渐进地来研究这些东西。  
**为什么可访问性很重要？**

之所以说可访问性很重要，除了有一个主要理由，还有一大堆次要理由。主要理由就是我

们每个人都是不同的，同时我们都享有使用网站的平等权利，此外还有许多其它理由，是关于为何要将可访问性纳入创建网站时的考虑范围的：

- 在某些国家这是法律所规定的。
- 你不会希望有任何的潜在客户/访问者被排除在外而不能使用你的站点。
- 无障碍网站在搜索引擎上的排名会更高。
- 你展示着良好的道德风范——客户会重视这一点的。
- 如果你按照 web 标准来创建网站的话，你就几乎用不着再为网站可访问性付出额外的努力，而且这会带来许多好处；在提高网站的可访问性与使网站更适合于手机浏览器这两者之间，也有许多交叉点——尽管是出于不同的原因，这也是使得网站更不易访问的另一种情况。事实上，对于网站可访问性与手机互联网开发最佳实践之间的关系，已经进行了一些研究——更多信息请参见 WAI 的“Web 内容可访问性与手机互联网”页面。
- 有利于残障人士的技术同时也有益于所有用户。

下面我将更为详尽地对上述某些要点进行讨论。

### 可访问性的法律义务

注意：了解一些法律基础是很重要的，但在对法律问题发表意见时应当特别小心，除非你是个律师而且非常明白自己在说些什么。

在英国，根据 DDA 法案，在征兵、招聘，以及提供服务和教育时歧视残疾人是违法的。歧视的定义是指没有做出“适当的调整”来适应所有人，无论他们是不是残疾人。这个法案当然也适用于通过网络媒体提供可用的服务或教育。

美国和欧盟也有针对政府网站的要求。在美国，联邦政府（以及某些州政府）网站是必须要遵守 Section 508 法案的。Section 508 法案是一份公文，它规定了实现可访问性的最低要求。Section 508 法案并不仅仅只涵盖了网站；它也涉及到可能会被联邦雇员用到的所有其它技术。在欧洲，欧盟委员会已经认可了 W3C 的 web 可访问性提议（WAI），并将其推荐给各成员国使用。WAI 为网站、网页编写工具发行商以及 web 浏览器（例如，接下来我们将谈到的 WCAG）提供了指导方针。

### 潜在市场

如果你只针对某个特定人群开发网站（或是其它任何东西）的话，即使你没有意识到，实际上你已经把其他类型的人群排除在外了，而这些人群加起来很可能占据了相当可观的（即使不是大多数）市场份额。2000 年英国的超市连锁店 Tesco 上马了一个项目，即对他们的线上食品杂货网站进行改造，建立了特别针对视觉障碍人群的页面版本。据 RNIB 的 Julie Howell 说，

“[Tesco.com](#) 开展的这项工作方便了盲人客户获得食品杂货送货上门的服务，并带来了超过每年 1300 万英镑的收益，如果他们的网站对盲人客户不可用的话，公司是不可能获得这笔收益的。”由此可见，如果 [Tesco](#) 公司没有考虑到视觉障碍人群的话，他们就会失去这个价值至少在 1300 万英镑以上的市场。

这件事的根本教训在于，无论残疾与否，人们都需要同样的服务；比如食品杂货，出租车，电力供应；而且享受同样的东西；比如电影，音乐，酒吧。认为某人因个人情况而改变了他参与到社会生活的方方面面中去的能力或愿望，这种主观臆断已经一次又一次地被证明是错误的。

### **搜索引擎**

搜索引擎可不是人。人们在创建网站的时候，常常没有考虑自己的网站在 [Google](#), [Yahoo](#) 等搜索引擎上是怎样被搜索到的。搜索引擎只是些计算机程序，它们只能用自己能理解的信息来给网站编制索引。这就使得它们同视觉障碍者使用的屏幕阅读器的情况非常相似。

图像是一个最好的例子，可以说明上述特性是如何影响到网页设计的。通过列出一张列表，写明各个像素点是什么颜色，并将该信息传输到显示器，计算机才能显示图像。如果你将一张含有文本的图片，比如一个标识，放在网页上，计算机并不理解这个文本的意思，它甚至连这个图片是否包含了文本都不知道。在 [HTML](#) 的图像元素中，有一种用文字来描述图片内容的方法，这就是 `alt` 属性。你应当为网站上所有的非装饰性图片提供文字描述信息，而且绝对不能将整段文字都用图片（或者 [Flash](#)）来展示——否则盲人和搜索引擎对这段文字将会一无所知！结果就是，你的搜索引擎排名（比如说，在 [Google](#) 之类的搜索引擎上查找你的网站的容易程度）就要遭殃了，而且你就会不必要地失去宝贵的市场。

### **道德规范与品牌**

虽然说大家都应该支持可访问性，但这并不代表大家都是这么做的。在支持可访问性的同时，你也是在尽力为社会利益服务。你可以为此而骄傲——显示你对每个社会成员的关心，这只会提升你的品牌形象。从职业的方面来说，尽力提供最优质的产品也是我们应尽的职责。在这个重视个人的社会中，不能因为某人有不同的需求就将他排除在外，这一点是很重要的。

如果你在选择方针政策时尽心尽责，在发表方针贯彻声明时诚诚恳恳，你就可以树立起非常积极的品牌形象。比起其他公司来说，那些显示出自己关心客户的公司能够保持高得多的客户忠诚度。

### **做设计时要时刻牢记可访问性**

实现可访问性的关键在于，无论是考虑问题还是解决问题，都要明白你所针对的是不止一种用户。如果你把可访问性看作是那种可以在最后才加以考虑的东西，那么你所做出来的将是一个生硬地捆绑到最后的东西。这将会花费你更多的时间，同时又根本不会奏效，而且还很难看。

为了实现设计良好的解决方案，最好的办法是从一开始就对所有的要求牢记于心。这并不是说你不能改变计划或者添补一些遗漏的东西，而是你应该认识到自己所要解决的整个问题是什么。对于网站来说，这意味着创建一个对所有用户都可用的解决方案，包括那些可能无法使用鼠标或键盘，或者显示器等设备的用户。

## 互操作性要求

互操作性要求很可能随情况不同而变化。

新技术常常在还不支持可访问性的时候就被采用了。比如说，微软最新的 **Silverlight** 插件就不能通过屏幕阅读器和其他辅助技术的可访问性应用程序接口来显示信息，尽管微软计划在未来实现这项支持。

即使是在理论上实现了可访问性支持，也需要一些时间来使它们得以运用在辅助技术上面。例如，较新的屏幕阅读器比起旧的版本来，能够更好地适应由 **Javascript** 触发的 **HTML** 结构更新。

即使是已经发布了很久的技术，其可访问性支持也可能因操作平台的不同而不同。比如说，**Adobe Flash Player** 插件很早就能将信息传递给 **Windows** 的可访问性应用程序接口了，但却不能在 **Apple** 或 **GNOME** 上同样地工作。

在支援技术的推出和其广泛应用之间也存在着滞后。浏览器和插件如今已倾向于免费，而主流的辅助技术却可能非常昂贵。比如说，现在最流行的屏幕阅读器之一是 **Freedom Scientific** 的 **JAWS Windows** 版。大体来说这个软件每年都会推出新版本。**JAWS Professional** 的零售价格是 1095 美元，而且即使你花了 200 美元购买了接下来的两个版本的软件维护协议，软件的升级也还要再花 500 美元以上。因此，尽管现在最新的版本是 **version 9**，也还是有许多的 **JAWS** 用户在使用旧版本。

因此当你针对公共网络建立网站时，你必须考虑到高度多样化的客户端用户/技术组合的互操作性。下面列出四种实现方法：

- 逐步强化你的网站功能，同时对支持性进行测试。
- 允许用户关闭有问题的增强功能。
- 提供相同内容或功能的替代版本。
- 就客户端需要支持的技术向你的客户提出建议，并举例说明哪些公司的产品支持这些技术。

在内联网中，就不必对向后兼容和多样性考虑那么多了。例如，一个特定的机构可能有能力保证所有的残疾员工都有办法获得辅助技术，而且该技术能够适当地支持 **DHTML**。在这种情

况下，以及对所提供的辅助技术进行适当的测试后，就可以将 **JavaScript** 作为基础配置了。

然而，向前兼容和跨平台兼容仍然是个问题，因此，比起私有的非标准化技术来，应当优先选择开放的标准化技术。

举个例子，假设你在为某家大公司开发一个内联网上的培训应用程序。他们要求你确保该应用程序的可访问性，但没有指定你必须要遵守的标准。在与他们的 **IT** 部门沟通后得知，每个员工都有最新版本的 **IE**，并且 **JavaScript** 是启用的，**Flash** 也是安装并启用了的，而且也提供了针对这几方面所需的最新辅助技术。现在，就算是该公司换成了基于 **Unix** 的操作平台，也有相应的辅助技术来支持 **JavaScript**，但 **Flash** 文本和控件只有在 **Windows** 下才能用。这种情况下，你完全可以将脚本语言和 **Flash** 作为该应用程序的基础配置。但由于只有在 **Windows** 下辅助技术才能支持 **Flash** 控件，你决定只能将 **Flash** 用于播放视频，并按照 **web** 标准来创建 **Flash** 视频的控制集。这样，就算该公司换成了 **Unix** 系统，你的应用程序仍然是可用的。

由于一个组织的 **IT** 策略可能会改变，而使得 **JavaScript** 功能可用，以及开发插件的可访问性特征集的努力也可能会失败，因此即使你有技术基础，从核心层的 **HTML** 逐步强化你的网站也仍不失为一个好主意。

### 易于访问的网页的特征

在这一部分中，我将讨论网站的各种可访问性特征——也就是说，一个易于访问的网站应当包含哪些东西。我将对每一项进行详细说明。

#### 语义结构

**web** 标准的基础之一就是对 **HTML** 中语义结构的使用。语义结构对于可访问性来说是非常重要的。这是因为它提供了页面信息的框架。如果人们无法看见网页的视窗界面，语义结构也能帮助他们获得大量的信息。语义结构可以提示他们在文件层级中所处的位置，还有他们可以如何与各种页面元素进行交互，以及在适当的地方对文本内容进行强调。

导航菜单可以作为一个好例子，用来说明一个文档的语义结构为何对其可访问性如此重要。一个结构良好的导航菜单应当是一系列项目的列表。你可以将导航菜单标记为 **HTML** 列表：

```

 Menu Item 1
 Menu Item 2
 Menu Item 3


```

通过将导航菜单构造为列表，就能很容易地让那些使用屏幕阅读器、同时无法看到列表的

人知道这是个列表。因为他们的屏幕阅读器会告诉他们这是一张列表。如果你没有使用列表标记，屏幕阅读器就没办法知道这是列表，因此也就不能告诉使用者了。

在本教程之前的许多文章，主要是那些涉及 **HTML** 的文章中，你可以找到更多关于如何在 **HTML** 中使用正确的语义标记的资料。

### 替代内容

就如在 [搜索引擎](#) 一节中所说的，确保页面内容和导航菜单具有替代内容可用是非常重要的。文本可以作为页面内容的通用替代内容，但也有一种例外情况，这一点我们将在下面谈到。文本内容可以很方便地由屏幕阅读器朗读出来，也可以放大或缩小，还可以方便地改变其对比度，或者进行其他许多变形操作。由于对文本进行操作非常容易，我们应该为更多醒目的页面内容提供以文本为主的替代内容。现在有的媒体已经集成了文本存取通道，比如新版本的 **Flash**，其中的文本内容可以直接被读取，从而不必为整个媒体提供替代内容。

文本替代内容无法提供帮助的残障人群只有一种，那就是认知障碍患者。对认知障碍患者提供帮助的困难在于，在大多数情况下他们需要的是不同的内容，而不是不同媒体上的相同内容。但这并不意味着你就可以不用管他们了。简化网站上的语言和术语对大家都有好处。像 [浅显语言计划委员会](#) 之类的组织正在向那些原材料公司发起倡议，建议他们用“浅显的语言”来对自己的客户进行法律要求和条款声明之类重要信息的通知。他们制定了一部 [浅明英语词典](#)，其中包括了一些术语，可以有助于用尽量简单的语言进行有效沟通。

怎样在你的网站上实施文本替换呢？第一步就是对非文本的内容进行识别。在 **HTML** 中有很多东西都不属于文本。最显而易见的非文本内容就是图像。下面的例子就是图片的一种可访问的使用方式：

```
<p>An interesting piece of art is Michelangelo's "God creates Adam"

.</p>
```

本例中的图片是页面内容的主要部分。这里 **alt** 属性包含了对该图片的简短描述，以便无法准确看到该图片的用户（或搜索引擎）使用。**longdesc** 属性负责链接到一个包含了对该图片更详细描述的 **HTML** 页面。一般而言，这种做法只用于描述那些作为页面主要内容的复杂图片。但这种方法也存在浏览器支持不良的问题。绝大多数时候你只会用到 **alt** 属性。

如果图片不是用在正文内容上，比如说只是用来作导航，或者纯粹只是起装饰作用的话，你可以不用像内容图片一样对其进行处理。用来增加按钮或页面导航条吸引力的图片应该对其 **alt** 属性进行定义，该属性值应与图片中的文本相同。**alt** 属性的功能是提供一条捷径，使计

算机得以理解图片中包含的文本（进而通过屏幕阅读器读给用户听）。

对于那些装饰性的图片，或者是用来作追踪广告的图片，以及任何用户不会在意或与之进行交互的图片，你应当将其 `alt` 属性设为空。这并不是说直接省略掉这个属性，而是故意设置成这样：`alt=""`。之所以这么做，是因为现在有种智能型屏幕阅读器，这种阅读器是用来帮助使用者对付那些可访问性非常不好的网页的。如果某张图片没有 `alt` 属性，尤其是当这种图片是某个链接的一部分时，这种屏幕阅读器就会将图片的 `URL` 读给使用者听。如果这张图片的文件名是 `add_to_cart.gif` 之类的话，用户就可以使用 `alt=""`，这样屏幕阅读器就不会读出图片全部的 `URL`，从而避免使用户感到郁闷。

并非所有形式的页面内容都像图片那么简单。比如 `Flash` (`Flash` 文件本身就可能是一个网站) 或影频之类更复杂的媒体文件就需要更复杂的描述。最新版本的 `Flash` 允许为 `Flash` 影片中的元素提供文本替代内容，就如同在 `HTML` 中一样。

### 定义交互

现在很多网站都涉及到除 `HTML` 之外的其他技术。如果使用不当，就算是像 `CSS` 一样的基础技术也可能会导致网页或交互的可访问性大打折扣。

注意，下面这个例子的目的是为了让你去仔细思考网页上各个部分的作用。为了确保这些部分的可访问性，它们所用到的 `HTML` 元素和视觉隐喻在语义上必须是正确的。如果你对这一点感到难以理解，可以把这个例子多读几遍，再多研究几个菜单和其他页面组成部分，同时想想看如果运用了适当的 `HTML`，以及如果该组成部分的外观和感觉就其功能而言有意义的话，结果又是怎样。别指望网页浏览器会用一个写着“输入你的邮箱地址，注册后方可浏览本资讯”的文本框来进行搜索，也别指望视觉正常的访问者能够发现什么感兴趣的内容，如果所有的标题都跟一般的文本一样的话(同样的，如果所有的“标题”都只是通过 `CSS` 或 `font` 元素做出更大的字体的话，你也别指望盲人用户能发现什么名堂)。

对这一点而言，普遍使用的标签视觉隐喻就是一个很好的例子。标签的隐喻源自那种可以按主题进行检索的活页夹。这种形式被用到了计算机上，在屏幕的某个特定区域以标签的形式显示出各种主题信息，每个标签都链接到相应主题的页面——在 [dev.opera.com](http://dev.opera.com) 上你就可以看到一个非常漂亮的标签示例——就在页面顶部。并且这种风格也相当直观。但问题就在于创建标签所需的技术——通常要通过 `JavaScript` 来实现标签的效果。

标签作为一种交互方式，一旦其功能超出“允许用户选择某项信息”，其原始隐喻就被打破了，但通常我们还是用同样的代码来描述标签。下面例子中的 `HTML` 代码说的就是一个用来显示信息的选项卡控件：

```

<div class="article_tabcontrol">
 <div class="article_hd">

 Dogs
 Cats

 Fish

 </div>
 <div class="article_bd">
 <p id="dogs" class="article_selected">Some information about dogs. The dogs tab is the default tab.</p>

 <p id="cats">Some information about cats.</p>
 <p id="fish">Some information about fish.</p>
 </div>
</div>

```

本例中 `selected` 类用于指定哪个标签是“最前面的标签”，你可以去看看前面说的那个网页顶部的“Articles”标签，该标签用的就是这个方法。

这种结构对信息内容非常适用。在本例中，值为 `selected` 的 `class` 属性是用来表示哪个标签正处于活动状态的，比如说，处于打开的状态，并正在显示信息的那个标签；而其余的标签就处于关闭状态(比如，它们的文字段落被隐藏起来)，直到其对应的链接被点开。在这里 `dogs` 的标签是默认的活动标签，如图 1 所示：

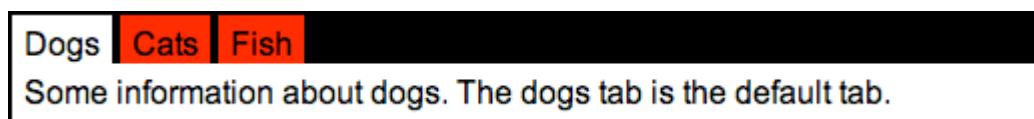


图 1：一个简单的标签控件，其默认的活动标签是 `dogs`。

一旦另一个链接被点开（如图 2 所示），就会激活 JavaScript，动态地将 `class="article_selected"` 属性移到那个链接去，然后对所对应的标签应用样式，使其得以显示，同时，之前显示的那个标签将被隐藏。



图 2：点开另一个链接，它对应的标签将被激活。

你可以在后面即将发布的关于 JavaScript 的章节中看到这种控件的实际使用例子。

让用户通过标签来选择各种类型的搜索方式，这样的做法也是很常见的。在本例中，如果你还是保持上个例子的代码风格的话，这个设计就会失败。

```
<div class="article_tabcontrol">
 <div class="article_hd">

 Dogs
 Cats

 Fish

 </div>
 <div class="article_bd">
 <form id="dogs" class="article_selected" action="search.html" method="GET"><div><label for="dogsearch"><input type="text" name="dogsearch" id="dogsearch"><input type="submit" value="Search for Dogs"></div></form>

 <form id="cats" action="search.html" method="GET"><div><label for="catsearch"><input type="text" name="catsearch" id="catsearch"><input type="submit" value="Search for cats"></div></form>

 <form id="fish" action="search.html" method="GET"><div><label for="fishsearch"><input type="text" name="fishsearch" id="fishsearch"><input type="submit" value="Search for fish"></div></form>
 </div>
</div>
```

继续使用同样的代码结构已经没有意义了——在本例中，为了实现内容的更换，你一再地对同样的表单元素进行重复，这其实是在浪费标记。比起外观上的考虑来，关注交互本身更为重要。在本例中，你应该通过搜索表单来改变用户已有的交互，而不是通过选择新的信息来浏览标签。实际上，标签所应有的唯一用途就是用来选择用户正在搜索的动物类型。如果你使用这个办法，就可以创建一个对网站的所有用户都要好得多的交互，而且标记更为简洁，维护也更容易。

```
<form action="search.html" method="GET">
 <fieldset>
 <legend>Search within:</legend>

 <label for="dogs">Dogs</label><input id="dogs" type="radio" name="animal" value="dog" checked>
 <label for="cats">Cats</label><input id="cats" type="radio" name="animal" value="cat">

 </fieldset>
</form>
```

```
value="cat">

 <label for="fish">Fish</label><input id="fish" type="radio" name="animal" value="fish">

</fieldset>
<input type="text" id="searchfield" name="search">

<input type="submit" value="Search">
</form>
```

由于我们先创建了交互，标记变得更简洁了，该网站的所有用户也得到了最佳的用户体验。与前一个例子比较来看，如果我们先拓展视觉隐喻的话，我们就会立刻毁了这个交互，而且还会制造出一大堆标记来。如果我们是通过 **AJAX** 来插入页面内容，而不是直接将其写在页面上的话，情况就会更糟糕。那样的话，没有 **JavaScript** 的用户就只好加载一整个新页面，才能获得猫或者鱼的搜索表单。如果先考虑基本交互（而不是仅仅只加载视觉效果的部分）的话，你就完全可以简化这个问题。现在我们可以只用一个表单来进行所有的搜索，而同时仍然能保持标签隐喻（虽然这需要一点样式和脚本）。

这就是如何创建可访问的交互的精髓。**HTML** 最棒的地方之一就是：如何在 **HTML** 中创建易用的交互，这一问题最难的部分已经被解决了。只要你不**在 HTML 上应用那些破坏隐喻的技术**，你就可以轻松地做出适合大多数人群的东西。

## 可访问性标准

在这一部分中，我们将会回顾一些可利用的标准和原则，这些标准和原则是为了定义 **web 可访问性**，以及帮助 **web 开发员** 创建可访问的网站。大多数的标准体系都包括检查表格系统，因此开发员可以检查网站对各种可访问性标准的一致性如何。

### Web 内容可访问性指南 1.0

**W3C** 是最早的网络标准机构之一。他们于 1999 年 5 月公布了第一版 **web 可访问性倡议 (WAI)**，该倡议是为了提高网站的可访问性。**Web 内容可访问性原则 (WCAG)** 是应用最为广泛的 **web 可访问性标准**。**WCAG 1.0** 得到了包括欧盟和意大利政府在内的许多政府机构的大力推荐，或是干脆强制执行。

**WCAG 1.0** 由 14 条准则构成，这些准则概括了一个可访问的页面所必需达到的标准。每条准则中都有若干检查点，这些检查点就是该文件的核心内容。准则是用来对那些存在于设计员心目中的概念进行定义，检查点则是用来检验网站对该标准的一致性。每个检查点都按照其优先级来排列，优先级有 3 个，从 1 到 3，说明了该检查点的重要程度。你必须做到所有优先级

为 1 的检查点，这样才算是遵守了 WCAG 1.0。如果你做到了所有优先级为 1 的检查点，你的一致性等级就为“A”，如果你同时满足了所有优先级为 2 的检查点，你的等级就为“AA”。如果你遵守了优先级为 1, 2, 3 的全部检查点，你的等级就是“AAA”，这是最高的等级。

事实上 WCAG 1.0 已经有点过时了。许多公司选择从“A”或“AA”级标准开始，然后再增加其它一些准则，比如 RNIB 的 See it Right。WCAG 1.0 是一个很好的起点，但你需要更新的标准，尤其是当你应用了大量的 JavaScript，或者其它在 1999 年之后才成熟的技术的话。这是因为 WCAG 1.0 就是在 1999 年公布的。

关于 WCAG 1.0 另一件需要注意的事情是它是一套三个文件的一部分。另一个文件涵盖了“用户代理”方面的标准，用户代理是指浏览器（比如说 Opera）以及人们使用网络所需要的附加技术（比如屏幕阅读器）。第三个文件涵盖了 Dreamweaver 或内容管理系统之类的开发工具——该文件旨在让这些工具在提高网页可访问性方面承担更多的工作。可惜的是，这一部分并没有进行到底，三个文件中唯一被广泛使用的标准就是 WCAG 1.0。这意味着在用户代理方面，对 WCAG 1.0 的期望并没有得到满足，开发工具并没有帮你分担你肩上的确保网站可访问性的重担。当然这也不是说你就不要遵循 WCAG 1.0；只是说 WCAG 1.0 只满足了可访问性的一部分要求，它并不是一个完备的解决方案。

## Web 内容可访问性指南 2.0

WCAG 1.0 发布之后，W3C 又致力于 WCAG 2.0。这个升级版本的标准目前仍在编写之中。经过 W3C 的种种程序之后，该标准可能会在 2009 年初发布。

WCAG 2.0 跟 WCAG 1.0 比起来稍微有些不同，它的技术无关性要更强一些，比如，WCAG 2.0 可以被用在 HTML, CSS, Flash 等等上面。WCAG 2.0 基于四个可访问性原则，分别是：

可感知

人们可以通过适合自己的媒体来获知网页内容。比如应当让盲人得以收听页面内容。

可操作

人们可以与 web 应用程序或内容进行交互。

可理解

使用者可以弄懂页面内容和用户界面。

健壮性

所提供的服务都应当不受平台或操作系统的限制。这样就可以避免人们提供一些不太完善的服务，这些服务会因为硬件/软件的限制而导致大多数人都无法使用。

应该注意到，网站并不是必须满足全部这些要求。用户所使用的技术也应当承担起一部分

工作。比如说一个屏幕阅读器应该将页面内容读给需要的人听，而不是让每个网站都提供一份音频版的内容。然而，为了实现这一点，网站应当确保其页面可以用一般的屏幕阅读技术读取。这种差别是很重要的，因为这是一个只具有“可访问性组件”（比如说一个可以放大字体的按钮）的网站与一个可以在许多不同情况下（比如不可预见的不同的浏览器和设备）运行的网页之间的差异。

WCAG 2.0 与 WCAG 1.0 之间的不同之处还在于它们对待技术的方式。由于该标准的技术无关性更强，针对的对象是可访问性的概念，而不是具体的技术细节，研究该标准的其它文件就显得很重要了。WCAG 2.0 的“标准”文档提供的是相关的协议，而“技巧”文档则为开发员提供可靠的可实施代码片段。这部分分成“一般”技巧（针对多种技术的）以及针对特定 W3C 技术的细节两个部分。W3C 没有为私有技术提供这种文件，因此你只能从其它来源寻找针对 Flash 和 Silverlight 之类技术的技巧。

## Section 508 法案

Section 508 法案是对 1973 年颁布的美国劳工复健法的改进。Section 508 法案的一个版本是在 1998 年成为法律的，该版本的法案规定了在美国联邦政府采购过程中必须遵循的程序。这意味着所有由联邦拨款的美国政府机构都必须遵守 Section 508 法案中规定的程序和准则。这些准则涵盖了 web 可访问性和其它与计算机和电子通信技术有关的可访问性问题。不管你听到的说法是什么，联邦法律并没有强行规定那些不属于上述组织范围的机构也必须遵守 Section 508 法案。不过，美国的一些州和公司也采用了 Section 508 法案来对自己的采购过程中的“可访问性”进行界定。

Section 508 中涉及网页可访问性的部分是 Subpart B § 1194.22。第 1194.22 条法规分成 16 个要求，分别用以 *a* 到 *p* 的字母命名。前面 11 条要求 (*a* 到 *k*) 与 WCAG 1.0 中的部分内容等同。这些要求和它们在 WCAG 1.0 中的对应项列出在 section 508 法案文件的参考表中。除了一种情况以外，section 508 法案的其它要求将在 WCAG 2.0 中得到满足。要求 *m* 涉及到 section 508 法案的 Subpart B § 1194.21 条款。该条款要求在 WCAG 2.0 的强壮性原则部分有个不完全相同的对应项。

在撰写本文的时候，有项关于新版 section 508 法案的调查正在进行之中，该调查是由电信，电气和信息技术咨询委员会 (TEITAC) 负责的。2008 年 4 月，TEITAC 将这项调查的结果呈交给 Section 508 法案的评估委员会。

## 其它标准

W3C 制定的另一个重要标准就是 WAI-ARIA 标准。WAI-ARIA 代表 web 可访问性倡议——可访问的富互联网应用。这是一套文件，定义了如何提高那些使用 HTML，JavaScript 和

AJAX 之类技术的复杂的 web 应用程序的可访问性。该标准获得了市场上即将推出/现行版本的大多数主流浏览器的官方支持：Opera 9.5, Internet Explorer 8 和 Firefox 3。

还有许多其它的 web 可访问性标准，因为太多了，在此无法一一赘述。不过 W3C 仍然是一个优秀的 关于 web 可访问性的国际政策表——相对于查找你所在之地的政府的政策文件来说，这是一个很不错的资源。

## 总结

可访问性对于经济和社会来说都是一个重要话题。它并不是网站的特性，而是网站建设质量的一个衡量标准。如果你在创建网站的时候（以及开始创建前）顾及你的用户的话，你就能创建可访问性更好的网页，并且享受它所带来的一切好处。现在有许多著名的准则可以帮助你做到这一点——通过遵守这些准则，你就可以确保你所创建的网站在网页可访问性方面符合专业标准。

## 练习题

- 说出三个理由，为什么说建立可访问的网站很重要。
  - 通过互联网来调查一下你所在国家关于可访问性的法规，列出你认为适用于你的网站的法规。如果他们要求你采用 WCAG 或 Section 508 法案之类的 web 标准的话，得确保你的网站包含了这些标准。
  - 说明一下为什么对搜索引擎优化来说可访问性很重要。
  - 用你自己的东西，比如照片，来创建一个范例，说明替代内容在提高网页的可访问性方面的作用。
  - 通过互联网研究一下，你该怎样提高 Flash 或 Silverlight 之类技术的可访问性，并写出如何提高这些技术的可访问性与如何提高 HTML 的可访问性之间的区别。
  - 说明一下你如何在某个网页上设计一个易于访问的交互。写出创建树形控件的操作步骤（你可以不必实际做出来）。
- 
- 上一篇：检验你的 HTML
  - 下一篇：可访问性测试
  - 目录

## 作者简介



Tom Hughes-Croucher 一开始就从事网络产业。他曾参与了诸如万维网联盟（W3C）和英国标准学会（BSI）之类的标准机构的多个 Web 技术标准的制定。后来他又从事数字音乐行业，为像 Tesco、Three telecom 和 Channel 4 这样的英国著名品牌提供数字音乐解决方案。

Tom 现在是 Yahoo! 的技术专员。现在他专门从事前端 web 技术和 REST 架构的 web 服务开发，而且总是尽可能地宣传最佳实践。在此之前由他负责的 European Frontpages 每个月都有数百万的欧洲访客——他只是找不到更合适的词来形容这么惊人的访问量罢了。

## 26. 可访问性测试

Posted 12/15/2008 - 16:55 by Lewis

- [网络标准教程](#)

作者: Benjamin Hawkes-Lewis · 2008 年 9 月 26 日

- [上一篇: 可访问性基础知识](#)
- [下一篇: CSS 基础知识](#)
- [目录](#)

### 序言

**web** 可访问性测试是易用性测试的子集之一，这种测试是针对那些使用互联网的方式受到残障影响的用户的。对于易用性和可访问性两者而言，最终目标都是探求人们怎样才能轻松利用网站以及反馈那些可以改进设计和实施的信息。

可访问性评估通常比易用性测试更为正式。法律和公众舆论倾向于不歧视残障人，为了公平对待所有公民，政府和其它一些组织致力于遵循各种网站的可访问性标准，如美国联邦政府制订的 [Section 508 法案](#) 和 [W3C 的 Web 内容可访问性指南（WCAG）](#)。

然而，重要的是区分符合标准和最大化网站可访问性之间的差别。在最理想的情况下，两者是一致的，但是任何给定的标准仍可能无法做到：

- 满足所有残障人士的需求
- 平衡不同残障人士的需求
- 用最佳技术与那些需求相匹配
- 使用明确的语言表达需求或技术

这些弱点可能会导致有些人带着美好愿望却误入歧途，并且可能被追求官样文章而忽略产品可用性的人所利用。

此外，互联网可访问性是一个目标，而不是一个是/否的设置，这是一种人类需求和技术之间的关系。正如我们了解人类需求的演变，以及技术对这些需求的不断适应，可访问性的环境要求也将改变，现行标准也会过时。不同的网站，不同的网页，采用不同的技术服务不同的需求。像 [Skype](#) 这种语音聊天方式对盲人来说是极好的，而 [video](#) 视频聊天则是手语用户的一种福利。

在解决一种产品如何轻松使用的问题时，残障人士是一个特殊的问题，这是因为在用户和评估者之间引入了额外的经验差距。可访问性评估必须考虑在不同的情况下对网络的体验会如何，这些不同的情况包括不同的感官意识和认知能力，以及为了使网络更贴近人们特别是残障人士而采用的各种独特的配置选项和专业软件。

在你试图评估网站的易用性或可访问性时，把自己设想为正使用网站的青少年电影爱好者或是 50 岁的银行经理都有困难，更别提把自己设想成残障人士了。但是，要是青少年电影爱好者正好是个聋哑人并且需要观看电影的字幕时该怎么办？如果那位 50 岁的银行经理正好是盲人，而且他为了与自己的桌面环境和网络浏览器交互，正在使用对于评估者来说是陌生的特殊技术（如屏幕阅读器）时又该怎么办？

可访问性的指导方针和工具有助于弥合这些经验的差距。然而，由于需要传神的想象力、创新的技术以及同用户交谈，这些指导方针和工具也只能作为一种补充，而不是替代品。

本文中，我将从建立正规的一致性和使可访问性最大化两方面来讨论网页可访问性的评估办法。本文的结构如下：

- 测试应当在什么时候进行?
- 了解你的要求
  - 外部需求
  - 一致性细节
  - 超出预期
  - 用户界面的重要性
  - 残障人士角色
  - 选择一种可访问性标准
  - 法律的精神
- 应该由谁来测试?
- 专家测试
  - 半自动化可访问性检查工具
  - 结构检查
  - 检查和使用终端用户辅助技术
  - 详细检查
    - 可感知性
    - 可操作性
    - 可理解性
    - 健壮性
- 用户测试
  - 招募测试者
  - 实际的考虑
  - 选择任务
  - 解释结果
- 对可访问性测试结果进行沟通
- 总结
- 练习题

### **测试应当在什么时候进行?**

一位资深软件工程师说过：“尽早测试，经常测试”。在开发过程结束后跟踪测试结果存在两种风险：

1. 项目往往是既超时又超过预算的。由于这些压力，常常使测试变得既仓促、又大意，或者直接被忽略掉。
2. 在一次开发过程中，修复后来才发现的问题与从一开始就把事情做对相比，需要更多的工作量。

因此，要确保质量和节省时间与金钱，可访问性评估应该从最初设计产品时就启动，并且延续到随后的反复开发过程之中直到最后交货。

### 了解你的要求

在你开始评估一个可访问性项目之前，你需要确定在既定环境、意向受众和资源的情况下，对于该项目来说有哪些关键要求。有些要求将由第三方如政府和客户来设置；还有一些你可以自己来选择。

#### 外部需求

通常需求都来源于外部，例如：

- **政府。**这类需求通常采取立法的形式来禁止歧视残障人士，而并非通过规定某种特定标准或列举出详细的一致性要求。一个重要的例外是当法律强制公共部门执行某一标准时的情况。例如，**Section 508** 是美国联邦立法条款之一，该法案规定了为联邦机构制作的网站必须符合至少一套具体定义的要求。**WAI** 的关于 **Web** 可访问性的政策页面提供了一份类似法案的不完整清单。但是，如果想得到一份关于你司法责任的权威声明，请咨询律师。
- **客户政策。**例如，壳牌公司目前正在努力确保他们的网站符合 **WCAG 1.0** 的“**AA**”一致性级别，所以如果你正在为其开发一个网站，你就需要（至少）满足相同的标准。
- **营销工具。**符合特定的标准，如 **Section 508** 法案，可能有助于向那些关注可访问性的顾客销售一个项目。
- **你所在机构的内部可访问性政策。**例如，英国广播公司（**BBC**）制作的项目需要遵守 **BBC** 的可访问性指南 **v1.3** 标准。

#### 一致性细节

获取尽可能明确的外部需求是很重要的。一些可访问性标准可能有一个以上的一致性级别或类型，所以敲定哪个是必需满足的级别就显得尤为重要了。例如，**WCAG 1.0** 有三个一致性级别：

1. 对未通过“**A**”级的 **HTML** 文档来说，残障人士“将无法访问信息”。
2. 对未通过“**AA**”级的 **HTML** 文档来说，残障人士“将难于访问信息”。

- 对未通过“AAA”级的 HTML 文档来说，残障人士“将稍微有点难于访问信息”。

WCAG 2.0 草案也有三个级别，但一致性的可能情况变得更加复杂。但凡某种资源是某个过程（例如一个在线商店的产品查找、产品选择、结账以及购买确认）中一系列资源的其中之一时，该系列中所有资源的一致性级别就是级别最低的那个资源的级别。

一致性要求必须建立在“支持可访问性”的内容技术基础之上。作为一种支持可访问性的内容技术，这种技术必须满足以下条件：

- 已经被证实可以与用户的辅助技术一同工作。
- 具备可与用户辅助技术一同工作的用户代理（浏览器、插件等），并可为残障用户提供与无残障用户相同费用的服务。

请注意，在内联网环境中，你可以保证所有用户都将获得这些用户代理，但你无法保证同样的情况能发生在互联网上。例如，某个应用程序可能在无任何商业技术的条件下仍然可用，但却只能被含有某个商业插件的屏幕阅读器所读取，使用该插件需要一个针对某个机构的企业内部授权。当该应用程序部署在该机构的内联网之中时，它可能符合 WCAG 2.0，但当其部署在公共网站之上时则无法符合 WCAG 2.0。

WCAG 2.0 还承认受限的一致性声明。如果提供了一种可访问的替代品，某种原本不可访问的资源就能够符合可访问性的规定。如果页面内容是由其它来源所综合得到的，发行商则可以发表一项部分一致性的声明。

### **超出预期**

确定外部需求只是可访问性测试的开始；外部需求应被视为最低要求的集合，应当在此集合的基础上添加进一步的目标，以使可访问性最大化。作为可访问性的评估者，你的职责就是要引起更多对可访问性的关注，因为你是这个领域的专家。

在提交最终报告时，你可能需要区分两种情况。例如，某家网上超市的忠实客户可能会提到，他们想要一家对盲人用户来说可访问的商店。考虑到目标人群，你还应该评估该商店对其它残障人士是否也可访问。

请注意，遵守特定标准的外部需求不一定非得禁止从其它现行标准中获得最佳的实践指南。例如，你可能会为某联邦机构对一个为老年市民服务的网站进行评估，并且还要求该网站遵守 Section 508 法案。Section 508 法案规定了：

§ 1194.22 (c) 网页内容设计若使用颜色来传递信息，也应同时提供不使用颜色来传递信息的方式，例如从上下文或标记中获得信息。

这一条款可帮助那些知道如何定制网页内容外观的用户，但并没有在建议的颜色之间确保具足够的对比度，从而为目标受众提供可访问性最大化的默认外观内容。幸运的是，并没有什么东西能够阻止一个网站履行这一要求，并且同时符合来 WCAG 2.0 草案中的以下条款：

**1.4.3 对比度(最小值):** 文字和文字图片的色彩对比度应至少在 5:1 以上，除了以下几项：  
(AA 级别)

- **大型印记:** 大型文字和文字图片的色彩对比度应至少在 3:1 以上；
- **附带的内容:** 对那些无效的用户界面组件的文字或文字图片没有最小对比度要求，因为那是纯粹的装饰，是附带的图片内文字，或者说任何人都无法看到的。

**注:** 可以通过在网页上或从页面内控制对比度而满足 1.4.3 和 1.4.6 条款。

**1.4.6 对比度(增强型):** 文字和文字图片其色彩对比度至少在 7:1 以上，除了以下几项：  
(AAA 级别)

- **大型印记:** 大规模文字和文字图片的色彩对比度应至少在 5:1 以上；
- **附带的内容:** 对那些无效的用户界面组件的文字或文字图片没有最小对比度要求，因为那是纯粹的装饰，是附带的图片内文字，或者说任何人都无法看到的。

**注:** 可以通过在网页上或从页面内控制对比度而满足 1.4.3 和 1.4.6 条款。

通过对比度限制，该标准意在说明你应该提供一种提高颜色对比度的办法。

WCAG 2.0 正在设计之中，它将会具有对其他标准（特别是 WCAG 1.0 和 Section 508 法案）的高度向后兼容性。

### 用户界面的重要性

想想看，使一个网站的用户界面易于访问有多重要。即使内容无法以合适的形式来访问，一种可访问的用户界面也可以帮助用户确定感兴趣的内容，并通过一些外部帮助使其转换为他们能使用的形式。例如，一个听觉有障碍的人可能会提出，某个视频共享网站的视频讲座没有字幕。然而，由于 URL 是该视频的唯一标识，而且他们使用播放器观看视频，所以他们可将其提交给第三方，如免费的 [readOn](#) 项目服务，来制作字幕。

### 残障人士角色

一个理想的办法是在项目中将关键残障人士正确创建到其它用户角色中：在考虑那些特殊类型的用户如何使用一个网站时，采用虚构的用户作为原型。假设你正在评估某个视频共享网站的设计原型，其中的角色包括：

- 23 岁的 James Smith，一位足球迷，特别想与朋友分享体育要闻。
- 34 岁的 Sarah Maddison 是一位职业母亲，她可能一般没有时间进入视频共享网站。

但是她 3 岁的女儿很热衷于观看视频，并且莎拉想坐下来帮她女儿找到她想要观看的合适影片。

你可以采用这些角色并将残障人士纳入其中，包括（例如）：

- 视力受损人士。
- 色盲人士。
- 失明者。
- 耳聋者。
- 听力障碍者。
- 聋盲者。
- 癫痫。
- 阅读障碍者。

举例来说，你可能会决定，James 同时也是聋哑人，并希望将比赛录像的评论做成字幕，Sarah 视力低下，她阅读花式字体和微小文字很费力。这些角色会指导你拒绝那些在无字幕条件下，视频播放器就无法提供服务的设计原型，或那些在需要图片的地方使用复杂的文本标题的设计原型。

WAI 的残障人士如何使用网页和 Shawn Lawton Henry 的试问：如何在整个设计中整合可访问性的理念包含了更多受残障影响的角色的例子，可以让你轻松上路。

虽然没必要说明，但我们不能假定残障人士是可以互换角色的。残障的表现形式非常多，并且，除了残障之外，残障人士也具有无残障人士所有的各种差异，（例如）不同的性别、年龄、兴趣、价值观念和技能（也许与他们的计算机操作技能最相关）。

同样，与可访问性指南相对比，产品可以帮助填补角色所没有涉及到的空白。例如，也许你正遵循 WCAG 2.0 来创建视频共享网站，但角色中未包含一位癫痫用户。然而，你可以通过阅读指南 2.3（“癫痫发作：不要设计那些众所周知会引起癫痫发作的内容”）来决定该系统必须能够在视频显示前防止其闪烁。

### 选择一个可访问性标准

如果你需要选择某个可访问性标准以引起整个团队对可访问性的关注，或仅仅是需要一个可访问性标准以在测试时指导操作，我想建议你看一看 WCAG 2.0，因为它：

- 是围绕着适用于除 HTML 和 CSS 之外技术（如 Flash）的人类核心需求所设计的。
- 详细列出了每个一致性标准的原因。
- 为了满足一致性标准，提出了用当前技术实现的实际技巧。
- 确保了每一条款均可测试。
- 包含了比目前替代品更新的研究。
- 其设计广泛符合现有的可访问性标准。
- 将是一个国际标准。

你可以援引 WCAG 2.0 的某个特定草案；然而，出于市场营销的目的，最好也使其符合已完善的标准，如 Section 508 法案和 WCAG 1.0。

### 法律的精神

在依照指南进行测试时，重要的是要领会所有具体技术指导意见的基本原理：必须遵守法律的精神，而不仅仅是法律的文字。

下面是一个警示故事。Section 508 法案（§ 1194.22）包括一项规定，即：“任何非纯文本元素均应提供相同意义的文字说明（例如：通过在元素内容中使用 alt, longdesc）”。同样的，WCAG 1.0 也包括了如下的检查点：

为每个非纯文本元素提供一个相同意义的文字说明（例如：通过 alt, longdesc，或元素内容）。这包括：图片、图形化表示的文字（包括符号）、图像映射区域、动画（例如：动画 GIF），小应用程序和程序性对象、ASCII 艺术字符、框架、脚本、用作列表项目符号的图片、空白、图形按钮、声音（通过或不通过用户交互播放）、独立的音频文件、视频的音轨，以及视频。

可惜的是，许多阅读指南的人误解了该条款，而把修饰性元素与真正的文本等同起来，因而产生了类似这样的标记：

```

```

事实上，由于这些图片不传递任何新的信息，也不具有功能，对于这些图片来说，等效的文本应该是空的字符串（alt=""），这样屏幕阅读器就会直接跳过 alt 属性，而不必读出来。不得不听着阅读器一遍又一遍地朗读“fancy border”这样的文字对于一个屏幕阅读器用户来说是非常郁闷的，因为这样的文本并不能为他们提供任何有用的信息。

WCAG 2.0 试图定义得更明晰一些。等同准则中规定：“除以下所列情况，任何非纯文本内容均应提供一个等同的备选文字信息。”这些情况之一是：“修饰、格式化、不可见的：如果是纯粹的装饰，或只用于视觉格式化，或如果它不会显示给用户，那么它可以以一种会被辅助技术忽

略的方式来实施。”同样重要的是，WCAG 2.0 试图详细说明该准则背后的原因：

本准则的目的是确保所有非文本内容的文本也同样可以以文本方式获得。“文本”是指电子文本，而并非一个文字图片。电子文本具有独特的优势，即它在外观上是中立的。也就是说，它可以渲染成视觉上、听觉上、触觉上或其任何感官组合上的形式。因此，电子文字信息可以采用任何能最好地满足用户需求的形式来提交给用户。它也可以很方便地被放大，或者以一种易于理解的语言读出来，或被渲染成可以最好地满足用户需求的任何触觉形式。

### 应该由谁来测试？

进行测试的人基本可以分为两组：专家和用户。

专家测试是很重要的，因为专家懂得基本的网络技术是如何互相作用的，可以担当关于不同用户群的知识的采播中心的任务，并且愿意学习专用的测试工具。

用户测试是至关重要的，因为用户在其自身能力和自身辅助技术方面是真正的专家。用户测试还可以揭示不同技术水平的用户之间、熟悉当前网站的人（如专家测试者本身）和不熟悉当前网站的人（新用户）之间的可用性的差距。

知道如何使用屏幕阅读器的网页开发人员不可能与一般的屏幕阅读器用户一样；对于会编写自己的脚本程序的屏幕阅读器用户来说，他们也不可能与那些仅仅利用计算机来做象写邮件这类的普通任务的屏幕阅读器用户一样。

用户测试过程中获得的知识可在下一次测试中反馈到专家测试流程中（无论是在同一项目上的另一次测试，还是一个完全不同的项目）。用户测试还有一个更微妙的优势。通过使最终用户与开发员会面，可以增加创建可访问性网站的动力。

### 专家测试

专家测试有四个组成部分：

- **工具引导评估：**指的是运用工具寻找可访问性问题并将其提交给评估者（这包括可访问性校验工具和代码检查工具）。
- **筛检：**在这一部分中专家将模拟最终用户的网站体验。通常你不需要进行深入钻研来发现可访问性问题。你可能只需要在浏览器中打开网页，就会注意到某些文字非常难于阅读。
- **以工具为基础的检查：**在这一部分中，评估者将使用某种工具来调查网站的各个部分如何共同工作。
- **代码检查：**在这一部分里，评估者将直接查看代码和网站资源以查找问题。

虽然初学者可能特别依赖于工具引导评估，但是，各种经验水平的评估者都可以从每一个

组成部分中获益。甚至于初学者也可以在 **HTML** 标记内发现缺少同义文字说明的 `img` 元素，而当你获得更多的经验时，在你进展到更严格的测试之前，就可以更快的发现问题。对于从事更大项目的专家们来说，手动审查所有的客户端代码或检查网站的所有部份可能是不可行的，但是工具引导的评价却能够找到值得进一步研究的特殊问题区域。此外，人类评估者可能会忽略掉机器所发现的问题。

不幸的是，虽然有很多的可访问性工具，但其中大多数是存在这样那样的缺陷的。例如，一种在 **HTML** 文件中列出标题的工具犯了未在 `img` 元素中包含 `alt` 属性的错误。正如你应该牢牢记住法律的精神和标准一致性，你也应该记得工具是有缺陷的。在向某人抱怨某项可访问性问题之前，先确定这是一个真正的问题而不是一个工具错误。

### **半自动化可访问性校验工具**

一旦已经修复了第一次看到的问题，下一步就是将该网页扔给一个半自动化可访问性校验工具。如果你正在评估与特定标准的一致性，你可能会想要选择一个专门为你所使用的标准所设计的半自动化可访问性校验工具。

如果你正在评估与 **Section 508** 法案或 **WCAG 1.0** 的一致性，**Cynthia Says** 是一个不错的选择。如果你正在针对德国 **BITV 1.0 Level2**、意大利 **Stanca** 法或 **WCAG 2.0** 草案进行测试，目前唯一的选择是测试版的 **ATRC Web Acc** 可访问性校验工具。

这些工具有很大的局限性。没有类似于全自动化的可访问性测试。例如，鉴于目前人工智能的原始性质，计算机程序对一些文本是否真正的与上下文中的一张照片等同没有最后的发言权。即使那些理论上可以完全自动化的区域，校验工具程序员也可能会在诠释可访问性指南时犯错误，或是丢失法律文字下所代表的精神。

良好的工具会检查网页的可访问性问题，并生成一份它认为是错误的列表，以及另一份它认为值得人们去研究的问题。例如，如果 **Cynthia Says** 发现 `img` 元素带有 `alt=""`，就会发出警告（而不是发出一个错误！），指示用户“确认此图片只用于补白或设计，或不具有意义。”如果与那张图片等同的正确文本是一个空字符串，你就可以继续查看下一错误或警告。

也许可访问性校验工具的最大优势是，如果你选择了一个工具，比如说 **TAW 3**，该工具可以同时检测多个 **URL**，你就有可能找到需要进一步检查的网页大集合。

### **结构检查工具**

许多检查工具的设计目的是探测网页内容的结构。结构，简单来说，就是界定一个网站有哪些组成部分，以及它们之间的相互关系如何。例如，在 **HTML** 文档对象模型（**DOM**）中，文字可以通过 `label` 元素被指定为一个表单域的标签。浏览器将 **HTML** 解析成一个文档对象模型。该浏览器会将各种行为与特定的组成部分联系起来。例如，如果你按一下复选框的标签，通

常它们就会被勾选。

桌面环境和应用程序通过支持屏幕阅读器、语音识别软件及其他辅助性技术，来提供与可视情况下相同的结构与内容。在 Windows 环境下，主要结构系统称之为微软主动式辅助 (MSAA) 或 Vista 上的用户界面自动化。例如，一个对话框上含有一系列相关的子元素，如对话框的标题、区域、按钮和它们的标签等。

典型的辅助技术主要是从这些结构系统的角度来处理浏览器和插件对网页内容的表示的，而并非直接处理网页文件对象模型。

有些检查工具既可检查桌面级别的结构又可检查网页级别的对象模型。在桌面级别方面，OS X 配备了 可访问性检查工具和 可访问性核查工具。Microsoft 提供了 Microsoft Active Accessibility 2.0 和 Microsoft Active Accessibility 1.3 的检查工具。Accerciser 工具可用于 GNOME 辅助技术—SPI API。

查看 (X) HTML 文档对象模型的工具，包括在 Opera Dragonfly 和 Firebug 中的 DOM 检查，和可访问性工具包，如 专用于网络浏览器和 Opera 的 Web 可访问性工具栏以及 ICITA Firefox 可访问性工具栏。

DOM 检查工具显示了构造 (X) HTML 的元素和属性以及文本树，而 web 可访问性检查工具则提取特定的组成部分或关系，并将其列表说明。例如，它可能会列出所有的表单域及其标签，或全部的标题，或是全部的链接。

如果你认为某种浏览器将正确的 (X) HTML 结构向辅助技术描述得不够正确，那么你也可能会想要调查这一层次，但是通常对于 (X) HTML 来说，挖掘可访问性模型不是必要的。相反，通常你会直接检查 (X) HTML 结构。

并非所有的内容都可以用 DOM 或 Web 可访问性检查工具来检查。检查那些内容插件（媒体播放器、Flash 内容和 Java 小应用程序）所使用的可访问性辅助技术也是非常重要的。

一般情况下，你应该检查是否所有的控件都在模型之内扮演适当的角色（举例来说就是，文本框就是文本框，按钮就是按钮）并具有必要的属性。

### 调查和使用最终用户辅助技术

调查涉及在测试时模拟残障人士的体验。这种调查可能使用辅助技术与一个网站交互，或试图以某种方式限制一个人的能力。例如：

- 当测试键盘可访问性时，使用一个口含棒来按键。
- 采用色盲模拟工具来 (Vischeck simulator) 观看网页，该工具会显示出那些带有不同形式色盲的人士所看到的网页和其中所包含的图片。

- 关闭显示器，同时配合浏览器使用屏幕读取器。

调查可以帮助开发人员创建对残障人士需求的评定，还可以揭示基本的设计缺陷。使用辅助技术可以消除某些关于支持或不支持 web 交互标准的误解。举例来说，流行的屏幕阅读器没有用为 `aural` 或 `braille` CSS 媒体类型所推荐的样式，而是试图去描述可视化浏览器所显示的 `screen` 类型。

使用辅助技术不是一项可以掉以轻心的任务，因为如果要很好地了解如何使用该系统，就需要某种程度的投入和培训。此外，还存在一种制造新误解的严重风险。开发人员可能会努力地想要对某种屏幕阅读器进行一些处理，并且臆断该屏幕阅读器在某方面还有不足，但这其实反映了他们对这种工具缺乏专业知识。他们可能会尝试以错误的方式使用工具，例如尝试按顺序读取一个网页，而这种情况下，一个真正的屏幕阅读器用户会利用标题和其他元素跳来跳去地寻找自己感兴趣的内容。换句话说，他们也可能无法正确阅读屏幕。使用屏幕阅读器仔细阅读一个你可以用肉眼看见或很了解的网页，同探索一个你无法看见的全新网站是有很大差异的。

辅助技术的使用需要依据用户在日常生活中运用该技术的经验，并且由此获得的经验总结理想情况下应该通过专家用户所确认。整体说来，初学者最好将辅助技术的使用留给用户测试者处理。

### 细节检查

一旦所有由你选定的检查工具所确定的真实问题已得到修复，你就可以进入手动测试、探索并回顾该项目阶段了。

WCAG 2.0 将其最佳实践标准分为四个原则。网页内容和功能必须是：

- 可感知的（例如，图像应该有文本对应体）。
- 可操作的（例如，用户应该可以不用鼠标也能与某个网站进行交互，并且可以通过屏幕阅读器来进行导航）。
- 可理解的（例如，正文不应该比它需要的更加复杂，且网站应以可预测的方式来运行）。
- 健壮性（例如，不同的用户代理能够一起使用网站，且导航应该是一致的）。

在这一节中，我将提出一些实例，说明测试专家可以怎样对网页内容与上述原则的一致性进行评估。请注意本节并不能代替对 WCAG 及其技术的综述。

### 可感知性

可感知性问题的子集之一是围绕着对各种类型的媒体提供替代内容而展开的。你可以通过在浏览器中关闭图像和多媒体技术并查看网页来测试文本对应体。但是，你需要特别关注 `img`

和 `input` 元素。通常情况下，建议你将所有纯粹装饰性图像的 `alt` 属性设置成空白 (`alt=""`)，从而使屏幕阅读器直接跳过它们。然而，在下述情况下：

- 图像是链接的唯一内容
- 表单按钮

如果给这些因素设定 `alt=""` 属性的话，屏幕阅读器通常就会按照 `alt=""` 属性缺失的情况来对图像或按钮进行处理，并会试图提供一个属性值（例如，通过读出图像的网址）。

因此，在这些特殊情况下，你应当确保链接或按钮内的图像具有 `alt` 属性，以说明链接的目标或按钮的动作，即使这样做有点啰嗦。

对与多媒体同步的替代内容的测试，如字幕和音频描述，可以通过对媒体播放器参数进行挖掘，从而显示出可访问性设置而达成。

另一组可感知性问题涉及到对网页进行样式化。这里有三个需要调查的方面：

- 该页面的建议外观的可访问性很高吗？举例来说，是否有足够的色彩对比度？文本大小是否看着比较舒服？除了自己一个人对着网页眯眼，你可以使用诸如 [Juicy Studio CSS](#) 分析器之类的工具，这些工具可以依照某些度量易读性的公式来检查背景色和前景色的组合。
- 发布者提出的外观建议能否切实地与普通用户的参数相结合？这些参数旨在使页面内容更易读，比如增加字体大小、放大和使用各种缺省颜色。试着通过约 2-5 个步骤来增加文字大小；不必担心结果在像素上是不是完美的，而应关心布局是否被破坏了，从而导致内容难以阅读。试着改变你的颜色参数，看看会发生什么。如果发行商的 CSS 设置了颜色，它应该是将背景和前景色一起显式地指定的，以确保独特的用户偏好和发行商样式的结合不会导致无法阅读或出现不可见的文本。流行的浏览器允许用户实施他们自己的色彩偏好，并关闭 CSS 背景图像。如果你自己如此尝试的话，就会发现错误的 CSS 图像置换技术使得文本不显示在屏幕上，因为虽然图像不会被载入，但文字仍会看不见。
- 如果发布者的外观建议被弃置不用，那么所有由该建议表达的信息是由用户代理的默认样式还是由用户自定义样式来显示的？不妨尝试关闭 CSS，并检查文档对象模型，校验标题是否标记为标题，表格是否被用于没有布局的表格化数据。

## 可操作性

虽然很少被考虑到，但对于提高网站的可操作性来说，健康和安全是至关重要的一部分。闪烁性内容具有触发光敏性癫痫的风险。你可以对正在使用中的网站进行屏幕捕捉，并将其提交给 [跟踪中心光敏性癫痫分析工具 \(PEAT\)](#) 以测试该页面是否具有可能对用户造成危险的闪烁内

容。显然，如果你要创建一个视频共享网站，这是一个特别值得关注的问题。在产品设计阶段，你可能得在网站中包含一个自动化的上传筛选程序。

除此之外，提高一个测试网站可操作性的最好办法，就是去观察你是否能够使用不同设备访问所有的基本内容和功能：

- 试着只用键盘使用你的网站。当前的焦点总是清楚地指示出来了吗？所有的功能都可以通过键盘获取吗？
- 试着以触摸屏设备使用你的网站。
- 试着通过 Windows 版的 Opera 及其语音附加组件，或 Windows Vista 语音识别及 IE，用语音命令对网页进行导航（注：商业的语音识别软件最近已经以 MacSpeech Dictate 的形式引入到 Mac OS X 中，但目前在免费的 \*nix 平台上还没有对应产品）。

屏幕阅读器和其它辅助技术可以充分利用 (X)HTML 语义结构来正确地关联内容，并启用内容导航。例如，屏幕阅读器可以让用户跳转到下一个出现的标题或其它元素类型，也可以列出某一特定类型发生的所有事件。正确使用 `label` 和 `legend` 元素可以让辅助技术将标签关联到正确的表单域上；正确使用 `th` 元素和 `header`、`scope` 和 `axis` 可以让辅助技术把表格标题和表格数据单元格关联起来。语义结构可以用 Opera Dragonfly 中的文档对象模型 (DOM) 检查工具来进行评估。像 Firefox 可访问性扩展之类的可访问性检查工具可以使工作变得更容易，例如，可以通过列出网页上的标题，或列出表单域的属性（快速地显示出哪些缺失了相关标签）来实现。具体示例参见图 1。

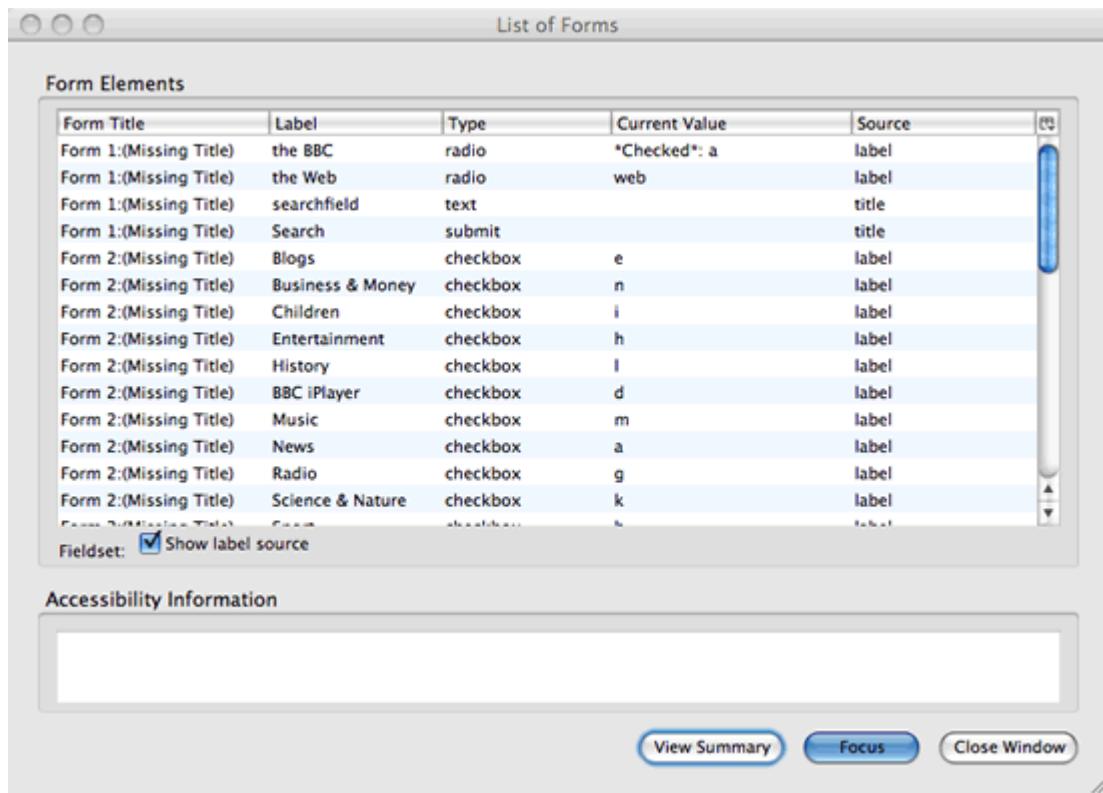


图1：BBC 新主页的 Firefox 可访问性扩展信息的屏幕截图。

## 可理解性

评估可理解性比评估易读性更加主观。除非评价者对一个项目不熟悉或是一个专业校订者，否则其他人也许不是评估正文可理解性的最佳人选。然而，你可以使用 Juicy Studio 易读性测试工具来得到网站的正文有多简单的一个粗略概念。

不过，客观说来，有些方面是高度客观可测试的，例如内容是否具有语言元数据，以使（例如）屏幕阅读器和语音浏览器用正确发音将页面内容读出来。对于 HTML，你可以使用一个 DOM 检查工具检查某份文档中的 lang 属性是否存在以及每一种语言的变化。

留意网站中不一致的地方，无论从内部一致性角度还是来自于互联网公共惯例的可预测角度都可以。每次只能看到一页中部分内容的屏幕放大镜用户极大地依赖于这种一致性，这样他们才能知道在哪里可以找到特定的内容和功能。

## 健壮性

测试页面内容是否健壮涉及到检查技术的使用是否正确。在非常基础的水平上，你可以用下面这些工具来检测标记和代码，如：

- 启用了警告的 WDG HTML 验证器
- W3C 的 CSS 验证器

- [JSLint 的 JavaScript linter](#)

接下来，你可以对代码进行深入检查，以检验功能是否得到了正确使用。例如，你可以检验是否用的是 `HTML` 原装控件，而不是那些含有无意义元素和 `JavaScript` 的伪造控件，并且你还可检验 `JavaScript` 是否在可能范围内尽量使用了特征检测而不是浏览器嗅探法。

然后，你就可以在多个用户代理和辅助技术情况下进行测试，检验该网站是否是可感知的，可操作的，并且是可理解的，无论发布者 `CSS`, `JavaScript` 和插件的组合是怎样的，也不管这种组合是启用还是禁用的。

最常见的问题可能是生硬的 `JavaScript`，比如，网页中的锚记和按钮实际上却要依赖于 `JavaScript` 才有效。还有更微妙的问题，这些问题产生于 `JavaScript` 与技术堆栈中其他层次过于紧密的耦合。例如，`JavaScript` 可能应用 `CSS` 的 `display: none;` 来隐藏内容，但在没有应用发行商 `CSS` 的情况下，又会发生什么呢？

另一个例子是就多媒体控件。通常，当插件内容被包含在内时，该插件的原始用户界面会被禁用，并且该插件会由脚本化的 `HTML` 窗口小部件所控制。只有经过基于 `JavaScript` 的插件检测后，插件的内容通过 `JavaScript` 得以添加时，它才会运转正常。但有时插件的内容包含在预制脚本状态的页面内。在这种情况下，就不仅要检验以确保该网站包含了一种防止正在进行处理的插件无法使用的备选方案，还要检验以确保该插件的原始用户界面在 `JavaScript` 被禁用的情况下依然可运行。如果前者的情况并非如此，那么用户根本不会看到备选方案的内容；如果后者不为真，那么用户将可以看到该插件，但却无法控制它。

## 用户测试

开发员的检查和筛选再多，都无法取代用户和网站的原始冲突。由于难于了解 `web` 内容和辅助技术之间的所有微妙的相互作用，而且难以准确地估计残疾人用户的体验，用户测试对于残障用户来说就更重要了。如果可能的话，你应该让真正的残障用户来测试你的网站。规模庞大而昂贵的用户测试可以做到这一点，但也不要低估小规模用户测试所能带来的好处。

### 招募测试者

测试者通常可以象你寻找易用性测试候选人一样的办法去寻找（例如通过广告和中介机构）。当地的残障人士组织应该可以推荐适当的论坛以招募测试对象。

测试实际上也是工作，因此最好能提供报酬。对用户测试而言，大约 70 美元一个小时的报酬是很常见的。

你可能可以找到愿意免费对小型项目进行测试的人员。在你的朋友、亲戚和同事之中会有一些残障人士。此外，还有一些致力于软件的可访问性问题的在线讨论小组，如：

- [WebAIM Accessibility Discussion List](#)。
- [Web Accessibility Initiative Interest Group Mailing List](#): 一个讨论 Web 可访问性相关问题的论坛。
- [British Computer Association of the Blind mailing list](#): 讨论视障人群的信息通讯技术(ICT)。
- [Magnifiers Yahoo! Group](#).
- [jfw@freelists.org](mailto:jfw@freelists.org) :一份针对 JAWS 屏幕阅读器用户的邮件发送列表。
- [GW-Info](#):一份针对 GW Micro Window-Eyes 屏幕阅读器用户的邮件发送列表。
- [Dolphin software users Yahoo! Group](#).
- [NVDA users mailing list](#).
- [Thunder users mailing list](#).
- [discuss@macvisionaries.com](mailto:discuss@macvisionaries.com) : 使用 OS X 的盲人名单。
- [macvoiceover@freelists.org](mailto:macvoiceover@freelists.org) : 苹果 VoiceOver 用户.
- [Blinux-list](#): 使用 Linux 的盲人和视障人士用户论坛名单。
- [GNOME Orca users](#).
- [Ai Squared Forums](#):包括流行的扩视放大镜的用户。
- [Deaf-Macs Yahoo! Group](#):专门针对聋人、听觉障碍者及其教员或聋盲人的 Mac 用户。
- [deaf-uk-technology Yahoo! Group](#) : 聋人相关技术讨论

这些小组通常都欢迎来自于 web 开发员关于他们网站的可访问性或特定技术的提问。

### 实际的考虑

请记住，测试环境本身也应该是可访问的。例如，如果你正准备笔试材料，你必须同时准备提供这些笔试材料的替代品。在常规的测试区域中，复现用户浏览环境的后勤工作非常复杂，所以在用户的家中测试可能更切合实际。如果做不到这一点，远程测试也是极有用的。

对于用户所熟悉的技术的特殊考虑，可能对残障用户来说，比其他用户更为重要。辅助技术能给他们的计算机体验添加多层次的复杂性，从而在新手计算机用户和老手计算机用户之间产生大的分歧，并将用户划分成不同的团体，这些团体可能对他们自己的配置非常熟练，而对不熟悉的技术则会十分困惑。(想想那些没有残障以影响到自己对计算机的使用的用户在 Mac 和 PC 之间切换是多为难啊！)

如果你找到一个长期使用 Window-Eyes 屏幕阅读器的用户，而让他坐在一台安装了对他来说很陌生的 JAWS 屏幕阅读器的机器前，并请他来测试一个网站，那么，到底是他对 JAWS

的使用有问题还是对网站的使用有问题，将会变得非常难于区分。即便你提供了 Window-Eyes，由于版本之间存在着显著的差异，以及用户可能会自定义设置，这些情况仍可能使区分问题变得很困难。有鉴于此，除非你正好在专门测试网站的可访问性在陌生的设置（例如在图书馆内或朋友的电脑上）下能有多好，否则最好还是允许用户采用他们自己的设置进行测试，或者尽可能接近这种设置。

同样，除非你要专门测试新手用户或专家用户，否则你就应该选择那些对使用自己的当前设置访问网络大约有一年经验的用户。不管是辅助技术还是网络自身的协议都是不容易学会的。选择新手用户，你将不知道问题是出自于你的网站还是学习过程中的固有问题，专家们则可能有其他人所没有的技巧。

### **选择任务**

即使是对用户探索某个网站时进行观察，也可以得到令人难以置信的启发。正如其它任何的用户测试一样：

- 尝试为用户设置一些具体的需要完成的任务。
- 询问他们的想法，倾听他们的发言。
- 关注他们所做的事情，因为这可能有别于他们所说的话：偏好陈述是行动的最差指南。

在设计网站时，你需要把重点放在那些用户希望要于完成的事务上，而不是他们所需要使用的特殊控件上。同样，在进行可访问性测试时，你所制定的任务应该（至少在开头的时候）反映出使用网站的访问者的真实目的，而不是专注于他们与特殊控件之间的交互。这些事务对于有或没有残障的人来说通常是类似的。

例如，如果你正在测试一个视频共享网站的可访问性，不要在一开始时就询问他们是否可以使用特殊控件（“这是音量滑块。你可以调整音量吗？”）。相反，给他们一些情景，并请他们完成关键用户任务。例如：

- 浏览视频并选择一个去播放。
- 搜索视频。
- 上传视频。
- 暂停视频、播放视频、使视频静音、取消视频静音、快退和再次播放视频。
- 评估视频。
- 与朋友分享视频。

以这种方式，你很可能你会发现很多预料之外的问题。例如，屏幕阅读器用户可能无法找到

搜索框或视频控件。相反，用户可能已经有导航策略来处理甚至连你都不知道的网页。

### 解释结果

在理想的环境下，我们可以测试各种可能的组合，并且从每个人身上得到反馈。但在现实中，时间和金钱会限制用户测试。有鉴于此，反馈可能是一把双刃剑。虽然它可以教会你很多，但也存在着过于重视某个人的看法的危险，某个人的看法可能对更多的目标受众不具有代表性。例如，某些屏幕阅读器用户倾向于寻找一个为盲人用户简化了的体验；另一些则急于知道有关他们的非盲人朋友和同事所看到的网站的一切。

这就是像 WCAG 这样的可访问性标准真正获得承认的原因。通过遵循这样的指南，你就比较可能创建出即使那些无法测试到的用户群也可以得到满足的基本可访问性。

当你在观察问题，分析问题产生的原因时。例如，你的视频共享网站包含一个显示热门影片数据表的网页，各栏包括定格画面、标题、上传日期、上次播放日期、总体评价，并以视频目录的形式排列在横排组内。在用户测试方面，屏幕阅读器用户可能无法使用数据表。这可能反映了：

- 网站代码有问题。举例来说，也许是开发人员用的是毫无意义的 `div` 元素来创建数据表，而不是使用正确的数据表标记。在这里，适当的做法是对表格重新编码。
- 对于部分用户缺乏专门知识。例如，JAWS 用户可能不熟悉 JAWS 的浏览和阅读数据表的特性。这里适当的做法是对不那么专业的用户提供补充文件或提示。如果专家用户也不能完成理想的测试目标，他们则可以成为很好的顾问。
- 用户代理有问题。例如，Safari 向 Apple 可访问性模型显示数据表时，是以一系列布局框的形式，而不是以一组数据关系的形式。在这方面适当的做法包括向用户代理发行商或开发商报告漏洞，研究一种能在该用户代理中起作用的技术，或注意文件方面的限制，并提供一些建议，告诉人们哪些用户代理的替代品能够正确浏览你的网站。
- 屏幕阅读器有问题。例如，开发者可能会使用 `abbr` 属性缩短表头长度，但屏幕阅读器可能无法阅读缩短后的版本。在这方面适当的做法包括向屏幕阅读器发行商或开发商报告漏洞，也可以是寻求一种能在该屏幕阅读器上起作用的技术，或注意到文件方面的限制，并提供一些建议，告诉人们哪些替代工具或导航策略能够正确浏览你的网站。

### 对可访问性测试结果进行沟通

在就可访问性评估的结果进行沟通时，应当精确地记录所评估的内容。如果你针对某个特定标准的一致性进行了测试，那么你就应该具体说明一致性在哪些方面做得好，在哪些方面做得不好。在提出问题的时候，请先用实在的，人性化的术语提问，并说明该问题可能会怎样对用户

产生不利的影响。还要描述出如何重现问题并测试其是否已经解决的办法。还应当为实现一致性或改善可访问性推荐实用技术。

例如，你可能会像以下这样报告视频共享网站存在的某个问题：

- **问题:** 不使用鼠标悬停在菜单项目的顶部时，下拉菜单就无法打开，并且当你通过 TAB 键移动菜单时，键盘焦点会消失于屏幕之外。
- **如何重现:** 在浏览器内打开网页，并尽力试着仅使用键盘访问某个菜单的子项目。
- **说明:** 网站导航应独立于设备，因此，使用除鼠标之外设备的用户，如盲人用户或运动残障人用户，也可以访问内容和功能。目前，此类用户无法访问项目的子菜单，并且，当聚焦指示器出现时，使用键盘的非盲人用户可能会混淆。
- **一致性含义:** 键盘操作性是 WCAG 1.0 和 WCAG 2.0 一致性的“A”级的要求（见 WCAG 1.0 指南 9 和 WCAG 2.0 指南 2.1）。
- **建议的补救措施:** 当 JavaScript 无法使用时，可使用一简单的链接列表将导航的每个子列表链接到子页面上。并在这些子页面上，显示出带子列表的主导航栏。当 JavaScript 可用时，从 DOM 中删除子列表并为每个菜单项的子列表添加 click 事件，该事件可同样地由键盘、鼠标、语音识别和触摸屏所触发。

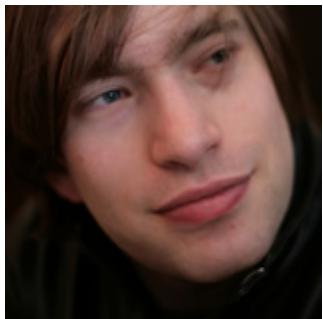
## 总结

并非每个网页都能得到专家们或付费测试者的可访问性评估。但是，任何网页开发人员都可以学习可访问性的原则，在代码中尽力贯彻这些准则，并通过邮件列表向用户发布其劳动成果，以了解进一步的问题，从而为未来的开发获取新知识。

## 练习题

- 试着不用鼠标来对你选择的某个复杂的网站进行导航。你遇到什么困难么？网站开发人员如何才能帮助你呢？
- 关闭 CSS，然后进行你每天的网页浏览。你遇到了什么问题？
- 关闭 JavaScript，然后进行你每天的网页浏览。你遇到了什么问题？
- 选择一个最喜欢的网站，为该网站设计一些角色，然后作为一个专家测试者来评估其是否满足 WCAG 1.0 和常规的可访问性。为一个网站设计一个用户测试计划，包括征募需求和任务以进行测试。写一份关于如何改善其可访问性的报告。
- 上一篇：可访问性基础知识
- 下一篇：CSS 基础知识

## 作者简介



在大学里研究完世纪国王们、十八世纪的科学家们以及其他历史奇人之后，**Benjamin Hawkes-Lewis** 不知怎么搞的结束了作为雅虎 Web 开发人员的工作，更多地投入到自己的兴趣之中。他最喜欢的事情包括与朋友共进一餐，在电影院里看场好电影，在草地上懒洋洋地晒会儿太阳，以及通过参考一些资料、基本原理和实验证据而解决各类难题。

## 27. CSS 基础知识

- 网络标准教程

作者: Christian Heilmann · 2008 年 9 月 26 日

- 上一篇: 可访问性测试
- 下一篇: 继承和层叠
- 目录

### 序言

在本教程前面的文章中, 我们讨论了网站的内容以及如何通过 **HTML** 来对网站内容进行组织。**HTML** 是非常重要的, 因为这意味着我们给文档赋予了意义, 并为与其它技术的紧密结合提供了便利。接下来将要讨论的重要 **web** 技术是 **CSS** (层叠样式表), 该技术是用来样式化 **HTML**, 并在网页中对其进行定位的。在本文中, 我将向你们介绍 **CSS**——什么是 **CSS**, 如何将其应用于 **HTML**, 以及基础的 **CSS** 语法是怎样的。本文的结构如下:

- 什么是 **CSS**?
- 定义样式规则
  - **CSS** 注释
  - 对选择器进行分组
- 高级 **CSS** 选择器
  - 通配选择器
  - 属性选择器
  - 子对象选择器
  - 派生选择器
  - 邻接选择器
  - 伪类
  - 伪元素
- **CSS** 缩写
  - 属性值全写与缩写的对比
  - 缩写的边距属性可以不必将四个值全都列出来

- 列出所有属性值还是使用缩写
- 其它缩写参考

- 将 CSS 应用到 HTML
  - 内联样式
  - 嵌入样式
  - 外部样式表
  - @importing 样式表
- 总结
- 练习题

## 什么是 CSS?

在使用 `HTML` 对文件进行结构化，并告诉浏览器某个元素的功能时（这是到其它页面的链接吗？是标题吗？），`CSS` 向浏览器发出如何显示特定元素的指示——比如样式，空格和定位，如果说 `HTML` 是构建房屋结构的梁柱和砖块，`CSS` 就是装修用的灰泥和涂料。

这是通过一套规则体系来实现的，接下来你将学到更多关于这些规则的具体的语法。这些规则指定了哪些 `HTML` 元素应该添加样式，然后在每个规则内列出它们想要对其进行操作的 `HTML` 元素的属性（比如颜色，大小，字体等等），以及它们想要将这些属性改成什么样的值。比如，某条 `CSS` 规则可能表示“我想要找出所有的 `h2` 元素，并将它们全部涂成绿色”，或者“我想要找出所有类名为 `author-name` 的段落，将它们的背景设为红色，并将里面的文本大小设为普通段落文本的两倍，然后给每个这样的段落加 10 个像素的间距。

`CSS` 不是 `Jacascript` 那样的程序语言，也不是 `HTML` 那样的标记语言——实际上没有可以和它相对比的语言。在早期的 `web` 开发中，那些用于对界面进行定义的技术总是把外观和结构混为一谈，但在网络这种时常变化的环境中，这可不是一个明智的做法，这也就是发明 `CSS` 的原因。

## 定义样式规则

闲话少说，我们先来看一段 `CSS` 代码示例，然后对其进行分析：

```
selector {
 property1:value;
 property2:value;
 property3:value;
}
```

相关部分的说明如下：

- 选择器用于确定将要应用规则的 **HTML** 元素，这可以通过实际元素名，比如 `body`，或者像 `class` 属性值之类的方式来确定的，稍后我们将看到更具体的例子。
- 大括号中包含了属性/属性值对，用分号隔开；属性与它们各自的值用冒号隔开。
- 属性将定义你想对所选择的元素定义何种样式。属性有很多种，可以用来改变元素的颜色，背景色，位置，边距，填充距，字体类型以及许多其它东西。
- 所谓的属性值就是你想要将所选元素的某个属性改变成哪个值。值取决于属性，比如决定颜色的属性可以采用十六进制的颜色值来表示，例如 `#336699`，还有 RGB 值，例如 `rgb(12,134,22)`，或象 `red`、`green` 或 `blue` 之类的色彩名称。决定位置、边距，宽度、高度等等的属性可以用像素，`em`，百分比、厘米或其它单位来衡量。

下面我们来看一个具体的例子：

```
p {
 margin:5px;
 font-family:arial;
 color:blue;
}
```

这条规则所选择的 **HTML** 元素是 `p`——**HTML** 文档中的所有 `p` 元素都将应用该 **CSS** 规则，除非它们还应用了更具体的规则，在这种情况下，更具体的规则将会覆盖这条规则。受这条规则影响的属性是段落周围的边距，段落中文本的字体，以及该文本的颜色。这里边距被设置为 5 个像素，字体设置为 `Arial`，文本颜色设置为蓝色。

我们等会将回顾所有这些细节——本文的主要目的是介绍 **CSS** 基础，而不是过于具体的细节。

所有这些规则共同构成了一张样式表。这就是最基础的 **CSS** 语法。还有更多的语法，但不太——也许 **CSS** 最酷的地方就是它的简洁性。

### **CSS 注释**

你从一开始就应该知道如何在 **CSS** 中进行注释。你可以在 `/*` 和 `*/` 之间添加注释。注释可以占好几行，而且浏览器将忽略它们：

```
/* These are basic element selectors */
```

```
selector{
 property1:value;
 /*
 property2:value;
 property3:value;
 */
}
```

你既可以在规则之间，也可以在属性模块内部添加注释，比如，在下面的 CSS 中，第二个和第三个属性都在注释分隔符中，因此它们将被浏览器忽略。在检查 CSS 的某个部分对网页有什么影响时，这一招很管用。你只需要将它们标记为注释，保存 CSS，重新加载 HTML 就可以了。

```
selector{
 property1:value;
 /*
 property2:value;
 property3:value;
 */
}
```

与其它语言不同的是，CSS 仅有块级注释——单行注释是不存在的。当然如果你愿意，你也可以强制使用单行注释，但你还是需要用到开头和结尾的注释分隔符（/\* 和 \*/）。

### 对选择器进行分组

你同样可以对不同的选择器进行分组，如果你想将同样的样式应用到 h1 和 p 元素上——你可以编写如下的 CSS：

```
h1 {color:red}

p {color:red}
```

但这不是很理想，因为你重复了同样的信息。因此你可以通过使用一个逗号来将选择器分组，从而简化 CSS——花括号内的规则将应用到两个选择器中：

```
h1, p {color:red}
```

选择器有许多不同的类型，每个都与标记的不同部分相匹配。最常遇到的三种基本选择器如下：

`p {}`: 元素选择器

与页面上所有叫该名称的元素相匹配（比如示例中的 `p` 元素）

`.example{}`: 类选择器

与所有 `class` 属性为指定值的元素相匹配，因此上面可以匹配 `<p class="example">`, `<li class="example">` 或者 `<div class="example">`，或者其它 `class` 值为 `example` 的元素。注意，类选择器不会对任何特定的元素名称进行测试。

`#example{}`: id 选择器

与所有具有特定 `id` 属性值的元素相匹配，因此上面的例子可以匹配 `<p id="example">`, `<li id="example">` 或者 `<div id="example">` 或者其它 `id` 值为 `example` 的元素。注意 ID 选择器不会对任何元素名称进行测试，而且每个 ID 值在一个 HTML 文档中只能出现一次——它们对每个页面都是独一无二的。

你可以在下面的例子中看到上述选择器的使用方法。注意，当你在浏览器中打开该示例的时候，`warning` 样式会被应用到列表项和段落上（如果项目符号消失了，那是因为你在白色背景上设置了白色文本）。

- [示例- `selectors.html`](#)
- [`selectors.css`](#)

你可以将一些选择器联合起来定义更具体的规则：

```
p.warning{}
```

匹配 `class` 值为 `warning` 的所有段落

```
div#example{}
```

匹配 `id` 属性值为 `example` 的元素，但只适用于 `div` 元素

```
p.info, li.highlight{}
```

匹配 `class` 值为 `info` 的段落元素以及 `class` 值为 `highlight` 的列表元素

在下面的例子中我会用它们来区分各种警告样式：

- [示例- specificselectors.html](#)
- [specificselectors.css](#)

## 高级 CSS 选择器

在上面的章节中，我向你们介绍了最基本的 CSS 选择器，元素，类和 ID 选择器。有了这些选择器，你可以完成许多任务，但这当然不是全部的选择器——还有其它选择器可以让你用更具体的标准来样式化所选定的元素。

- **通配选择器：**通配选择器可以用于选择页面内的任何元素。
- **属性选择器：**就如它们的名称所指的那样，属性选择器让你能根据属性选择元素。
- **子对象选择器：**如果你想选择某个特定元素，而该元素又是其它特定元素的子元素，你就可以使用这种选择器。
- **派生选择器：**如果你想选择某个特定元素，而该元素又是其它特定元素所派生出的元素（不仅是直接子对象，也可以是树形目录中更下层的对象），你就可以用这个类型的选择器。
- **邻接选择器：**如果你想选择跟在其它特定元素后面的特定元素，可以用这些选择器。
- **伪类：**它们使你能够不基于元素的性质，而是基于更复杂的因素来样式化页面元素——比如链接状态（比如，当它们处于鼠标悬停状态，或是已经被访问过）。
- **伪元素：**它们让你能样式化元素的特定部分，而不是整个元素（比如该某个元素内的第一个字母）；它们也能让你在特定元素的前后插入内容。

随着对本教程剩下部分的深入学习，你将看到一些更加复杂的选择器的相关资料，如果你不能马上理解全部的内容，也不必担心——随着你对样式化网页的经验积累，你一定会取得成功的！最好是从上面提到的 3 个基本选择器开始学习，随着你信心的增加，再深入到其它的选择器。

### 通配选择器

简而言之，通配选择器选择页面的所有元素进行样式化。比如说，下面的规则将对页面的每个元素都添加一个 1 像素的实心黑色边框。

```
* {
 border: 1px solid #000000;
}
```

### 属性选择器

属性选择器让你能基于元素所包含的属性来对它们进行选择。比如，你可以用下面的选择器来选择所有具有 alt 属性的 img 元素：

```
img[alt] {

 border: 1px solid #000000;

}
```

请注意方括符。

使用以上选择器，你就可以对具有 `alt` 属性的所有图像都添加一个黑色边框，并对其它图像设置一个亮红色边框——这招对可访问性测试很有用。

当你通过属性值，而不仅仅是属性名称来进行选择的时候，属性选择器会变得更加有用。

下面的规则适用于所有 `src` 属性值为 `alert.gif` 图像：

```
img[src="alert.gif"] {

 border: 1px solid #000000;

}
```

你可能不会觉得这非常有用，但是，在调试方面它是很管用的。更有用的是它能够选择属性的特定部分，比如文件扩展名。而且这项功能正在完善中——实际上，CSS3 引入了 3 种新类型的属性选择器，可以基于属性值中的文本串（可以位于属性值中开始、结束或任何的地方）来进行选择。[点此阅读 Christopher Schmitt 关于 CSS3 属性选择器的文章。](#)

## 子对象选择器

你可以用一个子对象选择器来选择作为其它特定元素子对象的元素。比如，下面的规则会把作为 `h3` 元素子对象的 `strong` 元素中的文本转换为蓝色，但不会转换其它的 `strong` 元素。

```
h3 > strong {

 color: blue;

}
```

IE6 及以下版本的浏览器不支持子对象选择器。

## 派生选择器

派生选择器与子对象选择器非常相似，只是子对象选择器仅选择 **直接** 派生元素；派生选择器则可以选择元素层级中的任何适当元素，而不仅仅是直接派生元素。让我们更仔细的看看它的含义，细想以下 HTML 片断：

```
<div>
 hello

 <p>In this paragraph I will say
 goodbye.

 </p>

</div>
```

在这个片断里，`div` 元素是所有其它元素的父对象。它有两个子对象，一个 `em` 和一个 `p`，`p` 元素还有一个独生的子元素，即另一个 `em`。

你可以用一个子对象选择器来选择紧接在 `div` 里面的那个 `em`，像这样：

```
div > em {

 ...

}
```

如果你换用一个派生选择器，就成了这样了：

```
div em {

 ...

}
```

这样两个 `em` 元素都将被选中。

## 邻接选择器

这种选择器让你能在同一层级的元素层次中，选择跟在其它特定元素后的某个特定元素。比如，如果你想选择紧跟在 `h2` 元素后面的所有 `p` 元素，而非其它 `p` 元素，你可以使用以下规则：

```
h2 + p {
 ...
}
```

IE6 及以下版本的浏览器不支持邻接选择器

## 伪类

伪类不是用来为元素提供样式的，而是为元素的各种状态提供样式。最常见的用法就是样式化链接状态，因此我们先看看这部分。下面的列表给出了各种伪类，以及它们所选择的链接状态的描述：

- `:link` — 链接的一般默认状态，就是你刚刚看到它们时所处的状态。
- `:visited` — 在你正在使用的浏览器已经访问过的链接。
- `:focus` — 目前键盘光标正位于其中的链接（或表单域，或其它元素）。
- `:hover` — 鼠标指针目前正悬停于其上的链接。
- `:active` — 目前正在点击的链接。

以下 CSS 规则将默认状态下的链接设置为蓝色（大多数浏览器的默认设置）。如果鼠标悬停的话，默认的链接下划线就会消失。我们希望在键盘光标聚焦于该链接上的时候，能有同样的效果，因此我们将 `:hover` 规则复制到 `:focus` 一起。如果某个链接已经被访问过，它会变成灰色。最后，如果某个链接处于激活状态，它就会被加粗，作为某些事情将要发生的一个特别提示。

```
a:link{

 color: blue;

}

a:visited{

 color: gray;

}

a:hover, a:focus{

 color: black;
 text-decoration: none;

}

a:active{

 font-weight: bold;
}
```

```
a:visited{

 color: gray;

}

a:hover a:focus{

 text-decoration: none;

}

a:active{

 font-weight: bold;

}
```

如果你没有按照以上所示的相同顺序来制定规则，那你可要注意了。因为它们可能不会按照你的期望来工作。这是因为特属性会造成样式表中后面的规则覆盖之前的规则。你可以在下一篇文档中学到更多关于特属性的内容。

伪类 `:focus` 同样是表单中的有用工具。比如，你可以用下面的规则将光标闪烁的输入区变成高亮：

```
input:focus {

 border: 2px solid black;
 background-color: lightgray;

}
```

下面，我们来看看 `:first-child`——该伪类会选择父对象的第一子元素实例。比如，下

面的规则选择了所有列表中的第一列表项（带有项目符号或编号），并将其文本变成粗体。

```
li:first-child {

 font-weight: bold;

}
```

最后，我简要提一下伪类 `:lang`，它会选择那些由 `lang` 属性将其语言设置为特定语言的元素。

```
<p lang="en-US">A paragraph of American text, gee whiz!<p>
```

可以用下面的规则来进行选择：

```
p:lang(en-US) {

 ...

}
```

## 伪元素

伪元素有两个用途：首先，你可以使用它们来选择既定元素内文本的首字母或首行，从而方便地给你文档中的每个段落创建一个首字下沉效果，你可以使用如下规则：

```
p:first-letter {

 font-weight: bold;
 font-size: 300%;
 background-color: red;

}
```

每个段落的首字母都会被加粗，并比于段落的其它部分大 300%，而且带有红色背景。

要将每个段落的首行变成粗体，你可以使用如下规则：

```
.....
p:first-line {

 font-weight: bold;

}
.....
```

伪元素的第二个用途是通过 **CSS** 生成页面内容，这更加复杂。你可以使用伪元素 `:before` 或 `:after`，来指定应该在所选元素的前面还是后面插入内容。然后你就可以指定想要插入的内容是什么。比如，你可以使用如下规则在页面的每个链接后面插入一个装饰图片：

```
.....
a:after{

 content: " " url(flower.gif);

}
.....
```

你还可以使用 `attr()` 功能在元素后面插入其属性值。比如，你可以使用以下规则，在文档里每个链接后面的括号中插入它们的链接目标：

```
.....
a:after{

 content: "(" attr(href) ")";

}
.....
```

这样的规则对于打印样式表来说是非常棒的，打印样式表也是你可以编写的样式表，并会在用户打印某个页面时自动应用。这种规则对用户的好处在于，你可以隐藏用户在打印出的资料中无法跟踪的导航，而且通过使用上述的技术，读者就可以看到某个页面上所有引用的 **URL**。

你还可以使用 `counter()` 功能，将递增数值插入到重复元素后面（如项目符号或段落）——在 [dev.opera.com](http://dev.opera.com) 关于 **CSS counters** 的文章中有更加详细的说明。

**IE6** 及更早版本的浏览器不支持这些选择器。同样须注意，你不应当用 **CSS** 来插入重要信息，否则，如果用户选择不用你的样式表的话，这些内容对辅助技术来说将不可用。**CSS** 的黄金规则就是：**CSS** 是用来进行样式化操作的，而 **HTML** 则用于结构化内容。

## CSS 缩写

本教程中你经常会遇到的另外一个东西就是 CSS 缩写。可以将若干个相关的 CSS 属性结合为一个属性，以节省你的时间和精力。在这一部分中我们将学习缩写的可用类型。

其实在本文中我已经使用了缩写，只是没有提及。`border: 2px solid black;` 规则就是缩写的，分别指定了 `border-width: 2px;`, `border-style: solid;` 以及 `border-color: black;`。

### 属性值全写与缩写的对比

请看下面的边距规则（填充距和边框的缩写操作是一样的）：

```
div.foo {
 margin-top: 1em;
 margin-right: 1.5em;
 margin-bottom: 2em;
 margin-left: 2.5em;
}
```

这样的规则也可以写成：

```
div.foo {
 margin: 1em 1.5em 2em 2.5em;
}
```

### 缩写的属性可以不必将四个值全都写出来

根据以下所列内容，一个缩写的属性值可以少于四个。以下内容按照所列出值的数量进行排序：

1. 四个边全都应用相同的值，比如 `margin: 2px;`。
2. 第一个值针对顶部和底部，第二个值针对左侧和右侧，比如 `margin: 2px 5px;`。
3. 第一个和第三个值分别应用到顶端和底部，第二个值应用到左侧和右侧，比如 `margin: 2px 5px 1px;`。
4. 根据 CSS 源顺序，属性值分别应用到顶端、右侧、底部和左侧，如上例所示。

一般说来，为了保证易读性，最明智的办法还是将缩写属性的四个值全部列出来。这个建议也同样适用于 `padding` 的缩写特性。

### 列出所有属性值还是使用缩写

缩写 `margin` 和 `padding` 属性是为了达到最佳的使用效率，但是在某些情况下最好避免使用缩写，或至少慎重考虑，比如以下情况：

- **只需设置一个边距。**在只需要设定一个属性的情况下，同时设定多个属性通常会违反 KISS（保持简单、易懂）原则，该原则在词汇表中有解释。
- **某些属性所使用的选择器从属于多种边界的情况。**如果这种情况发生的话——而它迟早都会发生——在修订或更改你的布局时，你将难以跟踪那一大堆缩写值。
- **你正在编写的样式表将由技术不精通的人来维护。**如果你相信他们会读到这篇文章，那你就不必担心这种情况了，但最好还是不要做出任何假设。
- **你需要换掉一个值来解决某种边界情况。**这往往意味着一个设计不佳的 HTML 文档或样式表……而且这种情况也不是从来没有出现过。

### 其它缩写参考

1. 各种属性的边框缩写：这一点已经在本章的开头说明过了。你甚至可以对一个元素的单侧边框设置边框属性值，其方法如下：

```
2. border-left-width: 2px;
3. border-left-style: solid;

border-left-color: black;
```

4. 同一属性的边距、填充距和边框缩写：它们的缩写方式都相同；如上面的 [属性值全写与缩写的对比](#)部分所示。

5. 字体缩写：你可以使用一行缩写来指定字体大小，粗细，样式，字体体系和行高。比如，请看下面的 CSS：

```
6. font-size: 1.5em;
7. line-height: 200%;
8. font-weight: bold;
9. font-style: italic;

font-family: Georgia, "Times New Roman", serif;
```

