

提示 Geneva 字体栈通常会包含 Verdana，将其作为第三个选项，以应对既没有 Geneva 也没有 Tahoma 的少量操作系统。这里没有包含该字体，主要是为了演示字体栈可以包含不同数量的字体，不过在接下来的例子中，会将它放进字体栈里。

提示 可以在同一个 font-family 规则中为不同的字母表指定字体（如日语和英语），从而对包含不同语言和书写体系的文本进行格式化。

10.4 创建斜体

在传统出版业中，斜体通常用来表示引述、强调的文本、外文单词（如 *de rigueur*）、学名（如 *Homo sapiens*）、电影片名等。

浏览器通常让一些 HTML 元素（如 cite、em 和 i）默认以斜体显示，因此，不必在 CSS 中对这些元素设置斜体。有时你需要让一些内容以斜体显示，但上述元素从语义上来说都不合适。使用 CSS 的 font-style 属性可以让任何元素中的文本以斜体显示。

作为示例，图 10.4.1 说明了如何对段落文本添加斜体样式。（由于这样显示阅读体验太差，如图 10.4.2 所示，因此，接下来的例子中不会保留这一样式规则。）

1. 创建斜体

(1) 输入 font-style:

(2) 在 : (冒号) 后输入 italic (创建斜体文本) 或 oblique (创建倾斜文本)。(99% 的情况下都会使用 italic。在所有情况下，很难察觉使用 oblique 与 italic 的差异。)

```
body {
    font-family: Geneva, Tahoma, Verdana,
        → sans-serif;
}

h1,
h2 {
    font-family: "Gill Sans", "Gill Sans MT",
        → Calibri, sans-serif;
}

p {
    font-style: italic;
}
```

图 10.4.1 在这个例子中，段落以斜体显示

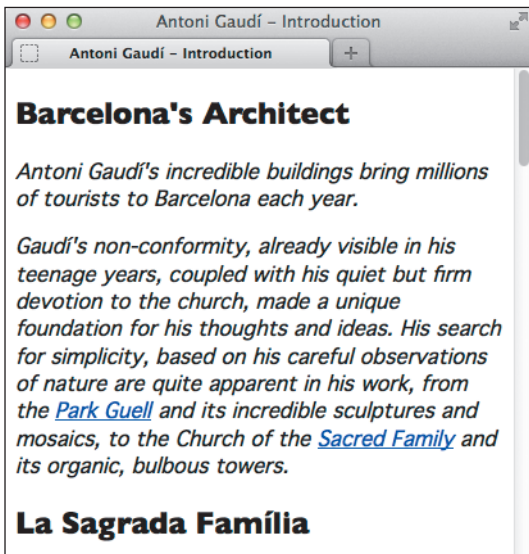


图 10.4.2 两个标题以正体显示，它们之间的段落以斜体显示

2. 取消斜体

输入 font-style: normal。

提示 想取消斜体的一个可能的原因是，要在从父元素中继承了斜体格式的段落中对某些文字进行强调。关于继承的更多细节，参考 7.3 节。

提示 关于编写语义化 HTML 的重要性参见 1.8 节。

提示 `font-style` 属性是继承的。

真斜体、假斜体、倾斜版本的区别

字体的斜体版本通常是由字体设计师从头创建的，尤其是有衬线字体。字体的斜体版本不仅仅是普通文本的倾斜版本，它们的形式有一些合理的差异。例如，Palatino Linotype 就有一份真正的斜体字体，参见图 10.4.3。通过字母 a 可以很清晰地看到，斜体不仅仅是对字母进行倾斜产生的效果。

不过，有的字体并没有斜体版本。如果对这些字体的文本设置 `font-style: italic`，浏览器就会显示一种计算机模拟出来的假斜体，即简单地对普通字母进行倾斜以模拟斜体风格（参见图 10.4.3）。这两种方式的质量是有所不同的。

此外，字体设计师还可能专门为字体创建倾斜版本，它们通常是对普通字母进行倾斜，同时可能对字符间距等进行微调而产生的，与普通字母相比，字母本身的形状是相同的。要使用字体的倾斜版本，可以设置

`font-style: oblique;`，不过这样做并不常见。如果字体没有斜体版本或倾斜版本，就会以伪斜体显示。



图 10.4.3 这里有两组应用了不同字体的文字，每组的第一行都是普通的文本，第二行是添加了 `font-style: italic;` 的文本。如果仔细观察，你会发现，Palatino Linotype 字体对斜体字母作了特殊处理。正体与斜体的区别在 a、p 和 y 等字母上表现得最为明显。由于 Geneva 没有斜体字母，因此浏览器只是简单地让普通文本倾斜，模拟斜体效果

10.5 应用粗体格式

粗体格式可能是让文本突出显示的最常见、最有效的方式。例如，浏览器通常默认为 `h1 ~ h6` 添加粗体格式。同斜体一样，可以为任何格式添加或取消粗体。添加粗体和取消粗体使用 `font-weight` 属性（参见图 10.5.1 和图 10.5.2）。

1. 应用粗体格式

(1) 输入 `font-weight:`。

(2) 输入 `bold`，让文本显示为具有平均加粗值的粗体。这一属性值适用于大多数情况。

或者输入 100 ~ 900 之间的 100 的倍数，其中 400 代表正常粗细，700 代表粗体。如果使用的是具有多种粗细版本的字体，这种方法就很有用（有些 Web 字体就采用了这种方法）。

或者输入 `bolder`（更粗）或 `lighter`（更细）以设置相对当前粗细的值。

```
body {
  font-family: Geneva, Tahoma, Verdana,
    → sans-serif;
}

h1,
h2 {
  font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
  font-weight: normal;
}

em,
a:link,
.intro.subhead {
  font-weight: bold;
}
```

图 10.5.1 浏览器会自动为标题 h1 ~ h6 添加粗体格式。我为所有的 h1 和 h2 元素应用了正常的粗细，取消了自动添加的粗体格式。同时，我为 em 文本及链接、.subhead 添加了粗体格式（参见图 10.5.2）



图 10.5.2 标题是以常规粗细显示的，而不是默认的粗体（参见图 10.4.2）。.subhead 类段落是以粗体显示的，链接也是

2. 取消粗体格式

输入 `font-weight: normal`。（有些 Web 字体服务，如 Google Fonts，要求使用 `font-weight: 400`。）

提示 由于各种字体定义粗细的方式并不相同，因此预定义值在不同字体上的表现可能不一致。预定义值用以表示某一给定字体下的相对值，如 700 在一种字体中看起来可能比在另一种字体中要粗。

提示 字体本身有时候不包括与相关值对应的不同程度的粗细。如果字体的粗细少于九种，或者这些粗细集中于范围的一端，那么一些数值就可能对应于同样的粗细，在网页上的显示效果就都一样，参见图 10.5.3。

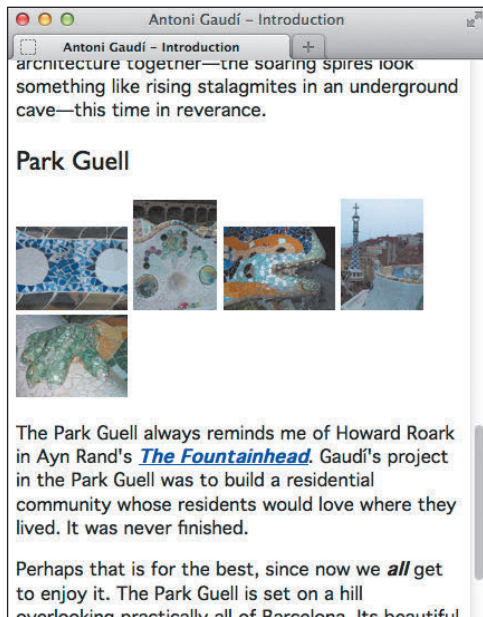


图 10.5.3 在页面的底部，可以看到一个链接（The Fountainhead）和单词 all。根据我们新添加的样式，它们都以粗体显示，同时它们还会因为浏览器的默认样式而显示为斜体。根据其含义这两个短语分别由 cite 和 em 标记（注意 h2 文字 Park Guell 是普通字体，没有加粗）