你可以用下面这一行代码来对它们进行全部指定：

```
font: 1.5em/200% bold italic Georgia,"Times New Roman",serif;
```

10. 背景缩写：你可以用一行 CSS 代码来指定背景颜色、背景图像、图像重复和图像位置。请看下面的代码：

```
11. background-color: #000;
12. background-image: url(image.gif);
13. background-repeat: no-repeat;

background-position: top left;
```

上述代码这可以用以下缩写方式来表示：

```
background: #000 url(image.gif) no-repeat top left;
```

14. 列表缩写：可以用同样的办法来处理列表属性，你可以用单行代码来设置列表项目符号类型，位置和图像。看看下面的 CSS 代码：

```
15. list-style-type: circle;
16. list-style-position: inside;

list-style-image: url(bullet.gif);
```

这等同于：

```
list-style: circle inside url(bullet.gif);
```

注意 #000 是一个缩写的十六进制色彩值；等同于 #000000 的普通写法，这一点之前我们已经讲过。让我们来看一个更复杂的例子：#6c9 与 #66cc99 是一样的。

## 将 CSS 应用到 HTML

有三种方法可以将 CSS 应用到 HTML 文档：内联样式，嵌入样式和外部样式表。除非你有很好的理由要使用前两者中的其中一种，否则请使用第三种选择。很快你就会知道这是为什么，但我们要先回顾一下这几个选项。

### 内联样式

你可以通过 style 属性将样式应用到一个元素，像这样：

```
<p style="background:blue; color:white; padding:5px;">Paragraph</p>
```

在这个属性内部你列出了所有的 CSS 属性和它们的值（每个属性/值对之间都用分号隔开，每对中的属性与自己的值都用冒号隔开）。通过这种方式你就可以使用 CSS 来定义样式。

[点此查看本示例。](#)

如果你在浏览器中打开这个例子，你会发现应用了该 `style` 属性的段落是蓝色的，而文本则是白色的，并且与其它段落的文本大小不一样，如图 1 所示。

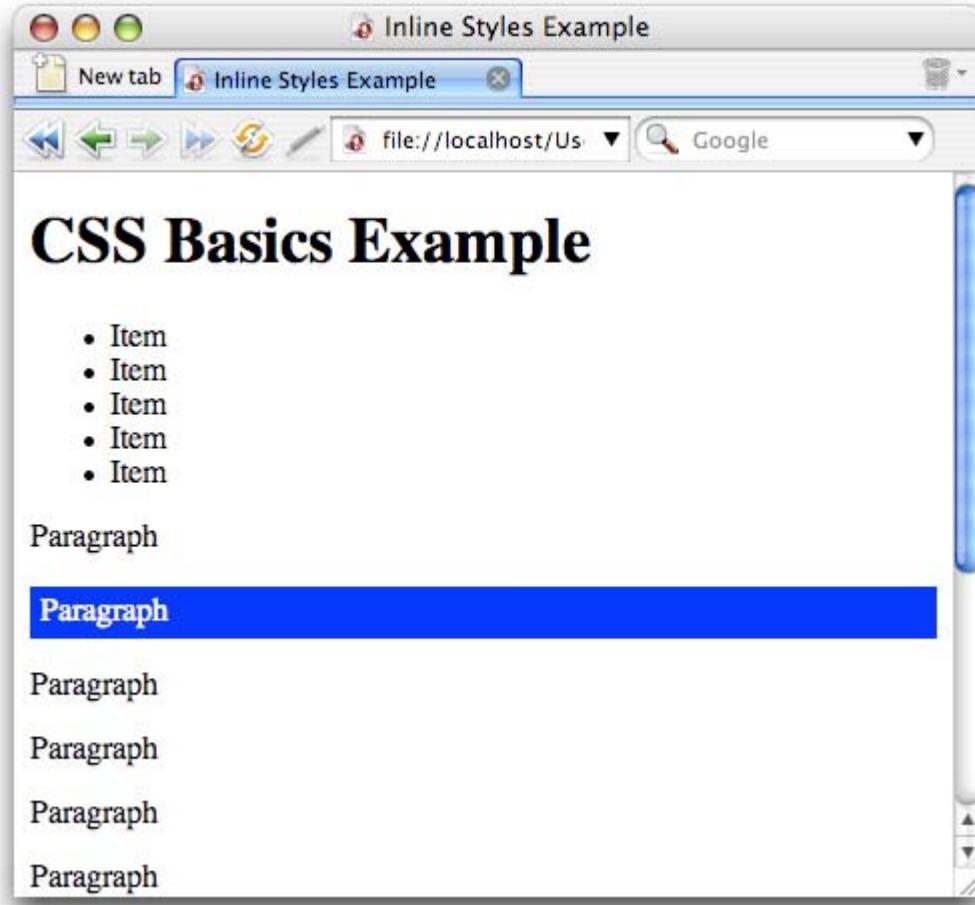


图 1: Opera 中应用了内联样式的段落与其它段落的外观有所不同。

内联样式的优点在于可以强制命令浏览器应用这些设置。在其它样式表中定义的样式，甚至那些在文档的 `head` 部分插入的样式都将被这些样式覆盖。

内联样式最大的问题在于它们会大大增加维护 HTML 文档困难。CSS 主要是将文档的展示方式从其架构中分离出来，但内联样式所做的却是这样——将表达规则分散到整个文档中去。

除了维护的问题之外，你也没有用到 CSS 最强大的功能，那就是：层叠。在下篇文章中我们将对层叠进行详细探讨，不过，现在你只需要知道这一点就可以了：使用层叠意味着你在一个地方定义了某种外观，而浏览器会将其应用到所有符合某个规则的元素。

## 嵌入样式

嵌入样式处于文件 `head` 部分的 `style` 元素中，如下面的例子所示：

```
<style type="text/css" media="screen">
```

```
p {
 color:white;
 background:blue;
 padding:5px;
}

</style>
```

如果你在浏览器中打开上述链接，就会看到该 HTML 文档的所有段落都应用了这里所定义的样式，如图 2 所示。同样地，你可以查看范例页面的源代码，看看 head 里面的 CSS。

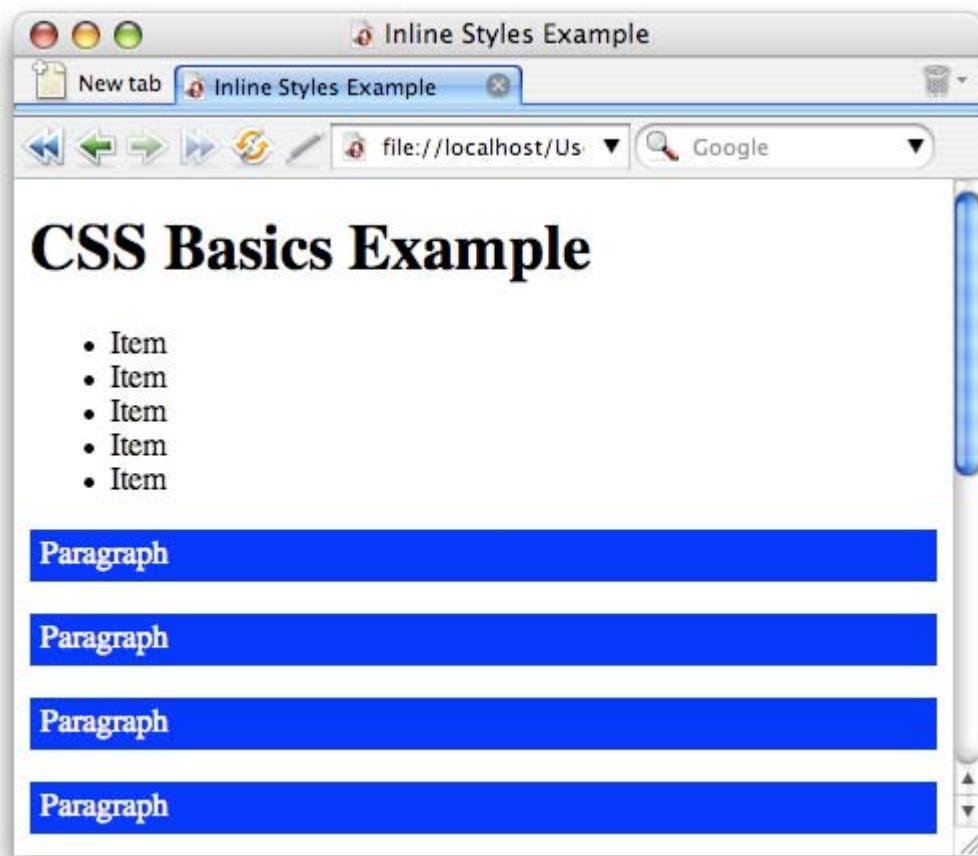


图2：在 Opera 中可以看到通过嵌入样式定义了的所有段落

使用嵌入样式的优点在于你不必在每个段落上都添加一次 style 属性——你可以用一个定义来样式化全部的段落。这也意味着如果你要改变所有段落的外观，就只需要在一个地方进行处理，但是这也只能对一个 HTML 文档有效——如果你想一口气对整个网站所有的段落外观进行定义的话，又该怎么办呢？下面我们就来看看外部样式表。

## 外部样式表

所谓的外部样式表就是将所有 CSS 定义放在一个文件中，然后将其另存为一个扩展名为 CSS 的文件，然后通过在文档的 head 部分中插入 link 元素，来将其应用到你的 HTML 文档中。你可以在我们的 范例页面 中查看源代码，注意在这个文档的 head 部分中包含了一个 link 元素，该元素引用了这个 外部 CSS 文件，你可以检查看看外部 CSS 文件中定义的样式是否被应用到了该 HTML 文档中。下面我们来详细分析一下 link 元素：

```
<link rel="stylesheet" href="styles.css" type="text/css" media="screen">
```

在本教程的前面部分我们已经讲过 link 元素了。现在我们简短地概括一下： href 属性指向 CSS 文件，media 属性定义了这些样式应当被应用到哪些媒体上（本例中是 screen，因为我们不想让打印效果也像这样），而 type 则定义了资源链接是什么（仅仅只有文件扩展名还不足以决定这一点）。

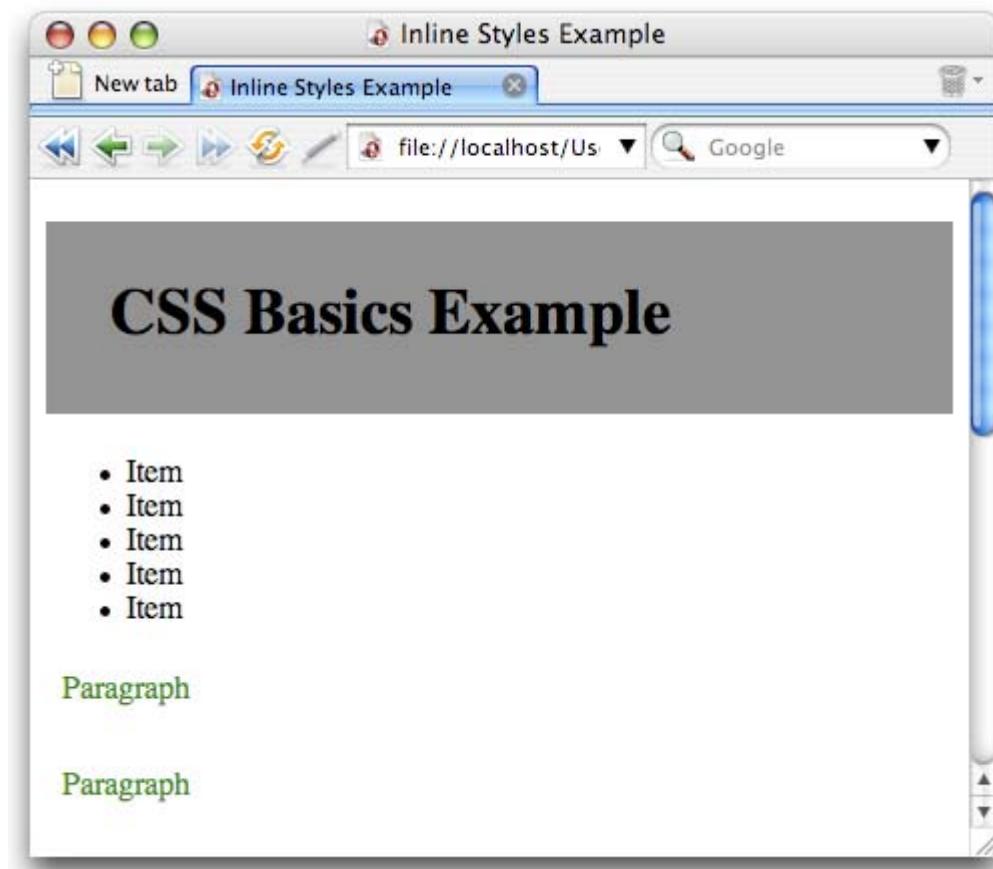


图 3：在 Opera 中可以看到外部样式表所定义的样式，该表是通过 HTML 文档中的 link 元素链接的。

最好的办法是：将你的外观定义保存在一个单独的文件中，这样你就可以通过修改一个文件来修改整个网站，浏览器可以立即加载该文件，然后将其下载到本地缓存中，所有引用了该文件的 HTML 文档都可以访问它，这样就可以减少下载量。

## @importing 样式表

实际上还有另一种将外部式样表导入到 HTML 文件的方法——那就是 `@import` 属性。该属性是插入在一个嵌入样式表中的，跟上面的嵌入式 CSS 一样。其语法如下：

```
<style type="text/css" media="screen">
 @import url("styles.css");

 ...other import statements or CSS styles could go here...

</style>
```

有时你会遇到没有加括号的导入声明，但它的作用是一样的。你还需要意识到另外一件事，就是 `@import` 应当总是位于嵌入样式的最前面。最后，你可以通过在导入声明的结尾写上一些媒体类型，来规定导入的样式表只能应用在某些类型的媒体上（除了 IE6 及更早版本的浏览器，这种办法对其它所有浏览器都能有效）。下面的代码跟上一个代码示例是一回事：

```
<style type="text/css">
 @import url("styles.css") screen;

 ...other import statements or CSS styles could go here...

</style>
```

你可能会问的第一个问题就是“我干嘛非得要用另外一种方式，来将外部样式表应用到我的 HTML 文档中去？”你当然可以不必这样——为了保证完整性，在这里我主要介绍的是关于 `@import` 的信息。与 `link` 元素比起来，使用 `@import` 是有一些小优点/缺点，但它们是非常微小的，因此要选择怎样的方式由你自己决定。`link` 元素是目前公认的用来链接外部样式表的最佳方式。

- 老式浏览器无法识别 `@import`，因此会完全忽略它（如果你省略了文件名前后的括符，Netscape 4 及更老版本的浏览器，还有 IE4 及更老版本的浏览器将无法识别该文件）。因此在有漏洞的老式浏览器中，你可以使用 `@import` 声明来隐藏那些可能无法正确应用的样式。你可以将最新的样式放在一个外部样式表中，并用 `@import` 来导入它们，然后在嵌入

样式表中编写一些不会造成 IE/Netscape4 卡壳的基础样式。这招非常管用，不过目前你很少用得着保证 IE/Netscape4 的兼容性。

- 在前面我们提到过，IE6 不支持在 `@import` 的尾部加入 `media` 类型，因此如果你想针对不同媒体插入多个样式表，这条路就行不通了。
- 你可能会争辩说多个 `@import` 声明的代码要比多个 `link` 元素的代码少，但这一点实在是可以忽略不计的。

## 总结

在这篇教程里，你了解了怎样将 CSS 应用到 HTML 文档中，既可以通过 `style` 属性作为内联样式加载，也可以通过在文件的 `head` 部分的 `style` 元素中作为嵌入样式加载，还可以通过一个单独的文件作为外部样式而加载。你还学到了用最后那种方式——通过 `link` 元素来链接外部样式表——来使维护和缓存的效率更高。我们还讨论了 CSS 的基础语法并对注释、各种选择器类型，以及选择器分组进行了讲解。

在下一篇教程中我们将进一步深入到 CSS 的细节中，详细地探讨层叠和选择器的特异性。

## 练习题

- 内联样式的优点和问题有哪些？你是怎样将它们应用到元素中的？
  - 什么是样式规则？描述一下各种组成部分和语法。
  - 如果你有一串样式规则，该怎样将它们放入外部样式表？
  - 下面的 CSS 选择器与什么相匹配：`ul#nav{}`？
  - 对选择器进行分组的好处是什么？
  - 你怎样对一份打印样式表进行定义？
- 
- 上一篇：可访问性测试
  - 下一篇：继承和层叠
  - 目录

## 作者简介



Chris Heilmann 从事 Web 开发工作已达 10 年之久，之前他曾从事电台新闻工作。他在英国做过 Yahoo! 的培训师和高级开发工程师，同时负责监控面向欧洲和亚洲的前端代码质量。

Chris 的博客是 [Wait till I come](#)，在许多社会网络上你都可以通过输入“`codepo8`”找到他。

## 28. 继承和层叠

Posted 12/15/2008 - 16:55 by Lewis

- 网络标准教程

作者: Tommy Olsson · 2008 年 9 月 26 日

- 上一篇: CSS 基础知识
- 下一篇: 使用 CSS 设置文本样式
- 目录

### 序言

继承和层叠是 CSS 最基本的两个概念。每个使用 CSS 的人都必须理解它们。幸运的是，这两个概念并不难掌握，虽然有些细节有点难以记忆。

这两个概念是紧密相关但又各不相同的。继承关系到 HTML 标记中的元素如何从其父（包含）元素继承属性，并将这些属性传递给它们的子对象，而层叠与应用到某个 HTML 文档的 CSS 声明有关，也与相互冲突的规则会不会互相覆盖有关。

在本章中我们将详细讨论这两个概念。继承可能比较容易掌握一些，因此我将从这一部分开始，然后逐渐进入到错综复杂的层叠概念。点此下载本章范例的源代码；压缩文件中包含有已制作完成的 CSS 和 HTML，以及 HTML 初始模板，让你在学习这些例子的时候可以自己试着操作。

本文内容如下：

- 继承

- 为什么说继承很有用
- 继承是怎样进行的
- 一个关于继承的例子
- 强制继承
- 层叠
  - 重要性
  - 特异性
  - 源顺序
- 总结
- 练习题

## 继承

CSS 中的继承是指某些属性从父元素传递到其子元素的机制。这与基因遗传是非常相似的。如果父母有着蓝色的眼睛，他们的孩子很可能也会有蓝色的眼睛。

不是所有 CSS 属性都是继承的，因为这对某些属性是没有意义的。比如，边距就不是继承的，因为一个子元素不太可能需要和其父对象具有同样的边距。在大多数情况下，仅靠常识就能知道哪些属性是继承的，哪些不是，但为了真正弄清楚，你还是需要在 [CSS 2.1 规范属性一览表](#)里面查看每个属性。

### 为什么说继承很有用

为什么 CSS 会有继承机制呢？想想看如果没有继承会怎样，这样你就能明白这个问题的答案了。你必须逐个指定字体类型、字体大小和文本颜色之类的属性——而且是为每个元素类型单独指定。

利用继承，你就可以指定比如说 `html` 或 `body` 元素的字体属性，而所有其它所有元素就都会继承这些属性。你可以对某个特定的容器元素指定背景和前景颜色，并且前景颜色会自动被容器中的所有子元素继承。背景颜色是不能继承的，但 `background-color` 的初始值是 `transparent`，这表示父对象的背景将会透过子对象。这个效果就好像通过继承获得背景颜色一样，但不妨设想一下如果背景图片也被继承的话将会发生什么！每个子对象都将有和其父对象一样的背景图片，结果看起来就会像一张乱七八糟的拼图，因为每个元素的背景都将会“重来一次”。

### 继承是如何进行的

HTML 文档中的每个元素都将从其父对象那里继承所有的可继承属性，除了根元素 (`html`)，它没有父对象。

继承属性有效与否是取决于其它因素的，这一点将在稍后的层叠部分讲到。就像一个有蓝色眼睛的妈妈也可能会有棕色眼睛的小孩，如果爸爸是棕色眼睛的话，CSS 中的继承属性也可以被覆盖。

### 一个关于继承的例子

- 在你喜欢的文本编辑器里新建一个文件，将下面的 HTML 文档拷进去，另存为 `inherit.html`。

```
2. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">

3. <html lang="en">

4. <head>

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

6. <title>Inheritance</title>

7. </head>

8.

9. <body>

10. <h1>Heading</h1>

11. <p>A paragraph of text.</p>

12. </body>
```

13. </html>

如果你在 web 浏览器中打开该文件，将会看到一个非常枯燥的页面，跟你的浏览器的默认风格一样。

- 在你的文本编辑器中新建一个的空白文件，将下面的 CSS 规则拷进去，将其另存为 `style.css`，保存路径跟 HTML 文件一样。

```
14. html {
15. font: 75% Verdana, sans-serif;
```

16. }

- 在`</head>`标签前面插入下面代码，将样式表链接到你的 HTML 文档：

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

17. 保存修改后的 HTML 文件，你的浏览器中对其重新加载。字体将从浏览器的默认值（通常是 Times 或者 Times New Roman）变成 Verdana。如果你的电脑没有安装 Verdana 字体，则会按照你在浏览器设置中指定的值，以默认的 sans serif 字体来显示文本。此外，文本大小还会缩小，仅有未经样式处理之前的  $3/4$ 。

在这里我们指定的 CSS 规则只会用到 html 元素上。我们没有对标题或段落指定任何的规则，但现在所有的文本都成了 Verdana，大小都变成了默认大小的 75%。这是为什么呢？答案就是继承。

font 属性是一种缩写属性，用来设置所有的字体相关属性。在这里我们只写出了两个字体属性——字体大小和字体类型——但这个规则与下面的代码是一回事：

```
html {
 font-style: normal;
 font-variant: normal;
 font-weight: normal;
 font-size: 75%;
 line-height: normal;
 font-family: Verdana, sans-serif;
}
```

所有这些属性都是继承的，因此 body 元素将从 html 元素中继承它们，然后将它们传递给其子对象——标题和段落。

等一下！在继承字体大小的时候发生了一些可疑的事情，是吧？html 元素的字体大小设置的是 75%，但是是什么的 75% 呢？而且

body 的字体大小本来应该是其父对象字体大小的 75%，而标题和段落的字体大小应该是 body 元素的 75%，对吧？

所继承的值并不是我们所指定的值——即我们在样式表上写的值——而是一种叫作计算值的东西。对字体大小来说，计算值就是一个以像素为单位的绝对值。对于 html 元素，它没有可以继承的父对象，其字体大小就被设成一个与浏览器默认字体大小有关的百分比值。现在大多数的浏览器都有默认的字体大小设置，其值为 16px 的设置，16 的 75% 是 12，因此 html 元素的字体大小计算值可能就是 12px。而这就是被 body 继承的那个值，并且会被传递给标题和段落。

(标题的字体大小会大一些，因为浏览器使用了一些它自己的内置样式。相关内容将会在下面的层叠部分讲到。)

18. 在你的 CSS 样式表中再加两项声明:

```
19. html {
20. font: 75% Verdana, sans-serif;
21. background-color: blue;
22. color: white;
}
}
```

23. 保存 CSS 文件并在浏览器中重新加载该 HTML 文档。

现在背景成了鲜蓝色，所有的文本都成了白色。该规则被应用到了 `html` 元素上——也就是整个 HTML 文档——这样整个页面背景就成了蓝色。白色的前景颜色是由 `body` 元素继承过来的，并传递给 `body` 的所有子对象——在本例中就是标题和段落。它们没有继承背景，但其背景将默认为 `transparent`，因此最终视觉效果就是蓝色背景上的白色文本。

24. 将另一个新规则添加到样式表中，然后保存并重新加载 HTML 文档:

```
25. h1 {
26. font-size: 300%;
}
}
```

这个规则设置了标题的字体大小。所继承的字体大小用的是百分比值——就是浏览器默认值的 75%，我们假设它等于 12px——因此标题大小将是 12px 的 300%，或者说 36px.

### 强制继承

你可以强行规定某种继承——甚至是那些默认情况下不能继承的属性——通过 `inherit` 这个关键字就可以做到。但是你应当谨慎使用该关键字，因为微软的 IE (IE7 及以下版本) 并不支持它。

下面的规则将使所有的段落从它们的父对象那里继承所有的背景属性:

```
p {
 background: inherit;
}
}
```

通过缩写属性，你就可以使用 `inherit`，而不必出所有的属性值。你只能这样：要么对所有对象都使用缩写，要么就不用缩写——你不能用普通写法指定一些值，同时又对其它值使用 `inherit`，因为这些值可以以任何顺序出现，因此无法指定哪个是我们想要继承的值。

你并不需要经常用到强制继承。强制继承对“取消”一个冲突规则中的某个声明，或者避免写死某些值是很用的。比如，看看下面这个典型的导航菜单。

```
<ul id="nav">

Home

News

Products

Services

About Us


```

如果想用一个水平菜单来显示这个链接列表的话，你可以使用下面的 CSS：

```
#nav {
 background: blue;
 color: white;
 margin: 0;
 padding: 0;
}

#nav li {
 display: inline;
 margin: 0;
 padding: 0 0.5em;
 border-right: 1px solid;
}
```

```
#nav li a {
 color: inherit;
 text-decoration: none;
}
```

本例中，我们在 `#nav` 规则中将整个列表的背景颜色设置成蓝色。这段 CSS 代码也将前景颜色设置为白色，而且每个列表项和每个链接都继承了这种设置。在列表项规则中我们设置了右边框，但没有指定边框颜色，这意味着它将会使用继承来的前景颜色（白）。对于链接，我们已经使用了 `color: inherit` 来强制继承和覆盖浏览器的默认链接颜色。

当然我也可以将边框颜色和链接文本颜色设置为白色，但让继承来完成这个工作的好处，在于如果你决定要使用另一个色系，你只需要在一个地方更改颜色就可以了。

## 层叠

CSS 的意思就是层叠样式表，因此如果告诉你层叠是一个很重要的概念，你应该不会感到惊讶。层叠是一种机制，用于在多个互相冲突的 CSS 声明作用于同一个元素时控制最终结果。下面是三个用于控制 CSS 声明的应用顺序的主要概念：

1. 重要性
2. 特异性
3. 源顺序

下面我们来逐个地看看这些概念。

重要性是最……呃……重要的。如果两个声明有着相同的重要性，规则的特异性就会决定该应用哪一个声明。如果两个规则有着相同的特异性，最终结果就会由源顺序决定。

### 重要性

一个 CSS 声明的重要性取决于它是在什么地方被指定的。互相冲突的声明会按照下面的顺序被应用；后面的声明会覆盖先前的声明：

1. 用户代理样式表
2. 作者样式表中的一般声明
3. 用户样式表中的一般声明
4. 作者样式表中的重要声明
5. 用户样式表中的重要声明

**用户代理样式表**是浏览器的内置样式表。每个浏览器都有自己的默认规则，用于在用户或网页设计者没有指定样式的情况下显示各种 **HTML** 元素。比如，未访问链接通常都是蓝色并带有下划线的。

**用户样式表**是由**用户**定义的样式表。并非所有浏览器都支持用户样式表，但它们是非常有用的，尤其是对有某种残疾的用户。比如一个有阅读障碍的人可以用一个用户样式表来指定某些字体和颜色。

在 **Opera** 中，你可以通过 Tools(在 Mac 计算机上则是 **Opera** 菜单) > Preferences… > Advanced tab > Content，点击 Style Options…按钮，然后在对话框中的 Display 里的 My style sheet 文本框中输入你的用户样式表。如果你希望用户样式表覆盖作者样式表，你也可以在 Presentation 标签中进行指定，甚至可以在用户界面中添加一个按钮来让你在用户和作者样式表之间进行切换。为了达到这个目的，在 Preferences… 菜单中全部选择 **OK**,然后在 **Opera** 浏览器界面的某处单击右键或者按住 **Ctrl** 键点击，选择 Customize… > Buttons 标签 > Browser 试图,然后将 Author Mode 按钮拖到你工具栏中的某个地方。

**作者样式表**就是我们通常说的“**样式表**”。这是 **HTML** 文档的作者（或更确切地说是网站设计者）编写的和链接到的（或包含的）**样式表**。

**一般声明**就是一般写法的声明。与一般声明相对的就是**重要声明**，重要声明的后面会写着一个 **!important** 指令。

你可以发现，在用户样式表中的重要声明会其它任何声明的优先级都要高，这是合理的。比如说，如果阅读障碍患者想让所有的文本都显示为 **Comic Sans MS** 字体，因为他觉得这样更易于阅读的话，他可能会使用包含以下规则的用户样式表：

```
* {
 font-family: "Comic Sans MS" !important;
}
```

在这种情况下，不管设计者指定了什么，也不管浏览器的默认字体类型设置如何，一切都会以 **Comic Sans MS** 显示出来。

浏览器的默认渲染只能在其声明没有被任何用户样式表或作者样式表覆盖的时候才能使用，因为用户代理样式表的优先级是最低的。

坦率的讲，大多数设计者不必对重要性考虑得太多，因为我们对其无能为力。如果某个用户定义了一份可以覆盖我们的 **CSS** 的用户样式表，我们也无从知道。无论如何，如果他们要这样做，肯定是有道理的。当然，知道什么是重要性以及它会如何影响我们的 **HTML** 文档的

外观还是很有好处的。

## 特异性

特异性是每个 CSS 设计者必须理解和琢磨的。它可以被看成一个计量单位，用来衡量一个规则选择符的具体性如何。低特异性的选择符可以跟许多元素匹配（像`*`可以匹配文件中所有的元素），而一个高特异性的选择符可能只能匹配一个页面中的某个特定元素（象 `#nav` 只能匹配带有 `nav` 的 `id` 的元素）。

选择符的特异性很容易计算，我们很快就会看到这一点。如果一个既定元素有两个或更多的声明彼此冲突，而所有声明都有着同样的重要性，那么规则中最具体的那个选择符就会“胜出”。

特异性有四个组成部分，我们称之为 **a**、**b**、**c** 和 **d**。**a** 组件的权重最高，“d”则最低。

“a”组件是非常简单的，对于一个在 `style` 属性中的声明来说，它的值是 **1**，否则就是 **0**。

“b”组件的值等于选择符（以 `a #` 开始的）中的 `id` 选择符的数量。

“c”组件的值等于属性选择符——包括类选择符——和伪类的数量。

“d”组件的值等于选择符中元素类型和伪元素的数量。

在经过数算之后，我们就可以将这四个组成部分结合起来，从而得到任何规则的特异性。

在 `style` 属性中的 CSS 声明没有选择符，因此它们的特异性值就是 **1, 0, 0, 0**。

我们来看几个例子——看完之后你就会明白特异性值是如何计算的了。

选择符	a	b	c	d	特异性值
<code>h1</code>	0	0	0	1	0,0,0,1
<code>.foo</code>	0	0	1	0	0,0,1,0
<code>#bar</code>	0	1	0	0	0,1,0,0
<code>html&gt;head+body ul#nav *.home a:link</code>	0	1	2	5	0,1,2,5

让我们更详细地来看看最后一个例子。在这里 **a=0**，因为它是一个选择符，而不是 `style` 属性中的声明。**ID** 选择符有一个 (`#nav`)，因此 **b=1**。还有一个属性选择符 (`.home`) 和一个伪类 (`:link`)，因此 **c=2**。还有 5 个元素类型 (`html`, `head`, `body`, `ul` 和 `a`)，因此 **d=5**。因此最终的特异性值就是 **0, 1, 2, 5**。

值得注意的是连结符（像 `>`, `+` 以及空格）不会影响一个选择符的特异性。通配选择符 (`*`)

也不会加到特异性值里去。

另外要注意的是引用某个 `id` 属性的时候，`id` 选择符与属性选择符的特异性值有着巨大的差异。虽然它们匹配的是同一个元素，它们的特异性值的差异非常大。`#nav` 的特异性值是 `0, 1, 0, 0, 0`，但 `[id="nav"]` 的特异性值仅仅是 `0, 0, 1, 0`。

让我们来看看该算法的实际应用。

1. 我们先在 `HTML` 文档中添加一个段落。

```
2. <body>
3. <h1>Heading</h1>
4.
5. <p>A paragraph of text.</p>
6. <p>A second paragraph of text.</p>
7.
8. </body>
```

7. 在你的样式表中添加一个规则，给段落中的文本设置颜色：

```
8. p {
9. color: cyan;
10. }
```

10. 保存这两个文件，并在浏览器中重新加载该文档，现在应该出现了两段青色的文本。

11. 给第一段设置一个 `id`，这样你可以比较容易地用 `CSS` 选择符选择它。

```
12. <body>
13. <h1>Heading</h1>
14. <p id="special">A paragraph of text.</p>
15. <p>A second paragraph of text.</p>
16.
17.
18. </body>
```

17. 在你的样式表中对这个特殊段落添加一个规则：

```
18. #special {
```

```
19. background-color: red;
20. color: yellow;
```

```
}
```

21. 保存这两个文件，并在浏览器中重新加载该文档，就可以看到艳丽多彩的最终效果。

让我们来看看应用到这两个段落的声明。

你添加的第一个规则对所有段落设置了 `color:cyan`。第二个规则设置了一个红色背景，并对 `id` 为 `special` 的特定元素设置了 `color:yellow`。你的第一个段落同时与两个规则相匹配；因为它是一个段落而且 `id` 为 `special`。

红色背景并不是问题所在，因为它仅仅是为 `#special` 设定的。但两个规则都包含有 `color` 属性的声明，这就意味着会有冲突。这两个规则重要性相同——它们都是作者样式表单中的一般声明——因此你得看看这两个选择符的特异性。

第一个规则的选择符是 `p`，由于它包含了一个特定的元素类型，因此根据上面的算法它的特异性值为 `0, 0, 0, 1`。第二个规则的选择符是 `#special`，由于它包含了一个 `id` 选择符，它的特异性值就是 `0,1,0,0`。`0,1,0,0` 比 `0,0,0,1` 更具体，因此 `color:yellow` 声明胜出，你会得到红色背景黄色文本的最终效果。

## 源顺序

如果两个声明作用于同一个元素，并有着同样的重要性和同样的特异性，那么最终的辨识标记就是源顺序。在样式表后面的声明将“胜过”前面的声明。

如果你有一个单独的外部样式表，如果存在冲突的话，文件末尾的声明将覆盖前面的那些声明。互相冲突的声明也可以来自于不同的样式表。这种情况下，HTML 文档中样式表的链接，植入或导入顺序将决定哪个声明会得到应用，因此如果在 HTML 文档的 `head` 部分中链接了两个样式表，后一个链接的样式表会覆盖前一个链接的样式表。让我们在实际例子中来看看这个法则稍后如何运作的：

1. 在你的样式表文件的最末尾处添加一条新规则，如下所示：

```
2. p {
3. background-color: yellow;
4. color: black;
```

```
}
```

5. 保存并重新加载网页。现在你有两个匹配所有段落的规则。它们有同样的重要性和同样的特异性（因为选择符是一样的），因此源顺序将会是用来消除歧义的最终机制。

最后的规则指定了 `color:black`，这个设置将覆盖先前的规则所指定的 `color:cyan`。

注意第一个段落并没有受到这个新规则的影响。尽管这个新规则出现在最后，但它的选择符的特异性值比 `#special` 要低。这清楚地说明了特异性值的优先级比源顺序要高。

## 总结

继承是层叠是每个 web 设计师必需理解的基础概念。

通过继承，我们可以声明高层元素的属性，并让这些属性遗传给所有派生元素。在默认情况下只有一部分属性才能继承，但可以通过 `inherit` 关键字来强制继承那些默认不能继承的属性。

在多个声明作用于同一个元素时，层叠解决了所有的冲突问题。重要声明会覆盖重要性较低的声明。在重要性相同的声明之间，规则的特异性值决定了哪个声明会得到应用。此外，若其它所有条件都相同，最终结果将取决于源顺序。

## 练习题

- `width` 属性是可以继承的吗？先想想——这样的继承有意义吗？—— 然后再去 [CSS 规范](#) 中寻找正确答案。
- 如果我们在 [示例样式表](#) 最后的规则中，对 `color:black` 声明添加 `!important`，这对第一个“`special`”段落的文本颜色有影响吗？
- 哪个选择符更具体，“`#special`”还是“`html>head+body>h1+p`”？
- 应该怎样定义一份用户样式表，才能让我们的测试文档变成黑色的 `Comic Sans MS` 字体，而背景变成白色的，无论作者样式表是怎样的？
  
- 上一篇：[CSS 基础知识](#)
- 下一篇：[使用 CSS 设置文本样式](#)
- 目录

## 作者简介



Tommy Olsson 是一位 web 标准和网页可访问性的实用主义倡导人，他住在瑞典中部。

Tommy Olsson 在 1993 年就写出了自己的第一份 **HTML** 文档，现在他在一家瑞典政府机构任专业 web 站点管理员。

迄今为止他已经写了一本书——CSS 终极参考大全（与 Paul O'Brien 一起完成的）——还有一个令人遗憾地被忽视了的博客 [The Autistic Cuckoo](#)。

## 29. 使用 CSS 设置文本样式

Posted 12/15/2008 - 16:56 by Lewis

- 网络标准教程

作者: Ben Henick · 2008 年 10 月 3 日

- 上一篇: 继承和层叠
- 下一篇: CSS 布局模型——boxes、borderes、margins、padding
- 目录

### 序言

由于 Web 是 HTML 文档的集合体——有些是动态的，有些是静态的，有些是功能性的——它们的排版所遵循的惯例来自于我们的最佳参考对象：六个世纪的印刷传统。这种传统包括了排印。但是网络是一个全新而且截然不同的媒体，对于网站的排版来说，我们需要对印刷设计具备一种截然不同的技能组合，而且其受到的限制也更多。

本文建立在本系列教程前面那些课程的知识基础上，为你提供了一个用CSS来对文本进行有效地样式化的详细指南。下面链接有许多范例，这些范例演示了我们所讨论的技术——你可以[点此下载第 29 篇文章的所有示例](#)。

本文结构如下：

- Web 排版回顾
  - 对比度
  - 可辨认性和易读性
- CSS 字体属性：改变你的文本外观

- **font-size** 和单位类型选择
  - 这是怎样做到的？
  - CSS 文本属性中可以用哪些类型的单位？
  - 这么多不同的单位类型的用途是什么？
  - 桌面像素等同于实际中的什么？
  - CSS 中的字体高，百分比和点数
  - 官方 CSS 2.1 推荐的简要说明
  - 大小关键字
- **font-family** 和字样选择
- **font-style**、**font-variant** 和 **font-weight**: 改变细节
  - 为什么在 **em** 和 **i** 元素足以胜任的情况下还要使用 **font-style** 属性？
    - 实现短文突出显示的另一种工具：**font-variant**
    - **font-weight**: 加粗和不加粗
- 字体缩写属性
- **CSS** 文本和对齐属性：排字修改
  - 文本对齐和两端对齐
    - 对用英文字母编写的文字说明进行适当调整
  - 改变字距：**letter-spacing** 和 **word-spacing** 属性
    - 用 **em** 单位来实现更好的控制
  - 首行缩进：**text-indent** 属性
  - 大写：**text-transform** 属性
  - 链接样式化与缺失显示：**text-decoration** 属性
    - 带 **acronym** 和 **abbr** 的边框，而不是下划线
  - 行间距调整与行高
  - 替换 **pre** 和 **br: white-space** 属性
- 总结
- 延伸阅读
- 练习题

## Web 排版回顾

在网络上，设计者对页面外观的控制比在其它媒体上要少得多。这种情况在涉及到文件的

大小，分辨率和对比度之类的画布属性的时候显得尤为明显。Web排版的质量还会受到一些严格的限制，这一点在关于排版的基础原则的文章中已经讨论过了。

其它值得一提的概念还有**对比度**，**可辨认性**和**易读性**——下面我们就来看看这些概念。

## 对比度

通过类型**对比度**，我们就可以很容易地将文章从空白和相邻文章中区别出来，类型对比度受到许多因素的影响，比如亮度、颜色、大小和组合。之所以在这里提到对比度，是为了指出低对比度的文本应该设置成所允许的最大尺寸。

## 可辨认性和易读性

在设计情景中，通过**可辨认性**，我们可以轻而易举地对一个文本段落进行特定信息片断**扫描**，而通过**易读性**，则使得我们能够很容易地**理解**一篇文章。表 1 列出了提高这两者之中某一性质的质量的设计方法。

表 1：可辨认性和可读性特征

对象	文本行长度	中缝(文本列之间的间距) & 行间距	字体类型选择	对齐
易读性	中等	增大	<code>serif</code>	右侧不齐 [左对齐]
可辨认性	短	一般	<code>sans-serif</code>	不固定，通常是两端对齐

在下一篇文章——CSS 布局模式中我们将讨论最佳列宽。

## CSS 字体属性：改变你的文本外观

排版涉及到针对文本的操作，这种操作与组合以及单个字母和词语的外观有关。后面这一类任务是由 CSS 字体属性处理的，我们将在下面进行讨论。

### 字体大小和单位类型选择

由于字号的变化比起字体的变化来说，更能影响页面外观，而用户代理样式表通常都能很好地处理字体的变形，因此对于文本来说，最基本的属性就是 `font-size`。在一条规则中应用该属性的时候，它后面会跟着一个属性值，这个值用某种单位指定了字体的大小，也有时是用关键字来表示大小的（比如 `small` 或 `medium`）。

### 这是怎样做到的

作为样式表中最重要的声明，`font-size` 声明的格式是像这样的：

```
body {...
```

```
font-size: 14px;
...}
```

由于继承的缘故，HTML 文档中所有的字号规格都是基于 body 部分的字号声明的，不管该声明是在浏览器的样式表里面还是你的样式表里面。

浏览器默认的 16 像素对你的正文字体大小来说是个不错的起点，但大多数访问者都可以轻松阅读更小的字体。因此，许多设计者选择了更小的规格——大约 11-14 像素。

如果我们用相对值来指定字体大小，或者用关键字来为一个元素指定字体大小，而这个元素的父对象的字体大小不是关键字的话，字号设置就会通过继承来实现。

### CSS 文本属性中可以用哪些类型的单位？

在样式表的规则里，最常见的文本字体大小的单位是像素（px），字体高（em，将在下面解释），百分比（%），以及点数（pt）。表 2 中说明了使用这些单位来调整文本大小的方式。

表 2：方便设置文本大小的 CSS 单位

单位	定义 1	使用
CSS 字体高	$\Delta = x$	1.333em
关键字	UA 定义 2	large, &c.
百分比	$\Delta = x \div 100$	133.3%
像素	绝对单位	16px
点数	绝对单位	12pt

1  $\Delta$  是字号对继承值的变化率。

2 只能继承最近的那个非关键字的字体大小值。

其它的可用单位类型还有大小关键字，Picas1pica 正好等于 12 个点数），以及 exes（ex）。CSS2 支持的其它许多单位类型也是可用的，但在进行文本处理时很少用到。

### 这么多不同的单位类型的用途是什么？

正如表 2 中所指出的那样，存在着相对和绝对大小单位。关键字则两个性质都有，绝对大

小是相对于其它文本而言的，而相对大小则与它们所继承的非关键字值有关。它们的最优使用方法如下：

- **绝对大小** (px, 或者像 pt 之类的标准化单位) 最好应用在不会随文档画布属性改变的布局中——也就是所谓的“固定”、“静态”或 “**Ice**” 布局。
- **相对大小** (em、%) 应该用在非静态的布局中，以及需要在网站可用性和设计者对布局的控制之间做出妥协的情况下。
- **关键字大小** (将在下面进行解释) 应该用在可用性比其它所有设计考虑都重要的情况下。

## 绝对大小和可用性

较早版本的 IE 不允许访问者对那些已经被设置为绝对大小的文本进行大小调整，而那些允许用户进行这种操作的浏览器，它们的文本大小调整界面又很难找到 (Opera 和 Firebox 的用户只要分别使用 Shift + Ctrl/Cmd + plus/minus 和 Ctrl/Cmd + plus/minus 就能轻易调出该界面)。因为这些限制，将 body 的 font-size 设置成相对值就成了大家的习惯做法，通常我们用 em 单位来设置相对值，因为 em 单位是基于浏览器默认值来定义的。

## 桌面像素在实际中等同于什么？

对这个问题最准确的答案就是没有这样的东西。像素是显示器分辨率的一个单位，仅此而已，但是……

尽管从字面上定义或预计一个像素的大小是完全不可能的，在发现他们的网站设计在客户端变成了另一种外观，与他们自己的设置截然不同的时候，商业项目主办方会非常震惊，并且会因此向网站设计者发火。为了避免这种情况，了解一下像素是怎样运作的会很有帮助——如果你为之创建网站的主办方抱怨说文本在不同的机子上的外观没有完全一样的话，这些知识就可以作为你争辩的论据。

软件发行商有一个不成文的规定，那就是 **96ppi** (像素/每英寸) 是一个合理的数量。因此 16 像素的正文将按 1/6 英寸或 12 个点数来打印。在日益普及的 17" 1280x800 液晶显示器上，这样的 16 像素的文本在屏幕上的大小将接近 13 个点数，但在类似的 15 寸笔记本显示器上，其大小将是 11.44 个点数。

这些数量对默认设置是有效的。大多数环境下，终端用户都可以设置一个自定义 ppi 数量，因此会造成一些边缘情况。

综上所述：1 像素就是 1 像素，但其它一切都是变化的。

## CSS 中的字体高, 百分比和点数

传统上, `em` 等于其所应用到的字体中的大写字母“M”的高度, 但是, 在 CSS 中, `em` 单位实际是等于一行的总高度的; 换言之, 对于一个 `font-size` 为 14 像素的元素来说:

`1em = 100% = 14px`

在典型环境下, 上面的命题可以扩展为:

`1em = 100% = 14px = 10.5pt`

大小的增减是用乘法操作的, 因此如果你想将第二元素的大小设置为 16 像素, 在正常继承的情况下, 下面所有的值都能得到理想的结果:

`1.143em = 114.3% ≈ 16px = 12pt`

### 官方 CSS2.1 推荐的简要说明

有时你会发现自己得考虑到万维网联盟的 CSS 2.1 规范的候选推荐标准。像 W3C 的其它推荐一样, 这份文件可以被看作是一部 Web 标准的明确描述; 其中有些部分比起其它部分来说, 在浏览器发行商和 web 开发员中得到了更加切实的遵守。

尽管在广度和深度上具备关于 W3C 推荐的知识是很有好处的, 但本教程的目的是为你提供 web 开发/设计的简明而易于消化的导论, 而 W3C 推荐至少可以说有点冗长, 因此就不赘述了。所有将你引向 CSS2.1 推荐的案例所指向的资料都很不容易理解, 因此就不能在本文中进行精确解释了。

### 大小关键字

就如在上面所提到的, 你也可以使用大小关键字。从 `xx-small` 直到 `xx-large`, 还有作为中间值(也是默认值)的 `medium`, 共有 7 个关键字。所有的 7 个值都列在下面的表 3 中, 该表包括了本文所讨论的各种 CSS 属性能支持的关键字。

CSS2.1 推荐提供了大量关于 `font-size` 关键字的补充细节。

### 示例 1

这篇教材将不时地链接到渐进的样式处理示例文件, 该文件的作用是在实际运用中展示我们所讨论的 CSS 属性。第一个示例展示了完全没有经过样式处理的 HTML 示例文档。

链接:

- 未经样式处理过的演示文档
- 大小调整后的标题, 出处和正文
- 示例 1 的样式表

## 新规则

```
body { font-size: 14px; }

h1 { font-size: x-large; }

.sectionNote { font-size: medium; }

.attribution { font-size: small; }
```

### font-family 和字样选择

在文本大小令你满意之后，你就可以开始进行字样选择。这是通过 `font-family` 属性来完成的，如下面的示例 2 所示。

在为 `font-family` 属性设定属性值的时候，你应当遵循如下规则：

1. 字样的名称必须完全与客户端字体库中的名称一致，其命名的参照标准是不变字体。
2. 所有命名后的字样必须用逗号分开，逗号后面可以跟空格也可以不跟。
3. 如果字样的名称包含了一个以上的词，必须用单引号或双引号框起来。**比如**: 'Times New Roman'.
4. 字样的命名应当根据可获得性之类的标准进行**升序排列**。比如，如果你希望 Macintosh 用户看到字体设置为 `Palatino` 的一个网页，你的属性值声明应该按照如下方式排列：  
`font-family: Palatino, Georgia, serif;` `Palatino` 正是你想要的字体，但有些用户可能没有这种字体；`Georgia` 的可获得性要大得多，而且又与 `Palatino` 相似；`Serif` 指的是普通的默认 `serif` 字体。如果 `Palatino` 和 `Georgia` 都不可用，系统将回到其默认的 `serif` 字体。

5. 作为一种故障应对机制，`font-family` 的值应该总是以适当的常用名来结尾，如上所述。图 1 展示了 MacOS 10.5 下普通字型体系中的字样。

<i>Excitably</i>	<code>cursive:</code> Apple Chancery
Excitably	<code>fantasy:</code> Papyrus
Excitably	<code>monospace:</code> Monaco
Excitably	<code>sans-serif:</code> Helvetica
Excitably	<code>serif:</code> Times New Roman

图 1：MacOS 10.5 下的默认“普通”字样，在 Safari3.1 中渲染为 24px

CSS2.1 推荐罗列出了几个可以应用到每个字型体系的字样。

## 示例 2

既然文本大小是可以预见的，我们就应该优化所使用的字样。为了保证可辨认性，标题最好设置为 **sans serif** 字样，叙述部分则使用 **serif** 字样。

链接：

- 新字体
- [示例 2 的样式表](#)

### 新规则

```
body { font-family: Palatino,'Palatino Linotype',Georgia,
 sans-serif; }

h1 { font-family: Helvetica,Arial,sans-serif; }

blockquote { font-family: Helvetica,Arial,sans-serif; }
```

### **font-style, font-variant, 和 font-weight: 改变细节**

**font-style** 属性可以在不用 **i** 元素的情况下设置斜体字。它的三个有效值是 **italic**, **oblique** 以及 **normal**。

**italic** 和 **oblique** 属性值在市场上畅销的所有最新版本的浏览器中有着同样的效果，虽然这两种样式之间有着重要的技术差别，如本教程附带的 [词汇表](#) 中所示。

### **为什么在 em 和 i 元素足以胜任的情况下还要使用 font-style 属性？**

这些元素每个都有着特定用途：**em** 是进行强调，而在这几个适合使用 **i** 元素的范例中，它是作为 `<span style="font-style: italic;">...</span>` 的替换对象。一般 `<i>` 是完全不适合使用的，因为它是一个表象元素，但也有些文本片段按照惯例是要设为斜体的，比如像书的标题（虽然这还存在争议；有些人认为对于书标题来说 **cite** 元素更适合。你可以采用看起来最适合的形式）。**<em>** 一般会更加适合使用，因为它将强调规定为一个概念，而不是将斜体字规定为一个特定的样式——强调的实际外观可能会由于浏览器的不同而有所差异。

在有些情况下，**em** 和其近亲 **strong** 需要采用不同的使用方法。比如，假设你想要通过放大字体来强调某些文字。加粗强调的一贯步骤会在后面添加斜体，如下面的规则所示：

```
em {

 font-size: large;

 font-style: normal;

}
```

```
strong {
 font-size: large;
 font-weight: normal;
 font-style: italic;
}
```

上面的样式表规则将产生类似于下面的结果：

The **quick** red fox jumped over the **lazy** brown dog.

### 示例 3

在本示例的正文中并没有斜体字或斜体段落，由于使用了 `cite` 元素，本示例的出处中已经包含了必要的斜体设置。如果缺少应用对象，标题将是使用斜体的最佳候选对象。

链接：

- 将标题设置为斜体
- 示例 3 的样式表

### 新规则

```
h1 { font-style: italic; }
.sectionNote { font-style: normal; }
```

### 实现短文突出显示的另一种工具：font-variant

`font-variant` 属性有两个合法值，`small-caps` 和 `normal`。一些设计者用“Small caps”（也叫做“铜板印刷”字体）来突出显示一段文字的开头的词组，或者对引用的标记进行强调，比如下面内容：

ABANDON ALL HOPE, YE WHO ENTER HERE.

### 示例 4

链接：

- 将开头的词组设为 `small caps`
- 示例 4 的样式表

**新规则:**

```
.opening { font-variant: small-caps; }
```

## **font-weight: 加粗与不加粗**

`font-weight` 属性让你能改变应用了该属性的任何文本段落的加粗级别。

`font-weight` 属性支持的属性值中最常见的是 `normal` 和 `bold`, 而 CSS2.1 推荐提供了一些其它的属性值, 网站排版的现行支持状态使得这里的其它属性值对除了专业用户之外的所有用户毫无功能上的意义。

### **示例 5**

对作者姓名使用粗体字样是杂志期刊经常用到的设计技术, 这也同样适用于网站上的各种情况。

**链接:**

- 对作者姓名使用粗体字样
- [示例 5 的样式表](#)

**新规则:**

```
.author { font-weight: bold; }
```

## **字体缩写属性**

如果有必要的话, 可以按照与背景属性缩写相似的方式, 在一个属性里面设置多个文本属性值。

字体的缩写规则如下:

```
h1 {
font: italic normal bold x-large/1.167em Helvetica,Arial,sans-serif;
}
```

为了达到最佳效果, 你为这个属性设置的值应该包括你想对下面的所有属性设置的值, 设置的时候也要按照下面的顺序, 属性值之间用空格符分开:

1. `font-style`
2. `font-variant`
3. `font-weight`

4. `font-size`, 如果必要的话, 后面可以跟一个斜线号“/”, 以及 `line-height` 值 (如下所示)
5. `font-family`, 在本例中该属性值也可以是一个指定了某个系统字体保留字; 多个值之间应当用逗号而不是空格来隔开。

**表 3:** 本文中所讨论的属性可支持的关键字值

属性	值
<code>font-family</code>	<code>cursive</code> , <code>fantasy</code> , <code>monospace</code> , <code>sans-serif</code> , <code>serif</code>
<code>font-size</code>	<code>xx-small</code> , <code>x-small</code> , <code>small</code> , <code>medium</code> , <code>large</code> , <code>x-large</code> , <code>xx-large</code>
<code>font-style</code>	<code>italic</code> , <code>normal</code> , <code>oblique</code>
<code>font-variant</code>	<code>normal</code> , <code>small-caps</code>
<code>font-weight</code>	<code>bold</code> , <code>normal</code>
<code>line-height</code>	<code>normal</code>
<code>text-align</code>	<code>center</code> , <code>justify</code> , <code>left</code> , <code>right</code>
<code>text-decoration</code>	<code>line-through</code> , <code>none</code> , <code>overline</code> , <code>underline</code>
<code>text-transform</code>	<code>capitalize</code> , <code>lowercase</code> , <code>none</code> , <code>uppercase</code>
<code>white-space</code>	<code>normal</code> , <code>nowrap</code> , <code>pre</code> , <code>pre-line</code> , <code>pre-wrap</code>

### CSS 文本和对齐属性: 排字修改

一名设计师在处理字母和单词的细节: 线条, 曲线, 圆点, 像素和设计中的其它因素。但是设计是一个整体; 它还有其它更大的组成部分, 这些组成部分通过排字集中起来, 而排字主要是由 CSS 布局模式控制的。除了布局模式之外, CSS 还会运用文本和对齐属性来对排字进行调整。

#### 文本对齐和两端对齐

在文字处理环境中，`text-align` 属性支持四种调整值：`left`、`right`、`center` 和 `justify`。最后一个值可以进行两端对齐：该属性值会对一段文字的可见的文本边距进行调整，使左边距和右边距一致。

### 对用英文字母编写的文字说明进行适当的对齐

各种可用的对齐方式的最佳应用方法如下：

- **左对齐**是最适合长篇的叙述性文章的。
- **右对齐**最适合数据表的最左边一栏以及多栏式布局。当相邻栏向左对齐并位于一条宽度适当的中缝的一侧时，就可以提高两个栏的可辨认性。
- **两端对齐**对于区块引用和小块文本之类的小型区块效果很好。如果用在设置了最佳宽度的记叙文的长段落上，全面调整也能提高你的布局的条理性。
- **居中对齐**通常是用在界面元素和诸如网站页脚中的列表之类的连续列表上。

### 示例 6

既然本示例是来源于一本书中的小说，为什么不对它进行两端对齐呢？

链接：

- 对这篇文章的正文文本应用两端对齐
- 示例 6 的样式表

新规则：

```
p { text-align: justify; }

blockquote p { text-align: left; }
```

### 改变字距：`letter-spacing` 和 `word-spacing` 属性

这些属性是非常好理解的；它们能让你分别改变字母和单词之间的空格数量。

`letter-spacing` 的最常见的用法是进行一些细微的强调，其效果与 `font-variant: small-caps`；相似，该属性也可以用来对界面元素的组合进行细微改变。

`word-spacing` 属性最适合用在人为改变文本行中出现的单词数量。比如说，如果你有一段文字块，该文字块具有典型宽度和非典型字号，你就可以用这种属性。

在打印时，字间距和词间距是用来实现避免组合缺点的特殊用途的，但在 Web 上这项技术的用处非常有限。

除了单位属性值，这两个属性也支持 `normal` 值。

### 使用 `em` 单位来实现更好的控制

当你改变文本字间距的时候，会发生长距离的位移；默认的字间距是在一个 `em` 的  $1/10$  到  $1/20$  之间，因此 `letter-spacing` 值所进行的修改会使字间距大于默认设置两倍或者是默认设置的一半，这很可能会使文本变得难以辨认。

### 示例 7

在末尾的标志文本可以使用一些细微的强调……由于格言引用部分设置了更大的尺寸，因此可以通过使用一些词间距来对其进行改进。

链接:

- 对文章的倒数第二段中的招呼语添加字间距
- 示例 7 的样式表

新规则:

```
q { letter-spacing: .143em; }
.pullQuote { word-spacing: .143em; }
```

### 首行缩进: `text-indent` 属性

`text-indent` 属性使我们能够在印刷品的传统样式中实现段落缩进。同样也有一些高级布局技术是基于本属性实现的，并且该属性支持负属性值。

此属性所支持的值与 `font-size` 所支持的一样，此外它还支持 `normal`。

### 示例 8

根据同样的原理，文本进行了两端对齐处理，也许我们应该对它的所有段落应用缩进：

链接:

- 对正文段落进行首行缩进
- 示例 8 的样式表

新规则:

```
p { text-indent: 1.429em; }
blockquote p { text-indent: 0; }
```

### 大写: `the text-transform` 属性

正如 `font-variant` 属性让你能使用铜版印刷字体, `text-transform` 属性专门对大写进行处理。它的合法值可以对文本进行全部大写、全部小写或者全部首行大写的渲染。关于合法值的列表, 请参见上面的表 3。

## 示例 9

链接:

- 用全部大写来对作者名进行强调
- 示例 9 中的样式表

新规则:

```
.author { text-transform: uppercase; }
```

## 链接样式和缺失显示: `text-decoration` 属性

此属性可以在文本的上方、下方或者中间添加直线。其最常见的用法是用来控制默认的链接下划线, 不过 `wikis`、`satire` 和其它设置也会要求使用删除线。在这些情况下, 你可能会想要使用 `ins` (插入) 和 `del` (删除) 元素, 这些元素可以提供与下面代码同样的样式:

```
ins {
 text-decoration: underline;
}

del {
 text-decoration: line-through;
}
```

就算没有自定义样式表规则, `ins` 和 `del` 也可以将标记样式化, 如下所示:

Benjamin Disraeli is remembered for many witticisms, including “there are lies, damned lies, and statistics.”

## 带 `acronym` 和 `abbr` 的边框, 而非下划线

在一些设计者中很流行变换 `acronym` 和 `abbr` 元素的外观, 这样乍一看它们就像点线状的下划线一样。但是, 这个效果实际上要通过 `border-bottom` 值来实现。(别忘了有些浏览器会自动生成这个效果, 但其它像 IE6 这样的就没有)

## 示例 10

链接:

- 将示例文本中连接到 Gutenberg E-Text 的链接的下划线去掉
- 示例 10 的样式表

新规则:

```
.source a { text-decoration: none; }
```

## 行间距调整和行高

大家都知道, 为了提高易读性我们必须在文本行之间留有一些空白, 因为增加的空白可以保证相邻行的字母的上伸部分和下伸部分 (参见图 2) 不会产生视觉效果上的冲突。

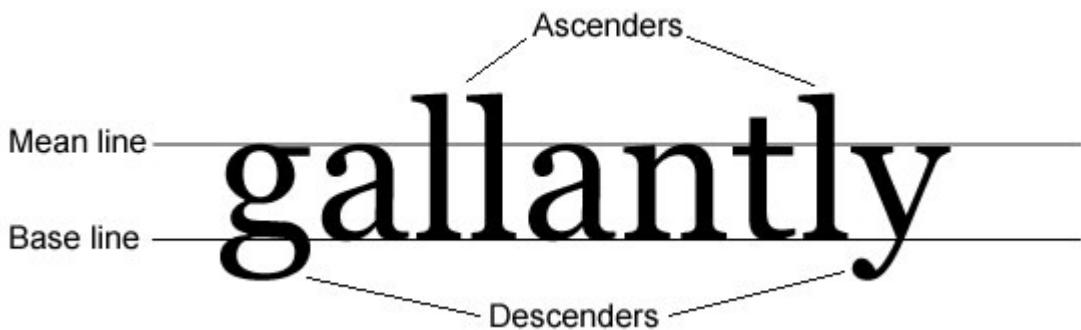


图2: 字母的上伸部分就是超过文本平均线之上的部分; 下伸部分就是越过文本基准线以下的部分。

一个元素的 `font-size` 和其 `line-height` 的关系并不紧密, 但默认情况下, 所有用户代理都会在文本的每一行插入少量的行间距——通常是字母本身高度的 10-15%。因为这个默认值会随字样而改变, 除了数字值之外, `line-height` 属性支持也支持 `normal`。

还需要注意的是, 与大部分 CSS 属性不同, `line-height` 属性可以接受没有单位的数字值, 这样的值将作为默认值的百分比来渲染。

## 示例 11

行间距和易读性是密切相关的, 在本示例中这一点就可以得到证明:

链接:

- 为段落添加适当的行间距
- 示例 11 的样式表

新规则:

```
p { line-height: 1.5; }

.attribution { line-height: 1.5; }
```

## 替换 pre 和 br: white-space 属性

从最严格的意义上来说，强制换行只是一种表象元素，但是在许多情况下它们是一种网站设计的可取元素。与浏览器在任意的地方进行强制换行的习惯一样，单独使用标记来达到我们想要的控制程度可能是很难的。

`pre` 元素可以用于处理这些困难，但是它也有着自己的问题，这就是为什么 CSS 提供 `white-space` 属性的原因。在表 3 中列出的它所支持的值，使设计师可以选择是否让浏览器对已经添加进源标记或作为生成内容插入的空白和换行进行渲染。

CSS2.1 推荐提供了 关于 `white-space` 属性的建议用法的全面细节。

## 总结

一个优秀的网站设计应当高度关注细节以及对许多元素的互动的恰当调节。

目前的浏览器所支持的 CSS 字体和文本属性，对排版提供了几乎是目前输出设备所允许的最高等级的控制，这取决于一个尽责的网站设计师是否学习如何使用它们。若能成功使用这些属性，网站的投入运行就可以激励我们寻求针对排版的质量标准，这些标准与跨越几个世纪的印刷媒体更普遍相关，尽管网络的出现还不到一代人的时间。

## 延伸阅读

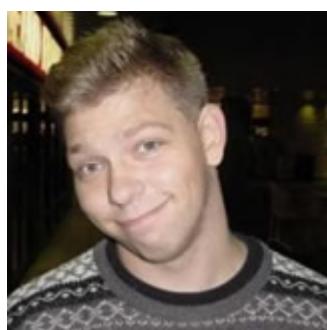
- Bos, Bert, et al. 2007. Cascading style sheets level 2 revision 1 (CSS 2.1) specification.  
World Wide Web Consortium. [http://www.w3.org/TR/2007/CR-CSS21-20070719 etc.](http://www.w3.org/TR/2007/CR-CSS21-20070719) (accessed 28 May 2008).
- Chaparro, Barbara, et al. 2004. Reading online text: a comparison of four white space layouts.  
Wichita State University Software Usability Research Laboratory Usability News. <http://www.surl.org/usabilitynews/62/whitespace.asp> (accessed 28 May 2008).
- Dean, Paul. 2008. Extreme type terminology.  
I Love Typography, the Typography Blog. [http://ilovetypography.com/2008/03/21/extreme-type-terminology/ etc.](http://ilovetypography.com/2008/03/21/extreme-type-terminology/) (accessed 28 May 2008).

- Horton, Sarah, and Lynch, Patrick. 2002. [Web style guide: basic principles for creating web sites](#), 2nd edition. New Haven, Conn.: Yale University Press.
- Roselli, Adrian. 2002. A simple character entity chart. Evolt.org. [http://www.evolt.org/article/A\\_Simple\\_Character\\_Entity\\_Chart/17/21234/](http://www.evolt.org/article/A_Simple_Character_Entity_Chart/17/21234/) (accessed 28 May 2008).

## 练习题

- 列举 `b` 和 `i` 以外的三个可由默认的变形字体渲染的 `html` 元素。简要描述你所列出的元素的用途，并说明怎样将一个变形字体适当地应用到这些元素中。
- 主观测试一下你选择的长篇文本在不同的 `line-height` 值下的可读性。简要总结你的结果。
- 将 `text-transform: uppercase;` 应用到前面习题中的文本的某个段落；重复你的可读性主观测试。简要总结你的结果。
- 参考本文中的 [排版回顾](#)，简要说明反锯齿的优缺点。
- 描述在 `font-family` 值中，应该按照什么顺序来指定字样。
- 在完全没有看到过的设置里面，找出本文描述过的至少三个属性，并列出每个属性的至少一个有效值（除了默认值）。演示或描述在样式表中这些属性/值对的应用效果。
- 创建一个元素，应用到文本里面，并为其设置 `9px` 的 `font-size` 值。在 IE 中打开包含有这个元素的文档，并逐一尝试 `View → Text Size` 菜单中提供的字号。评估一下在一个拥有大量的中老年访问者的网站上，这些字号的效果的适宜性如何。
- 上一篇：继承和层叠
- 下一篇：[CSS 布局模型——boxes、borderes、margins、padding](#)
- 目录

## 关于作者



Ben Henick 从 1995 年 9 月就开始以各种身份参与创建网站了，那时他以一个学术志愿者

的身份进行了他的第一个 Web 项目。从那时开始，他的大多数工作都是在自由职业的基础上完成的。

Ben 是一个多面手。他的技能触及了网站设计和开发的几乎每一个方面，从 CSS 到 HTML，再到设计和文案撰写，再到 PHP/MySQL 和 JavaScript/Ajax。

他生活在堪萨斯州的 Lawrence，有三台电脑，却没有电视机。你可以在 [henick.net](http://henick.net) 上读到更多关于他和他工作的内容。

## 30. CSS 布局模型——boxes、borderes、margins、padding

Posted 12/15/2008 - 16:56 by Lewis

- 网络标准教程

作者: Ben Henick · 2008 年 9 月 26 日

- 上一篇: 使用 CSS 设置文本样式
- 下一篇: CSS 背景图片
- 目录

### 序言

乍看之下, CSS 布局模式简单明了。盒子, 边框和边距都是相当简单的对象, CSS 语法对它们的特性描述并不复杂。

然而, 浏览器渲染引擎遵循着 CSS 2.1 推荐中所规定的一大堆规则, 此外还有一些自己的规则。因此, 对设计师而言, 在将某个高级技巧添加到自己的技术集之前, 需要先了解大量细节问题。

在本文中, 我们将对那些控制 HTML 元素的布局的 CSS 属性进行介绍, 这些 CSS 属性包括 HTML 元素的边框, 边距和许多其它属性。本文所涵盖的范围还包括上面提到的一些规则。高级的栏式布局以及栅网聚焦技术将在后面的文章中讨论, 这些文章将更加详细地对布局, 浮动, 清除, 以及定位进行探讨。本文链接有许多代码示例来演示所讨论的技巧, 但如果你想在本地计算机上研究这些代码, 你可以 [点击此处下载所有代码示例](#)。

本文内容如下:

- 变化的组合: CSS 中的边距, 边框和填充距
  - 在对象周围添加空白: margin-top、margin-right、margin-bottom、margin-left  
以及边距属性

- 自动边距
- 负边距
- 合并边距
- [示例 1](#)
- 给对象添加边框：边框属性
  - [border-width 属性](#)
  - [border-color 属性](#)
  - 详述边框缩写属性和它的 4 个近亲
  - 创建线条：五个...而不是一个边框缩写属性的基本原理
  - ...为什么要有这么多属性？它们只是边框而已，不是吗？
- [示例 2](#)
- 只有边距是不够的：填充距属性
  - [示例 3](#)
- 对元素宽度和高度的操作
  - 宽度和高度基础
  - [min-width、max-width、min-height 以及 max-height](#)
  - [示例 4](#)
  - 溢出：是对内容进行限制，还是给它自由
    - 四个溢出值的效果
- 盒模型：将一切组合起来
  - 为你的布局选择适当的单位
    - 元素大小调整的重要原则：混用比例单位和静态单位时应慎重，或者干脆不要混用
    - 为布局选择适当的单位类型：优缺点
  - 盒模型组件
  - [W3C 的盒模型：一切都是附加的](#)
  - [W3C 盒模型中的百分比边距与填充距](#)
- 对文件流进行处理
  - 元素类型与显示属性
    - [示例 5](#)
  - 使元素环绕其它元素周围：浮动属性
    - [示例 6](#)

- 强制使元素位于前一个浮动元素之下：清除属性
- 总结
- 延伸阅读
- 练习题

### 变化的组合：CSS 中的边距，边框和填充距

在默认情况下，许多 HTML 元素如 `div` 元素和标题会渲染到占据浏览器画布的全部宽度，并且会强制实行末端断行，因此成串的这种元素的渲染效果在文档画布上会处于从上到下的堆叠中。

但是，HTML 元素和其通常的浏览器样式并不能满足开发员想达成的所有用途。把 CSS 与 HTML 合在一起用来“弥合差距”，从而 `class` 和 `id` 给标签添加语义，而样式表规则可以精确地改变布局和页面内容外观——甚至可能通过抵消掉浏览器默认样式的大部分效果来实现这一点。

对空白的细致控制是设计者最重要的工具之一——笔者认为它是重中之重。然而，尽管对空白的控制度能给网站设计带来高产品价值，但在默认的浏览器样式表中却缺乏这种控制，这就意味着设计师会频繁用到本文所说的边距，边框，填充距，和其它 CSS 布局属性。

边距，边框和填充距的分布如图 1 所示。

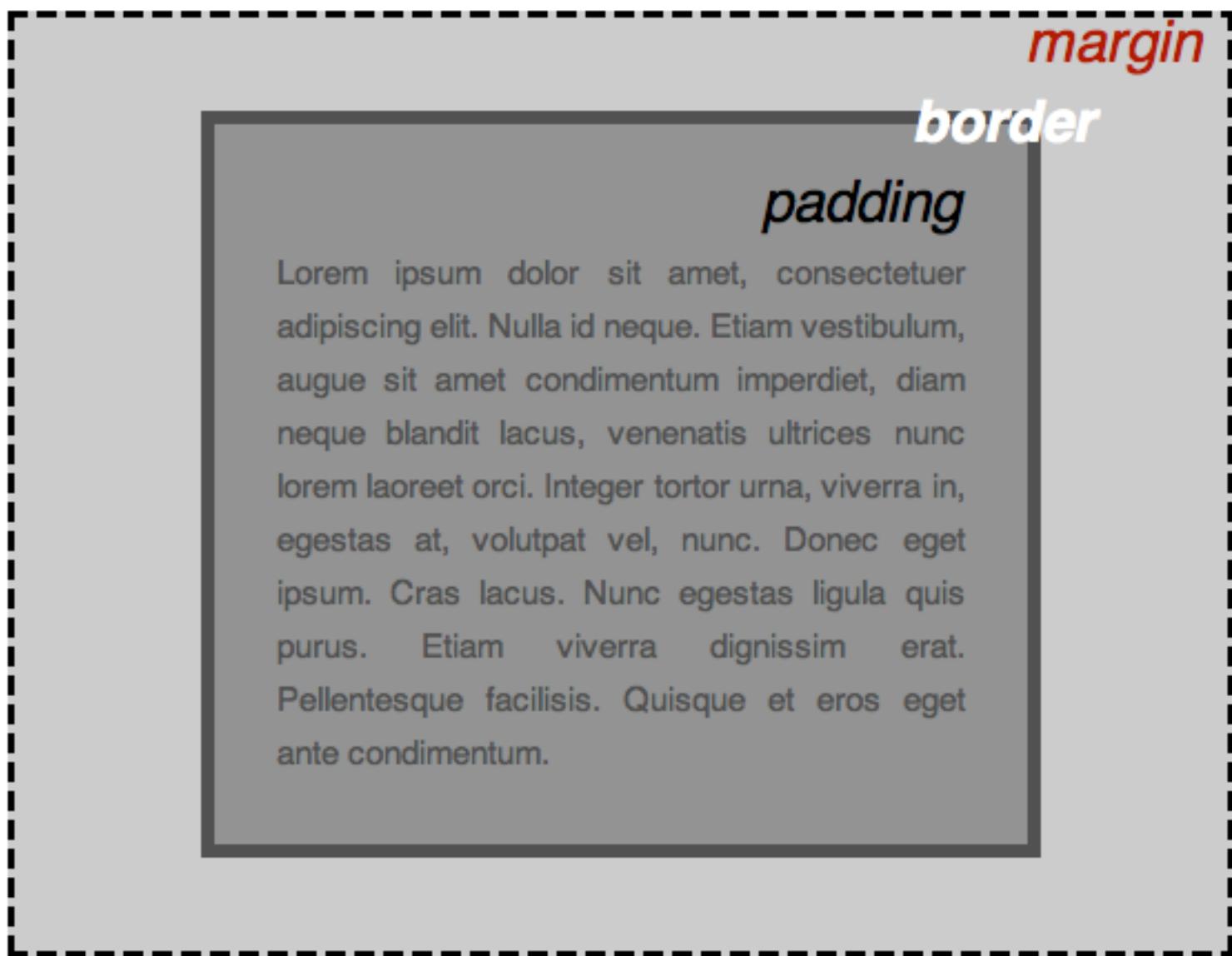


图 1：一个元素盒中各组成成分的详细图示，标有相关 CSS 属性。

#### 在对象周围添加空白：**margin-top**、**margin-right**、**margin-bottom**、**margin-left** 及边距属性

边距可单独指定，也可在一条缩写规则中加以指定。另外缩写规则还可以对某个对象周围的各个边框进行控制。合法的值通常是用 `px` 或 `em` 单位指定的（像素或字体高）。而在 `print-specific` 样式表单中会相反用 `in`、`cm` 或 `pt` 单位（英寸、厘米或点）。

在所有情况下 `%`（百分比）值都是合法的，但须谨慎使用；此类值是按照父元素宽度的比例来计算的，漫不经心的赋值会造成意外的后果。这一问题将会在下面对 CSS 盒模型的讨论中加以详述。

除图像外所有内联元素都没有边距，因而无需边距值。关于这些元素的列表，请参阅下面

的表 2.

## 自动边距

依据不同的情况，`auto` 的赋值会指示浏览器按照自己样式表中的值来渲染边距。但是，在将这样的边距应用到某个元素上，而该元素的宽度具有意义时，`auto` 边距会将所有可用空间渲染成空白。

看看下列规则：

```
.narrowWaisted {
 width: 16.667em;
 margin: 1em auto 1em auto;
}
```

...class `narrowWaisted` 的块元素会将自身对齐到可用画布的中心。

...或可将某个适当元素的右边距设成某个相对较小的值，而左边距设为 `auto` 值。

设置完成后，这样的元素就会接近于右对齐。

## 负边距

所有边距属性都可以设为 **negative**。进行这种设置之后，就能以任意程度“抵消”相邻的边距。如果将一个足够大的负边距应用到一个足够大的元素上，受其影响的相邻元素甚至可以被重叠。

例如，看看下面简单的 `div` 元素（取自示例文件 `negativemargin1.html`。）

```
<div id="header"><h1>Lovely header</h1></div>
<div id="content"><p>Overlapping text is entirely unreadable</p></div>
```

如果用下面的 CSS 来样式化

```
body {background-color:white font-family:Geneva, Arial, Helvetica, sans-serif;}
#header { background-color:yellow }
h1 { color:red; font-size:2em; }
```

就会产生如图 2 所示的效果：

# Lovely header

Overlapping text is entirely unreadable

图 2：这个简单示例中的两个元素。看起来很普通。

有意思的是下面这部分。现在我们要通过下面的规则给位于下边的元素的顶部添加一个相当大的负边距：

```
#content {margin-top:-3em;}
```

这样我们就会看到该元素上移了，重叠在标题上，如图 3 所示（实际例子参见 [negativemargins2.html](#) 示例文件）。

# Lovely header

Overlapping text is entirely unreadable

图 3：应用了负边距之后，下边的元素上移并重叠在标题上。

## 合并边距

两个相似并相邻的块元素都有正边距时，两个边距中只有较大那个才会得以应用。如，试试下面的规则：

```
p {
 margin: 1em auto 1.5em auto;
}
```

如果按照字面含义对含有这种样式规则的文件进行渲染，连续的两个段落之间的最终边距应该是 2.5em，即段落 1(1.5em)的底部边距与段落 2(1em)的顶部边距之和。但是，由于合并边距的应用，它们之间的边距只有 1.5em。

在块元素中列表和标题是特例，它们的边距不会与其它块元素的边距相合并。

## 示例 1

在文本样式化一文中, F. Scott Fitzgerald 作品的开头部分的排版是由 CSS 做成的。本文中的示例用的是同一篇文章, 只是对这篇文章做了些小改动 (主要是在正文周围加上一个容器元素)。没有对其文本式样作改动, 只是去掉了该示例中用到的一些布局样式。

首先, 我们将边距加到所有需要边距的元素上。

链接:

- 只做了最低程度的样式化处理的示例文件
- 初始样式表
- body, 标题, 格言引用, 文件容器和段落上的新边距
- 示例 1 的样式表

新规则:

```
body { margin: 0; }

#main { margin: 0 auto 0 auto; }

h1 { margin: 0 0 1em 0; }

.pullQuote { margin: auto 0 1em 1em; }

p { margin: 0; }

.attribution { margin: 0 0 1.5em 0; }
```

## 给对象添加边框 : 边框属性

border 缩写属性有是有, 但是只有在给一个元素四边都加上完整一致的边框时才能用到它。通过下列属性的任意有意义的组合, 可以给一个元素的四个边框之一设置粗细 (宽度)、样式和颜色:

- border-width
- border-style
- border-color
- border-top
- border-top-width
- border-top-style
- border-top-color

- border-right
- border-right-width
- border-right-style
- border-right-color
- border-bottom
- border-bottom-width
- border-bottom-style
- border-bottom-color
- border-left
- border-left-width
- border-left-style
- border-left-color

### **border-width 属性**

这些属性的作用与人们所预期的一致：它们给一个或多个边框指定明确的粗细.

`border-width` 缩写属性可支持的属性值表示法与 `margin` 缩写属性的一样，只是不支持百分比值。你可以像下面这样写规则：

```
td {
 border-width: 1px 0 0 1px;
}
```

### **border-style 属性**

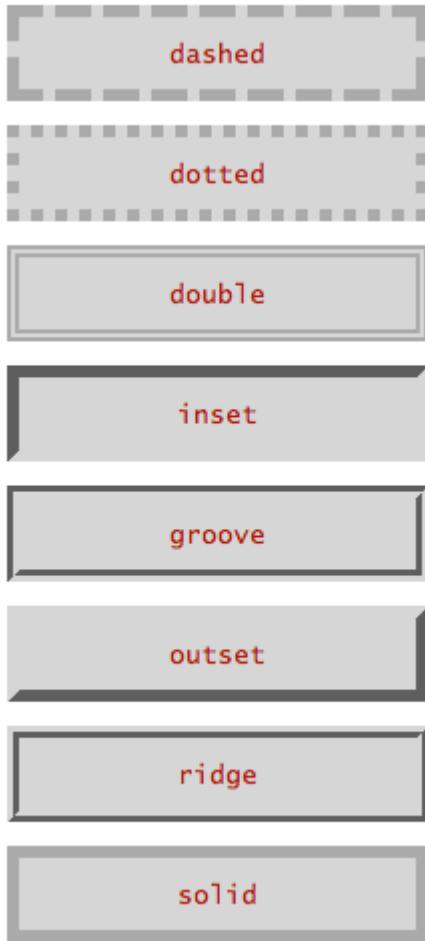


图 4: 八个常见的边框样式的运行效果。

`border-style` 属性通常接受下列这些值：

`dashed`

虚线的长度，及它们之间的空白的大小视浏览器而定。

`dotted`

点（形状不定，长宽比为 1）与点之间的空白的大小视浏览器而定。

`double`

设置的宽度将分成三段，按照填充—不填充—填充的顺序渲染。

`groove`

会紧贴在边框内部渲染出 `outset`，并 `inset` 对齐。

`inset`

边框会带有阴影效果，看起来像是把应用了该样式的元素压进了画布。

`none`

相当于对 `-width` 属性赋零值。

`outset`

边框会带有阴影效果，看起来像是把应用了该样式的元素挤压出容器之中。

`ridge`

会紧贴在边框内部渲染出 `inset`，并与 `outset` 对齐。

`solid`

边框表现为一条没有断点也没有阴影的线。

应用 `border-style` 缩写属性时，一次至多可写四个值，其形式与 `margin` 缩写形式一样。

边框的淡化（而不是省略边框）处理是由 `-color` 属性进行的。

## border-color 属性

最后，我们可以对任何一个边框设置任意颜色，这种操作可以通过单独对上面所列出的属性进行设置来完成，也可以通过 `border-color` 缩写属性来完成。对于在缩写属性中给出的值少于四个时产生的效果，详见 `margin` 缩写属性说明。

像 `background-color` 一样，`border-color` 可以取 `transparent`。这一点在对付要求组合一致但不要求边框使用一致的边缘情况时会很有用。

## 详述边框缩写属性和它的四个近亲

和各种 `-width`、`-style` 和 `-color` 属性不同，你可以用这五个属性来定义某个对象的四侧边框的三个特征，或每次定义任一边框的三个特征。合法的 `border`（等等）缩写值包含所有或任一应用于此边框的宽度，样式和颜色属性；唯一的限制是你必须要么一次只提交某个元素的一侧边框，要么四侧边框全部提交。

看看下列 `border` 规则：

```
#borderShorthandExample {
 border: 2px outset rgb(160,0,0);
 padding: .857em;
 background-color: rgb(255,224,224);
}
```

应用了上面规则的元素会产生与此处相同的效果。

如果在 `border` 缩写规则中省略某个值时，渲染出来的元素就会表现出对应的默认效果；

- 边框宽度视浏览器而定。
- 边框样式将是 `solid`。
- 边框颜色将与该元素所应用的 `color` 一致。

### 创建线条: 五个... 而不是一个边框缩写属性的基本原理

这里所说的“rules”是指划过某个布局的线条，而不是要遵从的指令。这样的线条可以增强一个元素与它旁边的空白的对比度，在许多情况下，它们有助于制作一个布局中的立体感效果。`inset` 和 `outset` 边框样式就是这种效果的例子。

对一个元素的四边都加上边框可以得到相同的效果，而这种在布局上划出精确界定的线条的技能可以使设计者对细节有相当大的控制权。

### ...为什么要这么多属性?它们只是边框而已, 不是吗?

如果一个布局的创建要求设计师具备高超的技艺，这就需要对边缘情况进行处理；这个问题已在早先讨论边距时提出了。

因为网站设计方式的缘故，你会遇到许多这样的情况，同一 HTML 文档中的某个元素与其它元素结构属性相似，却要求有不同的外观。在这些情况下，对大多数元素编写写一种规则，而对每个边缘情况编写附加规则就会非常有用。这就是 `auto` 和 `inherit` 值存在的原因：将默认样式作为边缘情况处理。

对边框来说，边缘情况很可能要求对某个元素某侧的边框的某个属性进行改变——如果某个人明智地选择遵从 KISS 法则，最好坚持只改变需要变动的细枝末梢。

### 示例 2

文件中的某些部分需要以规则和边框的形式进行修饰。

链接:

- 在 `title` 的尾部添加规则，并给格言引用添加边框
- [示例 2 的样式表](#)

新规则:

```
h1 { border-bottom: 1px solid rgb(153,153,153); }
.pullQuote { border: 1px solid rgb(153,153,153); }
```

### 只有边距是不够的: 填充距属性

你可能会遇到背景颜色处于次要地位或带有需要在内容和边距之间留出缝隙的强调色的元

素。你也可能遇到需要在边框和靠近边框的文本之间留出空隙的情况。

在此如此种种情况下，`padding`、`padding-top`、`padding-right`、`padding-bottom` 和 `padding-left` 属性就可以大显身手。这些属性用于在元素和其内容的边距或边框之间插入负间距。边距，边框和填充距之间关系的明确图解请参见上面的图 1。

这些属性的运作方式与边距属性完全相同，除了以下例外：

- 对填充距属性来说，`auto` 值并不起作用。
- 负填充距值是不合法的。
- 填充距不会合并。
- 边距值不是用于内联元素的，而填充距值才是。

### 示例 3

应当在之前已经添加了边框的元素上添加间隙。

链接：

- 在之前添加到标题和格言引用的边框和文本之间插入空隙
- [示例 3 的样式表](#)

新规则：

```
body { padding: 0; }

h1 { padding: .5em 0 .5em 0; }

.pullQuote { padding: .5em; }
```

### 对元素的宽度和高度进行处理

通常大部分元素的尺寸可以被更改。这种性质已在前面讨论 `auto` 边距时说明过了。

用来控制元素尺寸的 CSS 属性包括 `width`、`height`、`min-width`、`max-width`、`min-height` 和 `max-height`。这些属性可通过 `overflow` 属性与元素内容的尺寸相分离（或连接）。

此外还有 `clip` 属性，用于将元素的一部分隐藏于其边距中。但由于并不常用，所以在本文中就不详述了。

### 宽度和高度基础

通常，人们可以预见 `width` 和 `height` 的效果。但是，使用时也有些地方需要注意。

- **width 和 height 不能用于 inline 元素...** 在特定情况下一些元素（如 `span`、`strong` 和 `em`）会无视 `width` 和 `height` 的值。这些元素的清单将在后面对元素类型进行探讨时列出。
- **...尽管图像是内联元素,但它们是可以指定 width 和 height 的。** CSS 2.1 推荐将图像看做“被替代”元素,这意味着浏览器会把它们当作含有静态尺寸对待。因此,这些尺寸可以被任意更改。
- **只有 width 和 height 属性可以影响一个元素的功能性尺寸。** 因此,常有这种情况出现:一个元素太小(常常是太窄)而不能像预期的那样装下它的内容,从而引起冒出。下面论及的 CSS 盒模型将会牵扯到这一问题。
- **微软 Internet Explorer (IE) 的渲染漏洞,使得明确指定一些元素的 width 或 height 属性/值组成为必要。** IE 的渲染引擎的一些缺陷只能以强力解决掉(参见词汇表)。人们已了解了很大一部分缺陷,并准备要在 IE 8 中把它们消除掉,但在 IE 8 在 IE 安装基础中取代其前辈之前,这个问题将是无法回避的测试用例。[PositionIsEverything.net](#) 和 [CSS-Discuss Wiki](#) 提供了大量有关于这个问题和其解决技巧的信息。
- **有时取整算法会导致浏览器间不合规格的布局差异,这些浏览器是通过 LCD、LED 或 CRT (`type="screen"`) 显示媒体来展示内容的。** `screen` 媒体类型最终会要求所有单位换算为像素度量单位,而浏览器间的像素度量单位会各不相同。

### **min-width、max-width、min-height、和max-height**

有时,你会遇到这种情况:你需要对一个元素的大小进行约束——通常是为了保证一个大小为比例值的列能够保持在一个易于阅读的宽度上。各种 `min-` 和 `max-` 属性正是作用于此。与 `width` 和 `height` 情况一样,通常这些属性的使用效果是可以预见的。

但是以笔者的经验来看,这些属性的用途有限(尽管替他一些作者持异议)。像以前简单的 `width` 和 `height` 一样,它们也受到取整误差的困扰,这种误差很能会导致完全意外的后果。更重要的是,IE 6 完全不支持它们。到 2008 年七月为止,IE 6 在市场上仍占有很大份额。

### **示例 4**

`auto` 边距是放在页面容器的左侧和右侧的。现在为了使这些边距值有意义,我们来为其设置 `width`。此外,还要给格言引用设置一个 `float` 值,使它也获得宽度。

**链接:**

- 改变文档容器和格言引用的宽度
- [示例 4 的样式表](#)

## 新规则：

```
#main { width: 42em; }
.pullQuote { width: 14em; }
```

## 溢出：是对内容进行限制，还是给它自由

设定元素的 `width` 或 `height` 时，有时会需要考虑在该元素的内容占据的空间超过了严格意义上可获得的空间时，我们期望得到什么样的效果。在那些既有用户生成内容，又有严格布局规范的网站上尤其需要注意这一点。

`overflow` 属性和它的四个有效值——`visible`、`hidden`、`auto` 和 `scroll`——可用于处理这种情况。图 5 展示了将这几个属性值应用到一个内容超出其边界框的元素上所产生的效果。

<b>visible:</b> <i>  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut magna quam, auctor non, rhoncus facilisis, pharetra at, massa. Phasellus sagittis rhoncus turpis ipsum, volutpat vel, sodales et, lacin. Phasellus consectetur cursus vel, lectus. In libero. Aliquam sed eros. Ut sagittis. Nullam ac justo id urna semper fringilla.</i>
<b>hidden:</b> <i>  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut magna quam, auctor non, rhoncus facilisis, pharetra at, massa.</i>
<b>auto:</b> <i>  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut magna quam, auctor non, rhoncus facilisis, pharetra at, massa.</i>
<b>scroll:</b> <i>  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut magna quam, auctor non,</i>
<i>  Morbi elementum fringilla enim. Maecenas turpis ipsum, volutpat vel, sodales et, cursus vel, lectus. In libero. Aliquam sed eros. Ut sagittis. Nullam ac justo id urna semper fringilla.</i>

图 5: CSS overflow 属性的效果。

四种溢出属性值的效果

### `visible` (默认)

在不影响相邻元素的 `flow` 或边距的同时，显示超出元素可用空间的内容。因此，某个元素的内容将与其相邻元素的内容相冲突。在后面的教程中，会对避免这种情况的技巧，以及在 `IE` 中因渲染问题而产生的特殊情况进行讨论。

### `hidden`

超出该元素的界限的所有内容将被隐藏而不可见。

### `auto`

跟用 `hidden` 值时一样，元素的尺寸会受到限制，但是会创建一个滚动条，使访问者可以获得溢出的内容。

### `scroll`

即使在不必要的情况下，也会在元素上添加垂直和水平滚动条。

## CSS 盒模型：将一切组合起来

既然我们已经讲完了基本的布局属性，接下来我们就来看看浏览器是如何按照一个元素的 CSS 属性来渲染它的宽度的——以及避免你的元素从布局中溢出。有些效果非常好，而另一些似乎毫无道理可言。麻烦的是有两种布局算法需要考虑：在 `CSS 2.1` 推荐中，由万维网联盟（W3C）所指定的模型，和较早版本的 `IE` 中所用的模型。

### 为你的布局选择适当的单位

对于文本来说，元素的大小可以用比例单位 `%` 或 `em`，或者像 `px` 这样的静态单位来设置。需要考虑的还有浏览器画布的大小永远是静态值，如果不用客户端 `script` 代码，是不可能恢复这个尺寸或调整窗口大小的——这种情况对于可访问性，可用性和媒体便携性的要求来说很不合适。

### 元素大小调整的重要规则：混用比例单位和静态单位时应慎重，或者干脆不要混用

默认的 `width` 和 `height` 值是 `auto`，这在标准英语中意即“使用所有可用空间”的指令。对块元素来说其效果是，计算出的 `width` 占完了所有可用空间。对 `height` 来说，默认情况下元素会扩展来容纳它们的所有内容。

改变 `width` 和 `height` 值时，必须小心确保元素的内容（带有边距，边框和填充距）与你所指定的宽度相适应。为了实现这一点，最简单的办法就是进行下列步骤：

1. 在普通的显示器和/或字号条件下，考虑布局可用的最大宽度。到本文撰写时为止，这个数值一般是大约 800 或 1024 像素。你的网站受众越广，越应选这些值中较小的值。
2. 为你的整个文档创建一个容器元素，该元素的宽度在第一步所得宽度之内。

3. 为第二步所创建的容器元素中的元素设置布局属性时，采用与该容器元素相同的单位类型。

### 为布局选择适当的单位类型：优缺点

单位	优点	缺点
em	最适合于创建精确的二维布局 栅格 用在与文档容器有关的方面时，会使布局根据正文大小而扩展或收缩 可以使元素大小的计算值变得很容易估计	分数单元可能会由于浏览器之间的细微差异而扩张或缩小 为了达到最佳效果，文档中所有 <code>font-size</code> 和 <code>line-height</code> 值都应当设为明确而可预计的值
百分比	最适用于完全弹性布局 对于创建等宽栏之类的对象来说这是最简单的办法	为了避免内容冒出，可能会需要创建额外的容器元素 可能会导致元素过宽或过窄 其效果高度依赖于环境（参见下面的盒模型部分）
px	可以提供最高的布局控制权 可以消除大多数因浏览器差异而造成的变化	最不能满足可访问性和跨媒体支持要求 s 最容易受到内容冒出的影响

表格 1：设置布局属性时，百分比，em 和像素单位的优缺点

### 盒模型组件

盒模型实际上只是一系列指令，用来定义元素的种种布局规范如何相互作用。盒模型涵盖的组件有：

1. 文档画布
2. 边距
3. 边框
4. 填充距
5. 元素的宽度和高度
6. 子元素属性

上面的最后一项依次包含了其它 5 项。但是为简明起见，这一部分将集中讨论简单的父元素一子元素关系，多级盒模型交互作用将保留到后面详谈页面布局的文章中再做深入讨论。

### **W3C 的盒模型：一切都是附加的**

基本规则就是一个元素的宽度或高度计算值等于：

```
margin + border + padding + (width|height)
```

在许多情况下，`width` 和/或 `height` 都设置的是其默认的 `auto` 值，这就是说为页面内容预备的画布区域等于：

```
available_canvas ? margin ? padding ? border
```

在这样的等式中，`available_canvas` 本身是一个离散值（如果是自动计算的话），用它来减去边距，边框和填充距。这个数字对元素的宽度极为重要，设计者的计算错误将会导致在浏览器窗口上出现水平滚动条这种让人头疼的结果。另外，浏览器通常会将元素置于浏览器画布的左边，但它可能会溢出浏览器窗口的右边，除非有其它命令来限定。

看看下面的样式表规则：

```
#myLayoutColumn {
 width: 50em;
 margin: 1.5em auto 1.5em auto;
 border: .1em;
 padding: .9em;
}
```

如前面在讨论边距属性时说过的那样，`#myLayoutColumn` 会将自身对齐到其容器元素的中心，无论其容器是 `body`，还是由开发团队做出的其它东西。

另外，如果“严格模式”的激活（通过一个适当的 `!DOCTYPE` 声明）应用了 W3C 的盒模型的话，也可估算出非边距 *al* 宽度是：

```
.1em + .9em + 50em + .9em + .1em = 52em
```

在 `screen` 媒体上，浏览器会采用此值，将所有属性值分别取整为最接近的像素值，并按照这些值来进行最终效果渲染。

### **W3C 盒模型中的百分比边距和填充距**

使用 W3C 盒模型时，百分比的边距和填充距是相对于其包含元素的 `width` 计算值来计算的。例如，如果为一个包含在 800 像素宽的元素中的元素指定 `margin: 20%`，该元素周围

的边距就是每边 160 像素（800 的 20% 是 160）。

如果对这个元素指定 `padding: 5%`，它的内容宽度计算值就是 400 像素：

```
20% + 5% + 5% + 20% = 50%
0.50 × 800 = 400
800 - 400 = 400
```

## 对文件流进行处理

由于后面的教程要讨论多栏式布局的制作，本文下面将介绍三个 CSS 属性：`display`、`float` 和 `clear`。

### 元素类型和显示属性

除 `html`、`body` 和 `table` 部分之外，在 HTML 4.01 推荐中与主要内容有关的所有元素都有一种相关联的类型：内联或块。每种类型以不同方式决定了默认布局表现：

#### inline

- 紧接在内联元素前/后的文本和图片与内联元素的内容渲染为同一基准线。但如果该文本或图片过长，它们就会与包含元素的边缘相重叠，在这种情况下，内嵌内容就会跑到一条新基准线上面，而该基准线位于之前的那条基线的下面。
- 内联元素中，文本行的布置会视需要（或允许）而带有软换行符，除非用 `white-space` 属性修改了这种方式。
- 样式表规则中可应用于这些元素的 `margin`、`width`、`height` 和 `float` 属性被忽略。（除了 `img` 和 `object`）
- 内联元素只能容纳文本或其它内联元素。

#### block

- 这些元素将在其容器中渲染为离散块。
- 这些元素的前后将总是渲染有断行点，除非将 `float` 值设为 `left` 或 `right`。
- 如果嵌套块元素之间没有任何内容，它们之间的断行点通常会被合并。
- 宽度为 `auto`（默认）的块元素会一直扩展，直到占满所有可获得的宽度。

`display` 属性有三个常用的值——`block`、`inline` 和 `none`——其中两个是指对应的元素类型。改变元素的 `display` 值可以使内联元素的表现方式像块元素那样，也可以使块元素的表现方式像内联元素那样，还可以改变文件的渲染效果，使之看起来就像元素（和它的所有内容）

根本不存在一样。

一般来说,了解默认情况下哪些元素与哪些类型相对应是很关键的,其关系简列于表 2 中:

元素	类型	子类型	说明
a	inline	special	
abbr	inline	phrase	
acronym	inline	phrase	
address	block		一般与 p 的行为方式类似
blockquote	block		在!DOCTYPE 声明为 strict 时,必须包含至少一个块元素
body			封装了整个文档画布;一般会有 10px 的边距(在 IE, Firefox 和 Safari 中)或是填充距(在 Opera 中),像素大小视所用的 screen 媒体而定
cite	inline	phrase	
div	block		
em	inline	phrase	
fieldset	block		默认情况下一般按照 border: 1px black; 渲染
form	block		
h1 ... h6	block	headin g	
input	inline	formctrl	
img	inline	special	
label	inline	formctrl	

元素	类型	子类型	说明
li	block		文档类型定义中没有指明该元素的类型，但该元素可包含块元素和内联元素；完成版的 CSS2.1 推荐为列表项留出了 display 值
ol	block	list	
p	block		只能包含内联元素；一般会渲染有顶边距和底边距
span	inline	special	
strong	inline	phrase	
table	block		
ul	block	list	

表2: 常用 HTML 元素和它们的类型。只有具有相同子类型的相邻块元素之间的边距才会合并。

### 示例 5

只是为了做个演示，将标题中的“Prologue”注释掉怎么样？

链接:

- 从标题中移除不相干的部分
- 示例 5 的样式表

新规则:

```
.sectionNote { display: none; }
```

使元素环绕于其它元素周围: 浮动属性



本段左边放了一张照片。当然你们都会看到后面的文本自然地环绕于其周围，尽管有人得先停止琢磨为什么著名的科幻小说家要在他的猫身上绑块儿熏肉——即使今天是他的慢调生活日。我们可以用 HTML 属性来指定你所看见的布局表现，但在这个例子中该效果是由 CSS 完成的。

可以猜到，这个戏法是由 `float: left;` 这一属性/值组完成的。在后面的文章中将会介绍运用浮动这个好点子，但有必要在这里谈谈它的基础。`float: right` 是极有用的属性/值组。如果你要取消一个引用了 `float` 的 `class` 赋值，你可以指定 `float: none`。

`float` 属性有一些用法说明：

- 只有将 `float` 值应用于带有显式 `width` 的块元素时，它才起作用。
- 可以将 `float`、`clear` 和 `margin` 属性一起用在样式表中，来在布局中创建栏。
- 让一个浮动元素延伸到其容器的底部比较难，但也不是不可能。这样做的常用方法被称作 伪列布局。

## 示例 6

之前已经谈过了在格言引用上赋 `float` 值的问题。那么完成之后，我们就可以来看看效果。我们可以添加一些背景颜色使它与主要内容区别开来。

链接：

- 将格言引用浮动至右边距处
- [示例 6 样式表](#)

新规则：

```
.pullQuote { float: right;
background-color: rgb(204, 204, 204); }
```

### 强制使元素处于其前一个浮动元素之下：清除属性

像 `float` 属性一样，可指定 `clear` 属性为 `left`、`right` 或 `none` 三者之一。清除属性也支持 `both` 值。

`float` 属性指令其它元素的内容如何环绕浮动元素，`clear` 属性则规定了元素如何环绕其相邻元素——在许多实例中，是规定不要浮动。

图 6 显示了 `clear: left;` 的表现；此布局中两个原来连在一起的元素被指定了相同的 `height` 值，而 `float` 值分别设为 `left` 和 `right`：

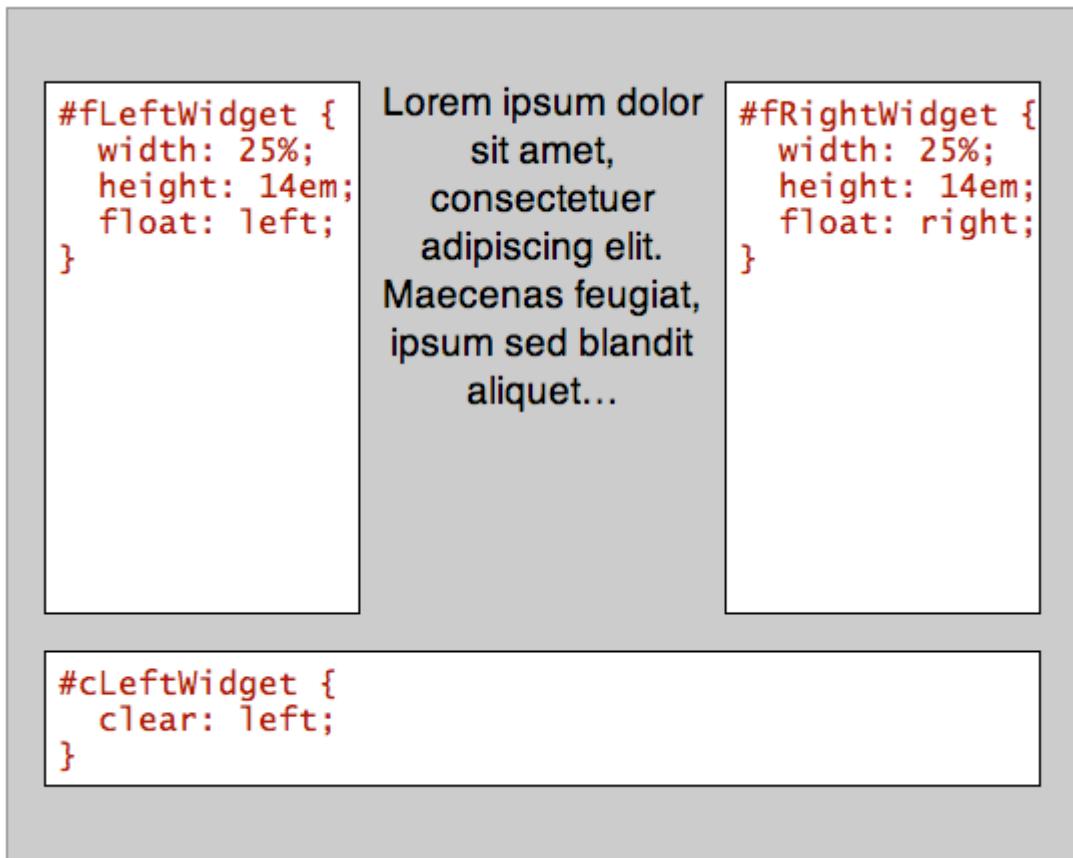


图 6: 因为高度相当, `clear:left` 使底部盒子清除了两列的浮动。

前一示例中 `#cLeftWidget` 的默认环绕会将它置于希腊语文本下面——即, `#fLeftWidget` 和 `#fRightWidget` 之间。

想想看, 如果这一组元素中前一个元素比其右对齐的同僚短, 会出现什么效果? 如图 7 所示:

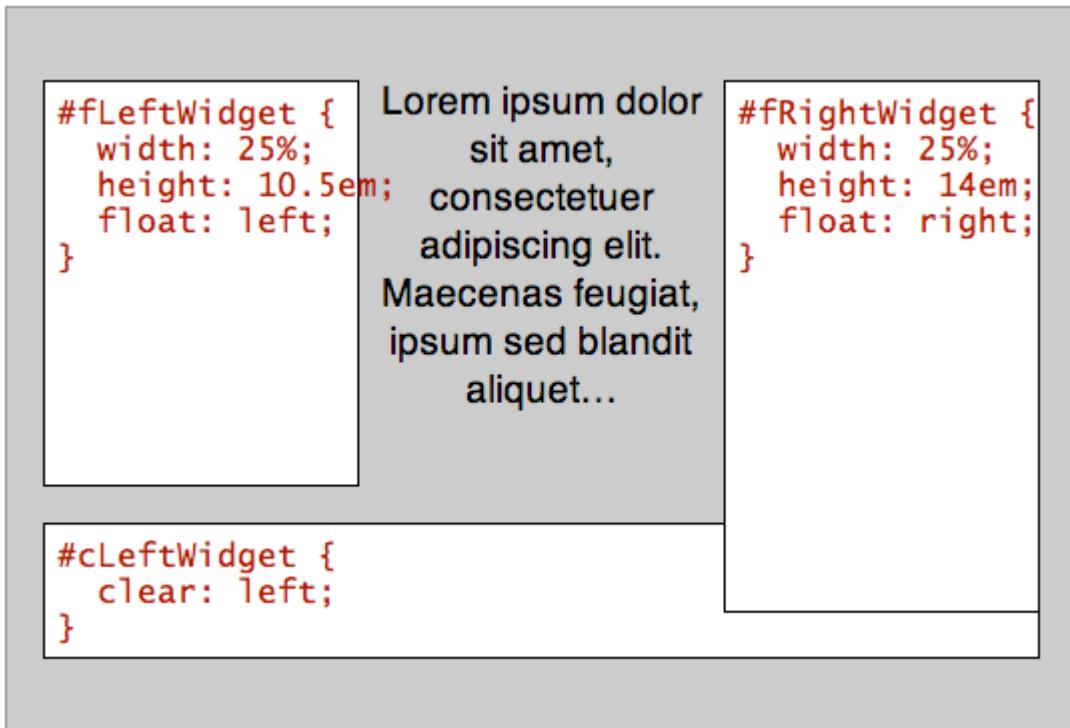


图 7: 当右列比左列长时, `clear:left` 不会清除两列, 所以必须用 `clear:both`。

在前一个例子中, 后续元素的 `clear` 值设为 `left` 是为了表明一个观点: 因为两个 `float` 的元素等高, `clear` 的元素会被推到两者下面。但是, 第二个例子证明为了从高度不等的 `float` 元素取得相同的结果, 必须用到 `clear: both;`。

这里对 `clear` 属性的探讨可看做其效果的简单介绍, 后面的文章会更深入地对该属性的用途和技巧进行讨论。

## 总结

由于渲染引擎的种种差异, 对全面涵盖传统的定义基础的需要, 对浏览器窗口大小预计的无能为力, 对 Web 文档的布局都充满了困难和限制。但是, 一般的 CSS 支持水平已经可以轻易地使 Web 文档在浏览器上得到满意的效果。

## 延伸阅读

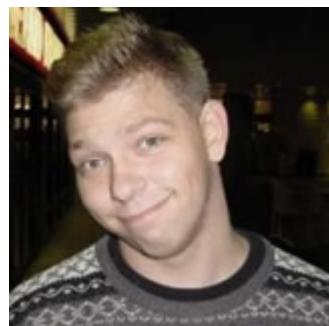
- Bergevin, Holly, and Gallant, John. 2006. Explorer exposed. *Position Is Everything*.  
<http://positioniseverything.net/explorer.html> (accessed 1 July 2008).
- Bos, Bert, et al. 2007. Cascading style sheets level 2 revision 1 (CSS 2.1) specification. World Wide Web Consortium.  
<http://www.w3.org/TR/2007/CR-CSS21-20070719> etc. (accessed 30 June 2008).
- Raggett, Dave, et. al. 1999. HTML 4.01 specification. World Wide Web Consortium.  
<http://www.w3.org/TR/1999/REC-html401-19991224> etc. (accessed 30 June 2008).