提示 鉴于前两条提示，添加粗体样式的通行做法是简单地写成 `font-weight: bold`。

提示 有的字体没有设计粗体字母。对于这些字体，如果声明 `font-weight: bold;`，浏览器会显示伪粗体（如图 10.5.4 和图 10.5.5 所示）。对伪斜体的讨论参见 10.4 节。

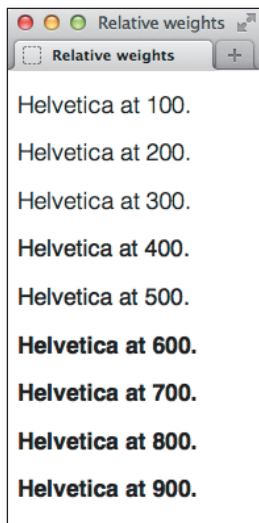


图 10.5.4 有的粗细值不同的文本显示的效果一致，是因为 Helvetica 并没有为每种粗细值都建立一套字母集

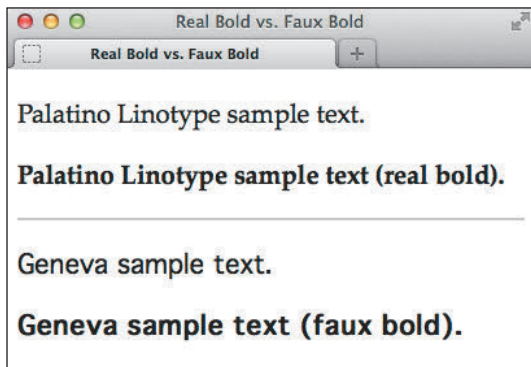


图 10.5.5 同斜体类似，Palatino Linotype 字体有粗体字母，而 Geneva 没有。因此，对于 Geneva，浏览器会将字母加粗，模拟粗体效果，而这样做的结果则是不够锐利、优雅，字母间隔也不是特别理想

提示 哪些元素是有可能需要取消粗体的呢？这包括任何自动添加了粗体格式的元素（应该能想到 `strong`、`h1 ~ h6` 和 `b`），以及继承了父元素粗体格式的元素（参见 7.3 节）。

提示 `font-weight` 属性是继承的。

10.6 设置字体大小

为网页里的文本设置字体大小有两种方式：直接使用像素指定要使用的特定字号（如图 10.6.1 ~ 10.6.3 所示），或使用百分数、`em`（如图 10.6.4 和图 10.6.5 所示）或者 `rem` 指定相对于父元素文本的大小。关于 `rem`，参见补充材料“使用 `rem` 指定字体大小”。

```
... [previous CSS] ...

em,
a:link,
.intro .subhead {
    font-weight: bold;
}

h1 {
    font-size: 35px;
}

h2 {
    font-size: 28px;
}

.intro .subhead {
    font-size: 18px;
}

.intro p {
    font-size: 17px;
}

.project p {
    font-size: 15px;
}
```

图 10.6.1 这里使用像素值指定文本大小（效果参见图 10.6.2 和图 10.6.3）



图 10.6.2 图 10.6.1 中指定的字体在浏览器中的显示效果。标题字号的大小显示了信息的层级关系。主标题下面的介绍区域中的段落文本字号稍大

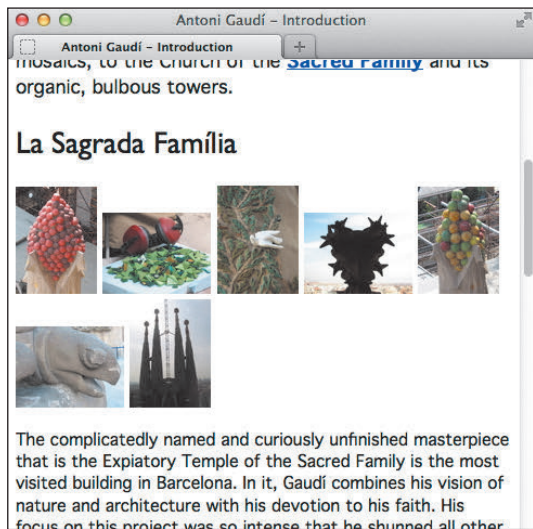


图 10.6.3 页面中剩余部分的段落文本

对初学者来说，像素更容易理解，不过强烈建议你一开始就学习使用 em 这样的相对单位。使用相对单位可以有更大的灵活性，而且对定义页面中特定的设计部件（如空白、边距等）的尺寸很有帮助。在各种尺寸的设

备（智能手机、平板电脑等）不断涌现的今天，使用相对单位有助于建立在各种设备都能显示良好的页面（这就是响应式 Web 设计涉及的内容，参见第 12 章）。

```
body {
  font-family: Geneva, Tahoma, Verdana,
    → sans-serif;
  font-size: 100%; /* 16px */
}

h1,
h2 {
  font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
  font-weight: normal;
}

h1 {
  font-size: 2.1875em; /* 35px/16px */
}

h2 {
  font-size: 1.75em; /* 28px/16px */
}

em,
a:link,
.intro .subhead {
  font-weight: bold;
}

.intro .subhead {
  font-size: 1.125em; /* 18px/16px */
}

.intro p {
  font-size: 1.0625em; /* 17px/16px */
}

.project p {
  font-size: .9375em; /* 15px/16px */
}
```

图 10.6.4 body 里的 font-size: 100% 声明为 em 字体大小设置了参考的基准。这里的 100% 将被翻译为默认字体大小（大多数系统下为 16px）。这样，样式表的结果同图 10.6.1 所示的结果便是一样的。每个 font-size 属性值后面的注释解释了该值的计算方法，同时显示了等价的像素值



图 10.6.5 在大多数系统中，如果使用默认设置，以 em 为基础的字体大小同以像素为基础的版本是一样的（如图 10.6.2 和图 10.6.3 所示）

1. 理解 em 和百分数字体大小

设置相对于父元素的字体大小需要一点时间去习惯。你需要理解浏览器处理这些相对于父元素的单位的方式，下面花一点时间解释一下。

不过首先要说明，使用这种方法时，最好在 body 元素上建立一个基准，即声明 `body { font-size: 100%; }`（参见图 10.6.4）。大多数时候，这项设置等价于将字体大小设为 16px，即大多数系统的默认字体大小。照例，该属性值传递至其他的元素（记住，font-size 是继承性属性），除非其他元素通过浏览器默认设置了自己的 font-size，或者在样式表指定了自己的 font-size。（多数元素是

16px，但默认情况下 h1 ~ h6 稍大。）

那么，如何计算要指定的 em^① 值呢？实际上，1em 总是等于默认的字号大小，这是 em 的工作原理。在这里，1em 就等于 16px，因为我们将其创建为元素的默认字号。据此可以通过一点点除法确定 em 值（或百分比）。

要指定的字体大小 / 父元素的字体大小 = 值

让我们来看看如何将图 10.6.1 中的像素值改为 em。例如，如果要将 h1 设为 35px，同时已知其父元素的字体大小为 16px，因此

$$35 / 16 = 2.1875$$

因此，设置 `h1 { font-size: 2.1875em; }` 就可以了（参见图 10.6.4）。这条规则表示“将 h1 文本的大小设置为其父元素文本大小的 2.1875 倍”。另一种方式是写成 `h1 { font-size: 218.75%; }`。不过，除了在 body 中设置基准 font-size 时使用百分数以外，设置字体大小时使用 em 比百分数更为常见。

再如，要将 project 类段落文本设置为 15px，而

$$15 / 16 = .9375$$

因此，设置 `p { font-size: 0.9375em; }`，参见图 10.6.4（这个值也可以设成 93.75%）。

再讨论一个例子。这可能是你感到困惑的地方。页面中的第二个段落 Gaudi's non-conformity... 包含两个链接（如图 10.6.5 和图 10.6.6 所示）。段落本身的字体样式设置为 `.intro p { font-size: 1.0625em; }`，由于 $17/16=1.0625$ ，因而段落文本字号为 17 像素。

假设要让链接显示为 16px，同时让段落里的其他文本仍显示为 17px。你可能打算将链接的 font-size 设为 1em（`.intro a { font-size: 1em; }`），因为 $1em = 16px$ 。

① 在排版领域，em 是一种量度单位，1em 等于当前指定的字号大小。em 这一名称与字母 M 有关，起初，1em 等于给定字体环境下字母 M 的宽度。——译者注

```

...
<div class="intro">
  <h1>Barcelona's Architect</h1>
  ...
  <p>Gaudí's non-conformity...His search for simplicity...is quite apparent in his work, from
  → the <a href="#park-guell">Park Guell</a> ... to the Church of the <a href="#sagrada-
  → familia">Sacred Family</a> and its organic, bulbous towers.</p>
</div>
...