- Raymond, Eric, and Steele, Guy, eds. 2003. Brute force. The Jargon File (Version 4.4.7). <http://www.catb.org/jargon/html/B/brute-force.html> (accessed 30 June 2008).
- Scalzi, John. 2006. Clearly you people thought I was kidding. Whatever. <http://www.scalzi.com/whatever/004457.html> (accessed 30 June 2008).

## 练习题

1. 缩写的 `margin` 值或单个边距属性如 `margin-top` 分别适用于什么情况？
  2. 什么时候缩写的 `margin`、`padding` 和 `border-width` 属性要写出全部四个值，这些值是按什么顺序作用于元素的四边的？
  3. **I** 如果你想在一个文档里所有标题的文本下都画一条线，你该使用那种属性？
  4. 为了使一个元素看起来像一个界面按钮，你会用那种 `border-style` 值？
  5. **判断：** 默认情况下，在元素周围添加边框会不会也在这个元素的内容周围添加缝隙？如果创建的元素跟它的容器不一样宽，你需要设置那组属性/值以确保元素在其容器内水平居中？
  6. **判断：** 如果将一个容器元素置于 `body` 中，给它的 `width` 设一个大于 `100%` 的值，文档画布的表现会不会变？
  7. 如果图片对包含元素来说过大，你会用哪种属性/值来保证你的页面不会溢出，为什么？
  8. 给 `a`（链接）元素的 `display` 值设为 `block`，并给它设一个合理的高度和宽度，在 `screen` 显示媒体上这个链接的鼠标经过行为是如何变化的？
  9. 通常情况下，块元素会扩张来填满其容器的宽度（除了边距，边框和填充距）。默认情况下，如果该元素前面有一个 `float` 的元素，该元素的行为是确实发生了变化，还是只是看起来发生了变化？
  10. 如果想要给某个元素应用一个 `float` 值，在那个元素上你还需要设定什么属性？
  11. 如果你想**绝对**保证一个元素总会扩展而填满其容器的宽度，你该设置哪组属性/值？
- 上一篇：使用 CSS 设置文本样式
  - 下一篇：CSS 背景图片
  - 目录

## 作者简介



Ben Henick 从 1995 年 9 月就开始以各种身份参与创建网站了，那时他以一个学术志愿者的身份进行了他的第一个 Web 项目。从那时开始，他的大多数工作都是以自由工作者的形式完成的。

Ben 是一个多面手。他的技能触及了网站设计和开发的几乎每一个方面，从 CSS 到 HTML，再到设计和文案撰写，再到 PHP/MySQL 和 JavaScript/Ajax。

他生活在堪萨斯州的 Lawrence，有三台电脑，却没有电视机。你可以在 [henick.net](http://henick.net) 上读到更多关于他和他工作的内容。

## 31. CSS 背景图片

Posted 12/15/2008 - 16:57 by Lewis

- 网络标准教程

作者: Nicole Sullivan · 2008 年 9 月 26 日

- 上一篇: CSS 布局模型——boxes、borderes、margins、padding
- 下一篇: 样式列表和链接
- 目录

### 引言

承认吧, 读过本课程的第一篇文章后, 你一定非常渴望了解如何使你的网站看起来既美观又引人入胜。甚至你可能是直接跳到了这一节, 是吗?

背景图像就是用来使你的网站美观迷人的, 不过你可能会惊讶于它们与你之前学到的基本概念的联系是多么的紧密。

正如你之前在本教程中了解到的那样, 随 CSS 而来的最重要的变化就是将外观独立出来的能力, 从语义学方面来说就是某个东西看起来怎么样, 或者是它的意义是什么。CSS 背景图像是你可以随意支配的最重要的工具之一, 这是因为它使你可以将装饰图片应用到你的 HTML 的特定部分, 而无需向你的 HTML 添加任何冗余内容。过去, 网页制作者(也就是你!)必须将 `img` 标签穿插在代码中间, 代码间充满了 `img` 标签。

特别是 `background` 属性使你的 HTML 脱离了表述混乱。因此, 通过现代手段, 在网站的建设周期中重新设计和其它转变就可以更加顺利的完成。你可以通过仅仅更改样式表, 而非对每个 HTML 页面重新编码, 即可对你的整个网站进行更新。这将带来巨大的成本节约, 具体数额取决于你的网站的大小。

在本文中, 我们将会学习到 CSS 背景图像的工作原理, 包括通过 CSS 应用背景图像, 调整其位置, 对其进行水平或垂直铺设, 以及使用 CSS 图像子画面技术组合背景图像, 从而提高网站性能。

本文结构如下:

- 它是怎样工作的?
  - 背景属性

- 创建警报对话框
  - 代码
    - 创建 CSS 钩子或 CSS 选择符
    - 添加背景颜色
    - 应用背景图片
    - 控制背景平铺
    - 固定
    - 对图片进行定位
    - 像专业人员那样使用缩写集中表述
  - 用代码来做实验
  - 质量测试
- 子画面
  - 一个复杂的子画面和背景图像示例
    - 创建子画面
- 总结
  - 图片来源
- 练习题
- 延伸阅读

## 它是怎样工作的?

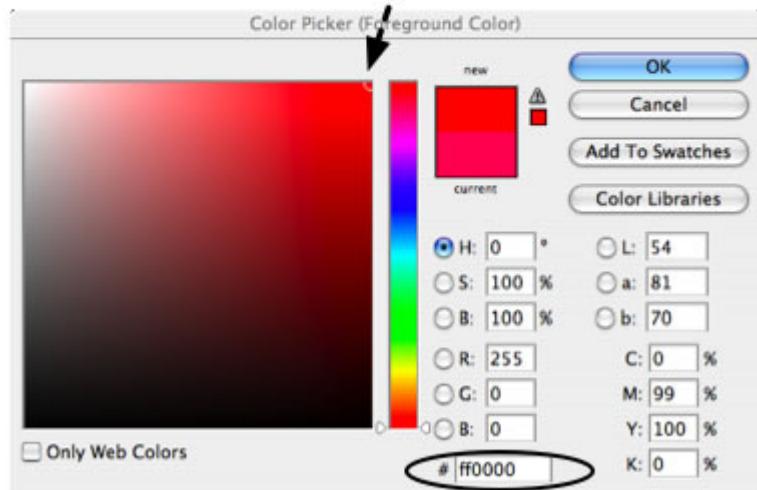
用来控制背景的 CSS 分成了若干不同的属性。使用 `position` 和 `color` 之类的属性，你就可以着手控制你的页面的感官效果。在本文中，你将全面了解 CSS 背景图像，并逐步建立一个警报对话框。

首先我们来进一步了解一下可供我们使用的各种属性。

### 背景属性

属性	定义	描述
<code>background-color</code>	设置 HTML 元素的背景颜色	<p>有多种方式可以用来指示 <code>background-color</code>，包括 RGB 值和关键字。大多数人使用十六进制符号——一个井号 (#)，后跟六个字符。#RRGGBB：前两个字符表示红色水平，第二对以及最后两个字符分别表示绿色和蓝色水平。</p> <p>许多颜色采集工具可帮助你找到特定颜色的十六进制符号。例如，纯红是 #FF0000。</p>

Tools like Photoshop will allow you to choose a color, such as red, and will give you the hex value.



有效值包括 color 值、transparent 或 inherit。

<code>image</code>	指示背景图像的路径或 URL。	通过使用 URL 告知浏览器图像的位置来设置 background-image。例如: url(alert.png)。注意, 路径以关键字 url 开头并带有圆括号。这种语法很重要, 因为这样浏览器才能明白你是在指定一个位置。 有效值包括 URL、none 或 inherit。
<code>repeat</code>	指示背景图像平铺的方向。	图像可以垂直平铺、水平平铺, 或者既有垂直又有水平方向的平铺, 来占满 HTML 元素的全部高度和宽度。使用 background-repeat 可指示浏览器对一个背景图片进行平铺。 有效值包括: repeat、repeat-x、repeat-y 和 no-repeat。
<code>attachment</code>	设置用户滚动页面时背景图像的行为。	图像既可以随内容滚动, 也可以固定在显示屏幕上。有效值包括 scroll、fixed 和 inherit。
<code>position</code>	告知浏览器背景图像的位置。	图片可以在其应用到 HTML 元素的边界内任何位置显示。使用 background-position 可将图像放置在恰当的位置, 以实现视觉效果和层次感。 有多种方式可以表示背景定位, 比如关键字和数值。关键字 (如 top 和 bottom) 非常管用而且易于阅读。像素值非常精确, 但不适用于更改高度和宽

	<p>度。使用 CSS 图像子画面技术时，负像素值很有用，这点稍后你就会了解到。</p> <p>尽管像素和百分比值定位图像的作用方式差异很大，但是使用像素和百分比值时，起点始终是 HTML 元素的左上角。不论图片和容器的大小如何，用像素来定位图像时总是将图像向容器的底部和右侧（或如果是负数值，则向顶部和左侧）移动设定的像素值。而用百分比值来定位图像时则是按照容器大小和图像大小之间差额的百分数来移动。如果容器和图像一样大，这种方式将无法移动图像。</p> <p>有效值包括 <code>length</code>（通常单位是像素），<code>percentage</code>（元素宽度的百分数），以及关键字 <code>top</code>、<code>right</code>、<code>bottom</code>、<code>left</code> 和 <code>center</code>。注意，中心可以指水平中心或垂直中心。另外注意，你可以将百分比和像素结合使用，但关键字和像素或关键字和百分数不能结合使用。</p>
<code>background</code>	<p>一种缩写属性，可用于在一行代码中描述所有其它属性的。</p> <p>缩写属性确实很好用。许多开发人员使用缩写属性使 CSS 尽可能精简，并集合所有有关联的属性。你可以使用缩写来编写一个一般规则，然后在需要的时候用某个特定属性将其覆盖。</p> <p>所有属性必须以一致的顺序表示，以便浏览器可以轻松解读你想要的样式：</p> <pre>color url repeat attachment (very rarely used; may be omitted) horizontal-position vertical-position</pre> <p>用到所有属性的缩写（<code>attachment</code> 除外）示例如下：</p> <pre>background: green url(logo.gif) no-repeat left top;</pre>

## 创建警报对话框

现在有关的基本概念已经介绍完了，下面我将带你建立一个完整的警报框实例，该实例将涉及背景图像的所有方面。

### 设计

我们假设一个平面设计师已提供了一个你要为你的网站创建的警报对话框的可视模板。看看这个警报，你看到的背景是浅橙色的，在周围的段落中很鲜明。左上角还有一个 10 像素的警报图标。

注意，这个模型有一行文本，但是以后可能会包含更多文本。`web` 开发员的一项最重要的技能就是，可以预见一项设计在未来会如何演变。在对网站给人的视觉上的艺术感受的考虑中，有一部分就是关注从开始设计到重新设计的连贯性。因此警报对话框可能包含一行以上的文本，甚至多个段落、列表或其它 HTML 元素。你应当尽量努力做一个元素不可知论者——尽可能相

信网站以后会有很多目前不可知的变化。这将增加代码再利用的可能性，并使网站尽可能高效。模型如图 1 所示。

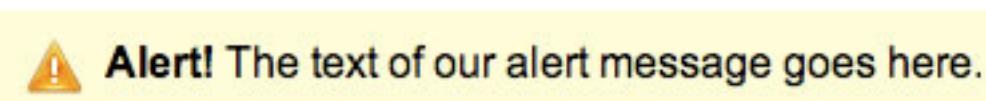


图 1：平面设计师所提供的我们的警报框模板。

平面设计师还提供了我们要用的图标，如图 2 所示。



图 2：警告图标。

## 代码

根据你在本文第一部分所了解到的 CSS 背景的相关内容，你可能已经在思考如何建立这种警报对话框。我希望你现在就开始尝试，然后把你的成果和我的示例作个比较。

好的，试过了吗？现在我们来逐步讲解一下。每个屏幕截图都链接到了代码示例上，因此你可以找到每一部分的源代码。用代码来进行试验，试着增加或减少属性值，或者尝试替换某个属性。你可能想要跟踪每一步的效果，可以在 Opera Dragonfly 或 Firebug 之类的工具中写下每行新代码，这样你将可以立即看到每一步的结果。

### 创建 CSS 钩子或选择符。

首先你需要创建一个 `alert` 类，以供 CSS 追踪。创建新的 CSS 和 HTML 框架文档，将 CSS 链接到 HTML 文档并将以下代码添加到这两个文件中：

#### CSS:

```
.alert { ... }
```

#### HTML:

```
<p class="alert">
 Alert! 此处是警报对话框文本。
</p>
```

在这里，我采用 `class` 来设计警报，而非 `id`，这是因为页面中可能有一个以上的警报，例如带有多个错误的表单元素。在建立 HTML 时，你应该使你的 CSS 尽可能灵活，并使所有

元素都与设计一致。

好了，你已创建了一个稳妥的基础，但由于你尚未添加任何修饰，因此看还是像普通段落易于。下面我们就来做这一步。

注意：我特意选择不限制 `alert` 类的段落；这样就可以方便地让警报框包含其它元素。你应当使你的 CSS 尽可能地灵活。

### 添加背景颜色

在第 29 章用 CSS 来样式化文本中你已经学会了将 背景颜色用在文本处理中。同样的概念适用于任何 HTML 元素并且可与背景图像结合来创建视觉效果。如果背景颜色既没有被设定，也没有继承其它的设置，将默认为透明。

我们将橙色背景颜色添加至警报框，使其从周围的本文中凸显出来。保持 文本和背景颜色的对比度的合理水平是很重要的，因而你不会希望颜色太暗。添加以下属性到 CSS 规则内。

```
.alert{background-color: #FFFFCC;}
```

警报框现在应当如图 3 所示。

**Alert!** The text of our alert goes here. Adding the background color really helps our alert stand out. You may have noticed that I've also added a small amount of space between the sides of the alert box and the text.

图 3：添加了背景颜色的警报框。

### 应用背景图像

现在让我们将图像添加至这个警报。背景图像的路径必须加 `url()`，如下面的代码所示。将下面高亮的代码行添加至 CSS 规则中。

```
.alert{
background-color: #FFFFCC;
background-image: url(alert.png);
}
```

现在，警报框将如图 4 所示。



图 4：已添加背景图像，但铺设效果看上去不好。

记住，每个背景属性都有一个默认值——如果你未指定属性值，将应用默认值。当然你会发现该图像平铺在了我们的整个警报上，很像是厨房地板上的镶嵌瓷砖。为什么不去掉这个呢？背景图像默认设为水平和垂直平铺。平铺背景对于占满屏幕或特定 HTML 元素的渐变和花纹特别有用，不过此例不需要实现这个效果。

#### 控制背景再现

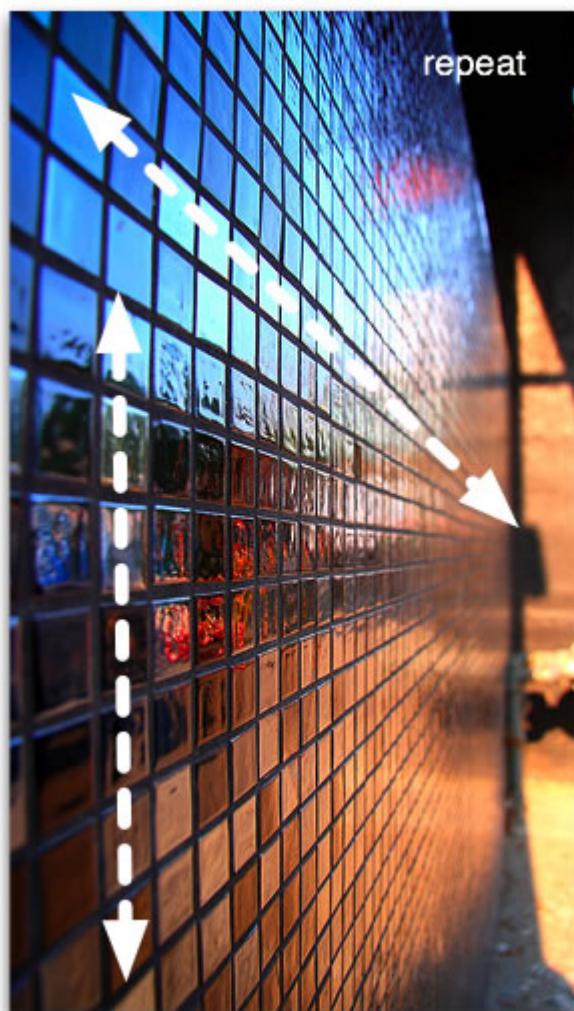


图 5：与我们的背景图像非常相似，这些瓷砖向水平和垂直两个方向平铺。

读规范的确很吓人，但在你钻研无数的浏览器的差异之前，规范真的是一个好去处，可以让你了解 CSS 应有的效果。请看一下 W3C 规范的颜色和背景部分，找出当你不想让某一背景图像平铺时应该使用的关键字。我们将在下面的例子中用到它。

找到了吗？注意，在该规范中有一章内容涵盖了每个背景属性，包括 `background-repeat`。在**Value**下，你将看到所有可能的选择，包括：`repeat`、`repeat-x`、`repeat-y`、`no-repeat` 和 `inherit`。默认情况下（最初设置）背景图像设为平铺。不指定方向，是指图像既有水平平铺，又有垂直平铺。你很可能猜测 `no-repeat` 就是你要找的防止图像在两个方向上平铺的值。将以下高亮代码添加至 CSS 规则。

```
.alert{
background-color: #FFFFCC;
background-image: url(alert.png);
background-repeat: no-repeat;
}
```

警报框现在如图 6 所示。



**Alert!** The text of our alert goes here. Phew! That's getting better.

图 6：警报框带有一个背景图片（无铺设）。

另外，你可以选择在两种方向上平铺（就像厨房瓷砖那样），或者两种方向上都没有平铺。渐变通常用水平或垂直平铺（参见图 7）。你不必知道 HTML 元素的大小，只需从渐变图像上切下一片，将其设置为在你想要的方向上平铺——x 轴水平，y 轴垂直两种方向。花纹通常在两种方向上平铺，而图标通常不平铺。在稍后的示例中，你将进一步了解 `background-repeat`。



图 7: 此例中的黄绿色瓷砖仅在水平方向上平铺。

我们来看一个我的网站中的实例——请看图 8。

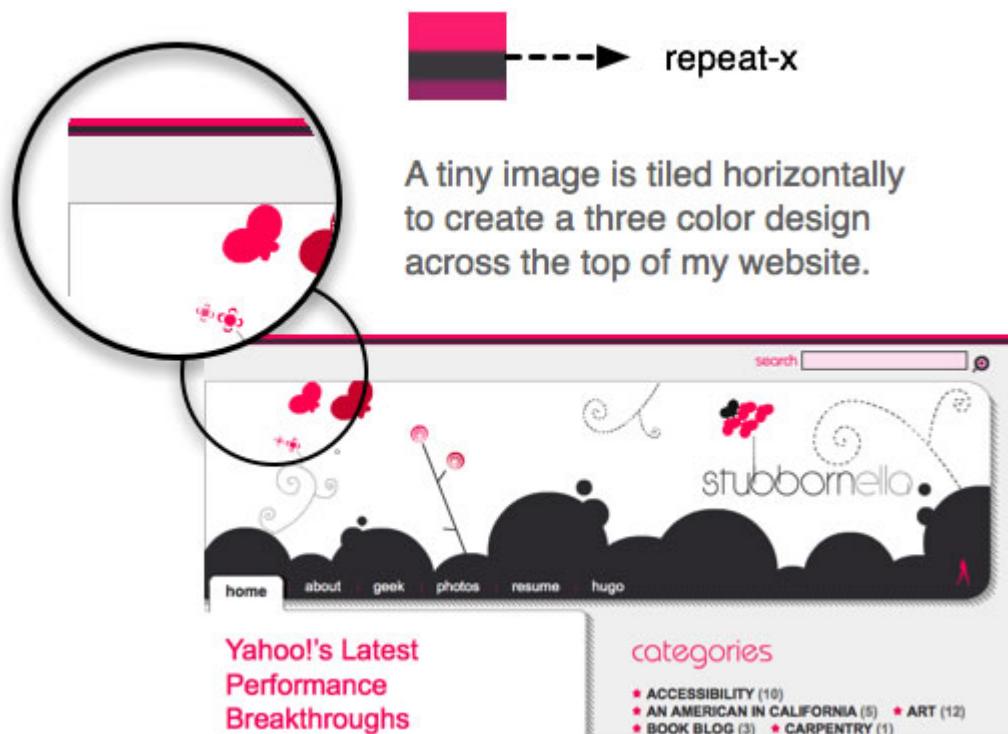


图 8：我自己的网站中的一个平铺实例。

我用于添加此装饰效果的 CSS 相对简单。我使用 `repeat-x` 使背景水平平铺：

```
body{background-repeat: repeat-x}
```

## 固定

`attachment` 使你可以指定用户滚动页面时背景的工作方式。默认行为是 `scroll`，它使背景图像随内容滚动。

而将 `background-attachment` 设为 `fixed` 则使元素固定在浏览器窗口，因此当背景图像固定到的元素内的内容滚动时，背景图像会保持不动。这会带来奇怪的效果，这一效果仅当你在其固定到的 HTML 元素上滚动鼠标时可以看到。W3C 用它标注其规范的状态，例如 [左上方的“W3C 候选推荐标准”图片](#)。滚动页面，图像还是位于左上方。图像被固定至 `body` 元素，因而总是可见的。

这一步对我们的演示不起什么作用，因为默认情况下浏览器将背景图像设定成了滚动，但我们还是要将其添加至代码以便你可以看到该属性的使用方式。将高亮代码添加至 CSS 规则：

```
.alert{
background-color: #FFFFCC;
background-image: url(alert.png);
background-repeat: no-repeat;
background-attachment: scroll;
}
```

如图 9 所示，警报框的显示与之前没有太大差别。



**Alert!** The text of our alert goes here. Our alert box is looking pretty good now, but it is still a little strange because the tiny background image is stuck to the upper left hand corner of our box.

图 9：没有太大差别。

## 对图像进行定位

定位是一种精细调整，使你可以精确地将背景图像水平和垂直地放在你想要的位置。这一属性采用 `top`、`center`、`right`、`100%`、`-10%`、`50px` 和 `-30em` 等关键字和数值。

图 10 显示的是一些属性值，当你要将你的背景图像放置在不同位置上的时候就会用到这些值。



图 10：各种使用关键字、百分数和像素的背景定位示例。

下面我们来定位背景图像。你想让其位于左上方，但不碰到边缘，因此你需要将其从顶部和左侧偏移 10 个像素，这可通过向 CSS 规则添加以下高亮代码来实现。现在你也做一下。

```
.alert{
background-color: #FFFFCC;
background-image: url(alert.png);
background-repeat: no-repeat;
background-attachment: scroll;
background-position: 10px 10px;}
```

```
}
```

第一个值是水平偏移量，第二个是垂直偏移量。在此例中，他们是相同的。现在，你的警报框应如图 11 所示。



**Alert!** The text of our alert goes here. Note that we include the word "alert" too so that we have a text equivalent for all this visual goodness.

图 11：使用定位功能放置背景图像。

提示：要么用关键字要么用数值——如果你同时使用两者，老式浏览器可能会忽略你的声明。使用关键字 `right` 和 `bottom` 效果相同 —— 分别是水平或垂直移动 100%。

### 像专业人员那样使用缩写集中表述

正如你所知道的，某些 CSS 属性可以组合在一起。`Background` 以及其所有子属性都包含在其中。迄今为止，我们编写的 CSS 代码缩写形式如下所示：

```
.alert{background: #FFFFCC url(alert.png) no-repeat scroll 10px 10px;}
```

提示：在组合 `background` 的子属性时，通常将这些属性按照以下顺序书写（这对跨浏览器的兼容性以及样式表的组织和维护都很重要）：

1. color
2. image
3. repeat
4. attachment
5. horizontal position
6. vertical position

用上面的缩写重置以前的 CSS 代码，你会看到如图 12 所示的结果：

 **Alert!** The text of our alert message goes here. Our final alert box is looking pretty slick, the CSS code is lean and efficient.

图 12: 缩写效果立竿见影!

### 用代码来做试验

记住 CSS 所有细微差别的最好途径就是亲手尝试——改变示例中的一些属性，然后看一下效果。将 `background-position` 设置为 `100% 100%`，然后你会注意到所得到的效果与使用关键字 `right` 和 `bottom` 的效果是相同的。那么如果将它改变为 `-5px 0` 又是什么效果呢？你觉得为什么会看不到完整的图片了呢？

### 质量测试

测试对于提供更好的用户体验来讲是很重要的。由于具体配置的差异，即使网站在你的电脑上看起来效果很好，也并不意味着在每个人的电脑上都有同样的效果。测试警示框的时候，你应该遵循以下几条最基本的步骤：

- 增加或者减少框内的文字量
- 在你的浏览器里增加文字大小至少两个字号。用 `ems` 定位图片是不是看其来效果要好？然后，当你增加文字大小的时候发生了什么？
- 在其它元素里应用这个 `class` 警示，如 `div`、`p`、`ul`、`strong` 或 `em`。你需要改变哪些地方才能不必专门使用 `class`？
- 在 `div` 警示中包含若干段落和一个列表——会依然有效吗？
- 确认警示在一级浏览器（也称为 A 级）上可见。我的建议是在好的浏览器上编码，一旦有效就在 Internet Explorer 应用。

严格的测试是学习编写 CSS 的一部分，学习越认真，你就掌握的越快

### Sprites (子画面)

用户们都会苛求完美。他们希望你的网站是吸引人的，是交互式的，并且希望浏览速度要快，当然，在某种程度上包含大量 CSS 背景图片会大大减慢你的网速，你发出的 HTTP 请求越多，你的网站速度就会越慢（HTTP 请求是指当你的网站连接到某个网站上并要求服务器端将该网站的其它资源发送过来——例如一个 CSS 文件或者图片时，每一个额外的请求就意味着该网站更长的加载时间）。要解决这个限制，你可以将相关图标结合到一张图片上，即众所周

知的 CSS 图像子画面技术。`background-position` 属性允许你将图片放置到合适的位置，以便图标通过应用了 CSS 图像子画面技术的 HTML 元素的窗口显示出来。

从图 13 可以看出，为了在 HTML 窗口显示地球图标，可以用 `left top` 放置图片。为了移动图片位置使警报图标显式出来，背景图片位置需要改为 `-80px 0`。水平方向的负值可以把图片拉到左侧。

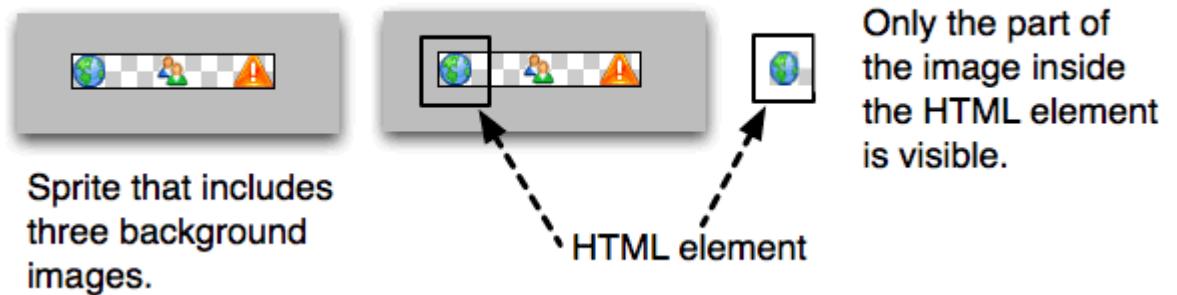


图 13: 使用 CSS 图像子画面技术减少 HTTP 请求

注意：如果你使用负背景定位值，就算你指定了 `no-repeat` 命令，Safari 也会平铺你的图片。这一点在你开始用背景图片创建更复杂布局的时候就要记住。

#### 复杂的子画面及背景图片示例

我们来看一下怎么利用 CSS 图像子画面技术来取得更好的效果。假如我们的平面设计师传给我们一个新的模板，这个模板是一个博客主页上的链接列表，链接到博主的 LinkedIn 资料、RSS 文件、照片以及书签等内容。看一下每个链接，我们看到链接是从图片中间的白色过渡到顶部和底部的灰色，而更复杂的是，设计者要求访客的鼠标悬停在上面的时候可以变成没有颜色变化曲线的纯白色，如图 14 所示：

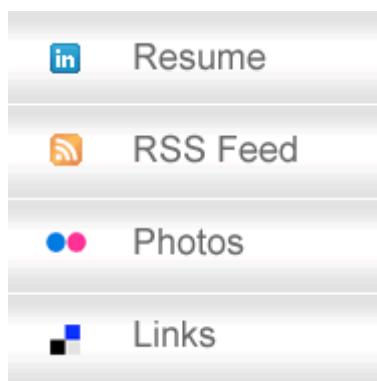


图 14: 新设计模板

标识中应通过 `img` 元素来包括 Logo，尽管如此，使用 CSS 图像子画面技术仍然是一个

使子画面下载更快的好办法，这样会使下载速度像只有一张图片那样快（而非 4 张），并且它简化了你的 HTML，减少了需要的标记数量。

## 创建子画面

首先的步骤就是剪切 4 个 logo 然后创建子画面集，如图 15 所示：



图 15：子画面集

你同时也需要剪切 1px 宽的渐变切片。为了让大家看清楚，我剪切的稍微大一点，但是你只需剪切 1px 大小就可以。如图 16：



图 16 渐变背景切片

该列表的 HTML 是一个包含了链接的无序列表，注意链接里的空的 span 元素，不要事先就固定包含文本的元素的 height 和 width，这很重要 —— 毕竟你并不知道文本到底有多大。如果文本翻译成德语会怎样？你可以使用额外的 span 来显示这些 logo。你可能不想用额外的非语义标记来显示 logo，因为那样会使你的 HTML 看起来很杂乱。这种情况下需要使用一个大的子画面并且在图标之间保留空白。但是要知道这样会使网速慢的用户的链接速度更慢，特别是使用手机上网的用户。列表的代码如下 —— 将其添加到 HTML 模板中：

```
<ul class="navigation">
 <li id="resume">
 Resume

 <li id="rss">
 RSS Feed

 <li id="photos">
 Photos

```

```
<li id="links">
 /span>Links


```

CSS 综合利用这两种背景图片。首先，看一下背景图层的图片，有三个有趣的现象值得注意：

1. 首先是图片在水平方向的平铺 (`repeat-x`)。这也是为什么我们可以使那么小的图片散布到整个列表的原因。
2. 其次是图片在垂直方向上的居中。你想让突出的白色部分出现在列表的中间位置，因此应该将背景定位设为 `left center`。
3. 1. 最后一点，在 CSS 中我应用了与渐变图片的灰色部分相同的背景色，这样，即使元素增大，也不会破坏整体效果。要了解更多这方面的技巧，我建议大家读读 Dan Cederholm 的 *Bulletproof Web Design* 这本书。

添加以下 CSS 到一个新的 CSS 文件，然后将其链接到 HTML 文件中：

```
.navigation, .navigation li {
 margin:0;
 padding:0;
}

.navigation li {
 border-top: 1px solid white;
 list-style-type:none
}

.navigation li a {
 background: #E2E2E2 url(sprite_gradient_bkg.jpg) repeat-x left center;
 padding:20px;
```

```

display:block;

font-family: Arial, Helvetica, sans-serif;

color:#333;

font-size:18px;

text-decoration:none

}

/* hover effects */

.navigation li a:hover, .navigation li a:focus{

background: transparent none;

}

}

```

最后一行意味着当用户通过鼠标悬停其上或者通过键盘对其聚焦的时候元素应无背景图片或者背景色。也许你想知道为什么我用了背景属性而非列表项？答案是 Internet Explorer 6 及其以前的版本在非链接元素上并不支持像 `hover` 这样的伪类。我的目的是使它适应这个限制。

下一步你可以为那些小 logo 创建 CSS 了。像往常一样，你可以从为导航模块里所有的 `span` 元素定义最常见的情况开始。这里你可以定义图片为所有 `span`、所有平铺和背景位置可用（每个都不一样，因此我们只说明第一个），你可以为其使用缩写。注意我用简短的语言来使用 CSS 命令将代码划分段。将下列代码添加到 CSS 文档的底部。

```

/* general case */

.navigation span {

background:url(sprite_logo.gif) no-repeat left top;

height:15px;

width: 15px;

margin-right:20px;

display:-moz-inline-box;
}

```

```
display:inline-block;
vertical-align: middle;
}
```

能够驾驭最普通的情况之后，你现在就可以定义例外的情况，或者是定义每一个 logo 与其它 logo 的差异了。在本例中，需要做的 CSS 代码变更仅仅只是 background-position。每一个列表项都需要将图片向左拉至少 15px，因为，每个 logo 都有 15px 的宽度。将以下代码添加到 CSS 文件的末尾：

```
/* exceptions */

#rss span {
 background-position: -15px 0;
}

#photos span {
 background-position: -30px 0;
}

#links span {
 background-position: -45px 0;
}
```

这个例子最开始看上去有点吓人。将你的注意力保持在背景图像上。在本例中，我用负的像素值来向左拉背景图片，以使图片相关部分可见。正值推动背景图片向下和向右，负值则拉动图片向上和向左。

在已完成的示例里调整 `background position` 值，这样就可以更好地理解如何调整子画面的位置。

## 总结

现在你应该已经掌握了 CSS 背景图像，此外，你已经可以更加自如地阅读规范，这样如果你对某个属性还有所怀疑的话，就可以自行查找了。本文涵盖了背景颜色、背景图片、背景平

铺、背景固定以及背景位置。你也了解了开发人员为何使用 CSS 图像子画面技术，以及怎么使用这门高级的技能。

### 图片来源

- [The Tiles](#), 作者 DimsumDarren
- [little glass tiles](#), 作者 emdot

### 练习题

- 一个段落大小为 40\*80px，而你的背景图片是 60\*200px，你会看到整张图片还是部分图片？为什么？
  - 如果你想把图片放置在 `blockquote` 元素的左下角，请填写正确的数值

```
blockquote{background: yellow url(quote.png) no-repeat scroll ____ ____ ;}
```
- 如果你想在文档中每一个具有“question”类的 `h2` 元素应用渐变图案，你是否会用 `repeat-x`、`repeat-y`、`no-repeat` 或 `repeat` 来达到类似于下面示例中的效果？为什么？

## Example Heading Level 2

- 在第三个问题中，示例中的背景位置是什么？你如何创造性地使用背景色来保证背景可以扩展到任何高度？为什么这个很重要？
- 你可以用那些缩写来移除所有背景属性？
- CSS 图像子画面技术的目的是什么？

### 延伸阅读

- [Performance and HTTP requests on YDN](#)
- [CSS2 Specification—Colors and Backgrounds](#)
- [CSS Sprites—Image Slicing's Kiss of Death](#)
- [Bulletproof Web Design](#)——我最喜欢的书
- 上一篇：[CSS 布局模型——boxes、borderes、margins、padding](#)
- 下一篇：[样式列表和链接](#)

## 作者简介



Nicole Sullivan, 著名的 CSS 应用大师, 现居于旧金山, 她的职业生涯始于 2000 年, 那时她未来的丈夫 (后就职于 W3C) 告诉她如果她的网站通不过验证他晚上就无法入眠。她就想最好弄清楚这个“验证”都是些什么东西, 于是从此她就对各种标准产生了异乎寻常的热爱。

随着她对网站性能以及大型网站的了解日益增长, 她后来从事于网上交易行业, 为许多欧洲乃至世界知名的品牌建立 CSS 框架解决方案, 如 FR、Club Med、SNCF、La Poste、FNAC、Accor Hotels、以及 Renault 等。

Nicole 现在在 Yahoo! 的网站性能优化部门工作, 她的工作涉及研究和宣传网站性能优化方法以及创建 Yslow 之类实用工具, 以帮助其它前端搜索引擎创建更好的网站。她将关于标准的文章, 她的宠物狗, 她的感情, 都用面向对象的 CSS 写在了她的网站 [www.stubbornella.org](http://www.stubbornella.org) 上。

## 32. 样式列表和链接

Posted 12/15/2008 - 16:57 by Lewis

- 网络标准教程

作者: Ben Buchanan · 2008 年 9 月 26 日

- 上一篇: [CSS 背景图片](#)
- 下一篇: [样式表格](#)
- 目录

### 序言

就设计而言, 网页上的许多元素是比较“宽松”的——即使不十分精确也没有大碍。但是对于列表和链接来说就不行了——如果列表和链接没有处理好, 将可能对网站的用户造成严重的影响。

尤其是链接, 对于链接会有一些重要的样式要求和用户期望。蹩脚的链接设计会毁了整个网站的用户体验, 因为人们不得不停下来去想到底该点击哪里。在最糟的情况下, 用户甚至可能弄不清楚到底网页上哪一处才是真正的链接。

在本文中我们将会学习创建良好的列表和链接样式所需的核心技术。本文也会讨论一些避免列表和链接上易犯的重大错误的方法, 并制作出跨浏览器且方便残疾用户使用的最终效果。

本文中用到了许多示例, 可以[下载列表与链接示例](#)照着学习。

本篇文章的内容目录如下:

- 样式化列表
  - 基本项目符号和编号
  - 使用图片的自定义项目符号
  - 列表边距与填充距
    - 无序列表

- 有序列表
- 那么，该怎么做呢？
- 使用列表样式位置
- 对于定义表来说是怎样的呢？
- 嵌套列表
- 水平列表
- 伪列布局
  - 老式浏览器
- 列表结语
  - 样式化链接
  - 了解链接状态
  - 浏览器的演变是如何影响用户期望的
  - 用户期望
  - 谨慎地使用颜色
  - 言归正传：CSS
    - 按照正确的顺序来样式化链接状态
    - 控制缺省值
      - 加下划线
      - 画轮廓
  - 示例：再现Netscape默认风格
    - 利用下边框来做伪下划线
    - 不依赖于颜色的样式
  - 链接图标
    - 全部用起来——一个简单的导航菜单
    - 总结
    - 练习题
    - 延伸阅读

## 样式化列表

首先，我会带你领略利用 CSS 样式化列表的基本原理，在这之前我们先来看一些稍微复杂一点的技术。

## 基本项目符号与数字

创建列表样式的时候要考虑的最基本的问题是你想用什么类型的项目符号或编号方式。你也可以将项目符号和编号完全删掉。就如你们在 [HTML列表](#) 一文中学到的那样，对于这一点有许多可用选项，对该项的设置用的是 `list-style-type` 属性。

例如，将你的站点上所有无序列表的项目符号都设置成实心方块，可以用下面的 CSS：

```
ul li {
 list-style-type: square;
}
```

这将会产生如图 1 所示的效果：

- List item
- List item
- List item

图 1：带方形项目符号的无序列表。

一些常见的列表类型如图 2 所示：

## Unordered lists

### Disc

- First item
- Second item
- Third item

### Square

- First item
- Second item
- Third item

### Circle

- First item
- Second item
- Third item

### None - no bullets

- First item
- Second item
- Third item

## Ordered lists

### Decimal

1. First item
2. Second item
3. Third item

### Decimal with leading zeros

01. First item
02. Second item
03. Third item

### Lowercase ascii letters

- a. First item
- b. Second item
- c. Third item

### Lowercase roman numerals

- i. First item
- ii. Second item
- iii. Third item

图 2：常见列表类型。

基本列表选项示例中列举了更多可供选择的类型。

注意，项目符号和编号是由 `li` 标签所设置或继承的 `color` 属性来渲染的。如果想要使项目符号的颜色与文本颜色不一样的话，只有用图片来代替项目符号，或者在列表项中用其它元素围绕该符号（当所有列表项都是链接的时候这样会比较简单）。

### 用图片自定义项目符号

项目符号的标准设置对于基本的内容来说已经足够了，然而将项目符号替换为自定义图片是一种常见的设计要求。

CSS 规范中包括了 `list-style-image` 属性，用以添加自定义列表图片。然而，该属性的背景图片位置选项有限，而且在有些情况下根本无法在 IE 中显示。所以直接在列表项上设置一张背景图片就成了非常普遍的做法。

我们假定你有一列 RSS 订阅，想将项目符号改成标准的橙色 RSS 图标。我们为这个列表定义一个“rss”类，以便与其它列表区别开来。

```
<ul class="rss">

 News
 Sport
 Weather

 Business
 Entertainment
 Funny News


```

首先我们将 `list-style-type` 设置为无，并清空边距和填充距。然后，给每个列表项直接加一张背景图片，并加上左侧填充距，将文本挪一下地方好让图片露出来，还有加下方填充距以使列表项产生间距。

```
.rss {
 margin: 0;
 padding: 0;
```

```

 list-style-type: none;
 }

.rss li {
 background: #fff url("icon-rssfeed.gif") 0 3px no-repeat;
 padding: 0 0 5px 15px;
}

```

这样生成的列表将会以 RSS 图片取代项目符号，如图 3 所示：

-  [News](#)
-  [Sport](#)
-  [Weather](#)
-  [Business](#)
-  [Entertainment](#)
-  [Funny News](#)

图 3：带图片项目符号的列表。

注意背景图片是以像素来实现精确定位的。你也可以使用%，em 或关键字，视你要创建的布局而定。当你的设计特征包含一些可能导致某个列表项换行的内容时，应当仔细一点——如果你的背景设置为垂直居中或 50% 的话，看起来可能会很奇怪，就像图 4 那样：

-  [News](#)
-  [Sport](#)
-  [Weather](#) with a long description to cause the list to wrap to new lines. This will show the way the vertically-centred bullet will sit in the middle of the text, instead of being centred to the first line of text.
-  [Business](#)
-  [Entertainment](#)
-  [Funny News](#)

图 4：多行列表项项目符号图片的垂直居中演示。

通过将图片设置在列表项的顶端，就可以维持项目符号的默认风格（即项目符号位于首行）——参见图 5：

 [News](#)

 [Sport](#)

 [Weather](#) with a long description to cause the list to wrap to new lines. This will show the way the vertically-centred bullet will sit in the middle of the text, instead of being centred to the first line of text.

 [Business](#)

 [Entertainment](#)

 [Funny News](#)

图 5: 多行列表项项目符号图片的顶端对齐演示。

#### 列表边距和填充距

巧用边距和填充距可以使列表看起来更精炼、更专业，但你要知道自己到底要做什么，并且要记住不同类型的列表所对应的情况是不同的。

#### 无序列表

你可能很快就会注意到列表在其默认样式下比默认样式的段落缩进得更多——参见图 6:

**A paragraph for reference.**

- First item
- Second item
- Third item

**A paragraph for reference.**

图 6: 默认风格的列表是左端缩进的。

如果想使无序列表项与其它页面内容的左对齐点相同的话，你需要设置一些样式来将缩进调节成你喜欢的样子。不同的浏览器对设置的要求也不同——有的需要清除边距，还有的需要清除填充距。因此，为了适应所有的浏览器，我们将二者都重设一下：

```
ul {
 margin: 0;
 padding: 0;
}
```

这段代码可能达不到你想要的效果，因为它会使文本左对齐，但项目符号会比文本的位置更靠左，如图 7 所示：

A paragraph for reference.

- First item
- Second item
- Third item

A paragraph for reference.

图 7：项目符号的位置比文本更靠左。

因此，为了使项目符号在左侧对齐，现在你可以设置列表项的边距，来使它们排成一队：

```
ul {
 margin-left: 0;
 padding-left: 0;
}

ul li {
 margin-left: 1em;
}
```

...这时候你仍会发现在不同的浏览器上会有像素级别的差异，但页面效果已基本上达到了尽量一致——参见图 8：

A paragraph for reference.

- First item
- Second item
- Third item

A paragraph for reference.

图 8：项目符号与周围段落一同定位。

## 有序列表

现在你要考虑的是在应用到有序列表中时如何处理同样的问题。由于有序列表中的数字标号是根据数字最大的那个列表项目来对齐的，所以在有序列表中的应用会比较复杂一些。比如说，假如  
有 10 个列表项，十进制数字的定位会考虑到两位数“10”的列表项，如图 9 所示：

1. First item
2. Second item
3. Third item
4. Fourth item
5. Fifth item
6. Sixth item
7. Seventh item
8. Eighth item
9. Ninth item
10. Tenth item

图 9: 1—9 项的数字标号会在前面进行填充以便与第 10 项右对齐。

所以，除了使用这个语句 `list-style-type: decimal-leading-zero;` 来设置列表，没有其它任何办法来使列表项与周围文字的左对齐位置一致。该语句可以掩盖空缺部分，如图 10 所示：

01. First item
02. Second item
03. Third item
04. Fourth item
05. Fifth item
06. Sixth item
07. Seventh item
08. Eighth item
09. Ninth item
10. Tenth item

图 10: 开头的 0 填补了 1-9 项的间隙。

更常见的做法是容忍这种间隙差异。然而这就意味着你的有序列表和无序列表的标记符号不能顺利地保持左对齐一致了。你只能对列表中的文本进行排列。

```
ul, ol {
 margin-left: 0;
 padding-left: 0;
}

li {
 margin-left: 2em;
}
```

你至少需要 `2em` 的左边距来满足有序和无序列表的需要。注意图 11 中两种列表中列表项文本的排列方式：

A paragraph for reference.

- First item
- Second item
- Third item
- Fourth item
- Fifth item
- Sixth item
- Seventh item
- Eighth item
- Ninth item
- Tenth item

A paragraph for reference.

1. First item
2. Second item
3. Third item
4. Fourth item
5. Fifth item
6. Sixth item
7. Seventh item
8. Eighth item
9. Ninth item
10. Tenth item

A paragraph for reference.

图 11：在有序和无序列表中文本都排成一列。

那么，该怎么办呢？

大致说来，你有三种选择：

1. 忍受列表和其标记符号的默认定位
2. 对文本进行显式排列
3. 在 ul 和 ol 中设置另外一种样式。

方法无所谓“对”还是“错”，而且对一般内容中的列表一概直接使用默认样式也是一种常见的做法。

#### 利用 **list-style-position**

如果想使多行列表项的文本占据列表标记符号的下方位置，应当将 `list-style-position` 属性设为 `inside`，这样就会产生图 12 所示的效果：

- First item - Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus quis ipsum. Quisque eget tortor mattis nunc laoreet tempus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vivamus quam. In varius justo ultricies dolor. Duis nec pede sed dui vehicula tincidunt.
- Second item - Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus quis ipsum. Quisque eget tortor mattis nunc laoreet tempus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vivamus quam. In varius justo ultricies dolor. Duis nec pede sed dui vehicula tincidunt.
- Third item - Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus quis ipsum. Quisque eget tortor mattis nunc laoreet tempus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vivamus quam. In varius justo ultricies dolor. Duis nec pede sed dui vehicula tincidunt.

图 12：列表位置“inside”设置会使文本占据标号下方的位置，而不是与缩进的文本对齐。

设置标记符号位置为内侧并不是一种特别流行的样式。默认的 `list-style-position` 是将标记符号位置设置为 `outside`，由此产生的效果将会在本文的其它地方谈到。

对于定义列表来说是怎样的呢？

一般来说，除了设置 `dt` 样式（一般是粗体文本）以及调节定义内容的缩进之外，定义列表并没有需要特别留心的地方。

```
dt {
 font-weight: bold;
}

dd {
 margin-left: 2em;
}
```

上面的代码能使定义列表产生清晰简明的风格，如图 13 所示：

### **Term**

Definition of the term.

### **Term**

Definition of the term.

### **Term**

Definition of the term.

图 13：一张样式简单的定义表。

尽管定义列表可由浮动和位置控制来调整，但由于它们比较易变所以一般还是保持简单比较好。

就这样定义列表已经很令人满意了，如果再稍微加工一下，使定义项更突出，并且使定义内容的

缩进更精确的话，就更好了。

### 嵌套列表

在 [HTML](#) 列表一文中你们学习了有关嵌套式列表的知识。当你创建 [CSS](#) 时要注意保持设计线索明晰，以便显示嵌套列表与其上级列表之间的关系。为了达到这个目的，目前最常见的办法是缩进嵌套列表项——这其实是跨浏览器的默认设置。

如果你自己设置了列表缩进，你的基础设置的效果会层层递推。比如说，看看下面这段 [CSS](#) 代码：

```
ul, ol {
 margin-left: 0;
 padding-left: 0;
}

li {
 margin-left: 2em;
}
```

这一系列列表项中的每个子列表项都会从其上级列表项继承 `margin` 值，除了它们自己的缩进值之外还会在顶端另加上 `2em` 的缩进。因此顶层列表项(即没有上级项的列表项)的左边距为 `2em`，它的子列表项会从其上级列表项继承 `2em` 的边距，然后再加上 `2em`，总共就有 `4em` 的左边距...以此类推下去。

### 水平列表

对于列表来说，最普遍的一种变化就是生成水平列表——即，使列表项的呈现一个挨着一个的效果，而不是一个接着一个。这是网站导航上最常见的技巧。我们用一下[导航菜单](#)一文中的一个示例（见图 14）：

- [Home](#)
- [About Us](#)
- [Our Clients](#)
- [Our Products](#)
- [Our Services](#)
- [Contact Us](#)

图 14：一个简单的列表。

我们来将它转换为水平列表，如图 15 所示：

[Home](#) [About Us](#) [Our Clients](#) [Our Products](#) [Our Services](#) [Contact Us](#)

图 15: 一个简单的水平列表。

为了达到这种效果，我们需要对列表做三件事：

1. 从`<ul>`标签中清空 `margin` 和 `padding`
2. 将列表项设置为：`display: inline;`
3. 在列表项右侧留一些间隔，以免它们挤到一起

本例中该列表的 ID 为"mainmenu"，故我们可以将其作为一个上下文选择器，以确保我们只改动了我们要改的那个列表。CSS 代码为：

```
#mainmenu {
 margin: 0;
 padding: 0;
}

#mainmenu li {
 display: inline;
 padding: 0 1em 0 0;
}
```

在这个简单的例子中，将列表项设置为`display: inline;`就可以了，记住，使用`float: left;`也可以达到类似的效果。在本课程中稍后你将学习更多关于浮动的知识。

### 伪列布局

早先我们创建了一个RSS 订阅列表。现在让我们想象这个列表已经放在你网站的侧边栏中。设计师想让这个列表以两列的形式出现，并且在整个列表外围加上一个边框，如图 16 所示：(参见 [styling-lists-example-columns.html](#))



图 16: 一张两列的信息源列表，每个项目符号都是 RSS 图标。

我们假设该列表位于设置宽度与边框的`<div>`标签中。初步的列表外观如图 17 所示：(参见 [styling-lists-example-image-bullet.html](#))

- [News](#)
- [Sport](#)
- [Weather](#)
- [Business](#)
- [Entertainment](#)
- [Funny News](#)

图 17：边框内未经样式处理过的列表。

首先，RSS 图标之前已经演示过了；接下来在顶端，左侧和右侧各加 5px 边距：

```
.rss {
 margin: 5px 5px 0 5px;
 padding: 0;
}

.rss li {
 list-style-type: none;
 background: #fff url("icon-rssfeed.gif") 0 3px no-repeat;
 padding: 0 0 5px 15px;
 display:-moz-inline-box;
}
```

要注意 `display:-moz-inline-box;` 是用来保证在 Firefox 中显示正确而添加的。

我们不用设置下边距，因为最末一个列表项是用填充距来添加适当的间距的，如图 18 所示：

-  [News](#)
-  [Sport](#)
-  [Weather](#)
-  [Business](#)
-  [Entertainment](#)
-  [Funny News](#)

图 18：中间效果——现在间距和图标都处理好了。

现在将列表项设为 `display: inline-block;`，其宽度设为 40%，右边距为 2%（也可以用像素宽度）。还要在 `<ul>` 标签中显式设置宽度为 100%，以确保列表的换行和大小排列正确：

```

.rss {
 margin: 5px 5px 0 5px;
 padding: 0;
 width: 100%;
}

.rss li {
 display: inline-block;
 width: 40%;
 margin: 0 2% 0 0;
 list-style-type: none;
 background: #fff url("icon-rssfeed.gif") 0 3px no-repeat;
 padding: 0 0 5px 15px;
 display:-moz-inline-box;
}

```

在绝大多数的浏览器下，这样已经足够创建分栏效果，但在 IE 下你得显式设置列表项向左浮动。我们用一个条件样式来适应 IE7 及其之下（因为我们还不知道未来的浏览器将会怎样）的所有浏览器：

```

<!--[if lte IE 7]>

<style type="text/css">

.rss li {
 float: left;
}

</style>

<![endif]-->

```

现在我们想要的两栏效果就出来了，如图 19 所示：



图 19：完成后的列表。

## 老式浏览器

如果要求你为不支持内联区块的老式浏览器设计网页，就需要使列表项在所有浏览器下都向左浮动，并使用类似 清除无源标记的浮动容器一文中所描述的那种清除浮动。由于最新系列的浏览器版本都已经开始支持`inline-block`属性，所以除非你所面对的目标人群大多数都用firefox 2之类的老式浏览器，你应该是可以使用`inline-block`方法的。

## 列表总结

现在我们学完了列表的样式化选项和方法的核心内容。你可以以这些示例为基础，将它们结合起来，来创作大量的网页设计。由于列表经常与链接结合使用，我们下面来看看链接。

## 样式化链接

样式化链接带着点艺术形式的味道。现在对于链接有许多各式各样的要求，在制作美观的效果的同时全部满足这些要求是比较困难的。不过，假如你记住几个简单的规则，是可以做好的：

- 了解各种链接状态
- 不要离用户期望差得太远
- 谨慎使用颜色

如果你遵行上述原则，就可以制作出明晰又易用的链接。

### 了解链接状态

在样式化链接之前，你需要了解各种链接状态。一共有五种链接状态：未访问/默认，已访问，焦点，悬停和激活。

#### unvisited

链接在未被激活或未被访问过时的默认状态。

#### visited

用户已访问过的链接状态。

#### focus

链接获取焦点时的状态——例如当键盘用户的光标位于某个链接上时。注意：现在的IE不支持焦点状态，只能用`active`代替`focus`。

#### hover

用户的鼠标指针“悬停”在链接上，但还没有点击时的状态。

## active

用户激活链接时的状态——从字面上讲是在用户点击链接的那一段时间。在有的浏览器下也指在新窗口或标签中打开链接时的状态。

你可以为上述每个状态指定 CSS。每个状态都在告诉用户他们正在与链接进行互动。如果对 `focus`, `hover` 和 `active` 这几个状态存在疑问，你也可以直接以同样的方法样式化 `focus` 和 `hover`，因为它们的功能很相似，同样的链接样式也不至于引起混淆。然后你可以为 `active` 添加一些简单的变化，比如将文本设置为斜体。必要时你可以用同样的方法样式化这三种状态。

注意，上述这些状态并不是互斥的（尽管同一个链接实际上不可能同时既是未访问的又是已访问的）——然而一个链接很可能同时处于悬停，激活和已访问的状态。

## 浏览器的演变是如何影响用户期望的

为了更好地理解一些常见的关于链接的用户期望，了解一点网络的历史会很有帮助。

你可能听人们提起过链接的“Netscape 默认风格”；或者说到链接应该都是蓝色和紫色的。这让我们回想起网络的早年时期，那时浏览器设定了页面内容的颜色，而网页作者不能控制网页的渲染。

文本是黑色的；背景是灰色的；所有的链接都是加了下划线的。未访问链接是蓝色的，已访问链接是紫色的，激活的链接是红色的...这就是早期的网页。图 20 就是早期网页的一个图示。



图20: Mosaic 浏览器的屏幕截图。

这确实有点单调乏味，然而也是和谐的——而且它设定了用户期望的基准线。特别是，至今用户们仍希望下划线文本是一个链接。他们可能并不期望所有的链接都有下划线，但是他们肯定期望下划线文本是可点击的。最好不要与这个用户期望相抵触。

一些网站仍然沿用着蓝色和紫色的链接；而在绝大多数浏览器下这些链接颜色也仍然是未经样式处理过的页面内容的默认风格。你可以一直走复古路线并继续使用这种颜色设置，而用户们在其它设置下也相当地得心应手——在一定的范围之内。

### 用户期望

用户对于链接的期望具有下面几条基本规则：

- 用户期望链接与其它非链接文本看起来不一样

- 当用户悬停或聚焦在链接上时，他们期望链接会有反应
- 用户期望在他们访问过某个链接过后，该链接会发生变化
- 用户期望同样功能的链接在风格上具有一致性以便他们明白该点击什么
- 用户期望下划线文本是链接——所以不要将下划线作任何其它用途

你应当时刻迎合这些基本规则，因为它们有助于你的用户迅速地识别及使用链接。你应该创建的样式是：让任何人都不用停下来去思索“到底哪里才是链接”？！

这些用户期望可以转化为简单的代码规则：

- 为所有链接状态设置样式
- 下划线只供链接使用

### 谨慎地使用颜色

在样式化链接时，注意连接状态的区别不能完全依赖于颜色。不是所有人都能看到同样的颜色（比如说患有色盲的人），所以你应当同时使用颜色与样式，比如各种下划线，图标或反向色。

还应当检查你的颜色选择是否有明显的对照物——利用类似 [色彩对比度分析器](#)（适用于PC机和Mac机）或 [Opera网站可访问性工具条](#)（二者都来自Paciello Group）之类的工具就可以轻易做到。

色彩对比度分析器（如图 21）可以用拾色器选取屏幕上的前景和背景颜色，并对它们的对比度作出简单的评价。

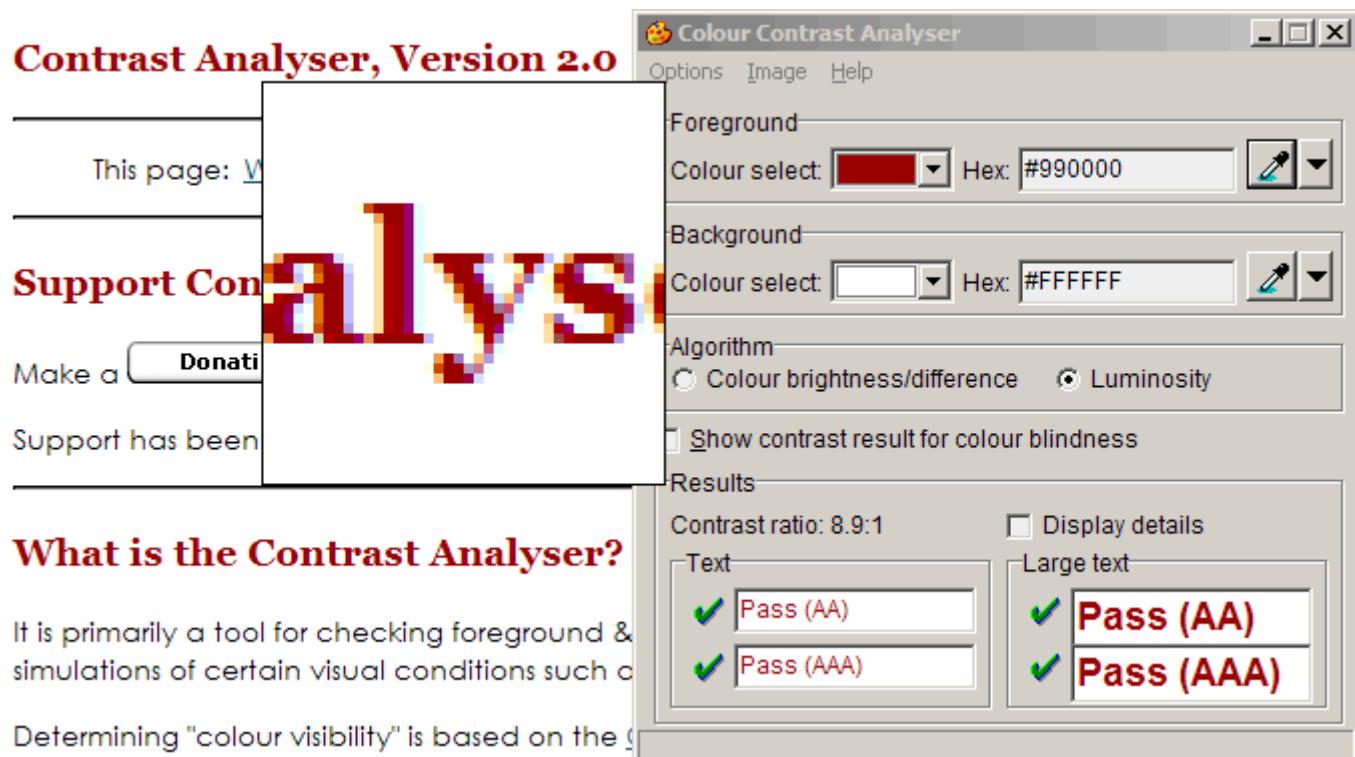


图21：使用中的色彩对比度分析器的屏幕截图

如果四个结果都表示通过，那么色彩就成功了。要记得检查所有的链接状态。你可能需要在“hex”框中手动输入颜色值来检测焦点，悬停和激活状态的色彩对比度。

#### 言归正传: CSS

现在你们明白了一些关于链接的基本法则，让我们来看看代码——这一部分给出了成功进行链接样式化所需的所有 CSS 的详细资料。

#### 按正确的顺序来样式化链接状态

首先，要注意如果你没有在样式表中将链接状态正确排序，那么设置就会互相覆盖，链接状态就不能奏效了。链接样式必须始终遵照下面的顺序：

1. link
2. visited
3. focus
4. hover
5. active

有一个常用的助记方法"Lord Vader's Former Handle, Anakin"可以帮助我们记住这个顺序。如果你不是星球大战的粉丝，我想你恐怕只能死记硬背，或者复制粘贴下面的代码块了！

还有一种流行的助记方法是"LoVe Fears HAte"，其中"Fears"代表"Focus"。

不同的链接状态是通过其“伪类”来实现样式的——`:link :visited :focus :hover :active`——将其加在链接对象选择符 `a` 后面。所以你的 CSS 代码应当这样开始：

```
a:link{}
a:visited{}
a:focus{}
a:hover{}
a:active{}
```

如果你想为所有链接的所有状态设置一条 CSS 规则，可以直接对 `a` 进行设计。不过要记得首先应置入通用法则以保持连接状态的顺序。

```
a {}
a:link{}
a:visited{}
a:focus{}
a:hover{}
a:active{}
```

这对于将替换默认的下划线为下边框是很有用的，下边框是获得更佳视觉效果的常见手法。

### 控制缺省值

在默认风格下，大多数浏览器都将链接设置为带有下划线，而焦点状态的链接则带有轮廓，如图 22 所示：



图22：从左到右依次是：Opera 9, Firefox 2 和 IE 7 的默认焦点样式。

如果你想将这些样式替换成别的什么，可以改变或禁用这些缺省值。

## Underlining

通过设置 `text-decoration` 属性，可以实现加下划线的效果。

```
a {
 text-decoration: underline;
}
```

你可以通过将该属性设置为 `none` 来禁用下划线。

```
a {
 text-decoration: none;
}
```

即使你想保存下划线风格，你也会发现禁用 `text-decoration` 并利用 `border-bottom` 来做一个伪下划线要更容易些。我们将通过下面的例子来感受一下。

### Outline轮廓

焦点轮廓是由 `outline` 属性来控制的。轮廓与边框几乎一样，但轮廓不占用页面空间，也不会在出现时导致页面重新排版（注意，IE7 及其之前版本的浏览器不支持轮廓）。最简便的设置轮廓的方法是使用简写的属性：

```
a:focus{
 outline: thick solid #000;
}
```

该例的渲染效果如图 23 所示：



图 23：粗黑轮廓线的渲染示例。

如果你不知道该怎样处理轮廓，可以直接使用浏览器默认风格。

### 例：再现 Netscape 默认风格

作为一个链接样式的简单示例，我们来再现一下 Netscape 默认风格的蓝色、紫色和红色链接。

我们会保留下划线，但将会用斜体字来标记激活状态。为了突出本例的效果，我们将字体放大并

将背景设为白色。

```
body {
 background: #fff;
 color: #000;
 font-size: 2em;
}

a {
 text-decoration: underline;
}

a:link{
 color: #0000CC;
}

a:visited{
 color: #6D006D;
}

a:focus{
 color: #CC0000;
}

a:hover{
 color: #CC0000;
}

a:active{
 color: #CC0000;
 font-style: italic;
```

```
}
```

这段代码的效果如图 24 所示：

## link, visited, focus, hover, active

图 24：再现 Netscape 默认风格。

利用下边框来做伪下划线

许多网页设计者都注意到下划线有点太粗了，而且会穿过小写字母的下半部分——更确切地说，就是下划线会穿过 g, j, p, q 和 y 这些字母的下端。如图 25 所示：

pygmy

图 25：下划线会穿过小写字母的下半部分。

我们假设为你设计网站的人同意，并且乐意让下划线变得细一点儿，不要接触到文字。为了达到这个要求，我们用边框来代替下划线，如图 26 所示：

pygmy

图 26：用边框来代替下划线可以使效果更美观。

首先，禁用所有链接状态的下划线，然后将下边框与各状态下的链接颜色相匹配：

```
body {
background: #fff;
color: #000;
font-size: 2em;
line-height: 2em;
}

a {
text-decoration: none;
}

a:link{
color: #00c;
```

```
border-bottom: 1px solid #00c;
}

a:visited{
 color: #6D006D;
 border-bottom: 1px solid #6D006D;
}

a:focus{
 color: #c00;
 border-bottom: 1px solid #c00;
}

a:hover{
 color: #c00;
 border-bottom: 1px solid #c00;
}

a:active{
 color: #c00;
 border-bottom: 1px solid #c00;
 font-style: italic;
}
```

效果如图 27 所示：

link, visited, focus, hover, active

图 27：伪下划线运行效果。

如果你使用伪边框方法的话，可以设置足够的 line-height 来避免下划线与下一行文字相冲突。

不依赖于颜色的样式

到目前为止的例子中，五个链接状态中有四个都是完全依靠颜色来区分的，我们应该开始进行下一步，改变已访问、焦点和悬停状态的下边框。我们来将已访问状态的下边框设为点线，而悬停和激活状态的下边框设为虚线：

```
body {
background: #fff;
color: #000;
font-size: 2em;
}

a {
text-decoration: none;
}

a:link{
color: #00c;
border-bottom: 1px solid #00c;
}

a:visited{
color: #6D006D;
border-bottom: 1px dotted #6D006D;
}

a:focus{
color: #c00;
border-bottom: 1px dashed #c00;
}

a:hover{
color: #c00;
```

```
border-bottom: 1px dashed #c00;
}

a:active{
color: #c00;
border-bottom: 1px solid #c00;
font-style: italic;
}
```

代码效果如图 28 所示：

## link, visited, focus, hover, active

图 28：改变各链接状态下的边框样式。

如果我们将焦点和悬停作为等价的样式状态的话，这就意味着链接状态的区分是不依赖于颜色的。即使是在黑白模式下浏览这些链接，你也能辨认各种链接状态，如图 29 所示：

## link, visited, focus, hover, active

图 29：即使在黑白模式下，链接状态也是可分辨的。

### 链接图标

有些网站用图标和标志来补充链接信息。例如，有的网站用箭头来表示某个链接是指向外部站点的；或者用一个记号来表示该链接已被访问过。

用背景图片就能很容易地实现这些效果，如图 30 所示：

## external link ↗, visited link ✓

图 30：带辨识图标的链接示例。

你可以在链接标签中加入 "external" 类，来给外部链接加上箭头图标：

```
external link
```

接下来，在样式表中为该类设置背景图片——记得要添加填充距，以容纳该图片。

```
a.external {

background: #fff url("icon-external.gif") center right no-repeat;

padding-right: 30px;
```

```
}
```

这段代码会使该图标出现在所有状态下的所有已访问链接上。如果你想将该图标设为仅出现在未访问的外部链接上，你可以在选择符中将各种类与链接状态伪类结合起来：

```
a.external:link{

background: #fff url("icon-external.gif") center right no-repeat;

padding-right: 30px;
}
```

将类与状态结合使用，大大发掘了链接的潜在创新可能。此外别忘了检查你的配色，从今以后，唯一能限制你的就是创造力的问题了。

### 全部用起来——一个简单的导航菜单

为了阐释一种综合应用链接和列表的方法，该示例的压缩包包括了一个简单的弹出式导航菜单，如图 31 所示。弹出式菜单是一种非常常见的导航系统。

## Home

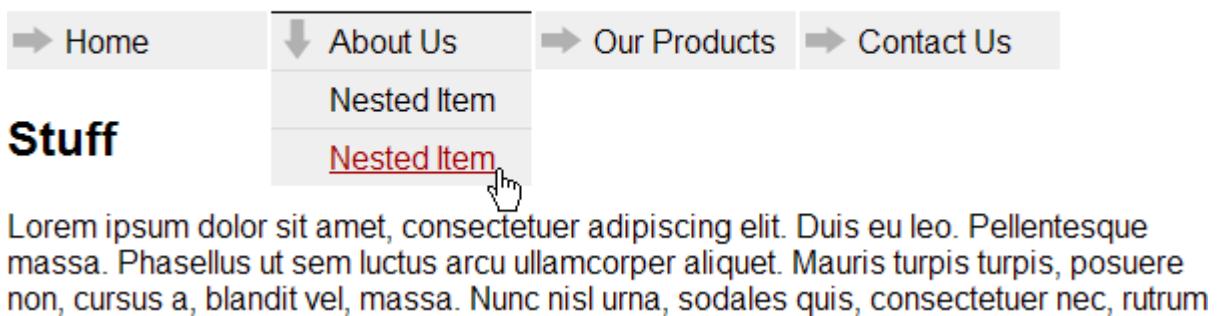


图 31：弹出式菜单示例的屏幕截图。

### 总结

充分地掌握列表和链接的样式化对于网站开发员来说是非常重要的，因为处处都要用到它们。创建网站导航时二者通常都是结合使用的，而明晰的链接样式对于任何网站的易用性来说都是十分关键的。不良的链接样式可能引起大家的严重混淆，甚至会导致某些用户无法使用网站。

### 练习题

- 你该如何在基础链接样式中做出选择，比如说，对于有序列表你是选实心方块还是罗马数字？
- 何谓图像拼合技术，为什么你要使用它？
- 为什么说色彩对比度很重要？如何保证你的链接选用的颜色是适宜的？

- 给各个链接状态设置样式时的正确顺序是什么？

## 延伸阅读

- WCAG Samurai Errata for WCAG 1.0, with specific reference to Guideline 2.  
*Don't rely on colour alone.*
- Type and Colour (a chapter from *Building Accessible Websites*, by Joe Clark)
- Juicy Studio: Highlighting Links
- Max Design—Simple, accessible external links
- Resource Center—Contrast Analyser 2.0 (Paciello Group)
- A List Apart: CSS Sprites: Image Slicing's Kiss of Death
- 上一篇：CSS 背景图片
- 下一篇：样式表格
- 目录

## 作者简介



Ben Buchanan 10 多年前就开始创作网页，那时他就读的学位课程其实并不是IT课程。他在公共机构（大学）和私营部门都曾工作过。他曾参与数家著名网站的改版，包括 澳大利亚人报的网站（The Australian）和前后三个版本的 格里菲斯大学的官方网站。目前他在 新闻数字媒体公司（News Digital Media）任前端架构师。他的博客地址为：the 200ok weblog。

### 33. 样式表格

Posted 12/15/2008 - 16:58 by Lewis

- 网络标准教程

作者: Ben Buchanan · 2008 年 9 月 26 日

- 上一篇: 样式列表和链接
- 下一篇: 样式表单
- 目录

## 序言

在目前的 web 开发中, 表格有时候似乎有点儿被误解了。人们非常在意“不要使用表格！”, 但往往忘记了这句话其实是在说“不要用表格来进行布局”。表格在其真正用途——显示表状数据方面是非常卓越的。所以, 了解如何妥善地对其进行样式化是很有意义的。

本教程侧重于如何高效地应用 CSS 来创建明晰易读的数据表风格。本文也会涉及到一些常见的表格设计要求。本文结构如下:

- 表格结构
- 基本原理
  - 表格和单元格宽度
  - 文本对齐
  - 边框
  - 填充距
  - 标题摆放
  - 背景
  - 利用条件样式来适应IE
- 常见的变化
  - 斑马式条纹
  - 不均匀列
  - 不完全网格
  - 内部网格
- 两个常见漏洞
  - 边框合并漏洞

- 边距/标题漏洞
- 总结
- 练习题
- 延伸阅读

下载本文中表格的代码示例可能会对你有所帮助，这样你就可以跟着本文的进度来练习了。

## 表格结构

在我们开始钻研 CSS 之前，先来了解一下创建明晰的样式所需的表格关键结构元素。

- 表格的表头
- 表格数据单元
- 表格标题

当网站用户浏览你的表格时，应该让他们能轻松看懂表格的结构。为了达到这个目的，最常见的办法是使用边框或背景颜色，或者二者兼用。

你并不是非得遵守这些样式惯例，但你应当确保 `th` 和 `td` 单元格之间有明显的差别；同样，`caption` 应当与表格明显相关，并与页面上的其它文本有显著的差别。

## 基本原理

看看下面这个未经样式处理的表格是如何提供信息的（下面这个例子跟你在 第 19 篇——HTML 表格中看到的那个例子是一样的）：

### Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Compiled in 2008 by Ms Jen			
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption

这张表格读是能读懂，但是得花点儿功夫才能弄明白到底在讲什么。下面我们给它加上一些样式来使它更易读。

## 表格和单元格宽度

首先我们要决定表格的宽度。浏览器的默认值等价于这个设置：`table { width: auto; }`，这会使表格宽度随内容的宽度而改变。一般来说，这样的效果会显得比较凌乱。

假设我们的表格处在一个宽 `600px` 的内容栏中。为了充分利用空间，我们将表格宽度设置为可用宽度的 `100%`。由于表格有四列，我们将每个表格单元的宽度都设为 `25%`：

```
table {
 width: 100%;
}

th, td {
 width: 25%;
}
```

实际上你可以仅在 `th` 上设置宽度，该值会设定所有列的宽度；但是这种做法并不是很严密。这个简单的样式的效果如图 1 所示：

Recent Major Volcanic Eruptions in the Pacific Northwest			
Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption

Compiled in 2008 by Ms Jen

图 1：简单设置了宽度的示例表。

现在单元格都是等宽的。稍后我们会做不等宽的单元格，不过现在我们还是继续向前推进。

## 文本对齐

现在表格还是有点难以阅读，为了让它整洁一点儿，我们来设置一下文本对齐——下面添加的代码可以使表头左对齐，以便与其下方的内容相匹配（默认状态下，浏览器会使表头居中对齐）：

```
table {
 width: 100%;
}

th {
 text-align: left;
}
```

```
th, td {
 width: 25%;
 text-align: left;
}
```

现在表格要整洁一点了，如图 2 所示：

Recent Major Volcanic Eruptions in the Pacific Northwest

<b>Volcano Name</b>	<b>Location</b>	<b>Last Major Eruption</b>	<b>Type of Eruption</b>
<b>Mt. Lassen</b>	California	1914-17	Explosive Eruption
<b>Mt. Hood</b>	Oregon	1790s	Pyroclastic flows and Mudflows
<b>Mt. St. Helens</b>	Washington	1980	Explosive Eruption

Compiled in 2008 by Ms Jen

图2：应用左对齐后的表格。

现在所有单元格都是垂直居中对齐的了。如果你愿意的话，可以将文本对齐到单元格顶部或底部，或任何你喜欢的 vertical-align 设置。下面新添加的代码是将文本对齐到顶部：

```
table {
 width: 100%;
}

th, td {
 width: 25%;
 text-align: left;
 vertical-align: top;
}
```

现在的表格外观如图 3 所示：

Recent Major Volcanic Eruptions in the Pacific Northwest			
Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption

Compiled in 2008 by Ms Jen

图3: 添加了垂直对齐的表格。

注意，所有的表头都处在单元格顶部，即便是占了两行的"Last Major Eruption"也是如此。

### 边框

这张表现在看齐来要美观一点了，但逐行阅读还是有点困难。是该设置一下边框来让表格更易读了。你需要为表格的每个部分设置边框，然后再来看各个边框该怎样合并。

为了说明边框的位置，图4标出了各种不同的边框: `table (solid black)`, `caption (solid grey)`, `th (dashed blue)`和 `td (dotted red)`:

Recent Major Volcanic Eruptions in the Pacific Northwest			
Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption

Compiled in 2008 by Ms Jen

图4: 表格内各种不同元素的边框演示。

注意，`table` 边框环绕在所有表头和数据单元格外面，并位于单元格和标题之间。`th` 和 `td`彼此之间留有间距，你也可以在默认状态下看到这种效果。

来看看另一种样式的表格——你可以用 `border` 属性为表格和单元格创建简单的黑色边框——可通过下面的新增代码来实现：

```
table {
 width: 100%;

 border: 1px solid #000;
}
```

```

th, td {
 width: 25%;

 text-align: left;
 vertical-align: top;

 border: 1px solid #000;
}

```

效果如图 4:

Recent Major Volcanic Eruptions in the Pacific Northwest

<b>Volcano Name</b>	<b>Location</b>	<b>Last Major Eruption</b>	<b>Type of Eruption</b>
<b>Mt. Lassen</b>	California	1914-17	Explosive Eruption
<b>Mt. Hood</b>	Oregon	1790s	Pyroclastic flows and Mudflows
<b>Mt. St. Helens</b>	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 4: 带简单黑色边框的表格。

这样，每一行就好读多了，但你也许不会喜欢单元格之间的间隙。有两种方法可以改变这种外观。

第一，你可以直接用 border-spacing 属性来弥合间距，就像这样：

```

table {
 width: 100%;

 border: 1px solid #000;
}

th, td {
 width: 25%;

 text-align: left;
 vertical-align: top;

 border: 1px solid #000;

 border-spacing: 0;
}

```

```
}
```

这样边框就会彼此相接，而不是相离。它将 1px 宽的边框变成了 2px 宽，如图 5 所示：

Recent Major Volcanic Eruptions in the Pacific Northwest			
Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 5：无边框间距的表格，具有 2px 边框效果。

你也可以用 border-spacing 属性来增加单元格间距，但要记住这项属性在 IE 中不起作用。

如果你想保留 1px 边框效果，就得设置表格，使边框彼此“合并”。可用 border-collapse 属性来代替 border-spacing 属性以达到这种效果：

```
table {
 width: 100%;
 border: 1px solid #000;
}

th, td {
 width: 25%;
 text-align: left;
 vertical-align: top;
 border: 1px solid #000;
 border-collapse: collapse;
}
```

这样就会产生图 6 所示的只有 1px 边框的表格了：

Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 6: 用 `border-collapse` 设置来合并边框, 将边框宽度减为 1px 的表格。

当你设置边框合并时, 你得记住如果邻近的单元格的边框样式不同的话, 就会产生问题。不同的边框样式彼此合并的时候, 它们会互相“冲突”。在 W3C CSS2 表格边框冲突解决法则中这个问题得到了解决, 该法则规定了在合并时, 哪一种样式会“赢”。

### 填充距

现在单元格已经加上了边框, 你可能会想为标题和单元格加一些空白。直接用填充距就可以达到这种效果:

```
table {
 width: 100%;

 border: 1px solid #000;
}

th, td {
 width: 25%;

 text-align: left;

 vertical-align: top;

 border: 1px solid #000;

 border-collapse: collapse;

 padding: 0.3em;
}

caption {
 padding: 0.3em;
}
```

这样就可以让文本稍微得到一点儿“放松”了，如图 7 所示：

Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 7：给各单元格加上填充距后的表格。

#### 标题摆放

到目前为止，表格标题都还是位于表格顶部。不过你可能会想把标题移到别的地方去。可惜在 IE 中没法实现这个，但是在其它浏览器下你都可以用 `caption-side` 属性来改变标题的位置。该属性的选项有顶部，底部，左侧和右侧。下面我们来把标题移到底部：

```
table {
 width: 100%;
 border: 1px solid #000;
}

th, td {
 width: 25%;
 text-align: left;
 vertical-align: top;
 border: 1px solid #000;
 border-collapse: collapse;
 padding: 0.3em;
 caption-side: bottom;
}

caption {
 padding: 0.3em;
```

```
}
```

效果如图 8。

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

Recent Major Volcanic Eruptions in the Pacific Northwest

图 8：标题位于底部的表格。

如果你想移动标题的话，要牢记任何 `side-specific` 样式在 IE 下都无法奏效。比如，即使你加了三条边框来使标题与表格底部连接，在 IE 下也不会有效果，标题仍然会位于表格顶部。你只有用条件注释来在 IE 下改变你的表格式样。更多详细内容将会在稍后的 [利用条件样式来满足 IE 的要求](#) 部分谈到。

至于其它的示例，我在开头已经给出了。

## 背景

另一种样式化表格的简单方法是添加背景颜色和图片。这是通过 `background` 属性来完成的，但你要注意的是，表格的各个部分会“一层叠一层”。CSS2 规范说明了背景层次的一些细节，简短地说就是背景会以下面的顺序彼此重叠：

1. 表格（位于“底部”或“后部”）
2. 列组
3. 列
4. 行组
5. 行
6. 单元格（位于“顶部”或“前部”，也就是说单元格的背景覆盖在所有其它背景之上）

因此，如果你为表格设置了一个背景，而为单元格设置了另一个背景颜色的话，单元格的背景将会覆盖表格的背景。如果你的边框设置为 `collapse`，就根本看不见表格的背景了。然而，如果你将 `border-collapse` 设为 `separate` 的话，表格背景将显示在单元格间隙之中。

注意，网页中不同元素互相覆盖的情况是可调节的；通过改变某个元素的`z-index`属性，你可以控制该元素在“堆”中相对于其它元素的位置。在第 37 篇文章中你可以学到更多关于`z-index`的知识。

假设你将表格背景设为红色，单元格背景设为白色。不相连的单元格可以露出红色来，但单元格本身仍然是白色的，如图 9 所示：

Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 9：单元格的白色背景之间露出红色的表格背景。

你也可以使用背景图片。比如，假如你想在单元格之间显出渐变，你可以将`th` 和 `td` 单元格设为白色背景，将`table` 背景设为一张渐变图片：

```
table {
 border-collapse: separate;
 border-spacing: 5px;
 background: #000 url("gradient.gif") bottom left repeat-x;
 color: #fff;
}

td, th {
 background: #fff;
 color: #000;
}
```

注意，背景颜色是设为黑色的，用以充满渐变图片结束后在表格顶部留下的空白（你应当使表格总比背景图片长一点）。前景色设为白色，以免默认颜色在单元格内容上显露出来。一般来说，单元格样式会覆盖`table {}`样式所设置的文本颜色，但是你还是应当声明每一层的前景

和背景对比色。

在大多数浏览器下，上述样式的效果如图 10 所示：

Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 10：渐变背景图像在单元格之间的显示

由于 IE 不支持 border-spacing，在默认状态下它是不能显示同样的背景效果的。但你仍可以得到大致差不多的效果，如图 11 所示：

Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 11：IE 渲染出来的 border-spacing 间距变小了

随着情况的不同，你可能会很乐意就这么接受浏览器之间的渲染效果差异。当然这并非总是可行，比如某个客户可能会特别要求你的设计在所有浏览器下外观完全一致。

### 利用条件样式来适应 IE

有一个变通方案可以解决上述的 IE 问题。尽管这需要一点技巧以及一张附加样式表，但它相当有效。你可以用 expression 来生成更宽的间距，然后用条件注释来载入该表达式。该表达式语法为：

```
table {
 border-collapse: expression("separate", cellSpacing = "5px");
}
```

```
}
```

这段 CSS 仅对 IE 有效，因此你只能将其用在 IE 上。这个表达式还会使你的样式表失效，所以许多开发员宁愿单独用一张只在 IE 上加载的样式表来书写 IE 伎俩。

为了达到这个目的，我们新建一张样式表，命名为 `ie-only.css`，并将其链接在条件注释中。

```
<!--[if lte IE 7]><link rel="stylesheet" media="screen" href="ie-only.css">
/><!--[endif]-->
```

注意 `[if lte IE 7]` 的意思是“如果浏览器版本低于或等于 IE7”。该语句将这段代码应用于 IE7 及更早的 IE 版本上，而周围的 HTML 注释将这段代码对其它所有浏览器隐藏。你可以将该语句调整为针对 IE 的任何其它版本，比如你想指向 IE6 的话，就用 `[if IE 6]`。

在主样式表中设置普通样式：

```
table {
 border: 1px solid #000;
 border-collapse: separate;
 border-spacing: 5px;
 background: #000 url("gradient.gif") bottom left repeat-x;
}
```

然后在 `ie-only.css` 中设置仅针对 IE 的样式：

```
table {
 border-collapse: expression("separate", cellSpacing = "5px");
}
```

这样 IE 就会生成宽单元格间距的表格。别忘了对你的特殊宽度设置进行维护——如果你更新了主样式表的话，也必须要更新 `ie-only.css` 才行。显然，比起单单样式化表格来说，条件注释使你能做出更多的效果，因为附加样式表足以容纳适应 IE 漏洞所需的所有 CSS。

### 一个简单的设计

大多数网页设计使用的都是相对简单的背景组合。现在我们来将表头背景设置为灰色，将标题改成黑底白字：

```
table {
 width: 100%;
```

```
border: 1px solid #000;
}

th, td {
 width: 25%;
 text-align: left;
 vertical-align: top;
 border: 1px solid #000;
 border-collapse: collapse;
 padding: 0.3em;
 caption-side: bottom;
}

caption {
 padding: 0.3em;
 color: #fff;
 background: #000;
}

th {
 background: #eee;
}
```

效果如图 12 所示:

Recent Major Volcanic Eruptions in the Pacific Northwest			
Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 12: 将标题颜色反向成黑底白字，表头单元格背景改成浅灰色的表格

### 常见的变化

在这一部分中我们将会看到一些常见的设计原型，这些设计在网页表格上总是一再地出现。

#### 斑马纹

有种常见的表格设计要求是使表格行的颜色产生交替。这种颜色交替通常被称作“斑马纹”。虽然 对于斑马纹是否真的有助于阅读存在一些猜测，但它们的确是一种流行的样式。斑马纹示例如图 13 所示：

Common table elements		
Element	Tag	Purpose
table	<table>	Encloses a table
table row	<tr>	Encloses a row of table cells
table header cell	<th>	Sets a heading for a row or column
table data cell	<td>	Contains data
caption	<caption>	Sets a caption for the table

图 13：一张“斑马纹”表格，表格行设置为白色与灰色交替。

实现斑马纹效果最简单的办法是添加一个类来使表格行产生交替，然后用 CSS 上下文选择符来样式化行中的单元格。首先，将“odd”类和“even”类添加到表格行中，就像这样：

```

...
<tr class="odd">

```

```
...

<tr class="even">

...
...
```

由于表头已经有自己的样式了，你可以将其跳过不管。然后添加上下文类来设置 `odd` 类表格行中所有的单元格背景。

```
.odd th, .odd td {
background: #eee;
}
...
```

这是为跨浏览器的 HTML 表格添加条纹的最简单的办法，但是该方法并不是很完美——要是你在表格中插入了行的话又该怎么办呢？那你就得互换所有的 `odd` 和 `even` 类名来使一切重新搞定。

还有另外两种办法：

- 就如在 [A List Apart: 多条纹表格](#) 中所阐述的那样，你可以用非干扰性的JavaScript语言来添加类。大多数JavaScript框架也有适用于多条纹表格的方法：[多条纹表格之一决胜负](#) 中就将一系列框架的实现进行了比较。
- 你可以用 CSS3 `:nth-child` 选择符，但该方法尚未获得所有主流浏览器的支持。不过随着时间的推移，浏览器支持也会得到改善。

你可以在 [dev.opera.com](http://dev.opera.com) 上找到更多专门讨论用 `nth-child` 来做斑马纹效果的文章。

## 不完全网格

S 有的网页设计对于结构化程度较低、外观较开放的网格响应速度很快。对此，最简单的一种变化就是删除竖边框并省略标题栏的背景，如图 14 所示：

### ***Recent Major Volcanic Eruptions in the Pacific Northwest***

<b>Volcano Name</b>	<b>Location</b>	<b>Last Major Eruption</b>	<b>Type of Eruption</b>
<b>Mt. Lassen</b>	California	1914-17	Explosive Eruption
<b>Mt. Hood</b>	Oregon	1790s	Pyroclastic flows and Mudflows
<b>Mt. St. Helens</b>	Washington	1980	Explosive Eruption
Compiled in 2008 by Ms Jen			

图 14:仅在外缘和单元格底部有浅灰色边框的表格

实现该效果的 CSS 代码如下:

```
table {
 width: 100%;
 border: 1px solid #999;
 text-align: left;
 border-collapse: collapse;
 margin: 0 0 1em 0;
 caption-side: top;
}

caption, td, th {
 padding: 0.3em;
}

th, td {
 border-bottom: 1px solid #999;
 width: 25%;
}

caption {
 font-weight: bold;
 font-style: italic;
```

```
}
```

在此之上你可以更进一步，删掉所有边框，只留下表格顶部和底部的边框以画出表格主体的界线——参见图 15：

Recent Major Volcanic Eruptions in the Pacific Northwest

Volcano Name	Location	Last Major Eruption	Type of Eruption
Mt. Lassen	California	1914-17	Explosive Eruption
Mt. Hood	Oregon	1790s	Pyroclastic flows and Mudflows
Mt. St. Helens	Washington	1980	Explosive Eruption

Compiled in 2008 by Ms Jen

图 15:仅在主体的顶部和底部有边框的表格。

实现该效果的 CSS 如下：

```
table {
 width: 100%;

 text-align: left;

 border-collapse: collapse;

 margin: 0 0 1em 0;

 caption-side: top;
}

caption, td, th {
 padding: 0.3em;
}

tbody {
 border-top: 1px solid #000;

 border-bottom: 1px solid #000;
}

tbody th, tfoot th {
```

```
border: 0;
}

th.name {
 width: 25%;
}

th.location {
 width: 20%;
}

th.lasteruption {
 width: 30%;
}

th.eruptiontype {
 width: 25%;
}

tfoot {
 text-align: center;
 color: #555;
 font-size: 0.8em;
}
```

## 内部网格

有时你可能会想删除表格的外边框，只保留由内部边框组成的网格，如图 16：

## Recent Major Volcanic Eruptions in the Pacific Northwest

<b>Volcano Name</b>	<b>Location</b>	<b>Last Major Eruption</b>	<b>Type of Eruption</b>
<b>Mt. Lassen</b>	California	1914-17	Explosive Eruption
<b>Mt. Hood</b>	Oregon	1790s	Pyroclastic flows and Mudflows
<b>Mt. St. Helens</b>	Washington	1980	Explosive Eruption

Compiled in 2008 by Ms Jen

图 16: 内部网格式的表格

为了在现行的所有浏览器上实现该效果，你需要为每行的最后一个 th 和 td 单元格添加一个类：

```

 ...
<tr>
 <th scope="col">Volcano Name</th>
 <th scope="col">Location</th>
 <th scope="col">Last Major Eruption</th>
 <th scope="col" class="last">Type of Eruption</th>
</tr>
...