```

图 10.6.6 这段 HTML 包含两个 a 元素，它们都位于一个父元素 p 中。div 容器的 intro 类是为了在不影响页面中其他元素样式的情况下，给该 div 中的段落和链接设置样式而引入的（如图 10.6.1 至图 10.6.4 所示）

但需要记住的是，该值应该是相对于这些元素的父元素的。在这个例子中，它们的父元素是 p。段落的字体大小是 17px，而不是 16px，因此要让 1em = 17px，前面说过，1em 应该始终等于父元素的字体大小。

因此，要让链接显示为 16px，需要设置一个比 1em 大的 em 值：

$$16 / 17 = .941176$$

因此，使用这个有点冗长的 .intro a { font-size: 0.941176em; } 将得到想要的结果（数字太长了不好看，因此小数点后我省略了几位）。

最后需要指出的是，在大多数情况下，100% 的 body font-size 等于 16px。不过有一种例外的情况，即用户对浏览器的设置覆盖了该默认值。例如，如果用户是视障人士，他们可能将默认字体大小设置为 22px。将 body 设置为 100%，页面就会以此为基准设置其余的文本的相对大小。这是使用 em 和百分数设置字体大小的美之所在。（如果你将 html 元素设置为 100%，同样的道理也适用于 rem。）参见最后一条提示，了解使用 em 的更多原因。

使用 rem 设置字体大小

CSS3 引入了一些新的单位，其中很有意思的一个便是 rem（root em 的简称）。它同 em 很像，不过它总是以根元素为参照系设置其他元素的字体大小，而不是父元素，参见图 10.6.7。根元素是 html 元素（因为它包含 body，也就包含了所有内容）。

这样做简化了字体大小的设置，因为 html 的字体大小通常不会变，不像父元素的大小是不确定的，就像在段落中设置链接字体大小的例子那样。（参见“理解 em 和百分数字体大小”部分。）因此，我们的公式（在

前面公式的基础上调整而来）为：

要指定的字体大小 / 根元素字体大小 = 值

实际上就是

要指定的字体大小 / 16 = 值

是吧？我已经能听到你的欢呼了。

先别着急，虽然现代浏览器对它的支持程度很高，但 Internet Explorer 直到 IE9 才开始支持它（<http://caniuse.com/#search=rem>）。在世界上很多地方，IE8 仍然占据较大的市场份额，建站时不得不考虑对其兼容。这也是很多代码编写人员尚未使用 rem 的原因。

不过，有一种策略就是针对旧版本的 IE 提供以 px 为单位的值，再紧跟着为现代浏览器准备的 rem 值（如图 10.6.7 所示）。当然，这样做的一个缺点是需要维护的代码量和浏览器需要加载的代码量变多了。另一个缺点是，在 IE8 中如果文本的字体大小是以像素为单位的，用户就无法改变文字的大小。诚然，这会影响一小部分用户。请你自己权衡并作出取舍。

```
html {
    font-size: 100%; /* 通常是16px */
}

.intro p {
    font-size: 17px; /* 可选 */
    font-size: 1.0625rem; /* 17px/16px */
}

.intro a {
    font-size: 16px; /* 可选 */
    font-size: 1rem; /* 是 .941176em */
}
```

图 10.6.7 使用 rem，则只关心根元素的字体大小，而不必关心链接的父元素 p 的字体大小。由于根元素（html）设置为 100%（通常为 16px），则将 a 元素设置为 1rem，则表示其字体大小也是 16px。此外，标记了 /* 可选 */ 的样式表示可以提供一个以像素为单位的字体大小，从而为 IE8 及其他不支持 rem 的旧浏览器提供备选方案

2. 指定特定字体大小

(1) 输入 font-size:。

(2) 在冒号 (:) 后输入准确的字号，如 13px（关于像素示例，参见图 10.6.1）。

或者使用关键字指定字体大小，即使用 xx-small、x-small、small、medium、large、x-large 或 xx-large。

关于单位的详细信息，参见 7.5 节。

提示 数字与单位之间不应有任何空格。

提示 如果以像素为单位设置字体大小，使用 Internet Explorer 的访问者将无法使用浏览器的文本大小选项对文本进行放大或缩小。这是要用 em 或百分比控制字体大小的原因之一。从 IE7 开始，访问者可以对整个网页放大或缩小，不过这跟只改变文字大小不同。如今，对于大多数网站来说，编码人员已经无需再担心 IE6 和 IE7 了，因为这两个版本

的浏览器用户已经非常少了。不过，截至本书写作之时，在中国还有 25% 的用户使用 IE6，要查看它在世界各地的份额，可以访问 www.ie6countdown.com。

提示 不同的浏览器对关键字的解释方式可能有所不同。

提示 磅 (pt) 用做打印样式表的单位。

提示 font-size 属性是继承的。

3. 根据父元素设置字体大小

(1) 输入 font-size:。

(2) 在冒号 (:) 后输入相对值，如 1.5em 或 150%（em 和百分数示例参见图 10.6.4）。

或者使用相对关键字，即使用 larger 或 smaller（这种用法没有百分数常见，而百分

数又没有 `em` 常见)。

4. 根据根元素设置字体大小

(1) 输入 `font-size:`。

(2) 在冒号(:)后输入相对值, 如 `0.875rem` (`rem` 示例参见图 10.6.7)。

提示 父元素的字体大小可能是由用户(比较少见)或设计人员设置的, 可能是从父元素继承的, 也可能来自浏览器的默认设置。上文已经提到, 大多数浏览器对 `body` 元素设置的默认字体大小为 16 像素。

提示 设置了相对字体大小的元素的子元素会继续继承这个大小, 而不是继承相对值。因此, `p` (参见图 10.6.6) 中 `a` 元素会继承 15 像素的字体大小 (参见图 10.6.4), 而不是相对值 `0.9375em`。链接将显示为 15px, 除非被其他样式覆盖。

提示 可以将字体大小同其他字体值一起进行设置。参见 10.7 节。

提示 关于在设计网页过程中使用 `em` 优于使用像素, Chris Coyier 给出了多种原因, 参见 <http://css-tricks.com/why-ems/>。

10.7 设置行高

行高指的是段落的行距, 即段落内每行之间的距离, 参见图 10.7.1 和图 10.7.2。使用大一些的行高有时候会使主体文本更容易阅读。对于超过一行的标题, 使用较小的行高则会让它们看起来更美观。

... 省略前面的 CSS ...

```
.intro {
    line-height: 1.45;
}

.intro .subhead {
    font-size: 1.125em;
}

.intro p {
    font-size: 1.0625em;
}

.project p {
    font-size: .9375em; /* 15px/16px */
    line-height: 1.65; /* 15px*1.65 =
    → 24.75px */
}
```