```

然后我们用该类来删除这些单元格的右边框，完整的 CSS 如下：

```

table {
 width: 100%;
 text-align: left;
 border-collapse: collapse;
 margin: 0 0 1em 0;
 caption-side: top;
}

```

```
caption, td, th {
 padding: 0.3em;
}

th, td {
 border-bottom: 1px solid #000;
 border-right: 1px solid #000;
}

th.last, td.last {
 border-right: 0;
}

tfoot th, tfoot td {
 border-bottom: 0;
 text-align: center;
}

th {
 width: 25%;
}
```

## 使用:`:lastchild`的内部网格

如果浏览器支持改善的话，我们就可以不用类，而用`:lastchild`这个伪选择符来达到这种效果了。其 CSS 如下：

```
table {
 width: 100%;
 text-align: left;
 border-collapse: collapse;
```

```
margin: 0 0 1em 0;
caption-side: top;
}

caption, td, th {
padding: 0.3em;
}

th, td {
border-bottom: 1px solid #000;
border-right: 1px solid #000;
}

th:lastchild, td:lastchild {
border-right: 0;
}

th {
width: 25%;
}
```

目前这段代码只能在最新版的 **Opera**, **Firefox** 和 **Safari** 浏览器下工作。

## 两个常见的漏洞

在本文最后，我们来看两个十分常见的漏洞，这样当它们突然冒出来的时候你好有个准备。这两个漏洞与表格的边框和标题有关。

### 边框合并漏洞

当你将表格设置为 `border-collapse: collapse;` 时，你将会发现在 **Firefox** 和 **Safari** 浏览器中表格特征的宽度显示有误。例如，你将表格、单元格和标题的边框都设为 `1px`, **Firefox** 将会在标题的左侧少渲染 `1px`，如图 17 所示：

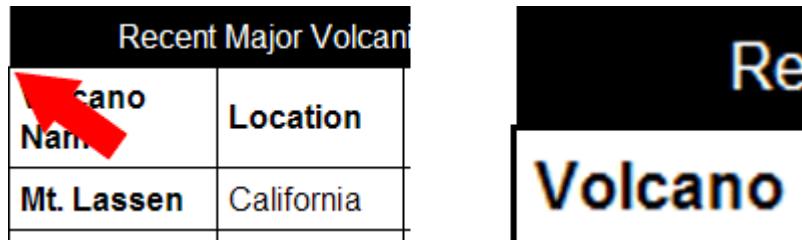


图 17: `border-collapse` 漏洞影响了 Firefox 和 Safari 浏览器。

Safari 浏览器也有同样的问题，只不过是在右侧。这个漏洞是由舍入问题引起的，归根结底就是如何显示“0.5 像素”的问题。可能有人会争论说从本质上讲这并不是漏洞，但由于浏览器不兼容，所以这事实上就是漏洞。

那么，有什么解决办法呢？如果你想同时使用 `1px` 的边框和标题背景，那就只有“忍”了。毕竟这只是一点小小的差别，而且不是什么致命的问题——也就是说，表格还是完全可以用的。所以许多人选择了接受浏览器之间的差异。就让网络顺其自然吧。

如果你愿意用宽一点的边框，比如 `2px`，你可以直接给表格、单元格和标题设置 `1px` 的边框；然后将表格边框设为 `separate`，并将它们的间距设为零：

```
table {
 border-collapse: separate;
 border-spacing: 0;
 border: 1px solid #000;
}

th, td, caption {
 border: 1px solid #000;
}
```

至少在 Firefox 下，`1px` 的边框能凑成我们想要的 `2px` 边框效果，避免了舍入问题的发生。但在 Safari 下还是会有点缺陷。

或者，你可以通过去掉标题的边框和背景颜色来隐藏这个问题。问题仍旧存在，只是你看不见罢了。这可能是最简单也最有效的解决办法了。

### 边距/标题漏洞

如果你使用了标题并在 `table` 中设置了边距，要注意，Firefox 和 Safari 可能会将表格边距放在单元格和标题之间。

为了在 Firefox 中消除这种现象，你可以设置 `table` 的三个侧面的边距，显式设置 `caption-side`，然后在 `caption` 中加入第四个边距。不幸的是，这种办法会激活 Safari 中的漏洞。所以，这其实算不上真正的解决办法，除非你愿意忍受 Firefox 或 Safari 二者之一的漏洞。

唯一可以同时避免 Firefox 和 Safari 漏洞的办法是在标题侧设置零边距。例如，如果你的标题是位于表格顶部的话，你可以只设置右边距，底边距和左边距；或只设置底边距。如果所有内容元素都只设置了同一侧的边距的话，这个办法也能奏效，所以你并非必须设置边距来将邻近的内容与表格隔开。

## 总结

现在你应该已经充分掌握了表格的基本样式化选项。虽然浏览器的不协调导致了一些限制，但一般说来你还是应该能够毫不费力地创建清晰易读的表格了。还有要注意一下你的边框，给文本留点喘息的空间，并要注意背景的处理。

## 练习题

- 如何调节表格和单元格边框之间的间距？
- 当 `table` 的背景是一个颜色，`th` 和 `td` 单元格的背景是另一个颜色，而 `border-collapse` 设置为 `collapse` 的时候，会发生什么？
- 如何将不同的列设置为不同的宽度？

## 延伸阅读

- [W3C: CSS2 Tables, with particular reference to the CSS2 table background layering section.](#)
  - [A List Apart: A Dao of Web Design—"let the web be the web". A timeless article which will explain why a 1px difference between browsers doesn't truly matter.](#)
  - [A List Apart: Zebra Tables and A List Apart: Zebra Striping: Does it Really Help?](#)
  - [Zebra striping tables with CSS3](#)
  - [Supporting IE with conditional comments](#)
  - [A CSS styled table | Veerle's blog & A CSS styled calendar | Veerle's blog](#)
  - [Data Tables and Cascading Style Sheets Gallery shows off a variety of table designs \(although be aware many do not meet W3C colour contrast recommendations\).](#)
- 
- [上一篇：样式列表和链接](#)
  - [下一篇：样式表单](#)
  - [目录](#)

## 作者简介



Ben Buchanan 10 多年前就开始创作网页，那时他就读的学位课程其实并不是IT课程。他在公共机构（大学）和私营部门都曾工作过。他曾参与数家著名网站的改版，包括 澳大利亚人报的网站（[The Australian](#)）和前后三个版本的 格里菲斯大学的官方网站。目前他在 新闻数字媒体公司（[News Digital Media](#)）任前端架构师。他的博客地址为： [the 200ok weblog](#)。

## 34. 样式表单

Posted 12/15/2008 - 16:58 by Lewis

- 网络标准教程

作者: Ben Henick · 2008 年 9 月 26 日

- 上一篇: 样式表格
- 下一篇: 浮动及清除
- 目录

### 引言

HTML表单一文中介绍了关于表单的创建和样式化的基础内容。本文从该文章停下的地方开始，提供了关于表单元素和样式的更多详细内容，以及在真实的web应用程序设计中表单是如何运用的。

点击此处可下载示例代码，这样你就可以在本机主机上进行练习了。此外，本文中还有许多链接，可以链接到实际运行的例子上。

本文目录如下：

- 本文中引入的概念
  - 规则和标记超载
  - 新表单域元素
  - 表单设计原则
  - 三分法则
  - 栅格
  - 平台支持等级
- 一张简单的联系表单
  - 标记
  - 由上一张表单变化而来
  - 显而易见的缺点
  - 新表单域？什么东西？
    - 选择类型: `input type="checkbox"`
    - 从互斥选项中选择: `input type="radio"`

- 在选项太多的时候: `input type="select/option"`
- 控件系列分组: `"fieldset"`
- 从头开始, 一步步制作出最终表单效果
  - [示例 1](#)
  - [示例 1: 背景考虑因素](#)
  - [示例 2](#)
  - [示例 2: 背景考虑因素](#)
  - [示例 3](#)
  - [示例 3: 背景考虑因素](#)
  - [创建栅格](#)
  - 在布局中创建栅格框架
  - 在样式表中实现栅格
  - [示例 4](#)
  - [示例 4: 背景考虑因素](#)
  - [三分法则](#)
  - [示例 5](#)
  - [示例 5: 背景考虑因素](#)
  - [示例 6](#)
  - [示例 6: 背景考虑因素](#)
  - [示例 7](#)
  - [示例 7: 背景考虑因素](#)
  - [示例 8](#)
  - [示例 8: 背景考虑因素](#)
  - [示例 9](#)
  - [示例 9: 背景考虑因素](#)
  - [示例 10](#)
  - [示例 10: 背景考虑因素](#)
  - [示例 11](#)
  - [示例 11: 背景考虑因素](#)
  - [示例 12](#)
  - [示例 12: 背景考虑因素](#)
- 建立平台支持等级

- 实际应用(...而不是理论)中的复杂表单布局
- 总结
- 练习题
- 表：分数对小数转换表
- 参考书目

## 本文中引入的概念

这一部分提供了关于表单实现和界面布局的新信息。

### 规则和标记超载

大量使用`class`和`id`标记是违反KISS（保持简洁）准则的（在 CSS 盒模型与基础布局一文中已经解释过了）。然而，难度大的布局却经常会在级联上遇到很多冲突——这些冲突最简单的解决方法就是在元素中添加标记，以及编写包含若干选择符的样式表规则。

难度最大的布局中到处都是边缘情况，对这些边缘情况最好的处理办法就是给元素赋一个`id`，来定义一个狭义而唯一的环境。

### 新表单域元素

通常一个实际的表单所需要的不仅仅是按钮和文本输入区域，因为我们常常会需要按照选项来构建用户响应。HTML 为有这种要求的设计人员提供了若干选项。

### 表单设计原则

对于网站来说，表单常常是用户交互和数据搜集的焦点所在。因此，表单对于一个网站的成功与否是非常关键的，这就要求我们对表单的设计给予高度的关注。

### 三分法则

最容易吸引网站用户的注意力的是浏览器画布（以及穿过布局的线条）上的四个特定点。本文将向你介绍这种现象，并提供一些建议，告诉你如何通过 CSS 来最大程度地利用这种现象。

### Grids 棚格

前面的文章中我们介绍了如何 确保排版的一致性以及 最大化地利用空白。本文中我们会更进一步地阐述如何利用`em`单位来实现一定程度的布局一致性，这种一致性只有通过CSS才能实现。

### 平台支持等级

商业项目中常见的一种要求是，一个被认可的网站设计应当在一种或多种浏览器上渲染效果一致。本文将会对这种要求的原因，效果，以及用于满足这种要求的处理方法进行简要的探讨。

### 一张简单的联系表单

通过联系表单，网站访客可以直接将 e-mail 发送到站内信箱中，联系表单的使用是非常普遍的，这是因为：只要用户具有激活的 e-mail 地址，他就可以使用联系表单，而且联系表单可以很方便地结合到专门的邮件文件夹中。

在前面的 表单一 文中，所涉及到的标记就是用来创建这样的表单的，我们还对这个表单进行了一些修饰：

## 标记

```
<form id="contactForm" method="post" action="/cgi-bin/service_email_script.php">

 <li id="nameField" class="required"><label for="realname">Name:</label><input type="text" name="name" value="" class="medium" id="realname" />required

 <li id="addressField" class="required"><label for="address">Email:</label><input type="text" name="email" value="" class="medium" id="address" />required

 <li id="subjectField"><label for="natureOfInquiry">General subject:</label>

 <select name="subject" class="medium" id="natureOfInquiry">

 <option value="support">Support</option>

 <option value="billing">Accounts & billing</option>

 <option value="press">Press</option>

 <option value="other_q">Other questions</option>

 </select>

 <li id="acctTypeField"><label for="acctNone">Account type:</label>
```

```

<fieldset>

 <label for="acctGold">Gold</label><input type="radio" name="acct_type"
id="goldAcct"

 class="rInput" />

 <label for="acctSilver">Silver</label><input type="radio" name="acct_type"
id="acctSilver" class="rInput" />

 <label for="acctBronze">Bronze</label><input type="radio" name="acct_type"
id="acctBronze" class="rInput" />

 <label for="acctNone">None</label><input type="radio" name="acct_type"
id="acctNone"

 class="rInput" checked="checked" />

</fieldset>

required

<li id="availabilityField">

 <label for="availability">My account is unavailable:</label><input
type="checkbox"

 name="is_down" id="availability" class="rInput" />

 <li id="messageField"><label for="messageBody">Comments:</label><textarea
name="comments"

 cols="32" rows="8" class="long" id="messageBody"></textarea>

<li class="submitField"><input type="submit" value="Send" class="submitButton"
/>

</form>

```

## 由之前的表单变化而来

除了包含了几个新元素之外，这段标记之中还添加了许多 **class** 和 **ID**，这些 **class** 和 **ID** 可以在样式表中加以引用。这样就可以对每个表单，表单域/值组，以及表单域分别加以引用，而

不用管上下文。

此外，通过新的标志符，设计师就可以将必须填写的表单域和不是必填的表单域区分开来。

最后，这段代码中还有一些新的类，用来提示自身所在的表单元素应该显示的信息的数量和类型。通过这些类，就可以将布局细节同时应用到多个任意元素上。

### 显而易见的缺点

由于该示例表单只具备最基本的内容，我们用标题替换掉了之前文章的表单中的 `legend` 元素。

标签是最适合用在 `fieldset` 中的，而不是 `label` (`label` 更多地是与单个控件相关)。在本例中我们完全忽略掉了 `legend` 元素，因为它很难样式化。

另外还需要注意的是，在源顺序中，表单域的“`required`”标签最好是放在表单域自身之前，以满足屏幕阅读器用户的需要。然而，为了对这些项目进行合理安排，`position` 属性（不属于本文的讨论范围）是必需的。因此，“`required`”标签在源顺序中是放在其相关控件之后的（即使是在同一个上下文中）。

### 新表单域？什么东西？

文本框和提交控件在前面的文章中已经介绍过了。就如上面所说的，我们会遇到很多实际用例，在这些情况中要让用户能够选择两个或两个以上的选项。下面我们将简要地谈一下涉及这些操作的元素。

#### 选择类型: `input type="checkbox"`

```
...
<label for="availability">My account is unavailable:</label><input type="checkbox" name="is_down" id="availability" class="rInput" />
```

选择加入或选择退出这类问题通常都是通过这些控件之一来实现的。此外，这类控件还可以用在需要从几若干选项中任选几种的时候，比如说，一张个人爱好清单。

#### 在互斥选项中选择: `input type="radio"`

```
...
<label for="acctNone">Account type:</label>
<fieldset>
 <label for="acctGold">Gold</label><input type="radio" name="acct_type" id="goldAcct">
```

```

 class="rInput" />

<label for="acctSilver">Silver</label><input type="radio" name="acct_type"
id="acctSilver" class="rInput" />

<label for="acctBronze">Bronze</label><input type="radio" name="acct_type"
id="acctBronze" class="rInput" />

<label for="acctNone">None</label><input type="radio" name="acct_type"
id="acctNone"

class="rInput" checked="checked" />

</fieldset>

```

通过一组单选框，你可以把若干选项排列在一起，在这些选项中只能选择一个。比如，在 1-5 或 1-10 的等级范围内指定一个数值，这个例子直观地阐述了 `radio` 控件的使用。

跟其它表单控件不同的是，`radio` 控件不仅是允许，更是要求各个相互关联的控件的 `name` 相同。

这些元素的得名是来自于常见的机械式调音的汽车收音机界面。跟那些常见于数字调音收音机的由程序控制的预设不同，机械“预设”按钮只要一按下，就会从一系列波段中将收音机聚焦在要收听的那个波段上。

`checkbox` 和 `radio` 控件都有 `checked` 属性，只要设置了该属性，就会在初次渲染的时候默认激活该控件。

关于是用 `radio` 控件，还是 `checkbox` 控件的问题，最好是在考虑完各种不同因素后再作决定。如果你想让用户对一项主观性的选择（比如，加入邮件列表）进行确认，那么 `checkbox` 控件可能是最好的选择。如果你想让用户在两个客观性的选项（比如说性别）中做出选择的话，就应该用 `radio` 控件。

### 在选项太多的时候： `select/option`

```

...
<label for="natureOfInquiry">General
subject:</label>

<select name="subject" class="medium" id="natureOfInquiry">
<option value="support">Support</option>
<option value="billing">Accounts & billing</option>

```

```
<option value="press">Press</option>

<option value="other_q">Other questions</option>

</select>
```

`select` 和 `option` 元素跟一系列 `radio` 控件的效果差不多，但占的空间却要少得多。是否用 `select` 元素来代替一堆 `radio` 控件，这通常是关系到如何使用用户界面空间的问题；对于电子商务网站里的一长串地理区域的列表或部门列表这样的内容，一般都是用 `select` 元素比较好，而简短的选项（比如，是/否，真/伪，年龄段，收入范围）则应该用 `radio` 控件来排列。

慎密的自测表明，操作 `select` 列表所需的运动控制水平是很高的，但随着其包含的 `option` 数量的增加，所需的控制水平的增长是很微小的。实际结果就是，简短的互斥选项列表最好是采用带有适当 `label` 的 `radio` 控件的形式。

### 将控件系列分组：`fieldset`

`fieldset` 元素最重要的目的就是为一组紧密相关的控件划分一个单独的语境（比如将一组 `text` 控件归为电话号码，将 `select` 元素归为日期，等等。）。

### 从头开始，一步步制作出最终表单效果

既然本文所涉及到的新概念已经概述完了，现在我们就该来看一看实际的应用——下面的十二个示例全面包含了 Web 表单开发过程中遇到的各种设计概念和样式化问题。

**强烈推荐读者将示例材料保存到自己的硬盘上，并对其中的样式表规则进行尝试。**

这些示例按照源代码顺序逐渐深入，而不是按照样式表的制作顺序。之所以这样做的原因和含义将在本文后面部分进行讨论。

### 示例 1

我们从 `html { margin: 0; padding: 0; }` 规则开始，第一步是对包含该表单的页面的 `body` 进行配置。

#### 链接：

- 几乎未经样式处理过的页面
- 应用了 `body` 样式之后

#### 新样式：

```
body {
```

```
margin: 0;

padding: 1.714em;

background-image: url(images/bg_grid.gif);

font-size: 14px;

font-family: Helvetica,Arial,sans-serif;

line-height: 1.714em;

}
```

## 示例1：背景考虑因素

- 在 XHTML 与适当的 Content-Type 一起提交给能够对其进行正确支持的用户代理的情况下，默认的页面 margin 和/或 padding 是在 html 元素中渲染的。
- 为了避免上一项中所描述的情况以外的其它情况，应该在该页面周围加上 10px 的空隙；Opera 将这种空隙规定为 padding 值，而其它主流浏览器将它（有点违反直觉）规定为 margin 值。本示例中的样式表对最终效果进行了校正。
- 虽然许多可访问性专家反对 14px 的字号，对在样式表中其它地方指定的各种边框和字体属性来说，这种字号却是不可分割的一部分，这些属性绝大部分都采用了 1em 的七分之几。本文末尾附有一张分数到小数的转换表，如果你想要更深入地了解本文中用到的运算的话，可以自行参考。
- 之所以选择 14px 的值是因为，要使正文能够被几乎所有矫正视力的人阅读，14px 是最小的字号。
- 由于本文的目的之一是阐述如何实现一致性最好的栅格，本示例的页面采用了网格宽度为 24px 的栅格背景。

## 示例2

现在页面容器已经做好了，接下来的几个步骤就是改变或删除用户代理样式。

### 链接：

- 样式化原始标题并删除不想要的空隙

### 新样式：

```
h3 {

margin: 0 0 1.2em 0;
```

```

border-bottom: .05em solid rgb(0,96,192);

font-size: 1.429em;

line-height: 1.15em;

}

form {

width: 35.929em;

margin: 0;

}

ul {

margin: 0;

padding: 0;

list-style-type: none;

}

```

## 示例 2：背景考虑因素

- 标题的字号系列因平台而异；然而，默认值始终是与未经样式处理过的段落文本上使用的 medium 值成比例的，并通过级联而继承。此处设置的值是为了改变默认的比例。
- 对于页面上的第一个标题来说，用 h1 是最好的；但此处忽略了该规则，因为在商业环境下，网站的标题常常是页面上的 h1，而页面标题在标题层级上就相对要低一些。在许多情况下，这种表单的重要性是与同一文档中其它内容或表单的重要性相同的。
  - 对 h3 进行样式化的目标是创建 24 像素高的内容框，下面要紧跟 24 像素的空隙，也就是：

- $((14 \times 1.429) \times 1.15) + (20 \times .05) \approx 24$
- 

```

14 * 1.429 ≈ 20; 20 * 1.15 == 23; 20 * .05 == 1;

(20 * 1.2) = 24

```

- 如果想对元素进行适当的对齐，而不用依赖定位的话，对 `form` 或列表项进行 `width` 赋值是必须的。此处设置的值会产生一个 503 像素的静态值；剩下的一个像素的差距（给定最小栅格单位 24 像素）是由于取整和反锯齿造成的误差。
- 列表的默认用户代理样式因浏览器而异。`IE` 为每个列表项设置了一个 40 像素的左边距，并将项目符号放在边距造成的空隙中，而其它的浏览器将表单内容区块作为一个整体，在其左侧应用填充距。与前面说到的改变 `body` 规则中的布局属性一样，此处的规则是特别设计来对跨浏览器的显示进行校正的。

### 示例 3

.....现在来为表单元素创建容器。

#### 链接:

- 样式化列表项(标签`label`/值`value`组的容器)以及`fieldset`

#### 新样式:

```

li {
 clear: both;
 height: 1.714em;
 margin: 0;
}

fieldset {
 height: 1.429em;
 margin: 0 0 -.143em 0;
 border: 0;
}

```

### 示例 3: 背景考虑因素

- 如果将表单看作是一系列的 `rowsem`，那么对每一个 `rowsem` 的高度进行样式化以便维持栅格，这种需要就变得很明显了。为 `IE` 着想，也为了便于以后额外增添样式规则，我们将列表项的边距设为零。

- 由于我们将对表单中的许多元素设置 `float` 值，因此这里将包含列表项设为 `clear:both`，以确保每个列表项都会自然地降到前一个列表项之下。
- 除了删掉了边框（边框是用户代理默认样式的一部分）之外，`fieldset` 的布局属性似乎设置得很武断。事实上，这些属性是在跨浏览器测试之后才设置的，我们还会在示例 11 的注释中简短地讨论跨浏览器测试。
- 现在，我们还没有设置 `display` 或 `float` 值，因此 `fieldset` 的内容与后面的 `select` 控件相冲突了。

## 创建栅格

良好的平面设计（随之而来的就是良好的界面设计）最大的优点之一就是元素是按照可预测的间距来铺设的。这些间距又称作**栅格**。

上面已经说过，本示例中的最小栅格单元是 24 像素的正方形，但比起确保设计元素按照细小而可预测的间距来放置，保证布局的协调性所要考虑的事情更多。真正有效的栅格应当具备如下特性：

- 整个文档每列之间的空白宽度都是按照一致的网格间距来排列的。
- 同一文档中的顺序的部分与相邻列中的项目共有上边距。
- 一个布局中的插图是按照与列和最小栅格间距的宽度有关的尺寸来裁切或铺设的。
- 即便页面内容是一个单列，中间夹杂着 `float` 的元素，这种元素在大小和布局方面也是具有共同特征的。

明显具有这些特征的布局将会更富吸引力，而且更容易理解，因此也将有利于提高网站的可用性。

## 在布局中创建栅格框架

大多数专业人员用来创建站点布局构图的工具是 **Adobe Photoshop**，它的优点之一就是能轻松使用网格线。为了在 Photoshop 中显示最小栅格，你可以选择 **View → Show → Grid**，这样就能按照在 **Guides & Grid Preferences** 中设置的间距来显示网格了。

通过选择 **View → Rulers**，切换到移动工具，将标尺上的指针往标尺外拖动，就可以对诸如列之类的东西添加定位标记了。

## 在样式表中实现栅格

正如所指出的那样——示例样式表中的一些规则强化了这个概念——在一个布局中实施最小栅格最好的方法是依靠`em`单位。然而，仅靠`em`是不够的；设计师在处理替换字号，空隙，以

及边框时还必须保证分数到小数的转换正确。

在示例样式表中还展示了另一种实现栅格的技巧：使用与文档中各种元素和列的大小有关的 `class` 标记。如果保持这些标记的一致的话，大部分实施栅格的工作都可以通过它们来完成。

#### 示例 4

使元素对齐到栅格就是为标签和表单控件设置布局属性。

##### 链接：

- 将两个原始列对齐

##### 新样式：

```
label {
 display: block;
 float: left;
 clear: left;
 width: 10.286em;
 overflow: auto;
 padding-right: 1.714em;
 text-align: right;
}

input {
 height: 1.143em;
 border: .071em solid rgb(96,96,96);
 padding: .071em;
 line-height: 1;
}
```

#### 示例 4：背景考虑因素

- 所有的表单控件，包括文本编辑区和标签，默认情况下都是作为`%inline` 元素渲染的。
- 为了创建一个对齐的左侧栏，需要给 `label` 元素设置 `width` 值；在“严格”渲染模式中，`padding` 可以在控件和标签中制造出空隙。

- 使标签和控件都排齐到一个共同的边距，可以使表单变得容易读懂。这一点将和其它布局要点一起作为三分法则讨论的一部分来进行探讨——见下文。
- 现在，可以看出该表单存在着明显的问题：
  - 绑定到 `radio` 的 `label` 标签都设置了跟表单中其它 `label` 同样的 `display` 和 `float` 值。在某些浏览器中，由于已有的 `height` 和 `overflow` 值，这些元素会跟它后面的标签-控件组相冲突。
  - 在 `Safari 3` 浏览器中，本示例的文本控件的边框会消失。有人怀疑这是一个渲染漏洞。
  - `radio` 控件现在按照它们的源顺序挤成了一堆；之所以出现这样的情况，是因为穿插在其间的 `label` 控件现在处在另一个布局环境中了。
- 另一个需要注意的地方是，我们对标签设置了 `overflow: auto`。我们用在这里的小技巧也可以这样描述：假设一个`%block` 元素嵌套在另一个`%block` 元素中，并假设这两个元素中只有一个元素的 `height` 是以静态单位 `em` 来设定的，而 `em` 单位对像素的换算是已知的，对另一个元素设置 `overflow: auto`——就是没有设置 `height` 的元素——将会使其扩展到与那个设置了单独的 `height` 值的元素同一高度。这个技巧在示例 11 中也有用到。

### 三分法则



图 1：俄勒冈州 *Portland* 的早春景色。我们对这幅照片添加了线条，来阐释三分法则；注意看右下方的交叉点和形成该交叉点的线条是如何约束视觉活动的。照片作者版权所有，©2000。

在考虑如何实现优秀的布局的时候，有一个普遍存在的规律：如果你将布局或图片分成三部分，浏览者的注意力就会集中在分隔这些部分的线条（尤其是线条的交点）上。如果没有在设计中运用这个奇异的规律的话，你的布局就会显得不均衡。

对这种现象最简单的解释就是，这四条线与符合黄金比例的栅格非常接近，该比例的值接近六分之一。在各种数学领域和自然世界中经常可以遇到黄金比例的例子。



图2: [msnbc.msn.com](http://msnbc.msn.com) 的屏幕截图，上面加上了七个金色的矩形。挨在一起的第四和第五个矩形整体说明了页面布局栅格的本质。

该示例表单的布局中，表单控件都对齐到一个左边缘，这个左边缘位于从左边到假想的表单右边缘的整个距离的三分之一处，这是经过慎重考虑后出的选择。而表单的垂直布局就更是如此了——将标题考虑在内，文本区域正符合前面所说的那两条规律。就算不将标题考虑在内，必填的表单域也符合最重要的那条规律。

对于设计师来说最重要的一点就是，如果在样式表设计一开始的时候就将三分法则和栅格纳入考虑范围的话，样式表的规范化工作就会大大简化。

## 示例5

为了在水平和垂直方向上维持我们想要的栅格，还需要做一些细节调整。这些调整几乎完全是装饰性的。

### 链接:

- 对 `textarea` 和 `select` 控件外观的细微调节/a>

### 新样式:

```
textarea {

 height: 4.714em;

 margin-bottom: .286em;

 border: .071em solid rgb(96,96,96);

 padding: 0;

}
```

```

select {
 display: block;
 float: left;
 height: 1.571em;
 font-family: Futura, 'Century Gothic', sans-serif;
}

option { font-size: 100%; }

```

### 示例5：背景考虑因素

- 在 Windows 平台下，可以对 `select` 控件进行修改，以避免倒角效果会让它们看起来与文本控件很相似。
- 一般的提高表单的易用性的方法是，提供一种方式，使用户能一眼就从表单的各个部分中认出他们的输入区域在哪里，因此表单输入区域的字样应该跟其它部分的字样不同。考虑到这一点，本示例保持文本输入区的字体不变，而改变了其它部分的字体。
- 在对表单中的文本进行渲染的时候，级联会被忽略，这就是为什么这里的样式表中会有那么显眼的一大串文本/字体值的另一个原因。之所以避免用 `inherit`，完全是出于个人喜好和习惯，而不是为了什么客观的目的。

### 示例6

前一个示例对字体渲染进行了一些调整；现在我们接着来完成这个工作。

#### 链接：

- 对 `text` 控件的外观进行校正，并调整 `select` 控件文本的级联效果。

#### 新样式：

```

input, textarea {
 display: block;
 float: left;
 overflow: hidden;
 font-family: Futura, 'Century Gothic', sans-serif;
 font-size: 1em;
}

```

```
}

input, textarea, select {

 margin-top: 0;

 font-size: 100%;

}
```

### 示例 6：背景考虑因素

- 在本示例中，我们可以看到本文中第一次出现的通过复合选择符同时赋值的例子，这些值就是专门留到现在来演示复合选择符同时赋值的。这些规则中没有 `border` 值，这是我们在制作过程中特意这么做的，这个表单的实际制作过程跟这些示例的出场顺序不一样——我们后面会详细讨论这一点。
- 就像上面提到的，表单中的文本和字体值都**不会**受到级联的影响。上述规则对这个问题进行了处理。因此，现在大多数的表单控件的布局都符合我们的要求了。
- 出于对 IE6 的考虑，表单控件的协调是通过将它们的 `float` 值设为 `left` 而实现的。之所以设置这些值，是出于本人的习惯，这些习惯是由一些（不愉快的）经历所造成的，但这些设置不是必须要用的。
- 通过对文本控件指定 `block` 值，前两个示例中出现的 Safari 浏览器的渲染问题一下子就被解决掉了。像这样的怪异问题在对表单进行样式化的时候是很常见的。
- 正如还没有对 `border` 值进行适当的校正一样，`font-size` 值也还没有校正；为文本控件设置的 `1em` 值，还有为 `select` 控件设置的 `100%` 值，都是完全出于主观的。

### 示例 7

我们需要将各个文本控件的宽度改变一下，不让它们跟默认值相等。

#### 链接：

- 我们修改一下文本控件的宽度，来让它们变得更容易使用，或者至少变得更协调

#### 新样式：

```
.medium { width: 11.714em; }

select.medium { width: 12em; }
```

```
.long { width: 20.429em; }

.rInput { border: 0; }
```

### 示例7：背景考虑因素

- 重新检查一下标记源代码，你就会发现每个控件都带有一个 class——上面的规则中，三个与宽度相关的规则之中有一个是针对文本和控件的，其它的类是针对那些依靠指针/光标输入，而不是依靠键盘输入的控件的。
- 之所以要为控件设置 Class，最大的原因在于 IE 对高级选择符的支持很有限。使用选择符的时候，rInput class 可以很方便地用 input [type="radio"]，input [type="checkbox"] ... 来代替，如果当前版本的 IE 并不支持后面这种表达方式的话。
- 对三个文本控件的可能值的设置是完全随意的，可以由设计员自行决定。在商业环境中，有些设计员提交的设计在布局方面非常特殊，以至于在这里用到的那些 class 选择符可能在功能上完全无效。通过对每个标签/控件组合设置一个 id，就可以为这个表单中的每个元素提供最简单的引用方式——对那些坚持要全面操控网站布局的方方面面的设计员来说，这对样式化他们自己的设计是很有用的。

### 示例8

我们这个表单的提交按钮等候处理已经等了很久了.....

#### 链接:

- 对表单提交按钮的组成部分进行精确调整

#### 新样式:

```
.submitButton {

 display: block;

 clear: both;

 width: 7.2em;

 height: 2em;

 margin: 0 0 0 16.8em;

 border: 1px solid rgb(128,128,128);
```

```
padding: 0;

font-size: 10px;

text-align: center;

}
```

### 示例8：背景考虑因素

- 在对提交按钮进行样式化的时候所面对的最大问题就是如何让它们精确地排列成我们想要的效果。在常见的做法中，只有通过多次对布局和 `line-height` 属性进行微调，才能实现想要的外观效果；一些开发人员可能会觉得用图像替代（参见参考书目）或 `input type="image"` 要更节省时间一些。
- 乍一看，对提交按钮设置 `display: block` 有点多余——确实也是如此，如果我们仅只考虑单个页面上的单张表单的话。然而，在整个网站中，这种设置可能就不是多余的了；许多网站和应用程序在同一份文档中会有多个表单，而且其 `display` 值各不相同。

### 示例9

把“`required`”标签放到它该去的地方。

#### 链接:

- 将“`required`”标签对齐到表单中假想的右边缘去，并修改它们的文本属性

#### 新样式:

```
li.required span.note {

display: block;

width: auto;

float: right;

color: rgb(128,128,128);

font-size: .714em;

line-height: 2.4em;

font-style: italic;

}
```

### 示例9：背景考虑因素

- 在当前状态下，包含 `radio` 控件的 `fieldset` 仍然是块元素，所以它一直伸展到表单的右边距处。因此，与该控件集相关的标签被往下推到该 `fieldset` 的底部去了。
- 我们将“`required`”标签的 `width` 值设置成了 `auto`，从而规定了这些标签的宽度不能超过自身内容的宽度。
- 仔细看看“`required`”标签的排版所用到的运算的话，就会发现一个 **10** 像素的字号设置，以及 **24** 像素的行间距（等于我们所用到的栅格的最小单元）。
- 用来指示必填区域的结构是出于用户交互性的考虑而创建的；通过在该表单中应用的若干类，我们就可以用 `JavaScript` 来使用户输入生效，还可以对那些用户不能正确使用的表示，区域，和/或标示/控件集上的标签的式样进行修改。

## 示例 10

终于该来解决 `radio` 控件的冲突问题了，也就是说这些控件跟源顺序中位于它们之下的表单域之间的冲突问题。

### 链接:

- 将 `radio` 控件和它们的标签水平地排成一行

### 新样式:

```
fieldset label {
 margin-right: .25em;
 padding-right: 0;
 line-height: 1;
}

fieldset .rInput { margin-right: .75em; }

fieldset label, fieldset .rInput {
 width: auto;
 display: inline;
 float: none;
 font-size: .857em;
}
```

```
li.required fieldset {
 width: 18.857em;
 float: left;
}
}
```

### 示例 10：背景考虑因素

- 上述规则的主要效果除了进行装饰性的调整之外，还有将 `radio` 和 `checkbox` 控件的 `display` 值改回 `inline`。这样做的目的是为了避免这些控件像表单中其余的 `input` 元素一样浮动而带来的麻烦。
- 尽管我们为这些相关联的控件设置了 `display: inline`，它们仍然还是“被替换的”元素：在启动时（比如说，在浏览器实际开始对内容进行渲染之前）具有已知的静态尺寸的内联元素。因此我们可以对这些元素应用边距。
- `fieldset` 元素的特殊性质——也就是，它们是专门用在表单中的唯一的`%block` 元素——规定了所有的包含了用户必须填写的控件的 `fieldset` 必须应用单独的宽度值。（参见上面的对“`required`”标签布局属性值的讨论）。

### 示例 11

我们来做最后一步，将剩下的一点小小的参差对齐，让整个表单井井有条…

#### 链接：

- 对各个控件的布局进行最后的微调

#### 新样式：

```
#acctTypeField fieldset {
 padding: .286em 0 0 0;
 line-height: normal;
}

#acctTypeField .rInput { margin-top: .167em; }

#availabilityField label {
```

```

height: 3.143em;

padding-top: .286em;

line-height: normal;

}

#availabilityField .rInput { margin-top: .286em; }

#availabilityField, #messageField {

height: 1%;

overflow: auto;

}

```

### 示例 11：背景考虑因素

- 在示例 4 里面用到过的 `overflow` 的小技巧在这里又出现了；我们对 `#availabilityField` 中 `label` 的 `height` 设置了一个具体数值，而 `#messageField` 中的 `textarea` 也是这样。
- 为了改变其所包含的 `fieldset` 的 `padding` 值，我们用到了 `#acctTypeField` 标记，这种方法很可能是太不灵活了。然而，在编写某些非常容易受到相邻元素样式规则的影响的样式规则时，进行一些微妙的处理是必需的。
- 在这个样式表代码块中的其它规则都是对布局做微细调整用的，所有这些规则都是在测试的过程中确定的。可惜的是，长达数小时的测试和微调却没能找到在 **Safari 3** 和 **Firefox 3** 浏览器中对 `radio` 控件实现同样的排版的方法。其结果只是为 **Firefox 3** 下不协调的 `radio` 控件的 `label` 添加了一条基线。总的来说，对 `checkbox` 和 `radio` 控件进行样式化在某种意义上可以说是一种黑色艺术。

### 示例 12

前面所有的样式规则都是针对 **Opera** 或 **Safari** 的（随你挑，这两种浏览器都表现得相当好）。下面的这些样式规则是专门针对 **IE** 的，我们通过 **CSS** 文件中的 `link` 条件注释代码块来指定它只针对 **IE**。

#### 链接：

- 在**IE**中进行上面那一串最后的调整

## 新样式:

```
h3 { margin-bottom: 1.2em; }

li { margin: 0 0 -.214em 0; }

select { height: 1.429em; }

textarea { height: 4.571em; }

fieldset {

 height: 1.583em;

 padding-top: .417em;

}

.medium { width: 13.429em; }

select.medium { width: 13.714em; }

.long { width: 20.286em; }

fieldset .rInput { border: 0 !important; }

#subjectField { margin-bottom: -.214em; }

#availabilityField .rInput { margin-top: .286em; }

#messageField { padding-bottom: .286em; }

input.submitButton { margin-top: .15em; }

* html input, * html textarea { float: left; }
```

```
* html select { font-size: .643em; }

* html select.medium { width: 21.364em; }

* html textarea { height: 4.643em; }

* html #subjectField {
 margin-top: .071em;
 margin-bottom: 0;
}

* html #availabilityField label { padding-top: 0; }

* html input.submitButton { margin: .1em 0 0 7em; }
```

### 示例 12：背景考虑因素

- 如你所见，上面的所有样式规则都说明了 IE 中字号和布局盒模型的级联方式与其它浏览器有着些微不同。
- 另一个值得一提的事情就是`* html`选择符。只有 IE5 和 IE6 能识别这种选择符，因此对于专门针对这两种浏览器的样式表规则来说，这个选择符是很有效的低通过滤器。

### 建立平台支持等级

本文示例的最后一部分展示了单独为 IE6 和 IE7 编写的样式表，而尽职的网站设计团队如何针对各种不同的浏览器进行处理，还需要更多的讨论。

Web 的现实情况是，用户们在形形色色的环境下使用着五花八门的浏览器。有的浏览器是老式的，而有的却是最前沿的。有的浏览器是在配置齐全的计算机上运行的，而有的却是在电话之类的移动设备上运行。所有浏览器都是在特定的操作系统上开发的，然后再移植到其它的标准支持程度不同的操作系统上去。除了 Opera，所有的浏览器发行商所发布的浏览器都被设计成要跟一系列产品中的其它类型的产品搭配使用——这种设计要求增加了这些浏览器的复杂度，而这种复杂度对于浏览网页这样的简单任务来说是不必要的。

多种多样的浏览器优缺点已经够设计员们琢磨的了，然而现实中却还存在着漏洞的问题——安全漏洞，组件漏洞，尤其是还有渲染漏洞。*Safari 3.x* 的用户们发现，这些演示文档揭示出他们自己的浏览器在某些地方存在着令人郁闷的渲染漏洞。

解决这些问题最好的办法就是定义支持等级。这种做法，又叫做“分等级浏览器支持”，最先是由 *Yahoo!* 的界面开发团队宣传开来的。

大体上说来，支持等级分成四个大类：

1. 在浏览器的性能范围之内，网站的渲染与原本的布局一致，而且其所有站点特性都是完全支持的。这种开发平台的定义有时又叫做“**A+**”等级。
2. 网页显示明显偏离了原来的布局，甚至可能偏离程度非常显著；不过，网站仍然是可用的，而且即使不是全部特性都得到支持，大多数的站点特性是得到了支持的。
3. 对使用这一等级的浏览器的用户来说，网站勉强可以使用，不至于砸了网站所有者的招牌，而且站点特性完全不可用。这一等级的浏览器的安装基础相当小，而且被认为是过时的。
4. 文件说明书中将这一级别称作“**X** 等级”支持，这一级别的支持是专供那些未经测试的平台之用的——最具代表性的就是新出现的安装基础很小的浏览器（当然了，在大多数情况下是 *Opera*）。一旦通过测试，这类的浏览器就会升入更高的等级。

至于形成该支持等级定义的要求收集过程以及将浏览器归类至各个等级之下的详情，实在是又长又乏味，由于本文已经够长了，这些内容就略去不谈了。

#### 实际中（...而不是理论）的复杂表单布局

在上述的背景说明中，我们将示例的出场顺序按照样式表源顺序来安排，而不是按照样式表中实际的规则添加顺序来安排。之所以这么做，原因包括：

- 为了显示出基于时序的演示系列，记日志是必需的（或者将样式表储存在版本划分系统中）——这个措施从来没有被执行过。等到疏忽被觉察到的时候，本文已经走得太远了，早就不能再修改了。
- 按照源顺序来安排，大大方便了演示文档的制作——这是理想和现实之间的又一个妥协。
- 由于原来的示例样式表在编写的同时对规范化和源顺序给予了相当多（如果不是完全的话）的关注——就像所有样式表所应该的那样——按照源顺序来组织示例能保证那些希望“查看来源”的学生们在弄明白到底在实现些什么效果的时候可以轻松得多。

Opera 9.6 的 OS X 版是开发用的用户代理；除了这个附加说明和在上面提到过的其它说明之外，下面是对该样式表进行修改和增添的一般顺序：

1. 文档（比如，`body`）样式的应用
2. 列表，表单，以及 `fieldset` 默认值的重设
3. 排版的指定
4. 列表项的约束和 `clear`
5. `label` 的整体安排
6. 表单控件布局（尤其是大小）的指定和规范化
7. 提交按钮的布局
8. 边缘情况的应用
9. Safari 和 Firefox 的测试以及样式表值的修改，以达到折衷（在可能范围内）
10. IE6 和 IE7 的测试，以及条件样式表中属性/值的调整

上述的过程以最宽泛的规则开始，逐渐变得越来越具特异性，直到个别浏览器的特定缺陷被囊括其中……与样式表自身的源顺序非常相似。然而，它们的结果并不完全相关。这是因为，浏览器渲染引擎的多样性以及像 `float` 语境之类的东西的特性，在混合进多种样式的时候导致了不可意料的结果，因此实际的处理比起折返，调整，以及再考虑要更复杂。

## 总结

本文提供了一个关于表单样式化和布局的构思慎密的基础，但在此基础之上还可以进行更加深入的研究。如果设计师需要创建一个向标准靠拢的 web 表单，由操作系统（创建 Web 表单控件时借用了其组件）造成的麻烦，以及浏览器渲染引擎之间的差异会使得摆在他面前的任务更具有挑战性。本文对那些有关于这种任务的浏览器缺陷进行了初步的实验，并说明了如何达到对 web 开发中较难的一个方面的熟练掌握。

## 练习题

- 接受用户输入的表单控件的浮动类型是什么，是哪两个特征使得它们对于布局来说非常重要？
- 除了 `value` 和 `disabled` 之外，还有哪两个属性可以在用户交互之前控制表单控件设置，它们是用在哪些元素上的？
- 本文的演示文档规定了必填的表单域。写出至少一条样式规则，可以用来对含有用户输入错误或疏漏的表单域的外观进行一次改变。

- 分别描述出 `select` 元素, `checkbox` 控件和 `radio` 控件的使用案例。说明你的选择与其它可能的选择相比有哪些优点。
- 从你的指导者所选中的在线咨询网站中, 找出并简短描述 `input type="submit"` 的替代方式。
- 通过追加 `select` 属性/值组来创建一个允许选择多个 `option` 的 `multiple="multiple"` 元素。对该元素的行为方式进行检查之后, 说明为什么在网站创建过程中很少用到它。

**表: 分数对小数转换表**

在下面的表格中, 括号内加了星号的数字表示它们是无限循环小数; 比如  $0.2(6^*)$  就等于  $0.26666666666666\dots$  ( $6$  是无限循环的)。

因为表格是从左到右阅读的, 表格的左边是最接近于零的数值, 并且越往右越大。

x	1/x	2/x	3/x	4/x	5/x	6/x	7/x	8/x	9/x	10/x	11/x	12/x	13/x	14/x	15/x
2	.5	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	(3*)	(6*)	-	-	-	-	-	-	-	-	-	-	-	-	-
4	.25	.5	.75	-	-	-	-	-	-	-	-	-	-	-	-
5	.2	.4	.6	.8	-	-	-	-	-	-	-	-	-	-	-
6	.1(6*)	(3*)	.5	(6*)	.8(3*)	-	-	-	-	-	-	-	-	-	-
7	(.142857)	(.28571)	(.428571)	(.57142)	(.714285)	(.85714)	-	-	-	-	-	-	-	-	-
*	4*)	*)	8*)	*)	2*)										
8	.125	.25	.375	.5	.625	.75	.875	-	-	-	-	-	-	-	-
9	(1*)	(2*)	(3*)	(4*)	(5*)	(6*)	(7*)	(8*)	-	-	-	-	-	-	-
10	.1	.2	.3	.4	.5	.6	.7	.8	.9	-	-	-	-	-	-
11	(.09*)	(.18*)	(.27*)	(.36*)	(.45*)	(.54*)	(.63*)	(.72*)	(.81*)	(.90*)	-	-	-	-	-

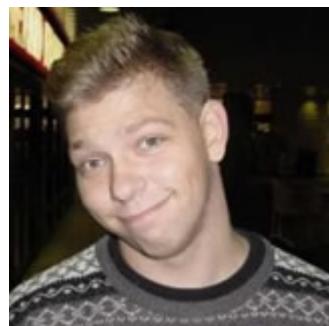
x	1/x	2/x	3/x	4/x	5/x	6/x	7/x	8/x	9/x	10/x	11/x	12/x	13/x	14/ x	15/ x
12	.08(3*)	.1(6*)	.25	.(3*)	.41(6*)	.5	.58(3*)	.(6*)	.75	.8(3*)	.91(6*)	-	-	-	-
13	.(076923 *)	.(15384 6*)	.(230769 *)	.(30769 2*)	.(384615 *)	.(46153 8*)	.(53846 1*)	.(615383 *)	.(692307 0*)	.(76923 *)	.(846153 6*)	.(92307 -)	-	-	-
14	.0(71428 5*)	.(14285 7*)	.2(14285 7*)	.(28571 4*)	.3(57142 8*)	.(42857 1*)	.5	.5(71428 5*)	.6(42857 1*)	.7(71428 5*)	.7(85714 2*)	.(85714 2*)	.9(28571 4*)	-	-
15	.0(6*)	.1(3*)	.2	.2(6*)	.(3*)	.4	.4(6*)	.5(3*)	.6	.(6*)	.7(3*)	8	.8(6*)	.9(3 *)	-
16	.0625	.125	.1875	.25	.3125	.375	.4375	.5	.5625	.625	.6875	.75	.8125	.87 5	.93 75

## 参考书目

- Bos, Bert, et al. 2007. Cascading style sheets level 2 revision 1 (CSS 2.1) specification. World Wide Web Consortium. <http://www.w3.org/TR/2007/CR-CSS21-20070719> etc. (accessed 28 May 2008).
- Henick, Ben. 2006. 12 lessons for those afraid of CSS and standards. A List Apart. <http://www.alistapart.com/articles/12lessonsCSSandstandards> (accessed 16 December 2008).
- Horton, Sarah, and Lynch, Patrick. 2002. Web style guide: basic principles for creating web sites, 2nd edition. New Haven, Conn.: Yale University Press.
- Knott, Ron. 2008. The golden section ratio: phi. Department of Mathematics, University of Surrey (UK). <http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/phi.html> (accessed 18 December 2008).
- Meyer, Eric. 2007. Formal weirdness. Meyerweb.com. <http://meyerweb.com/eric/thoughts/2007/05/15/formal-weirdness/> (accessed 17 December 2008).

- Microsoft Corporation. msnbc.com.  
<http://msnbc.msn.com/> (accessed 17 December 2008).
  - Raggett, Dave, *et al.* 1999. HTML 4.01 specification.  
World Wide Web Consortium. <http://www.w3.org/TR/1999/REC-html401-19991224> etc.  
(accessed 30 June 2008).
  - Reynolds, Garr. 2005. From golden mean to “rule of thirds.”  
Presentation Zen.  
[http://www.presentationzen.com/presentationzen/2005/08/from\\_golden\\_me.html](http://www.presentationzen.com/presentationzen/2005/08/from_golden_me.html)  
(accessed 16 December 2008).
  - Santa Maria, Jason. 2008. Making modular layout systems.  
24 Ways. <http://24ways.org/2008/making-modular-layout-systems> (accessed 16 December 2008).
  - Wikipedia. 2008. Fahrner image replacement.  
[http://en.wikipedia.org/wiki/Fahrner\\_Image\\_Replacement](http://en.wikipedia.org/wiki/Fahrner_Image_Replacement) (accessed 19 December 2008).
  - Yahoo! Developer Network. 2008. Graded browser support.  
<http://developer.yahoo.com/yui/articles/gbs/> (accessed 16 December 2008).
- 
- 上一篇：样式表格
  - 下一篇：浮动及清除
  - 目录

## 作者简介



Ben Henick 从 1995 年 9 月就开始以各种身份参与创建网站了，那时他以一个学术志愿者的身份进行了他的第一个 Web 项目。从那时开始，他的大多数工作都是以自由工作者的形式完成的。

Ben 是一个多面手。他的技能触及了网站设计和开发的几乎每一个方面，从 CSS 到 HTML，再到设计和文案撰写，再到 PHP/MySQL 和 JavaScript/Ajax。

他生活在堪萨斯州的 Lawrence，有三台电脑，却没有电视机。你可以在 [henick.net](http://henick.net) 上读到更多关于他和他工作的内容。

## 35. 浮动及清除

Posted 12/15/2008 - 16:58 by Lewis

- 网络标准教程

作者: Tommy Olsson · 2008 年 9 月 26 日

- 上一篇: 样式表单
- 下一篇: CSS的静态和相对定位
- 目录

### 序言

在本文中你将开始了解浮动与清除——现代网页设计员必备的两个工具。它们的用途很多，你可以利用它们来使文本环绕在图像周围，或者甚至用来创建多列布局。

本文结构如下：

- 浮动与清除为何而生?
- 一些乏味的理论
- 浮动是如何工作的?
  - 细节点
  - 多个浮动元素
  - 浮动元素的边距
- 清除
- 浮动元素的包含
- 收缩包围
- 浮动元素的居中
- 漏洞!
- 总结
- 练习题

### 浮动与清除为何而生?

如果往一本标准的杂志里面看去，你会看到文字环绕在文章插图周围。**CSS** 中的 `float` 属性就是为了在网页上实现这种布局样式而发明的。浮动一张图片——或者浮动任何其它页面元素

——就是将其推到一侧，使文本显示在另一侧。清除某个浮动元素就是在必要时将其下压，以防止它紧挨着浮动元素显示。

尽管任何页面元素都可设置为浮动，涉及人员普遍都是用它来达到多列布局的效果，以免滥用表格标记。

### 一些乏味的理论

为了说明浮动是如何起作用的，你需要仔细研究并观察 web 浏览器是如何渲染 HTML/CSS 文件的。别担心，我会尽量简短一些。

每个可见的 HTML 元素都会生成一个用于渲染的 盒模型。当你在电脑屏幕或移动电话上浏览该文件的时候，这些盒模型就会在屏幕上渲染出来。当你打印该文件的时候，这些盒模型就会在纸上渲染出来。当你使用屏幕阅读器的时候，这些盒模型的内容就会以听觉的方式渲染出来，就像语音一样。

就像 HTML 中有块级和内联元素一样，CSS 中也有块级和内联盒模型。从定义上讲，块级元素生成块状盒模型，而内联元素生成内联盒模型。除了由页面元素生成的盒模型之外，还有其它一些生成的盒模型，例如，为文件的文本内容而生成的盒模型。块状盒模型通常是按照元素在标记中的出现顺序从上到下排列的。如果不采用 CSS 的话，区块元素是不能并排显示的。内联盒模型是水平排列的。`direction` 属性决定了它们是从左到右还是从右到左排列（如果不特别指定的话，默认是从左到右排列）。

下面这种现象被称为文件流：内联盒模型在其上层块状盒模型中横向流动，而块状盒模型纵向流动。盒模型按照元素在 HTML 标记中的顺序出现。

看看下面这个简单的 HTML 文档（我只给出 body 元素之内的部分）：

```
<p>This is a very simple document.</p>
<p>It consists of two paragraphs.</p>
```

图 1 显示了该文件的屏幕截图，其上的装饰层显示了由 `p` 元素生成的两个块状盒模型以及由 `em` 元素生成的内联盒模型。

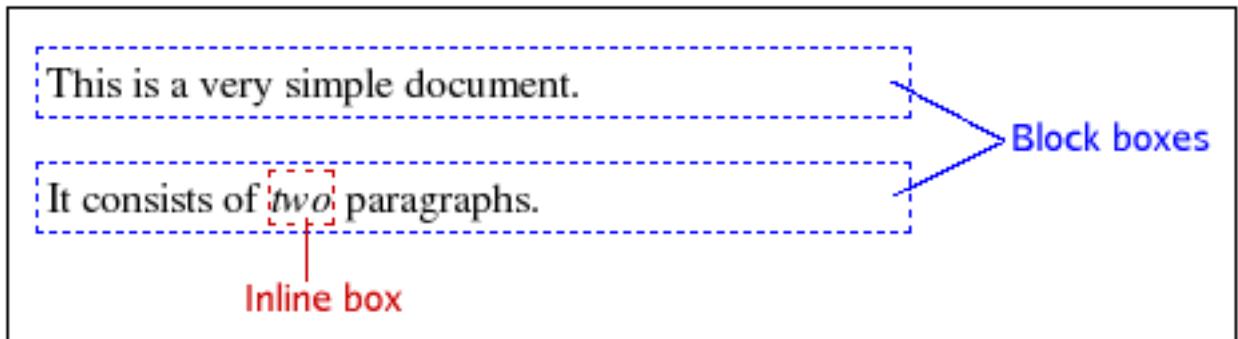


图 1：由 `p` 元素生成的块状盒模型，以及由 `em` 元素生成的内联盒模型之演示。

所有在输出设备上排成一“线”的内联盒模型都被装在想象中的长方形里，这种长方形叫做线框。线框总是从上到下排列的，彼此之间没有间距，如图 2 所示：

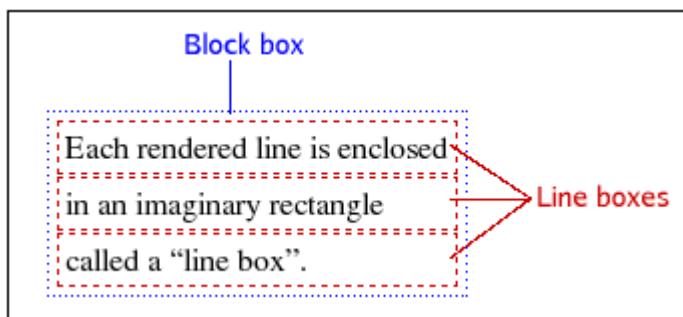


图 2：每个渲染行都被装在一个单独的线框中。

### 浮动是如何工作的？

好了！现在我们已经学完了所有乏味的理论知识，让我们来看看浮动和清除的语法规则，并学习几个例子。

`float` 属性有四个有效值：`left`, `right`, `none` 以及 `inherit`。其中，前两个值是使用得最多的，它们的作用是使一个盒模型浮动到左侧或右侧。作为默认声明，`float:none` 通常是用于在其它一些规则中“撤销”某个声明。`float:inherit` 一般很少用到——我从没见到谁在自然状态下用过它——它的存在大概仅仅是为了保持一致性。

浮动一个盒模型就是指将它从文件流中取出并按照指定的浮动方向尽可能地向左或向右移动。“尽可能”一般是指直到浮动元素的外缘接触到包含区块的边界为止（如果有填充距的话，就是到包含区块的填充距内侧）。因此，对于 `float:left` 来说，盒模型将向左移动，直到其左边界接触到父盒模型的左沿。

细心的读者可能会注意到我在上面用了“一般”这个词。当我们设置一个盒模型向左浮动时，如果已经有了一个向同样方向浮动的盒模型，后面这个盒模型就会在接触到第一个盒模型之后停止浮动。也就是说，浮动元素不会互相重叠。

下面我们要来看看浮动的运行效果，先准备好你的文本编辑器。

1. 新建一个文件，将下面代码拷进去，将文件保存为 float.html.

```
2. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">

3. <html>

4. <head>

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

6. <title>Floating</title>

7.

8. </head>

9. <body>

10. <p id="p1">Lorem ipsum
11. dolor sit amet
12.
13. consectetuer adipiscing elit.
14. Curabitur feugiat feugiat purus.
15. Aenean eu metus. Nulla facilisi.
16. Pellentesque quis justo vel massa suscipit sagittis.
17. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per
 inceptos hymenaeos.
18. Quisque mollis, justo vel rhoncus aliquam, urna tortor varius lacus, ut
 tincidunt metus arcu vel lorem.
19. Praesent metus orci, adipiscing eget, fermentum ut, pellentesque non, dui.
20. Sed sagittis, metus a semper dictum, sem libero sagittis nunc, vitae
 adipiscing leo neque vitae tellus.
21. Duis quis orci quis nisl nonummy dapibus.
22. Etiam ante. Phasellus imperdiet arcu at odio.
23. In hac habitasse platea dictumst. Aenean metus.
24. Quisque a nibh. Morbi mattis ullamcorper ipsum.
25. Nullam odio urna, feugiat sed, bibendum sed, vulputate in, magna.
```

```
26. Nulla tortor justo, convallis iaculis, porta condimentum, interdum nec,
 arcu.

27. Proin lectus purus, vehicula et, cursus ut, nonummy et, diam.</p>

28.

29. <p id="p2">Nunc ac elit. Vestibulum placerat dictum nibh. Proin massa.

30. Curabitur at lectus egestas quam interdum mollis.

31. Cras id velit a lacus sollicitudin faucibus.

32. Proin at ante id nisi porttitor scelerisque.

33. In metus. Aenean nonummy semper enim.

34. Aenean tristique neque quis arcu tincidunt auctor.

35. Fusce consequat auctor ligula.

36. Fusce nulla lorem, sagittis a, lacinia et, nonummy in, eros.

37. In nisi augue, aliquam eget, convallis vel, malesuada quis, libero.</p>

38.

39. <p id="p3">Hello, World!</p>

40.

41. </body>
```

```
</html>
```

这段内容可真多，但我们需要一些文本内容来显示浮动是如何运作的。

42. 用你的 web 浏览器打开这个文件来看看效果。看起来很烦人，对不对？

43. 在文本编辑器中再新建一个文件，填入以下代码，在同一路径下将其另存为 **style.css**。

```
44. #span-a {
45. float: left;
46. background-color: #cfc;
47. color: #030;
```

```
}
```

48. 将下面语句插入到</head>标签之前，使该样式表链接到 HTML 文档中：

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

49. 保存 HTML 文档，并在浏览器中刷新。现在可以看到包含“*Lorem ipsum*”字样的 `span` 元素浮动到了左侧。为了使其稍微显眼一点，我将它的背景设置成了浅绿色。

50. 这个效果现在还是不太容易看出来，所以我们来将浮动元素稍微变大一点。将下面的声明加入到你的样式表中：

```
51. #span-a {
52. float: left;
53. background-color: #cfc;
54. color: #030;
55. padding: 1em;
56. }
```

56. 保存，刷新，现在可以看到绿色区域变大了，这是因为我们在该盒模型的四边上都加上了一点填充。该浮动元素占据了三行文本的高度，我们可以清晰地看到其它文本环绕在浮动元素周围。

## 细节点

下面我将更为详细地分析这段代码的机理。由第一个 `span` 元素生成的浮动盒模型被移到了左侧，直至 HTML 文档的边缘，而与之相邻的线框被缩短了。虽然不容易看出来，但是由文本段落生成的容纳该浮动元素的块状盒模型却没有受到影响。为了更清楚地演示这一点，我们给文本段落加上高亮：

1. 将下面的 CSS 规则加入到样式表中：

```
2. p {
3. border: 1px solid #f00;
4. }
```

4. 再保存一下该 CSS 文件，刷新浏览器。可以看到每个段落周围都出现了红色边框——注意，该浮动元素位于其中一个段落内部。

5. 我们修改一下最后一段代码，以证明该浮动元素停驻在其父元素填充区域的内缘处：

```
6. p {
```

```
7. border: 1px solid #f00;
8. padding: 1em;
9. background-color: #ff9;
}
}
```

10. 保存，刷新，这个结果证明了之前我所说过的话：浮动盒模型被移动到了其包含区块的填充距边缘上。还可以看到在浮动盒模型的下面，文本段落的黄色背景仍在延伸。很显然，子盒模型的浮动没有影响到段落盒模型，只影响了其内部的线框。

11. 我们来做个更进一步的实验——如果浮动元素比其父盒模型更长，会发生什么呢？将浮动规则改成下面这一段代码：

```
12. #span-a {
13. float: left;
14. background-color: #cfc;
15. color: #030;
16. padding: 1em 1em 10em;
```

```
}
```

注意：如果你的浏览器窗口比较小，你可能得把底部填充的值设得比 `10em` 更大些，这样绿色区域才能超过段落的底边框。

现在你可以看到一些有趣的现象：浮动盒模型伸过了上级区块；而上级盒模型并没有进行相应地扩展来容纳该浮动子盒模型。你也可以看到（如果你的底部填充值够大的话）与浮动元素相邻的第二个段落的线框缩短了。

## 多个浮动元素

我们再创建一个浮动元素，来看看两个元素向同方向浮动时会产生什么效果。

1. 在样式表中加入下面规则，照样保存并刷新：

```
2. #span-b {
3. float: left;
4. background-color: #ccf;
5. color: #003;
```

```
6. padding: 1em;
```

```
}
```

现在包含“**dolor sit amet**”字样的 `span` 元素也浮动到左侧去了。可以看到，该元素向左移动，直到它接触到第一个浮动元素；也就是说，“尽可能”地移动。

7. 我们干嘛要满足于两个浮动元素？我们来做第三个——将下面规则添加到样式表中：

```
8. #span-c {
9. float: left;
10. background-color: #fcc;
11. color: #300;
12. padding: 2em 1em;
```

```
}
```

13. 我希望你再添一条临时代码，来看看当行宽度不够容纳一个浮动元素的时候会发生什么现象。将下面的规则添加到样式表尾部：

```
14. span {
15. width: 34%;
```

```
}
```

16. 照样保存样式表，刷新浏览器中的页面文件——你将会看到图 3 所示的输出效果：

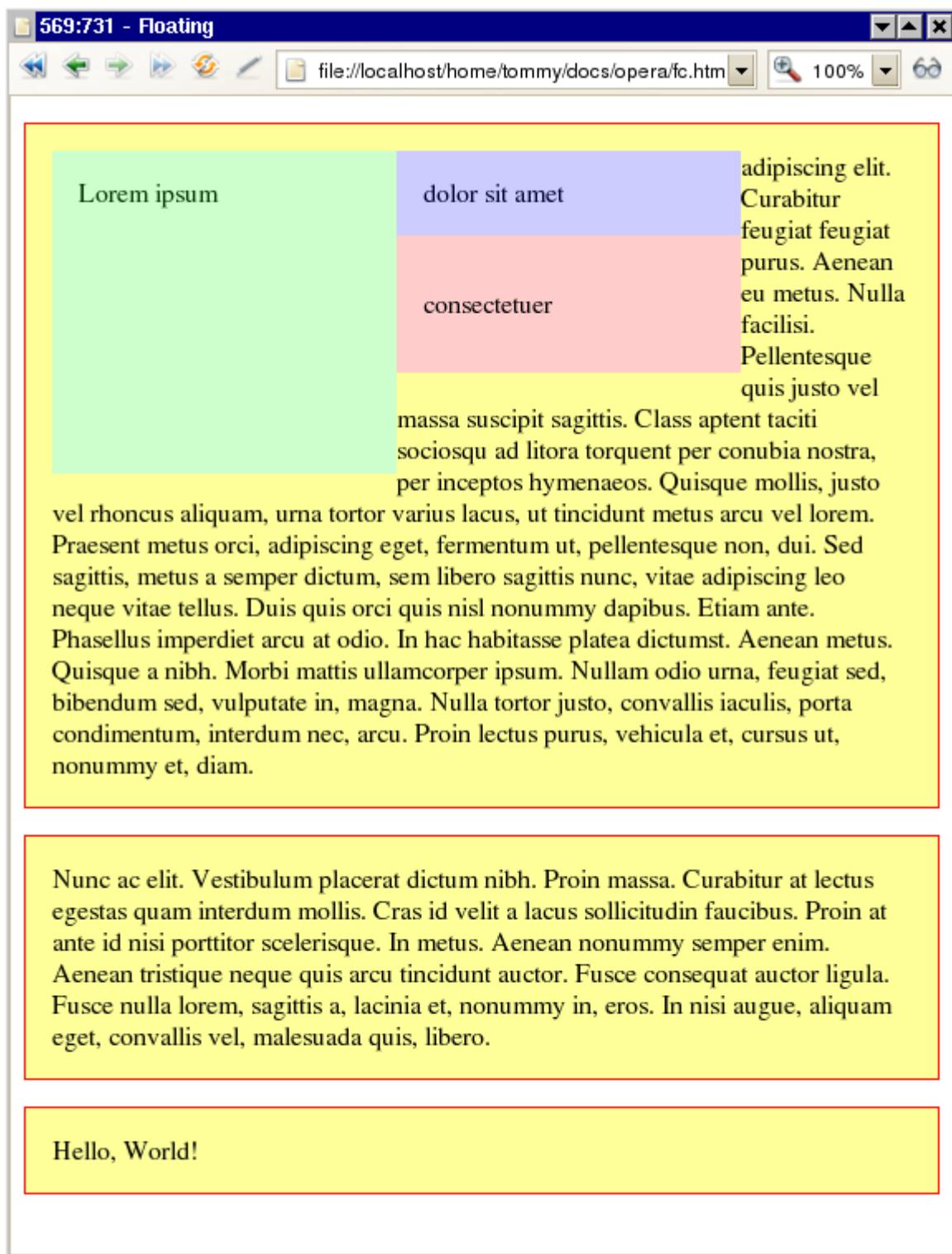


图3：是不是跟你预期的效果不大一样？

哇！刚才发生了什么了？第三个浮动元素现在跑到第二个浮动元素的下面去了！（在IE6中会产生其它一些奇怪的现象，现在我们先略过不谈。）由于每个span元素的宽度是段落宽度的34%（由第三步中加入的加码所指定），再算上填充距，总空间是不够让三个浮动元素并排显示的（ $3 \times 34\% = 102\%$ ）。前两个浮动元素位于同一行，而第三个浮动元素就只有下移了。需要注意的是

为了凑在这一行中，第三个浮动元素只下移了必要的距离。由于第一个浮动元素的右边有空间，第三个浮动元素就没有跑到这个很长的浮动元素下面去。

另一个值得一提的有趣现象是你为 `span` 元素指定了宽度。由于 `span` 是一种内联元素，指定不指定宽度本来是不会产生任何差别的。然而，在浮动某个盒模型时，会自动将该元素转换为块状盒模型，因此我们就可以对其指定大小和垂直边距。

### 浮动元素的边距

下面我们来探究一下如何利用浮动元素的边距。

- 首先，删掉你刚才添加的针对 `span` 元素的临时代码，然后保存，刷新，让三个浮动元素重新并排显示。换句话说就是删掉下面这段代码：

```
2. span {
3. width: 34%;
}
.....
```

现在浮动元素都紧紧地挨在一起了，相邻的文本紧跟在最后一个浮动元素后面显示（除非你用的是微软IE6 或更老的版本，在这种情况下会在右边出现一个 3 像素宽的间隙，这是由 3 像素慢移漏洞引起的）。怎样才能在浮动盒模型周围制造一些空档呢？答案就是**边距**！

- 我们在中间的浮动元素上尝试一下这种办法——将中间的浮动元素的 CSS 代码改成下面这段，然后保存并刷新：

```
5. #span-b {
6. float: left;
7. background-color: #ccf;
8. color: #003;
9. padding: 1em;
10. margin-left: 1em;
11.
12. margin-right: 1em;
}
.....
```

好了，现在中间那个浮动元素的两变都出现了间隔。

**13.** 你也可以在浮动盒模型上设置垂直边距——将第三个浮动元素的 CSS 代码改成下面的样子，然后保存并刷新。

```
14. #span-c {
15. float: left;
16. background-color: #fcc;
17. color: #300;
18. padding: 2em 1em;
19. margin-top: 2em;
20.
21. margin-bottom: 2em;
```

```
}
```

这样第三个浮动元素就会下移，而它下面也同样会有额外的间隔。

**22.** 让我们趁热打铁，来玩玩负边距！将第三个浮动元素的代码做如下改动，然后保存，

刷新：

```
23. #span-c {
24. float: left;
25. background-color: #fcc;
26. color: #300;
27. padding: 2em 1em;
28. margin-top: 2em;
29. margin-bottom: 2em;
30. margin-left: -4em;
```

```
}
```

你将看到如图 4 所示的输出效果：

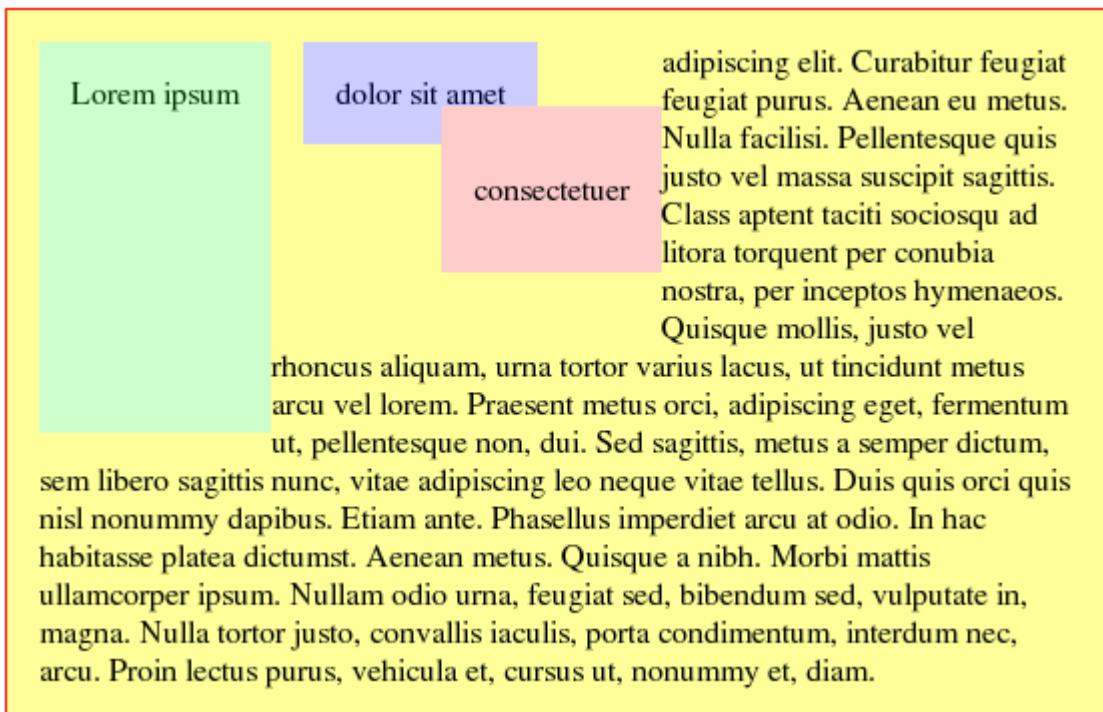


图4：现在可以看到浮动元素彼此重叠！

感觉怎么样，嗯？谁说浮动元素不能互相重叠的？注意观察负左边距是如何将整个浮动元素移到左边去的。

浮动元素负边距的使用对某些类型的多栏布局来说是非常有帮助的。

## 清除

既然我们学完了关于浮动的基础知识，下面我将讨论与之紧密相关的清除。

在本文的所有例子中，文本都是环绕在浮动元素周围的，而块状盒模型不会被浮动元素影响。有时我们需要保证某个元素不会与浮动元素相毗连。比如说，在一篇文章中，新章节开头的标题是不应该紧挨着前一章中的图片显示的。就算这幅图片伸到了前一个段落的下方，你大概也宁愿将标题放在该图片的下面。对此，唯一的解决办法就是在标题上运用 `clear` 属性。

还有一个例子就是那种随处可见的带全长度页脚的三栏式布局。如果每一栏都是浮动的，在页脚上使用 `clear` 属性就可以保证使它处于所有栏的下方——不管哪一栏最长。

`clear` 属性有三种有用的属性值：`left`, `right` 和 `both`。

`none`（默认值）和 `inherit` 也是合法的属性值。

`clear` 的使用：对某个元素设置 `left`，就是指确保该元素所生成的盒模型的位置始终处在它左侧所有浮动盒模型的下方。如果使用了 `clear:both` 的话，则是处在它两侧所有浮动盒模型的下方。

清除是通过下面操作来实现的：即在必要时将元素下移（在其顶边距上添加空格），直到其顶边降到指定方向（单个或多个方向）上所有浮动盒模型之下。下面这个例子更能说明问题：

1. 在进行操作之前，我们要清理样式表。将#span-b 和#span-c 的代码都删掉，只留下绿色的左浮动元素。确保它的底部填充距大得足够让它延伸到第二个段落去。
2. 为第二个段落添加如下规则，然后保存，刷新：

```
3. #p2 {
4. clear: left;
 }
 }
 }
 }
```

看！第二个段落下移了，一直下移到它摆脱浮动元素为止，如图 5 所示：

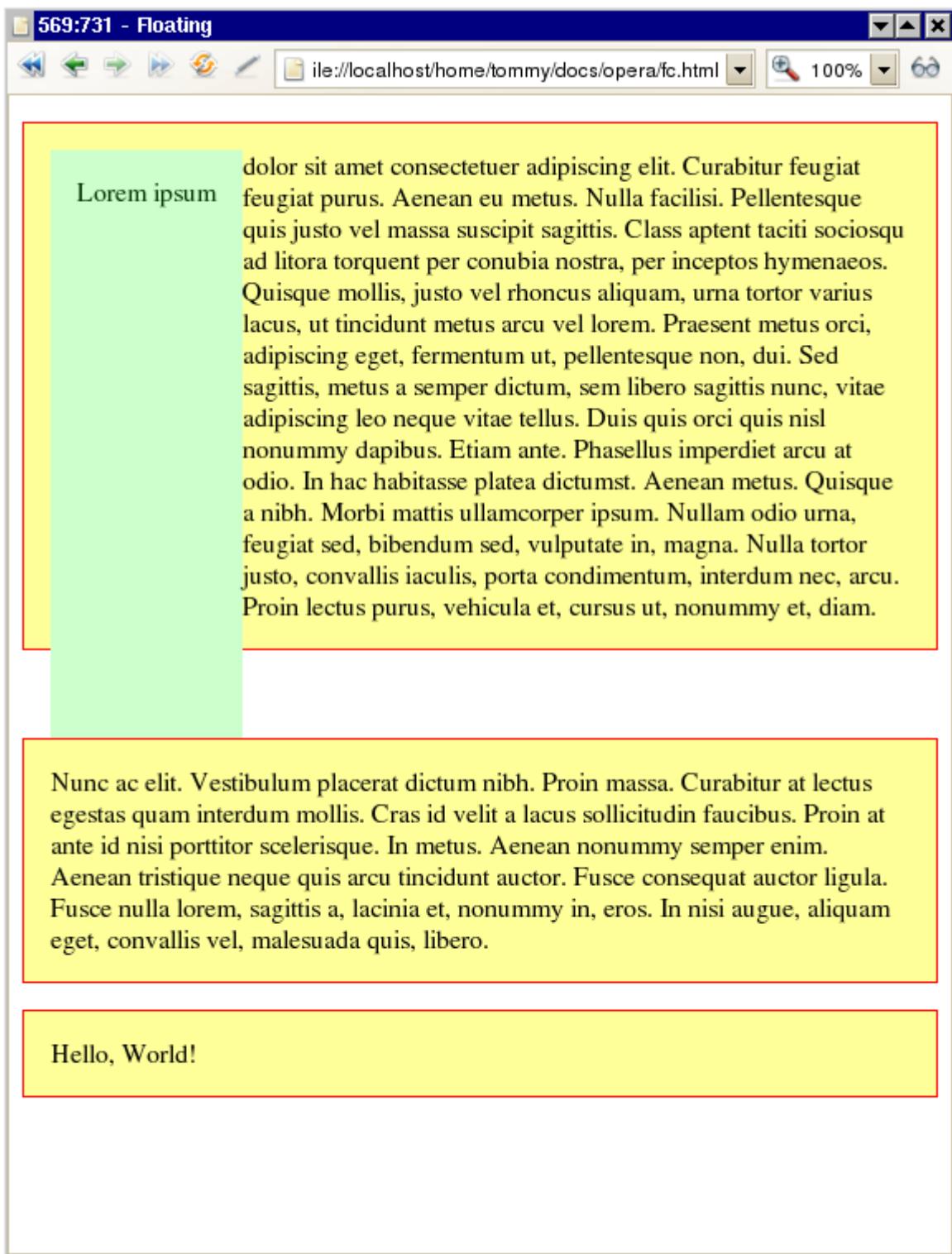


图 5:第二个段落在第一段下方实现了浮动清除。

为了制作复杂的效果，我们可以在同一个元素上运用 float 和 clear。

5. 为第二个浮动元素添加下面规则，使其对第一个浮动元素实现清除效果，然后保存并刷新页面：

```
6. #span-b {
```

```
7. float: left;
8. clear: left;
9. padding: 1em;
10. background-color: #ccf;
11. color: #003;

}
```

现在蓝色的浮动元素出现在绿色浮动元素的下面，完全跑到父段落外面去了。由于它也是向左浮动的，第二个段落就下移得更多，以实现对它的清除。

### 浮动元素的包含

就如你在上面所看到的，父盒模型并没有相应地扩展，来容纳浮动的子盒模型。这样往往会造成混乱，比如，假如你采用浮动所有 `li` 元素的办法，来从一张无序列表制作水平菜单，而这时正好某个元素的所有子盒模型也是浮动的，在这种情况下就可能引起混乱。由于浮动盒模型是从文件流中取出来的，并不会影响到父盒模型，浮动所有的子盒模型就可以将父盒模型腾空，从而使其高度归于零。有时这可能不是我们想要的，比如当你想为父盒模型设置背景的时候。这时如果父盒模型的高度为零，就无法看见背景了。

显然我们需要一些技巧来使父盒模型扩展，从而容纳其浮动子盒模型。传统方法是在标记中加入一个附加元素，将其放在父盒模型的结束标签之前，并设置为 `clear:both`。这的确有效，但是也太不方便了，因为它牵涉到引入额外的，不必要而又非语义化的标记。幸好还有其它的办法，下面我们就来探讨一下：

第一种方法是直接浮动父盒模型。浮动的盒模型总是会随着其浮动子盒模型而扩展的。

1. 为了在我们的实验文件上尝试这个办法，我们再次删掉`#span-b` 的代码，并照下面的代码将第一个段落设为浮动，然后保存并刷新：

```
2. #p1 {
3. float: left;

}
```

这样，这个段落就扩展开来，将绿色的浮动元素包含进去了。这个办法好倒是好，但有时浮动父盒模型并不可行。还有一个可以避免浮动父盒模型的办法，就是将父盒模型的 `overflow` 属性设置为除 `visible` 之外的值。如果你将其设置为 `hidden`，并且不设定高度，那么父盒

模型就会将子盒模型包含进去。

4. 将最后一行代码替换为下面这个，然后保存并刷新页面：

```
5. #p1 {
6. overflow: hidden;
}
}
```

注意，后一种办法在 IE6 及更早版本的浏览器中不能奏效。

### Shrink-wrapping 收缩包围

之前我曾经提到过，浮动某个内联盒模型会使其变成块状盒模型，从而使我们得以设定它的大小和垂直边距。浮动一个块状盒模型也同样会带来惊人的效果：如果没有指定宽度，该盒模型就会“收缩包围”从而与其内容相一致。如果你在示例文件中将第一段设置为浮动的话，就不会看到这种效果，因为第一段的内容足够填满整个窗口的宽度（除非你的显示器比它还宽）。

我们将最后一个段落设为浮动，来看看效果。为了产生一些更明显的改变，让我们大胆一些，将其浮动到右侧！

将下面的规则添加到样式表中，保存并刷新页面：

```
#p3 {
 float: right;
}
```

现在“Hello, World!”这个段落浮动到了右边，而且宽度也变了，其宽度等于文本宽度加上前面的规则中为所有段落设置的填充距。

### 浮动元素的居中

有时你也许会想要浮动某个元素——也许是让它容纳它的浮动子元素——同时使其在父盒模型中水平居中。这就产生了一个问题：你无法用普通的方法将浮动元素的左边距或右边距设置为 `auto`，也没有 `float:center` 这样的值存在。难道就没有办法来解决这个问题了吗？

事实上是有办法的。在 CSS 宗师 Paul O'Brien 的文章 [何时浮动元素才不成其为浮动元素？](#) 中说明了该问题的解决办法。它涉及到额外的包装元素，不过这个你应该不难接受。这个办法的原理是使用相对定位，关于这一点我们将在下一篇文章 [CSS 静态及相对定位](#) 中讨论。通过将包装元素向右移动，然后将浮动元素向左移回，就可以将一个宽度未知而收缩包围的浮动元素居中了！（明天你就可以用这个来给你的搭档留下深刻印象。这招可是屡试不爽的。）

让我们来试一试。在下面的例子中你将为你的网页添加一个水平菜单栏，该菜单栏是由一张带浮动项的无序列表而来的。

1. 将下面的标记插入到 HTML 文档的<body>标签后：

```
2. <div class="wrap">

3.

4. <ul id="menu">

5. Home

6. News

7.

8. Products

9. Services

10.

11.

12. </div>

13.

14. <!--Internet Explorer needs this-->
```

```
<div class="clear"></div>
```

15. 在样式表中添加如下规则，来样式化该菜单：

```
16. #menu {

17. margin: 0;

18. padding: 0.5em;

19. font-family: Verdana, sans-serif;

20. }

21.

22. #menu li {

23. float: left;

24. list-style-type: none;

25. margin: 0 0 0 0.5em;

26. padding: 0.25em;
```

```
27. background-color: #600;
28.
29. color: #ff9;
30. border: 2px solid #f00;
31.
32. #menu a {
33.
34. color: #ff9;
35. text-decoration: none;
36.
37. .wrap {
38.
39. float: left;
40. margin-bottom: 2em;
41.
42. .clear {
43.
44. clear: left;
45. height: 1px;
46. margin-top: -1px;
```

```
}
```

46. 保存这两个文件，刷新浏览器页面。可以看到你的菜单位于页面的左上角。下面我们来使它水平居中。

47. 按照下面的代码修改.wrap 规则，从而使包装元素移到页面中间：

```
48. .wrap {
49. float: left;
50. margin-bottom: 2em;
51. position: relative;
52.
```

```
53. left: 50%;
```

```
}
```

这样菜单就会从页面的水平中心开始了，但这可不是我们想要的——它离右边太近了，因此你得将它向左边移回来一点。移动的距离应当等于其宽度的一半，也就是包装宽度的一半，因此我们将其移动-50%.