图 10.7.1 假设 `body` 元素默认大小为 16 像素, `.project` 类 `p` 元素的字体大小为 `0.9375em`, 即 15 像素。行高将是 15 像素的 1.65 倍, 即大约 24.75 像素。我还设置了 `.intro` 容器的行高, 它会被后代元素继承

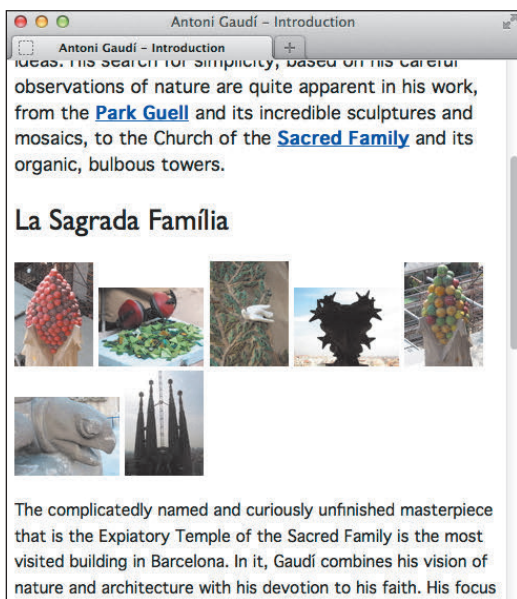


图 10.7.2 使用 `line-height` 拉大行距可以使其更容易阅读。在这个例子中, 底部的主文本之间的行距要比顶部的 `intro` 文本之间的行距明显大

设置行高的步骤

(1) 输入 `line-height`。

(2) 输入 n ，这里的 n 是一个数字，它与元素的字体大小相乘，得出需要的行高。（这是最为常用的方法，即一个没有单位的数字。）

或者输入 a ，这里的 a 是以 em、像素或磅（仅在打印样式表中使用磅）为单位的值。

或者输入 $p\%$ ，这里的 $p\%$ 是字体大小的百分数。

提示 如果使用数字设定行高（通常都这么做），那么所有的子元素都会继承这个因子。因此，如果父元素的字体大小是 16 像素（或以 em 等表示的等价大小），行高是 1.5，则该元素的行高就是 24（即 16×1.5 ）像素。如果子元素的字体大小是 10 像素，则该元素的行高就是 15（即 10×1.5 ）像素。

提示 如果使用百分数或 em 值，那么只会继承产生的行高（即计算出来的值）。因此，如果父元素的字体大小是 16 像素，行高是 150%，则该元素的行高就是 24 像素。所有的子元素都将继承 24 像素的行高，不管字体大小是多少。

提示 根据下一节将要讲到的，可以将行高跟字体、大小、粗细、字体样式和变体一起进行设置。

10.8 同时设置所有字体值

可以使用 `font` 简写属性同时设置字体样式、粗细、变体、大小、行高和字体系列，参见图 10.8.1 和图 10.8.2。这种方法要比单独声明各个属性高效，因此采用这种方法可以保持样式表简洁。

使用 `font` 简写属性不要求指定所有字体

属性，但至少应该包含字体大小和字体系列属性。图 10.8.1 中的两个注释说明了不能使用简写属性的情况。

```
body {
    font: 100% Geneva, Tahoma, Verdana,
    → sans-serif;
}

h1,
h2 {
    /* 这些声明无法使用font简记法，除非同时
    声明字体大小 */
    font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
    font-weight: normal;
}

h1 { font-size: 2.1875em; }
h2 { font-size: 1.75em; }

em,
a:link,
.intro .subhead {
    font-weight: bold;
}

.intro {
    line-height: 1.45;
}

.intro .subhead {
    font-size: 1.125em;
}

.intro p {
    font-size: 1.0625em;
}

.project p {
    /* 这些声明无法使用font简记法，除非同时
    声明字体系列 */
    font-size: .9375em;
    line-height: 1.65;
}
```

图 10.8.1 这个样式表与图 10.7.1 中的样式表是等效的，其效果如图 10.8.2 所示。这里只是将 `body` 的字体属性放在了一条 `font` 声明中。代码中无法将 `h1`、`h2` 或 `.project` 类 `p` 元素的字体属性放入 `font` 中，因为 `font` 简写属性至少应该包括字体系列和字体大小属性。更多示例参见图 10.8.3 和图 10.8.4

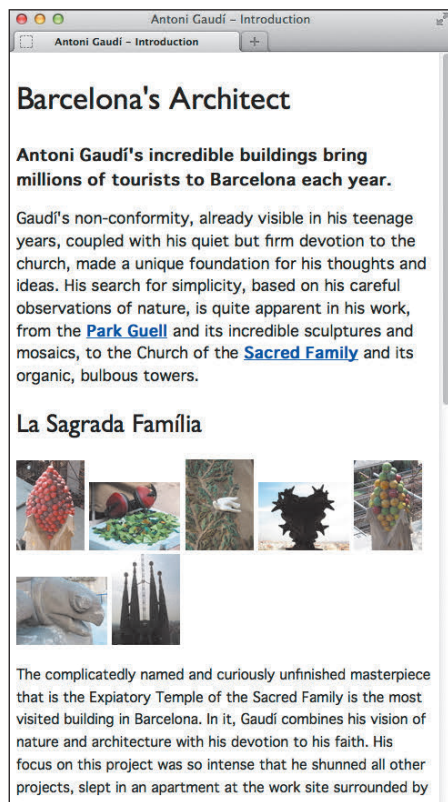


图 10.8.2 这个页面与图 10.7.2 是相同的

```
.example-2 {
  font: .875em/1.3 "Palatino Linotype",
    → Palatino, serif;
}
```

图 10.8.3 这是一个结合了 font-size、line-height 和 font-family 声明的 font 简记法的例子。行高跟在字体大小和一个斜杠后面。font 简写属性中还可以包含 font-style、font-variant 和 font-weight，参见图 10.8.4

```
.example-3 {
  font: italic small-caps bold .875em/1.3
    → "Palatino Linotype", Palatino, serif;
}
```

图 10.8.4 这个例子中包含了所有可能的属性。只要 font 中声明了字体大小和字体系列属性，其他属性可以任意组合。前三个属性的次序无关紧要

同时设置所有字体值的步骤

(1) 输入 font:

(2) 可选步骤，输入 normal、italic 或 oblique 以设置字型（参见 10.4 节）。

(3) 可选步骤，输入 normal、bold、bolder、lighter 或 100 的倍数（最大到 900）以设置粗细（参见 10.5 节）。

(4) 可选步骤，输入 normal 或 small-caps 来取消或设置小型大写字母（参见 10.15 节）。

(5) 可选步骤，输入需要的字体大小（参见 10.6 节）。

(6) 如果需要，输入 /line-height，这里的 line-height 是行与行之间的距离（参见 10.7 节）。

(7) 输入一个空格，再按优先次序输入需要的字体系列，以逗号分隔（参见 10.2 节）。

提示 必须始终显式地声明字体大小和字体系列属性：先是字体大小，再是字体系列。

提示 行高是可选的，但它如果出现，就必须紧跟在字体大小和斜杠后面（参见图 10.8.3 和图 10.8.4）。

提示 第 (2) ~ (4) 步中的属性可以按任意顺序指定（参见图 10.8.4），也可以忽略（参见图 10.8.1 和图 10.8.3）。如果忽略它们，它们就被自动设为 normal，而这也许不是你想要的。

提示 font 属性是继承的。

10.9 设置颜色

可以修改网页中文本的颜色。我们的样式表可以包含颜色名称、十六进制值、RGB、HSL、RGBA 和 HSLA 值的任意组合来定义颜色，参见图 10.9.1 和图 10.9.2。关于各种颜色值的信息参见 7.5 节中的“CSS 颜色”。

```
body {
  color: blue;
  font: 100% Geneva, Tahoma, Verdana,
  → sans-serif;
}

...

h2 {
  color: #7d717c;
  font-size: 1.75em;
}

...

/* :::: 链接 :::: */
a:link {
  color: #e10000; /* a red */
}

a:visited {
  color: #b44f4f;
}

a:hover {
  color: #f00;
}

.intro a {
  color: #fdb09d; /* 略带粉色 */
}

.intro a:hover {
  color: #fec4b6;
}
```

图 10.9.1 在 `body` 元素中设置页面的默认文本颜色，这样所有的元素都会继承设置（`link` 除外）。再在相应元素中设置特殊需求的颜色（如我在 `h2` 中的设置）。注意，`a:hover` 颜色（`#f00`）使用了第一条提示中说到的缩写（为展示效果，这个例子中的 `body` 元素使用了蓝色，后面不会保留这一设置）

Barcelona's Architect

Antoni Gaudi's incredible buildings bring millions of tourists to Barcelona each year.