54. 将#menu 规则改成下面的代码：

```
55. #menu {
56. margin: 0;
57. padding: 0.5em;
58. font-family: Verdana, sans-serif;
59. position: relative;
60. left: -50%;
```

```
}
```

现在菜单就居中了；唯一的问题是可能会有一根水平滚动条，视列表和浏览器窗口的宽度而定。这是因为你曾经将包装元素移到屏幕的中间；如果列表的宽度大于浏览器窗口宽度的一半的话，它的一部分就会跑到画面之外。

61. 52. 你可以通过在适当的父元素上设置 `overflow:hidden` 来隐藏溢出的部分，从而去掉滚动条，在本例中，包装元素的父元素是 `body`。有时我们不能对 `body` 元素设置溢出隐藏，那你就需要给包装元素再建一个包装；不过在本例中是可以对 `body` 元素进行设置的。

将下面的规则添加到样式表中：

```
body {
 overflow: hidden;
}
```

62. 事实上，还有一个问题。如果你在 IE 上观看这段代码的效果的话，你会发现它还是不能很好地工作。解决的办法是将浮动列表本身，但这仅限于 IE 浏览器，因为它会使其它的浏览器崩溃。你可以通过一点小伎俩来确保只有 IE 才能应用这个规则，从而避开这个问题。

将下面规则加入到样式表中：

```
* html #menu {
 float: left;
}
```

## 漏洞!

浮动和清除非常有用，但可惜的是大多数——即使不是全部——浏览器在这些属性的实现上都有漏洞。**IE6** 在浮动上有一大堆稀奇古怪的问题，包括内容消失，边距加倍和著名的 3 像素慢移漏洞等等。但是，在提到浮动和清除时，就连**Firefox**和**Opera**浏览器也不是完全没有漏洞的。**位置就是一切** 是一份宝贵的资源，它将这些漏洞一一罗列出来——与此同时还给出了在大多数情况下的解决办法。

## 总结

浮动一个盒模型是指在其父元素内部，将其尽可能地移动到左边或右边去。浮动盒模型是从文件流中取出来的，并不会影响到它的父盒模型或随后的块状盒模型，但是相邻的线框会被缩短。如果先前的浮动元素导致某行的空间不够容纳某个浮动盒模型的话，该浮动盒模型会下移到适合它的地方（或者直到没有其它浮动元素的地方）。

当某个内联盒模型被浮动时，它将变成块状盒模型。如果某个块状盒模型被浮动，而又没有显式指定其宽度的话，它就会收缩包围以适应自身内容的宽度。

清除浮动的功能是在必要的时候将页面内容下移，直至页面内容的顶端位于指定方向上所有浮动盒模型的底边之下为止。

使收缩包围的浮动盒模型居中可以通过添加包装元素以及对相对定位的明智应用来完成。

## 练习题

- 当你浮动某个段落中间的元素时会产生什么效果；比如说，如果在浮动元素的前面有文字的话会怎么样？一定要在不同的浏览器下进行练习，因为在不同的浏览器下效果也不同。在 **Opera** 和 **Safari** 下该项功能是正常的，但在 **Firefox** 和 **IE** 下则不然。
- 在不用布局表格的情况下，如何在一系列大小相同的“单元格”中利用浮动元素来显示缩略图？
- 如何在页面左侧制作垂直导航菜单，而内容栏位于页面右侧，从而使内容文本不再占据菜单下方？
- 常见的网站布局包括全长度页眉，在其下方是三列内容栏，接下来在底部是一个全长度页脚。如何用浮动与清除来实现这种布局？

- 上一篇：样式表单
- 下一篇：CSS的静态和相对定位
- 目录

作者简介



Tommy Olsson 是一位 web 标准和网页亲和力的实用主义倡导人，他住在瑞典中部。

Tommy Olsson 在 1993 年就写出了自己的第一份 HTML 文档，现在他在一家瑞典政府机构任专业 web 站点管理员。

迄今为止他已经写了一本书——CSS 终极参考大全（与 Paul O'Brien 一起完成的）——还有一个令人遗憾地被忽视了的博客 [The Autistic Cuckoo](#)。

## 36. CSS 的静态和相对定位

Posted 12/15/2008 - 16:59 by Lewis

- 网络标准教程

作者: Tommy Olsson · 2008 年 9 月 26 日

- 上一篇: 浮动及清除
- 下一篇: CSS 的绝对和固定定位
- 目录

### 序言

在这篇文章中我要开始深入地探讨如何利用 CSS 来随心所欲地定位 HTML 元素, 我们要用到 position CSS 属性和其它一些相关的属性。

在 CSS 中, position 属性有四个合法的属性值 (除大家都有的 `inherit`):

`static`, `relative`, `absolute` 和 `fixed`。

`static` 和 `relative` 这两个值密切相关, 我们将在本文中对它们进行详尽地讨论。同样, `absolute` 和 `fixed` 这两个属性值也是紧密相关的, 我将它们安排在本系列的下一篇文章中讨论。

本文结构如下:

- 精彩的矩形世界
- 静态定位
  - 块状盒模型布局
  - 内联盒模型布局
- 相对定位
  - 带源顺序要求的多栏布局
    - 创建栏
    - 对付IE中出现的奇怪问题
  - 相对定位的其它应用
- 总结
- 练习题

### 精彩的矩形世界

我们先来回顾一下 第 35 篇文章：浮动元素与清除中讨论过的 CSS 与 HTML 盒模型。HTML 文档包含了夹杂着字符数据（文本）的若干元素。当这样一份文档渲染在计算机屏幕上或者打印在纸上的时候，这些元素就会生成矩形盒模型。就像 HTML 元素分为块级元素和内联元素一样，CSS 中的盒模型实质上也有区块盒模型和内联盒模型之分。在默认情况下，浏览器中的嵌入式用户代理样式表会使 p 和 div 之类的 HTML 元素生成区块盒模型，而 strong 和 span 之类的内联元素则生成内联盒模型。我们可以通过 display 属性来控制生成的盒模型的类型。

由 HTML 文档中的元素生成的盒模型将按照

CSS2.1 规范所明确定义的法则来进行布局。这些法则针对的是少数编写浏览器来研究 CSS 工作原理的人，而不是针对我们这些靠设计网页谋生的人或网页设计爱好者。这就是本系列教程的意义所在！由于上述的原因，这个规范可能会有些艰深。在本文中我将以更适合网页设计师和开发员的方式来讲解这些基本原理。

## 静态定位

这个名称还真是用词不当。从 CSS 的意义上讲，设置了 position: static 的盒模型实际上根本没有“被定位”。它们的排列顺序就是自身在 HTML 标记中的顺序，同时它们会占据自身所需的空间——这其实是没有应用 CSS 时 HTML 文档的默认效果。

区块盒模型的布局和内联盒模型的布局之间有本质上的不同，我们将分别对这两种类型进行研究。因为区块盒模型比较简单，我们就从它开始：

### 块状盒模型布局

如果我们没有应用特定的 CSS 声明的话，区块盒模型会按照它们在 HTML 标记中的出现顺序从上到下垂直排列。每个盒模型通常都是与 HTML 文档（body 元素）等宽的，但就算我们把它们变窄一点，留出空间来，它们也不会一个挨一个横着排列；还是会一个接一个竖着排列。你可以想象它们前后都有隐式的换行符，使得每个盒模型都独占一“行”。

两个盒模型之间的纵向距离是由第一个盒模型的 margin-bottom 属性和第二个盒模型的 margin-top 属性控制的（之前你已经学过了如何操纵这两个属性）。对于 常规数据流 中的盒模型，例如不浮动的盒模型或绝对定位的盒模型，两个相邻块状盒模型的垂直边距会发生 合并——也就是部分重叠——因此最终结果并不是两个边距的和，而是二者之中较大的那个边距，如图 1 所示。

看看下面的 HTML 代码片段：

```
<p style="margin-bottom:40px">This paragraph has a 40px bottom margin.</p>
<p style="margin-top:20px">This paragraph has a 20px top margin.</p>
```

在浏览器中观看时，边距将会产生合并，如图 1 所示：

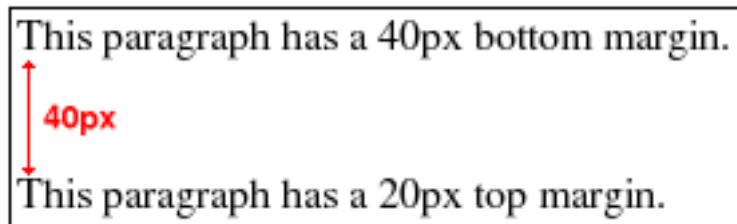


图 1：边距合并——上下两个盒模型之间的距离成了 40px，而不是 60px。

一个块状盒模型可能只包含其它块状盒模型，或者只包含内联盒模型。如果一个块级元素既包含块级子元素又包含内联子元素——这种情况是允许的，但会产生一种所谓的匿名块状盒模型来将内联子盒模型包含其中，因此父盒模型就只包含块状盒模型了。

你可以通过 `width` 和 `height` 属性来指定块状盒模型的大小，还可以对其进行垂直边距和水平边距的设置。`width` 和 `height` 属性的初始（默认）值为 `auto`，而边距的初始值为 `0`。这些值综合在一起就意味着块状盒模型在默认情况下是与其父盒模型等宽的，如图 2 所示：

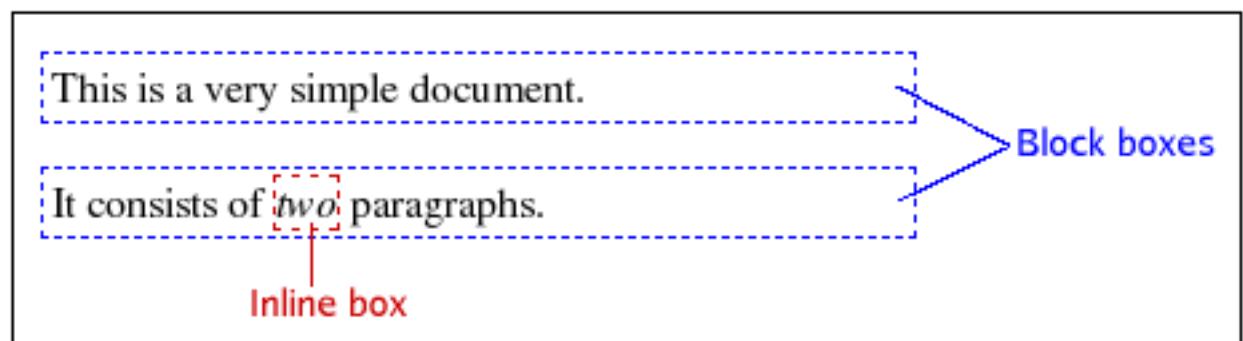


图 2：块状盒模型的排列是垂直的。

### 内联盒模型布局

对于 CSS 新手来说本章可能会比较艰深，但如果你第一遍没有读懂的话也不用灰心。，对于深入了解这些问题来说，自己下来多多尝试是最好的办法——不过你应该选用一种比较好而且标准兼容的浏览器来进行实验，比如说 Opera 或 Firefox 之类的。

内联盒模型默认是由内联 HTML 元素生成的，也同样会产生匿名内联盒模型来包含某些元素的文本内容。

内联盒模型的排列是水平的，按照它们在 HTML 标记中出现的顺序依次排列。`direction` 属性决定了内联盒模型的排列方向，当 `direction` 属性为 `direction:ltr` 时内联盒模型从左到右排列，而当 `direction` 属性为 `direction:rtl` 时则是从右到左排列。从左到右的方向可用在欧洲语言等的排列上，而从右到左的方向是用在阿拉伯和希伯来之类语言的排列上。

在屏幕上（或纸上）排成一行的内联盒模型又被装在另一个矩形中，这种矩形叫做线框。线框在其块级父盒模型中纵向排列，彼此之间没有空隙。我们可以通过 `line-height` 属性来调整线框的高度。

我们不能指定线框的方向。同时只能对其指定水平边距，而不能指定垂直边距。

必要的话，内联盒模型可以由一个分裂成几个，分散在两个或多个线框中。产生分裂时，所有的水平边距和填充距，以及所有垂直边框都只能出现在第一个盒模型之前和最后一个盒模型之后。下面的代码对 HTML 文档的 `em` 元素定义了规则：

```
em {
margin: 0 2em;
padding: 0 1em;
border: 1px dotted blue;
}
```

当样式化后的元素分散在多行时，这段代码将会产生图 3 所示的布局效果：

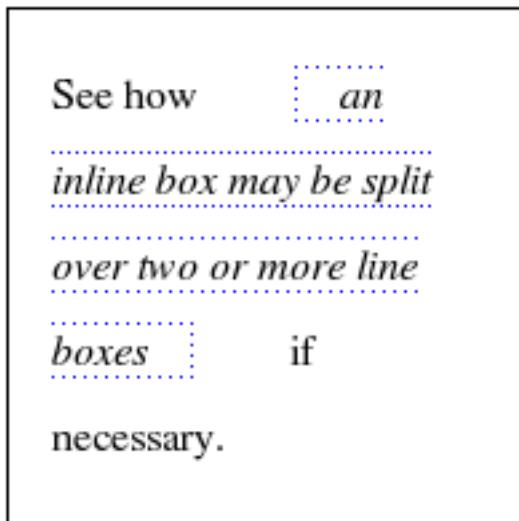


图 3: 在产生分裂的地方无法应用边距、填充距和边框。

`vertical-align` 属性决定了内联盒模型在外围线框中的垂直对齐。该属性的缺省值是 `baseline`，即内联线框的对齐使得其文本基线排成一线。基线是指没有伸尾部分的字母所依照的一条假想线。它位于线框底部之上，并留有一段距离，以便给某些小写字母的下伸部分留下地方，如图 4 所示：

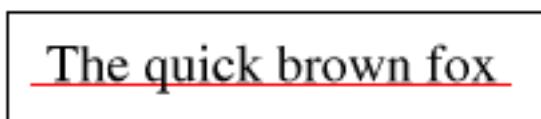


图 4: 字母依照着假想的基线排列。

注意, `vertical-align` 属性只能用在线框和表格单元格上, 并且不能继承。图 5 显示了一些在垂直对齐方面各不相同的小图片。

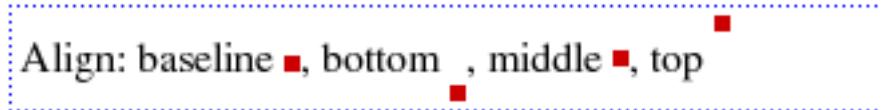


图 5:通过 CSS 属性 `vertical-align` 的设置来定位图片。

如果线框中的内联盒模型的总宽度小于该线框自身的宽度, 内联盒模型的水平对齐就由 `text-align` 属性来控制。必要的话可以通过 `text-align:justify` 向内联盒模型之间插入额外的间距, 来使内容两端对齐。这个属性可以应用于块状盒模型, 表格的单元格和内联区块, 并且可以继承——图 6 中演示的是将不同的 `text-align` 属性值应用于单元格内的文本:

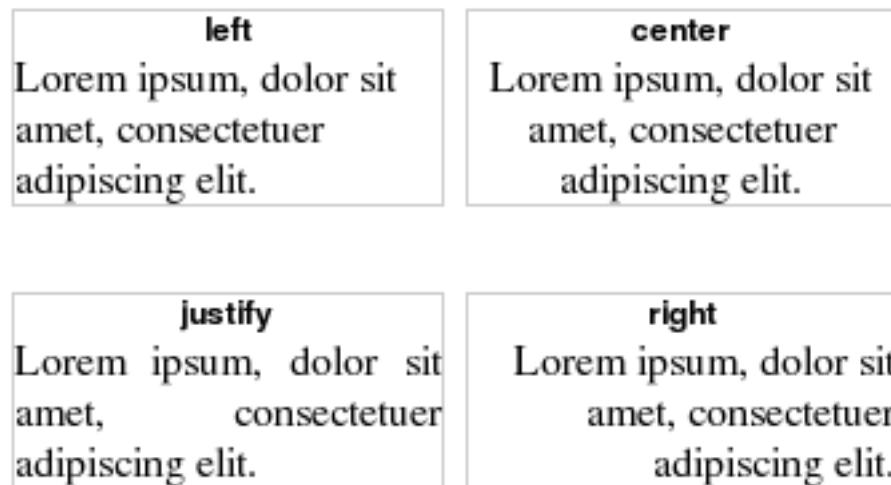


图 6: 通过 `text-align` 属性来控制文本的对齐方式。

## 相对定位

相对定位是 CSS 中的一种定位模式, 但比起它的近亲——绝对定位和固定定位来说, 它与静态“定位”的关系更为密切。

设置了 `position:relative` 属性的元素最初是像所有块级和内联静态元素一样排列的。但随后就会发生一些有趣的事情: 生成的盒模型会按照 `top`, `bottom`, `left` 和 `right` 这些属性来移动。

需要记住的是, 在相对定位下只有生成的盒模型才会移动。而元素在静态文件流中的位置保持不变。对其它元素来说, 该元素仍然“占有空间”。这就是说, 被移动的盒模型可能会与其它元素的盒模型相重叠, 因为这些元素仍然认为那些应用了相对定位的元素还处在它们原来的位置上, 跟应用相对定位之前一样。就文件流而言, 元素并没有发生位移——只是最终的视觉效果

会显示出盒模型发生了移动。我们来看一个实际的例子。

1. 在你喜欢的文本编辑器中新建一个文档，将下面的 HTML 代码拷进去，另存为 `relative.html`。

```
2. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">

3.

4. <html>

5. <head>

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

7. <title>Relative Positioning</title>

8. </head>

9. <body>

10.

11. <p>Lorem ipsum dolor sit amet consectetur adipiscing elit.

12. Curabitur feugiat feugiat purus.

13. Aenean eu metus. Nulla facilisi.

14. Pellentesque quis justo vel massa suscipit sagittis.

15. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per
 inceptos hymenaeos.

16. Quisque mollis, justo vel rhoncus aliquam, urna tortor varius lacus, ut
 tincidunt metus arcu vel lorem.

17. Praesent metus orci, adipiscing eget, fermentum ut, pellentesque non, dui.

18. Sed sagittis, metus a semper dictum, sem libero sagittis nunc,
 vitae adipiscing leo neque vitae tellus.

19. Duis quis orci quis nisl nonummy dapibus.

20. Etiam ante. Phasellus imperdiet arcu at odio.

21. In hac habitasse platea dictumst. Aenean metus.

22. Quisque a nibh. Morbi mattis ullamcorper ipsum.

23. Nullam odio urna, feugiat sed, bibendum sed, vulputate in, magna.

24. Nulla tortor justo, convallis iaculis, porta condimentum, interdum nec, arcu.

25. Proin lectus purus, vehicula et, cursus ut, nonummy et, diam.</p>
```

```
26. </body>
```

```
</html>
```

27. 浏览器中打开该文件，看看目前的效果——你应该只看到一段普通的文字。

28. 在文本编辑器中新建一个文档，将下面的 CSS 代码拷进去，并另存为 `style.css`。

```
29. p {
```

```
30. width: 20em;
```

```
31. }
```

```
32.
```

```
33. span {
```

```
34. background-color: lime;
```

```
}
```

35. 将下面语句插入到 `</head>` 标签之前，使该样式表链接到 HTML 文档中：

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

36. 保存这两个文件，并在浏览器中重新载入页面。我把段落宽度变窄了一点，这样即使浏览器窗口比较小，也能在同样的位置上产生换行符。现在 `span` 元素的背景换成了比较鲜艳的颜色，这样它就可以更醒目了。

37. 接下来，我们修改一下样式表，将下面的三个声明加入到 `span` 元素的样式规则中：

```
38. span {
```

```
39. position: relative;
```

```
40. top: 1em;
```

```
41. left: 2em;
```

```
42. background-color: lime;
```

```
}
```

43. 保存文件并在浏览器中重新加载页面，看看相对定位的效果如何。

你同时在垂直方向和水平方向上移动了 `span` 元素。注意看，该元素重叠在下一行文本上，

而它原先所在的地方空了。

生成盒模型的移动方式可能跟你想的不一样。你指定了 `top:1em`, 但盒模型却往下移动了。同样的, 你指定了 `left:2em`, 该盒模型却被移到了右边。为什么会这样?

要了解这些属性是怎样控制相对定位的, 关键是要知道它们指定的不是移动的方向, 而是应用该移动的边。也就是说, `top` 属性使盒模型相对于其顶边移动, `left` 属性使盒模型相对于其左边移动, 等等。该盒模型是从所指定的边移走, 因此 `top:1em` 是使盒模型从顶部移开 `1em`——也就是说, 下移。负数则会使盒模型向相反的方向移动, 因此 `bottom:-1em` 与 `top:1em` 效果相同。

我们可以得出另一个结论: 同时为同一个元素指定 `top` 属性和 `bottom` 属性 (或 `left` 和 `right` 属性) 没有意义的。CSS 规范规定了如果已经指定了 `top` 的话, `bottom` 就会失效。`direction` 属性控制着盒模型的横向移动。在从左到右的条件下, 如果同时指定 `left` 和 `right` 的话, `right` 就会失效; 而在从右到左的条件下, `left` 会失效。

刚才我们看到的例子对相对定位进行了一些阐述, 但似乎并不是很令人满意, 对吧? 那么相对定位到底有什么用呢? 让我们来看一个更深入的例子。

### 带源顺序要求的多栏式布局

在这里先提个醒: 本例稍微有一点复杂。如果你对CSS不熟悉的话, 可能会觉得气馁, 但我会慢慢地教你, 也会一步一步地解释给你听我到底在做什么。如果你还没有领会第 35 篇文章 (这篇文章讲的是 浮动与清除) 的内容, 现在就是个学习的好机会。

有一种布局类型在网站上非常常见。其构成是这样的: 上面是一个页眉, 页眉通常包含有一些报头图片, 再下面是并排的两个或多个“栏”, 再下面是一个全长度页脚, 页脚中可能会有版权标注或联系方式。图 7 展示了这种布局类型的一个示例。

The screenshot shows a web browser window with the title bar "600:560 - Acme Widgets Co". The address bar displays "file:///localhost/home/tommy/docs/opera/srex.html". The page content is as follows:

**Acme Widget Co.**

**Welcome to Acme Widgets!**

**Home**

**Products**

**Services**

**News**

**About Us**

**Etiam ante. Phasellus imperdiet arcu at odio. In hac habitasse platea dictumst. Aenean metus.**

**Quisque a nibh. Morbi mattis ullamcorper ipsum. Nullam odio urna, feugiat sed, bibendum sed, vulputate in, magna.**

**Nulla tortor justo, convallis iaculis, porta condimentum, interdum nec, arcu. Proin lectus purus, vehicula et, cursus ut, nonummy et, diam.**

**Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Quisque mollis, justo vel rhoncus aliquam, urna tortor varius lacinias, ut tincidunt metus arcu vel lorem.**

**Praesent metus orci, adipiscing eget, fermentum ut, pellentesque non, dui. Sed sagittis, metus a semper dictum, sem libero sagittis nunc, vitae adipiscing leo neque vitae tellus. Duis quis orci quis nisl nonummy dapibus.**

© 2008, Acme Widget Co., all rights reserved.

图7：一个典型的多栏式布局，所有的栏都夹在页眉和页脚之间。

在很早的时候（90年代），这种布局类型通常是通过布局表格来创建的。现在有种为了表面效果滥用HTML标记的倾向，这种做法是不明智的，因此在本教程中我们不会教你这样做。

CSS 提供了 `display:table-cell` 及类似的属性来实现这种布局，但是这种办法有一个很大的缺陷：那就是所有版本的 IE 目前全都不支持它，所以我们也就不对其进行探讨了。剩下的只有两种选择：浮动元素或绝对定位。这两种方法都各有优缺点，如果你想做一个全长度页脚，但事先又不知道哪一栏最长的话，你就必须用浮动元素来保证页面设计的完整性。

浮动元素的问题在于，它们只能一直向左或向右移动，直到碰到父区块或另一个浮动元素的边沿为止。这意味着浮动栏在 HTML 标记中的顺序必须正确。但有时我们希望显示顺序跟源顺序不同。比如说，你可能想把目录放在导航之前，来增加键盘导航的易用性，以及改进搜索引擎优化。

1. 将下面的代码拷到文本编辑器中，另存为 `layout.html`。

```
2. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
3. "http://www.w3.org/TR/html4/strict.dtd">

4. <html lang="en">

5. <head>

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

7. <title>Static and Relative Positioning</title>

8. <link rel="stylesheet" type="text/css" href="layout.css">

9. </head>

10.

11. <body>

12. <div id="header">Header</div>

13. <div id="main">Main content</div>

14. <div id="sidebar">Sidebar</div>

15.

16. <div id="nav">Navigation</div>

17. <div id="footer">Footer</div>

18. </body>

19. </html>
```

19. 接下来，我们来创建一个样式表的雏形。将下面代码拷到文本编辑器中，另存为

`layout.css`。

```
20. #header {
21. background-color: #369;
22. color: #fff;
23. }
24.
25. #sidebar {
26. background-color: #ff6;
27. }
28.
29. #nav {
30. background-color: #ddd;
31. }
32.
33. #footer {
34. border-top: 1px solid #369;
35. }
36.
```

35. 保存这两个文件，然后在浏览器中加载页面。可以看到五个部分从上到下依次排列。

假设你所在的设计部门明确要求导航栏在页面左侧，工具栏在页面右侧，而主要内容栏在页面中间。页眉和页脚应该是跨越整个页面宽度的，但我们不知道三栏中到底哪一栏会是最长的。页面元素的源顺序完全由辅助性和可用性专家们决定，而且没有讨价还价的余地。你该怎样将所有这些要求融合到一个有效的布局中去呢？

为了满足这些要求，你只有向 **HTML** 标记中添加一个附加元素。这是不可避免的，但你应该能忍受多一个附加元素。你需要一个元素来容纳这三个“栏”。

36. 把下面高亮的两行代码插入到前面的 **HTML** 文档中：

```
37. <div id="header">Header</div>
38. <div id="wrapper">
39. <div id="main">Main content</div>
```

```
40. <div id="sidebar">Sidebar</div>
41.
42. <div id="nav">Navigation</div>
43. </div>
44.
45. <div id="footer">Footer</div>
```

设计师们（幸运的是他们了解页面亲和力与设备独立性）约定了导航栏占 **12em** 宽，而工具栏占 **14em**。因为固定宽度的布局界面的用户友好性不佳，主要内容栏宽度应该是不固定的，以便适应各种窗口宽度。为了避免文本行过长，妨碍可读性，你需要强制设定该布局约束的最大宽度。同样的，为了避免在极端窄小的窗口中产生重叠现象，你也得强制设定该布局的最小宽度。除了上述约束条件，该布局还应该在浏览器窗口内水平居中。

**44.** 接下来，将下面的规则添加到 CSS 文件的末尾，为导航栏和工具栏设定宽度值，并设置宽度约束和整体居中：

```
45. body {
46. margin: 0 auto;
47. min-width: 40em;
48. max-width: 56em;
49. }
50.
51. #sidebar {
52. width: 13em;
53. padding: 0 0.5em;
54. background-color: #ff6;
55. }
56.
57. #nav {
58. width: 11em;
59.
60. padding: 0 0.5em;
61. background-color: #ddd;
```

```
}
```

62. 保存文件并重新加载页面——你应该能看到黄色的工具栏和灰色的导航栏元素的宽度符合你的要求。如果浏览器窗口足够大的话，你还可以看到整个页面在宽度上受到了约束，而且页面是水平居中的。

63. 改变窗口大小，看看该布局是如何适应窗口改变的。

注意：如果你用的是微软的 IE6 或更早版本的浏览器，你将看不到任何宽度约束效果。因为这些版本的 IE 不支持最小和最大宽度（或高度）。在本例结尾的部分我们来看看这个问题的解决办法。事实上，即使是在 IE7 中运行这个例子，也会产生许多奇怪的效果，因为 IE 浏览器有很多奇怪的渲染漏洞。下面我会着重从标准兼容方面来分析本例，然后在本文的末尾提出解决办法。

如果你仔细地研究代码的话，将会发现宽度从 14em 和 12em 变成了 13em 和 11em。这是因为水平填充距是必需的；你不会希望这些栏的内容与边沿紧紧地挤成一堆，这样的话就不美观了。因此宽度中加上了填充距， $13\text{em} + 0.5\text{em} + 0.5\text{em}$  加起来就等于你所要的 14em 了。

## 创建栏

好了，你的基础结构单元已经建好了，但它们还是竖直排列的。你想要的是三栏式的效果，因此现在我们来浮动这些元素。

1. 将下面代码添加到 CSS 文件中：

```
2. #main {
3. float: left;
4. }
5.
6. #sidebar {
7. float: left;
8. width: 13em;
9. padding: 0 0.5em;
10. background-color: #ff6;
11. }
12.
13. #nav {
```

```
14. float: left;
15.
16. width: 11em;
17. padding: 0 0.5em;
18. background-color: #ddd;
```

```
}
```

这段代码浮动了这些元素，但它们的顺序是错的。此外，主要内容栏也太窄了。还有，页脚到底是怎么了？

18. 我们先来处理一下页脚。页脚的问题在于三个栏都是浮动的，都从文件流中取出来了。页脚被挤得挨着页眉，而包含文本的线框被缩短了，使得“Footer”这个词紧挨在浮动元素的右边。你可以通过对页脚清除所有的浮动栏来纠正这个问题。将下面的代码添加到 CSS 文件中：

```
19. #footer {
20.
21. clear: left;
22. border-top: 1px solid #369;
```

```
}
```

22. 现在来对付那三个栏。这个问题得一步一步地解决，暂时效果会很难看，但是不要灰心——这个问题最终会获得解决的。

整个技巧的关键点就是包装元素。我们将对包装元素设置左边距和右边距，其值等于侧边栏（导航栏和工具栏）的宽度。主要内容栏将占据整个包装元素的宽度，而侧边栏将移动到由边距腾出来的空间去。听起来是不是有点复杂？别担心，我会带着你一点一点地前进。首先，我们对包装元素设置边距，将下面规则添加到 CSS 文件中：

```
#wrapper {
 margin: 0 14em 0 12em;
 padding: 0 1em;
}
```

别忘了 margin 属性值的速记顺序是 **TRouBLe order: top, right, bottom, left**。

我们将顶部和底部边距设为 0，右边距设为 14em（工具栏），左边距设为 12em（导航栏）。

还要添加 1em 的水平填充距，因为你不会希望内容栏跟侧边栏挤在一起，它需要一点喘息的空间。

23. 接下来要让主要内容栏占据父包装元素的整个宽度；下面的代码也会暂时给内容栏设置一个鲜艳的背景颜色：

```
24. #main {
25. float: left;
26. width: 100%;
27. background-color: lime;
}
.....
```

28. 保存并重新加载——你将会看到一个鲜艳的石灰绿的内容栏，而工具栏和导航栏在它下方。你也会注意到内容栏两侧有许多空白。我们的技巧就在于将侧边栏塞进这个空白空间去。

再接下来我们来对付工具栏——它是浮动的，而且宽度是正确的，但由于#main 栏占据了 100% 的宽度，工具栏就被挤到下面去了。怎样才能在#main 栏占据全部宽度的情况下让工具栏上升，并放在#main 样式旁边呢？我们分成两个小步骤来做：首先，把工具栏上移；然后把它移到边距中去。

29. 下面我们将用一个妙招来对付浮动的工具栏，它本来是被挤到下面的，现在我们要让它重新搬上去——将下面新增的代码添加到#sidebar 规则中：

```
30. #sidebar {
31. float: left;
32. width: 13em;
33. padding: 0 0.5em;
34. background-color: #ff6;
35. margin-left: -14em;
}
.....
```

36. 保存并重新加载页眉，你可以看到工具栏现在跟内容栏处在同样的垂直层面上了。通过设置一个与工具栏宽度大小相等的负左边距，我们就可以将工具栏元素移回到包装元素中去，这样它就不会被挤下去了。但这里又出现了一个问题，工具栏与内容栏发生了部分重叠。

37. 你想要的是将工具栏转移到边距那里去，不要再掉下来，在这里相对定位——终于——要登场了。它可以精确地达到我们想要的效果：只移动生成的盒模型，而不移动元素本身。将下面高亮的属性值添加到#sidebar 规则中：

```
38. #sidebar {
39. float: left;
40. width: 13em;
41. padding: 0 0.5em;
42. background-color: #ff6;
43. margin-left: -14em;
44. position: relative;
45. left: 15em;
 }
 }
```

别忘了你应该将工具栏移动 **15em**，而不是 **14em**——因为包装元素还有 **1em** 的右填充距。现在工具栏在它该在的地方了：在边距区域中，紧靠着内容栏，与页眉和页脚的右边沿精确地排成一线。

46. 现在要做的是对导航栏进行同样的操作，虽然用的是一样的方法，但导航栏有自己独特的问题。工具栏的挪动和位移比较简单，移动距离实质上等于其自身的宽度：**14em** 的负边距，然后向右位移 **14em+1em**。但是导航栏得穿越整个内容栏，往边距中位移更多的距离。这回我们得使用百分比了。导航栏边距的百分比值是和它的父元素，也就是包装元素成比例的。你应该使导航栏穿越整个包装盒模型——将下面高亮的属性值添加到#nav 的规则中去：

```
#nav {
 float: left;
 width: 11em;
 padding: 0 0.5em;
 background-color: #ddd;
 margin-left: -100%;
 }
 }
```

47. 变！再保存和重新加载一下，你应该能看到导航栏与内容栏的左侧重叠了。现在你只需要将导航栏移到边距中去就可以了。将下面高亮的属性值添加到#nav 的规则中去：

```
48. #nav {
49. float: left;
50. width: 11em;
51. padding: 0 0.5em;
52. background-color: #ddd;
53. margin-left: -100%;
54. position: relative;
55. right: 13em;
```

```
}
```

再说一遍，导航栏的宽度是 **12em**，但因为包装元素还有 **1em** 的填充距，所以你得将导航栏盒模型位移 **13em**。你要将它向左移动，也就是说从右边沿移开，这就是为什么我们用的是 **right** 属性。

56. 删掉内容栏的石灰绿背景，这个布局就做好了。

### 对付 IE 中出现的奇怪问题

有两个原因会导致这个布局在 Windows 的 IE6 中失效。一个是因为 IE6 不支持 **min-width** 和 **max-width** 属性，另一个原因是 IS 在百分比方面实在是太差了。

你可以用微软的私有标记 **expression()** 来模拟宽度约束。该标记以一个 **JScript** 表达式作为其自变量，并返回该表达式的返回值。每当浏览器需要获取 **body** 的宽度值时都得求解该表达式，因此如果该表达式的运算量很大的话就会导致效率问题。该标记还需要启用 **JScript**，不过你可以添加故障弱化，也就是说，即使 **JScript** 不可用，你的设计也可以回落到一个仍然可用的状态。在本例中，如果 **JScript** 被禁用的话，你可以将该布局设置为完全弹性，而不是上面创建的那种带约束的可变设计。

建议在样式规则中使用“条件注释”来作为 IE 浏览器的缺陷补丁。在 **HTML** 注释中嵌入条件逻辑（在 [dev.opera.com](http://dev.opera.com) 上有专门讨论条件注释的文章）是微软独有的特色。

1. 将下面行插入 **HTML** 代码，放在 **</head>** 标签之前：

```
2. <!-- [if lte IE 6]>
3. <link rel="stylesheet" type="text/css" href="layout-ie6.css">
```

4.

```
<! [endif] -->
```

5. 然后，创建一个新文件，加入下面的内容，并将其命名为 `layout-ie6.css`:

```
6. body {
7. width: 50em;
8. width: expression(w=document.documentElement.offsetWidth,
9. em=document.getElementById("nav").offsetWidth/12, (w<40*em?"40em"
10. w>56*em?"56em":"auto"));
11. }
12. #wrapper {
13. height: 1em;
14. }
15. #nav {
16. margin-left: -22em;
17. margin-left:
18. expression((- (document.getElementById("wrapper").clientWidth)+"px"));
19. left: 13em;
20. }
21. }
```

这样就可以解决IE6 的两个问题了。我们通过JScript表达式模拟IE6 不支持的`min-width`和`max-width`属性，还设置了一个 `50em`的弹性回落值。然后通过另一个JScript表达式将左边距单位设置为像素，而不是百分比，同样也设置了弹性回落值。`#wrapper`的高度的作用只是触发微软特有的`hasLayout`，这个属性是保证导航栏元素相对定位的正常运转所必需的。微软在MSDN上提供了`hasLayout`的文件记录，但这份文档不是轻轻松松就能读懂的。

那么IE7 又如何呢？IE7 的确支持`min-width`和`max-width`，但它仍然将导航栏元素放错了位置——跟IE6 相同的`hasLayout`漏洞又出现了。你得在`#wrapper`元素中触发属性。幸运的是，你可以通过一种不损害标准兼容浏览器的方法来解决这个问题，因此可以不必创建单独的IE7 样式表；你只需要加入下面的规则来控制包装盒模型就可以了：

```
#wrapper {
 margin: 0 14em 0 12em;
 padding: 0 1em;
 min-height: 1em;
}
```

通过对最小高度的设置，触发了 `hasLayout` 属性，而且这种做法不会在其它浏览器下引起问题，因此这段代码可以直接添加到主样式表中。

这些解决办法并不是完美无瑕的；在 IE6 和 IE7 中，当我们将浏览器窗口的大小调整到某个值时，布局效果还是会产生成奇怪的问题，不过只要重新加载一下页面，布局就恢复正常了。

### 相对定位的其它应用

相对定位的常见用法根本不会牵涉到移动生成的盒模型。这听起来可能有点奇怪：如果不想要对盒模型进行移动的话，你干嘛还要使用相对定位？这个问题涉及到绝对定位，因此答案将在下一篇文中揭晓。别换频道哦！

设置 `position: relative`（不用移动盒模型）对于 IE 中一些奇怪的渲染漏洞也有用。它可以对内部属性 `hasLayout` 进行设置，该属性深刻地影响着 IE 对页面元素的渲染。

### 总结

静态定位是元素定位的默认状态。块状盒模型会按照其源顺序纵向排列，而内联盒模型则是在块状盒模型内的线框中横向排列。

使用相对定位，你可以在一个或两个方向上移动某个生成的盒模型。元素仍会像静态时一样占有空间，但生成的盒模型可以被移动到另一个地方。当显示顺序和源顺序不同时，相对定位与浮动联合使用可以有效地创建布局。

### 练习题

- 当同一静态格式化范围内的两个相邻边距合并时，如果其中一个边距——或者两个边距——是负数，这时会产生什么现象？
- 在两个侧边栏之间添加竖直边框和主要内容栏。别忘了三栏都是浮动的，因此包装元素的高度缩成了零。
- 如何使所有栏的高度一致（或者至少看起来一样），从而使背景颜色向下延伸至页脚处，无论最长的是哪一栏？（提示：在你喜欢的搜索引擎中查找“伪列布局”。）
- 上一篇：浮动及清除

- [下一篇：CSS 的绝对和固定定位](#)
- [目录](#)

作者简介



Tommy Olsson 是一位 web 标准和网页亲和力的实用主义倡导人，他住在瑞典中部。

Tommy Olsson 在 1993 年就写出了自己的第一份 HTML 文档，现在他在一家瑞典政府机构任专业 web 站点管理员。

迄今为止他已经写了一本书——CSS 终极参考大全（与 Paul O'Brien 一起完成的）——还有一个令人遗憾地被忽视了的博客 [The Autistic Cuckoo](#)。

## 37. CSS 的绝对和固定定位

Posted 12/15/2008 - 16:59 by Lewis

- 网络标准教程

作者: Tommy Olsson · 2008 年 9 月 26 日

- 上一篇: CSS 的静态和相对定位
- 下一篇: 页首、页脚、列和模板
- 目录

### 引言

现在该将你的注意力转向第二对position属性值——absolute和fixed了。第一对属性值——static和relative——是密切相关的，这两个属性值我们已经在上一篇文章中非常详细地讨论过了。

绝对定位的元素不再包含于文件流之中。这意味着它们对自己的父元素或源代码中位于自己后面的其它元素完全没有任何影响。因此，一个绝对定位的元素将重叠在其它页面内容上，除非你采取行动来阻止它。当然，有些时候，这种重叠正是你想要的，但你必须小心，以确保自己能得到想要的布局。

固定定位实际上只是绝对定位的特殊形式；固定定位的元素是相对于视窗/浏览器窗口而固定，而不是相对于其包含元素；即使页面滚动了，它们仍然会处在浏览器窗口中跟原来完全一样的地方。

在本文中，我们将会看到用到 absolute 和 fixed 的一些实际例子，还会研究某些浏览器支持缺陷，以及对 z-index 概念进行探索。

本文结构如下：

- 包含区块
- 绝对定位
  - 指定位置
  - 指定大小
  - 第三维度—z-index
    - 局部堆叠环境

- 固定定位
- 总结
- 练习题

在讨论这些内容之前，我要先讲讲一个非常重要的前提概念——包含区块。

## 包含区块

在谈到绝对定位时，有一个非常重要的概念，这就是包含块：与绝对定位盒模型的位置和大小相关的块状盒模型。

对于静态盒模型和相对定位盒模型来说，其包含区块就是最近的块级先祖——也就是父元素。然而对绝对定位的元素来说，这个概念要稍微复杂一点。在这种情况下一个元素的包含区块是其最近的已定位先祖。我用“已定位”这个词的意思是指 `position` 属性已经设置为 `relative`, `absolute` 或 `fixed` 的元素——也就是说，除了标准静态元素之外的其它所有元素。

因此，通过对一个元素设置 `position:relative`，你就可以使它成为任何绝对定位的派生元素（子元素）的包含区块，无论这些元素在文件层级中是处在相对定位元素的下一层，还是处在文件层级的更下层。

如果一个绝对定位的元素没有已定位的祖先元素，那么它的包含区块就是所谓的“源容器区块”，实际上就等同于 `html` 元素。当你在屏幕上浏览网页时，该包含区块就是浏览器窗口；在打印网页时，就是页面边界。

对于固定定位元素来说这个概念要稍微有点区别——它们的包含区块始终是源容器区块。

我们用一串简单步骤来总结一下——为了找出设置了 `position:absolute` 的元素的包含区块，你只需按照下面的步骤进行：

1. 看看这个绝对定位元素的父元素——它的 `position` 属性值是 `relative`, `absolute` 或 `fixed` 之中的一个吗？
2. 如果是，那它就是你要找的包含区块。
3. 若不是，再查看该父元素的父元素，重复第一步的操作，直到你找到了该元素的包含区块，或者遍历完所有祖先元素。
4. 如果你已经上溯到 `html` 元素，还是没有找到任何已定位的祖先元素，那么该元素的包含区块就是 `html` 元素。

## 绝对定位

由于相对定位是绝对定位的特殊形式，因此我们稍后再讲它，现在先来看看更广义的情况。

除非另作说明，在本篇文章中，凡用到“绝对定位的”这个词的地方，都是指 `position:fixed` 的元素，以及 `position:absolute` 的元素。

### 指定位置

你已经了解到，在相对定位中可以用 `top`, `right`, `bottom` 和 `left` 属性来指定盒模型的位置。虽然在绝对定位中也是用同样的属性来指定一个绝对定位的元素的位置，但是它们的使用方法完全不同。

对于一个相对定位的元素来说，这四个属性指定了移动所生成的盒模型的相对距离。别忘了在相对定位情况下它们可以互为补充，比如说 `top:1em` 和 `bottom:-1em` 就是一回事，同时给一个元素指定 `top` 和 `bottom`（或 `left` 和 `right`）没什么意义，因为这两个属性值之中的一个会被忽略掉。

但在绝对定位情况下这些概念就不管用了。此时，四个属性可以同时用起来，来指定被定位的元素的四边到包含区块的对应边的距离。你也可以指定绝对定位的盒模型的某一个角的位置——比方说通过 `top` 和 `left` 来指定——然后再通过 `width` 和 `height`（如果你想让它收缩包围来适应自身内容的话，也可以不指定 `width` 和 `height`）来指定该盒模型的大小。

微软的 IE6 及更早版本的浏览器不支持同时指定四边距离的方法，但能支持先指定一角再指定大小的方法。

```
/* This method works in IE6 */

#foo {
 position: absolute;
 top: 3em;
 left: 0;
 width: 30em;
 height: 20em;
}

/* This method doesn't work in IE6 */

#foo {
 position: absolute;
 top: 3em;
 right: 0;
```

```
bottom: 3em;
left: 0;
}
```

需要记住的是，`top`, `right`, `bottom` 和 `left` 属性的值指定的是该元素的边到其包含区块的对应边的距离。而不是像坐标系中那样，每个值都是相对于同一个原点的距离。比如说，`right:2em` 就是指绝对定位的元素的右边到其包含区块的右边的距离是 `2em`。

当你用到绝对定位的时候，最要紧的就是要弄清楚你的包含区块是哪个。所以将你的包含区块设置为 `position:relative` 非常管用，即使你实际上并不想移动该盒模型的位置。这样你就可以将一个元素变成其绝对定位的派生元素的包含区块了——而这会带给你更多的支配权。

为了搞清楚绝对定位是怎样工作的，我们来看一个例子。

1. 将下面代码拷到你的文本编辑器中，另存为 `absolute.html`。

```
2. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

3. <html>

4. <head>

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

6. <title>Absolute Positioning</title>

7. <link rel="stylesheet" type="text/css" href="absolute.css">

8.

9. </head>

10. <body>

11. <div id="outer">

12. <div id="inner"></div>

13. </div>

14. </body>

15.

16. </html>
```

16. 接下来，新建一个文件，把下面代码拷进去，另存为 `absolute.css`。

```
17. html, body {
18. margin: 0;
19. padding: 0;
20. }
21.
22. #outer {
23. margin: 5em;
24. border: 1px solid #f00;
25. }
26.
27. #inner {
28. width: 6em;
29. height: 4em;
30. background-color: #999;
```

```
}
```

31. 保存这两个文件，将 HTML 文档载入你的浏览器中。你将看到一个灰色的矩形，它的周围还有稍微宽一点的红色边框。`#inner` 元素的宽度和高度都是指定的，它的背景颜色是灰色的。`#outer` 元素是`#inner` 元素结构上的父元素，那个红色边框就是它的。在它周围还有 5em 的边距，用来将其从浏览器窗口边缘移开，好让我们更清楚地看到这段代码的效果。

目前为止也没什么好惊奇的，对吧？`#outer` 元素的高度是由其子元素（`#inner`）决定的，而宽度是由水平边距决定的。

32. 如果我们把`#inner` 设成绝对定位又会怎么样？等着瞧！将下面高亮的声明加到`#inner` 的规则去中：

```
33. #inner {
34. width: 6em;
35. height: 4em;
36. background-color: #999;
37. position: absolute;
```

```
}
```

38. 保存，重新加载页面。现在看起来可不像是灰色矩形周围的红色边框了，更像是一个粗粗的顶部边框。而且灰色的方块根本没有移动！这跟你的预期一样吗？

这里有两个有趣的现象。首先，因为`#inner`被设成了绝对定位，它就完全脱离了文件流。这就意味着其父元素，也就是`#outer`，在常规文件流中不再有子元素了，因此其高度塌缩成了零。那个看起来有`2px`宽的线条其实只是零高度元素外面的`1px`边框——你看到的只是顶部边框和底部边框，而它们中间什么也没有。

第二件有意思的事情是，绝对定位的盒模型并没有发生移动。`top`, `right`, `bottom` 和 `left` 属性的默认值是`auto`，这就意味着这个绝对定位的元素的位置仍然跟没有定位时完全一样。由于该元素不再包含于文件流中，它就会重叠在常规文件流中原来位于它后面的所有元素上。

这实在是非常有用——如果你想让某个生成的盒模型只朝一个方向移动的话。比如说，在一个 CSS 编写的下拉菜单中，“下拉”面板的绝对定位就可以只指定 `top` 属性。这样它们就可以沿着 X 轴显示在你想要的坐标位置上（跟其父元素的 X 坐标一样）。

39. 接下来，我们为`#outer`元素设置一个高度，让它重新成为一个矩形，并将`#inner`元素移到旁边去。将下面高亮的代码行添加到 CSS 规则中：

```
40. #outer {
41. margin: 5em;
42. border: 1px solid #f00;
43. height: 4em;
44. }
45.
46. #inner {
47. width: 6em;
48. height: 4em;
49. background-color: #999;
50. position: absolute;
51. left: 1em;
```

```
}
```

52. 保存并重新加载页面，你将看到一些变化。`#outer` 元素现在又成了矩形了，因为你为它设置了高度。`#inner` 元素移到了旁边，但却不是你所预期的地方。它的左边缘并没有距离父元素的左边框 `1em`，反倒是离浏览器窗口的左边缘 `1em`！

就像前面解释过的那样，这个问题的原因在于，`#inner` 并没有已定位的祖先元素，因此它的包含区块就是源容器区块。如果你指定了除 `auto` 以外的属性值，那么它相对的就是包含区块的对应边。如果你设置了 `left:1em`，`#inner` 的左边缘就会距离浏览器窗口左边缘 `1em`。

53. 如果你想把它变成距离其父元素的左边缘 `1em`，就必须将该元素的父元素变成它的包含区块。为了达到这个目的，你就得用到我在本文前面提到的小伎俩——将父区块设为相对定位。将下面高亮的代码行添加到`#outer` 规则：

```
54. #outer {
55. margin: 5em;
56. border: 1px solid #f00;
57. height: 4em;
58. position: relative;
 }
.....
```

59. 保存并重新加载——你瞧！灰色的方块现在距离其父元素的左边框 `1em` 了。在`#outer` 规则中设置 `position:relative` 使得该元素成为已定位元素，并且将它变成了它自身所有绝对定位的派生元素的包含区块。为`#inner` 设置的 `left:1em` 现在是从`#outer` 的左边缘算起，而不是从浏览器窗口的左边缘算起。

### 指定大小

如果你没有指定其大小，绝对定位的元素就会收缩包围来适应自己所包含的内容。你可以通过对 `left` 和 `right` 属性，或者 `width` 属性进行设置来指定元素的宽度，而高度可以通过 `top` 和 `bottom` 属性，或 `height` 属性来指定。

这六个属性都可以用百分比来指定。从本质上讲，百分比就是指与其它元素的比例。在本例中，百分比就是指与包含区块的大小的比例。

对于绝对定位的元素来说，`left`, `right` 和 `width` 属性的百分比值与包含区块的宽度相关。`top`, `bottom` 和 `height` 属性的百分比值则是与包含区块的高度相关。

在 IE6 及更早版本的浏览器，以及 opera8 和更早的版本中，把百分比计算完全搞错了，它们用的是父区块的大小来进行计算。我们用另一个例子来看看这种错误会造成什么样的差别。

- 首先用百分比形式的属性值来指定#inner 元素的大小——将#inner 规则修改成下面的样子：

```
2. #inner {
3. width: 50%;
4. height: 50%;
5. background-color: #999;
6. position: absolute;
7. left: 1em;

8. }
```

- 保存，重新加载，你会看到灰色方块变宽了，也变短了（至少在最新的浏览器上是这样）。包含区块仍然是#outer 没有变，因为它设置了 position:relative。#inner 元素的宽度是#outer 宽度的一半，高度也是#outer 高度的一半。

- 我们再来把视窗改成包含区块，看看有什么不同！对#outer 做如下改动：

```
10. #outer {
11. margin: 5em;
12. border: 1px solid #f00;
13. height: 4em;
14. position: static;

15. }
```

- 保存，刷新——大不一样了，是吧？灰色方块现在有浏览器窗口的一半那么宽了。

正如你看到的一样，搞清楚你的包含区块是哪一个，这是绝对必要的！

### 第三维度——z-index

我们会很自然地把网页看作是二维的。现在的技术还没有发展到能使 3D 显示随处可见的地步，所以我们只能接受只有宽度和高度，以及仿 3D 效果的现状。但 CSS 渲染实际上是在三个维度上进行的！这并不是说你可以让某个元素悬浮在显示器的前方——现在还不能这样——而是说已定位的元素还有其它很实用的用途。

网页上最主要的两个坐标轴就是水平方向的 X 轴和垂直方向的 Y 轴。这个坐标系的原点位

于窗口的左上角，在那里 X 和 Y 值都为 0.

但是 Z 轴也是存在的，我们可以将它想象成垂直于显示器的表面（在打印时就是垂直于纸面）。Z 值较高的元素将位于 Z 值较低的元素的“前面”。Z 值也可以是负数，代表元素位于某个参照点的“后面”（我马上就来讲这种参照点）。

在我们继续进行之前，我得提醒你一下，这是 CSS 中最复杂的概念之一，所以如果你第一遍没有读懂的话也用不着灰心。

已定位的元素（包括相对定位元素）是在一种所谓的堆叠环境之中渲染的。在同一个堆叠环境中的元素在 Z 轴上的参照点是相同的。下面我会更详细地讲解这一点。你可以通过 `z-index` 属性来改变一个已定位的元素在 Z 轴上的位置（也叫做层叠级别）。该属性的值可以是整数（也可以是负整数），也可以是 `auto` 或 `inherit` 两个关键字之一。默认的属性值是 `auto`，表示该元素与自己的父元素的层叠级别相同。

Y 必须要注意，你只能指定元素在 Z 轴上的索引位置。你不能让一个元素出现在另一个元素后面 19 像素，或在它前面 5 厘米的地方。你可以把这些元素想象成一副牌，你可以洗牌作弊，并且使黑桃 A 位于方块三的上面——每张纸牌都有自己的层叠级别，或者说 Z index.

如果你将 `z-index` 指定为一个正整数，就是赋给这个元素一个层叠级别，表示在同一堆叠环境之中，它会位于比它的层叠级别低的其它元素的“前面”。`z-index` 值为 0（零）的意义与 `auto` 相同，但它们之间存在差异，我马上就会谈到这一点。如果 `z-index` 的值为负整数，就表示该元素的层叠级别在其父元素的层叠级别“之后”。

如果同一堆叠环境之中的两个元素的层叠级别相同，源代码中较晚出场的那个元素就会位于较早出现的那个元素之上。

实际上一个堆叠环境可以有多达七个层级，不过别担心——你不必处理一个堆叠环境的全部七个层级。同一个堆叠环境中元素（所有元素，而不仅仅是已定位的元素）的渲染顺序如下，下面列表中越靠前的元素在渲染时越排在后面：

1. 该堆叠环境中元素的背景和边框
2. 层叠级别为负数的已定位的派生元素
3. 常规文件流中的块级派生元素
4. 浮动的派生元素
5. 常规文件流中的内联派生元素
6. 层叠级别为 `auto` 或 0（零）的已定位的派生元素
7. 层叠级别为正数的已定位的派生元素

加了高亮的条目代表那些我们可以通过 `z-index` 属性来更改其层叠级别的元素。

这整个概念实在是难以理解，因此我们实际动手来做做实验，以便弄清楚 `Z-index`。

- 首先将下面高亮的代码行添加到你那小小的实验文档中：

```
2. <body>
3. <div id="outer">
4. <div id="inner"></div>
5. <div id="second"></div>
6.
7. </div>
```

8. </body>

- 然后再改改你的 `CSS`，让`#outer` 成为包含区块，并且将`#inner` 的大小用非百分比来表示。我们把`#outer` 设高一点，以便留出更多的空间来进行实验。按照下面高亮的代码行来改你的 `CSS` 规则：

```
9. #outer {
10. margin: 5em;
11. border: 1px solid #f00;
12. height: 8em;
13. position: relative;
14. }
15.
16. #inner {
17. width: 5em;
18. height: 5em;
19. background-color: #999;
20. position: absolute;
21. left: 1em;
```

}

22. A 为#second 元素添加规则:

```
23. #second {
24. width: 5em;
25. height: 5em;
26. background-color: #00f;
27. position: absolute;
28. top: 1em;
29. left: 2em;

30. }
```

30. 保存, 刷新, 你会看到一个鲜蓝色方块重叠在灰色方块上。这两个盒模型具有相同的层叠级别 (`auto`, 也就是初始值, 即层叠级别为 0), 但蓝色的盒模型被渲染为处于灰色盒模型的前面, 因为它在源代码中出现得更晚。你可以也让灰色盒模型跑到蓝色盒模型的前面去, 只需要赋给它正层叠级别就可以了。只需要将 `z-index` 值设成大于 0 就可以了——没必要那么过火地给它设一个 10000 之类的值。将下面高亮的代码行添加到#inner 规则:

```
31. #inner {
32. width: 5em;
33. height: 5em;
34. background-color: #999;
35. position: absolute;
36. left: 1em;
37. z-index: 1;

38. }
```

38. 保存, 重新加载页面, 现在灰色方块跑到蓝色方块前面去了。

## 局部堆叠环境

本章接下来的部分将讨论局部堆叠环境。除非你打算用绝对定位来做非常高级的效果, 在平常的设计工作中你也许根本不会碰到这个概念, 但是为了完备起见, 我还是会讲一讲它。如果你不想了解的话可以自行跳过这一部分。

每个 `z-index` 值为整数的元素都会建立一个新的，“局部”堆叠环境，在这种堆叠环境中，该元素自身的层叠级别为 0。这就是我上面说过的 `z-index: auto` 和 `z-index: 0` 之间的差别。在这两个值之中，前者不会建立新的堆叠环境，但后者会。

如果某个元素建立了局部堆叠环境，它的已定位的派生元素的层叠级别就只能在该局部环境中起作用了。这些派生元素可以相对于彼此重新堆叠，也可以相对于其父元素重新堆叠，但不能相对于其父元素的同级元素重新堆叠。就像是派生元素被圈在了父元素的“笼子”里一样，它们不能从其中逸出。派生元素可以在这个笼子里上下移动，但不能跑到笼子外面去。父元素和其子元素将在包围该父元素的堆叠环境中成为一个不可分割的单元。

你可以想象自己正在整理文书，以便将它们交到为你报税的会计师手中。你将费用报表，收据，确认订单以及类似的东西一张重一张地叠成一叠——为了让你的会计师轻松一点儿，你将每一类的文书分别装在不同的信封里。

局部堆叠环境与这种信封类似。它将相关的元素聚在一起，并防止其它元素插入到它们中间。你可以对每个信封内的内容进行排序，想怎么排都可以，但是这种排列顺序只限于每个信封内部，对整叠文书没有影响。现在你那一堆文件中包含了单张文书的组合（层叠级别为 `auto` 的元素），以及信封（层叠级别为整数的元素）。正层叠级别的信封在这叠文书的上面，而负层叠级别的信封在一叠文书的下面。

每当你给某个元素的 `z-index` 属性赋以整数值，你就创建了一个包含该元素及其子元素的“信封”。

我们来看看局部堆叠环境是怎样工作的。这看起来可能会让人有点晕，但其实跟前面已经学过的那些内容没有太大的差别。如果你照着这个例子来操作，就可以体验一下局部堆叠环境是怎样运作的了。

1. 首先给两个 `inner` 元素添加一些规则——将下面高亮的代码行添加到你的 HTML 文档中：

```
2. <div id="inner">
3.
4.
5. </div>
6. <div id="second">
7.
```

```
</div>
```

8. 添加一条 CSS 规则，使其可以同时应用到两个 span 元素上：

```
9. span {
10. position: absolute;
11. top: 2em;
12. left: 2em;
13. width: 3em;
14. height: 3em;

 }
```

这段代码将 span 元素设为绝对定位，并指定了它们的位置和大小。但是请等一等——span 元素是内联元素——你怎么能给内联元素指定大小呢？答案就是绝对定位的元素就像浮动元素一样，会自动生成区块盒模型。

这里我们指定的位置是相对于每个 span 元素的包含区块的。由于两个 span 元素的父元素都是绝对定位的 div 元素，这两个父元素就扮演了包含区块的角色。

15. 现在我们给 span 元素添加一下颜色，以便看清楚它们的显示位置，将下面规则添加到你的样式表中：

```
16. #inner span {
17. background-color: #ff0;
18. }
19.
20. #second span {
21. background-color: #0ff;

 }
```

22. 保存，刷新，你将在那个较大的灰色方块的右下角看到一个黄色方块，而较大的蓝色方块的右下角还有一个青色方块。灰色方块和黄色方块位于蓝色方块和青色方块的前面，因为灰色方块的设置是 `z-index:1`。

23. 如果你想让青色方块跑到所有方块的前面去，该怎么办呢？你只需要将它的层叠级别设得比灰色方块高就行了。实际上将它的层叠级别设成跟灰色方块一样就够了，因为青色方块在标记中出现得更晚。我们来试一试——对你的 CSS 做如下改动：

```
24. #second span {
25. background-color: #0ff;
26. z-index: 1;
27. }
```

27. 保存并重新加载页面。如果你的浏览器对 CSS 规范的支持到位的话，青色方块现在就应该处在最前面了。

灰色方块的设置是 `z-index: 1`，这意味着它将建立一个局部堆叠环境。也就是说，你已经创建了一个“信封”，并且将灰色方块和它的黄色子方块放了进去。

还没搞清楚吗？那我们就通过下一个实验来进一步地讲解这个问题。

1. 对黄色的方块设置更高的层叠级别，让它跑到最前面去——对你的 CSS 做如下改动：

```
2. #inner span {
3. background-color: #ff0;
4. z-index: 4;
5. }
```

5. 保存并重新加载页面，你会看到……没有任何改变！我们为黄色方块设置的层叠级别只有在灰色方块所建立的堆叠环境中才能有效——黄色方块与它的灰色父元素位于同一个信封之中。之所以你能将青色方块置于最前面，是因为它的父元素（蓝色方块）并没有建立自己的局部堆叠环境——它被隐式设置为 `z-index: auto`。这个蓝色方块就相当于文书堆中的单张文书。黄色和青色方块实际上是装在它们自己的小信封（它们的层叠级别是整数，因此它们建立了自己的局部堆叠环境）里的。

6. 如果你让蓝色方块也建立自己的局部堆叠环境，你就不能将青色方块移到最前面去了，除非你同时将它的父元素（蓝色方块）也移到最前面去。我们可以来试一试——对你的 CSS 做如下改动：

```
7. #inner {
8. }
```

```
9. ...
10.
11. z-index: 2;
12. }
13.
14. #second {
15.
16. ...
17.
18. z-index: 1;
19. }
20.
21. #second span {
22.
23. ...
24.
25. z-index: 3;
```

```
}
```

26. 保存并重新加载页面。现在灰色和蓝色方块都建立了自己的局部堆叠环境，给我们整了两个信封。在这个元素堆栈的最下面是层叠级别为 1 的信封，其中还包括两个内部信封（蓝色方块和青色方块）。元素堆栈的最上面是层叠级别为 2 的信封，其中还有两个内部信封（灰色和黄色方块）。在第一个信封里，蓝色方块的局部层叠级别为 0，因此它显示在青色方块的后面，因为青色方块的局部层叠级别为 3。在第二个信封里，灰色方块的局部层叠级别为 0，因此它显示在黄色方块的后面，因为黄色方块的局部层叠级别为 4。

图 1 显示了四个盒模型和两个局部堆叠环境沿 Z 轴方向的侧视图。

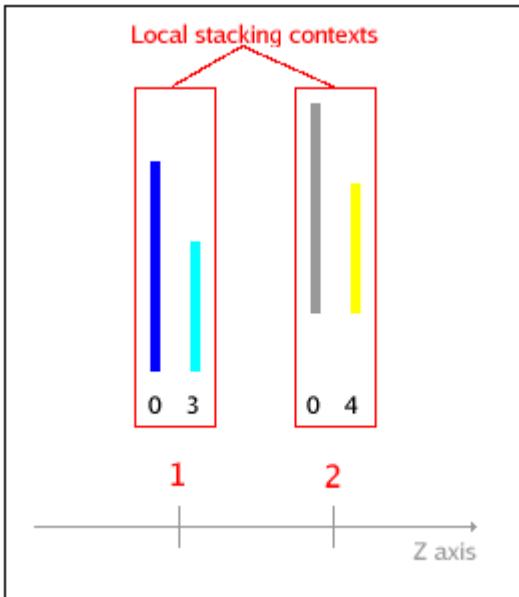


图 1：不同的堆叠环境的展示。“2”中的元素将始终显示在“1”中所有元素的前面。在每个堆叠环境中，`z-index` 值高的元素将出现在 `z-index` 值低的元素的前面。如果两个元素的 `z-index` 值相同，标记中出现得较晚的那个元素的位置将最靠前。

这一部分的内容可能是非常难以理解的，尤其是对 CSS 新手来说。重点是，如果你想要改变各种元素的层叠级别的话，就必须要了解你的堆叠环境。如果某个元素属于一个局部堆叠环境的话，你就只能改变它在该局部堆叠环境中的 Z 轴位置。一个局部堆叠环境中的元素是不能插入到另一个局部堆叠环境的元素中去的。

不过，幸运的是，你很少会碰到这些问题。对优良的布局来说，需要改动 `z-index` 值的情况是很少发生的，而且就算需要作改动，通常这种情况也只是发生在同一个堆叠环境之中。

### 固定定位

一个设置了 `position:fixed` 的元素是相对于视窗固定的。就算页面文档发生了滚动，它也会一直呆在相同的地方。如果设置 `media="print"`，固定定位的元素就会重复出现在每一页打印资料上相同的地方。

值得注意的是，IE6 及更早版本的浏览器压根不支持固定定位。如果你的浏览器属于上述系列的话，你将无法看到这一部分的例子的效果。

尽管 `position:absolute` 的元素的位置和大小是取决于其包含区块的，`position:fixed` 的元素的位置和大小却始终是取决于源容器区块的。源容器区块一般是视窗：也就是浏览器窗口或一张纸的页框架。

为了说明这一点，在下面的例子中我们会将一个元素设置为固定定位。为了使页面出现滚动条，我们将另一个元素的高度设得非常大，以便我们看到固定定位的效果。

1. 对你的 CSS 做如下改动：

```
2. #inner {
3. width: 5em;
4. height: 5em;
5. background-color: #999;
6. position: fixed;
7. top: 1em;
8. left: 1em;
9. }
10.
11. #second
12. width: 5em;
13. height: 150em;
14. background-color: #00f;
15. position: absolute;
16. top: 1em;
17. left: 2em;
```

```
}
```

18. 保存，刷新。如果你的窗口中并没有出现垂直滚动条，那就再把#second 的 height 值设大一点。（你到底用的什么巨型浏览器啊？）

长长的蓝色方块伸出了浏览器窗口底部。向下滚动页面，留心看左上角的灰色方块。元素现在固定定位在距浏览器窗口顶部和左边缘各 1em 的地方，因此在你滚动窗口的时候，灰色方块仍然位于屏幕上同样的地方。

## 总结

绝对定位的元素不再包含于文件流之中。除非你为它们腾出空间，它们就会重叠在其它页面内容上。如果某个容器的所有子元素都是绝对定位的，父元素的高度就会塌缩为零。

绝对定位的元素的定位与其包含区块相关，这里说的包含区块就是指该元素最近的已定位的祖先元素。如果已定位的祖先元素不存在，那么其包含区块就是视窗。

固定定位的元素是相对于视窗固定的——它们的包含区块就是视窗。在屏幕上浏览的时候，

它们总是显示在浏览器窗口内的同一个位置上；在打印的时候，它们会出现在每一页纸上。

绝对定位的元素每条边的位置可以通过 `top`, `right`, `bottom` 和 `left` 属性来指定。每个属性值指定的是它所代表的边到该元素的包含区块的对应边的距离。

所有已定位的元素都是在一个堆叠环境中按照某个层叠级别来渲染的。你可以通过 `z-index` 属性来改变一个已定位的元素的层叠级别。如果一个元素的 `z-index` 设置为整数值的话，该元素就会建立一个包含了自己的子元素的局部堆叠环境。

## 练习题

- 撤消对固定定位示例所做的改动，改变四个绝对定位的方块的堆叠顺序，使灰色方块位于最后面，然后依次是蓝色，黄色和青色方块。（建议：删除所有的 `z-index` 声明，然后再重新开始。）
  - 通过设置 `top:-1em` 和 `left:8em` 将黄色方块向右上方移动。然后使它显示在 `#outer` 元素的后面，从而使红色边框横穿黄色方块。
  - 用绝对定位来重做我们在 [静态定位和相对定位](#) 一文中创建的三栏式布局。由于在这里不可能做出全长度页脚，你可以删掉 `#footer` 元素，但不准改变标记中的其它任何东西（除了连到样式表的链接以外）。
  - 修改前一个练习中的布局，对导航栏实用固定定位。为了实现这个效果，你得删除 `body` 元素的自动水平边距。在主要内容栏和或侧边栏中添加足够多的内容来制造出滚动条，这样你就可以检验一下固定定位是否实现了。
- 
- 上一篇：[CSS 的静态和相对定位](#)
  - 下一篇：[页首、页脚、列和模板](#)
  - 目录

## 作者简介



Tommy Olsson 是一位 web 标准和网页亲和力的实用主义倡导人，他住在瑞典中部。Tommy Olsson 在 1993 年就写出了自己的第一份 HTML 文档，现在他在一家瑞典政府机构任专业 web 站点管理员。

迄今为止他已经写了一本书——CSS 终极参考大全（与 Paul O'Brien 一起完成的）——还有一个令人遗憾地被忽视了的博客 [The Autistic Cuckoo](#)。

## 38. 页首、页脚、列和模板

Posted 03/10/2009 - 23:00 by yuanc

- 网络标准教程

作者: Ben Henick · 2009 年 2 月 3 日

- 上一篇: CSS 布局模型——CSS 绝对定位与固定定位
- 下一篇: 程序设计——真正的基础!
- 目录

### 引言

迄今为止, 本教程中每篇文章只关注单一的主题, 涉及的话题从诸如排版和颜色之类的简单概念, 到关于 CSS2.1 子集的艰深技术说明。本篇文章所关注的范围要更广泛一些; 本文的目的在于向读者展示如何运用之前的文章所涵盖的知识, 并将其用来建立一个完整的网站模板。

阅读本文的重要前提, 是你已经熟悉了 CSS 的 `float`、`display` 和 `position` 属性。

我勉强建议那些希望接触 CSS 核心内容的自主的学习者直接跳到本文的第四部分, 即“单列布局的实现”——但是他们应当注意, 如果这么做的话, 就会遗漏掉关于优秀的项目规划是如何引导一个网站的布局和实施的讨论。

对于那些同样急切的, 无视前一段落中给出的警示说明的读者来说, 他们也会想要下载本文中提供的 [单列](#), [双列](#) 和 [三列式](#) 布局模板, 这些模板在各自的结论部分会再次给出链接。

本文结构如下:

- 样式设计师的关键性处理步骤
  - 目标采集
  - 内容分类
  - 协作草图与复合创意
  - 文档之间和文档内部结构的建立
- 对需求, 分类和命名空间进行更详细地论述
  - 业务目标定义
  - 访客目标的定义与满足
  - 内容分类

- 源顺序：可访问性与其它考虑
- 命名空间创建
- 单列式布局的实现
  - 布局的居中对齐
  - 全文档范围的容器是绝对必需的吗？
- 双列式布局的实现
  - 单列式布局和双列式布局的构图问题之比较
    - 不管源顺序如何，将 `#sidebar` 放在左侧
    - 伪列布局：在其内容长度不等时，利用背景图像来使列长度相等
    - 保持源顺序不变，将主导航条移至第二栏中
- 三列式布局的实现
  - 三列式布局设计中最易犯的错误，以及该问题的简单解决办法
- 博览页眉和页脚
  - 个人网站：Cindy Li
    - 标志
    - 对比
  - 社交网络：Facebook
  - 企业营销和客户服务：BNSF Railway
    - 再评标志
  - 页眉与页脚设计：初级细节
    - 更多关于实现导航布局的信息
- 列数不固定的网站：通过 `class` 和 `display` 来走捷径
- 总结
- 练习题
- 参考书目

注：你可以方便地下载所有示例代码的压缩包，以便在你的本地主机上进行实验。

即使网站的开发者是一个或两个人，而不是一整个专业团队，一个合理创建的网站也通常是源于一个深思熟虑的，大部分连续的过程。我们在图 1 中画出了对这种过程的一个相当详尽的阐释，在该图描述的这十个步骤中，本文尤其关注其中的四个步骤：

1. 基于业务目标以及随之而来的访客目标的需求采集

2. 内容分类
3. 协作草图与复合创意
4. 建立将要运用在该网站上的文档结构

这四个步骤一经完成，设计师就拥有了创建那种通常为单列，双列或三列式网站布局的绝大部分资料。不管布局的通式是什么，一个网站的某部分与另一部分之间都会有差异，这些差异反过来也会影响到特定元素和样式选择符是如何成为整个网站设计的一部分的。

即使在设计和构架上的选择已经做出之后，还是会面对这样的问题，就是如果该网站要建立在 Wordpress 和 Drupal 这样的内容管理系统（CMS）之上的话，该创建过程应如何进行处理。

本文强调了上面所提到的这四个步骤的重要性；为内容分类提供了一个简单的框架；还阐述了如何对一个具有完整的页眉，页脚和列的网站进行布局。

#### **样式设计师的关键性处理步骤**

接下来的两个章节旨在为本教程中的其它文章提供补充，这两个部分注重的是规划和工序，而不是实现。这两部分的核心思想是“三思而后行”——换言之，就是在你开始编写标记，样式表和代码之前，要对你到底要实现什么有个清楚的理解！

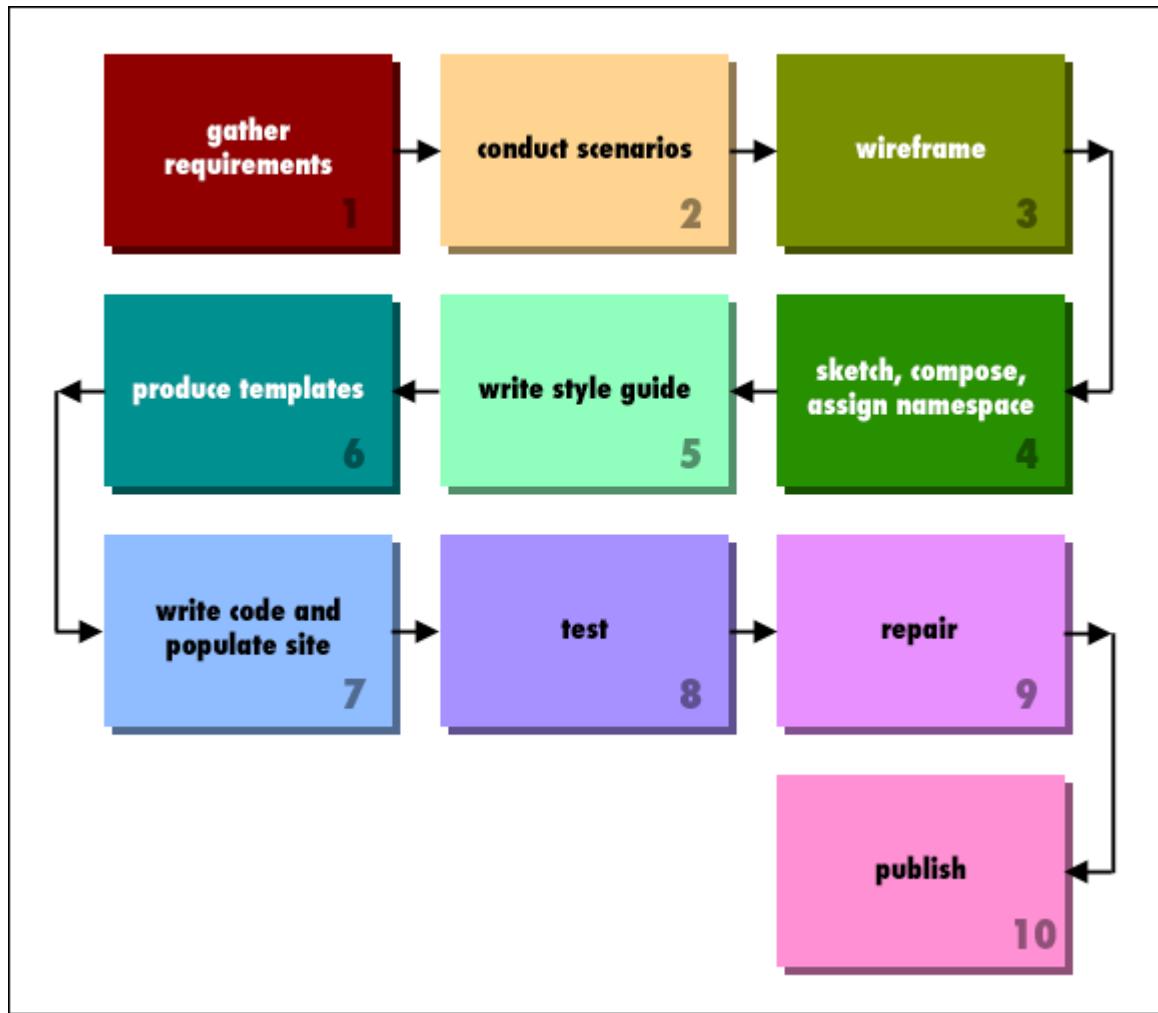


图 1：网站建设过程的十个常见步骤。与本文关系特别紧密的步骤用反色标出。

## 目标采集

良好的 CSS 有赖于一系列的依赖关系：

- CSS 依赖于工作结构；
- 结构是由页面内容反映的；
- 页面内容又需要有适用范围；
- 页面内容适用范围最终是由预期的访客目标来界定的；而
- 访客目标又是与业务目标密切相关的。

该必要条件链的核心是，你网站的访客的需求将直接决定你应该为该网站创建什么样的结构，从而决定了你的样式代码中的选择符和技巧的使用。

如果需求采集过程处理不好的话，样式设计师可能就得面对一些艰巨的任务了，这些问题可能会包括：

- 长处丢失
- 在布局的表现和窗体的几何性质之间的关系上缺乏指导
- 平台支持等级的定义缺失
- 由于在实施过程之中才仓促筹集需求，导致了要求频频改变
- 缺少流行的工具

## 内容分类

在你确定了自己网站内容的范围之后，就需要定义其对于网站整体的重要性，并决定要为访客提供什么样的导航系统以对其进行查找。

此外，如何处理像广告，链接列表，图表以及注释之类的东西也是个问题。

分类之后，页面内容就可以跟网站导航联系起来，并被赋以权重。内容加权在此处我们将以主要/次级/三级内容的形式来进行探讨，以便强调说明绝大多数用例都会偏向于使用那些对某些类型的内容赋以比其它类型的内容更高的优先级的设计决策。

## 协作草图与复合创意

如果你所做的项目的平面造型设计和样式实现是由不同的人来完成的话——商业项目通常都是这样的——平面设计师就有必要依靠一些 线框图（如果能得到的话）来开始对该网站的观感的设计了。对于界定诸如网站的整体主题，其一贯的特色，以及可能会使标记或class和id 的赋值复杂化的细节之类的东西来说，从简单的草图开始入手是个不错的办法。

对草图进行商定之后，平面造型设计师就可以着手完成最终复合图了，这种最终复合图应该是与网站投入使用后开发浏览器可能生成的屏幕截图相似的。

本文接下来的部分将不再关注平面造型设计，因为这个话题在本系列教程中的关于 线框图 和 复合图/实体模型的文章中已经讨论过了。

## 文档之间和文档内部结构的建立

拿到复合图之后，设计师就可以开始编写标记和 CSS 了。这项工作的第一步是对那些要用在网站成品中的内容顺序，元素嵌套， class 以及 id 做出决定——只有在设计师对该网站将要展现的内容以及这些内容的布局方式有透彻的理解的情况下，这项工作的完成才能达到最优。

## 对需求，分类，以及命名空间进行更详细地论述

如果一个网站在进行设计和开发决策时遵循最新的最优方法，并采用以用户为中心的设计（UCD）范式的话，那么访客目标就会告知我们设计过程的每一步。

然而，网站的主办方（或发布方）的目标实际上比设计团队的观点更重要，因为如果对那些目标没有了解的话，设计团队是无法对访客目标进行预计的。

## 业务目标定义

一个网站的“业务”目标可以归入下述的一个或一个以上的一般定义：

- 直接产生收益（电子商务）
- 通过 Web 界面提供发布，消息传递，和/或存储服务（例如，博客，网络相簿，文件分享，大文件传送服务网站，项目协作）
- 对某种产品或服务进行营销
- 提供信息
- 为访客提供娱乐

确定了一般目标之后，还要进行更进一步的细化，这种改良是基于任意数量的因素的，比如人口分布，转换目标，以及受限于项目预算或内容本身的性质（因为有可能是类似 Flash 视频这样的东西）的设计约束。

到目前为止，在涉及到对需求采集过程的这一部分进行指导时，经验是最好的老师。

## 访客目标的定义与满足

业务目标确定之后——“这就是我们想让我们的访客看到和做到的”——接下来就是将访客吸引和引导到那些目标网站以及最适合他们的功能类型了。

设计过程的这一部分中，最重要的假定就是“访客们不喜欢阻碍”。满足这个要求的最佳办法有：

- **定义并设计最适合于最有可能的那种访客路径搜寻策略的导航。**

这些策略可包括全文搜索，传统的分类学驱动的链接列表，或“加注标记”（不管是一般的还是用户自定义的）。注意，一个网站可以满足的路径搜寻策略不止一种。

- **总是提供可以表明访客在整个网站内所处位置的提示。**

用于实现这种效果的常见方法包括“浏览路径记录”，连接到相关内容的链接，以及诸如上下文相关的导航链接设计之类的视觉提示。

- **尽可能严格地实施视觉样式和书写样式，但不要严格遵循那种呆板的一致性。**

这项工作密切相关的是为访客提供一致的位置提示的需要；二者之间的不同之处在于，在这种办法中为用户提供其在整个网站内的方向的是颜色和导航栏布局，为了在单个页面内提供方向提示，跨页面的一致的外观是必要的。

- **总是用简明的语言来提示跟踪某个链接或提交一份表单的结果。**

有时这种办法非常简单，就像将一个提交按钮标记为“搜索”那样轻而易举，但有时你可能得

提供一个注释，来给予网站的访客更多的操作指南。

- 在那些用来对用户交互做出响应的设计元素和其它东西之间，要规定出明确的界线。

使得链接与一般的正文，不一致的按钮设计，以及非典型的 `cursor` 样式几乎没有任何差别的样式是非常常见的，而且非常令人迷惑的。高对比度的色彩，对 `padding` 的细心使用（以扩大交互式设计元素的封装），以及提供信息的 `title` 通常会更有效些。

- 把那些用于达成一般目的的用户交互减少到最低程度（尤其是链接的数量或按钮的点击次数），比如说一次购买，或者一般的资源之类的。

实际上，这个方案通常会需要角色扮演和访客选择分析。除非这些任务都完成了，对 **KISS** 准则的核心精神的遵照才能成为遵从此项指南的捷径。

在涉及到标记和样式表的时候，可以采用一些简单的技巧，以便使这些规则易于遵循：

- 在草拟你的样式表的时候，在规则中用简单的元素选择符为尽可能多的属性进行赋值。

按照定义来说，`id` 的值是唯一的，而 `class` 也最好是专供特殊情况使用（或者是用于离开它们老式浏览器就不能正确地支持外观要求的情况下）。这项建议意在说明，一个细心的样式设计师可以指出在哪些地方平面造型设计师对于严格控制的执着已经过度了，从而可能导致交货过晚，用户不满，或者二者皆有。

- 对每个页面的 `body` 赋一个 `id`。

如果每个文档（而不仅仅是网站的某些部分）都被指定了样式表标记，那些少见的外观个案就会变得容易处理得多。给每个页面的 `body` 赋一个 `id` 的另一个好处就是，在跟具有相同权重的导航元素一起使用的时候，样式设计师就可以在主导航条中为诸如当前浏览部分或网站之类的东西提供视觉提示，而不用编写冗长的服务器端逻辑。

- 避免那些对访客的运动控制能力要求太高的设计。

对于这项指南，更为严格的描述是“避免那些类似于用 **Suckerfish** 技术（点击此处查看 另一类 **suckerfish** 技术），同时也叫下拉菜单，创建的东西的浮动菜单”。这一类的设计具有明确的用例，而这些用例都涉及到采用单列或双列式布局的大型网站，但它们通常是可以避开的。另一方面，不熟练的用户以及有运动控制障碍的用户常常会觉得下拉菜单的使用很成问题：

- 为了讲究效果，这些元素通常都需要将其所包涵的链接变得比正文字体的基础字号更小——这种异于直觉的视觉提示会使菜单项看起来比周围其它的页面元素更不像链接。
- 为了使用这些元素所需要的运动控制的良好程度容易使那些不熟练的访客，碰巧来光顾的访客，以及运动控制受损的访客感到沮丧。

- 除非访客与这种元素进行交互，在该网站内某个特定部分中的可获得的潜在链接目标的范围是难以发现的；这就限制了访客们在该网站中维持方位感的能力，直到他们熟悉了该网站的使用为止。

## 内容分类

在制订出该网站所要呈现的内容的范围之后，你就可以对它进行架构了。一个网站的架构可以通过许多种方式来解决（具体示例请参见 [前面的一篇文章](#)）

一般来讲，你可以为你的页面内容赋以优先级，而优先级会告知我们合适的布局：

- 主要内容**，每个目标文档都是围绕着主要内容，比如文章，相册，或数据库，而创建的。
- 次级内容**包括与主要内容相关的可读的元数据，以及侧边栏内容（比如，说明文字；评论节选；连接到该网站上的相关文章的链接；图表列表，贴图，或表格）。
- 三级内容**包含了连接到相关资料（比如某个博客链接）的导出链接，从其它来源，比如从社交网络站点或评论信息源，以及广告而来的页面内容的永久快照。

除了内容之外，你的布局差不多一定会包括另外两个部分：

- 页眉**包括网站标题（通常链接到该网站的首页），主导航栏，连接到用户账户元数据的链接（在某个实际应用中），最后还有主内容搜索表单（如果部署得有的话）
- 最低限度的**页脚**包括版权声明。连接到由元数据（比如 RSS 信息源，网站地图，以及与联系信息分开的网站元内容）组成的文档的链接也放在一个网站的**次级导航**之中，该导航通常也是页脚的一部分。

在视觉环境中，网站的主导航和标题差不多总是其页眉的一部分；在标记级别上，主导航和标题是不是该网站的页眉的一部分，就完全由实施人员决定了。

## 源顺序：可访问性和其它考虑

网站模板设计的第一步就是对其内容的源顺序做出决定，该顺序应当在整个网站中保持一致。

由于可访问性和跨媒体支持的原因，即使没有样式表也易于阅读的文档源顺序是必要的。对前者而言，视力损害的用户可以使用一种叫做**屏幕阅读器**的技术：屏幕阅读器是一种用来将页面内容读给用户听的软件，如果为了外观效果而将内容排列得乱七八糟的话，这样的内容对视力损害的用户是没有多大意义或者没有任何意义的。

...就像按照屏幕显示的最优顺序来排列的源代码在读出来的时候可能会缺乏清晰性一样，针对诸如打印或手机显示之类的其它媒体进行样式化也许是不可能的。在那种情况下的输出通常会是重复内容，而重复内容会呈现出许多缺点：

- 至少，由于单个数据库记录的输出结果应该是可以以一种以上的方式展现的，为了解决这一点，必须实施额外的表现层逻辑。
- 在更糟的情况下，页面内容不仅在面向访客的环境中，也在数据库或计算机文件系统中都是重复的。这种情况会导致对某些维护的需求加倍。

因此，最常见的做法是将最外层的部分按照下面顺序来排列：

1. 页眉
  - 标题【更适合链接回首页/主页/登录页面】
  - 主导航栏
2. 主要内容
  - 文档标题
  - 正文
3. 次级内容
4. 三级内容
5. 页脚
  - 次级导航
  - 附加信息（例如，版权标志）

这些部分的嵌套方式取决于许多变化的条件，其中最常见的是网站布局中静态列的数量。

### 命名空间创建

撇开分类学的问题不谈，我们可以假设任意给定的网站都会在某个理解范围之内涵盖一系列相关主题——该理解范围可以是某个公司的运营和产品，特定类型的事件，也可以是特定种类的休闲娱乐，用以对某些实例进行命名，这些实例通常可以用来提高对那些非技术性的受众的吸引力。

因此，样式设计师很可能会发现自己正在将样式表标记与网站内容的结构性元素——比如，导航信息，页眉，正文——和内容范围关联起来，不管该范围是窄是宽。

具体的做法是多种多样的，不过作者通常都会将下面的布局标记用在自己的模板中，本文中我们也会看到这样的模板：

```
#main
 页面内容画布
 h1 (这个是唯一的)
 网站标题

 ul#nav
 网站导航代码

 #breadcrumb
 浏览路径记录（如果采用了的话）

 #bodyCopy
 主题专文
 #bodyCopy>h2(这个是唯一的)
 主要文档标题

 #sidebar
 次级内容

 #footer
 页脚代码

 ul#secondaryNav
 次级导航
```

除了上述标记之外——而且比这些标记更加重要的是，每个页面的 `body` 都添加了一个 `id`（就如前面所提到过的那样），从而对与整个文档相关的主要内容的范围给出了一些说明。此外一些项目还会要求对 `body` 元素的 `class` 进行赋值。

**单列式布局的实现**

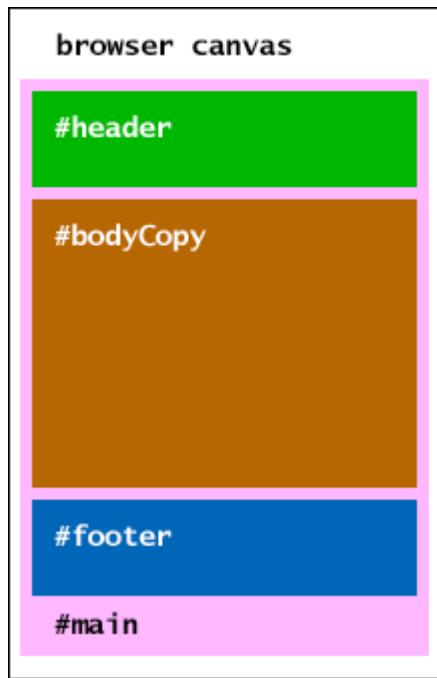


图1：一个单列式布局中的元素；单列式布局的标记可能会以如图所示的形式进行嵌套。

图1展示的是#main紧接在body之内，并依次包涵所有的#header, #bodyCopy和#footer。  
**布局的居中对齐**

将内容画布居中对齐就是将`#main { width: 960px; margin: auto; }`插入（在本例中是这样）到你的样式表中（width属性值的选择是随意的）。

#### 全文档范围的容器是绝对必需的吗？

在完全是单列布局的网站中，并不是绝对要带有的；你也可以轻易地将上面提到过的那些布局属性/值对应用于#header, #bodyCopy 和#footer 的组合上。不过，带有#main 也不是什么语义学上的错误，并且在涉及到规则，缝隙，背景图像，以及建立某些元素在该模板结构中的突出性时，包含#main 还可以为样式设计师提供更多的灵活性。

#### 双列式布局的实现

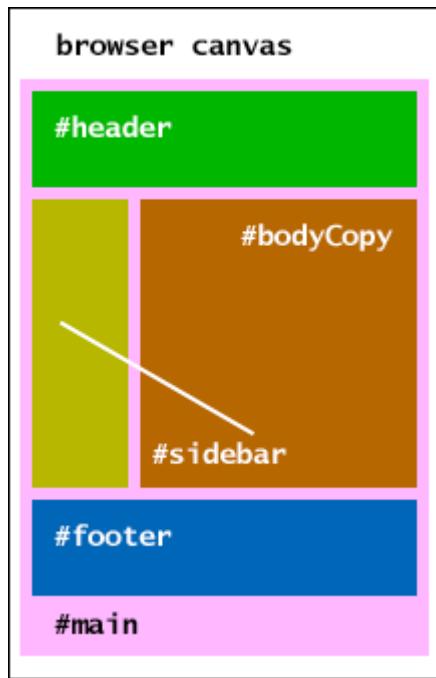


图 2：双列式布局中的元素；该布局可能会以如图所示的方式进行嵌套，不过要理解的是，实际上在源顺序中 #sidebar 是跟在 #bodyCopy 后面的。

单列式布局和双列式布局之间的差别在于双列式布局增加了另一个容器元素，用来容纳次级内容（#sidebar，在源顺序中它实际上是跟在#bodyCopy 后面的），并且双列式布局在单列式布局所使用的样式表的基础上也进行了一些改动。

#### **单列式和双列式布局在构图方面的比较**

要将一个单列式模板转化为双列式模板所需的标记改动是很简单的，但在许多情况下所需的新样式规则却不那么简单。

#### **不管其源顺序如何，将#sidebar 放在页面左边**

有两个办法可以实现这种效果；其中一个方法不管两个列的长度如何都能奏效，而另一个则要求#bodyCopy 比#sidebar 更长。

第一个办法，而且也是最常见的办法就是使用 `float`：

1. 为#bodyCopy 赋一个 `width` 值。
2. 在这条规则中添加一个 `float` 属性，其值为 `right`
3. 为#sidebar 指定一个适当的 `width` 值。
4. 为#bodyCopy 和/或#footer 赋以所需的 `margin` 和 `padding` 值，以确保在这两个元素之间留有我们想要的缝隙。
5. 给#footer 赋 `clear: both;` 值。

两个列均指定了 width 值，故此，这两个列都具有一致的边距。

在 #sidebar 位于布局的右边缘的情况下，除了将 #bodyCopy 的 float 值设为之外，仍然可以遵循上面所描述的步骤来操作。赋给 #sidebar 的 width 值应当用大小适当的 margin-left 值来替换。

也可以对未赋 float 的元素设置一个比较大 margin-left 或 margin-right 值（如果需要的话），来代替 width 值；关于这项做法的可行性在本系列教程的其它地方有简单的讨论。

第二种方法在 IE6 中引发漏洞的可能性较小，这种方法是酌情对 #bodyCopy 赋一个较大的 margin-left 或 margin-right 值，并将 #sidebar 设为绝对定位。然而，这种方法的灵活性较差，在比更长的情况下会导致前一个元素溢出到中去 #footer。

### 伪列布局：利用背景图片来使内容长度不同的列长度相等

对利用 float 属性来实施列式布局进行更深入的研究之后发现，在列之间需要不同的背景颜色或垂直线条时，如果和 background-color 或 border 属性一起使用的话，就不能靠着这些背景颜色和线条来占据主要内容区域的长度了。

对于这个问题最简单的解决办法是在这些列的某个祖先元素中创建并指定一幅背景图像（通常高度为 1 像素）——因为该祖先元素中引入了这个背景图像的缘故——该图片将按照最长的相关列的高度从顶部一直平铺到底部。因此：

```
#main {
background-image: url(images/bg_2column.gif);
background-repeat: repeat-y;
}
```

如果 bg\_2column.gif 是由两种对比度很高的颜色构成，而且每种颜色的宽度几乎精确地等于你给内容列所赋的宽度值，那么最终结果就是两列的高度会显得一模一样...但实际上它们的高度并不一样，如果其中一列接下来插入下面的规则的话，可能就会被发现：

```
#bodyCopy {
background-color: #ccc;
}

#sidebar {
background-color: #999;
```

```
}
```

采用伪列布局的同时并不是一定不能对某个给定的列使用 color 或 background-color 属性的，如果默认的文本颜色相对于某个或两个列来说难以认清的话，其背景和前景颜色就应当在样式表中进行显式指定，从而作为一种自动防故障装置来避免浏览器无法加载背景图像的潜在风险。

### 保持源顺序不变，将主导航栏移到第二个列中去

在布局中添加第二个列之后，将主导航栏放在该列的顶部可能会看起来自然一些...但是如果该导航栏位于该模板结构中的另一个部分的话，我们该怎么做呢？

这个问题的答案就在于定位：

1. 如果为#header 指定了 overflow: hidden; 值的话，可将#nav 设为#main 的直接子元素。在几乎所有的情况下，这一点都可以不用改变预定的源顺序而做到。
2. 为#nav 的直接祖先元素赋 position: relative; 值，并为#nav 自身赋 position: absolute; 值。
3. 由于绝对定位的缘故，#nav 将被默认置于其祖先元素的左上角，在这种情况下可按照需要调整#nav 的 left 和 top。
4. 对#sidebar 的 margin-top 或 padding-top 值进行调整（酌情），以反映出#nav 的预期高度。如果各个页面或部分的#nav 的高度不同的话，就有必要编写多条规则，也许在每条规则中要用到多个选择符——因此，就可以遵照前面我们提到的那个建议，对该网站的每个文档的 body 指定一个内容范围指示 id（也可能是 class）。

### 三列式布局的实现

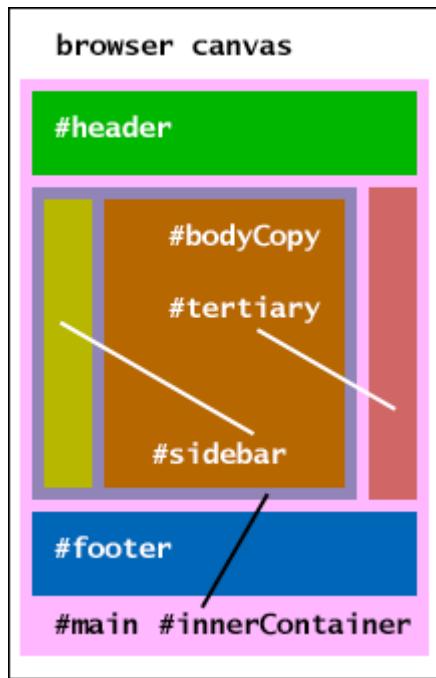


图 3：三列式布局中的元素；注意那两个新的容器元素，以及不同的 `id` 赋值。

由第三个列的增加造成的标记的主要变化有：

- 一个封装了两个相邻列，通常是第一个列和第二个列的容器；以及
- 第三个列位于自己的容器内。

在标记拟定之后，制作期望的布局就是如何适当地调整 `float` 值的问题了。别忘了对于非浮动的容器需要进行边距调整，来使它们妥善地排成一行。

注意，在图 3 中，最好是对前两列和第三列的容器元素赋给 `id` 值，以对各自的内容提供一些说明，而不是赋一个像该图表中那样的一般的 `id`。

### 三列式布局中每个列的 `float` 值指定

预期外观	容器内容	容器	主要	次级	第三级
2-1-3	1+2	left	right	none	none
2-3-1	2+3	left	none	left	none
3-1-2	1+2	right	left	none	none

## 三列式布局中每个列的 float 值指定

预期外观	容器内容	容器	主要	次级	第三级
3-2-1	1+2	right	left	none	none
1-2-3	1+2	left	left	none	none
1-3-2	2+3	right	none	right	none

表 1：在三列式布局中的四个容器 float 元素的值，按照从左到右的显示顺序排列。

### 三列式设计中最容易犯的错误，以及其简单解决办法

最灵活的三列式布局引入了一个在语义学上毫无意义的容器元素；替代方案就是对内容长度或源顺序施行转化，该项转化是针对那些会造成繁重的负担的内容长度或源顺序而进行的，这种负担要么会影响维护人员（就内容长度要求而言），要么就会影响访客（就源顺序限制而言）。

这种“毫无意义”的容器的引入在涉及到网站的重新设计时也会造成问题。我们来考虑一下下面的场景：

有一个网站，初次设计时该网站的布局是这样的，其列按照外观上 2-3-1 的顺序来放置的，但后来进行重新设计时，它的列是按照更加传统的 2-1-3 的顺序来摆放的。那么就需要对双列的容器元素进行移动，以便让其封装不同的列容器。接下来又该怎么办呢？

在这种情况下，很容易就能实现我们想要的结果；如果该网站是真正的模板驱动型的话，就（可能）要对一些文件进行改动。如果——从另一方面来讲——该网站所有的文档都使用同一标记框架的话，那么就必然要用到查找替换。不过，这并不会很难。

容器的标记的大体布置可采用下面两种形式中的一种：

- ```
<div id="#container"><div id="primary">...</div><div id="secondary">...</div></div><div id="tertiary">...</div>
```
- ```
<div id="primary">...</div><div id="#container"><div id="secondary">...</div><div id="tertiary">...</div></div>
```

在这两个例子中，与查找替换有关的标记片段我用粗体标出了。由于这些 id 在你的文档中是唯一的，而且两个结束标记的位置是可预测的（不是与第三个结束标记相邻，就是与#tertiary

相邻), 编写查找替换操作来对它们进行改变是相对容易的。

## 页眉和页脚博览

到现在为止, 我们已经讨论了应该放入页眉和页脚的内容——商标/标识语, 全站范围内的搜索, 连到用户账户信息的链接, 网站元数据——但除了本系列教程中关于 [线框图](#) 和 [工艺创意/实体模型](#) 的文章之外, 对页眉/页脚的效果和吸引力却几乎没有做任何探讨。

将读者放在一堆理论中间真是不好意思, 我们最好还是来看看三个著名的网站——它们或者是受欢迎程度很高, 或者是发布方的声望很高——并来看看它们在设计方面的各种各样的特点。

### 个人网站: Cindy Li



图4: 此处有两个特别值得讨论的重要问题: 标志和对比度。

### 标志

“标志”是在广告和市场营销领域内的一个具有特殊意义的术语, 它指的是商标和为某个商业企业和其产品的外观所特有的其它设计元素。通过在网站页眉中放置一张她自己的面部侧貌的漫画, 以及用一种不常见的字体来设置她的网站标题和主导航栏, Cindy Li从个人角度实现了标志。

我们将会在下面对企业网站上的标志实施方式进行更加详细的讨论。

### 对比度

在 cindyli.com 上, 访客们一看就能明白某个内容到底是什么: 标志, 内容画布, 并且主要内容都是放在覆盖区之内的, 这些覆盖区由它们的背景颜色分离开来。此外, 网站标题和导航栏采用了与它们的背景的对比度很高的颜色。

现在来看看网站页脚:



图 5: Cindy Li 的页脚内容有点少: 只有一个版权声明, 一个连到她所使用的发布平台的营销网站的链接(这可能是在它的使用授权条约中规定要有的), 以及一个连到关于她所发布的文章的 RSS 信息源的链接。

Unlike the other sites presented in this article, Cindy's site doesn't offer fulltext search, probably for technical reasons. However, since the site is a blog, its design allows the assumption that most readers confine their interest to new content.

#### 社交网络: Facebook



图 6: 就像许多社交网络目标站一样, Facebook 通过其布局和颜色来实现自己的标志, 因为这种应用程序本身就是目标站。

就像许多商业网站一样, Facebook 的网站导航提供全文搜索和分类搜索两种方式。



图 7: 就像它的页眉一样, Facebook 的页脚也很小, 即使将底部的永久应用程序窗口组件也纳入考虑范围, 也还是很小。在这里可以看到一件有趣的东西: 在版权声明的旁边, 有一个用来改变用户的默认语言的组件。

Facebook 页脚所展示的另一种习惯做法是, 用户寻路的有用链接放在页眉中, 而页脚则容纳所有的指向该网站自身信息以及网站操作员的链接。

#### 企业市场营销与客户服务: BNSF 铁路公司

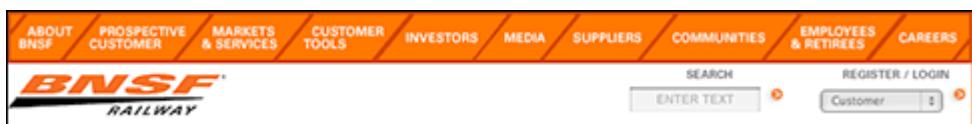


图 8: 像大多数“企业”网站一样, 该网站的发布方也喜欢在页眉特征上使用最高的对比度...而且这里唯一的色调跟商标上所用到的那种颜色一样。

#### 再评标志

关于那些不同的商标, 商标标准字, 以及其它起商业标志作用的设计比喻, 我们来看看下面这些图片, 这些图片都是随处可见的与 Opera 软件公司有生意关系的企业的标志:

- 诺基亚



- S三星集团和三星电子



- 谷歌



- IBM



除了上述四个之外，还有其它类似的图形设计的例子也是众所周知的：耐克强风（这个名字已经在官方注册过了），美国电话电报公司的“末日星球”商标，美国联邦快递商标标准字（以及它的最后两个字母间的空白所形成的箭头），还有“UPS（联合包裹服务公司）式的褐色”，这些都是普通大众家喻户晓的公司标志（至少在美国是家喻户晓的）。

与这些属性相关的要点在于，任何一个特意要开创与众不同的视觉标志的项目主办方都应该预先想到让自己的网站来对该标志起强化作用，而不是事后才来考虑。

至于在 BNSF 网站的页眉中除商标之外的其它元素，最值得注意的是两种寻路模式的使用（跟 Facebook 上的一样）。



**图9：**在这三个网站中，BNSF 网站的页脚布局最为“传统”，因为它几乎没有通过水平线条或不同的背景颜色来对次级导航进行任何加工，以将其放在一个独特的视觉平面内。

#### 页眉和页脚设计：初级细节

在浏览了更多页眉和页脚设计之后，下面所述的共性就很容易理解了：

- **企业和应用程序目标站往往将自己的主导航栏设计成浏览器画布的顶边上的一条横排。**这个特质使得它们与新闻网站和电子商务网站区分开来，新闻网站和电子商务网站的主导航栏的布局更多地采用的是纵栏的方式。
- **就像水平式的主导航栏一样，分立的次级导航在页脚中的使用也是很常见的。**此外，在本文前面的部分中描述过的目标站对网站元数据的划分也相当常见。

- 页眉往往包含着相当大的实体周围空间部分，但是对于页眉的构图来说，还是有一些必须遵守的规则。在 Cindy Li 网站上，内容画布是特意设计成窄小的，从而使页眉顶部留出了相当大的空白。
- 在含有全文搜索的页面上，全文搜索的输入栏往往位于页眉的右边缘处。这在很大程度上是由于设计良好的全文搜索只是一种全站导航的方式，跟传统的分类学驱动的导航链接一样是合法的——而且可能会被你网站用户中最不懂技术的部分访客所诟病。

### 更多关于导航布局实施的讨论

在进入细节部分之前，读者应先去查阅一下 [List-O-Matic](#)，这是在 [Accessify.com](#) 上的一个应用程序，用于创建将要插入到任意页面的布局中去的简单款式的导航元素。

除了可以代替你完成工作的工具之外，还有两种进行导航布局的基本方式：

- **导航条是网站页眉整体的一部分（如果在代码上不是一个整体，至少在视觉上是一个整体）并被设计为水平式。**在这种情况下，每个链接的 `display` 值都可能被改成 `block`，并且它们所包含的列表项将被赋以值为 `left` 的 `float` 属性。
- **导航条是纵式的，并位于主要内容的左侧，既可以处在自己的布局列中，也可以处在非主要内容的上方。**在这种情况下，你多半会看到某些类型的非 `static` 定位的使用。

### 列数不固定的网站：利用 `class` 和 `display` 来走捷径

有些网站的网页很可能是这样的：一些页面是单列或双列式布局的，其它的是双列和三列式布局的；灵活性是 CSS 的长处之一，同时也是美术设计员在他们执着追求对用户体验的强力控制的同时，无意中形成的缺陷之一。

一般说来，这类情况可以通过添加下面的脚本来得到部分的处理：允许在页面中程式化地添加永久代码片段，而不是只能重复拷贝代码片段的网站脚本。

然而，即使包括了这种脚本，样式设计师还是会遇到布局上的差异：如何才能最好地处理这些差异呢？

最简单的方法就是给所有可能有需要的页面的 `body` 绑定一个 `class`。这些 `class` 可能会具有与在标志指南那一部分中所提到的某些系列的布局相对应的次序性质，或者与页面内容相结合，并导致像下面这样的多选择符规则：

```
.about #bodyCopy,
.contact #bodyCopy,
.privacy #bodyCopy {
```

```
float: none; width: auto;
}

.about #sidebar,
.contact #sidebar,
.privacy #sidebar {

display: none;
}
```

仅仅使用这种方法而不使用包含语句（如果你愿意这么做的话，这是使内容消失或再现的一个不怎么样的办法）的缺点是很可能会导致搜索引擎操作策略降低这类网页的排名——或者在极其不灵活的实施的情况下，完全不列出任何使用这种方法的网站。由于这个原因（这只是多得数不过来的原因中的一个），你所得到的任何网站寄存解决办法都应该支持某种包含功能。

## 总结

虽然坐下来编写标记和代码——尤其是当你还是个初学者的时候——是很诱人的，但这种过程对于特别吸引人的，使用的或易于维护的网站来说并没有帮助。

然而，通过对将要发布到网站上的页面内容和应当对它进行的布局的方式加以仔细考虑，你就可以根据其要求推导出任何一个网站的框架，并将该框架在网站上实施出来。

基础模板：

- 单列式布局
- 双列式布局
- 三列式布局

## 练习题

1. 假定有一个由你的指导者提供的合理的链接列表，将列表中的链接划分为主要（页眉）和次要（页脚）链接。基于本文所提供的示例，证明你的划分正确。
2. 看看你的同学创建的布局组合，并进行识别：

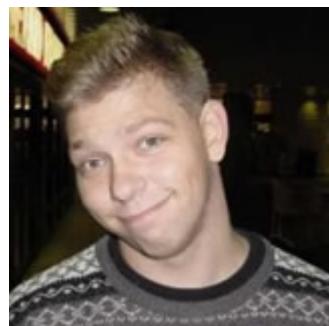
1. 该设计中所要实现的布局列数;
2. 这些布局列的宽度; 以及
3. 用于实现这些布局列的外观的 `float/width/margin` 方案, 如果有的话。
3. 给定一个商标标准语, 一份要求清单, 以及一个结构, 设计一个网站页眉。
4. 如果你不能证明上一个练习中所设计的页眉的构图符合黄金分割, 对其构图进行适当的改动并主观地估计一下最终效果的吸引性有多大。
5. 仅仅依靠搜索引擎结果, 来分析为什么对结构化导航元素来说列表比一堆 `div` (或其它元素) 更好的原因。在你的答案中要涉及到屏幕阅读器软件的特性。
6. 凭着记忆, 对上面提供的某个模板文件进行修改, 以使其支持另一种布局列数。另外还要更改主导航栏的构图, 使其相较于原来的模板文件中的样子发生显著的改变。

## 参考文献

- BNSF Railroad. 2006. <http://www.bnsf.com/> (last accessed 13 January 2009).
- Facebook. 2008. <http://www.facebook.com/home.php> (last accessed 13 January 2009).
- Henick, Ben. 2006. Avoid edge cases by designing up front.
- A List Apart. <http://www.alistapart.com/articles/avoidedgecases> (last accessed 13 January 2009).
- Li, Cindy. 2008. The Adventures of Cindy Li.  
<http://www.cindyli.com/> (last accessed 13 January 2009).
- Morville, Peter, and Rosenfeld, Louis. 2006. Information architecture for the world wide web, 3rd edition. Cambridge, Mass.: O'Reilly Media.

- 上一篇: CSS 布局模型——CSS 绝对定位与固定定位
- 下一篇: 程序设计——真正的基础!
- 目录

## 作者简介



Ben Henick 从 1995 年 9 月就开始以各种身份参与创建网站了，那时他以一个学术志愿者的身份进行了他的第一个 Web 项目。从那时开始，他的大多数工作都是在自由职业的基础上完成的。

Ben 是一个多面手。他的技能触及了网站设计和开发的几乎每一个方面，从 CSS 到 HTML，再到设计和文案撰写，再到 PHP/MySQL 和 JavaScript/Ajax。

他生活在堪萨斯州的 Lawrence，有三台电脑，却没有电视机。你可以在 [henick.net](http://henick.net) 上读到更多关于他和他工作的内容。

## 39. 程序设计--真正的基础!

Posted 03/10/2009 - 23:01 by yuanc

- 网络标准教程

作者: Christian Heilmann · 2009 年 2 月 3 日

- 上一篇: 页首、页脚、列和模板
- 下一篇: 你可以用 JavaScript 做什么?
- 目录

### 引言

作为一个经验丰富的开发员，你迟早都会面对不懂技术的人群，不管你做的是什么他们都会觉得那是奇异的魔术。相反的，作为一个非技术性的人，不知道别人交付给你的成品是什么，这可是一个不好的开端。本文简单地讲解了程序设计的一些知识，希望对于所牵涉到的两方都有帮助，使他们能够克服这种无法交流的状况，并达到更加富有成效的境地。

本文也能帮助那些 web 开发新手了解一些一般的程序设计原理，在你开始学习如何编写 JavaScript 之前，有必要先学懂这些准则。本文的开头可能看起来有些乏味，但是相信我，如果你从一开始就记住这些基本原理的话，你的作品将会变得健壮得多，精悍而且有创新性得多（又叫做：惊艳指数高得多）。在你开始用某种程序语言进行编程之前（就本教程而言，是 JavaScript），对最基础的程序设计基本法则的学习是很重要的

本文结构如下：

- 指令，我可以编写指令啦！
- 变量
  - 变量类型
    - 浮点型与整型
    - 布尔型
    - 字符串型
    - 数组型
    - 对象型
- 条件
- 循环语句

- 总结
- 练习题

## 指令，我可以编写指令啦！

程序设计最根本的形式就是将命令组合起来，并看着它们得到执行。程序设计是数学和语言学的混合物：你定义了大量的运算，并需要通过正确的语法来发出指令，告诉计算机来处理它们。在程序设计中，语法就是句法规则，并且各种程序设计语言的语法彼此大不相同。

比如说，下面的两段代码片段实现的是同样的效果，但前一段是用 **JavaScript** 编写的，而后一段是用 **PHP** 写的。

```
var fahrenheit = prompt('Enter temperature in Fahrenheit',0);

var celsius = (fahrenheit - 32) * 5 / 9;

alert(celsius);

$fahrenheit = $_GET['fahrenheit'];

$celsius = ($fahrenheit - 32) * 5 / 9;

echo $celsius;
```

自己试一试这个**JavaScript**写的 华氏度到摄氏度的转换示例。

为了转化为操作或结果，程序设计语言需要经过解释。某些语言，比如 **JavaScript**，是由 **web** 浏览器来解释的，你要做的就是将它们放在一个 **HTML** 文档中，并在浏览器中打开这个文档，从而让它们“一显身手”。其它的程序设计语言需要进行额外的翻译（编译）步骤，这样才能成为可执行文件。实际上，所有的计算机都只能读懂 0 和 1 的机器码，但在机器码以上还有多个层次的语言，每个层次的语言完成的是不同的任务。

在 **web** 环境中，解释和编译之间的界线已经开始模糊了，因为一些浏览器正在着手进行 **JavaScript** 快速编译引擎的实验。不过对于这一点你也不用太担心；目前的规范还是要求解释 **JavaScript** 代码的。

## 变量

为了理解程序设计，我们首先来回顾一下代数学。如果你能记得起来的话，在学校里的代数学习是以下面这样的等式开始的：

```
3 + 5 = 8
```

如果你引入了一个未知数，比如下面的 `x`，就可以开始演算了：

```
3 + x = 8
```

将常数移到等式的一边，你就可以算出 `x` 的值：

```
x = 8 - 3
-> x = 5
```

如果你引入了一个以上的未知数，你的等式就更加多变了——这样你就使用了变量：

```
x + y = 8
```

你可以改变 `x` 和 `y` 的值，而等式依然成立：

```
x = 4
y = 4
```

或者

```
x = 3
y = 5
```

在程序设计语言中也是这样——在编程时，变量就是用来盛装那些可以变化的值的容器。

变量可以包含所有类型的值，也可以用来容纳计算的结果。变量具有名字和值，名字和值之间由等号 (=) 分开。变量名可以是任意的字母或单词，但要记住各种程序设计语言对于你可以使用的变量名各有不同的限制，因为有的单词是专供别的功能使用的。

为了简单起见，本文中我们用 `JavaScript` 来作程序设计语言范例（从逻辑上来讲这样做是很自然的，因为本 web 标准教程的这个部分全是由 `JavaScript` 编程的！）下面的代码定义了两个变量，计算了这两个变量的和，并将计算结果定义为第三个变量。

注：此处的`<script>`标签的作用是告诉浏览器标签内的文本是脚本语言，并应当对其加以解释。类型属性`"text/javascript"`则告诉浏览器这是什么语言。

```
<script type="text/javascript">

var x = 5;

var y = 6;

var result = x + y;
```

```
</script>
```

解释程序将逐条地检查整段代码的指令，每条指令都以分号结束。分号告知解释程序一条指令的结束，就像在人类语言中句号或感叹号表示一个句子的结束一样。

用英语来讲，这一段代码的意思如下：

- 接下来的内容不是 HTML 的；生成一个能读懂基于文本的 JavaScript 类型的语言的翻译程序。
- 定义一个名叫 **x** 的新变量（就是 **var** 关键词），并给它赋一个大小为 **5** 的值。指令结束。
- 定义一个名叫 **y** 的新变量，并给它赋一个大小为 **6** 的值。指令结束。
- 定义一个名叫 **result** 的新变量，并将 **x** 加 **y** 的计算结果赋给它。语句结束。（由于 **result** 变量的赋值中有一个算式，解释程序会去查看 **x** 的值，然后查看 **y** 的值，将这两个值加总，并将 **result** 的值设置为计算结果——**11**）。
- 这个奇怪的语言终于结束了——回到 HTML 中，并告诉翻译程序离开这段代码。

你可以用变量来进行所有类型的计算，只需要在变量之间加入运算符即可。标准的计算方法有，用加号运算符实现加法，以及用减号运算符实现减法，等等。要实现乘法运算，你就得用星号 (\*)，而除法运算是用斜杠 (/)。要注意的是，前面加了双斜杠//的文本——这些文本是 JavaScript 中的注释。当 JavaScript 解释程序在脚本中遇到这两个符号时，就不会试图执行同一行中位于双斜杠后面的语句，而会直接跳过——因为这些是用来方便人们阅读代码的注释，而不是需要浏览器解释的代码。

```
<script type="text/javascript">

var x = 5;

var y = 6;

var z = 20;

var multiply = x * y * z;

// multiply 的值为 600

var divide = x / y;

// divide will be 0.833333333333334

var addAndDivide = (x + z) / y;

// addAndDivide = 4.166666666666667

var half = (y + z) / 2;
```

```
// half 的值为 13
</script>
```

如你所见，你可以对任意变量进行组合搭配，也可以在计算中同时使用变量与常量；还可以用括号对它们进行分组，以超越运算符的自然顺序（括号的优先级第一，然后是乘法或除法，然后是加减法，这些都是数学课上教过的标准）。

### 变量类型

然而，如果我们用简易计算器来完成所有运算的话，将会非常枯燥。计算机要更尖端一些，可以对更复杂的变量加以利用。这就是为什么会有变量类型的原因。变量有若干类型，并且不同的程序设计语言支持的类型也不同。最常见的有：

- 浮点型：类似 `1.21323, 4, 100004` 或 `0.123` 这样的数字
- 整型：像 `1, 12, 33, 140` 这样的自然数，而不是 `1.233`
- 字符串型：像`"boat", "elephant"` 或 `"damn, you are tall!"`这样的一行文字
- 布尔型：或者是 `true` 或者是 `false`，除此之外没有别的值了
- 数组：一个像 `1, 2, 3, 4, 'I am bored now'` 这样的值的集合
- 对象型：代表了一个对象。对象是一种构造，它试图通过属性和方法来建立真实世界中的事物的实例模型。比如说，你可以为一只猫建模，将其作为一个对象，并将其定义为有四条腿，一条尾巴，而且是姜黄色的。你还可以通过定义一个 `purr()` 方法来对它鸣叫的方式进行定义，还有它可能会爱吃芝士汉堡包，可以给它定义一个 `getCheeseburger()` 方法。你还可以重复使用这个 `cat` 对象来定义另一只所有属性都跟原来那只一样，但毛是灰色的猫。

**JavaScript** 是一种“宽松类型”的语言，这就意味着你不必显式地声明变量的数据类型。你只需利用 `var` 关键字来表明自己正在声明的是一个变量就可以了，浏览器会根据上下文自行计算出你所使用的是什么数据类型，你也可以用引号将值括起来。

### 浮点型和整型

这些数据类型并没有什么神奇或怪异之处。你可以定义变量，并将它们的值设置为任何数字类型。

```
<script type="text/javascript">

var fahrenheit = 123;

var celsius = (fahrenheit - 32) * 5/9;

var clue = 0.123123;
```

```
</script>
```

浮点型和整型可以通过任意的数学运算符来加以修改。

## 布尔型

布尔型是一种简单的“是或非”的定义。你可以通过 `true` 或 `false` 来对其进行赋值。

```
<script type="text/javascript">

 var doorClosed = true;

 var catCanLeave = false;

</script>
```

## 字符串型

字符串是指可以包括任意字符的文本行。在 JavaScript 中，你可以通过将文本括在单引号或双引号中来定义字符串。

```
<script type="text/javascript">

 var surname = 'Heilmann';

 var name = "Christian";

 var age = '33';

 var hair = 'Flickr famous';

</script>
```

你可以用`+`号操作符来将字符串串联（一种技术术语，意思是“连接起来”）起来。要修改字符串，你需要用到程序设计语言提供给你的函数。但在另一方面，简单的串联跟下面所示的一样容易：

```
<script type="text/javascript">

 var surname = 'Heilmann';

 var name = 'Christian';

 var age = '33';

 var hair = 'Flickr famous';

 var message = 'Hi, I am ' + name + ' ' + surname + '. ';
```

```
message += 'I am ' + age + 'years old and my hair is ' + hair;
alert(message);
</script>
```

试试看这个字符串连接示例。

**+=**操作符是“某某变量=该变量自身+”的缩写形式。这个脚本的结果就是字符串“Hi I am Christian Heilmann. I am 33 years old and my hair is Flickr famous”。

在用加号操作符来串联字符串和进行值的相加之间，有些细节问题需要记住。如果你想将两个值相加，你就应该确保这两个值都必须是数字，而不是字符串。串联对加法示例说明了二者之间的差别。“5”+“3”的结果是 53，而不是 8！最简单的将字符串转化为数字的方法是在它前面加上一个“+”，就像这个例子里面的那样。

大多数程序设计语言都不会在意你用的是双引号还是单引号来将一个字符串括起来，只要你不把单双引号混用就行。为了防止 JavaScript 对字符串到底在哪里结束感到困惑，你需要用斜杠将包含在字符串中的引号注释出来：

```
<script type="text/javascript">

// 这样声明将会导致程序出错，因为解释程序不知道

// '后面的东西到底是什么。此处声明的字符串会变成

// 'Isn'.

var stringWithError = 'Isn't it hard to get things right?';

// 这样声明就不会出错了，这样程序就会运转正常

var stringWithoutError = 'Isn\'t it hard to get things right?';

</script>
```

## 数组型

数组是一种非常强大的数据结构。一个数组就是一个值的集合，每个值都可以是变量，也可以是真正的数值。比如说：

```
<script type="text/javascript">

var pets = new Array('Boomer','Polly','Mr.Frisk');

</script>
```

你可以通过**数组计数器**来访问每一个值，数组计数器就是该数组中的索引编号——可以把

这看作是在一本书里进行章节的查寻一样。利用数组计数器来访问某个值的语法是 `arrayname[index]`。因此，比如说 `pets[1]` 的结果就是字符串“Polly”。但是请等一下！我听到有人在问——Polly 不是应该是 `pets[2]` 吗，因为它是这个数组中的第二个值？错——索引值可不是 2，因为计算机是从 0 开始数起的，而不是 1！这一点经常都会引起混淆和错误。

数组会自动为你提供一个特殊的信息源：`length`。如果你查看 `pets.length` 的值，就会得到 3，因为在这个数组中一共放了 3 项值。

对于描述具有共同点的事物的集合，数组是一个很好的选择，而且每种程序设计语言都会附有许多方便的函数来对数组进行操作——排序，筛选，查找，等等。

## 对象型

如果你有一批物品，需要对其进行更详细的描述，而不只是一个流水号的话，数组就不能满足你的要求了，你需要的是创建一个对象。对象是程序设计中的一种数据结构，代表了或者说是模仿了真实的事物，就比如说人，车辆，或者是工具。

对象是程序设计中一个重要而又十分灵活，而且用途很多的组成部分，要在这里对它们进行详细阐述，恐怕就有点超出了本文的范围了。我们这样讲，一个对象就是一个带有若干属性的东西。比如说，假定你有一个人物对象；你就可以通过在各种属性前面加上一个点号，来对它们进行定义：

```
<script type="text/javascript">

var person = new Object();

person.name = 'Chris';

person.surname = 'Heilmann';

person.age = 33;

person.hair = 'Flickr famous';

</script>
```

你可以通过点号来访问属性（`person.age` 会得出 33），或者用方括号来访问也可以（`person['name']` 的结果是“Chris”）。本教程随后就会教你更多关于 JavaScript 的知识。

以上这些就是关于变量可以取什么值类型的一个概括。程序设计的另一个重要部分就是你的程序的结构和逻辑。这就该轮到另外两个程序设计思想上场了：条件与循环。

## 条件

条件就是对某事的测试。条件对于程序设计来说是非常重要的，主要表现在以下几个方面：

首先，条件可以用来确保我们的程序能够运行，不管你交给它处理的是什么数据。如果你轻率地相信数据的话，就会陷入麻烦之中，而且你的程序也会失败。如果你的测试证明自己想要进行的事情是可行的，而且也具备了一切所需的正确格式的信息的话，这种情况就不会发生了，而你的程序也会稳定得多。采取这样的预防措施又叫做防御性编程。

条件可以帮你做的另一件事就是它可以进行分枝。你可能以前也遇到过分枝图，比如说在填写表单的时候。基本说来，这指的是对代码的各个“分枝”（部分）的执行，是否执行则取决于条件是否得到满足。

最简单的条件就是 `if` 语句，其语法是 `if(condition){ do this ... }`。只有当这个条件语句为真的时候，大括号内的代码才能得以运行。比如说，你可以对某个字符串进行测试，并根据其值来设置另一个字符串的值：

```
<script type="text/javascript">

var country = 'France';

var weather;

var food;

var currency;

if(country == 'England'){

 weather = 'horrible';

 food = 'filling';

 currency = 'pound sterling';

}

if(country == 'France'){

 weather = 'nice';

 food = 'stunning, but hardly ever vegetarian';

 currency = 'funny, small and colourful';

}

if(country == 'Germany'){

 weather = 'average';

 food = 'wurst thing ever';

 currency = 'funny, small and colourful';

}
```

```

}

var message = 'this is ' + country + ', the weather is ' +
 weather + ', the food is ' + food + ' and the ' +
 'currency is ' + currency;

alert(message);

</script>

```

你可以自己在我那个关于天气的 `if` 语句示例中试试看这段代码。改变 `country` 变量的值，这样就可以看到不同的消息。

该示例中的条件部分就是后面跟着三个等号的 `country`。三个等号的意思是该条件语句会测试 `country` 变量的值是否就是你所测试的那个值，以及是否是正确的变量（数据）类型。你也可以用双等于号来测试条件，但这个符号的意思是，无论 `x` 等于数字 `5` 还是等于字符“`5`”，`if(x == 5)` 的值都为真。随着你的程序所要做的事不同，这个语句所导致的结果也会大不相同。

其它的条件测试案例：

- `x > 0 - x` 是否大于零？
- `x < 0 - x` 是否小于零？
- `x <= 4 - x` 是否小于等于四？
- `x != 'hello' - x` 是否不等于 '`hello`'？
- `x - x` 是否存在？

条件也可以进行嵌套。比如说，假设你想确保前面那个例子中的 `country` 变量是进行了设置的；你就可以这样做：

```

<script type="text/javascript">

var country = 'Germany';

var weather;

var food;

var currency;

if(country) {
 if(country == 'England') {
 weather = 'horrible';
 food = 'filling';
 }
}

```

```

 currency = 'pound sterling';

 }

 if(country == 'France'){

 weather = 'nice';

 food = 'stunning, but hardly ever vegetarian';

 currency = 'funny, small and colourful';

 }

 if(country == 'Germany'){

 weather = 'average';

 food = 'wurst thing ever';

 currency = 'funny, small and colourful';

 }

 var message = 'this is ' + country + ', the weather is ' +

 weather + ', the food is ' + food + ' and the ' +

 'currency is ' + currency;

 alert(message);

}

</script>

```

你可以在我的那个 安全天气的 if 语句示例中自己试试看。你可以改变 `country` 变量的值，来看看不同的弹出消息。

但是前面那个例子会一直尝试创建消息，不管 `country` 是否合用——所以如果抛出一个错误，或者声明“此项未定义，天气是...”的话，这个版本的安全性就会好了。如果 `country` 未定义，就不会创建消息。

此外，你可以用“或”或“与”语句来将各种条件串联起来，以分别测试是否某个语句为真，或两个语句都为真。在 JavaScript 中，“或”的表达方式为 `||` 而“与”则写成 `&&`。假设你想测试 `x` 的值是否介于 10 和 20 之间——你就可以用条件声明 `if(x > 10 && x < 20)` 来实现。如果你想确保 `country` 是“England”或“Germany”二者之一，就可以用 `if(country == 'England' || country == 'Germany')`。

此外还有 `else` 子句，每次只要第一个条件不为真，该子句就会被执行。如果你想对任何

值做出反应，但要特别找出其中一个值来进行特殊处理的话，这个子句是非常有用的：

```
<script type="text/javascript">

var umbrellaMandatory;

if(country == 'England'){

 umbrellaMandatory = true;

} else {

 umbrellaMandatory = false;

}

</script>
```

条件是很有用的，但其作用稍微显得有限了些。如果你想反复进行某事的话又该怎么办呢？

假设你的工作是为数组中每个值添加一个段落标签的话该怎么做呢？如果只用条件语句的话，你就得为你可能会遇到的所有不同长度的数组逐个进行硬编码了：

```
<script type="text/javascript">

var names = new Array('Chris','Dion','Ben','Brendan');

var all = names.length;

if(all == 1){

 names[0] = '<p>' + names[0] + '</p>';

}

if(all == 2){

 names[0] = '<p>' + names[0] + '</p>';

 names[1] = '<p>' + names[1] + '</p>';

}

if(all == 3){

 names[0] = '<p>' + names[0] + '</p>';

 names[1] = '<p>' + names[1] + '</p>';

 names[2] = '<p>' + names[2] + '</p>';

}

if(all == 4){

 names[0] = '<p>' + names[0] + '</p>';
```

```
names[1] = '<p>' + names[1] + '</p>';

names[2] = '<p>' + names[2] + '</p>';

names[3] = '<p>' + names[3] + '</p>';

}

</script>
```

这真是很吓人，而且太不灵活了。程序设计的原意是使我们的工作更轻松，而如果你发现自己不断地重复编写同样的代码的话，你就可能在做错事了。良好的程序设计意味着将繁重的任务交给机器去做，而你只需要关注自己想要实现的目标就行了。

在这种情况下，我们就需要用**循环**来代替条件了，因为我们将数组中的每一项数据所做的是完全一样的事情，而数组的长度是无关紧要的。在下一章中我们会用循环来重新编写上面那个例子——对比一下这两段代码，你就会发现后者要精简得多！

## 循环

循环就是重复的条件，循环语句中会有一个变量随着每次循环而改变。最简单的循环形式就是 `for` 语句。该语句的语法与 `if` 语句相似，但选项要更多一些：

```
for(condition; end condition; change) {
 // 执行吧，立即执行
}
```

一般而言，你用 `for` 循环所做的就是将大括号中的代码重复执行若干次。为了实现这一点，你需要定义一个迭代器，并在循环中不断地改变它的值，直到该变量的值满足结束条件（结束条件会引发解释程序退出该循环，并继续下一部分代码）。比如：

```
<script type="text/javascript" charset="utf-8">

for(var i = 0; i < 11; i = i + 1) {
 // 执行吧，立即执行
}

</script>
```

此处我们定义了一个变量 `i`，其初始值为 `0`，然后进行检查，看是否该变量的值已经达到 `11`（该变量值是否小于 `11`?）。变动等式——`i = i + 1`——每循环一次就给 `i` 加上 `1`，然后进

行下一次迭代。这就意味着这项循环会执行 11 次。如果每次迭代给 `i` 加上 2，就只会执行 6 次了：

```
<script type="text/javascript">

for(var i = 0;i < 11;i = i + 2){

 // do it, do it now

}

</script>
```

利用循环，上面的段落添加示例就会变得精简得多，也简单得多了：

```
<script type="text/javascript">

var names = new Array('Chris','Dion','Ben','Brendan');

var all = names.length;

for(var i=0;i<all;i=i+1){

 names[i] = '<p>' + names[i] + '</p>';

}

</script>
```

注意，你也可以在循环中将 `i` 作为该数组的计数器。这就是循环的威力之所在——你不仅可以重复执行同样的任务；而且在每次迭代中你都知道自己已经进行了多少次迭代。

## 总结

这——从非常非常精简的角度上来说——就是程序设计。你采用了变量和用户输入，并通过一种或另一种方式对它们进行改动，比较，循环，并返回它们的值。没什么奇妙的魔术，也不会太令人困惑，只不过是对于事物如何运转的一个非常简化的观点。在本文中我们还没有涉及到函数，但可以这么说，一旦你对某个任务进行了编程，而且你的程序在重复使用方面具有意义的话，你就可以将这段代码变成一个函数，这样就可以在任何需要用到该功能的地方重复执行你的代码了。在本教程随后的文章中将对函数进行更加详细的讨论。现在，我希望你对这些概念的认识比刚开始要稍微清楚一点儿了。

## 练习题

- 为什么 `var x = hello world` 这段代码会失败？
- 这段代码合法吗？`var x = 'elephant';var y = "mouse";`

- 这个条件语句所测试的是什么? `if(x > 10 && x < 20 && x != 13 && y < 10)`
- 这个条件是做什么的? `if(b < 10 and b > 20)?`
- 对带有 “peter”,“paul”,“mary”,“paddington bear”,“mr.ben”,“zippy”和“bagpuss”这些数据项的数组进行循环。给每个数据项添加一个段落并给每第偶数个数据项就添加一个 `class` 为“odd”的段落。提示: 你可以通过同余运算符 `i % 2 == 0` 来测试每第偶数个数据项。
- 上一篇: [页首、页脚、列和模板](#)
- 下一篇: [你可以用 JavaScript 做什么?](#)
- [目录](#)

## 作者简介



Chris Heilmann 从事 Web 开发工作已达 10 年，之前他曾从事电台新闻工作。他在英国做 Yahoo! 的培训师和高级开发工程师，同时负责监控面向欧洲和亚洲的前端代码质量。

Chris 的博客是 [Wait till I come](#)，在许多社交网络上都可以通过输入“codepo8”找到他。

## 40. 你可以用 JavaScript 做什么？

Posted 03/10/2009 - 23:01 by yuanc

- 网络标准教程

作者: Christian Heilmann · 2009 年 2 月 3 日

- 上一篇: 程序设计——真正的基础!
- 下一篇: 初识 JavaScript
- 目录

### 引言

我们已经探讨过了必要的程序设计核心概念。现在是时候从细节中退出来，并高瞻远瞩地来看看你可以利用 JavaScript 来做些什么了——为什么你要花时间来学习这样一门复杂的功课，以及如何将其用到你的网页上？

过去的几年对我来说是一段有趣的时间，因为在这段时间里 JavaScript 的使用从一种边缘知识成为了 web 开发技能的主流。现在如果没有 JavaScript 技能的话，要想得到 web 开发师的工作是很艰难的。

让我们继续前进——本文的结构如下：

- 我是怎样喜欢上 JavaScript 的
- JavaScript 的缺点
- JavaScript 可以为你做些什么
- JavaScript 的常见应用
  - 进入 DOM 脚本
  - JavaScript 的其它时新应用
- 巧妙而负责地运用 JavaScript
- 总结

### 我是怎样喜欢上 JavaScript 的

当我第一次遇到 JavaScript 的时候，计算机的运行变得很慢，浏览器在解释 JavaScript 方面很糟糕，看起来 JavaScript 整个儿是个坏主意。我以前是从事后端开发的——将你要实现的所有功能都用 Perl 来编写，一切就不会有风险。

另一方面，互联网的速度很慢，文件的寄存成本非常高，这就是 **JavaScript** 产生的原因。该语言的执行是在访问网页的用户的计算机上进行的，这就意味着你用 **JavaScript** 编写的任何东西都不会增加你的服务器的处理负担。这就使得你对网站终端用户的响应灵敏得多，而且从服务器流量方面来说费用也更少。

近些年来，浏览器在执行 **Javascript** 方面要好一些了，计算机变得更快了，而带宽也便宜了许多，因此 **JavaScript** 的许多缺陷都变得不那么要紧了。无论如何，通过用 **JavaScript** 制作网页来减少与服务器的交互，都产生了响应更加灵敏的 **web** 应用程序，以及更好的用户体验。

### **JavaScript** 的缺点

尽管在当今时代已经有了很大的进步，**JavaScript** 仍然有个缺陷：**JavaScript** 是古怪的。这不是指该语言本身，而是指它的执行环境。你并不知道是自己网页的接收端是什么样的计算机，也不知道那台计算机是不是在做其它事情，到底有多繁忙，而且也不知道在浏览器的另一个标签中是否打开了其它的 **JavaScript**，而导致一切慢得要停止。除非大多数的浏览器都对各个标签和窗体拥有不同的处理资源（又叫线程），这仍将是个问题。

此外，由于安全性的问题，或者是因为 **JavaScript** 经常会被用来制作搅扰用户的东西，而不是用来提升用户体验，它常常会被人们在浏览器中关掉。比如说，你会发现有许多网站违背你的心意，试图跳出一些弹出窗口，或者是用广告遮掉页面内容，除非你点击某个链接来除掉它。

### **JavaScript** 可以为你做些什么

让我们退后一步，来数算一下 **JavaScript** 的优点：

- **JavaScript** 非常容易执行。你要做的仅仅是将它嵌入到 **HTML** 文档中，并告诉浏览器它是 **JavaScript** 即可。
- **JavaScript** 是在 **web** 用户的电脑上工作的——即使当他们不在线时也可以！
- **JavaScript** 使你能够创建高响应度的界面，该界面可以提升用户体验，并提供动态的功能，而不需要等待服务器响应并显示另一个页面。
- 如果用户需要的话，**JavaScript** 可以将内容载入文档，而不用重新加载整个页面——我们一般把这个叫做 **Ajax**。
- **JavaScript** 可以对你的浏览器可能支持什么进行测试并相应地做出反应——这叫做 **非干扰性JavaScript** 的原理，有时也叫做防御性脚本。
- **JavaScript** 有助于解决浏览器的问题，或为浏览器支持上的漏洞打补丁——比如解决某种浏览器中的 **CSS** 布局问题。

对于一门到最近为止一直被那些喜欢“更高级的程序设计语言”的程序员所蔑视的程序设计

语言来说，这些优点已经很多了。JavaScript 复兴的一部分原因在于我们现在所创建的 web 应用程序正变得越来越复杂，而要达到高交互性，不是需要 Flash（或其它插件），就是需要脚本。JavaScript 按理应该是最好的办法，因为它是一个 web 标准，生来就具有跨浏览器支持性（或多或少是这样的——不同的浏览器各有差异，而这些差异将在下一篇文章中适当的地方加以讨论），而且它与其它的开放 web 标准是相容的。

### JavaScript 的常见应用

JavaScript 的使用在我们对其进行利用的这些年间已经发生了变化。起初，网站上的 JavaScript 交互大多数都仅限于与表单的交互，为用户提供反馈信息，以及检测用户是否进行了特定操作。我们用 `alert()` 来对用户进行通知（如图 1 所示），用 `confirm()` 来询问用户是否要做某些操作，用 `prompt()` 或表单域来获取用户输入。

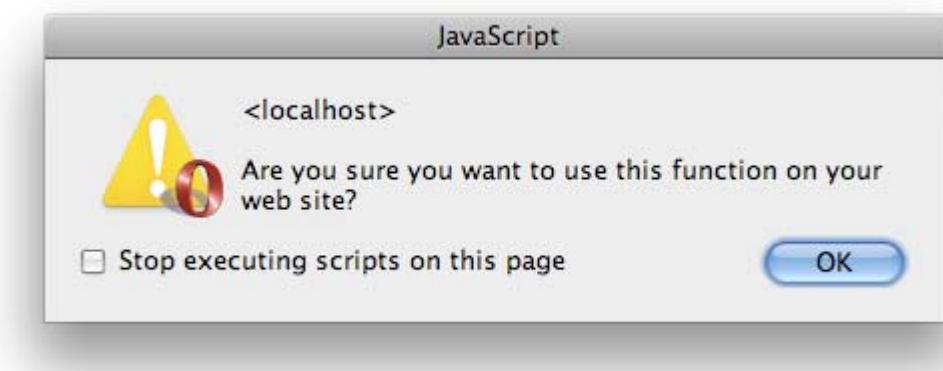


图 1：用 `alert()` 语句来对终端用户进行错误信息通知，这是我们在 JavaScript 早期唯一可以做的事情。这既不好看也不精巧。

在出错时，上面的这个例子通常会触发阻止用户向服务器发送某个表单的校验脚本，以及简单的转换器和计算器。此外，我们还成功地创建了像提示语这样的没什么使用价值的东西来询问用户名，仅仅只为了在下一刻将其打印出来。

我们以前还会用到的另一个函数是 `document.write()`，用来向文档中添加内容。我们还用过弹出窗口和框架，然后大大失去了继续研究的勇气，我们还曾经绞尽脑汁地试图使它们相互对话。琢磨这些技术可以叫任何一个开发师痛苦得前后摇晃，抱头痛吟“让它们离我远点”，所以我们还是不要老谈论这些东西——JavaScript 有更好的使用方法！

### 进入DOM脚本

在浏览器开始支持和执行 文档对象模型(DOM)的时候，JavaScript 就开始变得更加有趣了，文档对象模型可以让我们与网页之间的交互丰富得多。

DOM 是文档的一种对象表征。比如，前面那个段落（可以用“查看源文件”来查看其源代码）

从 DOM 的角度来讲，就是一个 `nodeName` 为 `p` 的元素节点。它包含了三个 `child nodes`——一个 `nodeValue` 值为“当浏览器开始支持和执行”的文本节点，一个 `nodeName` 为 `a` 的元素节点，以及另一个文本节点，其 `nodeValue` 为“JavaScript 就开始变得更加有趣了，文档对象模型可以让我们与网页之间的交互丰富得多。”。`a` 节点还有一个叫做 `href` 的属性节点，其值为 “`http://www.w3.org/DOM/`”，以及一个 `nodeValue` 为“文档对象模型(DOM)”的文本子节点。

你也可以用一张树状图来直观地表示这个段落，如图 2 所示。

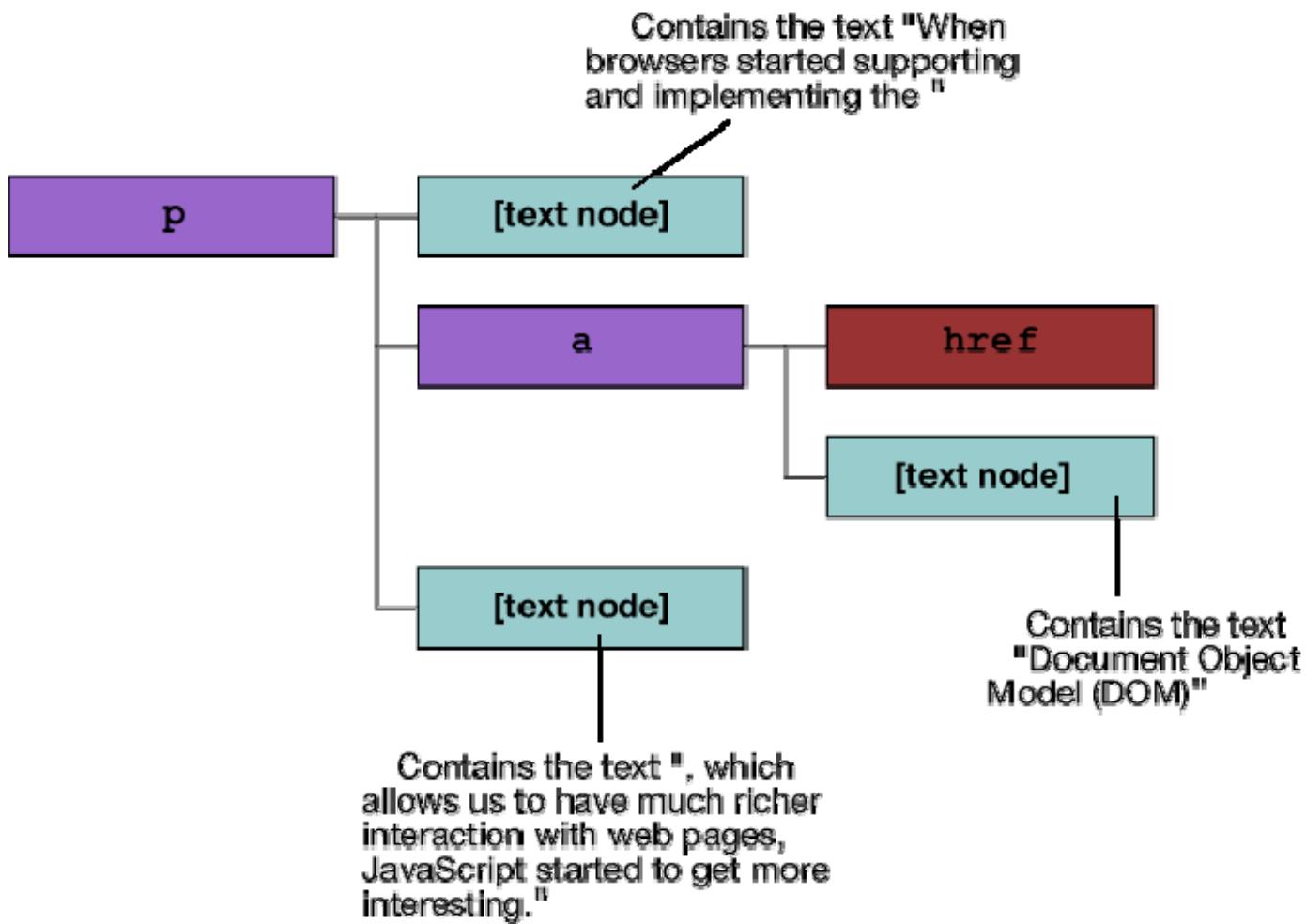


图 2：我们用来做范例的 DOM 树的直观表示。

用人类的语言来讲，你可以说DOM说明了该文档中一切内容的类型，值和层级——目前你并不需要知道得更多了。要了解关于DOM的更多信息，请参阅本教程后面的 [DOM之旅](#)一文。

利用 DOM，你可以：

- 访问文档中的任意元素并对其外观，内容和属性进行操作。

- 创建新元素和内容，并在需要时将它们应用到文档中去。

这就意味着我们可以不必再依赖于窗体，框架，表单和不咋好看的通知对话框了，而且可以用一种样式化得很好的形式为用户提供反馈信息，如图 3 所示：

## Oops!

We couldn't save your profile as entered. Please take a look at the following:

- >Login has already been taken
- Email address doesn't match confirmation

Desired Username	mills
	Must be at least 4 characters
Email	sample@sample.com
Retype Email	
Password	*****
Retype Password	*****

图 3：通过 DOM 你可以创建更美观而又不那么唐突的报错信息。

DOM 再加上事件处理，就构成了一个非常强大的装备库，可以创建出交互式的漂亮的界面。

事件处理是指我们的代码对浏览器中发生的事件做出反应。事件处理的对象可以使浏览器中自动发生的事件——比如页面载入完成——但大多数时候我们是针对用户对浏览器进行的操作做出反应。

用户可能会调整窗口大小，滚动页面，敲击特定按键或用鼠标点击链接，按钮以及元素。利用事件处理，我们就可以等这些事情发生的时候告诉 web 页面对这些行为做出我们想要的响应。但在以前的时候，对任何链接的点击都会为网站访客打开另一个文档，现在我们可以对这个功能进行劫持，并实现其它的效果，比如显示和隐藏某个面板，或者是获取该链接中的信息并用它来连接到 web 服务。

在本教程随后的 [JavaScript 中的事件处理](#)一文中，对事件的讨论要详细得多。

### JavaScript 的其它时新应用

这基本上就是指我们目前用 JavaScript 所做的事情。我们对那些老式的，可靠而且正确的

web 界面进行提升——链接的点击，信息的输入以及表单的发送，等等——从而使该 web 界面对终端用户的响应更加灵敏。比如说：

- 一张注册表单可以在你输入用户名的时候核对该用户名是否可用，免得你忍受令人郁闷的页面重新加载。
- 搜索框可以在你进行输入的时候为你提供建议的结果，该建议是基于你当时已输入的内容的（比如说，输入“bi”可能会产生供你选择的包含该字符串的建议，例如“bird”，“big”和“bicycle”）。这种使用方式叫做 [自动补全功能](#)。
- 经常发生变化的信息可以定时加载，而不需要经过用户的介入，比如体育比赛结果或股市行情显示器。
- 可有可无的而又对某些用户来说有多余之嫌的信息可以只在用户选择要访问它们的时候才被载入。比如说，某个网站的导航菜单可能有 6 个链接，但只会在用户激活某个菜单项的时候按需显示那些连接到更深层的页面的链接。
- [JavaScript](#) 可以解决布局问题。利用 [JavaScript](#) 你可以找到页面上任何一个元素所在的位置和区域，以及浏览器窗口的尺寸。利用这些信息你就可以避免元素的重叠和其它类似的问题。比如说，假设你有一个若干层级的菜单；通过在其显示之前检查是否有足够的空间来显示子菜单，你就可以避免滚动条的出现和菜单项的重叠。
- [JavaScript](#) 可以对 [HTML](#) 提供的界面进行美化。虽然页面上有一个文本输入框也不错，你可能还是会想要一个下拉列表框来让你在一个预设值列表中进行选择或自行输入。通过 [JavaScript](#) 你就可以增强一个普通输入框的功能，来实现这个效果。
- 你可以用 [JavaScript](#) 来为页面上的元素添加动画效果——比如显示和隐藏信息，或高亮显示页面上的特定部分——这个用处将有利于实现更实用，更丰富的用户体验。在本教程随后的 [JavaScript 动画](#) 一文中有更多关于这一点的信息。

## 巧妙而负责地使用 [JavaScript](#)

使用 [JavaScript](#) 你几乎什么都能做——尤其是当你将它与诸如 [Canvas](#) 或 [可缩放矢量图像 \(SVG\)](#) 之类的技术结合使用的时候。然而，能力越强，责任越大，在使用 [JavaScript](#) 的时候你应当牢记下面这些原则。

- [JavaScript](#) 可能会不可用——不过这种情况很容易检测到，因此实际上算不得一个问题。不过在创建依赖于 [JavaScript](#) 的效果时一定要记住这一点，同时你还应该当心，确保你的网站在 [JavaScript](#) 不可用时不会崩溃（例如，必不可少的功能变得不可用）。

- 如果 **JavaScript** 的使用不能帮助用户更快速而高效地达成某个目标，那么你对 **JavaScript** 的使用方式可能是错误的。
- 利用 **JavaScript** 我们常常会打破人们在长期使用网络的过程中所习惯了的惯例(比如，点击链接以跳转到另一个页面，或一个小筐子的图标就表示“购物车”)。虽然这些使用模式可能已经过时了，而且效率又低，对它们的改变还是意味着迫使用户改变自己所习惯的方式——而这会使人们觉得不自在。我们都喜欢拥有支配权，而且一旦我们学会了某事，就很难适应改变。你的 **JavaScript** 解决方案用起来的感觉应该比以前的交互要好些，但不要跟原来差别太大，以致用户无法将其与自己以前的经验联系起来。如果你成功地使一个网站访客说出“啊哈——这样我就不用干等着了”或者“酷——现在我就可以不必经过这个非常烦人的步骤了”这样的话，你就很好地使用了 **JavaScript**。
- **JavaScript** 绝不能作为一种安全措施。如果你需要防止用户访问数据，或者你可能要处理敏感数据的话，就不要依靠 **JavaScript**。任何 **JavaScript** 的保护措施都很容易被反向生成并克服，因为所有的代码都能在客户端主机上读到。此外，用户也可以在自己的浏览器中直接关掉 **JavaScript**。

## 总结

**JavaScript** 是一种用在网上的极好的技术。它不是非常难学，而且用途也很多。**JavaScript** 可以很好地与其它 **web** 技术一起使用——比如 **HTML** 和 **CSS**——而且甚至可以与 **Flash** 之类的插件进行交互。**JavaScript** 使我们可以创建高响应度的用户界面，避免令人郁闷的页面重新加载，甚至还可以解决 **CSS** 的支持问题。通过适当的浏览器附加软件（比如谷歌离线应用开发工具或雅虎 **Browser Plus**），你甚至可以用 **JavaScript** 来使联机系统在离线时仍然可用，并且在计算机联线之后立刻实现自动同步。

**JavaScript** 也不受浏览器的限制。与其它程序设计语言相比，**JavaScript** 的速度更快，内存占用更小，这使得它的应用变得越来越普及——从对像汇稿员一样的程序上的重复性工作进行自动化，直到作为一种带有独立的语法分析器的服务器端语言，都有它的影踪。**JavaScript** 的发展前途是光明的；不管你作为一个 **web** 开发师在不久的将来会做些什么，我确信你迟早会用到 **JavaScript**。

- 上一篇：程序设计——真正的基础！
- 下一篇：初识 **JavaScript**
- 目录

## 作者简介



Chris Heilmann 从事 Web 开发工作已达 10 年，之前他曾从事电台新闻工作。他在英国做 Yahoo! 的培训师和高级开发工程师，同时负责监控面向欧洲和亚洲的前端代码质量。

Chris 的博客是 [Wait till I come](#)，在许多社交网络上都可以通过输入“codepo8”找到他。

## 41. 初识 JavaScript

Posted 03/10/2009 - 23:09 by yuanc

- 网络标准教程

## 41:初识 JavaScript

作者 Christian Heilmann · 2009 年 2 月 3 日

- 前一篇文章——你可以用 JavaScript 来做些什么？
- 后一篇文章——JavaScript 最佳实践
- 目录

### 引言

在本文中，我们将学习 JavaScript 基础——如何使用，以及应该在何处使用 JavaScript；使用中应避免哪些问题；以及那些成为拔尖 JavaScript 开发者所必需的一般性的基础知识。

本文结构如下：

- 什么是 JavaScript？你该如何执行它？
  - 将你的 JavaScript 代码包含在 HTML 文档内
  - 链接到外部 JavaScript 文件
- JavaScript 和浏览器性能
- JavaScript 该放在哪儿
- JavaScript 的安全性以及它的不足
- 应该避免使用的技巧
- 总结
- 练习题

### 什么是 JavaScript？你该如何执行它？

JavaScript 是一种基于文本的程序设计语言，在被执行之前不需要进行任何转换。其它程

序设计语言比如 **Java** 和 **C++** 都需要先进行编译，才能成为可执行文件，而 **JavaScript** 可以通过一种用来解释代码的程序立即被执行，这种程序叫做解析器（几乎所有的 **web** 浏览器都包含有 **JavaScript** 解析器）。

为了在浏览器中执行 **JavaScript**，你有两个选择——要么将其放在 **HTML** 文档中任意位置的 **script** 元素内部，要么将其放在一个外部 **JavaScript** 文件（其扩展名为 **.js**）中，然后在 **HTML** 文档内部用一个带有 **src** 属性的空的 **script** 元素来引用该外部文件。本文对这两种方法都有讨论。

### 将你的 **JavaScript** 代码包含在 **HTML** 文档内

最简单的在 **HTML** 页面中包含 **JavaScript** 的方法可能如下：

```
<script>

var x = 3;

alert('hello there, I am JavaScript - x is '+x);

</script>
```

你可以将这段代码放在你的文档内的任意位置，这样就可以执行这段代码了。但有的地方比起其它地方来明显要好得多——关于这一点的具体建议请参见 **JavaScript** 该放在哪儿部分。

在未来的 **web** 页面上可能会有若干不同类型的脚本，所以最好为你的脚本添加一个 **MIME** 类型：

```
<script type="text/javascript">

var x = 3;

alert('hello there, I am JavaScript - x is '+x);

</script>
```

**注意：**你会发现在网上有些脚本带有 **language="javascript"** 属性。这种使用方法不属于任何标准，而且完全不起任何效果的；你要是能删的话就把它们都删掉好了。这种用法是过去的做法，那时 **VBScript** 在 **web** 页面上的使用还很流行。然而现在 **VBScript** 的使用已经销声匿迹了，因为它只在 **IE** 中才能起效。

过去我们需要把 **JavaScript** 写成 **HTML** 注释形式，以防止浏览器直接将 **JavaScript** 代码作为 **HTML** 显示出来。不过现在你不用再为此费心了，因为现在只有那些非常老旧的浏览器

才会这么做。如果你的 **DOCTYPE** 用的是严格型的 **XHTML**，那么任何 **JavaScript** 就都得放在 **CDATA** 数据块中，这样它才能生效（不必操心这是为什么——在你目前的学习阶段中，这个原因并不是很重要）：

```
<script type="text/javascript">

/* <! [CDATA[*/

var x = 3;

alert('hello there, I am JavaScript - x is ' +x);

/*]]> */

</script>
```

不过，对于严格型的 **XHTML** 文档来说，最好还是不要在其中嵌入任何 **JavaScript**，而应该将它们放在一个外部文档中。

### 链接到外部 **JavaScript** 文件

要链接到一个外部 **JavaScript** 文件（该文件可以是在同一台服务器上，也可以是在互联网上任意之处），你只需在你的 **script** 元素中加入一个 **src** 属性即可：

```
<script type="text/javascript" src="myscript.js"></script>
```

当浏览器在某个页面中遇到该元素时，就会加载并执行 **myscript.js** 文件。如果你设置了 **src** 属性，那么在该 **script** 元素内部的任何内容都将被忽略。下面的示例将加载 **myscript.js** 文件，并执行该文件内部的代码，但却根本不会执行 **script** 元素内的 **alert** 。

```
<script type="text/javascript" src="myscript.js">

alert('I am pointless as I won\'t be executed');

</script>
```

将你的代码放在外部 **JavaScript** 文件中有很多好处：

- 你可以将同一个 **JavaScript** 功能应用到若干 **HTML** 文档中去，而维护却依然十分容易：如果需要改变该功能的话，你只需修改一个文件就可以了。
- 你的 **JavaScript** 可以被浏览器缓存。缓存的意思是指浏览器将复制一份你的 **JavaScript**，并将其保存在浏览你网站的访问者的计算机里。下次该用户加载该脚本的时候

就不再需要从你的服务器下载了，他可以直接从自己的计算机中获取——因此载入速度会快得多。

- 如果你要修改自己的脚本的话，很容易就能找到它，从而避免了扫描长长的 HTML 文档来查找需要修改的地方。
- 由于调试工具或错误控制台会告诉你哪个文件中包含错误，而且还可以准确地报告出错的代码行号，错误调试也变得更容易了。

你可以往一个文档中添加任意数目的 **JavaScript** 文件，但在你开始这么干之前还有几件事情要考虑。

### **JavaScript 和浏览器性能**

将大量的 **JavaScript** 分成不同的文件，每份文件每次处理一项任务，对于保持功能维护的简便和快速漏洞修复来说这是一个极好的办法。比如说你可能会有像下面这样的若干脚本块：

```
<script type="text/javascript" src="config.js"></script>

<script type="text/javascript" src="base.js"></script>

<script type="text/javascript" src="effects.js"></script>

<script type="text/javascript" src="validation.js"></script>

<script type="text/javascript" src="widgets.js"></script>
```

这种做法在开发方面的优点却被它在性能方面的负面影响削弱了。虽然这种情况在不同的浏览器上有轻微的差别，但在最坏的情况下（令人遗憾的是这种情况发生在最常用的浏览器中）会发生下面的情况：

- 浏览器每遇到一个 `script` 元素，都会停止渲染（显示）该文档。
- 然后浏览器会加载在 `src` 属性中定义的 **JavaScript** 文件（如果你要用的脚本位于另一台服务器上的话，你还得等待浏览器找到那台服务器）。
- 然后浏览器将在访问下一个脚本之前先执行这一脚本。

以上种种意味着你网站的显示将变得缓慢，直到你页面中包含的所有脚本都经历完上述步骤为止。对于你的访客来说，这可是会非常糟糕的。

解决这个问题的办法之一是使用后端脚本，根据你所使用的所有文件来创建一个单独的文件。通过这种办法你就既可以保持维护的简便性，同时又可以减少自己网页在显示时的延迟。网

上有许多完成此任务的后端脚本——其中有一个是用 PHP 写的，可从 [Ed Elliot](#) 获取此脚本。

显示上的延迟还限定了你该将自己的 **JavaScript** 放在文档中的什么地方。

### JavaScript 该放在哪儿

从技术上来讲，你可以将 **JavaScript** 放在文档内的任意地方。你要做的决定就是，在性能与开发方便之间进行权衡，并确保你用来增强页面功能的 **JavaScript** 能够立刻产生效果。

传统的脚本放置最优方法是放在文档的 `head` 之内：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html lang="en-en">

<head>

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

<title></title>

<script type="text/javascript" src="myscripts.js"></script>

</head>

<body>

<!-- lots of HTML here -->

</body>

</html>
```

这种做法的好处在于脚本的位置是可预测的——开发者知道该到何处去找它们。这种做法还有一个优点就是可以确保在文档显示之前，所有的 **JavaScript** 都已被加载并执行。

而这种方法的缺点则在于，你的脚本会延迟文档的显示，而且该脚本无权使用文档中的 **HTML**。因此你就只得延迟所有会改变该文档 **HTML** 的脚本的执行，直到 **HTML** 文档加载完成为止。通过 `onload` 事件处理器或各种各样的 `DOMready` 或 `contentAvailable` 技术可以实现这一点——当然，它们都不是无懈可击的，而且其中绝大多数都依赖于特定的浏览器特性。

性能优化专家们最近开始提倡将 **JavaScript** 放在 `body` 的末尾：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
```

```
"http://www.w3.org/TR/html4/strict.dtd">

<html lang="en-en">

<head>

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

<title></title>

</head>

<body>

<!-- lots of HTML here -->

<script type="text/javascript" src="myscripts.js"></script>

</body>

</html></pre>
```

这种做法的好处是可以不会延迟 **HTML** 的显示，而且还意味着任何你想要用 **JavaScript** 来修改的 **HTML** 都已经是可用的了。

这种方法的缺点之一是此方法目前还不太常用，因此可能会使维护你的代码的开发师感到困惑。另一个更严重的缺点是 **HTML** 在你的 **JavaScript** 被加载之前就变得可用。虽然这也许正是你想要达到的效果，但它同时也意味着访问者在你通过 **JavaScript** 修改 **HTML** 之前就已经开始与你的界面进行交互。比如说，假定你想要在一张表单被提交到服务器之前用 **JavaScript** 来对其进行校验——该表单可能会在脚本加载之前就被提交了。如果你只是为了增强页面功能而编写脚本（而不是整个页面都依赖于该脚本），这个缺点倒也不是什么问题——只不过会比较恼人罢了。

具体哪种方法更适合你的网站，这完全由你决定；你甚至可以把两种方法混起来用——将功能非常重要的脚本放在 **head** 内，并将“可有可无”的脚本放在文档末尾。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

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

```
<head>

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

<title></title>

<script type="text/javascript" src="myimportantscripts.js"></script>

</head>

<body>

<!-- lots of HTML here -->

<script type="text/javascript">

applyFunctionality();

</script>

<script type="text/javascript" src="myscripts.js"></script>

</body>

</html>
```

不管你将 **JavaScript** 放在何处，都应该确保你的脚本的放置顺序是正确的，因为浏览器会依次加载并解析。由于这个特性，我们在使用 **JavaScript** 的时候还得顾及另一个问题。

### **JavaScript** 的安全性及其不足

对于这一点，无论如何强调也不为过。**JavaScript** 是一门非常好的程序设计语言，可以帮助你建立反应迅速且漂亮的网站和应用程序，但它有一个非常严重的缺点，那就是安全性。简而言之，**JavaScript** 中没有安全模型，你绝不应当用 **JavaScript** 来对任何重要或机密的东西进行保护，加密，保障或存储。

页面上的每个脚本都有同样的权限——它们都可以互相访问，读取变量，访问函数，以及互相覆盖。如果页面中的第一个脚本内有个叫做 `init()` 的函数，最后一个脚本中也有一个同名的函数，那么第一个函数就会被覆盖。在本教程中的 [最佳实践](#)一文中我们会再回来讨论这个问题。

如果你从头到尾都没有用过别人的脚本，上面这些倒也不是大问题。然而，由于大多数线上广告和统计跟踪都是用 **JavaScript** 来做的，情况就完全不一样了——你得一直使用第三方脚

本。

脚本还可以用来读取 `cookies`, 而且使用函数的 `prototype` 你还可以覆盖任何一个原生的 `JavaScript` 函数。最后, `JavaScript` 还可以轻易地被关掉, 所以, 你还是别再考虑把 `JavaScript` 保护作为一种良好的安全措施了。

`JavaScript` 总是很容易被其它开发者所读取和分析。当然你也可以将你的脚本打包（移除所有不必要的空格）并加以混淆（使用随机变量和函数名），但即便是这些做法也可能被反向工程破解；因此这样做只能防止你自己使用你的代码。源代码的可获得性以及对源代码的阅读和分析能力可能是 `JavaScript` 得以流行的主要原因——长时间以来我们都靠偷看别人的代码来进行学习。幸运的是，现在这种状况已经结束了，我们现在有很好的书籍和教程。

尽管打包和混淆在安全措施方面没什么价值，但在将代码放到网上去之前，它们还是经常作为发布过程的一部分被应用在大中型脚本上。这样做有助于减少向用户提供网站服务所需的带宽。零零散散地这里省几个字节，那里省几个字节，可能对你关于小猫的博客没多大意义，但在对 `google.com` 之类的大型网站节约数量就相当可观了。

### 应该避免使用的技巧

学习 `JavaScript` 最大的问题就是，现在有大量过时而且可能不安全的信息。尤其令人沮丧的是这类信息的表述通常非常好，而且通过复制和粘贴某些现成的代码，可以给许多初学者造成一种“速成”的感觉。

由于 `JavaScript` 的应用环境是完全未知的（用户的设置可能是五花八门的），而且我们并不知道我们在网上所找到的代码到底是出于什么样的决定，才创建成这种特殊形式的，我们就不应该指责那些解决方案。然而，下面这些办法都是已经过时的了，你只应在需要支持那些老旧落后的浏览器的时候才使用这些方法。

- `document.write()` — 你可以用 `document.write()` 来将内容写进文档中，但这种做法存在若干问题：你这样做就是将 `HTML` 与脚本混合起来，并且你还得十分精确地在自己想要该内容出现的地方添加一个脚本节点，而这样会使你的页面变慢。对于方便地显示某段代码的结果（比如说在教学过程中或在测试/调试你的代码时）来说，这是一个很好的办法。但它不能作为一个向用户展示页面的方法，你不应将它应用在实际代码中。
- `<noscript></noscript>` — 就像其名字所指的那样，`noscript` 元素刚好与 `script` 元素相反。在该元素内的内容将显示给那些禁用了 `JavaScript` 的用户看。使用 `noscript` 的主要原因是为了向那些浏览器中 `JavaScript` 不可用的用户提供替换内容。那些没有开启 `JavaScript` 的用户并不是成心想要为难你才那么做的——他们之所以这样做要么是出于安

全的考虑，要么是因为他们所使用的浏览器不支持 **JavaScript**。但可以放在 `noscript` 内来提供的内容是有限的，且你不应将其用来通知用户启用 **JavaScript**——因为对某些用户来说这是不可能做到的。

- `<a href="javascript:doStuff()">...</a>` — 这是一种非常常见的调用 **JavaScript** 功能的方法，大多数时候这种方法是用在不便使用按钮的情况下（在老式浏览器中你无法对按钮进行样式化）。问题在于这其实不是一个合法的链接，因为 `javascript` 并不是一个 **internet** 协议（比如 `ftp://` 或 `http://`）。如果 **JavaScript** 被禁用，该链接也还是会被显示，并会给予用户造成一种错误的期待，让他们觉得有点击后会随之发生什么事情。
- `document.layers` 和 `document.all` — 这两个是在老式浏览器中使用是 **DOM** 的解决方法（分别对应于 **Netscape 4.x** 和 **IE4**），而且如果你不需要支持这些浏览器（但如果你不得不这样做的话，我很同情...）的话，就完全不需要使用。

## 总结

初识 **JavaScript** 就到此结束了。本文没有对大量细节进行讲解，而是旨在为读者提供一个概述，告诉读者如何才能最好地将 **JavaScript** 应用到页面中去，以及该避免哪些问题。在下一篇教程中我们将通过一个真实的代码示例来讨论 **JavaScript** 编写最佳实践。

## 练习题

- 下面的链接是用来做什么的？它将会产生什么问题？

```
Read our Terms and Conditions
```

- 向脚本提供参数是一个很有效的提高脚本复用性的方法。使用结构紧凑而又易于使用的参数是很重要的。下面的代码（这段代码的参数结构紧凑又易于使用）的缺点是什么？

- `<script src="badge.js">`
- `var color = 'blue';`
- `var background = 'yellow';`
- `var width = 400;`

```
</script>
```

- 所谓的“全局变量”的问题是什么，怎样避免使用全局变量？

- 如果某个大型脚本是属于可有可无那一类的，而且对于网站的功能来说并不重要的话，你应该将它放在文档中的什么地方？为什么？
- 下面的脚本有什么问题？

```
<body onload="init()>
```

- 前一篇文章——你可以用 JavaScript 来做些什么？
- 后一篇文章——JavaScript 最佳实践
- [目录](#)

## 作者简介



Chris Heilmann 从事 Web 开发工作已达 10 年，之前他曾从事电台新闻工作。他在英国做 Yahoo! 的培训师和高级开发工程师，同时负责监控面向欧洲和亚洲的前端代码质量。

Chris 的博客是 [Wait till I come](#)，在许多社交网络上都可以通过输入“codepo8”找到他。

## 42. JavaScript 最佳实践

作者 Christian Heilmann (未完成)

## 43. 非干扰性 JavaScript 的原则

Posted 03/10/2009 - 23:10 by yuanc

- 网络标准教程

## 43: 非干扰性 JavaScript 的原则

作者 ppkoch · 2009 年 2 月 3 日

- 上一篇文章—JavaScript 最佳实践
- 下一篇文章—JavaScript 函数
- 教程目录

### 引言

在前面的文章中我们讨论了 HTML 和 CSS 以及如何正确地使用它们，我们还探讨了 JavaScript 的基础知识——JavaScript 是什么，能够用来做什么，以及为什么你应该了解它。在我们开始详细地探讨如何实际应用 JavaScript 之前，我们需要停下来思考一下如何聪明地使用 JavaScript，从而确保不会因为它的缘故使任何人无法访问你的网站。这就是**非干扰性的 JavaScript** 背后的核心理念。为了更好地理解什么是非干扰性 JavaScript 以及我们为什么需要它，我们先来看看在 JavaScript 程序设计中会碰到哪些不确定性因素：

- 因为某些浏览器不支持 JavaScript 或它们的支持的 JavaScript 版本太老了，这些浏览器可能会完全忽略你的脚本。
- 即使一个浏览器可以支持 JavaScript，用户们也可能会出于安全性的考虑而禁用 JavaScript，用户所在公司的防火墙也可能通过移除所有的 <script> 标签来阻止 JavaScript 的运行。

- 即便一个浏览器支持 **JavaScript**，它也可能不理解 **DOM** 规范中的某些浏览器专有特性（**IE** 经常被认为是这方面的祸首），这会导致该浏览器无法理解你的部分脚本。
- 即便是脚本能得到正确地解释，它也可能要依赖于非常复杂的 **HTML**，并且/或者依赖于可能以无法预测的方式被改变的 **HTML**。
- 就算 **JavaScript** 脚本运行在完美无误的 **HTML** 中，你也不能确定你的用户将会用什么类型的输入设备。许多脚本只有在用户使用鼠标时才会生效，而对那些使用键盘的人群则没有反应（许多残障用户无法使用鼠标，而且有的人偏偏喜欢用键盘）。
- 即使你的脚本回避了上述的所有风险而且运行得很好，其他的程序员们也有可能读不懂它。

这真是一条很长的问题清单，大多数问题从本质上来说并不是技术性的，而是概念性的问题，或者，如果你愿意这么说的话，是哲学上的问题。此外，尽管最后两个问题也可能会发生在其它程序设计语言中，但前面四个则是 **JavaScript** 程序设计环境所独有的问题，因此编程经验在这里也帮不上忙。

总之，非干扰性的 **JavaScript** 是一种编写 **JavaScript** 的方式，通过这种方式你网站的访客们就不会由于上述原因而被挡在你的网站外面了——即使你的 **JavaScript** 不能恰当地为他们效力，这些访客也仍然能够使用你的网站，尽管他们只能使用更为初级的功能。本文将会讨论非干扰性的 **JavaScript** 的原理和实际应用，并会为本教程随后的学习打下坚实的基础。

本文结构如下：

- 非干扰性的 **JavaScript** 的定义
- 分离结构与行为
- 添加一个可用性层
  - 示例——表单校验
  - 原理
  - 示例——弹出式窗口
  - 示例——**Ajax**
- 规则的，语义性的 **HTML**
- 浏览器兼容性
- 总结

## 非干扰性 **JavaScript** 的定义

为了有效地解决上面列出的问题，了解你在做什么以及这么做的原因是极其重要的。我们

需要一套关于正确开发 **JavaScript** 的理论。

非干扰性的 **JavaScript** 提供的就是这样的理论。它本身并不是一种技术；非干扰性的 **JavaScript** 并不是在你的脚本中添加一个 `makeUnobtrusive()` 函数来使你一劳永逸。它其实是一种思维方式。为了确保你的脚本不会给任何人添麻烦，你应该确保你的脚本不需要任何前提假设。

就目前而言，“不要做任何假设”实在有点苛刻了。幸运的是我们可以将非干扰性归为三类：你的脚本不应该干扰用户、浏览器、或同事程序员。

- **假设：每个人的浏览器都支持 **JavaScript**。** 错：为了达到不干扰用户的效果，应该做到去掉脚本也不会妨碍他们使用你的网站，即使这些用户能够使用的交互会比那些浏览器能够支持 **JavaScript** 的用户减少很多。
- **假设：所有浏览器的工作方式都是一样的。** 错：为了做到不干扰浏览器，你的脚本应该避开错误和兼容性问题，并要考虑到诸如语音浏览器或移动电话之类的特殊设备。
- **假设：每个人都能读懂我的代码。** 错：为了做到不干扰程序员搭档，你的脚本应该由简洁明晰的代码组成，并且添加了大量的注释，来解释某段代码是（打算）做什么用的。

本文将会详细讨论前两个假设。第三类问题将作为单独的一篇文章来进行探讨

除了规避这些假定之外，非干扰性的 **JavaScript** 还要求你将自己的 **JavaScript** 代码从 **HTML** 之中分离出来。由于这一点是这个程序设计哲学中技术性最强的部分，我们就先来讲讲这方面的内容，然后再去讨论那些假设。

### 分离结构与行为

就像我们将结构与外观分离开一样（通过将所有的 **CSS** 放在一个单独的文件中，并避免 **style** 属性或其它类似的表示性标记的使用），我们也应该将 **HTML** 结构和 **JavaScript** 行为分离开来。这么做的理由是一样的：它可以将你所关注的焦点独立出来，保持你的代码整洁，而且可以让你致力于 **JavaScript** 上的工作，而又不必牵扯到 **HTML** 或 **CSS**。

基本规则很简单：在 **HTML** 文件中不要包含任何 **JavaScript**，就像不要包含任何 **CSS** 一样。**JavaScript** 开发者常放在其 **HTML** 文件中的两样东西是：嵌入式脚本和事件处理器。实现结构与行为分离的最简单的方法就是将这两样东西移走。

就如你在前一篇文章中所学到的那样，嵌入式脚本就在一个 **HTML** 页面内的 `<script>` 标签之间的 **JavaScript** 代码片段。这些片段可以很方便地移到一个外部 **JavaScript** 文件中，所以移除嵌入式脚本是不成问题的。

内联事件处理器，比如 `<a href="somewhere.html" onmouseover="hideAll()">`，其移除难度

稍高一些。这个语句所定义的是当一个特定事件发生的时候，应该运行哪个事件处理器（`JavaScript` 函数）。为了使这段代码符合非干扰性，我们需要将该事件处理器的指派放到一个单独的脚本文件中去。这就意味着外部脚本需要先找到正确的元素，然后将一个事件处理器指派给它。

欲了解关于事件处理器的更多信息，请参阅 [处理 `JavaScript` 事件](#)。

使某个元素易于查找的最简单的方法就是给它赋一个 ID。例如：

```
<!-- HTML : -->

<!-- JavaScript : -->

var x = document.getElementById('somewhereLink');

if (x) {

 x.onmouseover = hideAll;

}
```

只要用户的鼠标经过该链接，`hideAll()` 函数就会被执行，并且这段代码是等价于 `<a href="somewhere. HTML " onmouseover="hideAll()">` 的。

更妙的是，这两行代码在每个包含 `id="somewhereLink"` 的元素的页面中都有效，这样你就不必一再地重复这段代码了。而且即使页面中不包含这样的元素，`if (x)` 检查语句也能确保不会产生错误的信息：只有在 `x` 元素真正存在的情况下，该语句才会给它指派一个事件处理器。

那么，要是用户的浏览器不支持 `JavaScript` 又会怎样呢？显然，在这种情况下鼠标经过效果就不会起作用了。但链接还是链接，仍可以跳转到该链接。

现在，我们的代码除了变得更整洁之外，也变得更易于维护了。

举例来说，在大多数情况下配合使用 `mouseover` 事件与 `focus` 事件是个不错的办法。`mouseover` 只在用户使用鼠标的情况下才能起作用。不用鼠标的人会用键盘来聚焦在他们想要点击的链接上，这样就会触发 `focus` 事件。如果我们为该事件注册了事件处理函数（也就是说，如果我们告知该函数，它应被这个事件所触发）的话，我们就确保了自己的应用程序也可以对键盘做出响应。

如果妥善的分离了脚本与 **HTML**，上述要求是很容易做到的：

```
var x = document.getElementById('somewhereLink');

if (x) {

 x.onmouseover = x.onfocus = hideAll;

}
```

只需要添加 12 个字符，我们的应用程序的这一部分可以被键盘所访问了。很简单，对吧？

现在我们来思考一下，如果我们使用了内联事件处理器的话，又应该怎么做。我们需要手动地遍历整个网站上所有的链接，并且将 `onfocus="hideAll()` 添加到带有 `onmouseover` 事件处理器所有链接中。这样一来，我们不仅会浪费掉许多本可以用在更有意义的事情上的时间，而且这样做还是一种易于犯错误的工作方法，因为很容易就会漏掉位于某个不显眼的模板中的某个链接，而这种模板仅用于非常特殊的情况；或者是直接输错某条 `onfocus` 语句。

因此，正如将 **HTML** 和 **CSS** 分离开来是一个明智之举，将 **HTML** 和 **JavaScript** 分离开也同样是一个好办法。移走内嵌脚本和内联事件处理器是很简单的事，而且这样做会使你的代码质量和可维护性得到立竿见影的改善。

### 添加一个可用性层

既然我们已经解决了非干扰性 **JavaScript** 中技术性最强的部分，现在就该回到前面我们讨论过的假设上了。最重要的事情是，绝不要假设每个人的浏览器都支持 **JavaScript**，而且这一点直接影响了在网站上添加脚本的目地。

**JavaScript** 的目的是在你的网站中添加一个可用性层。注意“添加”这个词：如果脚本就是整个可用性层（换句话说，如果这个网站离开了 **JavaScript** 就无法使用）的话，你就犯了一个严重的错误，而且你的脚本也不是非干扰性的了。

### 示例——表单校验

实际的例子可以更加清楚地说明这一点，因此我们来讨论一下表单校验。在添加到数据库中之前，应当总是在服务器上对表单中的用户输入进行检查，因为毫无保留地相信用户输入（可能是恶意的用户所输入的），这会导致你的网站更快，也更确定无疑地被黑掉——我们必须在接受数据之前对其进行检查。

然而，服务器端表单校验也有其缺点，那就是耗时要多一点。用户将表单提交给服务器，而服务器则生成一个响应页面，以通知用户提交成功或出错。无论服务器的反应多么快速，在用户提交和信息反馈之间都会有一小段延时——我们假设是半秒钟。

更糟的是，如果用户犯了错误——比如说没有填写她的地址——她就不得不更正这个问题，然后再次提交表单，这就会导致又一个半秒钟的等待。虽然实际上时间并不长，但却会在用户的感觉中形成累积效应，尤其是在她不得不来回折腾好几次的情况下。

这就是 **JavaScript** 大显身手的时候了。如果你添加了一个脚本，可以在表单发送到服务器之前对其进行检查的话，你就可以捕获典型的用户错误，同时还可以避免服务器和用户之间数据传输；你的用户界面的运行看起来也会显得更快，更平稳。

因此，使用 **JavaScript** 表单验证脚本是一个不错的主意。然而需要牢记的是——你只能将 **JavaScript** 作为服务器端表单验证的补充。前者能为你提供一个更流畅的界面，但只有后者才能为你提供适当的安全性（如果你打算调皮捣蛋的话，要绕开 **JavaScript** 是很容易的——你只需关掉 **JavaScript** 即可），后者还可以在用户浏览器不支持 **JavaScript** 的情况下发挥作用。

在 2009 年早期时候（在不久的将来这可能会改变）**Opera** 浏览器相对其他浏览器有一个优势，就是支持 **HTML 5 Web Forms 2.0** 规范，这是更高级的处理 **HTML** 表单的方法，不需要 **JavaScript** 就可以实现客户端表单验证，因此验证过程无法被用户关闭。要更多了解 **Web Forms 2.0**，请参阅 [使用 \*\*HTML 5\*\* 改善 \*\*HTML\*\* 表单](#)。

## 原则

这个例子强调了一些重要原则：

1. 你的网站应当在没有 **JavaScript** 的情况下依然能工作。
2. 如果 **JavaScript** 是启用的，你就可以为你的用户提供一个额外的可用性层；该可用性层可以让他们更快捷地完成自己的任务，而且可以最大限度地避免令人不快的页面重新载入。
3. **JavaScript** 的安全性不佳。也就是说，对于检验表单中的用户输入之类的重要任务，你\*绝不\*应该信任纯 **JavaScript** 的程序。毕竟，一个蓄意的用户可以直接关掉 **JavaScript** 来绕过你的防御。

那么，精确地来说，“你的网站应该在没有 **JavaScript** 的情况下依然能够工作”的意思是什么呢？它的意思是任何用户，不管他用的是什么浏览设备和软件，都应该能读取你网站的内容，并能使用网站导航功能及其它关键性的功能。其意义就是这么简单，不比这个多，但也（更重要的是）不比这个少。

还有一个重要的结论是，不须为那些不适用脚本的用户提供与启用脚本的用户**相同**的功能。在表单验证的例子中，在得到自己的脚本提交结果之前，不使用脚本的用户只能等上半秒钟，因此不使用脚本的用户比启用了脚本的浏览器用户的用户体验要稍微差一点。

实际上，这是一个基本规则。如果 **JavaScript** 被禁用，可用性就会打折扣，而作为一个

**web** 开发者，你的工作就变成了确保人们可以使用你网站的基本功能：内容和导航。其它的网站功能就成了可选项。

### 示例——弹出式窗口

我们来看看另一个例子——弹出式窗口。有的时候弹出式窗口真的很有用，而且创建弹出式窗口也非常简单——如果浏览器支持 **JavaScript** 的话。然而，如果浏览器不支持 **JavaScript**，那么使用该浏览器的用户就看不到弹出式窗口了。你该怎样在保证自己的网站可访问的同时，为启用了 **JavaScript** 的浏览器提供一个更流畅的界面呢？

用 **HTML** 来写：

```
Go somewhere
```

用 **JavaScript** 来写：

```
var x = document.getElementsByClassName('popup'); // 一个自定义函数

for (var i=0;i<x.length;i+=1) {

 x[i].onclick = function () {

 window.open(this.href,'popup','arguments');

 return false;

 }

}
```

此处最重要的部分就是链接的 **href** 属性。这个属性定义了应该在弹出菜单中显示的页面，除此之外它还确保了在 **JavaScript** 被禁用的情况下，用户也能跟踪该链接。因此本示例在没有 **JavaScript** 的情况下仍然是完全可访问的——只不过没那么美观罢了。

如果 **JavaScript** 是启用了的，你就可以添加一个额外的可用性层：弹出式窗口。你可以找出所有带有 **class="popup"** 的链接，并添加一个 **onclick** 事件处理器，用来打开某个弹出式窗口。弹出式窗口该显示那个页面呢？就是在该链接的 **href** 属性中定义的那个页面(**this.href**)。

不同的例子，同样的原理。首先是确保任何用户都能访问该信息；然后在此之上添加一点 **JavaScript** 程序片段，来让该界面工作得更流畅。

### 示例——Ajax

这个理论有时会难以应用到实际中，尤其是如果你对 Web 开发不怎么熟而又想要创建，比方说，一个 Ajax 网站的话。弄懂非干扰性 JavaScript 有一个很有用的诀窍，那就是层的思想。这个网站的基本功能是什么？基本功能应当被放在最底层，该层有没有 JavaScript 都能访问。接下来，你可以在最底层之上应用任意数量的 JavaScript 驱动的可用性层，从而使该网站更易于使用。这些层不会干扰基本的网站功能，它们只是用来提供一些附加功能。

我们举个例子来阐明该基本规则。假设你的网站是关于移动电话比较的。那么该网站的基本功能差不多就是下面几样：

1. 通过型号或诸如“支持无线上网”或“可与我的桌面电脑同步”这样的笼统的标准，来搜索某种移动电话。
2. 得出一份与你的搜索条件相匹配的移动电话列表。
3. 将该列表按价格，名称/型号，或相关性排序。

所有这些功能都可以不用任何 JavaScript 创建，并且你的首要任务就是实现这些功能。

因此，你要创建：

- 一个搜索页面。
- 一个列表页面，由服务器生成，该页面是基于你的搜索结果的。
- 在该列表页面中添加链接，这些链接的作用是从服务器获取排序方法不同的列表。

当你做好了这些无脚本的页面，你就创建了一个几乎能够在任何设备上的任何浏览器中工作的基础层。

做好这些之后，你就可以在这些仅有基本功能的页面上添加 JavaScript 功能了。最显而易见的改善是排序页面。毕竟，排序所必需的数据——比如价格和特性——已经在该页面上的表格中了，因此你不必再往服务器跑一趟来获取这些数据了。你只需要写一个脚本来从该表格读取这些数据，然后将该表格的 <tr> 排序就可以了。

写完自己的脚本之后，你就应该确保用户能使用它。为达到这个目的，最显而易见的解决方案就是为该用户提供一个链接。

此处需要注意的一点就是表中已经有链接了：这些链接负责从服务器获取一份按照不同顺序来排序的列表。因此，目前为止最好的解决方案就是改写这些链接的默认行为，就像我们在弹出式菜单一例中所做的一样。

因此你可以编写一个附加功能来为这些链接指定 onclick 事件处理器，以确保在这些连接

上点击鼠标时调用的是你的排序脚本，而不是服务器上的页面。

这样，所有的用户就都会看到相同的排序链接，但这些链接的真正行为取决于浏览器是否能处理附加的 **JavaScript** 的可用性层。如果能处理的话，该浏览器就会启动你的脚本，如果不能处理的话，则会从服务器获取一个新的页面。

改写 **HTML** 元素的默认行为是非干扰性的 **JavaScript** 的一个典型范例。如果用户的浏览器支持你的脚本，那当然好，你的脚本就会得到执行。如果不支持的话，就会回落到该 **HTML** 元素的默认行为。只要你遵循这个原则，你的脚本就符合非干扰性。

### 规则的，语义性的 **HTML**

脚本是在 **web** 页面的环境中运行的。那么是在哪种 **Web** 页面中运行的呢？从理论上讲任何页面都可以，只要页面中具备一些可以由你操作的 **HTML** 元素就行。不过实际上你很快就会发现，那些遵从标准的，使用了语义化而且结构化的 **HTML** 的，并且适当地分离了自身的 **HTML** 和 **CSS** 的 **Web** 页面要比那些所见即所得网页编辑器生成的，基于表格的页面容易处理得多。

假设你正在致力于一个新闻网站的开发，而你希望每个页面都包含一个可点击的内容表格，该表格中的内容是到新闻报道中的所有新闻标题的链接。要实现这个效果，最简单的办法是扫描出整个页面中的标题标签（`<h1>`, `<h2>`, 等等），创建一个列表，并确保在该列表上的点击可以将用户导向至正确的标题。脚本写好了，我们来看下一个工作。（顺便提一句，这个脚本也是非干扰性的。跳转至标题的功能是一种可有可无的附加功能，但如果失去该功能的话，用户也能通过导航来阅读该页面。）

现在假设不幸的是，你要开发的网站不是用你以前学过的语义性的 **HTML** 和 **CSS** 来创建的。假设它看起来像这样：

```
<div class="contentcontainer">

<p class="maincontent"><b class="headline">McCain suspends campaign to spend more time with his
economy

In a move that shocked political observers ... [etc etc]</p>

</div>
```

你还能为这样一个标记混乱的页面编写一个生成内容表格的脚本吗？当然能。只要依次遍历所有的 `<b>` 标签，并指明每个带有 `class="headline"` 的标签其实都是一个新闻标题——然后你的脚本就能起效了。

虽然上述方法也能生成内容表格，但实际上你还是丢失了一些信息，那就是新闻标题的结

构化嵌套层级。一个规则的，语义性的页面能直接告诉人们 `<h3>` 标题是前面的 `<h2>` 标题的副标题，而你的脚本大可以对这个特性加以利用。

在一个像上面那样的非语义性的页面中，你需要找出其它办法来确认某个 `<b class="headline">` 是否是主标题，副标题，副副标题，或其是它什么东西。

现在，即使是上面说的那样的问题也有办法解决了，但这不是重点。重点是，如果该 HTML 的作者用了适当的标题标签来标记这些标题，你的脚本就一定可以找出与之相符合的信息，而不管这个脚本在何时运行，也不管是在何处运行的。相反，如果 HTML 的作者用的是 `<b class="headline">` 这样的标记，这就说明他并不真正知道自己在做什么，因为稍后该 HTML 的版本可能轻易地转变为 `<strong>` 或 `<span>` 之类的标签。每次 HTML 发生了改变，你都得修改自己的脚本。

因此，用来处理规则的，语义性的页面的脚本通常比较容易编写，而且通常比那些用来处理标签大杂烩的脚本的维护要容易得多。

## 浏览器兼容性

非干扰性的 JavaScript 也要求脚本不干扰浏览器。实际上，这意味着该脚本应当能够运行在尽可能多的浏览器上，而且如果不能运行的话，该脚本也不得导致脚本错误；该脚本应该通过柔性的功能衰减给网站用户留下一个仍然能提供有效功能的页面。这听起来很不错，但这即使对 JavaScript 大师们来说也是 JavaScript 开发的最难的问题之一。

JavaScript 高手 Douglas Crockford 把浏览器叫做“世界上最不友好的程序设计环境。”这不仅是因为浏览器并不是一个真正的应用程序平台（目前还不是），在提供调试器之类的工具方面有不良记录（尽管这方面也在渐渐改善），而且两者之间的巨大差异很快就会让 JavaScript 新手抓狂。

不幸的是，浏览器的不兼容性是无可争辩的事实。一旦你开始编写任何复杂的 JavaScript，就会发现各种浏览器的行为之间差异非常大。最常见的情况是，你会发现你的脚本在除了 IE 的所有浏览器上都能用，尽管在其它的浏览器中也有各种小问题。

造成这些不兼容性的根本原因在于大多数的主流浏览器都有自己的 JavaScript 引擎，以解析、解释和执行你的脚本，而这些引擎是各不相同的。

不兼容的问题常常是来自于 IE；有不少功能在 IE 中跟在别的浏览器上的运行结果是不一样的，或者 IE 干脆就完全不支持。微软 IE 团队致力于使自己的浏览器更加地符合标准，但这项工作还没有完成。所以你还是会遇到一些麻烦。

即便是这个问题解决了，也还是会有大量的不兼容问题存在，其原因可能仅仅是因为某一

JavaScript 引擎还不支持某种方法，或者是因为 JavaScript 引擎开发者无意中犯的一个错误。

因此兼容性问题是存在的，而且这个问题短时间内是不会消失的。那么我们怎么应对？

最佳策略就是接受它并集中精力对付关键的技术细节，本系列教程随后的文章会详细探讨这些细节。不幸的是要想记住所有的浏览器兼容性问题是不可能的，但对那些刚起步的 JavaScript 的程序员来说，还是有几点帮助建议的。在 [QuirksMode.org](#) 上面有兼容性列表，该表格清楚地说明了哪种浏览器支持哪些方法和属性，并且还提供了大量的bug提示。

除此之外，许多人以前已经走过这条路了。其中一些人决定创建 JavaScript 程序库，以绕过这些兼容性问题。因此，利用 JavaScript 程序库来解决浏览器不兼容的问题是一个切实可行的办法。

虽然如此，还是要多加小心。作为一个 JavaScript 新手，你应该慎重考虑一下，如何在不利用程序库的情况下编写你的第一个大型项目。这么做的理由是一个成熟的 JavaScript 程序员必须要努力钻研，才能深刻了解在底层到底发生了什么。这样做你就能清楚地知道浏览器会怎样处理你的脚本，这些知识对 JavaScript 程序员的开发工作会很有帮助。

你当然可以用程序库来解决问题，但要抵制碰到问题立刻就使用程序库的诱惑。为了深入理解自己所面对的问题，你应该试着不用程序库来解决问题。

## 总结

非干扰性的 JavaScript 不仅仅是一种技术，更是一种程序设计思想。其最核心是对于哪个功能属于哪一层有个清楚的认识。所有至关重要的网站功能都应当完全用 HTML 来编写，当你成功创建此基础后，就可以在其上添加一个 JavaScript 层，来为支持 JavaScript 的浏览器提供一个看起来更美观，更清爽，也更快捷的界面。

另外，非干扰性的 JavaScript

1. 将结构和行为分离开了，这样就使得你的代码更简洁，脚本也更容易维护
2. 绕开浏览器不兼容问题
3. 处理的对象是规则的，语义性的 HTML 层

非干扰性的 JavaScript 并不是那么难的。它主要就是一种思维方式；一旦你花几个小时来好好规划一个非干扰性的 JavaScript，你就会发现做出正确的决策变得越来越容易了。