Gaudi's non-conformity, already visible in his teenage years, coupled with his quiet but firm devotion to the church, made a unique foundation for his thoughts and ideas. His search for simplicity, based on his careful observations of nature are quite apparent in his work, from the **Park Guell** and its incredible sculptures and mosaics, to the Church of the **Sacred Family** and its organic, bulbous towers.

La Sagrada Família



The complicatedly named and curiously unfinished masterpiece that is the Expiatory Temple of the Sacred Family is the most visited building in Barcelona. In it, Gaudi combines his vision of nature and architecture with his devotion to his faith. His focus on this project was so intense that he shunned all other projects, slept in an apartment at the work site surrounded by plans and drawings, and so completely ignored his disheveled appearance that when, in 1926, he was struck by a streetcar in front of the church, he was mistaken for an indigent and brought to a hospital for the poor where he died soon thereafter.

The Sagrada Familia attracts even the non-religious to its doors in large part due to this tragic story and its still unfinished state, of which the everpresent scaffolding and cranes are permanent reminders. But there is something more. In the Sagrada Familia, Gaudi again brings nature and architecture together—the soaring spires look something like rising stalagmites in an underground cave—this time in reverence.

Park Guell



The Park Guell always reminds me of the character Howard Roark in Ayn Rand's *The Fountainhead*. Gaudi's project in the Park Guell was to build a residential community whose residents would love where they lived. It was never finished.

Perhaps that is for the best, since now we *all* get to enjoy it. The Park Guell is set on a hill overlooking practically all of Barcelona. Its beautiful and even comfortable serpentine bench is filled with foreigners and locals alike every day of the week. Its mosaic lizard has become synonymous with the city itself.

图 10.9.2 不出所料，扎眼的蓝色应用到了页面的大部分文本中，`h2` 标题采用中度灰色（`#7d717c`）覆盖了设置。页面顶部 `.intro` 区域中粉红色（`#fdb09d`）的链接覆盖了默认红色（`#e10000`）链接。下一节应用背景颜色后我会将这里的文本设为白色，不过现在不行，现在设置成白色就看不到了

设置颜色的步骤

(1) 输入 `color`。

(2) 输入 `colorname`，这里的 `colorname` 是预定义颜色中的一种。

或者输入 `#rrggbb`，这里的 `#rrggbb` 是颜色的十六进制呈现。这是最常用的指定颜色的方法。

或者输入 `rgb(r, g, b)`，其中 `r`、`g`、`b` 是 0 ~ 255 之间的整数，分别表示所需颜色里红、绿、蓝的量。

或者输入 `rgb(r%, g%, b%)`，其中 `r`、`g`、`b` 分别是所需颜色里红、绿、蓝的百分数。

或者输入 `hsl(h, s, l)`，其中 `h` 是 0~360 之间的数值，表示所需颜色的色相；`s` 和 `l` 均是百分数，分别表示所需颜色的饱和度和亮度。（一般来说，对于不透明颜色，使用十六进制数或 RGB 值更好。）

或者输入 `rgba(r, g, b, a)`，其中 `r`、`g`、`b` 是 0 ~ 255 之间的整数，分别表示所需颜色里红、绿、蓝的量；`a` 是 0~1 之间的小数，表示所需颜色的 alpha 透明度。

或者输入 `hsla(h, s, l, a)`，其中 `h` 是 0 ~ 360 之间的数值，表示所需颜色的色相；`s` 和 `l` 均是百分数，分别表示所需颜色的饱和度和亮度；`a` 是 0 ~ 1 之间的小数，表示所需颜色的 alpha 透明度。

如果输入的 `r`、`g` 或 `b` 的值大于 255，就会使用 255。类似地，高于 100% 的百分数将被替换为 100%。

提示 当十六进制数值由三对重复的数字组成时，还可以使用 `#rgb` 设置颜色。事实上，我推荐使用这种方法。因此，可以（且应该）将 `#FF0099` 写做 `#F09` 或 `#f09`。类似地，可以将 `#CC0000` 写成 `#C00` 或 `#c00`。但是不能将 `#31AA55` 写成 `#31a5`，因为只有两对值是重复的。

提示 记住，Internet Explorer 在 IE9 之前的版本不支持 HSL、RGBA 和 HSLA，因此，如果要在颜色声明中使用这些记法，就需要为 IE 的旧版本定义备用颜色。详细说明参见 7.5 节中的“CSS 颜色”。

提示 图 10.16.2 展示了 `:hover` 链接即将激活的效果。

提示 除了 `link` 元素，其他元素都会继承 `body` 元素中的 `color` 属性。开发人员必须显式设置 `link` 元素的 `color` 属性，如图 10.9.1 所示。

10.10 设置背景

我们在设置背景样式时有很多选择。可以为单个元素设置背景，可以为整个页面设置背景，还可以为上述二者的任意组合设置背景。如此，便可以对几个段落、几个单词、不同状态的链接、内容区域等修改背景。总之，几乎可以对所有元素应用背景样式，甚至是表单和图像。（是的，你没看错，图像也可以有背景图像！）

设置背景有很多属性可以利用，包括 `background-color`、`background-image`、`background-repeat`、`background-attachment` 及 `background-position` 等。还可以使用 `background` 简记法，该属性将上述属性合并了，可以节省大量的输入。本节将介绍这些属性。

我们将首先为本章一直在使用的示例页面添加背景颜色，然后介绍一些额外的与背景相关的例子以深入探讨这一主题。

1. 修改文本的背景颜色

(1) 输入 `background`。

(2) 输入 `transparent` (表示允许透过父元素的背景颜色) 或 `color`, 这里的 `color` 是颜色名称、十六进制数值, 或 RGB、HSL、RGBA、HSLA 颜色值(参见 10.9 节第 (2) 步), 如图 10.10.1 ~ 图 10.10.6。十六进制值颜色是最为常用的。

```
body {
  background-color: #88b2d2;
  font: 100% Geneva, Tahoma, Verdana,
    → sans-serif;
}

...

h2 {
  background-color: #eaebef;
  color: #7d717c;
  font-size: 1.75em;
}

.intro {
  background-color: #686a63;
  color: #fff;
  line-height: 1.45;
}

...
```

图 10.10.1 除了那些单独设置背景颜色的元素之外, 对 `body` 元素设置背景颜色就是为整个页面设置背景颜色。`intro` 类 `div` 元素的背景设置将其与页面的其他部分区别开来, 参见图 10.10.2

```
.architect {
  background-color: #fff;
}
```

图 10.10.3 包含所有内容的 `article` 元素有 `class="architect"`, 因此这段 CSS 将让它拥有白色的背景(但一些部分除外), 如图 10.10.4 所示



图 10.10.2 整个页面(`body` 元素)的背景颜色为浅蓝色, 有两处浅蓝色背景被其他颜色覆盖: `intro` 类 `div` 的棕色背景, 以及二级标题的浅灰色背景



图 10.10.4 白色填充了整个内容区域。`.intro` 区域和 `h2` 也是 `architect` 类 `article` 的后代, 如果之前没有为它们添加不同的背景(如图 10.10.1 所示), 它们的背景也应该是白色的。在 `.architect article` 的外围还能看到浅蓝色的 `body` 背景, 这是由于浏览器默认为 `body` 设置了一些外边距

```
...

.architect {
  background-color: #fff;
  padding: 1.5em 1.75em;
}

.intro {
  background-color: #686a63;
  color: #fff;
  line-height: 1.45;
  padding: 1px 1.875em .7em;
}

...
```

图 10.10.5 这里用到了一个下一章才会讲解到的属性（参见 11.8 节），希望读者们原谅我

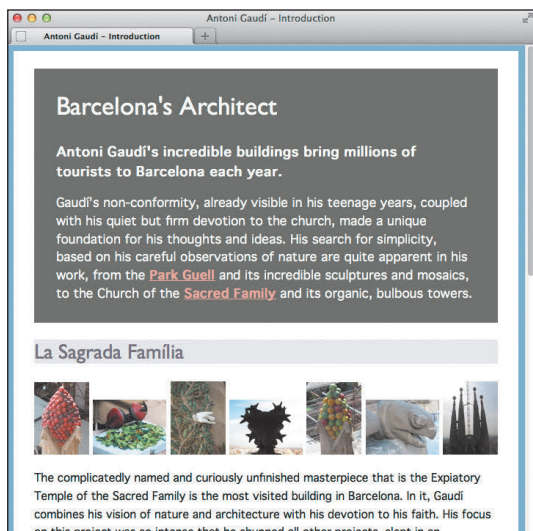


图 10.10.6 添加一点点内边距就能让页面大不一样，现在，页面看起来舒服多了

2. 使用背景图像作为背景

(1) 输入 `background-image:`。

(2) 输入 `url(image.png)`，这里的 `image.png` 是图像相对于样式表所在位置的路径和文件名，参见图 10.10.7。或者输入 `none`，表示不使用图像，如 `background-image: none;`（只有在覆盖应用背景图像的元素样式时才会

使用这个声明）。

```
body {
  background-color: #ccc;
  background-image: url(bg-pattern.png);
  ...
}
```

图 10.10.7 图像 URL 反映的是样式表相对于图像的位置。这里仅作简单处理，更多信息参见第二条提示。这里我希望图像能填充整个页面，但不必设置 `background-repeat`，因为背景图像默认就是重复的（如图 10.10.8 所示）。访问者只有在图像加载速度慢（例如在移动连接环境下）或图像由于某些原因没有加载成功时，才会看到背景颜色

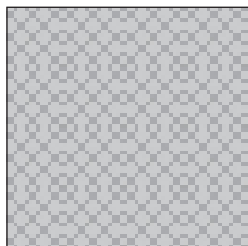


图 10.10.8 背景图像（上）包含一个漂亮的图案。将背景图像添加到页面主体后，它会在水平和竖直两个方向上不断重复。我将内容变窄了一些，以便看清效果

3. 重复背景图像

输入 `background-repeat: direction`，这里的 `direction` 可以取四个值：`repeat`、`repeat-x`、`repeat-y` 和 `no-repeat`，它们分别代

表同时横向和纵向重复图像（参见图 10.10.8）、只横向重复图像（参见图 10.10.9 ~ 图 10.10.12）、只纵向重复图像、不重复图像。忽略 `background-repeat` 相当于将其设为默认值 `repeat`，参见图 10.10.7。

```
body {
  background-image: url(sky.png);
  background-repeat: repeat-x;
  background-position: left bottom;
  ...
}
```

图 10.10.9 这次我们希望图像仅在水平方向（X 轴）重复。重复的开始位置是页面的左上角。注意这里没有提供背景色

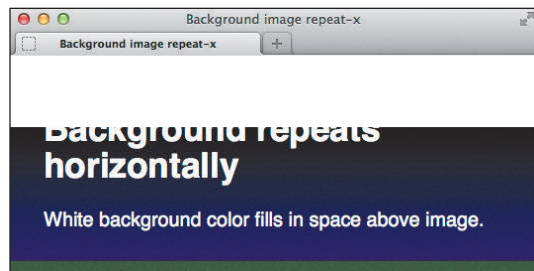
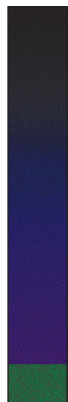


图 10.10.10 这张背景图像（上）由黑色向蓝色再向绿色渐变，很适合水平重复。不过它的高度不足以填充整个页面，在不设置背景色的情况下显得很难看。那一行文字其实并没有被切割，看不见它的上半部分是因为它的颜色与默认的背景色一样，都是白色。另见彩插

```
html {
  min-height: 100%;
}

body {
  background-color: #000;
  ...
}
```

图 10.10.11 在图 10.10.9 的基础上设置背景色，该背景色会在背景图像不覆盖的区域显示出来。这里还添加了一条规则，让 `html` 元素的最小高度为浏览器窗口高度的 100%，如图 10.10.12 所示（注意：如果将 `background-attachment` 的值设为 `fixed`，这一条规则就可以省略，下面很快就会讲到这一点）

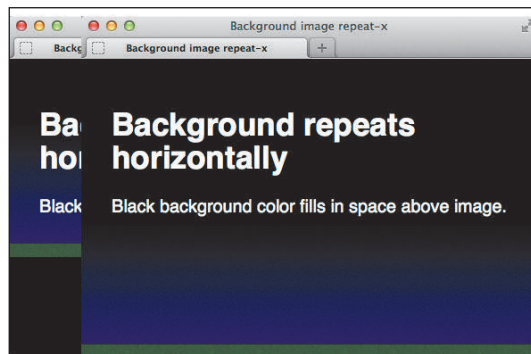


图 10.10.12 黑色（#000）背景色与图像顶端的黑色完美地融合在一起。现在，你可以看到那些文本了！这张图是由两个浏览器窗口截图合成的，这两个窗口的高度是一样的。如果没有为 `html` 元素添加图 10.10.11 中的规则，效果就如左图所示，背景图像的底部位于文字下面一点的位置（因为页面在那个位置结束了），因此黑色背景色会显示在它的下面。另见彩插

4. 控制背景图像是否随页面滚动

(1) 输入 `background-attachment: fixed`。

(2) 输入 `fixed`（代码如图 10.10.13 所示），背景图像会附着在浏览器窗口上（也就是说即使访问者滚动页面，图像仍会继续显示，参见图 10.10.14）；或 `scroll`，访问者滚动页面时背景图像会移动，参见图 10.10.15；

或 local，只有访问者滚动背景图像所在的元素（而不是整个页面）时，背景图像才移动。忽略此项，浏览器通常会使用默认值 scroll）。

```
body {
  background-color: #000;
  background-image: url(sky.png);
  background-repeat: repeat-x;
  background-attachment: fixed;
  background-position: left bottom;
  ...
}
```

图 10.10.13 配合我们设置的 background-position，背景图像将从浏览器窗口的左下角开始水平重复

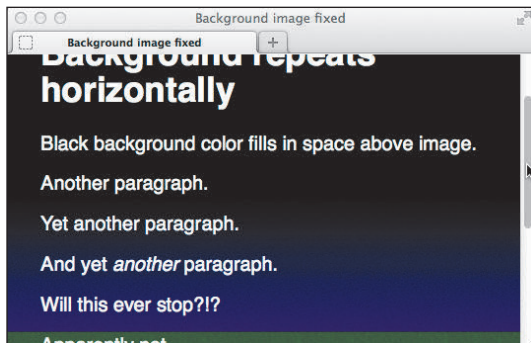


图 10.10.14 无论页面中有多少内容，无论用户如何滚动页面，背景图像固定不动

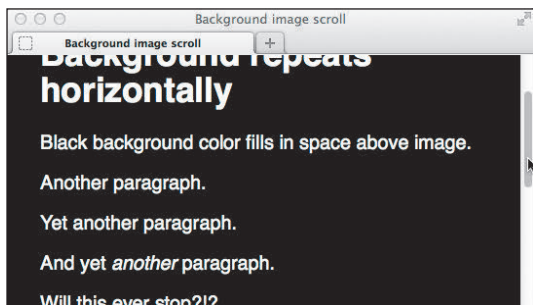


图 10.10.15 这是图 10.10.14 所示的页面在不设置 background-attachment: fixed; 情况下的显示效果。（不设置该规则等价于设置 background-attachment: scroll;，因为 scroll 是默认值。）背景图像的位置根据内容的长度而变，因此，直到滚动到页面的最底部，才能看到背景图像（如右图所示）