当你习惯了非干扰性的 JavaScript 之后，你就不会愿意再回到以前的那种干扰性的模式了。

- 上一篇文章— [JavaScript 最佳实践](#)

- 下一篇文章—JavaScript 函数
- 教程目录

## 作者简介



Peter-Paul Koch 是荷兰阿姆斯特丹的一名自由职业的 Web 开发者。 quirksmode.org 就是由他编写和维护的，该网站是目前互联网上最好的 JavaScript 资源之一。尤其是他的 W3C DOM 兼容性表格（W3C DOM Compatibility Tables），为全世界的 Web 开发者省下了约有五百万英镑的头发移植和护发产品的费用。诸如微软，Opera 和 Apple 等浏览器发行商也在用这些表格来解决自己产品中的问题。

## 44. JavaScript 函数

Posted 03/10/2009 - 23:10 by yuanc

- 网络标准教程

## 44: JavaScript 函数

作者 Mike West · 2009 年 2 月 3 日

- 上一篇 — 非干扰性 JavaScript 原则
- 下一篇 — JavaScript 中的对象
- 目录

### 引言

函数是几乎所有你用 JavaScript 编写的有用的功能的核心。一般而言，函数可以将一个程序分为若干逻辑块，每个块实现某个特定的功能。函数是 JavaScript 程序设计语言最主要特征，而 JavaScript 之所以具有这么大的吸引力原因之一就是它特有的使用和创建函数的方式。如果你之前曾经用 PHP 或 Java 之类语言编写过某些程序，那么你就会对 JavaScript 中的函数感觉心应手；如果你没有这样的经验，也不用担心。函数很重要，但它们也并不会难到把你搞糊涂。本文会阐述为什么应该了解函数，然后我们将介绍函数的语法，并说明如何创建和使用函数。

注意，点击此处可以下载本文函数示例，在本文接下来的部分中的适当地方也有链到这些示例的链接。

本文结构如下：

- 什么是函数以及为什么要使用函数
- 函数的语法

- 使用函数
- 参数
- 返回值
- 总结
- 练习题

## 什么是函数以及为什么要使用函数

你肯定不希望每次要执行一项特定运算的时候都得去拿你的规范说明来唤起自己的记忆；最好的做法是直接将该运算的步骤**一次性**编写好，然后将这些步骤打包成一个 `calculateSomething` 函数，下次要执行同样的操作时只需要指向该函数。这种打包一系列命令的简单做法意味着你可以专注于自己的代码所实现的**操作**，而不是这些操作内部的具体细节。你可以将自己编写的函数看作是位于 `JavaScript` 内置核心之上的一个层；你是在自己的特定应用程序的环境中创建表达性和可理解性更强的**新指令**。

有了这个理念，“为什么？”要使用函数的问题就有了一个非常简单的答案：函数是基本的组成模块，通过函数你就可以结构化自己的代码的，从而提高对其作用的理解，并可以重复使用你所编写的函数，以避免在多处地方重复编写同样的代码片段。如果你将自己的程序分为若干小片段，每个小片段完成一项确定的任务的话，该程序就更容易编写了。

此外，在深思熟虑之后，将你的代码分解为若干函数，会使得将来对代码的维护容易得多。比如说，我们假设明年的夏令时间会发生改变。如果你的整个项目中共有八十五次夏令时计算，你对代码的每一次更新的都可能会产生新的漏洞；而且这样做是重复的，手动的，而且很容易导致 `bug`。另一方面，如果是对单独的 `calculateDaylightSavings` 函数进行改动的话，你只需通过一次修改就可以将这种改动级联到程序的其它部分，这与 `CSS` 中样式在整个页面上的级联非常相似。这样，函数使得程序维护更不易出错，而且实现起来也更容易成功。

## 函数的语法

定义你自己的函数是非常简单的。作为范例，我们在一个页面上创建一个能 随机改变某个元素的背景颜色的函数：

```
function setElementBackground() {

 var red = Math.floor(Math.random() * 256);

 var green = Math.floor(Math.random() * 256);

 var blue = Math.floor(Math.random() * 256);
}
```

```
var obj = document.getElementById('element_to_change');

if (obj) {

 obj.style.background = 'rgb(' + red + ', ' + green + ', ' + blue + ')';

}

}
```

不必太担心这个函数内的代码，此时此刻我希望你关注的事这个函数的语法，它有四个重要特征：

1. 函数的声明常常由关键字 `function` 开头，这样就能直观地说明这是一个函数。
2. 第二点是函数名，在本例中是 `setElementBackground`（我一般用 驼峰式大小写来做函数名）。函数名是很重要的，这是因为为了使用和重用这段代码，你必须记住函数名。我们应该确保函数名精确地描述了该函数的作用；我想你一定会同意比起 `coloursAreNice` 或 `crazySetter` 来，`setElementBackground` 作为函数名要好得多，描述性也要更好一些。
3. 紧接在函数名之后的是一对圆括号。在圆括号之中是函数的参数列表，通过该列表你就可以提高自己的函数的通用性，从而提高其可复用性——你可以更容易地将它们用到更多的情况之中。参数一个很有用的概念，但是可选的，因此我将在下一部分来详细讨论这个问题。
4. 最后是一对大括号，其中包含着一些代码：大括号代表着一个 **JavaScript 代码块**。这个代码块之中的一切都会在函数被调用时按次序执行，就像你以前写过的其它 `JavaScript` 代码片段一样。

## 函数的使用

我们的函数定义完成后，在代码中其它地方调用这个函数只需要写：

```
setElementBackground();
```

这样就可以了！你不再需要关心 `setElementBackground` 的复杂的内部细节；你已经写好了它的代码，所以现在你就可以轻松地在自己想要的地方调用这个函数，并享受代码重复使用的好处。

可以看到，我们刚才写的那个函数是完全独立的。这个函数执行一些操作，然后就退出了；

它既不需要从调用自己的代码那里获得输入，也不会向调用自己的程序返回任何信息，告诉它发生了什么。当然，**JavaScript** 允许我们编写比这个函数更灵活的更富于信息交换的代码，下面我们就来看看该如何向函数输入的信息和从函数输出信息。

## 参数

向函数传递信息以影响其行为，在很多情况下这都能使函数更灵活，更有用。比如说，被 `setElementBackground` 函数改变背景颜色的元素的 `id` 是通过硬编码的方式确定的；如果我无论何时调用该函数，都能指定改变页面上某个元素的背景，那应该会很不错，因为这样我就可以将这个函数重复用于不同的元素，而不是把整段代码都复制下来了。通过**参数**我们就可以做到这一点。

在前面我们曾提到过，函数定义紧接在函数名之后的是一对圆括号。这就是该函数的**参数列表**。为了接受来自调用程序的输入信息，你只需指定一个用逗号分隔的参数列表，这些参数就是你的程序要接收的信息的对应变量。你可以指定任意多数量的变量，并且参数列表中所提到的所有变量名都可以在函数体之内被引用，就跟任何其它变量一样。修改后的 `setElementBackground` 函数如下所示（[点击查看其效果](#)）：

```
function setElementBackground(elementID) {

 var red = Math.floor(Math.random() * 256);

 var green = Math.floor(Math.random() * 256);

 var blue = Math.floor(Math.random() * 256);

 var obj = document.getElementById(elementID);

 if (obj) {

 obj.style.background = 'rgb(' + red + ', ' + green + ', ' + blue + ')';

 }
}
```

通过参数传入元素 `ID` 来调用这个函数是很简单的：

```
setElementBackground('element_to_change');
```

如果你调用了该函数，却忘记传给它一个参数值，那么该参数就会取 `undefined` 值。你可

以在自己的函数体内添加参数检验代码以防止无心之错：

```
if (elementID == undefined) {

 // 如果调用程序没有提供`elementID`变量，

 // 这个表达式的值就为‘true’

 // 你就可以在这个if语句中编写一些代码

 // 来阻止程序出错。

}
```

在函数参数方面，有一个容易搞糊涂但又有好处的特点，就是参数列表中的变量名与传入函数的变量名无关。如果 `elementID` 被定义为函数的自变量，**JavaScript** 就会在函数内部创建一个名为 `elementID` 的变量，且这个变量不会影响任何函数外部的变量——在这个函数的外部你还可以再写一个函数，用同样的变量名，但这个变量的值不会随着第一个函数内部的任何语句而改变。比如：

```
var elementID = "No change!";

setElementBackground('element_to_change');

alert(elementID); // Alerts "No change!"
```

这种做法有个非常严重的副作用。**JavaScript** 在该函数内部创建了一个新变量，这就意味着它对自己的内部变量所做的任何改变都不会影响到传入的变量。我将在[对象和 JavaScript 最佳实践中](#)详细讨论此概念（称做作用域）。现在我们来看一个简单的例子。我定义了一个 `substring` 函数，该函数会接收一个字符串和一个起始点：

```
function substring(obj, start) {

 obj = obj.substring(8);

}

var myString = "This is a string!";

substring(myString, 8);
```

```
alert(myString); // Alerts "This is a string!"
```

即使我们在函数内部对 `obj` 变量进行了重新赋值，将其值设为内嵌的 `substring` 方法的结果，`myString` 也不会受到一点影响；只有位于 `substring` 内部的 `myString` 的副本值被改变。外部变量完全不知道函数内部发生了什么。

这就造成了函数内外的通信问题：如果改变参数的值不会对函数外部造成任何影响，你该如何将来自函数的信息传回其调用程序呢？下面我们就来谈谈这个问题。

### 返回值

用函数来进行运算，并将运算结果传回位于其它地方的调用程序，这是很常见的做法。举例来说，让我们的 `setElementBackground` 函数返回一个颜色值的数组，以便在其它地方使用，这种功能可能会很有用处。只需要使用 `JavaScript` 提供的 `return` 关键字就可以轻易地做到这一点，如下所示：

```
function setElementBackground(elementID) {

 var red = Math.floor(Math.random() * 256);

 var green = Math.floor(Math.random() * 256);

 var blue = Math.floor(Math.random() * 256);

 var obj = document.getElementById(elementID);

 if (obj) {

 obj.style.background = 'rgb(' + red + ', ' + green + ', ' + blue + ')';
 }

 return [red, green, blue];
}
```

[点击此处查看效果。](#)

此处添加的这条简单代码意味着，你现在可以调用此函数，并通过变量保存该函数的返回

结果：

```
var my_result = setElementBackground('element_to_change');
```

即使你的函数不需要返回任何值，或者是没有可以返回的实际值，分别返回 true 或 false 以提示程序运行成功或失败也是很好的习惯。下面，我们来对 setElementBackground 函数进行修改，如果应该传入的 elementID 不存在的话，该函数就会返回 false：

```
function setElementBackground(elementID) {

 var red = Math.floor(Math.random() * 256);

 var green = Math.floor(Math.random() * 256);

 var blue = Math.floor(Math.random() * 256);

 var obj = document.getElementById(elementID);

 if (obj) {

 obj.style.background = 'rgb(' + red + ', ' + green + ', ' + blue + ')';

 return [red, green, blue];

 } else {

 return false;

 }
}
```

[点击此处查看效果。](#)

你可以通过检测其返回值来查看该函数的执行是否正确，如：

```
if (!setElementBackground('element_does_not_exist')) {

 alert("Something went wrong! `element_does_not_exist` doesn't exist!");

}
```

此外，请注意一下在调用 `return` 关键字时，该关键字实际上是结束了你的函数的执行，并将执行权利返回到调用函数的地方。在 `return` 后面的代码不会得到执行——它会被直接忽略掉。

## 总结

读完本文，现在你差不多已经学到了所需要知道的一切，这样你就可以在自己的程序中开始大肆使用函数了。这些知识是编写高质量 `JavaScript` 代码的基础知识，如果你经常将自己的代码打包进命名合理的函数，以利于重复使用的话，你的程序就会变得更有条理，更清晰，更易读，也易于理解了。

## 练习题

- 什么是函数？为什么说函数很有用？
- 如何定义函数？
- 怎样向函数传递信息？为什么要这么做？相反地，怎样从函数获取信息？
- 如果我们将一个颜色数组传给``setElementBackground``岂不更好？试试看修改一下代码以接收另一个参数，并在函数内部的使用该变量来替代随机背景颜色。
  
- 上一篇 — 非干扰性 `JavaScript` 原则
- 下一篇— `JavaScript` 中的对象
- 目录

## 作者简介



Mike West 本来是一个哲学系学生，但他却成了一个经验丰富且成功的 Web 开发者。他已经跟 Web 打了十多年的交道了，最近他在 Yahoo! 的欧洲新闻网站团队中工作。

在 2005 年离开了得克萨斯州郊外的广袤平原之后，Mike 定居在德国的慕尼黑，他现在对当地语言的掌握越来越好了。[mikewest.org](http://mikewest.org) 是他在网上的家，(缓慢地) 收集着他的著作和链接。他将自己的代码放在 [GitHub](#) 上

## 45. JavaScript 中的对象

Posted 03/10/2009 - 23:10 by yuanc

- [网络标准教程](#)

## 45: JavaScript 中的对象

作者 [Mike West](#) · 2009 年 2 月 3 日

- [上一篇—JavaScript 函数](#)
- [下一篇—DOM 之旅](#)
- [目录](#)

### 引言

前面的 [函数](#)一文引入了函数的概念，向我们讲授了如何将单个的程序任务划分成逻辑块，这样就可以在任何地方调用这些逻辑块，从而更好地组织和重复使用我们的代码。既然我们已经熟悉了这些 [JavaScript](#) 程序设计的基础知识，下面我们就更进一步，本文中我们会引入[对象](#)的概念。通过对象，你可以将自己定义为函数的相关功能集聚拢在一起，并将它们打成一个包，你可以把这个包像一个整体一样传送和引用。这种能力对于你所编写的代码来说有着非常实际的意义，尽管现在听起来有点抽象。

你可能还没有注意到，但在本系列的整个教程中你已经不知不觉地接触过对象了；在这里，我会更明白地告诉你在 [JavaScript](#) 中对象是如何工作的，还会教你如何利用对象来提高你的代码的可读性和可复用性。

本文结构如下：

- [为什么要使用对象？](#)
- [似曾相识之处](#)

- 创建对象
- 自我引用
- 作为关联数组的对象
- 对象常量
- 总结——还有很多东西要学
- 延伸阅读
- 练习题

注：你可以在此处下载一个示例，也可以实际运行它，该示例包括了用于计算一个三角形的面积的代码，既有使用了对象的版本，也有不使用对象的版本。在下面的文章中我们会逐步创建该程序。[点击此处运行该三角形对象示例。](#)

### 为什么要用对象？

关注对象的一个最重要的原因就是，它们能够改善你的代码对你所要实现的数据和处理过程的表达。作为一个简单的例子，我们来设想一下你该怎样编写代码以对一个三角形进行一些处理。我们知道三角形一般有三条边，因此为了对某个特定的三角形进行处理，显然需要创建三个变量：

```
// 这是一个三角形。

var sideA = 3;

var sideB = 4;

var sideC = 5;
```

现在，我们就有了一个三角形了！但是这还不够，对不对？我们其实只是创建了三个需要分别记录的变量，以及一个用来提醒自己此处是什么的注释而已。这样的三角形定义还不够清楚，或者说还不太好用。但是没关系，让我们继续，看看我们该怎样围绕该“三角形”来创建一些运算。为了得出该三角形的面积，你可能会写出如下的函数：

```

function getArea(a, b, c) {

 // 通过海伦公式来计算一个三角形的面积，并返回面积的值

 var semiperimeter = (a + b + c) / 2;

 var calculation = semiperimeter * (semiperimeter - a) * (semiperimeter - b) * (semiperimeter - c);

 return Math.sqrt(calculation);

}

alert(getArea(sideA, sideB, sideC));

```

你会发现自己需要将所有关于该三角形的信息传递给函数，从而让函数来进行计算。与三角形相关的操作与该三角形的数据被完全分离了，即使在分离状态下这些操作并没有什么实际意义。

此外，我用了一个很通用的名字来作为函数名，也给每个变量定义了一个通用的变量名：getArea, sideA, 等等。如果下一周我自己需要将这个程序扩展为可以将矩形也包含在内，又会发生什么呢？我想用 sideA 和 sideB 来代表该矩形的数据，但这两个变量名已经被占用了。我可以使用 side1 和 side2，但我敢肯定，你也能看出这么做会导致混淆和彻底失败。可能我最终会使用 rectangleSideA 和 rectangleSideB，而且为了保持一致性，我也只能退回去修改所有已经写好的三角形的代码，改成用 triangleSideA 等等，这会导致某些潜在的出错风险。对于函数名来说也一样：我想用 getArea 来作这两种图形的函数名，因为从概念上说它进行的是同样的计算，但是我无法做到。肯定有一种更好的方式来表达我的数据的！

我们曾经很明智地通过创建函数，将一系列指令打包成一个独立的、命名严谨的操作，同样地，此处可以通过创建对象来将所有的“东西”集合成一个独立的单元。通过对对象，我们就可以

建立属于自己的包含任意数量、任意类型变量的容器，而不用受限于 JavaScript 自带的原始数据类型（字符串型，数字型，布尔型，等等）。这种自由机动性使我们可以创建一些结构，这些结构可以直接映射到编写程序时我们所要关注的“东西”上，在我们的代码中可以直接应用这些结构，就像使用自带的原始数据类型一样。在这里，我要创建的是三角形对象和矩形对象，每个对象分别包含了对该形状进行智能化处理所需的一切数据，以及可能要对该图形执行的所有操作。为了实现这个目标，我们来学习一些语法。

## 似曾相识之处

如果你回顾一下前一篇文章中的最后一个函数示例，你就会发现像这样的代码片段：

```
var obj = document.getElementById(elementID);
```

还有：

```
obj.style.background = 'rgb(' +red+ ', ' +green+ ', ' +blue+')';
```

意外吧！你已经用到过对象了，而你甚至都不知道自己用的是对象！在全面讲述 JavaScript 的对象语法之前，我们先来详细研究一下这两段代码。

这句代码 var obj = document.getElementById( elementID )在某种程度上应该是似曾相识的。我们知道一句指令末尾的圆括号意味着正在执行某种类型的函数，而且可以看到这个函数的调用结果被存储在名叫 obj 的变量中。此处唯一的一点新东西就是中间那个点号。在 JavaScript 中，**点号**就是访问一个对象的内部数据的方式。点号(.) 其实就是位于运算对象之间的一个操作符，就跟+号和-号一样。

根据惯例，可以通过点号操作符访问的存储于某个对象之中的变量一般叫做**属性**。如果属性同时又刚好是函数的话，就叫做**方法**。这两个词语都没什么奇特的；方法只不过就是函数，而属性只不过是变量而已。

点号操作符左侧必须是一个对象，右侧则是一个属性名；上面的代码片段的含义就是访问内置的 document 对象的 getElementById 方法（关于这种用法，将在**遍历 DOM** 一文中详细介绍）。

第二段代码要更有意思一点：它有两个点号。JavaScript 的对象支持真正令人激动的特性之一就是可将点号**连在一起**使用，以访问复杂的结构。简而言之，你可以同样地将对象连在一起，就像你执行 var x = 2 + 3 + 4 + 5;，会得到值为 14 的结果一样。对象的连续引用会直接对自身进行解析，方向为从左到右（如果你跟你的朋友说，这一点使得 JavaScript 的点号操作符成为“左关联中缀操作符”的话，一定会让他们印象深刻）。在本例中，我们对 obj.style 进行了

求值，并将其解析成一个对象，然后访问该对象的 `background` 属性。如果你愿意的话，你可以在自己的代码中添加圆括号来显式地表示这一点：`(obj.style).background`。

## 创建对象

我将用下面的语法来显式地创建我自己的三角形对象：

```
var triangle = new Object();
```

`triangle` 现在是一个空白的地基，等着我们去修建一个有高高耸立的三条边的宏伟建筑。

通过使用点号操作符来为自己的对象添加属性，你就可以实现这个目标：

```
triangle.sideA = 3;

triangle.sideB = 4;

triangle.sideC = 5;
```

要向一个对象添加新的属性，你实际上并不需要做什么特殊的工作。`JavaScript` 在解析点号操作符时是很宽容的。如果你打算设置一个不存在的属性的话，`JavaScript` 会替你创建它。如果你试图读取不存在的某个属性的话，`JavaScript` 会返回“`undefined`”。这样的特性的确很方便，但如果你不仔细的话也会犯错误，因此在打字的时候要小心！

向对象添加方法也是一样的——下面是一个例子：

```
triangle.getArea = function (a, b, c) {

 // 通过海伦公式来计算一个三角形的面积，并返回面积的值

 var semiperimeter = (a + b + c) / 2;

 var calculation = semiperimeter * (semiperimeter - a) *
 (semiperimeter - b) * (semiperimeter - c);

 return Math.sqrt(calculation);
```

```
}; // 注意此处的分号; 这是必需的。
```

如果你觉得这些代码看起来跟函数定义非常相似，你算是说对了：我直接将函数名完全沿用了下来。JavaScript 有一个匿名函数的概念，这种函数自己并没有名字，但是它跟其它的值一样可以被存储在某个变量中。在这段代码中，我创建了一个匿名函数，并将其存储在 triangle 对象的 getArea 属性中。这样，这个对象就可以随身携带该函数了，就像携带其它所有属性一样。

## 自我引用

创建 triangle 对象的目的之一，就是在三角形的数据和在这些数据之上所能执行的操作之间创建一种联系。然而，我还没有完成这一点。你马上就会注意到 triangle.getArea 方法在执行时还需要传入边长数据，这样需要如下所示的代码：

```
triangle.getArea(triangle.sideA, triangle.sideB, triangle.sideC);
```

我觉得这样做比本文开头的那段代码要更好，因为它明确地表达了数据和操作之间的关系。不过，三角形面积和边长的关系意味着我们不必告诉此方法所要处理的值，它应该能够搜集自己所需要的关于该对象的数据并使用该数据，而不需要你手动输入并传给它。

解决此问题的关键就在于 this 关键字，你可以在对象方法的定义中使用此关键字，来引用同一对象中的其它属性或方法。通过用 this 来重写 getArea 方法，我们最终会得到如下的代码：

```
triangle.getArea = function () {
 // 通过海伦公式来计算一个三角形的面积，并返回面积的值

 var semiperimeter = (this.sideA + this.sideB + this.sideC) / 2;

 var calculation = semiperimeter * (semiperimeter - this.sideA) * (semiperimeter - this.sideB) *
 (semiperimeter - this.sideC);

 return Math.sqrt(calculation);
};
```

```
}; // 注意此处的分号; 这是必需的。
```

如你所见, `this` 的作用在某种意义上就像一面镜子一样。当 `getArea` 方法获得执行的时候, 它会读取自己的 `sideA`, `sideB` 和 `sideC` 属性。这样它就能够将这些值运用在对象方法中, 而不用依赖于从外部获取的输入信息了。

注: 这述说明有点过于简化了。`this` 所引用的并非总是定义此方法的对象, 而会根据特定环境而改变。对于此处的含糊我只能说声抱歉, 但这有点超出本文所关注的范围了。在此例中你大可放心, 此处的 `this` 一直是指向 `triangle` 对象的。

### 作为关联数组的对象

点号操作符并不是访问一个对象的属性和方法的唯一办法; 使用下标也可以非常有效地访问对象的属性和方法, 在前面的关于 数组 的讨论中你可能已经熟悉了下标。简而言之, 你可以将一个对象作为一个关联数组来对待, 该关联数组将一个字符串映射到一个值, 就像一般的数组将一个数字映射到一个值一样。通过利用这个标号, 你就可以用另一种方式重写 `triangle` :

```
var triangle = new Object();

triangle['sideA'] = 3;
triangle['sideB'] = 4;
triangle['sideC'] = 5;

triangle['getArea'] = function (a, b, c) {
 // 通过海伦公式来计算一个三角形的面积, 并返回面积的值

 var semiperimeter = (a + b + c) / 2;
 var calculation = semiperimeter * (semiperimeter - a) * (semiperimeter - b) * (semiperimeter - c);
 return Math.sqrt(calculation);
};

// 注意此处的分号; 这是必需的。
```

乍一看, 这样做似乎有点多余。为什么不干脆就用点号呢? 这种新语法的好处在于属性名不是硬编码到你的程序中的。你可以用变量来指定属性名, 这就意味着你可以创建更灵活的指令,

这样的指令可以基于上下文来完成不同的任务。比如说，你可以创建一个函数来比较两个对象，看看它们有没有某个共同的属性：

```
function isPropertyShared(objectA, objectB, propertyName) {
 if (typeof objectA[propertyName] !== undefined
 && typeof objectB[propertyName] !== undefined
) {
 alert("Both objects have a property named " + propertyName + "!");
 }
}
```

这个函数要是通过一般的使用点号的方法的话，是没办法写成的，因为这样我们就得在程序代码中显式地写出需要进行测试的属性的名称。你以后会经常用到这种语法的。

注：关联数组在在 Perl 中叫做“hash”，在 C#中叫“hashtable”，在 C++中叫做“map”，Java 中叫“hashmap”，Python 又叫做“dictionary”，等等。关联数组在程序设计中是一种非常强大而且基础性的概念，你可能早已经知道它了，只不过名称不同而已。

## 对象常量

下面我们要仔细研究一些看起来可能非常的眼熟的代码：

```
alert("Hello world");
```

你可以立刻辨认出 alert 是一个函数，该函数唯一的参数：字符串 “Hello world”。此处应当注意到的是，你不必这样写：

```
var temporaryString = "Hello world";

alert(temporaryString);
```

JavaScript 知道所有包含在一对双引号（" "）之中的内容都应当作为字符串来处理，无论

你在何处写下该字符串，**JavaScript** 都会进行必要的处理从而使该字符串生效。该字符串的创建与传入该函数是同时进行的。在形式上，“Hello world”是作为一个**字符串常量**来引用的；为了创建字符串常量，你需要将字符串常量的内容逐字打出来。

**JavaScript** 也有一个类似的“对象常量”语法，该语法使你能创建自己的对象，而不需要任何额外的语法。我们来重写一下前面那个按照第三种方式创建的对象，这次通过对象常量来创建它：

```
var triangle = {
 sideA: 3,
 sideB: 4,
 sideC: 5,
 getArea: function (a, b, c) {
 // 通过海伦公式来计算一个三角形的面积，并返回面积的值

 var semiperimeter = (a + b + c) / 2;
 var calculation = semiperimeter * (semiperimeter - a) * (semiperimeter - b) * (semiperimeter - c);

 return Math.sqrt(calculation);
 }
};
```

此处的语法很清楚：该对象常量用大括号来表明自己的开头和结尾，括号中包含任意数量的 *propertyName: PropertyValue* 对，彼此之间用逗号隔开。这样就能很方便地在你程序中建立对象，而不需要在每一行定义时重复对象名了。

然而，还有一件事情应该注意：我们最常犯的错误就是，在对象常量的属性列表中的最后一项后面多加上一个逗号（在本例中，就是在 `getArea` 定义后面）。逗号只能放在属性之间——如果在末尾有个多余的逗号的话，就会导致程序出错。尤其是在程序写好后再回去代码中插入或

删除的时候，你得多加小心了，保证每个逗号位置正确。

## 总结——还要很多东西要学

本文仅仅只触及了 JavaScript 对象的表面。读完本文后，你应该能够得心应手地创建自己的对象，添加属性和方法，并以自我引用的方式来使用它们了。本文尚未涉及到的东西还有很多，但未涉及的内容不影响你开始 JavaScript 对象之旅。本文旨在引导你上路，并为你提供所需的工具，以便你可以在今后深入钻研这一方面的课题时读懂所遇到的代码。

## 延伸阅读

- [Object Oriented JavaScript](#)：一个不错的关于 JavaScript 中更高级的面向对象概念的简介。
- [Private Members in JavaScript](#)： Douglas Crockford 的关于在 JavaScript 中实现封装的巨著。
- [Scope in JavaScript](#)：对于 this 关键字在各种环境中的用法的深入讨论。

## 练习题

- 什么时候你应该用下标代替点号来引用某个对象的属性？
- 对象如何引用自身？为什么这一点非常有用？
- 什么是对象常量？在创建对象常量的时候，逗号应该放在哪儿？
- 我创建了一个对象来代表一个三角形，并对其面积进行了计算。我希望你也能照样创建一个矩形对象并计算其面积。在该矩形的 getArea 方法中用 this 来避免不必要的数据传递。
- [上一篇—JavaScript 函数](#)
- [下一篇—DOM 之旅](#)
- [目录](#)

## 作者简介



**Mike West** 本来是一个哲学系学生，但他却成了一个经验丰富且成功的 Web 开发者。他已经跟 Web 打了十多年的交道了，最近他在 **Yahoo!** 的欧洲新闻网站团队中工作。

在 2005 年离开了得克萨斯州郊外的广袤平原之后，Mike 定居在德国的慕尼黑，他现在对当地语言的掌握越来越好了。[mikewest.org](http://mikewest.org) 是他在网上的家，(缓慢地)收集着他的著作和链接。他将自己的代码放在 [GitHub](#) 上。

## 46. DOM 之旅

Posted 03/10/2009 - 23:11 by yuanc

- 网络标准教程

## 46: DOM 之旅

作者 Mike West · 2009 年 2 月 3 日

- 上一篇文章—JavaScript 中的对象
- 下一篇文章—创建和修改 HTML
- 目录

### 引言

在网上很难找得到不与 HTML 有任何形式的交互的实用 JavaScript 代码示例。一般而言，我们的代码需要从页面上读取数据，以某种方式处理这些数据，然后以可见的页面变化或通知信息的形式生成输出结果。为了给页面和应用程序创建反应灵敏的界面，本文和下一篇文章引入了**文档对象模型( Document Object Model)**，该模型提供了对我们所创建的语义性和表象性层进行检查与操作的机制。

学完本文后，你就能对什么是 DOM 有深刻的理解，本文还会教你如何用 DOM 来遍历 HTML 页面，以定位要获取数据或进行改动的位置。本系列教程的下一篇文章（[创建和修改 HTML](#)）着重讲述我们该如何操作页面上的数据，修改数值或创建全新的元素和属性。

本文结构如下：

- 播种
- 生长
- 节点
- 从一个分枝到另一个分枝
- 直接访问
- 总结

- 练习题

### 播种

正如你可能会根据文档对象模型这个名字推断出来的那样，DOM 是一种在加载 Web 页面时由浏览器创建的 HTML 文档模型。JavaScript 可以访问这个模型中的所有信息。现在我们稍稍回顾一下，想想我们到底对什么建模。

当创建一个页面的时候，我的目标是通过将原始内容映射到的 HTML 标签来为其添加意义：这一小段内容是一个段落，因此我用 p 标签来标记它；下一段是一个链接，因此我用 a 标签，等等。我还会对元素之间的关系进行编码：每个 input 域都有一个 label，并且可能全都位于一个 fieldset 内部。此外，我还会做一点额外的事情，在适当的地方添加 id 和 class 属性，来给该页面加入更多的结构，这样我就可以对这些结构进行样式化或其它操作了。HTML 框架建立好之后，我就会用 CSS 来给这些纯语义的结构装扮出时尚的外观。这样，我们就创建了一个能令用户欣喜的页面。

但这些都还不是全部。我所创建的是一个充满了可供我用 JavaScript 操作的元信息的文档。我可以查找到特定元素或元素组，并根据用户定义的变量来对它们进行删除，添加和修改操作；我可以查找到外观信息（CSS）并在运行中对样式进行修改；我还可以校验用户输入到表单中的信息；以及进行各种其它操作。为了用 JavaScript 来完成这些操作，它就必须能获得并操作所需的信息，而 DOM 为 JavaScript 提供了所需的一切。

还有一件值得注意的事：结构良好的 HTML 和 CSS 是一颗种子，从这颗种子就能长出该页面的 JavaScript 模型来。结构不良的文档生成的 DOM 跟你所期待的相差甚远，并且在不同的浏览器上其行为表现也不一致。为了确保最终 JavaScript 所访问的模型跟你所期待的完全一致，你的 HTML 和 CSS 都应该是良构而且能通过检验的，这一点至关重要。

### 生长

在创建好你的网页文档，并对其进行样式化之后，接下来要做的就是将其提交给浏览器以显示给你的用户了。这下就该轮到 DOM 上场了，浏览器会通读你编写的网页文档，并动态地生成 DOM，以供你在脚本程序中使用。具体来讲，DOM 将 HTML 页面表示为一棵树，就像你描绘的“家谱树”一样。页面上的每个元素在 DOM 中都是一个节点，每个节点都带有分枝，连到自己直接包含的元素（其子元素）以及直接包含自己的元素（其父元素）。下面我们通过一个简单的 HTML 文档来对这些关系加以说明：

```
< HTML >
```

```
<head>
```

```
<title>This is a Document!</title>

</head>

<body>

<h1>This is a header!</h1>

<p id="excitingText">

 This is a paragraph! Excitemen!

</p>

<p>

 This is also a paragraph, but it's not nearly as exciting as the last one.

</p>

</body>

</ HTML >
```

如你所见，整个文档都包含在一个 `HTML` 元素中。该元素直接包含另外两个元素：`head` 和 `body`。这两个元素在我们的模型中就是 `HTML` 元素的子元素，而它们又分别指向自己的父元素 `HTML`。依此类推，在整个文档层级中，每个元素都指向自己的直接派生元素，即子元素，以及自己的直接祖先，即父元素：

- `title` 是 `head` 的子元素。
- `body` 有三个子元素 — 两个 `p` 元素和一个 `h1` 元素。
- `id="excitingText"` 的 `p` 元素的子元素是一个 `em` 元素。
- 元素的纯文本内容（比如“**This is a Document!**”）在 `DOM` 中被表示为**文本节点**。文本节点没有子元素，但会指向自己的包含元素，即父元素。

因此，如果将前面的 `HTML` 文档的 `DOM` 层级用可视化的图形描绘出来的话，就如图 1 所示：

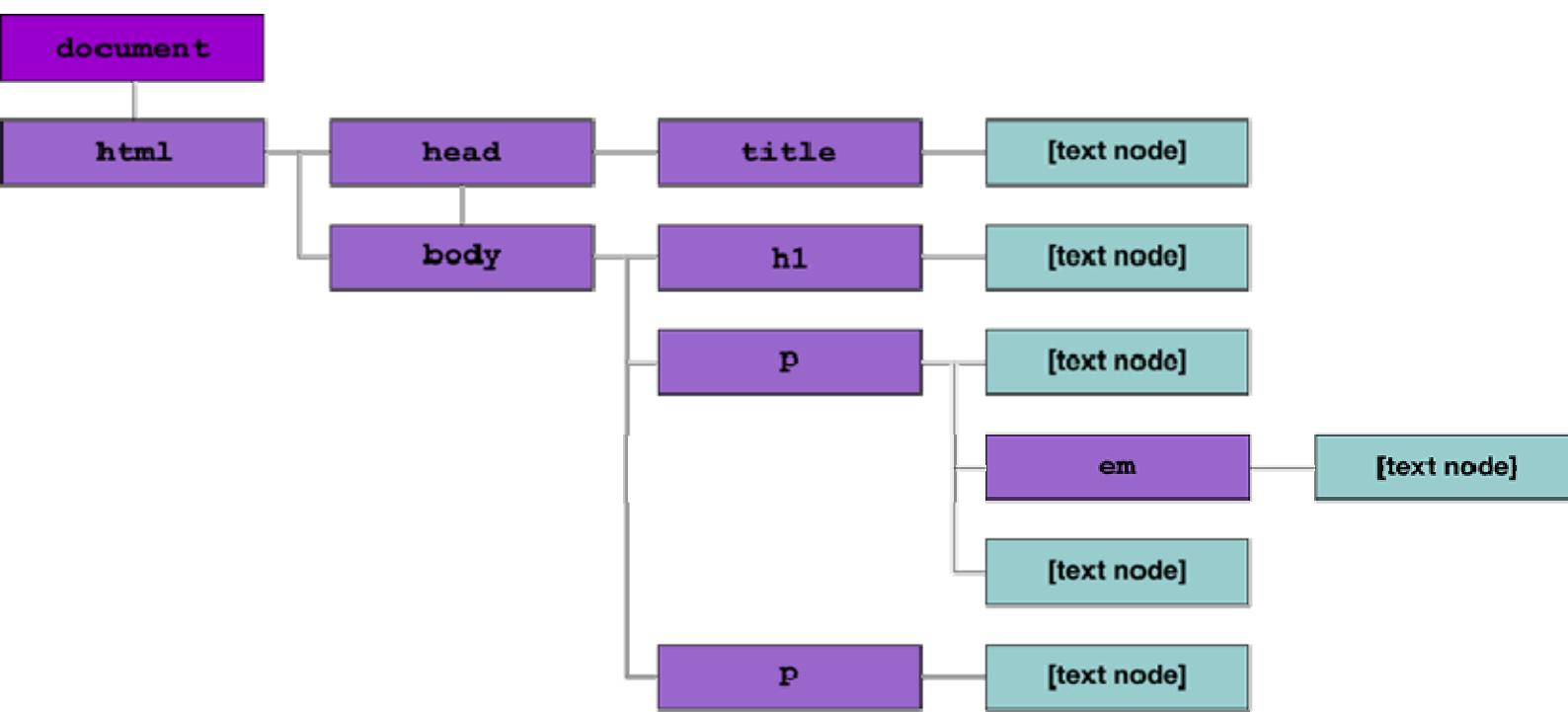


图 1：将前面的 HTML 文档可视化地描述成一个 DOM 树。

从 HTML 文档到这个树状结构的映射是很简单的，该图简洁地抓住了页面上的元素之间的直接关系，清楚地阐述了文档元素的层级。不过，你可能会注意到，我在 HTML 节点之上添加了一个 document 节点。这就是该文档的根节点，也是 JavaScript 操作该树状结构的入口。

## 节点

在我开始在这棵树上晃荡，并从一个分枝荡到另一个分枝之前，我们先来花点时间详细查看一下节点。

DOM 树上的每个节点都是一个 [对象](#)，代表了该页面上的某个元素。每个节点都知道自己与其它那些跟自己直接相邻的节点之间的关系，而且还包含着关于自身的大量信息。就像一个孩子在后院的橡树上从某根枝条爬到最近的另一根枝条上一样，我也可以从一个节点收集到所需的一切信息，以到达其父元素或子元素。

就像你可能会预料到的那样，JavaScript 是面向对象的，我们可以通过节点对象的属性来获得要查找的信息。具体来说就是 `parentNode` 和 `childNodes` 属性。由于页面上每个元素最多只有一个父元素，`parentNode` 属性很好理解：通过它可以获取当前节点的父节点。然而，每个节点可有任意数量的子节点，因此 `childNodes` 属性实际上是一个数组。该数组中的每个元素都指向一个子节点，其排列顺序跟这些节点在文档中的顺序一样。因此我们的示例文档的 `body` 元素的 `childNodes` 数组就依次包含着 `h1`，第一个 `p`，然后是第二个 `p`。

当然，上面所说的并没有包含所有节点属性。但这是个不错的起始点。那么，首先我该用什么样的代码来找到这些节点呢？我该从哪里开始我的 DOM 之旅呢？

### 从一个分枝到另一个分枝

最好的出发点就是该文档的根节点，通过那个命名很有创意的 `document` 的对象就可以访问该节点。由于 `document` 是根节点，它没有 `parentNode`，但有一个唯一的子节点：HTML 元素节点，我们可以通过 `document` 的 `childNodes` 数组来访问该节点：

```
var theHTMLNode = document.childNodes[0];
```

这一行代码创建了一个名叫 `theHTMLNode` 的新变量，并将它的值赋为 `document` 对象的第一个子节点（别忘了 JavaScript 数组的下标是从 0 开始的，而不是 1）。通过检查 `the HTML Node` 的 `nodeName` 属性可以确认自己得到的确实是 HTML 节点，该属性所提供的是你正在处理的节点类型信息：

```
alert("the HTML Node is a " + theHTMLNode.nodeName + " node!");
```

这段代码会弹出一个“**the HTML Node is a HTML node!**”的对话框。很好！`nodeName` 属性为你提供了该节点的类型。对于元素节点来说，该属性包含的是大写的标签名：此处是“**HTML**”；对于链接来说就是**A**，对于段落来说就是**P**，等等。文本节点的 `nodeName` 属性值是**#text**，而 `document` 的 `nodeName` 值就是**#document**”。

我们还可以看到，`theHTMLNode` 应当包含对自己的父节点的引用。下面这个测试可以确认这一点：

```
if (theHTMLNode.parentNode == document) {
 alert("Hooray! The HTML node's parent is the document object!");
}
```

它的确是按照我们的预期来运行的。利用这个信息，我们可以编写一些代码，来获取示例文档的 `body` 中的第一个段落的引用。该段落是 `body` 元素的第二个子节点，`body` 元素又是 `HTML` 元素的第二个子节点，而 `HTML` 元素又是 `document` 对象的第一个子节点。汗~~

```
var the HTML Node = document.childNodes[0];

var theBodyNode = the HTML Node.childNodes[1];
```

```
var theParagraphNode = theBodyNode.childNodes[1];

alert("theParagraphNode is a " + theParagraphNode.nodeName + " node!");
```

太好了。这段代码完全实现了我们想要的效果。不过它也实在是有点太长了；幸好我们有更好的方式来写这段代码。在 **JavaScript** 对象一文中，我们知道可以将对象引用链接在一起；此处我们也可以这么做，像下面这样写就可以免掉中间变量了：

```
var theParagraphNode = document.childNodes[0].childNodes[1].childNodes[1];

alert("theParagraphNode is a " + theParagraphNode.nodeName + " node!");
```

这样就清爽多了，而且还可以让你少写一点代码。

每个节点的第一个子节点无一例外的都是 `node.childNodes[0]`，而某个节点的最后一个子节点也总是 `node.childNodes[node.childNodes.length - 1]`。我经常需要访问这些节点，但重复上面代码有点麻烦。考虑到可能经常都要用到这两种表达方式，DOM 为我们提供了易于理解的快捷方式：分别是 `.firstChild` 和 `.lastChild`。由于 HTML 节点是 `document` 对象的第一个子节点，而 `body` 节点是 HTML 节点的最后一个子节点，我们可以更简便地重写前面那段代码：

```
var theParagraphNode = document.firstChild.lastChild.childNodes[1];

alert("theParagraphNode is a " + theParagraphNode.nodeName + " node!");
```

上述的一系列节点导航的方法很有用，可以让你到达文档中任何你想要去的地方，但它们也是比较麻烦的。即便是在上面简单的例子中，你也可以看出从根节点导航至所需节点有多费劲。一定还有什么办法能更好地解决这个问题！

## 直接访问

要显式地指定页面上每个感兴趣的元素的路径是很困难的。而且如果你所面对的页面是由动态方式生成的话（比如用 **PHP** 或 **ASP.NET** 之类的服务器端语言来生成的），显式指定路径就完全不可能了。因为你不可能保证，比如说，你要找的段落始终是 `body` 节点的第二个子节点。因此我们需要一个更好的方法来到达某个特定元素，同时又不用明确知道其周围的环境如何。

回顾一下前面的例子中的 **HTML** 文档，可以看到其中的段落元素有个 `id` 属性。这个 `id` 是唯一的，标志着该文档中的特定位置，这样你就可以通过利用 `document` 对象的 `getElementById` 方法来代替使用显式的路径。这个方法可以准确地实现我们想要的效果。如果你想让 **JavaScript** 查找一个在该页面上并不存在的 `id` 的话，该方法会返回 `null`，如果存在的话，则返回该元素

节点。为了试验该方法，我们比较一下新方法与老方法的结果：

```
var theParagraphNode = document.getElementById('excitingText');

if (document.firstChild.lastChild.childNodes[1] == theParagraphNode) {

 alert("theParagraphNode is exactly what we expect!");

}
```

这段代码会弹出一个确认信息，证明这两种方法具有同样的效果。`getElementById` 是最高效的获取页面上特定元素引用的方法：如果你要对某个页面上的某处进行一些处理的话（尤其是如果你不确定它到底是在哪个地方的话），在适当的地方添加一个 `id` 属性就可以为你节约不少时间。

DOM 的 `getElementsByTagName` 方法也很有用，该方法会返回页面上某个特定类型的元素的集合。举例来说，你可以通过 JavaScript 来显示出页面上所有的 `p` 元素。下面例子显示出一个精彩的段落，以及它的稍显逊色的同辈们：

```
var allParagraphs = document.getElementsByTagName('p');
```

最好是通过 `for` 循环来处理存储在 `allParagraphs` 集合中的结果：你可以像使用数组一样来使用它：

```
for (var i=0; i < allParagraphs.length; i++) {

 // 此处是我们要进行的操作的代码，我们通过

 // "allParagraphs[i]" 来引用集合中的当前元素

 alert("This is paragraph " + i + "!");
}
```

对于更复杂的文档来说，返回所有指定类型的元素可能很耗时。你很可能只要对某个特定部分的 `div` 进行操作，而不是处理某个庞大页面上的全部 200 个 `div`。在这种情况下你就可以将这两种方法联合使用，来过滤自己的结果：通过其 `id` 来获取某个元素，然后获取该元素中包

含的所有特定类型的元素。比如说，我可以通过下面的代码抓取前面那个"精彩"段落中的所有 `em` 元素：

```
document.getElementById('excitingText').getElementsByName('em')
```

## 总结

在网络中 **DOM** 几乎是 **JavaScript** 实现一切效果的基础。它是我们与自己的页面内容进行交互的接口，了解如何在该模型中进行遍历是非常必要的。

本文提供了更进一步学习所需的基本知识。现在你可以通过 `document` 获取 **DOM** 根节点，通过 `childNodes` 和 `parentNode` 来在 **DOM** 树中跳转到该节点的直系亲属节点，从而轻松地在 **DOM** 树中漫游了。你还可以通过 `getElementById` 和 `getElementsByName` 来创建自己的快捷方式，从而避开中间变量和硬编码的又长又麻烦的路径。

能遍历 **DOM** 树只是第一步。从逻辑上来讲，下一步就可以开始用你的 **JavaScript** 所返回的结果来实现有趣的效果了。你需要获取数据来为自己的脚本提供动力，并且对页面上的数据进行操作，以创建精彩的用户界面。我们将在下一篇文章探讨这些问题，下一篇文章还会告诉你如何运用 **DOM** 提供的方法来与节点及其属性进行交互，以及将该交互编写进你今后要创建的脚本和接口中去。

## 练习题

- 参考本文中的示例，写出三种不同的到达 `head` 元素的路径。别忘了你可以随心所欲地将 `childNodes` 和 `parentNode` 链接起来。
- 给定一个任意节点，你该如何测定其类型？
- 给定一个任意节点，你该怎样回到 `document` 对象？提示：别忘了 `document` 对象的 `parentNode` 属性将返回 `null`。
- [上一篇文章—\*\*JavaScript\*\* 中的对象](#)
- [下一篇文章—创建和修改 \*\*HTML\*\*](#)
- [目录](#)

## 作者简介



Mike West 本来是一个哲学系学生，但他却成了一个经验丰富且成功的 Web 开发者。他已经跟 Web 打了十多年的交道了，最近他在 Yahoo! 的欧洲新闻网站团队中工作。

在 2005 年离开了得克萨斯州郊外的广袤平原之后，Mike 定居在德国的慕尼黑，他现在对当地语言的掌握越来越好了。[mikewest.org](http://mikewest.org) 是他在网上的家，(缓慢地)收集着他的著作和链接。他将自己的代码放在 [GitHub](#) 上

## 47. HTML 的创建和修改

Posted 03/10/2009 - 23:11 by yuanc

- 网络标准教程

## 47: HTML 的创建和修改

作者 stuartlangridge · 2009 年 2 月 3 日

- 上一篇文章—DOM 之旅
- 下一篇文章——动态样式——用 JavaScript 来对 CSS 进行操作
- 目录

### 引言

在本文中我将会阐述一些基础知识，告诉你如何利用 JavaScript 来对我们的页面进行操作，这些操作包括显示或隐藏页面上部分内容，以及添加新的 HTML 元素或对其进行删除。学完本文之后你就会明白 JavaScript 的最基本的功能就是改动页面内容，此外你还会学到进行页面内容改动的最好的方法。.

本文目录如下：

- 隐藏与显示
  - 隐藏与显示例子
    - 正则表达式
    - 连接工作代码和页面
- 创建 HTML
- 总结
- 练习题

### 隐藏与显示

我们先从最容易的 JavaScript HTML 操作入手：隐藏或显示页面上已有的元素。为了实现这个目的，你可以用 JavaScript 来改变某个元素的样式。CSS 样式本身就是一种描述元素外观的有效方式，而元素外观的一个重要特性就是是否被显示。CSS 的 display 属性就是元素的显示和隐藏的关键所在：将其设为 display:none 就可以将一个元素隐藏起来。试想一下如下

的一个段落：

```
<p id="mypara" style="display: none">I am a paragraph</p>
```

该段落在页面上将会是不可见的。 **JavaScript** 允许我们动态地将 `display: none` 样式添加到某个元素上以隐藏此元素。

**JavaScript** 允许我们获取某个 **HTML** 元素的引用。举例来说，通过 `var el = document.getElementById("nav");` 就会得到 `id="nav"` 的元素的引用。在得到某个元素的引用后，就可以通过 `style` 属性来修改此元素的 **CSS**。比如说，前面的 “`mypara`” 段落目前是隐藏的（它设置了 `display: none`）。为了将它显示出来，只要将 `display` 样式属性设置为 `block` 就可以了：

```
var el = document.getElementById('mypara');

mypara.style.display = 'block';
```

瞧，这个段落出现了。通过 `style` 属性来设置某个元素的 **CSS** 跟在 **HTML** 中设置元素 `style` 属性是一样的，因此上面代码的 `mypara.style.display = 'block'` 的效果跟直接在 **HTML** 中放入 `style="display: block"` 是一样的。当用 **JavaScript** 的修改时动态的。隐藏任何元素都非常简单：

```
var el = document.getElementById('mypara');

mypara.style.display = 'none';
```

### 隐藏与显示示例

理论好是好，但此处要是有一个更实际的例子就更好了。假定有一系列的标签，在某个标签上点击鼠标就能显示该标签，同时隐藏其它所有标签。

下面就是一套标签的示例：

```
<ol class="tablinks">

Tab 1

Tab 2

Tab 3


```

```
<div class="tab" id="tab1">This is tab 1</div>

<div class="tab" id="tab2">This is tab 2</div>

<div class="tab" id="tab3">This is tab 3</div>
```

在示例页面的 head 部分(例子实际效果请参见 [tabs.html](#)), 你会看到如下所示的 CSS 和 JavaScript (一般这些代码会放在外部文件中, 并导入到 HTML 中去, 但此处我把所有代码都放在一个文件中以便阅读)。有些代码看起来很吓人, 不过别担心, 我们在后面会解释。

```
<style type="text/css">

ol.tablinks {
 margin: 0; padding: 0;
}

ol.tablinks li {
 float: left;
 border: 2px solid red;
 border-width: 2px 2px 0 2px;
 background: #eee;
 list-style: none;
 padding: 5px;
 margin: 0;
}

ol.tablinks li a {
 text-decoration: none;
 color: black;
}
```

```

div.tab {
 clear: left;
 border: 2px solid red;
 border-width: 1px 2px 2px 2px;
}

</style>

<script type="text/JavaScript">

var tabify = {

 hasClass: function(el, c) {
 return el.className.match(new RegExp('(\s|^)' + c + '(\s|$)'));
 },

 addClass: function(el, c) {
 if (!tabify.hasClass(el, c)) el.className += " " + c;
 },

 removeClass: function(el, c) {
 if (tabify.hasClass(el, c)) {
 el.className=el.className.replace(new RegExp('(\s|^)' + c + '(\s|$)'), ' ');
 }
 },

 hideAllTabs: function(o1) {
 var links = o1.getElementsByTagName("a");
 for (var i=0; i<links.length; i++) {
 tabify.setTabFromLink(links[i], "none");
 }
 }
}

```

```
 }

 },

setTabFromLink: function(link, style) {

 var dest = link.href.match(/#(.*)$/)[1];

 document.getElementById(dest).style.display = style;

 if (style == "none") {

 tabify.removeClass(link, "active");

 } else {

 tabify.addClass(link, "active");

 }

},

addEvent: function(obj, type, fn) {

 if (obj.attachEvent) {

 obj['e'+type+fn] = fn;

 obj[type+fn] = function() {obj['e'+type+fn](window.event);};

 obj.attachEvent('on'+type, obj[type+fn]);

 } else {

 obj.addEventListener(type, fn, false);

 }

},

init: function() {

 var ols = document.getElementsByTagName("ol");

 for (var i=0; i<ols.length; i++) {

 var ol = ols[i];
```

```

if (!/(^|\s)tablinks(\s|$)/.test(ol.className)) { continue; }

tabify.addEvent(ol, "click", function(e) {

 var target = window.event ? window.event.srcElement : e.target;

 if (target.nodeName.toLowerCase() === "a") {

 tabify.hideAllTabs(e.target.parentNode.parentNode);

 tabify.setTabFromLink(e.target, "block");

 if (e) e.preventDefault();

 if (window.event) window.event.returnValue = false;

 return false;

 }

}, true);

tabify.hideAllTabs(ol);

tabify.setTabFromLink(ol.getElementsByTagName("a")[0], "block");

}

}

};

tabify.addEvent(window, "load", tabify.init);

</script>

```

每个标签都是一个链接。每个链接上都有一个 `onclick` 事件处理器。该事件处理器所承担的工作有两件：它将所有的 `div` 都**隐藏**起来（通过上面的 `display: none` 方法），然后将要**显示**的标签的 `div` 的 `style` 设为 `display: block`。

你可能已经注意到了，此处的 `HTML` 的创建方式使得该页面在脚本失效的情况下依然能够运行；当我们点击一个不会显示或隐藏标签的链接时，该链接会跳转到适当的标签，所以该页面就算在不支持 `JavaScript` 的浏览器中也仍然能够正常运行。这一点很重要：这就是你可能听说过的叫做“渐进增强”的技术（或者是三年以前的术语“柔性衰减”，不过现在不再用这个名字了）。

这不仅对那些使用最新的浏览器但关闭了 **JavaScript** 的人群来说有重要意义，而且对大量的其它用户类型来说也是如此。像专供盲人用户使用的屏幕阅读器这样的辅助技术就严重依赖于语义性而且有意义的 **HTML** 结构，并且这些辅助技术对 **JavaScript** 的页面功能增强效果的支持也不如正常用户所使用的。移动电话浏览器的脚本支持也同样有限。搜索引擎读取的也是你的 **HTML**，而不是你的脚本——**Google** 可以说是世界上最视而不见的浏览器用户了。这就是“渐进增强”的意义之所在：从一个可用的页面开始，将其内容显示在最简单的环境中，比如纯文本的 **web** 浏览器，然后再逐渐添加额外的功能，并保证在添加每一步功能的同时，你的网站也仍然是可用的，而且功能也正常。

所有的 **JavaScript** 代码基本上由两部分组成：实际执行任务的部分，以及将该代码片段与 **HTML** 挂钩的部分。在本例中实际执行任务的代码是很简单的：用来显示某个链接的对应标签只需要两行 **JavaScript** 代码：

```
var dest = link.href.match(/#(.*)$/)[1];

document.getElementById(dest).style.display = "block";
```

如果你还记得的话，链接的形式类似于[Tab 1](#tab1)；第一行代码使用了一个正则表达式（参见下面的注释）来提取该链接位于# 符号之后的部分：在本例中就是字符串 tab1。链接的这一部分跟对应的 div 的 ID 是一样的（如曾经提到过的那样，该页面是按照语义意义创建的，因此“tab”链接到对应的 div 的）。然后我们通过 `document.getElementById` 来获取该 div 的引用，并像前面所谈到的那样将 `style.display` 设为 `block`。

## 正则表达式

正则表达式是一种微型程序设计语言，它被设计用来解决文本“解析”问题——比如说，用来自回答像“在那个字符串的内部有出现过这个字符串吗？”和“在字符串‘abc123def’中，‘c’和‘d’之间的数字是什么？”这样的问题。正则表达式是非常强大的工具，但也十分复杂。后面我们会对正则表达式的用途这一问题进行描述；现在先不管它们的工作原理，待会儿再回来思考这个问题吧。

本例中的“**regex**”（正则表达式英文“**regular expression**”的缩写）是`/#(.*)$/`。正则表达式中的每个字符都是用来和目标字符串中的字符序列进行比较的；正则表达式就是逐段地表达出某个目标字符串是如何构成的。

- `/ ... /` —— 斜杠标出了一个正则表达式的开头和结尾，就像双引号或单引号表示一个“字符串”一样。
- `#` —— 井号（#）的实际意思是“与一个井号相匹配”。
- `( ... )` —— 括号表示“此处是该字符串的一部分，我以后才会用到它”。
- `.*` —— 这个符号的意思是：“你想要的任意字符”；实际上它是一个点号（.），表示“任意一个字符”，后面跟着一个星号（\*），代表了“重复任意多次”。
- `$` — 美元符号表示“字符串的结束”。

因此，我们的正则表达式所描述的“匹配模式”是“一个井号，后面跟着你想要的任何字符”。`link.href` 就是我们正在处理的链接的目标地址（比如说，`#tab1`，因为它是一个符合“后跟任意字符的井号”的描述的字符串），“匹配模式”就适用于该字符串了。

因此，`link.href.match(our_regexp)` 将会返回 `true` 而不是 `false`；它真正的返回值是一个只有两项的数组，`["#tab1", "tab1"]`。第一项是与整个正则表达式相匹配的子串，第二项是与括号内部的表达式相匹配的文本；这就是为什么正则式中要有括号的原因——这是为了标记出该字符串的“这是我们感兴趣的那几个字符”部分。`tab1` 就是我们想要的字符串，因此我们就可以将其从返回数组中抓取出来（因为它是数组的第二项，所以用`[1]`就可以将它提取出来了——数组下标是从零开始的。）

## 连接工作代码和页面

就如我们前面所提到的，所有的代码都有两部分：实际执行任务的部分，以及将该代码片段与 **HTML** 挂钩的部分。将工作代码与 **HTML** 挂钩就是事件存在的目的。在这个特别的例子中，我们关注的是两个事件：用于宣告“全面开始”的 `window` 的 `load` 事件，以及当用户在某个标签上点击时触发的 `click` 事件。在页面加载的时候，我们需要运行该连接代码，而连接代码则应当将标签的 `click` 事件与前面那段代码连接起来，以显示适当的标签。

在某个事件发生时运行某段代码，这一点在大多数浏览器中都是由 `addEventListener` 函数来完成的，在 **IE** 中则是由 `attachEvent` 函数来完成的。此处我们创建的是“包装”函数，它会依据浏览器支持情况调用适当的函数：

```
addEvent: function(obj, type, fn) {
 if (obj.attachEvent) {
 obj['e' + type + fn] = fn;
```

```
obj[type+fn] = function() {obj['e'+type+fn](window.event);};

obj.attachEvent('on'+type, obj[type+fn]);

} else {

obj.addEventListener(type, fn, false);

}

}
```

(不必太担心上面代码的工作原理；现在只要不加考虑地接受就可以了——随着你的 JavaScript 经验越来越丰富，自然就会更好地理解了。) 这个函数有三个参数，`obj`，`type` 和 `fn`，分别代表“触发该事件的对象”，“我们所关注的事件”，以及“当事件被触发时要运行的函数”。在页面加载时我们要运行的是叫做 `tabify.init` 的函数；然后 `tabify.init` 函数就会负责连接每个标签的 `click` 事件。

```
addEvent(window, "load", tabify.init);
```

在前面的 `HTML` 结构中可以看到，我们实际上是把一套标签表示成了一个 `class="tablinks"` 的有序列表。因此

`tabify.init` 函数就需要完成下面这些任务：

1. 查找到页面上所有的 `class` 为 `tablinks` 的 `<ol>`。
2. 在每个 `<ol>` 的 `click` 事件上绑定如下代码：
  1. 准确地计算出用户所点击的是哪个标签链接
  2. 算出与该标签链接相对应的是哪个标签
  3. 显示该标签
  4. 隐藏其它所有标签

下面的 `init` 函数实现了全部这些功能：

```
init: function() {

var ols = document.getElementsByTagName("ol");

for (var i=0; i<ols.length; i++) {
```

```

var ols = document.getElementsByTagName("ol");

for (var i=0; i<ols.length; i++) {
 var ol = ols[i];

 if (!/(^|\s)tablinks(\s|$)/.test(ol.className)) { continue; }

 tabify.addEvent(ol, "click", function(e) {

 var target = window.event ? window.event.srcElement : e.target;

 if (target.nodeName.toLowerCase() === "a") {

 tabify.hideAllTabs(e.target.parentNode.parentNode);

 tabify.setTabFromLink(e.target, "block");

 if (e) e.preventDefault();

 if (window.event) window.event.returnValue = false;

 return false;

 }

 }, true);

 tabify.hideAllTabs(ol);

 tabify.setTabFromLink(ol.getElementsByTagName("a")[0], "block");
}
}

```

我们一步一步地来分析这个函数，依次来看该函数每一部分代码的功能：

```

var ols = document.getElementsByTagName("ol");

for (var i=0; i<ols.length; i++) {
 var ol = ols[i];

 if (!/(^|\s)tablinks(\s|$)/.test(ol.className)) { continue; }

}

```

这一部分的作用是查找出页面上所有的 **class** 为 **tablinks** 的 **<ol>** ——首先建立一张包

含所有 `<ol>` 的列表，对其中每一个 `<ol>` 执行“如果该 `<ol>` 的 `class` 不是 ‘tablinks’ 的话，就跳过它”。(`class` 的核对工作是由一个正则表达式来完成的；`continue` 的意思是“跳过这一个，到下一个去”。)

```
tabify.addEvent(ol, "click", function(e) {
 ...
});
```

这段代码的作用是将一些代码绑定到每个 `<ol>` 的 `click` 事件上去。

```
var target = window.event ? window.event.srcElement : e.target;
```

这段代码会准确地计算出用户所点击的是哪个标签链接...

```
tabify.setTabFromLink(e.target, "block");
```

...然后这段代码就会显示该标签...

```
tabify.hideAllTabs(e.target.parentNode.parentNode);
```

...最后，这行代码会隐藏其它所有标签。

代码中还有 `setTabFromLink` 和 `hideAllTabs` 函数；这两个函数用上面说过的方法隐藏或显示某个标签。我们来看看它们是怎样工作的——它们是两个独立的函数，因为将一个在多处地方都会被调用的代码块分离出来，将其写为一个函数，这通常是很管用的。（这样就能使你的代码更易于理解，而且可以在多处地方重复使用。人们有时把这叫做将代码“卸开”或“重构”到某个函数中。）

此处还有一点“额外学分”的工作，上面的代码巧妙地演示了如何添加与删除属性。  
`setTabFromLink` 函数不仅显示了适当的标签，还将“被激活”的标签的 `class` 设置成了 `class="active"`。通过三个函数，`hasClass`，`removeClass` 和 `addClass` 可以实现这个功能的。同样地，通过 `removeClass` 来从所有的 `tablinks` 的 `className` 中删掉“`active`”，然后用 `addClass` 将“`active`”添加到实际处于激活状态的那个 `tablink`。仅仅向某个链接添加一个 `class` 看起来可能没什么意义——毕竟，`class` 是不可见的——但可用通过 `class` 进行样式化。我们可以（而且已经这么做了）通过添加额外的 `CSS`，来使带有 `class="active"` 的链接的外观与其它链接区分开来：

```
ol.tablinks li a {

 background-color: red;

}

ol.tablinks li a.active {

 background-color: white;

}
```

因此，现在“active”的那个标签就成了白色的，而其它的标签是红色的。通过 JavaScript 来添加和删除 class 是非常常见的一种技术，你应该尽可能地使用它；CSS 在 HTML 元素布局，控制 HTML 元素位置和外观方面非常出色，因此利用 JavaScript 来改变这些元素的 class 就意味着它们可以采用各种不同的 CSS 样式。这是一种理想的联合方式：JavaScript 可以使你的元素改变状态，但它本身并不会对元素做多大的改变。JavaScript 做的只是给该元素添加一个 class，然后把其它活儿留给 CSS 去做。

## 创建 HTML

DOM 脚本不止能直接修改已有的 HTML 的 CSS 属性。你也可以从动态地创建页面中的新 HTML 元素。比如说，在技术新闻网站 Slashdot 上，某个位于注释之中的链接会在方括号中显示自己的目标地址，因此像

<a href="http://opera.com">A web browser</a>这样的链接就会显示成这样  
**<a href="http://opera.com">A web browser</a> [opera.com]**。（之所以这样做是为了防止人们感觉被捉弄了）我们可以很方便地用 JavaScript 来添加这段 HTML 代码，并显示页面上每个链接的目标域名。

创建新的HTML元素是通过

document.createElement 函数来实现的。在本例中，我们只需添加一样东西：在每个链接之后，我们会添加一个包含着文本的

span 元素，该元素内的文本列出了该链接的域名（实际效果请点击 [linkify.html](#)）。本例的 HTML 如下所示：

```
<p id="start">This is some text that has

links in it
```

to various places, including <a href="http://www.opera.com">Opera</a>,  
<a href="http://www.bbc.co.uk/">the BBC</a> and an internal link to  
<a href="#start">the beginning of this section</a>. All the external links  
should have <span>[domain]</span> after them.</p>

本例的 JavaScript 如下所示：

```
<script type="text/JavaScript">

var linksuffix = {

 addEvent: function(obj, type, fn) {

 if (obj.attachEvent) {

 obj['e'+type+fn] = fn;

 obj[type+fn] = function() {obj['e'+type+fn](window.event);};

 obj.attachEvent('on'+type, obj[type+fn]);
 } else {

 obj.addEventListener(type, fn, false);
 }
 },

 init: function() {

 var links = document.getElementById("linksuffixtext").getElementsByTagName("a");

 for (var i=0; i<links.length; i++) {

 var matches = links[i].href.match(/^http:\/\/(.*)\//);

 if (matches) {

 var linkdomain = matches[1];

 var span = document.createElement("span");

```

```

var spantext = document.createTextNode(" ["+linkdomain+"]");

span.appendChild(spantext);

links[i].parentNode.insertBefore(span, links[i].nextSibling);

}

}

}

};

linksuffix.addEvent(window, "load", linksuffix.init);

</script>

```

本例中实现具体功能的脚本如下：

```

var links = document.getElementsByTagName("a");

for (var i=0; i<links.length; i++) {

var matches = links[i].href.match(/^http:\/\/(.*)\//);

if (matches) {

var linkdomain = matches[1];

var span = document.createElement("span");

var spantext = document.createTextNode(" ["+linkdomain+"]");

span.appendChild(spantext);

links[i].parentNode.insertBefore(span, links[i].nextSibling);

}

}

```

下面分步解释上面代码：

```

var links = document.getElementsByTagName("a");

```

```
for (var i=0; i<links.length; i++) {
 ...
}
```

首先，它会查找出文档中所有的链接（`getElementsByName("a")`）。

```
var matches = links[i].href.match(/^http:\//(.*)\//);

if (matches) {
 ...
}
```

这一行代码在每个链接上都使用了一个正则表达式，来查看该链接的目标地址是否由 `http://something/` 开始的。如果是的话...

```
var linkdomain = matches[1];

var span = document.createElement("span");

var spantext = document.createTextNode("["+linkdomain+"]");

span.appendChild(spantext);
```

...接下来的这部分代码会首先获取“域名链接”，也就是该链接的 `www.opera.com` 部分。然后它通过

`document.createElement` 创建了一个 `<span>` 元素。接着，这段代码创建了一个“文本节点”。由于 `HTML` 元素本身是由 `document.createElement` 创建的，而 `HTML` 文档中的所有文本实际上都包含在“文本节点”中，我们就得分开创建这些文本节点。在编写具体的 `HTML` 时你不必为此担心（你甚至不必知道这一点），但在用 `DOM` 脚本创建元素的时候你就必须对这一点有所了解了。文本节点中的文本就是具体的“[域名]”，该文本是通过合并字符串（添加到一起）而创建的。最后通过 `<span>` 的 `appendChild` 方法来将文本节点放入 `span` 之内。

```
links[i].parentNode.insertBefore(span, links[i].nextSibling);
```

上面这行代码的作用是将 `span` 添加到文档中。此处，`span` 是一个 `HTML` 元素的引用，此元素如下所示：

```
 [example.com]
```

但这个元素并不是该文档的组成部分。它目前还不是任何文档的组成部分；它还在四处飘流，毫无着落。我们可以通过下面两种方式之一来将该元素添加到文档中：使用上面所说的 appendChild，或通过 insertBefore。appendChild 函数会将我们的新元素添加到某个已有元素的末端（所以它才叫做添加）。由于我们想将该元素添加到已有元素之间，我们就需要用到 insertBefore。我们当前的 HTML 是：

```
<p>... text that has
 links in it
 to ...
```

这段代码等价于图 1 所示的 DOM 树：

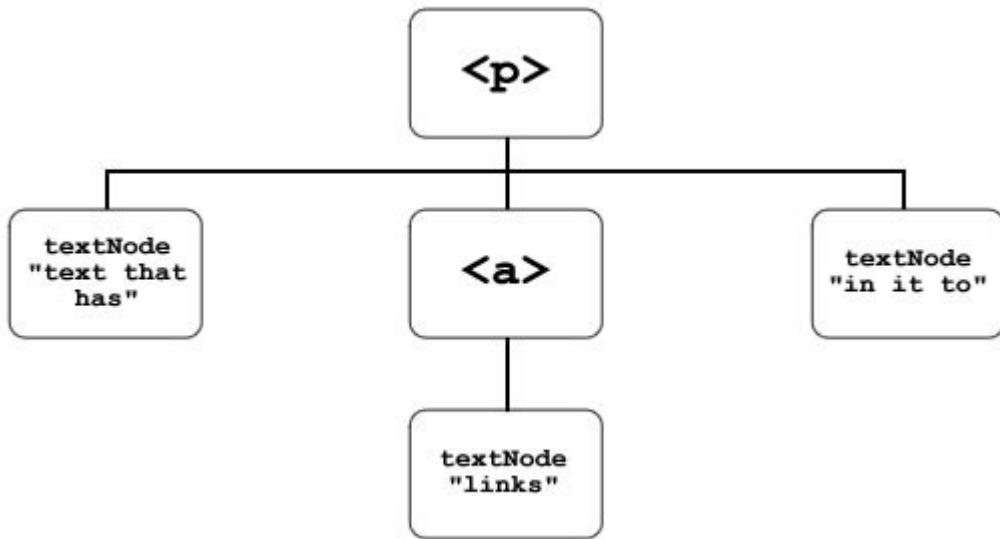


图 1：尚未在链接之后添加 span 元素时的 DOM 树。

我们应该将新的 span 元素添加到 <a> 和 “in it to” 文本节点之间，这样该元素就会出现在 <a> 后面了。改动之后的 DOM 树如图 2 所示：

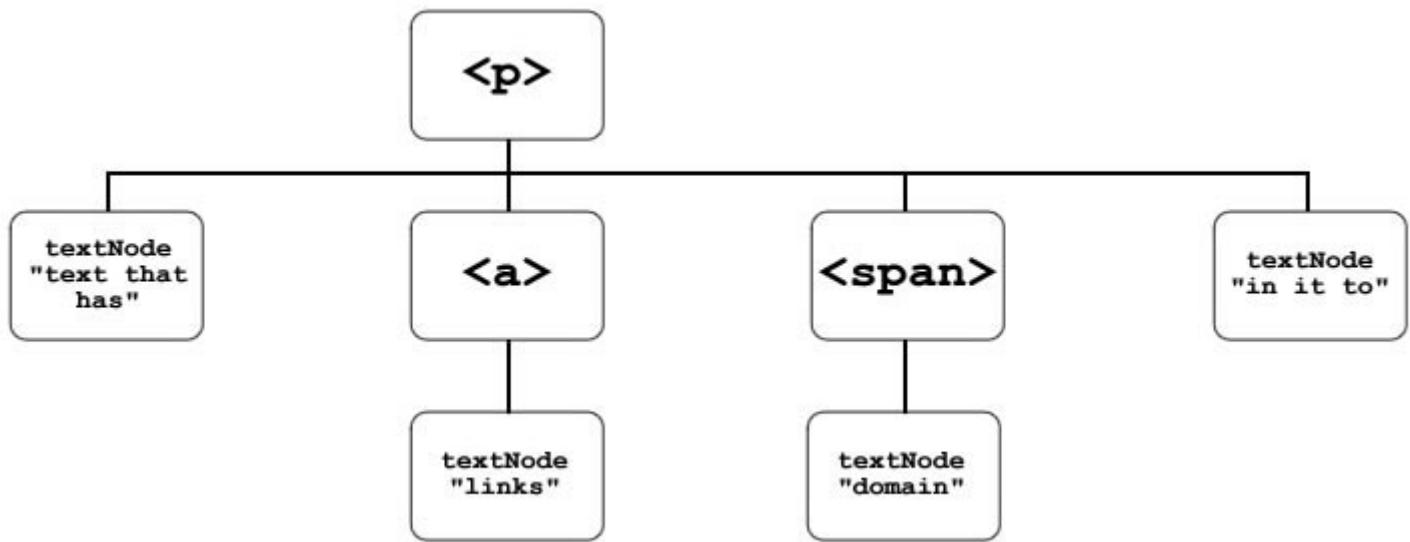


图 2：添加 span 元素之后的 DOM 树

其 HTML 如下：

```
<p>... text which has

links

 [domain] in it to ...
```

如果在此处能够直接“将我们的新 span 插入到链接之后”就更方便了。不幸的是，并没有 `insertAfter` 函数。所以，我们就得将其插入到紧跟在该链接之后的内容之前（这实在是有点混乱，但是仔细想想就能明白其含义）。到“标记为 `el` 的元素之后的内容”的最快的途径就是 `el.nextSibling`。我们需要在自己要插入的元素上调用 `insertBefore` 函数，也就是该链接的父元素 `<p>`，我们可以通过

`link.parentNode` 来快速访问其父元素。上述调用的完整形式如下：

```
links[i].parentNode.insertBefore(span, links[i].nextSibling);
```

这段代码的功能是找出我们当前正在处理的链接(`links[i]`)的父元素(`<p>`)，并将我们创建好的 `span` 元素(`span`)插入到跟在该链接之后的内容(`links[i].nextSibling`)之前。将 `HTML` 作为 `DOM` 树来处理并使用上面方法向其插入新元素的，这可能会让刚接触的人感到困惑，但随着练习增多，你很快就会变得越来越清楚的。

**总结**

**HTML** 所提供的是页面的结构，而 **CSS** 提供的是对其外观的描述。**JavaScript** 所带来的是灵活性；你的 **HTML** 结构和 **CSS** 样式都会因此成为动态的。你可以基于用户的操作动态改变页面的外观和功能。接下来我们讨论的是：从经过深思熟虑而且结构良好的信息到随用户需求而改变的数据。你可以显示和隐藏元素，改变样式和外观，还可以按照需要，添加新 **HTML** 或删除旧的 **HTML**。

## 练习题

- 如何设置某个元素的 **CSS** 的 **display** 属性，从而隐藏该元素？
- 元素和文本节点之间的区别是什么？
- 举出两个理由，说明为什么渐进增强很重要。
- **appendChild** 和 **insertBefore** 之间的差别是什么？
- 为什么我们要用 **addClass** 函数，而不是直接将新的 **class** 名和已有元素的 **className** 属性连接在一起？
  - 在下面的 **HTML** 结构中，**document.getElementById("marker").nextSibling** 所指的元素是什么？

- <p>This is a <strong>block of HTML with

```
various markup in it.</p>
```

- 上一篇文章——DOM 之旅
- 下一篇文章——动态样式——用 **JavaScript** 来对 **CSS** 进行操作
- 目录

## 作者简介



Stuart Langridge 很可能是世界上唯一一个同时拥有计算机科学和哲学学位的人了。当他没瞎捣鼓电脑时，他是 Canonical 上的一个 JavaScript , Django 及 Python 脱胎换骨者，SitePoint 的 DHTML Utopia 的作者，同时还是一个上等啤酒爱好者。他还是世界上第一个免费开源软件广播秀 LugRadio 的团队的四分之一。你可以在 [kryogenix.org](http://kryogenix.org) 上面找到他关于 Web, 脚本, 开源软件，以及所有飘过他窗前的东西的杂乱感想；你很可能在屋外找到正在寻找吸烟区的 Stuart 。

## 48. 动态样式 —— 用 JavaScript 操作 CSS

Posted 03/10/2009 - 23:12 by yuanc

- 网络标准教程

作者 Greg Schechter 2009 年 2 月 3 日

引言

在本教程的 **JavaScript** 部分中，到目前为止我们已经学过了 **JavaScript** 的基础用法，研究了如何通过 **DOM** 来获取元素并如何在获取成功后对其进行操作。

本文我们会讨论如何通过 **JavaScript** 在运行时操作 **CSS**，从而动态地更新应用到我们的元素上的式样。本文所用的技术是我们已经看到过的，但在利用 **CSS DOM** 来进行操作的时候还需要注意几个特殊点。下面的章节将讲述这些内容：

- 访问样式表
- 样式表属性
- 添加和删除 **CSS** 规则
- 改变元素样式
- 元素的 **class** 名
- 总结
- 练习题

访问样式表

浏览器提供了与样式表进行交互的接口——在 **JavaScript** 代码中可以通过 `document.styleSheets` 来访问页面上所应用的所有样式表列表，包括通过 `link` 元素引用的外部样式表，以及位于 `style` 元素之内的内部样式表。如果我们的 `style` 元素包含 `id` 属性，就可以通过 `document.getElementById(element_id)` 来快速地对其进行引用。

我们也可以给页面添加新样式表——可以通过 `document.createElement` 函数来创建新的 `style` 元素。这个函数会很有用，比如你想向网站用户提供一些选择，使他们可以动态改变你网站的风格，你可以用一些按钮控件来实现这个效果。下面就是关于你该如何创建一个新样式表的一个例子：

```
var sheet = document.createElement('style')
```

```
sheet.innerHTML = "div {border: 2px solid black; background-color: blue;}";

document.body.appendChild(sheet);
```

删除样式表的操作也很简单。首先你必须要通过 `document.getElementById` 获取自己要删除的样式表，具体代码如下面的简例所示。你可以使用 `DOM` 函数 `parent.removeChild(element)` 来删除某个样式表，此处的 `element` 就是你想要删除的样式表对象，而 `parent` 就是该样式表的父节点。如下面的例子所示，要删除该样式表(`sheetToBeRemoved`)，你首先要获取该样式表的父元素——`var sheetParent = sheetToBeRemoved.parentNode`——然后调用 `removeChild`，参数为 `sheetToBeRemoved`——`sheetParent.removeChild(sheetToBeRemoved)`。

```
var sheetToBeRemoved = document.getElementById('styleSheetId');

var sheetParent = sheetToBeRemoved.parentNode;

sheetParent.removeChild(sheetToBeRemoved);
```

样式表访问示例演示显示了如何访问该页面上所有的样式表，并向该页面添加或删除样式表。

### 样式表属性

通过 `JavaScript` 可以获取 `stylesheet` 对象，可以访问被当前的 `Web` 页面所引用的样式表的信息，比如该样式表是否是禁用的，该样式表的地址，以及其包含的所有 `CSS` 规则的列表。要了解 `stylesheet` 对象的全部属性清单（以及其他信息），请查看 [W3C 文档对象模型样式表文档](#)。

我们来看一个的例子——假设我们有一个展示一系列的技术性文章的网站。我们想要以一种美观的动画卷动的方式来突出显示某些文章，但对于那些没有启用 `JavaScript` 的人来说又该怎么办呢？想到 [非干扰性的 JavaScript](#)，我们希望这一类用户仍能使用自己网站的功能——针对这些用户，我们应以不同方式来样式化网站，从而使他们在没有图片卷动的情况下仍然能得到舒适的用户体验。

我们需要的就是一张仅在 `JavaScript` 被禁用时才有效的样式表。你运气不错——`DOM` 样式表接口为我们提供了使用 `disabled` 属性的权限，通过该属性我们就可以启用或关闭样式表了。

`stylesheet` 对象的大多数属性都是只读的，但有些属性，比如 `disabled`，就不是这样。

你也可以利用样式表属性来区分一个页面上的多个样式表。`src` 属性有助于识别外部样式

表，但它不能帮你引用内部样式表。有一种更好的方法可以让你引用内部和外部样式表，那就是使用 `title` 属性。对 `document.styleSheets` 进行迭代处理，就可以区分你网页中包含的所有样式表了。下面的例子展示的如何来实现该迭代：

```
function getStyleSheet(unique_title) {

 for(var i=0; i<document.styleSheets.length; i++) {

 var sheet = document.styleSheets[i];

 if(sheet.title == unique_title) {

 return sheet;

 }

 }

}
```

对于每个从 `stylesheet` 数组中检索到的 `stylesheet` 对象，我们都可以对其 `title` 属性进行访问，以检查它是否具有我们的代码所要查找的那个标题。你可以在规则的添加和删除示例中看到关于这一点的一个实用的例子，在下一章节中我就会谈到 规则的添加和删除。

按照用户喜好在不同的样式表之间进行切换是一种非常常见的 Web 站点功能——通过我们迄今为止所讨论的这些技术，你可以建立起多个样式表，但仅启用当前的网站用户选择的那些样式表。我们来看一个实际的例子——开始的时候文本是样式化过的，但如果我们将 `disabled` 属性设置为 `true`，我们定义好的 CSS 就会被禁用。只需将 `disabled` 设置为 `false`，你就可以把 CS 改回去。关于如何使用这一技术，具体情况你可以查看 [样式表属性示例](#)。

### 添加和删除 CSS 规则

还记得我们前面讨论过的网站吗？假设该网站有一系列文章；有些文章是关于 CSS 的，有些是关于 HTML 的，还有些文章是关于 JavaScript 的。在我们的网页上，我们会同时显示出所有的文章，但用户只想看关于 CSS 的文章。我们该怎么来满足这个需求呢？由于所有的文章都已经被显示出来了，我们不愿意再往服务器跑一趟，来获取一个只包含了 CSS 文章的页面——这是在浪费时间。

为了解决这个问题，我们可以用 JavaScript 来对所有文章进行迭代，并仅仅让 CSS 文章可见（稍后我会谈论如何实现这一点）；另一个方法是向我们的其中一个样式表添加一条规则，该规则使 CSS 文章成为唯一可见的。

`stylesheet` 对象有两个函数可以帮助我们解决这个问题。第一个是 `insertRule` 函数，就像这样：

```
stylesheet.insertRule(rule, index)
```

`rule` 是一个字符串，它包含的是我们要添加到该样式表的规则。`index` 所指定的是该规则应放在该样式表的规则列表中的什么地方。下面的代码是一个具体的使用例子：

```
stylesheet.insertRule(".have-border { border: 1px solid black;}", 0);
```

我创建了一个例子来演示如何使用 `insertRule` 函数。在这个例子中有一张列表，包含了一个样式表中的所有规则。如果我们按下该按钮，该函数就会添加一条 `index` 为 2 的规则，通过为 `p { ... }` 规则添加 `color: red` 属性将文本变成红色。点击查看这个 [添加和删除规则示例](#)。

如果想删掉这条规则，可以调用 `stylesheet.deleteRule(index)` 函数，此处 `index` 就是你想要删除的规则的 `index`。

在我们的文章陈列网站示例中，可以创建一条规则来将所有 `HTML` 和 `JavaScript` 文章的 `display` 都改成 `none` —— 具体实例请自行查看 [图片卷动实例](#)。

注：IE 没有依据标准来实现规则修改。它用的是 `rules`，而不是 `cssRules` 属性。IE 还用 `removeRule` 代替 `deleteRule`，用 `addRule(selector, rule, index)` 代替 `insertRule`。

### 改变元素样式

现在，你应该知道如何编辑与页面相连接的样式表，以及如何创建和修改其中的 CSS 规则了。如果你想要改变 DOM 中的某个特定元素的话，又该怎么办？通过 DOM API 可以访问页面上的特定元素。回顾一下我们的 [图片卷动示例](#)，当我们点击某篇文章的时候，该文章标题就会被高亮，而文章的正文则会被显示在下面。

通过 DOM 可以访问描述某元素的样式的 `style` 对象。`style` 对象被定义为 `CSSStyleDeclaration`；详细解释可以查看 [W3C CSSStyleDeclaration 接口文档](#)。`style` 对象与 HTML 元素中定义的其它属性的工作方式并不是完全一样的。与 `element.href` 或 `element.id` 不同，`element.style` 返回的是一个对象而不是字符串。因此我们不能通过将一个字符串赋给 `element.style` 的方式来设置样式。

`style` 对象的属性对应我们所设置的各种 CSS 属性。比如说，`style.color` 返回的是某个元素上所设置的颜色。通过调用 `element.style.color = "red"`；你就可以动态地改变样式。下面是一个函数，如果我们将某个元素的 `id` 传递给它的话，该函数就可以将该元素的颜色改成红色。

```
function colorElementRed(id) {

 var el = document.getElementById(id);

 el.style.color = "red";

}
```

你还可以用 `setAttribute(key, value)` 来为一个元素设置样式。比如说，你可以在某个元素上调用 `element.setAttribute('style', 'color: red')`；来将该元素的颜色设成红色，但是这种方式会擦去你之前对 `style` 对象所做的一切改动，所以在使用的时候必须小心。

通过这种方式来设置元素的样式，跟在 `html` 元素的 `style` 属性中对其进行声明是一样的。只有当该规则的重要性和具体性比应用在该元素上的其它规则要高时（具体见 [CSS 的继承和级联](#)），该样式才会得以应用。

有人可能会想，要是我们要设置的 `CSS` 属性名带有连字符的话，又会发生什么呢？在这种情况下语法就不能再跟以前一样了，我们举个例子来说明其原因，如果你写的是 `element.style.fontSize`，`JavaScript` 就从 `element.style.font` 中减去 `size`，这可不是你想要的结果。为了避免这种问题，所有的 `CSS` 属性都是用驼峰大小写来书写的。如下所示，你可以这样写 `element.style.fontSize`，来访问该元素的字号：

```
function changeElement(id) {

 var el = document.getElementById(id);

 el.style.color = "red";

 el.style.fontSize = "15px";

 el.style.backgroundColor = "#FFFFFF";

}
```

还记得那些样式表对象吗？`styleSheet.cssRules` 会返回 `style` 对象列表，该列表包含了所有位于该样式表中的 `CSS` 规则。你可以像使用其它元素的 `style` 对象一样来使用这些 `style` 对象。但此处的改动会改变所有应用了该 `CSS` 规则的元素，而不是只改变我们页面上的某一个特定元素。

在下面的例子中，使文本变大的函数使用了 `style` 对象，使文本变小的函数则用了

`setAttribute`。如果你将文本设置为红色，然后点击缩小文本按钮调用 `setAttribute`，就会发现我们所做的改动被重写了。具体实例请看 [元素样式改变示例](#)。

### 元素的 `Class` 名

另一种修改元素样式的方法是改变其 `class` 属性。`class` 是一个 `JavaScript` 保留字，你可以用 `element.className` 来访问某个元素的 `class`。如果你想将一个 `class` 添加到某个元素，你可以在 `className` 后面追加字符串，或者直接重写 `className`，并为其赋一个全新的 `class`。具体示例请看 [元素的 `class` 名示例](#)。

### 总结

知道如何动态修改页面上所应用的样式，对创建时髦而又富于交互的 `Web` 页面来说是极其有用的——本文中所阐述的知识构成了诸如 `JavaScript` 动画之类更高级的技术的基础。需要注意的是，你应当负责任地使用动态样式修改，而且不要过度滥用。如前面所述，样式修改还能提高 `Web` 效率——内容的显示和隐藏可以有助于避免在客户端和服务器之间不必要的数据交互。

### 练习题

- `setAttribute` 和通过 `CSSStyleDeclaration` 对象来设置样式之间的区别是什么？
- 列出两种方式，说明该怎样实现这样的效果：当用户点击一个按钮时，所有的图像都出现绿色的边框。
- 改变某个元素的 `CSSStyleDeclaration` 对象总是会引起该元素样式的改变吗？为什么？
- 举出两种方式，说明如何访问某个特定的样式表。

## 作者简介



Greg Schechter 是 Web 技术行业里的一个新人——目前他在伊利诺伊大学的 Urbana Champaign 分校就读计算机科学专业。他十分渴望进入互联网行业，并希望自己能为全世界的用户创建出丰富多彩而又实用的网站。关于 Greg 的更多详细资料可以在 [GregTheBusker.com](http://GregTheBusker.com) 上面找到。

## 49. 用 JavaScript 处理事件

Posted 03/10/2009 - 23:12 by yuanc

- 网络标准教程

## 49: 用 JavaScript 处理事件

作者 robnyman · 2009 年 2 月 3 日

- 上一篇文章—动态样式——用 JavaScript 操作 CSS
- 下一篇文章—JavaScript 动画
- 目录

### 引言

现在你应该能够得心应手地用 CSS 来进行样式化和页面布局了，而且也蹒跚地迈出了使用变量，函数，方法等的第一步。现在我们该开始运用这些知识来为我们的网站用户提供交互性和动态特性（比如说拖放与拖拽，动画，等等）了。用 JavaScript 来对事件进行控制，你就就可以像 Frankenstein 博士一样，为你的作品添加活力了。

我们对 JavaScript 的乐趣已经谈的够多了——本文将会更注重实际一些，告诉你什么是事件，以及如何在网页中利用事件。本文目录如下：

- 什么是事件？
- 事件是如何工作的
- 事件的演变
  - DOM level 2 events 规范
  - Internet Explorer 事件模型
  - 事件的跨浏览器应用
- 事件与可访问性
- 事件控制
  - 将事件应用到特定元素上
- 事件对象引用
  - 查看事件的某个属性
- 事件默认行为和事件冒泡

- 阻止事件的默认行为
- 阻止事件冒泡
- 完整的事件处理示例
- 总结
- 练习题

别忘了，你可以 [点击此处](#) 下载本文的代码示例，并自行尝试。

### 什么是事件？

当 Web 页面中发生了某些类型的交互时，事件就发生了。事件可能是用户在某些内容上的点击、鼠标经过某个特定元素或按下键盘上的某些按键。事件还可能是 Web 浏览器中发生的事情，比如说某个 Web 页面加载完成，或者是用户滚动窗口或改变窗口大小。

通过使用 **JavaScript**，你可以监测特定事件的发生，并规定让某些事情发生以对这些事件做出响应。

### 事件是如何工作的

如果事件发生在 Web 页面的某个 HTML 元素上，它就会检查该元素是否绑定有任何事件处理器。如果说有的话，就会按顺序分别调用这些事件处理器，并同时发送发生的事件的引用和补充信息。然后事件处理器就会对该事件进行处理。

事件顺序有两种类型：**事件捕捉** 和 **事件冒泡**。

事件捕捉是由 DOM 中 最外部的元素开始的，向内直到该事件发生之所在的 HTML 元素。比如说，在某个 Web 页面上的点击事件可能会首先检查 HTML 元素的 `onclick` 事件处理器，然后检查 `body` 元素，依次类推，直到它到达该事件的目标为止。

事件冒泡的工作方式则是刚好相反：它首先会检查该事件的目标有没有绑定任何的事件处理器，然后往上冒泡，分别经过每个父元素，直到它到达 HTML 元素为止。

### 事件的演变

在 **JavaScript** 编程的早期，我们直接将事件处理器放在 HTML 元素之内来使用，就像这样：

```
Say hello
```

这种方法的问题在于它导致了事件处理器散布在整个代码中，无法集中控制，并且无法像外部 **JavaScript** 文件那样利用浏览器的缓存功能。

在事件的演变史中接下来的用法是将事件放在一个 **JavaScript** 代码块中，比如说：

```
<script type="text/JavaScript">

document.getElementById("my-link").onclick = waveToAudience;

function waveToAudience() {

 alert("Waving like I've never waved before!");

}

</script>

My link
```

注意上面的例子中清爽的 **HTML** 代码。一般的，这种用法被称作非干扰性的 **JavaScript**。这样的优点除了能利用 **JavaScript** 的缓存和便于控制代码之外，好处还有代码分离：所有的页面内容放在一个地方，而页面交互的代码放在另一个地方。这样做还可以提供一种可访问性更好的方式，使得链接即使在 **JavaScript** 被禁用的时候也能很好地工作；这样的做法也更受搜索引擎欢迎。

### **DOM Level 2 Events 规范**

在 2000 年 11 月的时候，W3C 推出了文档对象模型（**DOM**）**Level 2 Events** 规范，该规范提供了一种更详细的更细致的方式以控制 **Web** 页面中的事件。这种新的将事件应用到 **HTML** 元素上的方法如下：

```
document.getElementById("my-link").addEventListener("click", myFunction, false);
```

`addEventListener` 方法的第一个参数就是该事件的名字，应该注意的是这里不再使用“`on`”前缀了。第二个参数是想要在该事件发生时调用的函数的引用。第三个参数控制了该事件的所谓的 `useCapture`，比如说，该使用事件捕捉还是事件冒泡。

与 `addEventListener` 对应的是 `removeEventListener`，后者会从 **HTML** 元素中删除掉所应用的任何事件。

### **Internet Explorer 事件模型**

遗憾的是，IE 到目前为止还没有实现 **DOM Level 2** 事件模型，IE 用的是它自己私有的 `attachEvent` 方法。该方法用法如下：

```
document.getElementById("my-link").attachEvent("onclick", myFunction);
```

注意 `attachEvent` 在实际事件名的前面仍然使用了“`on`”前缀，而且它也没有包含任何支持以确定使用捕获阶段。

与 `attachEvent` 相对应的是 `detachEvent`，后者会从 `HTML` 元素中删除掉所应用的任何事件。

### 事件的跨浏览器应用

由于浏览器之间在事件处理实现上的不一致，`Web` 开发者对于提供一种跨越所有主流浏览器的解决方案进行了大量尝试。这些解决方案各有其优缺点，通常我们把它们叫做 `addEvent` 函数。

大多数主流的 `JavaScript` 程序库都有内嵌这些函数，此外在网上还可以找到许多单独的解决方案。我的一个建议是使用 `Dean Edwards` 的 `addEvent`；你也可以考虑一下类似于 `jQuery JavaScript` 库的事件处理这样的解决方案。

### 事件与可访问性

在我们更深入地说明如何控制和调用事件之前，我想强调一下可访问性。可访问性对大多数人来讲是一个广义的概念，在这里我是用它来表达一个意思，就是你想通过事件的使用来实现的功能应该在 `JavaScript` 被禁用的时候，或者是由于其它原因导致 `Web` 浏览器无法使用 `JavaScript` 的时候，仍然能够工作。

有些人确实是无意关掉自己的 `Web` 浏览器中的 `JavaScript` 的，但更常见的是代理服务器，防火墙，以及过度敏感的防病毒程序阻止了 `JavaScript` 的正常运行。千万别受这些特殊情况打击；我的目的就是教你创建包含可访问的备用版本的事件，以防 `JavaScript` 不可用的情况。

一般而言，不要在那些不具备默认行为的 `HTML` 元素上应用事件。你只能对 `a` 这样的元素应用 `onclick` 事件，因为该元素已经具备了针对点击事件的备用行为（比如，转到该链接所指定的位置，或提交表单）。

### 事件控制

我们先来看一个关于事件的简单例子，演示如何对其做出反应。为了保持简单，我会用前面提过的 `addEvent` 方案，免得在每个例子中都要牵扯到错综复杂跨浏览器措施。

我们的第一个例子是 `.onload` 事件，该事件属于 `window` 对象。一般而言，任何会影响浏览器窗口的事件（比如说 `onload`, `onresize` 和 `onscroll`）都可以通过 `window` 对象来获取。