5. 指定元素背景图像的位置

输入 background-position_x y_，其中 x 和 y 可以表示为距离左上角的绝对距离或百分数，如图 10.10.16 所示，如 20px 147px（允许负值）。也可以用 left（左对齐）、center（居中）或 right（右对齐）表示 x，用 top（顶端对齐）、center（居中）或 bottom（底端对齐）表示 y，如图 10.10.13 所示。如果两个值都使用关键字表示，那么两者的顺序无关紧要，如 top right 跟 right top 一样。

```
body {
  background: #004 url(..img/ufo.png)
    → no-repeat 170px 20px;
  color: greenyellow;
  ...
}
```

图 10.10.16 现在，我们使用 background 简记法将所有这些单独的声明压缩成一条规则。不使用 background-position，而是将水平和竖直属性值直接写在最后。注意，使用 no-repeat 会让图像仅显示一次，不重复显示（如图 10.10.17 所示）

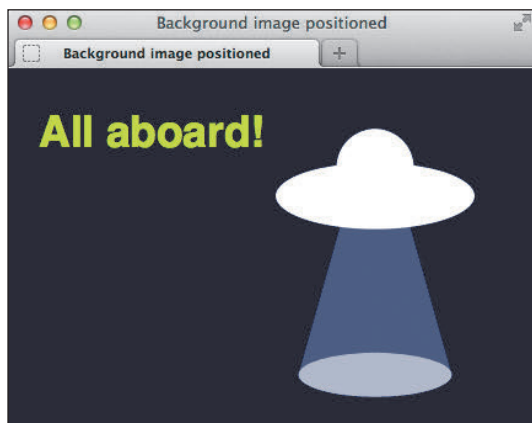


图 10.10.17 UFO 出现在距离页面左边缘 170 像素，距离上边缘 20 像素的位置，从而为绿色文本腾出空间。图像的空白区域都是透明的（光束则具有 alpha 透明度），因此它可以和深蓝色的 body 背景色融合在一起

6. 在一条声明中设置所有背景属性

(1) 输入 `background:`。

(2) 指定任何可接受的 `background` 属性值（从“修改文本的背景颜色”到“指定元素背景图像的位置”介绍的内容都适用）。这些值的排列顺序没有要求（如图 10.10.16 所示）。

提示 尽管大部分例子都是对 `body` 样式所做的修改，但实际上你可以为几乎所有的 HTML 元素添加背景。

提示 为了让例子变得简单，背景图像 URL 均使用基本的 URL。实际上，你的 URL 可能是类似于 `../img/ufo.png`（如图 10.10.16 所示）的形式，因为通常会避免将图像与样式表放在同一个目录里。本书配套网站上针对本章的例子均使用如图 10.10.18 所示的文件夹结构，因此 CSS 中的背景图像 URL 也会做相应调整。

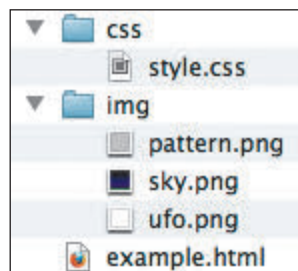


图 10.10.18 在这样的结构中，`style.css` 中显示背景图像的声明可能形如 `background: url(../img/pattern.png);`。更多信息参见 1.7 节

提示 `background-color` 的默认值是 `transparent`（透明），`background-image` 的默认值是 `none`（无），`background-repeat` 的默认值是 `repeat`（重复），`background-attachment` 的默认值是 `scroll`（滚动），`background-position` 的默认值是 `0 0`（等价于 `top left`，即左上角）。只有在需要覆盖其他样式规则的时候才会显式地指定默认值。

提示 通常，尽量使用 `background` 简记法。使用该简记法的时候，不必指定所有的属性。例如，在实践中，图 10.10.7 中的规则通常写作 `background: #ccc url(bg-pattern.png);`。不过需要注意的是，在简记法中，任何没有指定的属性都将设置为默认值，这有可能会覆盖先前定义的样式。

提示 默认情况下，元素的背景会填充其内容和内边距所在的区域（如图 10.10.5 和图 10.10.6 所示），并延伸到边框的边缘（参见 11.5 节）。可以通过 `background-clip` 改变这种规则（参见补充材料“更多 CSS3 的背景控制”）。

提示 在 CSS3 中, `background-attachment` 还可以设置成 `local`。Lea Verou 的一篇文章演示了该属性值的效果 (参见 <http://lea.verou.me/2012/04/background-attachment-local/>)。支持该属性值的浏览器仅包括 IE9+、Chrome、Safari 和 Opera (没有 Firefox) ^①。

提示 可以在 `background-position` 属性中使用负数。例如, `background-position: -45px 80px` 会将图像定位在元素左边外侧 45 像素 (因此不会看到图像在水平方向上的头 45 个像素)、顶部内侧 80 像素。

提示 文本和背景之间应有足够的对比度。这样做不仅能帮助普通用户, 对视力障碍的访问者尤其重要 (参见 <http://contrastrebellion.com>)。这里我们再次提到 Lea Verou, 他创建了一个帮助选择颜色的工具, 使用该工具可以选择对比度符合无障碍访问指南要求的颜色。

提示 如果图 10.10.14 中不使用 `background-position: left bottom`, 那么效果是相反的。我们将会看到图像在页面的顶端水平重复 (顶端是默认位置)。当页面向下滚动时, 该背景图像就会随着内容一起离开我们的视野。

更多 CSS3 的背景控制

CSS3 提供了更多的背景效果。本书将在第 14 章介绍渐变背景和多重背景。

`background-clip` 和 `background-origin` 两个属性分别控制元素背景显示的范围和开始的位置。这两个属性都接受相同的值: `content-box` 包含内容, `padding-box` 包含内容和内边距, `border-box` 包含内容、内边距和边框 (参见 11.5 节)。`background-clip` 的默认值为 `border-box`, 而 `background-origin` 的默认值则为 `padding-box`。更多信息参见 <http://css-tricks.com/transparent-borders-with-background-clip/>。

`background-size` 属性可以通过以下属性值控制背景图像的显示尺寸。

- ❑ `contain`: 在显示图像完整宽度和高度的情况下, 尽可能地扩大图像的尺寸。使用该值, 背景图像可能不会填充整个背景区域。
- ❑ `cover`: 在填充元素整个背景区域的情况下, 让图像尽可能地小。使用该值, 图像的一部分可能会超出元素的范围, 因而不可见。
- ❑ 一个长度、百分数或 `auto`: 如 `background-size: 250px 400px`; 或 `background-size: 50% 50%`。

关于该属性的更多信息参见 www.css3.info/background-size/。该属性擅长巧用 `sprite` (参见 12.5 节)。

大多数关于新的背景样式的讨论都可以在 www.sitepoint.com/new-properties-and-values-inbackgrounds-with-css3/ 找到。

^① 2013 年 10 月发布的 Firefox 25 版已支持该属性值。——译者注

10.11 控制间距

可以增加或减少单词之间或字母之间的距离，前者称为字间距（tracking），后者称为字偶距（kerning），参见图 10.11.1。

```
body {
  background-color: #88b2d2;
  font: 100% Geneva, Tahoma, Verdana,
    → sans-serif;
}

h1,
h2 {
  font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
  font-weight: normal;

  /* 出于演示目的进行临时设置 */
  letter-spacing: 7px;
}

h1 {
  font-size: 2.1875em;
}

... 其他CSS ...
```