`Web` 页面上的所有内容加载完毕时就会触发  `onload` 事件，这里的内容包括 `HTML` 代码

以及像图像， CSS 文件和 JavaScript 文件之类的外部依赖。所有这一切都加载完毕后，就会调用 window.onload，这样你就触发相应的 Web 页面功能了。下面这个非常简单的示例在页面加载完毕的时候弹出一个警报消息框：

```
addEvent(window, "load", sayHi);

function sayHi() {
 alert("Hello there, stranger!");
}
```

还不错，对吧？如果你愿意的话，你还可以用所谓的匿名函数来代替这种方式，就不需要函数名了。如此一来，上面的代码就变成了这样：

```
addEvent(window, "load", function () {
 alert("Hello there, stranger!");
});
```

### 将事件应用到特定元素上

为了更深入地阐述这个问题，我们先来看看如何将事件添加到页面中其它元素上去。我们假设你想让一个事件在每次点击链接的时候都发生。联系前面所学到的知识，也许我们可以这样来处理：

```
addEvent(window, "load", function () {

 var links = document.getElementsByTagName("a");

 for (var i=0; i<links.length; i++) {

 addEvent(links[i], "click", function () {
 alert("NOPE! I won't take you there!");

 // This line's support added through the addEvent function. See below.

 evt.preventDefault();
 });
 }
});
```

```
 }

});
```

OK，刚刚发生了什么？首先，我们使用了 `onload` 事件以检测 Web 页面是否加载完毕。然后通过使用 `document` 对象的 `getElementsByName` 方法，我们找出了该页面中的所有链接。我们在所有链接中循环，并将事件应用到这些链接上，这样，一旦这些链接被点击就会引发某个操作。

“won’t take you there”部分之后的代码又是怎么回事呢？在显示 `alert` 信息之后，它下面的一行代码的作用如同 `return false`。它的意思是，返回 `false` 来阻止默认操作的运行。在本文最后的部分中我们还会涉及此问题。

### 事件对象引用

为了更好的处理事件，你可以根据所发生的事件的特定属性来采取不同的操作。比如说，如果你要处理 `onkeypress` 事件，你可能会希望该事件只在用户按下 `enter` 键的时候才发生，而在按下其他键时不发生。

如事件模型一样，IE 和其他浏览器处理方法不同：IE 使用一个叫做 `event` 的全局事件对象来处理对象，而其它所有浏览器采用的 W3C 推荐的方式，仅传递特定事件的事件对象爱你个。跨浏览器实现这样的功能时，最常见的问题就是获取事件本身的引用及获取该事件的目标元素的引用。下面这段代码就为你解决了这个问题：

```
addEvent(document.getElementById("check-it-out"), "click", eventCheck);

function eventCheck (evt) {

 var eventReference = (typeof evt !== "undefined")? evt : event;

 var eventTarget = (typeof eventReference.target !== "undefined")? eventReference.target :
 eventReference.srcElement;

}
```

`eventCheck` 函数中的第一行代码会检查是否有传递某个事件对象给该函数。如果有的话，该事件对象就会自动成为该函数的第一个参数，在本例中就是 `evt`。如果不存在这样的事件对象的话，意味当前的 Web 浏览器是 IE，该函数就会引用 `window` 对象的 `event` 全局属性。

第二行代码会在已经建立好的事件引用的基础上查找 `target` 属性。如果该属性不存在，

则会使用 IE 实现的 `srcElement` 属性。

注：上文提到的 `addEvent` 函数中也解决了此问题，该函数中事件对象在所有的 Web 浏览器中有完全相同的工作方式。不过，上面的代码故意忽视 `addEvent` 函数这一特点，为了让你对 Web 浏览器之间的差异有个深刻理解。

### 查看事件的某个属性

让我们来付诸实践。下面的例子会根据所按的键执行不同的代码：

```
addEvent(document.getElementById("user-name"), "keyup", whatKey);

function whatKey (evt) {

 var eventReference = (typeof evt !== "undefined")? evt : event;

 var keyCode = eventReference.keyCode;

 if (keyCode === 13) {

 // The Enter key was pressed

 // Code to validate the form and then submit it

 }

 else if (keyCode === 9) {

 // The Tab key was pressed

 // Code to, perhaps, clear the field

 }

}
```

`whatKey` 函数内部的代码会检查所发生的事件的特定属性，也就是 `keyCode`，来看看到底按下的是键盘上的哪个按键。数字 13 的意思是 Enter 键，而数字 9 代表 Tab 键。

### 事件默认行为和事件冒泡

在许多情况下你可能需要阻止某个事件的默认行为。比如说，你可能想在特定区域未被填写的情况下阻止用户提交表单。同样的，有时你可能会需要阻止事件冒泡，本文的这一部分讲述的就是在这种情况下如何阻止事件默认行为和事件冒泡：

### 阻止事件的默认行为

就像事件模型和事件对象差异一样，在 IE 和其它所有浏览器中阻止事件的默认行为的方法也不同。用前面的代码作为获取事件对象引用的基础，下面的代码介绍如何在链接被点击时，阻止链接默认行为：

```
addEvent (document.getElementById("stop-default"), "click", stopDefaultBehavior);

function stopDefaultBehavior (evt) {

 var eventReference = (typeof evt !== "undefined")? evt : event;

 if (eventReference.preventDefault) {

 eventReference.preventDefault();

 }

 else {

 eventReference.returnValue = false;

 }

}
```

这种方法用到了叫做对象检测的技术，以在某个方法被调用之前先确认它是否真正可用，这样做有助于防止潜在的错误。这里的 preventDefault 方法在除 IE 外的所有浏览器中都可用，它可以阻止事件的默认行为的发生。

若此方法不被支持，该函数将全局事件对象的 returnValue 设置为 false，从而阻止该事件的默认行为在 IE 中的运行。

### 阻止事件冒泡

考虑下面的 HTML 代码：

```
<div>

Opera


```

```


Opera Dragonfly

</div>
```

假设你在所有的 `a` 元素, `li` 元素和 `ul` 元素上都应用了 `onclick` 事件。`onclick` 事件首先调用的是链接的事件处理器, 然后是列表项, 最后调用的是无序列表的事件处理器。

用户点击链接时, 你很有可能不需要调用其父元素 `li` 元素的事件处理器, 只需要使用户跳转至对应页面去。不过, 如果用户点击的是该链接旁边的 `li` 项, 你可能会想要触发 `li` 的事件处理器, 同时也触发 `ul` 的事件处理器。

注意, 在 **DOM level 2** 模型和 `useCapture` 都启用的情况下, 也就是使用事件捕捉时, 事件处理就会从无序列表开始, 然后是列表项, 最后才是链接。然而由于 IE 不支持事件捕捉, 实际应用中很少使用该功能。

关于如何编写代码以阻止某个事件的冒泡, 可以参照下面的程序:

```
addEvent(document.getElementById("stop-default"), "click", cancelEventBubbling);

function cancelEventBubbling (evt) {

 var eventReference = (typeof evt !== "undefined")? evt : event;

 if (eventReference.stopPropagation) {

 eventReference.stopPropagation();

 }

 else {

 eventReference.cancelBubble = true;

 }

}
```

## 完整的事件处理示例

我在 [示例页面](#) 中，演示了如何添加事件处理器，及在某种条件下阻止事件的默认行为。。这例子中的事件处理器会根据用户是否填写完所有的表单域来确定是否可以提交表单。本示例的 **JavaScript** 代码如下：

```
addEvent(window, "load", function () {

 var contactForm = document.getElementById("contact-form");

 if (contactForm) {

 addEvent(contactForm, "submit", function (evt) {

 var firstName = document.getElementById("first-name");

 var lastName = document.getElementById("last-name");

 if (firstName && lastName) {

 if (firstName.value.length === 0 || lastName.value.length === 0) {

 alert("You have to fill in all fields, please.");

 evt.preventDefault();

 }

 }

 });

 }

});
```

## 总结

本文仅仅只触及了事件处理的毛皮，但我希望你能理解事件是如何工作的。对初学者来说，本文中关于浏览器不一致性的部分可能有点太难了，但我相信从一开始就知道兼容性问题是非常重要的。

一旦你领会了这些问题，并掌握了前面所说的那些解决方案，你通过 **JavaScript** 和事件处理所能实现的效果将是无限量的！

## 练习题

- 什么是事件？
  - 事件捕捉和事件冒泡之间的差别是什么？
  - 有没有可能对一个事件的执行加以控制，比如阻止其默认行为？该怎么做？
  - `attachEvent` 的主要问题和使用范围是什么？
- 
- 上一篇文章—动态样式——用 JavaScript 操作 CSS
  - 下一篇文章—JavaScript 动画
  - 目录

作者简介



Robert Nyman 从事 Web 界面开发已经有十年了，而 JavaScript 始终是他的主要兴趣之所在。他充满热情地在 [Robert's talk](#) 上面写些关于 Web 开发的博客文章。

他和他美妙的家庭一起居住在瑞典，并且在家人都熟睡了的时候还在熬夜写作。另外，他偷偷地梦想着有一天自己会富得不能再富，还希望有机会能超越 Web 世界的限制，写上一本关于真实事物的书。

## 50. JavaScript 动画

Posted 03/10/2009 - 23:12 by yuanc

- 网络标准教程

## 50: JavaScript 动画

作者 stuartlangridge · 2009 年 2 月 3 日

- 上一篇文章— 用 JavaScript 处理事件
- 下一篇文章— 平稳退化对渐进增强
- 目录

### 引言

在本文中，我们将学习通过 JavaScript 创建动画效果——动画效果常常用来提高那些浏览器支持该功能的用户的体验。最常见的动画效果是可平滑伸缩面板、进度条、以及表单中的视觉反馈等。

看过卡通片或翻书动画的人都知道，动画其实是通过许多时间间隔很小的帧来实现的，这些帧连续播放使它看起来就像在动一样。动画是一种强大的技术，在吸引用户注意力这方面非常出色。但动画的缺点就是它总是会吸引人们的注意力，不管你想不想要它这么做。动画效果可以提高用户的 Web 应用程序使用体验，但就像辣椒一样：不要加太多哦。

本文目录如下：

- 一个简单的示例：黄色淡出技术
- 用 JavaScript 程序库来做动画
- CSS 过渡
- 更复杂的示例：位置的移动和大小的改变
- 总结
- 练习题

### 一个简单的示例：黄色淡出技术

一种常见的动画应用就是 黄色淡出技术：页面上被改动的区域的背景颜色会被设为黄色，然后会逐渐消褪，直到还回原来的背景。这是一个既美观又非干扰性的强调改动过的内容（比如

显示了更多内容，或某些表单反馈信息）的方式，而不会干扰到用户正在进行的事情。点击此处查看 [黄色淡出技术实例](#)。

黄色淡出技术背后的原理就是，将元素的背景颜色设置为黄色，然后通过一系列的步骤，将其逐渐变回原来的颜色。因此如果原来的颜色是红色的话，背景颜色就会先设成黄色，然后是橙黄色，然后是橙色，然后是桔红色，然后成为红色。你所采用的步骤的数量决定了颜色变化的平滑度，而步骤之间的时间间隔决定了整个颜色变化的过程有多长。进行颜色变化时，我们可以利用一个很实用的 CSS 特性：颜色可以由三个一组的普通数字来定义，也可以用十六进制的字符串来定义。因此 #FF0000（红色）也可以写成 rgb(255, 0, 0)。经过五个步骤从 rgb(255, 255, 0)（黄色）改变成 rgb(255, 0, 0)（红色）的可以写成到：

```
rgb(255, 255, 0)
```

```
rgb(255, 192, 0)
```

```
rgb(255, 128, 0)
```

```
rgb(255, 64, 0)
```

```
rgb(255, 0, 0)
```

我们先将元素的背景颜色设置为 rgb(255, 255, 0)，然后过一段时间（假设是 100 毫秒），将背景颜色改成 rgb(255, 192, 0)，然后再过 100 毫秒，又设置成 rgb(255, 128, 0)，依此类推：

颜色	时间
rgb(255,255,0)	0
rgb(255,192,0)	100ms
rgb(255,128,0)	200ms
rgb(255,64,0)	300ms
rgb(255,0,0)	400ms

整个过程耗时 400 毫秒（还不到半秒钟），从黄色到红色的褪色过程很平滑。很方便的是，此处我们只需对颜色的一个通道进行改变（就是绿色的那个通道；rgb 色彩的三个通道是红色，绿色，和蓝色通道）；当然也可以同时改变多个颜色通道。在本例中，我们经四个步骤将绿色通道

道从 255 变到了 0，也就是说每一步将其值改变 64。

在 JavaScript 中，要在一段特定的时间之后触发某项操作，是通过 `setTimeout` 和 `setInterval` 函数来实现的。`setTimeout` 函数会在某个时间延迟之后执行你规定的操作；而 `setInterval` 则会一遍又一遍地执行该操作，保持每两次执行之间等待一段延时；此函数很合适实现动画效果。从本质上讲，实现这个褪色效果的方法就是计算出每个步骤，然后用 `setInterval` 来依次地调用它们。`setInterval` 函数有两个参数：一个是作为要执行的操作而调用的函数，还有一个以毫秒为单位的时间延迟。

显然地，你并不是每次都要从黄色变到红色的，因此这个函数的通用性应该更好一些。如果你知道起始颜色和最终颜色，以及所经过的步骤的数量的话，这个问题就成了计算出每一步要对每个颜色改变多少的数学问题了。如果你将 `startcolour` 数组定义为一张三个数字的数组 (`[255, 255, 0]`)，并将 `endcolour` 定义为类似的数组 (`[255, 0, 0]`) 的话，每一步对每个颜色的改变量就是：

```
var red_change = (startcolour[0] - endcolour[0]) / steps;

var green_change = (startcolour[1] - endcolour[1]) / steps;

var blue_change = (startcolour[2] - endcolour[2]) / steps;
```

用 `setInterval` 来逐步地改变该元素的背景颜色：

```
var currentcolour = [255, 255, 0];

var timer = setInterval(function() {

 currentcolour[0] = parseInt(currentcolour[0] - red_change);

 currentcolour[1] = parseInt(currentcolour[1] - green_change);

 currentcolour[2] = parseInt(currentcolour[2] - blue_change);

 element.style.backgroundColor = 'rgb(' + currentcolour.toString() + ')';

}, 50);
```

在每一步中，该函数都会读取 `currentcolour`，并将红色通道改动 `red_change`，将绿色通道改动 `green_change`，并将蓝色通道改动 `blue_change`。首先将该元素的背景颜色设置为新颜色：`[255, 255, 0].toString()` 也就是 "255,255,0"，因此为了得到颜色 `rgb(255, 255, 0)`，我们用 `toString()`

来创建 `rgb(255, 255, 0)`，并将其设置为该元素的背景颜色。

然而，`setInterval` 会无限次地调用你的函数；即使目标颜色已经达到了，它也不会停下来。为了使 `setInterval` 停下来，我们可以使用 `clearInterval()`；下面的代码会对该操作的调用次数进行计数，并在适当的步骤数之后停止动画：

```
var currentcolour = startcolour;

var stepcount = 0;

var timer = setInterval(function() {

 currentcolour[0] = parseInt(currentcolour[0] - red_change);

 currentcolour[1] = parseInt(currentcolour[1] - green_change);

 currentcolour[2] = parseInt(currentcolour[2] - blue_change);

 element.style.backgroundColor = 'rgb(' + currentcolour.toString() + ')';

 stepcount += 1;

 if (stepcount >= steps) {

 element.style.backgroundColor = 'rgb(' + endcolour.toString() + ')';

 clearInterval(timer);

 }

}, 50);
```

这就是实现动画的方法：每次执行一个步骤。

怎样设置 `startcolour` 和 `endcolour` 呢？一个简单的方法就是将上面的代码封装到一个 `fade` 函数中：

```
fade: function(element, startcolour, endcolour, time_elapsed) {

 ... code from above...

}
```

然后就可以通过一个函数调来以实现某个元素的黄色淡出效果了：

```
fade(document.getElementById("yft"), [255, 255, 60], [0, 0, 255], 750);
```

或者也可以是“红色淡出”，也就是将该元素设为红色，然后逐渐褪化为蓝色（该元素的背景颜色），就像这样：

```
fade(document.getElementById("yft"), [255, 0, 0], [0, 0, 255], 750);
```

这个例子是用来改变背景颜色的，但它也适用于其它任何东西的改变：前景颜色（用于制作 1960 年代那种风格的令人眼花缭乱的迷幻文字特效），不透明度（用来使某些事物淡出或淡入），位置（用于使某个元素在页面上移来移去），高度和宽度（使某个元素变大，或将其缩小至无物以实现消失效果）。

### 用 JavaScript 程序库实现动画

动画是一种常用的效果，因此大多数 JavaScript 程序库都具有某些动画效果支持，支持常见动画的。举例来说，jQuery 就有一种内置支持能使元素逐渐淡化直至透明：

```
$("#myelement").fadeOut();
```

还有一个 animate() 函数，可以用来实现更复杂的自定义效果：

```
$("#block").animate({
 width: "70%",
}, 1500);
```

这段代码非常直观——它将会读取该元素，并在 1500 毫秒时间内将 CSS 的 width 属性改为原来的 70%——此处是 animate 函数的文档。

Prototype 的 scriptaculous 框架也提供了类似的功能，比如 Effect.Fade('id\_of\_element')，以及许许多多其它的效果。Yahoo UI 库也可以实现类似效果：

```
new Y.Anim({ node: '#demo', to: { width: 70%, }}).run();
```

如果你在自己的代码中已经用到过某个 JavaScript 程序库，你就该知道它们所提供的动画功能比你自己用 setInterval 来做动画要容易得多。不过，我觉得了解藏在表面下的机制是很重要的——从长远来看，这会使你的脚本编写能力越来越强。这就是为什么我要在讨论程序库之前以例子的形式介绍如何创建简单动画。

你可以在 [dev.opera.com](http://dev.opera.com) 的 [JavaScript 工具包简介](#)一文中找到更多关于如何使用各种 JavaScript 程序库的资源。

### 一个更复杂的示例：位置的移动和大小的改变

虽然黄色渐变技术的示例阐释了动画基本原理，但它有点太乏味。大多数人在想到动画的时候，他们所想到的是运动。如果 Wile E. Coyote 所做的东西都只是改变改变颜色的话，他可能就没那么有趣了。

有一个很不错的技巧可以用来提醒用户页面中所发生的事情，而又不会打断他们的工作流程，这就是 [非模态消息](#)。与弹出一个 `alert()` 对话框不同（然后只有等用户点击了 `K` 他们才能继续操作），非模态消息会直接将消息放在页面上的一个浮动的 `div` 上，该 `div` 会一直不显眼地呆在那里，直到用户确认已收到信息。另一个优点是允许用户再次阅读自己已确认的消息。因此，让我们来实现一个浮动信息，点击该信息后，它就会“缩到”屏幕的一角去，然后再点击一下还可以还回来。你可以 [点击此处查看“缩放信息”效果](#)。

如果你要做任何正式的动画效果，或任何正式的 `JavaScript` 工作的话，`JavaScript` 程序库总是值得一用的。这就使得你可以为用户提供你所希望提供的用户体验，而不用操心实现该动画所需的数学细节了。（读完前面的第一个例子后，现在你知道该怎样进行计算，以及如何使用 `setInterval` 了，不过，让别人帮你做那些重活儿可以为你节省不少时间，还有脑细胞。）

前面的演示中用到了 `jQuery` 程序库来实现效果，但就如我们提到过的那样，大多数程序库都提供了非常相似的动画支持，因此你应该能用自己喜欢的程序库来实现动画。大体上来讲，我们应该这么做：

1. 在屏幕中央显示出一个浮动信息
2. 当该信息被点击时：
  1. 将其在水平方向上的位置移至右侧
  2. 将其在垂直方向上的位置移至顶部
  3. 将其宽度改为 `20 px`
  4. 将其高度改为 `20 px`
  5. 将其不透明度改为 `20%`，这样该信息就接近透明了，并将其中的文字隐藏
3. 点击这个“迷你”版的信息时，使其回到屏幕中央（跟我们刚才使其缩小的操作刚好相反）

这样，用户就能清楚地了解自己的信息到底怎么了，从完整大小的信息到迷你信息的转化过程应当是平滑的动画（这样用户就能目睹自己的信息“缩到”窗口的一角）。

用 `jQuery` 来实现这个动画是很简单的：只需使用 `.animate()` 函数，并告诉它你想要的该

动画的最终效果就行了（以及你希望整个过程持续多长时间）：

```
$(ourObject).animate({
 width: "20px", height: "20px", top: "20px",
 right: "20px", marginRight: "0px", opacity: "0.2"
}, 300);
```

上面这段代码会获取 `ourObject`，并在 300 毫秒的时间内，将其宽度和高度变成 `20px`，将其顶部和右侧的位置变为 `20px`，将其 `margin-right` 样式属性变成 `0px`，并将其不透明度（在支持不透明度的浏览器中）变成 `20%`。然后只需使用 `jQuery` 风格编程，以实现在点击该信息时产生动画效果的问题了：

```
$(ourObject.click, function() {

 $(this).animate({
 width: "20px", height: "20px", top: "20px",
 right: "20px", marginRight: "0px", opacity: "0.2"
 }, 300)

});
```

再次点击时恢复该信息，要实现这个效果只需再次调用 `.animate()` 就可以了

```
$(ourObject).animate({

 width: "400px", height: "75px", top: "50px",
 right: "50%", marginRight: "-200px", opacity: "0.9"
}, 300);
```

此外还需要用一点逻辑结构来确定当前该信息是处于显示状态还是收缩状态（这样你才能知道该执行哪个动画），以及需要一些 `CSS`，用来描述该信息的初始样式（大小，颜色为绿色，位置为水平居中），这就是所需的全部东西。仅仅三十行代码的脚本就可以实现这个效果。这就是为什么说脚本库真是个好东西的原因！

## CSS 过渡

最后要说明的是，有些（但不是所有）动画实际上不需要任何 JavaScript 就可以实现！Safari 和其它基于 Webkit 的浏览器，还有马上就要发行的 Firefox3.1，都可以实现从一个 CSS 值到另一个值的平滑过渡，而不需要任何的 JavaScript。下面这段代码：

```
div { opacity: 1; -webkit-transition: opacity 1s linear; }

div:hover { opacity: 0; }
```

在支持此特性的浏览器中，上面代码在鼠标悬停的时在一秒钟的时间内将 div 平滑地淡出。不过，CSS 过渡是很新的技术，只有最新的浏览器才支持此特性，因此如果你想让自己的动画能为大多数人群所使用，那你就得通过 DOM 脚本来实现了。

## 总结

本文介绍了如何利用 JavaScript 实现 Web 页面动画功能——我们探讨了如何通过 setTimeout 和 setInterval 函数创建动画示例，然后我们讨论了如何利用 JavaScript 脚本库来更快捷地创建动画。

## 练习题

1. setTimeout 和 setInterval 之间的差别是什么？
  2. 如果不存在 setInterval 这个函数，你可以怎样模拟出一个这样的函数？
  3. 你该如何使一个元素从完全可见褪化到完全不可见，整个过程分 20 步，耗时 1.5 秒？
  4. 你该如何使一个元素从完全可见褪化到完全不可见，然后又回到可见状态，整个过程分 20 步，耗时 1.5 秒？
- 上一篇文章—用 JavaScript 处理事件
  - 下一篇文章—平稳退化对渐进增强
  - 目录

## 作者简介



**Stuart Langridge** 很可能是世界上唯一一个同时拥有计算机科学和哲学学位的人了。当他没瞎捣鼓电脑时，他是 [Canonical](#) 上的一个 JavaScript , Django 及 Python 脱产，[SitePoint](#) 的 [DHTML Utopia](#) 的作者，同时还是一个上等啤酒爱好者。他还是世界上第一个免费开源软件广播秀 [LugRadio](#) 的团队的四分之一。你可以在 [kryogenix.org](#) 上面找到他关于 Web , 脚本，开源软件，以及所有飘过他窗前的东西的杂乱感想；你很可能在屋外找到正在寻找吸烟区的 **Stuart** 。

Posted 03/10/2009 - 23:13 by yuanc

- 网络标准教程

## 51: 功能衰减 VS 渐进增强

作者 Christian Heilmann · 2009 年 2 月 3 日

- 上一篇文章—JavaScript 动画
- 目录

### 引言

在本篇教程中，我们将讨论两种开发方法之间的差别：功能衰减和渐进增强。下面是其简单定义：

#### 功能衰减

作为一种确保某种产品的可用性的安全措施，功能衰减可以为你的功能提供一种替换版本，或让用户意识到该产品的缺点。

#### 渐进增强

渐进增强是从一个功能可用的基本版本出发，然后逐步增加用户体验的丰富程度，并在应用增强功能之前先测试对该功能的支持性。

你可能会觉得这两种方法听起来非常相似，而且它们的效果也应该是几乎一样的，但这两种方法之间的确有需要加以注意的差别，下面我们就对这个问题进行讨论。

我们首先会说明为什么需要这两种技术。然后我们会关注一下更深层次的定义，本文还展示了一些实际案例，再接下来是这两个概念之间的对比，以及讨论在什么时候该使用哪一种方法。我们先来说明为什么 Web 开发需要着两种开发方法。

本文结构如下：

- “Mobilis in mobile”——迁入一个不断变化的环境中
- 功能衰减和渐进增强
- 功能衰减对渐进增强的一个示例
  - “打印本页面”链接
- 什么时候该用什么
- 总结

- 练习题

### “Mobilis in mobile”——迁入一个不断变化的环境中

就像“海底两万里”中的尼莫船长一样， Web 开发者们也发现他们自己处在一个不断变化的环境中，这个环境可能对我们想要实现的东西不太友好。

Web 在发明和定义之初，就是要能在任何显示设备中使用，而且可以使用任何语言，及在你想要的任何地方使用。终端用户只期望一件事，那就是他们所使用的浏览设备可以触及网络，并能理解用来传递信息的协议——`http`, `https`, `ftp`, 等等。

这就意味着我们不能对我们的终端用户的设置和设备能力存有任何假设。我们还可以很肯定的是，作为开发者，我们对网络的理解跟我们想要触及的人群完全不一样。

你不能强制用户进行技术升级以获取互联网上的内容。个人用户和公司用户会倾向于待在一个确定的环境中，他们不会仅仅因为我们的希望就选择改变或升级。许多人只想使用网络，并不会去注意其背后的技术——他们所期望的只是获取我们所承诺提供的内容。让用户的系统保持在最新状态，这是操作系统和浏览器开发商的事情——作为 Web 开发者的我们对此没有任何发言权。

所有这些导致了一个非常脆弱的开发环境，比如某些办公室的默认配置是 9 年以前的老式浏览器，而且脚本和插件都是禁用的（由于安全性的原因），低分辨率的显示器，还有仅仅只够运行 Office 软件的计算机，这样的办公室环境是很常见的。

现在我们可以转头走掉，并声称这样的公司是“坐失良机”的，试图支持那些过时的技术是没有意义的。但这种态度可能会导致我们忽视这些人群，而他们可能对于我们产品的成功与否是至关重要的。在许多情况下他们没有权限改变自己的技术配置。在涉及到可访问性的时候，这些问题就更加地显而易见了：一个诵读困难的终端用户没办法理解我们那些复杂的说明，而一个盲人用户也不能“点击绿色按钮以继续”，尽管我们已经下令说这对于使用我们的系统而言是必要的。

我们要跟未知打交道，我们需要找到一种方法来保证我们的产品正常运行。这就是功能衰减和渐进增强派上用场的地方。

#### 功能衰减和渐进增强

前面我们已经给出了关于两者的一个简单的定义；在这一部分中我会给出更具有技术性的定义，并探讨对这些方法论进行网页开发到底有什么意义。

**功能衰减**是创建 Web 功能的一种方法，通过这种方法，如果用户使用的是先进的现代浏览器中，则网站能提供高水平的用户体验；如果用户使用的是较老式的浏览器，网站也能柔性地

衰减到一个较低水平的用户体验。这个较低的水平对于网站用户来说，使用起来不那么美观，但它仍然能为用户提供所需的基本功能；对这些用户来说你的网站也不会变得不可用。

**渐进增强**也是类似的，但它是以另一种方式来实现这个目的的。你是从建立一个基础水平的用户体验开始的，所有的浏览器在渲染你的网站时都能提供这种水平的体验；不过你还会在此基础上加入更高级的功能，支持这些功能的浏览器会自动使用这些功能。

换句话说，功能衰减是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要。功能衰减意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带。

### 功能衰减对渐进增强的一个简单示例

我们来看一个例子，这个示例展示了两种实现方式，一种利用渐进增强，而另一种利用功能衰减。

#### “打印本页面”链接

按理说，允许用户打印当前文档的链接是没什么用处的——点击浏览器中的打印机图标也可以达到同样的效果。然而用户测试表明，作为预订过程的最后一个步骤（比如在某个航空公司的网站上），这些链接是一种不错的可供用户进行重新确认的方式，值得一用。通过这些链接，用户会感觉自己拥有控制权，而且也会觉得这样就完成了自己开头要做的事情。

“打印本页面”链接的问题在于，我们无法用 `HTML` 来链接到浏览器中的打印按钮——你得用 `JavaScript` 来实现这一点。在 `JavaScript` 中这是很容易的——浏览器的 `window` 对象有一个 `print()` 方法，可以调用此方法来启动打印输出。实现这一点的最常见的方法可能就是使用 `javascript:` 伪协议了：

```
<p id="printthis">
 Print this page
</p>
```

这段代码只有在 `JavaScript` 可用，同时也处于启用状态，并且浏览器支持 `print` 命令的情况下才有效。如果 `JavaScript` 不可用（比如说在某些移动设备上），这个链接就无法工作——点击该链接不会出现任何反应。这就导致了一个问题，就是作为网站开发者，你可是向你的访客承诺了提供这个功能的。如果他们点击这个链接，而该链接完全没有用的话，用户们就会觉得很困惑，还会觉得被欺骗了，他们就有权谴责你提供了不好的用户体验。

为了改善这个问题，网站开发师们一般会利用**功能衰减**的方法：告诉用户该链接可能无法

使用以及无法使用的原因，可能还会建议一种替代方案来帮助用户达成自己想要做的事情。实现这一点的常用技巧就是利用 `noscript` 元素。在这个元素内部的一切内容都会在 `JavaScript` 不可用的时候显示给终端用户。在我们这个例子中，可以像下面这样：

```
<p id="printthis">
 Print this page
</p>

<noscript>
 <p class="scriptwarning">
 Printing the page requires JavaScript to be enabled.

 Please turn it on in your browser.
 </p>
</noscript>
```

这就是柔性地衰减——我们向用户说明有些地方出问题了，并告知他们如何解决这个问题。然而，这样做的前提是假定你网站的用户都：

- 知道什么是 `JavaScript`
- 知道如何启用 `JavaScript`
- 拥有启用 `JavaScript` 的权利
- 乐意启用 `JavaScript` 以打印文档

下面的方法可能要稍微好些：

```
<p id="printthis">
 Print this page
</p>

<noscript>
 <p class="scriptwarning">
```

```
Print a copy of your confirmation.

Select the "Print" icon in your browser,

or select "Print" from the "File" menu.

</p>

</noscript>
```

这段代码解决了上面提到的部分问题，但它仍有前提假设，就是浏览器的打印功能在所有的浏览器中都是完全一样的。可是事实情况并非如此——这类方法的问题在于，我们提供了一些功能，但同时我们完全知道它可能无法运行，然后我们还得自己对其加以说明。从技术上讲“打印本页”按钮是没必要的，因此对于同一个问题，渐进增强的方法就用不着假定打印功能可以运行。

如果我们要用**渐进增强**来解决这个问题，第一步就是找出是否存在一种不用脚本的打印当前页面的方法。实际上并不存在，这说明了链接不是合适的的 **HTML** 元素。如果你想要提供一种仅当 **JavaScript** 可用时才能运行的功能的话，你就可以使用按钮：从定义上讲，按钮就是用来支持脚本功能的。**W3C** 的规范中说道：

**按钮：**按钮并没有默认的行为。每个按钮都可能有与该元素的事件属性相关联的客户端脚本。当某个事件发生（比如，用户按下该按钮，松开该按钮，等等）的时候，相关联的脚本就会被触发。

第二步就是不要假设用户的 **JavaScript** 是启用的，不能假设他们的浏览器具有打印功能。我们要做的只是坦白地告诉用户他们需要打印该文档，并将“如何打印”这个问题留给用户自己去解决：

```
<p id="printthis">Thank you for your order. Please print this page for your records.</p>
```

这个方法不管在什么情况下都有效。我们可以在浏览器支持 **JavaScript** 的时候用 非干扰性的 **JavaScript** 来添加一个打印按钮：

```
<p id="printthis">Thank you for your order. Please print this page for your records.</p>

<script type="text/javascript">

(function() {
```

```
if(document.getElementById) {

 var pt = document.getElementById('printthis');

 if(pt && typeof window.print === 'function') {

 var but = document.createElement('input');

 but.setAttribute('type', 'button');

 but.setAttribute('value', 'Print this now');

 but.onclick = function() {

 window.print();

 };

 pt.appendChild(but);

 }

}

})();

</script>
```

注意看这个脚本防御性有多强——我们什么前提假设都不需要。

- 通过将整个功能封装在一个匿名函数中，并立即对其进行执行——这就是`(function() {})()` 所做的工作——我们就不会遗留下任何全局变量了。
- 我们进行了 DOM 支持性测试，并尝试获取我们想要添加按钮的那个元素。
- 然后我们对该元素是否存在，以及浏览器是否有 `window` 对象和 `print` 方法进行测试（这一点是通过测试这个属性的 `type` 是否是 `function` 来实现的）。
- 如果两个都为真，我们就会创建一个新的单击按钮，并应用 `window.print()` 作为单击事件处理器。
- 最后一步就是将该按钮添加到该段落中去。

这一招对每个用户都有效，不管他们的应用环境如何。我们并没有向用户承诺提供一个可能无法使用的界面元素——我们只在该元素能正常工作的时候才将它显示出来。

## 什么时候该用什么

可能是我太理想主义了吧，但我实在不喜欢功能衰减这个概念。如果我创建了某些东西，但它在其它环境中只能勉强运行（或需要用户进行升级），我就对环境和用户的升级能力做了太多的假设。

在我的便携式计算机无法获得无线网络服务时，我用的是一款黑莓手机，当那些 Web 产品告知我它们需要启用 **JavaScript** 才能运行，而我应该开启我的 **JavaScript** 时，我郁闷透了。我无法开启 **JavaScript**，但我肯定有资格作你们产品的用户——尤其是在我为了访问你们的服务花了许多钱在 **GPRS** 或 **EDGE** 网络上的时候！

然而，**功能衰减**在某些情况下也是切实可行的：

- 在你想将一个旧产品翻新，而又没有时间、权限或能力来改变或替换它的情况下。
- 如果你没有时间来完全完成一个渐进增强式的产品（通常是由于规划不良，或预算耗尽）的话。
- 如果你的产品是特殊情况的话，比如说那些极高流量的网站，每毫秒的运行都意味着数以百万计的美元的波动。
- 如果你的产品天然的非常依赖脚本，而使得维持“基础”版本比增强版本（地图，email 客户端程序，RSS 阅读程序）更有意义的话。

在其它所有情况下，**渐进增强**都能使终端用户和你自己更满意：

- 不管产品的应用环境和浏览器能力如何，都需要产品可用。
- 当一款新的浏览器推出或某个浏览器的扩展被广泛采用时，你就可以将你的产品功能提升至另一层次，而不用触及原始解决方案——功能衰减却可能会要求你修改原始解决方案。
- 你还技术为其原本该有的样子——技术就是一种辅助措施而已，拥有该技术能更快地达成某个目标而已，而不是说“必须要有它”才能达到某个目标。
- 如果你要添加新的功能，你可以在检查对这些功能的支持是否达到了某个水平之后，再予以实施，或者你可以将其添加到最初级的功能上，从而使你的网站在更高级的条件下能表现得更好。无论如何，这种维护都是在原来的版本上进行的，而不是在两个不同的版本上。使一个渐进增强版的产品保持在最新状态，要比同时维护两个版本的工作量小得多。

## 总结

可以这么说，渐进增强和功能衰减都试图实现同样的目标：使我们的产品对每个用户都可用。渐进增强要更好一些，同时在保证产品的可用性方面要更为稳定一些，但它需要花费更多

的时间和精力。功能衰减使用更容易一些，可以作为已有产品的补丁来使用；它的后期维护要更难一些，但所需的初始工作量相比较少。

## 练习题

- 本文所展示的打印链接示例既可以通过功能衰减的方法来实现，也可以通过渐进增强来实现。还有什么其它的例子也可以使用这两种方法，你能想到吗？
  - 假设你想用 `JavaScript` 来在提交某个表单之前，确保它的某个表单域中包含了一个 `email` 地址。你可以用什么不同的方法来实现？还有什么其它问题应该考虑的？
  - 假设你想显示一幅地图，而且你想用渐进增强的方式来实现。你该从什么样的基础功能开始？
  - 假设你有一个由两个下拉列表控件构成的表单界面。在第一个控件中选择一个选项，就会改变第二个控件中供选择的选项。这种表单功能的备用方案是什么？会有什么问题？
- 
- [上一篇文章—`JavaScript` 动画](#)
  - [目录](#)

## 作者简介



Chris Heilmann 从事 Web 开发工作已达 10 年，之前他曾从事电台新闻工作。他在英国做 Yahoo! 的培训师和高级开发工程师，同时负责监控面向欧洲和亚洲的前端代码质量。

Chris 的博客是 [Wait till I come](#)，在许多社交网络上都可以通过输入“codepo8”找到他。

# 将你的内容发布到网上

Posted 12/15/2008 - 16:59 by Lewis

- 网络标准教程

## 补充材料: 将你的内容发布到网上

作者 CraigGrannell · 2008 年 7 月 8 日

- 目录

### 引言

本文的目的在于为你发布网站提供一份便捷指南。在本文中，你将了解到如何获取域名以及虚拟主机。本文还会告诉你上传网站所需要的软件，还有结构化网站的最优方法，以及如何确保你的网站总有稳妥的备份。本文分成四个部分：

- 对于域名有哪些注意事项?: 如何精选及购买域名;
- 最适合的虚拟主机: 关于网站寄存账号——选择的目标及注意事项;
- 线上发布: 用软件来将你的网站上传到网上;
- 工作进行中: 处理本地文件和远程文件，以及网站结构化的最优方法

### 对于域名有哪些注意事项?

创建网站的时候，域名是一个很重要的组成部分。域名是你面向全世界的文字性的链接——就是人们在浏览器地址栏输入的东西，这样才能访问你的网站（比如 `google.com`, 或 `apple.com`）。一个好的域名又好记又简单，因此要优于那些免费网络空间的 URL，这种 URL 通常都会包括你的网络服务提供商的域名和你的宽带或拨号用户名（比如 `nameofisp.com/~username`）。

选择域名的时候，千万要避免复杂的域名。不妨设想一下你要在电话里把它读出来——如果在这个域名里，你不得不念出“**hyphen**”或“**the numeral two**”这样的词，或是要拼出不方便或者不标准的词语的话，还是另外想一个域名吧。由于已经有数以百万计的域名被别人用掉了，你最好先想好至少半打的替代品。比如，你可以尝试用你的名字来突出你的域名：你不可能买到 `gardening.com` 的域名了，但如果在“`gardening`”后面加上一个“`in`”和你的地址（比如“`gardeninginsomerset.com`”），这样的域名可能就没有人用过了。

为了弄清楚某个域名到底有没有被占用，你可以搜索一下。大多数的域名转销商都有搜索服务，你可以通过这个来搜索带各种后缀名的域名（比如，.com,.net,.org等等——后面还会讲更多关于后缀名的内容），此外 [www.internic.net/whois.html](http://www.internic.net/whois.html) 也是一个出色，全面而且非商业性质的资源。

如果你是在一个商业性的网站上进行搜索的话，如果你的第一个域名选择已经被占用了，它们可能会给你提供其它的备选域名。在这种情况下，如果这些替代品正好是你想要的，那你就可以直接选择其中的一个。不要被稀奇古怪的单词组合或者罕见的后缀名所吸引。人们通常会记得“点 com”或者与他们的所在地区相同意思的域名（比如，在英国就是“.co.uk”）。但人们一般不会记住那些使用不便的词语组合，比如“.uk.com”，或者生疏的后缀名，如“.info”。此外，如果别人的域名后缀更大众化的话，你的网站流量就可能会跑到他那边去。

值得注意的是，即使你是在创建个人站点或博客（就比方说我的网站 [Revert to Saved](#) ——参见图 1），也需要抢注一个好记的域名。比如说，如果你的域名是基于你自己的名字的话，你的朋友和家人就会更容易记住它，而且这样的域名也可以使与之相关的email账号更好用，使你的 email 账号可以像“你的名字@你的域名.com”这样，而不是一串在你的网络服务提供商域名前面的无意义的随机字符组合。此外，域名还有一个好处就是它具有不变性——如果你换了网络服务提供商而“失去”了你的免费网络空间的话，你就只有重新开始了。然而，如果你用的是域名的话，你的网站地址就始终是一样的，这意味着就算你换成了另一种完全不同的虚拟主机服务，你的网站也只需要停机个两三天。

在购买域名的时候，你可以选择从某家转销商那里仅仅只购买一个域名，也可以连虚拟主机也一起购买。对于初学者来说，我强烈推荐将域名和主机服务一起购买。这样你只需要就支持问题跟一家公司打交道就可以了，并且该公司的系统可以清楚地获知你的动向，从而使你能够将新近获得的域名“绑定”到你通过在线管理面板购买的虚拟主机上。如果你不想两个一起购买，也可以从一家公司购买域名，再将它“指向”你在其它地方买的虚拟主机。但是在这种情况下，你就必须更新你的域名的域名服务器和 IP 地址（IP 地址是用来使域名“明白”自己该指向哪个网站的），使之符合你的网站主机的要求。

需要注意的是，有时在域名购买过程中会出现意想不到的困难。有些经销商抬高了价格来获取更多的利润（在现在的市场中这是很不讲理的），有些经销商一开始的收费比较低廉，但在你后来想更换域名的时候，他们还会向你收费。因此，在购买域名之前，一定要先看看能不能让你免费更换。此外，要小心不要被人骗而去买额外的域名，除非你真的需要它们——如果你已经选好了名字，并且后面是一个.com，就不用再买 .biz 或 .info 之类的东西了，这样可以省钱。最后，如果在购买过程中提供有私有注册的话，是值得考虑一下的。在默认情况下，你的

详细信息（姓名，地址，电话号码）在人们查询你的域名所有权时是可以看到的；然而，只要再交一点钱，大多数域名转销商都会提供“隐藏”你的详细信息服务，这样在查询的时候就只会显示出注册商的一般信息。

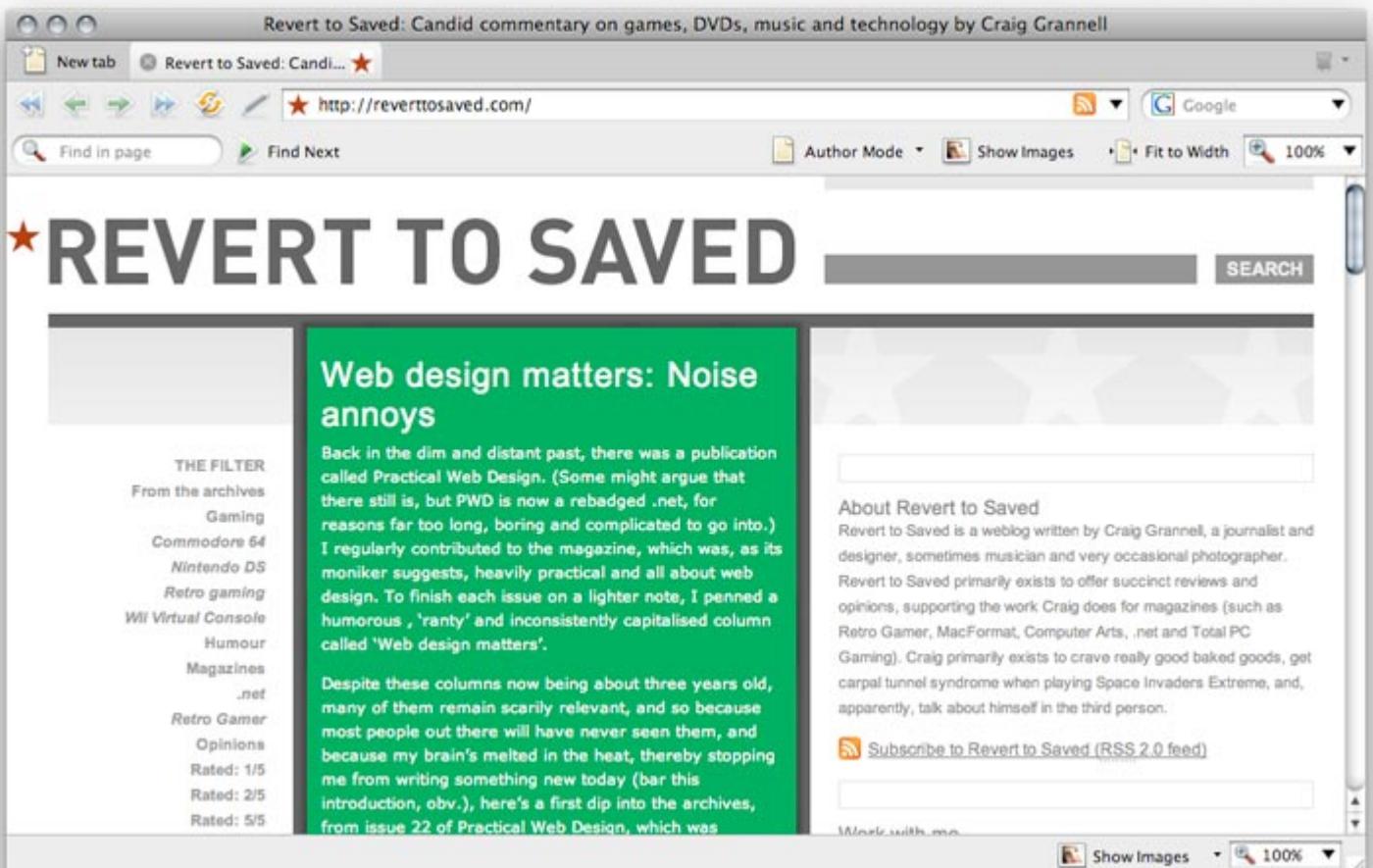


图 1：我的博客的域名，Revert to Saved，就是经过仔细挑选的。这个名字很好记，它是一个字符串，没有用到连字符，而且后缀名是最常见的.com

### 最适合的虚拟主机

如果你已经想好了域名（或者已经购买好了），你就需要把你的网站上传到某个地方去。可能随着你的宽带或拨号账户，你已经有了一个免费网络空间，但是这类的空间几乎总是受到某种程度的限制的，可能会有带宽（就是用户在一个规定的时间段所能下载的数据数量）的限制，存储分配（用来储存你的信息的兆字节或 G 字节的数量）的限制，或者是其它技术（比如对各种脚本或数据库之类的技术的支持）的限制。免费的虚拟主机只提供最低限度的支持，这意味着如果你将来想扩展你的网站规模或功能集的话，就会遇到困难。

幸运的是，技术的进步使虚拟主机服务越来越容易获得，而竞争则压低了价格。便宜的硬

盘驱动器使我们可以在付费的虚拟主机服务中获取大量的空间，甚至可以以非常低廉的价格获得。此外，对虚拟主机包含功能的期望在变迁，许多付费的虚拟主机服务也都提供了对 PHP 和 MySQL 的支持。在所有的情况下，在付费之前要先调查清楚托管你所建立的网站需要什么样的技术；如果你没有把握的话，可以跟你打算购买的虚拟主机服务的提供商的支持部门进行沟通，或者确保万一你的网站的要求改变了的话，他们能够提供一个简单而又让你负担得起的升级路径。

除了上述内容，在上传网站过程中还有更进一步的问题需要解决。你得了解数据传输的流量，还有如果你超出了这个流量的话会发生什么（你的网站可能会被暂时“关闭”，或者你可能得付更多的费用），尽管这只是高流量网站才会关注的事项；你也可能想看看有没有现成项目，比如预装脚本，论坛和联系表单；如果你的规划实际上是不止一个单独网站的话，你还需要了解关于在一个虚拟主机方案上绑定多个域名或网站的限制。同样的，如果你对建立网站不熟悉，最好选择那种你可以在需要的时候跟真人交谈的虚拟主机服务提供商，而不是只能一动不动地等着从半自动帮助系统发来的邮件。

对于虚拟主机服务来说，最好的办法是货比三家。有很多网站，比如 [web-hosting-review.toptenreviews.com](http://web-hosting-review.toptenreviews.com)（参见图 2），都提供关于当下最好的虚拟主机服务的看法和建议，你可以直接联系服务商，向他们提出咨询。如果他们的回应又敏捷又合宜，这就是一个很好的迹象，表面如果以后出现问题的话，你能够得到可靠的帮助。无论如何，不要着急，你没必要跑到最便宜的商家那里去——货比三家，自己下来做做功课，确保你选定的主机适合你的需要。就像之前提到的那样，我们建议新手从同一家公司购买域名和虚拟主机；这样，一般你都能通过在线管理面板来将你的域名“绑定”到你的虚拟主机上，而且方法也很简单，没有什么技术性。

The chart displays a grid of 10 web hosts. Each host has a row of colored squares representing its rating in different categories. The categories include: Reviewer Comments, Lowest Price, Special Offers, Overall Rating, Ratings (Feature Set, Customer Service, Control Panel), Costs (based on basic plan for one year), and Features (Disk Space, Monthly Data Transfer, Server Uptime Guarantee, Sub-Domain Supported, Domain Name Search/Registration). The colors range from light blue (Excellent) to dark blue (Poor).

图2：许多网站都提供关于虚拟主机服务的比较评论，在付钱之前先来这些网站搜索一下是个不错的主意。

### 线上发布

搞定了你的域名和虚拟主机之后，你就可以开始将你的页面发布到网上去了。你的虚拟主机机会提供给你一些信息，这些信息你应当妥善保管，保证其安全。你也可以获取更详细的信息，这些信息会教你如何在该主机上访问你的账号，以及如何访问在线管理功能。你也可以了解关于如何通过 **FTP** 来访问你的站点的详细信息，**FTP** 是指“文件传输协议”。尽管虚拟主机提供的信息各不相同，你一般都会得到用户名，密码，上传文件地址（通常是你自己的 **URL**，不过会因主机而异），可能还会有你的网页的存储文件夹的路径。（注意，尽管你能通过 **FTP** 快速访问自己的空间，有时还要等上三天，整个互联网才能“看”到你的域名，因此如果人们不能在你上传网站之后马上对其进行访问的话，也不必苦恼。）

网络上有无数的 **FTP** 应用工具，比如 **Windows** 版的免费（但是非常出色）的 **CoffeeCup Free FTP**，还有非常优秀的 **Mac OS X** 版的 **Transmit**。有些网页设计工具，比如 **Dreamweaver**，也提供嵌入式的客户端，尽管大多数这样的工具并不具备成为独立服务器的全部功能。**FTP** 应用工具之间的差异非常多，但是大多数在工作流程方面是差不多的。一般地，它们都会提供一些方式来让你收藏你喜欢的地址（其中之一就是你自己的站点），以便进行链接。每收藏一个地址，你都需要用到你的主机提供给你的详细信息，就像上面提到过的那样。

当你链接到你的网络空间的时候，将会看到你网站的空的目录结构，这种结构也是因主机而异的。在有些情况下，你根本什么也看不到。在其它情况下，可能会有若干个默认文件夹，用来存储脚本和访客统计之类的东西。大多数 **FTP** 客户端也会提供一个本地视图（该视图中显示

的是你硬盘上的文件)——具体例子参见 *Transmit* 的屏幕截图(图 3)。要上传文件到你的网站,你只需将其从本地拖到远程存储位置上去,或者在本地文件上点击鼠标,选择一个适当的“上传”选项即可。

如果你在进行脚本处理,它们一般也会提供给你相应的操作指南,告诉你该如何改变某个文件的权限。通常这种操作是通过获取某个文件的信息,然后在相关的复选框中点击来完成的,不过也可能是通过“chmod”选项——“chmod”是一个 Unix 下的命令,用来修改文件和目录模式。大多数 FTP 客户端还提供了更多的功能,包括本地和远程文件夹的对比与同步,以及在访问了某个收藏站点时自动设置某个本地文件夹。同样的,在选择 FTP 应用工具的时候应该货比三家,要记住大多数 FTP 客户端是很便宜(免费)的,而且还有完全可以运行的演示版。

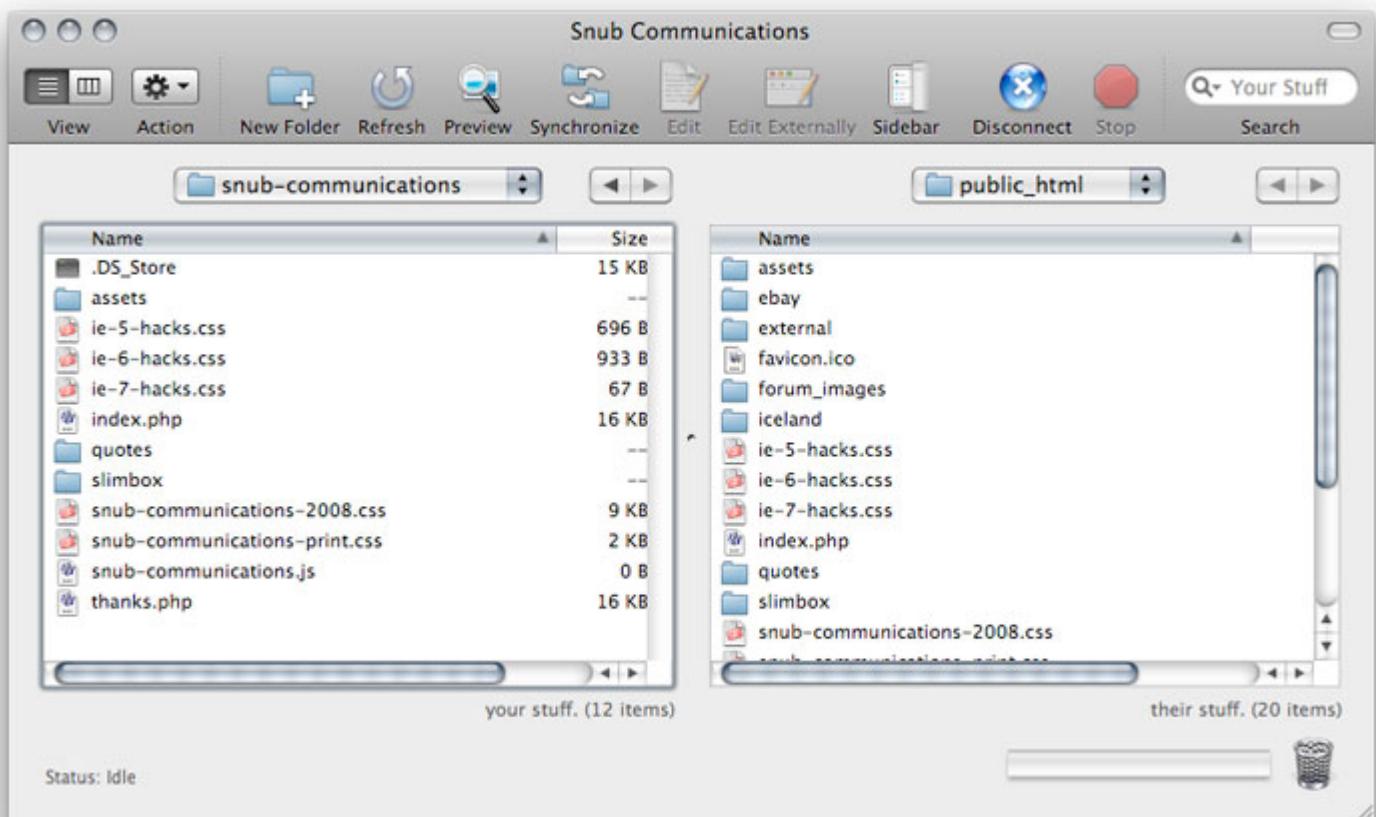


图 3: *Transmit*, 在 Max OS X 下可用, 是一款典型的双面板 FTP 客户端程序, 在左侧的是本地视图, 在右侧的是远程文件视图。

### 工作进行中

在前面的部分中,我们讲述了 FTP 客户端是如何同时显示远程和本地文件的。这真是一件非常好的事情——任何一个杰出的 web 设计员都会告诉你,单单只是在线修改是非常冒险

的。如果你在对一个实况站点进行修改的时候弄错了一处改动，那全世界都只有等到你校正好了才能再次浏览这个网站了，而且假如该站点发生了什么事故（主机虽然进行了备份，但这些备份并不是每次都管用的，或者是不像它们应该的那样管用），而你只是在线修改的话，你就会丢失所有资料。

相反的，你应该为你的文件制作本地副本，准备好之后再将它们上传。通过这种方式，你就可以在上传之前对改动进行测试，以确保这些改动能够运行，并保证文本和图片都经过校对而且是易读的。你也可以在做出大的改动前对一个网站进行备份，以确保万一改动方案完全失败，你也可以转而使用另一个版本。只有当你对自己所做的改动完全满意的时候，你才能将其上传。

在网站结构方面，所有内容如何组织在很大程度上是取决于个人的，但创建一个合理的目录结构是很有益的，这样你就可以将图片，PDF，MP3 和电影之类的东西分别存放在特定的，具有对应名称的文件夹中（见图 4），而不是将所有的东西都塞进网站的根目录下，而这样会显得非常的杂乱无章，而且随着时间的增加，网站内容会变得越来越难以组织和整理。一些 web 设计员还提倡将样式表，JavaScript 文件，甚至网页组都放进对应名字的文件夹中，尽管只有当它们的数量非常大的时候这样做才有必要。随着时间的推移，如果网站规模变大了，你可能就需要在文件夹下再创建子文件夹，以便更好地管理你的媒体。

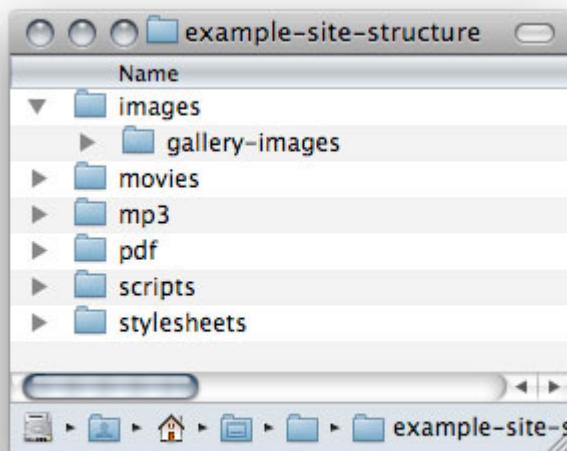


图 4：一个非常典型的网站结构，正在等候上传的内容。

从开发的角度来说，除了确保你的本地和远程文件夹在结构上相同之外，更新并保持你的“测试”和“实况”站点一致也并非不可能。（也要注意，有些主机会要求某种类型的文件放置在特定的文件夹下。关于这一点，最常见的例子就是 CGI 脚本，CGI 脚本通常必须放在 cgi-bin 文件夹下才能运行。此外，一些配置选择——比如数据库配置——也是因机器而异的。同样的，如果

你没有把握的话，可以去咨询你的虚拟主机服务商。)

## 总结

总的说来，对于本概述中所有的内容来讲，慎重和研究是两件最重要的事。不要轻率地开始任何事情，这样你就不会犯那些代价高昂的错误。好好地研究研究（针对域名，虚拟主机，如何对你的网络空间进行最大的利用，以及如何上传和维护你的网站进行研究），在实际面对这些问题的时候就能得心应手。

## • 目录

### 作者简介



**Craig Grannell** 一开始学的是美术，十多年之前他义无反顾地一头扎进了数字媒体的世界，而且从未后悔过。除了为大量客户设计网站之外，他还为若干面向设计的刊物撰写文章，还编写了一些关于 web 设计的书籍，同时在音乐和摄影方面依然富有创造力。在网页设计方面，**Craig** 是 web 标准和简单设计的忠实拥护者。

你可以在 [Snub Communications](#) 上找到更多关于 **Craig** 的设计和著作的内容。**Craig** 还会定期写写他的博客 [Revert to Saved](#)，偶尔有功夫了还会在 [Project Nois](#) 上搞点儿音乐。

# 有关文档中 <head> 元素的更多知识

Posted 12/15/2008 - 17:00 by Lewis

- 网络标准教程

## 补充材料：有关文档中 <head> 元素的更多知识

作者 Christian Heilmann · 2008 年 7 月 8 日

- 目录

### 引言

在本教程的 第 13 篇文章中你了解到了在 HTML 文档的 head 部分中有哪些重要的内容。在这篇教程里，我会更深入地探讨这一部分的知识，还会讲到一些其他的——较少用到的——内容，你可以将这些内容添加到 HTML 文档的 head 部分；这些东西的重要性虽然没有那么高，但是还是很有用的。学完了这篇教程，你就会知道如何将几份 HTML 文档组合成一个更大的文件集，什么是收藏夹图标，怎样使用它，以及关于 RSS 的一切。在你开始学习之前，先 点击此处 下载本文附带的压缩文件，这样你就能照着例子来学习了。本文目录如下：

- 文档关系——将若干 HTML 文档组合成一个集合
- 链接到文档的备选版本
  - 译文
  - 信息源
- 让收藏夹变得更有趣——收藏夹图标的使用

- 总结
- 练习题

## 文档关系——将若干 HTML 文档组合成一个集合

Web 是作为文档的储存库而产生的，由此而来的 HTML 的一个特征就是文档关系。文档关系定义了一份文档与另一份文档之间是如何关联的，举例来说就是，某文档是否是其他文档的上一个或下一个文档，或者是否是整个系列文档的索引。

从某种意义上来说，在 第 13 篇文章中你已经学过这部分的内容了，就在你通过 link 元素将一份样式表应用到某个文档中，来改变该文档的外观和感觉的时候：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

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

<title>Breeding Dogs – Tips about Alsatians</title>

<link rel="stylesheet" type="text/css" media="screen" href="styles.css">

<link rel="stylesheet" type="text/css" media="print" href="printstyles.css">

</head>

<body>

</body>

</html>
```

这份文档与其它文档的关系是通过 link 元素和 rel 或 rev 属性来定义的。rel 属性（关系）定义了被链接文档到当前文档的关系，rev 属性（逆关系）则定义的是当前文档到被链接文档的关系。

不必过于担心 rev 属性——虽然让人有点晕，但你很少会用到它。

对于 rel 和 rev 属性来说，并没有强制规定要用哪些属性值，但却有浏览器和索引工具所支持的分类，因此在大多数情况下，使用 rel 时你都应该考虑到下面这些分类：

**home**

当前集合的主文档

**index**

当前集合的索引

**contents**

当前集合的目录列表

**search**

当前集合的搜索页面

**glossary**

当前集合的词汇表

**help**

当前集合的帮助页面

**first**

当前集合的第一篇文档

**previous**

当前文档在当前集合逻辑顺序上的前一篇文章

**next**

当前文档在当前集合逻辑顺序上的后一篇文章

**last**

当前集合的最后一篇文档

**up**

当前集合文件层级中的上一级文档

**copyright**

当前集合的版权信息

**author**

当前集合的作者信息页面

大多数浏览器不会对文档关系信息作任何处理。然而有的浏览器会跟踪链接并在后台中加载文档，这样读者看起来就会感觉页面载入更快了。在这方面真正例外的浏览器是 **Opera**，它有一个额外的导航栏，你可以通过在菜单中选择视图> 工具栏> 导航栏 来打开该导航栏。打开之后你就会在一个额外工具栏中看到文档中定义的链接关系。图 1 显示了 **Opera** 中的 W3C 的 **HTML** 标准文档。

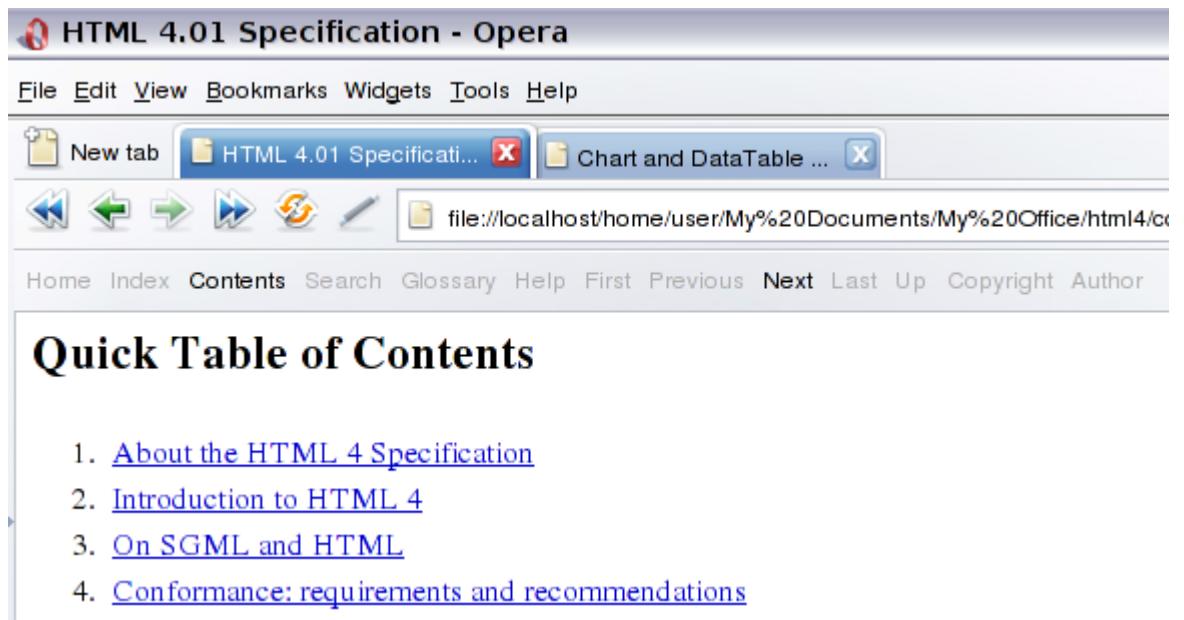


图 1：Opera 在一个特别的导航栏中显示了当前文档的链接关系。

即使文档关系信息不会在视觉意义上显示出来，但在 title 属性中提供一些可读的说明，来解释所链接到的文档是什么，也不失为一个好办法，因为仅仅只有文件名是不够的

下面我们来看看如何用链接关系将几份文档组合成一个集合。我们举个例子，某个在线教程的首页 `start.html` 可能涵盖了下面几份文档：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

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

<title>Link relationship example</title>

<link rel="contents" title="table of contents" href="toc.html">

<link rel="next" title="next: chapter one" href="chapter1.html">

</head>

<body>
```

```
<h1>Course example</h1>

<p>This would be the cover page of an article series or course</p>

 Let's start with Chapter One

</body>

</html>
```

### 第一章课程如下(chapter1.html):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

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

 <title>Chapter One - Link relationship example</title>

 <link rel="contents" title="Table of Contents" href="toc.html">

 <link rel="home" title="Home Page" href="start.html">

 <link rel="prev" title="previous: Home Page" href="start.html">

 <link rel="next" title="next: Second Chapter" href="chapter2.html">

</head>

<body>

 <h1>Chapter One</h1>

 <p>This would be the chapter one page of an article series or course</p>


```

```
Back to Start

Table of contents

Go on to Chapter Two

</body>

</html>
```

## 第二章 (chapter2.html):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

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

<title>Link relationship example</title>

<link rel="contents" title="Table of Contents" href="toc.html">

<link rel="home" title="Home page" href="start.html">

<link rel="prev" title="previous: first chapter" href="chapter1.html">

</head>

<body>

<h1>Chapter Two</h1>

<p>This would be the second chapter page of an article series or course</p>

Back to chapter 1

Table of contents
```

```


</body>

</html>
```

### 最后是目录(toc.html):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

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

<title>Table of contents – Link relationship example</title>

<link rel="home" title="home page" href="start.html">

</head>

<body>

<h1>Table of contents</h1>

Chapter One – about stuff

Chapter Two – about other stuff

Back to home

</body>

</html>
```

你也可以在该文档的链接中用 `rel` 和 `rev` 属性来告诉浏览器和辅助技术，这些锚跟链接关系有关。

你也可以用 `rel` 和 `rev` 来作其它用途，比如微格式——[点击这里查看 XFN Microformat 的一些用法。](#)

### 连接到某文档的备选版本

链接到与当前文档有某种关系的其它文档，当然也包括了链接到该文档的不同语言的版本，或者不同版式的版本。为了实现这两种链接，你可以在一个链接中加入一个值为 `alternate` 的 `rel` 属性来指示一个备选版本。

### 译文

译文是使用文档互连很重要的对象。比如说，可能会出现这种情况：一份文档的某种语言的版本非常受欢迎，那些不懂这种语言的访客希望自己也能获取该文档的信息。通过从源文档链接到备选语言版的文档，就能使另一种语言的读者更容易地理解并进而宣传该文档的内容，这样就有可能使备选语言版的文档同样受欢迎。下面的例子告诉了我们该如何定义另一种语言的版本 (`languageexample.html`)；注意语法——这是非常容易理解的：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 1//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>
 <head>
 <meta http-equiv="content-type" content="text/html; charset=utf-8">
 <title>Multiple Languages example</title>
 <link rel="contents" title="table of contents" href="toc.html">
 <link rel="next" title="next: chapter one" href="chapter1.html">
 <link rel="alternate" title="The course in Dutch" type="text/html" hreflang="nl" href="../nl/start.html">
 <link rel="alternate" title="The course in German" type="text/html" hreflang="de" href="../de/start.html">
 </head>
 <body>
```

```
<h1>Course example</h1>

<p>This would be the cover page of an article series or course</p>

 Let's start with Chapter One

 Other languages:

 Deutsch

 Nederlands

</body>

</html>
```

如果要提供整个网站的国际版本，比起上面这份文档来，还需要进行多得多的研究，我们希望在本系列课程的后面还会推出专门针对这个问题的教程。你可能已经注意到 hreflang 和 lang 属性是自己从来没见过的。链接和锚上的 hreflang 属性说明了被链接文件的语言，而 lang 属性则说明了该属性所属元素内的文本的语言。这对可访问性来说是非常重要的，因为语音合成软件需要在各种语言之间转换发音。

从互联网产生的时候起，语言差异就是一个突出的问题（其实在网络产生之前几千年就已经是这样了），此外还有另一种类型的替代网页，在你爬网的时候应该见到过很多——信息源（比如 RSS 信息源）。信息源是非常流行的，尤其是对那些经常改变的文档，比如新网站来说更是如此。下面我们就来谈谈这个问题。

## 信息源(Feeds)

一个信息源就是一份文档，包含了一些浓缩的信息，按照年月日的顺序详细列出了你的站

点的最新更新。用户可以订阅信息源，从而了解你的网站最近发生了什么变化，而不必访问你的站点。他们可以通过信息源阅读器之类的工具，比如 **Google Reader**, **Netvibes** 或 **Bloglines**，来订阅信息源。一些最新的浏览器（比如 **Opera**）和 e-mail 客户端（比如 **Mac Mail**，或 **Windows** 的 **Outlook**）也可以处理并显示信息源。你可以通过网站地址旁边的 RSS 图标来识别某个网站是否提供了信息源，如图 2 所示：

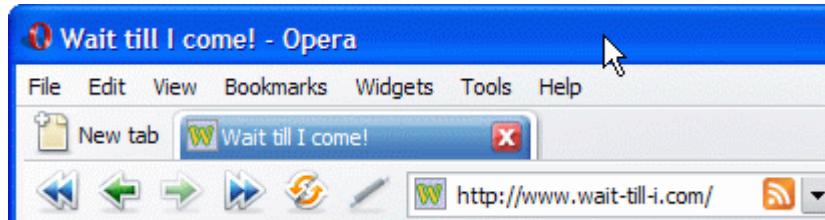


图 2： Opera 显示了地址旁边的橙色 RSS 图标，表示该网站提供了信息源。

信息源页面是使用 **HTML** 或某种 **XML** 格式，如 **RSS** 或 **Atom**，它们很少是手动生成的。大多数时候个人发布系统会替你完成这项工作，而你只需要在你的文档开头部分用正确的 **meta** 元素链接到 **XML** 文件，就可以向全世界提供你网站的信息源了。下面代码是从我的博客 <http://wait-till-i.com> 上摘录下来的，并且指向 RSS 信息源 (**feedexample.html**)：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

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

<link rel="alternate" type="application/rss+xml" title="Wait till I come! RSS Feed"
href="http://www.wait-till-i.com/feed/">

<title>Wait till I come!</title>

</head>

<body>

</body>

</html>
```

对于那些经常更新的具有大量内容的网站(如博客网站和网络相册)来说，提供一份信息源是非常明智的，通过信息源阅读工具以及订阅信息源，你就可以节省大量的网上冲浪和查看的时间。

如果你的网站更新并不频繁，但因为你的内容很多，希望用形象提示来提醒人们注意你的网站的话，你可能会考虑在人们的收藏夹列表中使用快捷图标来吸引他们的注意力。这就是我接下来要讨论的问题。

### 让收藏夹变得更有趣——收藏夹图标的使用

我要谈的最后一个话题就是快捷图标，或者说收藏夹图标。这些图标是一些小图片，文件格式为.ico——如果你放一个在你的 web 服务器上的话，就可以用它来在用户的收藏夹列表中你网站的条目旁边显示出一个小图标，如图 3 所示：

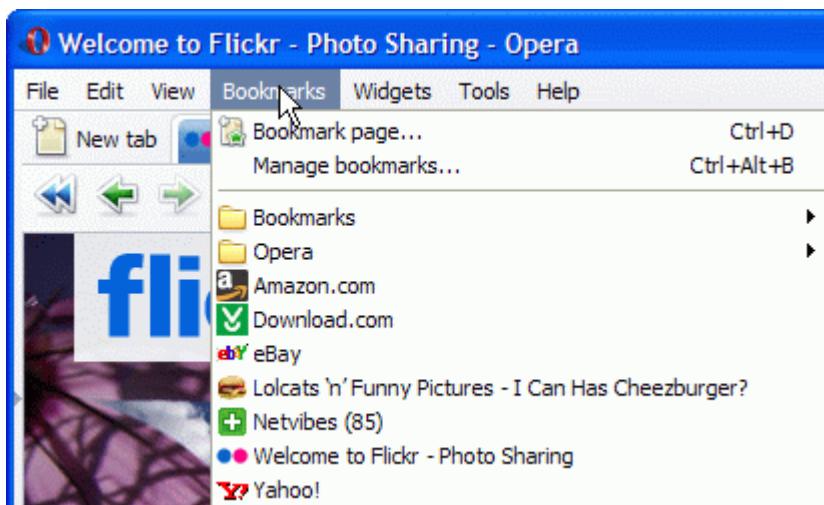


图 3：书签旁边的图标方便了人们记住网站。你可以通过使用快捷图标 meta 元素来在访客的收藏夹里添加这样一个图标。

由于支持 ico 格式的图像制作工具不多，因此添加快捷图标的最大障碍就是按照正确的格式来创建图标。有一个解决办法是用免费的在线工具 [genfavicon..](#)。一旦图标做好了，将它添加到你的文档就像添加其它 meta 元素一样简单，只需要将 rel 值设为“Shortcut Icon”就可以了，如下面例子 ([favicon-example.html](#)) 所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>
```

```
<head>
```

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

<title>Shortcut Icon example</title>

<link rel="Shortcut Icon" href="favicon.ico" type="image/x-icon">

</head>

<body>

<h1>Example of a shortcut icon</h1>

</body>

</html>
```

如果你用浏览器打开这个文档，你就会在地址栏中的地址旁看到 **Opera** 图标。如果你将这个网页添加到收藏夹，同样的图标就会出现在书签旁。

## 总结

本文的全部内容到此结束，同时关于 **HTML** 文档的 `head` 部分的内容也到此结束了。其实我还有其它东西要讲，但是它们是很高级的应用，而且通常不是什么好办法——这里我所讲的内容，以及在 [第 13 篇文章](#) 中讲的内容，应该已经提供了所有进一步学习所需的信息。在本文中我谈到了：

- 在 `link` 元素中用 `rel` 和 `rev` 属性来定义文档关系
- 链接到同一份文档的备选版本，比如译文或信息源
- 在文档中添加快捷图标，以便显示在收藏夹和浏览器标签中

## 练习题

- 为什么定义链接关系很有意义，即使它们并不会显示出来？
- 你该如何链接到一个搜索页面？
- 向你的访客提供信息源有什么用处？你用什么 `rel` 值来链接到信息源？
- 如果你要链接到另一种语言的文档的话，哪些地方是需要弄清楚的？
- 如果你将示例文档在文本编辑器中打开的话，将会发现另一种我们没有讨论过的 `meta` 元素，其属性值为 `content-type`，还有叫做 `utf-8` 的东西。什么是 `utf-8` ？

- [目录](#)

## 作者简介



Chris Heilmann 从事 Web 开发工作已达 10 年，之前他曾从事电台新闻工作。他在英国做 Yahoo! 的培训师和高级开发工程师，同时负责监控面向欧洲和亚洲的前端代码质量。

Chris 的博客是 [Wait till I come](#)，在许多社交网络上都可以通过输入“codepo8”找到他。

## 补充材料：用于排版的通用 HTML 实体

Posted 12/15/2008 - 17:00 by Lewis

- 网络标准教程

## 补充材料：用于排版的通用 HTML 实体

作者 Ben Henick · 2008 年 12 月 26 日

- 目录

### 引言

每当需要一流的排版的时候，有许多 HTML 实体都派得上用场。表 1 列出的实体中有许多只对外国语言的文本（以及用特定英语方言书写的文本）有用，因此在决定使用这些实体之前应该先考虑一下语境。

考虑到可移植性， Unicode 的实体引用应当专供 UTF-8 或 UTF-16 字符集编码写成的文档使用。在其它所有情况下，应该使用字符形式。

符号	字符	字符形式值	Unicode 值	推荐使用
美分(货币)	¢	&cent;	&#162;	
Pound (currency)	£	&pound;	&#163;	
章节号 <sup>1</sup>	§	&sect;	&#167;	
版权符号	©	&copy;	&#169;	(c)
书名号 <sup>2</sup>	« »	&laquo; &raquo;	&#171; &#187;	&quot;
注册商标	®	&reg;	&#174;	(R)
度	°	&deg;	&#176;	
加/减号	±	&plusmn;	&#177;	+/-

符号	字符	字符形式值	Unicode 值	推荐使用
段落标记符号 <sup>3</sup>	¶	&para;	&#182;	
中间点 <sup>4</sup>	.	&middot;	&#183;	
分数 1/2 <sup>5</sup>	½	&frac12;	&#188;	1/2
短破折号 <sup>6, 7</sup>	—	&ndash;	&#8211;	– 用来表示一个范围
长破折号 <sup>7, 8</sup>	—	&mdash;	&#8212;	– 前后各有一个空格，或 --
单引号 <sup>9, 10</sup>	‘’	&lsquo; &rsquo;	&#8216; &#8217;	’ 或 &apos;
单低引号 <sup>11</sup>	,	&sbquo;	&#8218;	，或逗号
双引号 <sup>9</sup>	“”	&ldquo; &rdquo;	&#8220; &#8221;	”， &quot;， ’， 或 ``
双低引号 <sup>11</sup>	„	&bdquo;	&#8222;	&quot; 或 ，
单&双短剑号	†‡	&dagger; &Dagger;	&#8224; &#8225;	* 和 **
项目符号	•	&bull;	&#8226;	*
省略号 <sup>12</sup>	...	&hellip;	&#8230;	...
分钟&秒钟符号 <sup>13</sup>	‘’	&prime; &Prime;	&#8242; &#8243;	’， ’， &apos;， &quot;， 分:秒
欧元符号	€	&euro;	&#8364;	
商标符号	™	&trade;	&#8482;	(tm)
约等于号	≈	&asymp;	&#8776;	~
不等于号	≠	&ne;	&#8800;	!=
小于等于/大于等于号	≤ ≥	&le; &ge;	&#8804; &#8805;	<= or >=

符号	字符	字符形式值	Unicode 值	推荐使用
小于/大于号	< >	&lt; &gt;	&#062; &#060;	

**表 1:** 对正确的排版很有帮助的 HTML 实体，按照十进制 Unicode 编码数值顺序排列

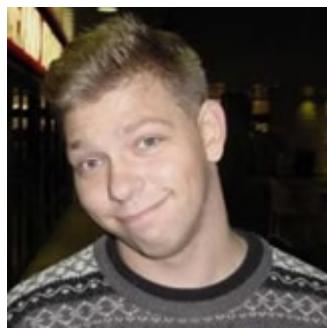
注意书名号在某些欧洲语言（比如法语和挪威语）中是用作援引，在这些情况下，你应当用 `q` 元素来代替书名号。

### HTML 实体使用注释

1. 对成文法的引用，如“29 USC § 794 (d)”，是最有可能用到这个字符的。
2. 书名号中通常包括了故事，歌曲，电影，公共设施（如，《Rick's Café Americain》）的名称，以及欧洲语言，特别是拉丁语系的分支中的常见地名。书名号在某些欧洲语言（比如法语和挪威语）中也用来作引用符号；在这些情况下，你应当使用 `q` 元素来代替书名号。
3. 段落标记符号是用来标记段首的，以避免出现模糊不清的现象，但在为摘要文本排字的时候也非常有用。`Rolling Stone` 杂志的印刷就经常会用到这种方法。在技术性写作中，也可以用于将首行独立成段。标记了¶符号的段落的 `display` 值通常都会被赋为 `inline`，具体说明将放在 CSS 布局模式的介绍中。
4. 中间点是一种类似小数点的过时的符号，现在仍有一些设计者用它来对转换成十进制的货币的数值进行计数。
5. HTML 也提供了对四分之一和四分之三这一类分数的引用。
6. 短破折号是用在两个数量或日期之间的，以表示这是一个区间，而且容易与减号 (&minus;/&#8722;) 相混淆。然而，短破折号与连字符&#45;) 应该要加以区分，连字符是用在某个特定的复合词中间，用来将各个部分分开的。
7. 浏览器会在连字符（参上）后面生成软换行符，但不会在短破折号或长破折号之后生成换行符。
8. 长破折号在英语中有着专门的用途，用来代替圆括号对标记从句的一端或两端进行标记，并表明如果读出声的话该从句的前后应该是不变音的停顿。在其它一些语言中——尤其是斯拉夫语系的分支语言——长破折号表示从某个段落开始就是对话。传统上来说，该字符的前后是不应该有空格的，但考虑周到的使用者们却可能为了避免参差不齐而添加空格。
9. 双引号是自动化的“灵活引用”字符集的成员，这种字符集已经并入了最流行的文字处理平台。它们通常是按照供应商特定的编码数值来编码，而不是像 `Unicode` 或 `ISO` 这样的拉丁文编码数值，这样在拷到 `Web` 文档的时候就可能会出问题。

10. 单后引号在英语中也用作撇号。
11. 在一些中欧和东欧语言中低引号比起英语中类似的开放性引用字符来更经常出现。
12. 由于省略号是一个单独的字符，它里面的那几个点之间的距离不会被 `letter-spacing` 或 `text-align` 属性的任何值所影响。
13. 分钟符号用来表示分 (时间或者角度) 或英尺；秒钟符号用来表示秒或者英寸。最近几年这些符号用于时间的情况越来越少，这主要是由于文字处理软件的流行 (而且使用这些软件的通常不是专业排版人士)。许多字体中分钟符号和秒钟符号与单双后引号相同，但为了可移植性起见，需要的时候最好还是应该使用这些实体符号，而不要考虑的字体显示效果如何。

## 关于作者



**Ben Henick** 从 1995 年 9 月就开始以各种身份参与创建网站了，那时他以一个学术志愿者的身份进行了他的第一个 Web 项目。从那时开始，他的大多数工作都是在自由职业的基础上完成的。

Ben 是一个多面手。他的技能触及了网站设计和开发的几乎每一个方面，从 CSS 到 HTML，再到设计和文案撰写，再到 PHP/MySQL 和 JavaScript/Ajax。

他生活在堪萨斯州的 Lawrence，有三台电脑，却没有电视机。你可以在 [henick.net](http://henick.net) 上读到更多关于他和他工作的内容。

## Opera 公司 Web 标准课程词汇表

Posted 12/15/2008 - 17:00 by Lewis

- 网络标准教程

## Opera 公司 Web 标准课程词汇表

- 目录

## 引言

在本文档中我们列出了大量关于 web 设计和 web 开发的最常见/重要的术语，以及这些术语的定义，如果你在学习 Opera Web 标准课程的过程中碰到不确定的术语的话，可以在这里进行查找。如果你在本文件中发现了什么错误，或者你认为需要添加什么术语的话，请告知我们。

## A

**absolute unit** —— 绝对单位

在 **css** 中包括 **in, cm, mm, pt, pc**

**alignment** —— 对齐

重要概念：**left, right, centred, justified**

**ascender** —— 上伸部分

某个 **glyph**，比如“d”或“h”，的符干部分，它们在一个字形的大写顶线附近结束。

## B

**baseline** —— 基线

绝大多数最常用的字符的尾端都在这条线上结束，除了那些带有下伸部分的字符以外。

整体说来，基线就是终止线。

**Blackletter typeface** —— Blackletter 字样

据我所知这种字样在 **CSS** 中没有通用字体

**blowout** —— 撑破

撑破是指一种事件，由于宽度和/或高度计算错误而导致布局的显示完全异常。如果是由主流浏览器渲染引擎漏洞导致的话，撑破将成为一种不可避免 [测试用例](#)。

**bold** —— 粗体

**brute force** —— 暴力

取自 [黑客字典](#)中的第二种意义：“早期的程序设计风格下，程序员多依赖于计算机的处理能力，而不是自己开动脑筋来简化问题，他们常常忽略了规模的问题，把那些只适用于解决小问题的天真方法直接用来解决大规模的问题。这个术语也可以用在程序设计风格上：暴力程序的编写风格笨拙而单调，充满了重复，并且缺乏优雅或实用的抽象化……”虽然仅仅只需要对Web标记做语法分析和解读，渲染引擎的漏洞以及缺乏培训的设计员在数不清的样式表中大大容忍了暴力程序的存在，每份样式表中都充满了重复的属性/

值组，无法概括成缩写形式，或更有条理的规则代码块。

## C

**cap line** —— 大写顶线

大写字母正好碰到大写顶线，一些小写字母的上伸部分则会稍微超出大写顶线一点。

**canvas** —— 画布

浏览器界面的一部分，网页的实际显示就是在这一部分中。画布是由 `html` 或 `body` 元素表述的，具体由哪一种元素来代表取决于实际使用的浏览器、`!DOCTYPE` 声明和 `Content-Type` 型而定。注意不要和 `HTML 5` 的 `canvas` 元素相混淆。

**centred** —— 居中对齐

**character** —— 字符

是指一个字形，或在一种字体中通常一起出现的一组字形。

**character encoding** —— 字符编码

**character set** —— 字符集

字符集是一整套字符，可以被列入供电子文档使用的某种字体中，在这种字体中每个字符都有唯一的编码数值。电子文档按照某种特定的字符集编码，在客户端主机上，如果首选字体不能与源字符集完全兼容的话，主机会在显示之前自动将该文档按照另一种字符集重新编码。这种情况可能会导致浏览者在浏览的时候出现乱码，这在创建国际性质的网站时是一个要重点考虑的问题。

**code position** —— 编码数值

一个字符在其字符集中的位元编码的十进制或十六进制转换。利用HTML实体进行开发的设计员会接触到这类编码数值。

**copy**

一种广义的术语，用来描述作者发布的文章。

**container element** —— 容器元素

用来确定当前文件流和/或放置上下文的 `%block` 元素。

## D

**descender** —— 下伸部分

像“p”和“y”之类的字母的符干或字脚，这些部分会超出基线之下。

## E

**edge case** —— 边缘情况

在一个网站布局中，受到少数元素或文档限制的设计要求。

**em**

**基础概念：**用于排版的单位，其定义为当前字体的点数值，传统上 **1em** 就等于大写的**M**字母的宽度，但在最新的字样中通常没有遵守这个值。

## en

**基础概念：**用于排版的单位，其定义为当前字体高度的一半，通常也等于字母**n**的宽度。

## F

**flush** —— 排齐

在网站布局中，是指两个元素之间没有**空白**，或指内容和其包含元素的边缘之间没有空隙。

**flush left** —— 左排齐

**flush right** —— 右排齐

**font** —— 字体

某种特定字样中，特定**样式**和/或粗细（例如，斜体）的所有**字符**的集合。在**CSS**中，字体是通过 **font-style** 和 **font-weight** 属性来设置的。

**functional notation** —— 函数符号

**CSS**中函数符号是用来表示颜色、属性和**URL**值的。其格式为函数名（参数值），如e.g.

**url(<http://eg.com>)**

## G

**generic font family** —— 通用字形体系

**glyph** —— 字形

特定**字体**的单个字母，数字，标记，符号，或连字；通常但不总是**字符**的同义词。

**grid** —— 网格

用来确保布局中的每个元素都显示在各自的坐标位置上的一种排字方法，通过使用栅格可以使元素的显示位置变得可预测。有效地使用栅格可以使布局显得更协调。

**gutter** —— 缝隙

两个相邻边距之间的**空白**，通常是由两端**对齐**，加边框，或**放置线条**造成的。

## H

**hanging punctuation** —— 悬挂标点

**hyphenation** —— 连字符

**重要概念：**连字符是用来在行尾处将单词分开以避免空格的。

## I

**italic** —— 斜体

一种**字体名称**，这种字体将书法效果与某种**字样的**标准设计效果相混合。

## J

**justification** —— 对齐

一种将多行文本精确地对齐到其一端或两端的共有边缘的做法。在两端对齐的情况下，通常会伴有字间距的增加，增加的数量视每行的单词数而定。**Web** 正文的对齐是由 `text-align` CSS 属性控制的

**justified** —— 两端对齐的

是在这里解释两端对齐，还是合并到上面的内容中去？

## K

**kerning** —— 字距微调

字间距/字距和字距微调之间的差别是很重要的

**keyword value** —— 关键字值

**KISS Principle** —— KISS 准则

**KISS** 是“*Keep It Simple, Stupid.*”（保持简约）的缩写。工程学的一个最基本的公理就是，一个系统中的组成部分越多，或者各组成部分之间的交互越多，要设计的故障模式也就越多。**KISS** 准则的思想就是，通过减少组成部分或交互的数量，设计员自然就可以减少故障模式的数量。在 **CSS** 和 **HTML** 中，组成部分就是各个元素和与文档树的深度有关的交互——因此通过减少元素和与布局相关的样式表规则的数量，开发团队就可以降低该布局失效的可能性，否则将来就只好全部重新设计。

## L

**leading** —— 行间距

相邻两行文本之间的空白，之所叫做“**leading**”是因为在胶版印刷的时代，空白的数量是通过在文本行之间插入铅(**lead**)条来控制的。在**CSS**中，这个元素是由 `line-height` 属性来控制的。

**letter-spacing** —— 字间距

又叫字距(**tracking**)

**ligature** —— 连字

**lining figures** —— 齐线数字

除 **georgia** 之外所有核心字体都支持齐线数字。

**lorem ipsum**

关键词：希腊语文本，占位符。

## M

**mean line** —— 中间线

**重要概念：**沿着无上伸部分的字母上端的一种假想线条。

**monospaced typeface** —— 等宽字样

**注：**其通用字形体系是 `monospace`

## N

**non-negative number value** —— 非负数值

**number value** —— 数值

## O

**oblique**

一种 [字体名称](#)，该字体将某种 [字样的](#)标准字体的笔画轻微地向顺时针方向倾斜。

**oldstyle figures** —— 旧式风格数字

核心字体中只有 **Georgia** 才支持旧式风格数字（但不是齐线数字）

**ornamental typeface** —— ornamental 字样

其通用字形体系是 `fantasy`

**orphan** —— 孤行

## P

**pagination** —— 分页

对说明 **CSS** 打印规范/媒体类型等等非常有用。

**pica**

基础概念：现在计算机上 **1pica** 等于 **4.233mm** 或 **0.166 英寸**

**point** —— 点数

基础概念：现在的点数大小是 **1 英寸 72 点数**，或 **1pica 12 点数**。

## Q

## R

**ragging** —— 不齐

与 [对齐](#)相反，它保持词间距不变，从而使具有共同边缘的文本行的长度各不相同。

**Recommendation** —— 推荐标准

万维网联盟（W3C）对 [web](#) 标准的称呼。由于 W3C 并没有对任何一个推荐进行认证，而且也没有相应的法规或程序来对违反推荐的组织施以惩罚，并且 W3C 的文献资料上提出的规定也不具备成文法那样的法律效力，因此 W3C 只将“标准”这个词酌情用在称呼其它组织制定的规范上。

**relative unit** —— 相对单位

在 **CSS** 中包括 `em`, `ex`, `px`

**rendering engine** —— 渲染引擎

浏览器实际上运行着许多一般 web 用户无法看见的功能：网络处理，加密，用户交互，图像解码，以及与某个客户端主机的文件系统进行交互（在其它许多种交互之中）。渲染引擎是浏览器代码库的一部分，用于将开发员的 HTML 和 CSS 转化为页面布局。渲染引擎又叫做布局引擎。

**river** —— 隔空白道

**重要概念：**段落中由空格形成的垂直带

**rule**

也指一个布局中的任意长度和宽度的线条的总称。

**S**

**sans serif typefaces** sans serif —— 字样

**script typefaces** script —— 字样

其通用字形体系为 **cursive**

**serif**

这种字体在笔画的一端或两端都有额外的装饰，这种装饰通常是在一行文本的 基线和 大写顶线附近； **serif** 也是所有包含这种特征的字样（与 **sans-serif** 相反）的总类。

**serif typeface**

注：已在上面解释过了。

**small caps** —— 小型大写字母

**stem** —— 符干

**string value** —— 字符串值

**subscript** —— 下标

**superscript** —— 上标

**symbol typefaces** symbol —— 字样

据我所知在 **CSS** 中没有该字样的通用字形体系——极有可能是因为不同的 **symbol** 字体中每个字符的象征符号都不同，使得回落失去了意义。

**T**

**test case** —— 测试用例

事先由开发团队制定出的网站的可用或不可用情景。为了确定网站的外观品质保证，这样的情景会被故意地引发，以便确保在一般情况下以及可预见的情况下该网站都能够按照预期来工作。

## **typeface** —— 字样

字样是一个 [字体族](#)，在该字体族中的所有字体都具有严格的设计共同点。[笔画](#)是一个字形的一部分，和至少一个字符或字符的一部分相等。[字体](#)是全套具有相同粗细和/或样式字符集合，而一套具有严格的设计共同点的字体的集合就是一个字样。

**U**

**V**

**W**

## **weight**

(i): **weight** 可指诸如样式表规则之类的指令的重要性

(ii) 此外，**weight** 也指线条，边框和字母的粗细（也就是宽度，如果你喜欢这么说的话）。加“粗”。

## **whitespace** —— 空白

在布局中所有未被正文，标题，图表，边框或线条占据的空间。

## **widow** —— 寡行

**X**

## **x-heightx** —— 高度

小写的“x”的高度，在同一 [字体](#)中大多数小写字母的高度也与x的高度相等。x的高度在[CSS](#)中是用ex单位来引用的。由于这个定义不是完全准确（有点），因此需要重新解释一下。x高度的定义是中间线到基线的距离。这个距离在大多数，但不是全部的字体中也是x，u，v和z的高度。其它的字母不是有上伸部分，就是有下伸部分，或为了美观而超过了x的高度（顶部是弧线的字母，比如c，o，等等）。

**Y**

**Z**

- [目录](#)