图 10.11.1 这里为标题的字母之间添加了 7 像素的额外间距，letter-spacing 的效果非常明显了。在后面的例子中我会将这个值缩小为 1 像素

1. 指定字间距

输入 word-spacing: *length*, 这里的 *length* 是一个带单位的数字，如 0.4em 或 5px。

2. 指定字偶距

输入 letter-spacing: *length*, 这里的 *length* 是一个带单位的数字，如 0.4em 或 5px。

提示 可以对单词间距和字母间距使用负数。

提示 单词间距和字母间距的值还可能受到所选的对齐方式（用 text-alignment 设置）和字体系列的影响。



图 10.11.2 标题中字母之间的距离变大了。将这里的字母间距跟图 10.10.6 图中的字母间距进行比较，很容易看出效果。不过，后续例子不会保留这里的设置

提示 要将字母间距和单词间距设为默认值（即不添加额外的间距），可以使用 normal 或 0。

提示 如果要使用 em 值，那么只有结果大小（即“计算出来的值”）会被继承。因此，如果父元素字体大小为 16 像素，额外的单词间距为 0.1em，则每个单词之间的额外间距为 1.6 像素。同时，所有子元素每个单词之间也有 1.6 像素的额外间距，不管它们的字体大小是多少。如果要覆盖继承的值，可以显式地为子元素设置间距。

提示 word-spacing 和 letter-spacing 属性是继承的。

10.12 添加缩进

通过设置 text-indent 属性，可以指定段落第一行前面应该空出多大的空间，如图 10.12.1 和图 10.12.2 所示。

```

...
h1,
h2 {
    font-family: "Gill Sans", "Gill Sans MT",
        → Calibri, sans-serif;
    font-weight: normal;
    letter-spacing: 1px;
}

...

.project p {
    font-size: .9375em; /* 15px/16px */
    line-height: 1.65;
    text-indent: 2em; /* 30px */
}

... 其他CSS ...

```

图 10.12.1 这段代码为 p 元素添加了 2em 的缩进。由于 0.9375 的字体大小等于 15 像素（ $15/16=0.9375$ ），因此这个缩进约为 30 像素（参见图 10.12.2）

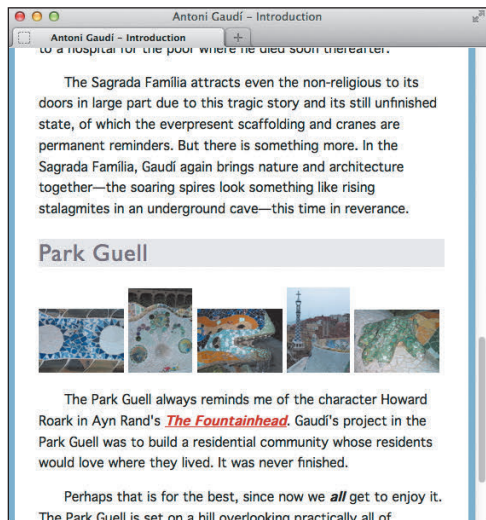


图 10.12.2 每个段落都有 30 像素的缩进

增加缩进的方法

输入 `text-indent: length`，这里的 *length* 是一个带单位的数字，如 1.5em 或 18px。

提示 我们可以为其他元素应用 `text-indent` 属性，而不仅是段落元素。但默认情况下对 `em`、`strong`、`cite` 等内联元素没有效果。可以将它们设置为 `display: block`；或 `display: inline-block`，就可以强制为其应用 `text-indent` 属性。

提示 使用负数会产生悬挂缩进。使用悬挂缩进时，可能还需要增加文字框周围的内边距或外边距，从而让容器可容纳伸到外边的文本（参见 11.8 节和 11.10 节）。

提示 与通常情况一样，`em` 值要根据元素的字体大小进行计算（参见图 10.12.1），百分数要根据父元素的宽度进行计算。

提示 `text-indent` 属性是继承的。

提示 如果要使用百分数或 `em` 值，那么只有生成的大小（即“计算出来的值”）会被继承。因此，如果父元素为 300 像素宽，那么 10% 的 `text-indent` 就是 30 像素，同时，所有子元素第一行也都会缩进 30 像素，而不管它们各自父元素的宽度是多少。

提示 要忽略继承的缩进，使用 0 即可。

10.13 对齐文本

根据需要，可以让文本左对齐、右对齐、居中对齐或两端对齐，参见图 10.13.1。

对齐文本的步骤

- (1) 输入 `text-align:`;
- (2) 输入 `left` 让文本左对齐；
或者输入 `right` 让文本右对齐；
或者输入 `center` 让文本居于屏幕的中间；

或者输入 `justify` 让文本两端对齐。

```
...
h1, h2 {
  font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
  font-weight: normal;
  letter-spacing: 1px;
  text-align: center;
}
...

p {
  text-align: justify;
}

.intro .subhead {
  font-size: 1.125em;
  text-align: center;
}

... 其他CSS ...
```

图 10.13.1 有的文本会居中对齐，但大多数文本都会两端对齐，参见图 10.13.2

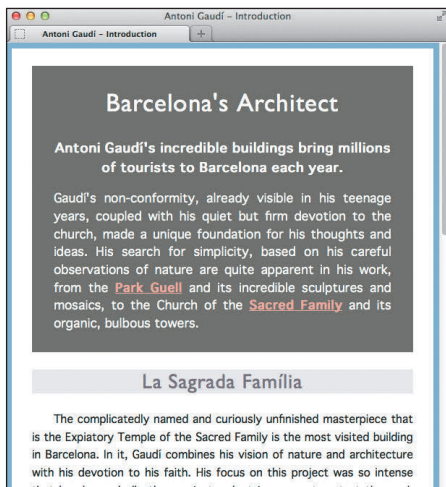


图 10.13.2 做出更改以后，标题和 `.subhead` 段落变成居中对齐，其他段落文本为两端对齐。之前为段落设置的 `text-indent` 值保留下来了。注意：我删除了第二个标题下面的图像，以便你能从这个图中看到更多文本

提示 如果选择让文本两端对齐，要注意单词间距和字母间距有可能受到严重的影响。更多信息参见 10.11 节。

提示 `text-align` 对适用于盒模型的元素（如 `h1 ~ h6`、`p` 等，显示为单独的行）来说是起作用的，但对短语内容元素（如 `strong`、`em`、`a`、`cite` 等，显示在行内）来说则不起作用，除非将它们的显示改为块级。（短语内容元素在 HTML5 之前称为“行内”元素，参见 1.9 节。）为了让短语内容元素中的文字对齐，区别于周边的文本，必须先将它们默认的 `display: inline;` 样式改为 `display: block;` 或 `display: inline-block;`（前者会让它们像段落一样显示为单独的行），然后再设置相应的 `text-align` 值。对于设置了 `display: inline-block;` 的元素，需要设置一定的宽度才能看到对齐效果。实际上，需要为“行内”元素设置 `text-align` 属性的情况是非常少见的。

提示 `text-align` 属性是继承的。它的默认值取决于文档的语言和书写系统（从右到左或从左到右），不过在大多数情况下，它会被不加区分地设置为左对齐。

10.14 修改文本的大小写

使用 `text-transform` 属性，可以为样式定义文本的大小写（参见图 10.14.1）。通过这种方法，可以将文本显示为首字母大写、全部大写（参见图 10.14.2）、全部小写或按原样显示。

```
body {
  background-color: #88b2d2;
  font: 100% Geneva, Tahoma, Verdana,
    → sans-serif;
}

h1,
h2 {
  font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
  font-weight: normal;
  letter-spacing: 1px;
  text-align: center;
}

h1 {
  font-size: 2.1875em;
  text-transform: uppercase;
}

... 其余的CSS ...
```

图 10.14.1 将一级标题显示为全部大写，以便突出它，代码实现参见图 10.14.2

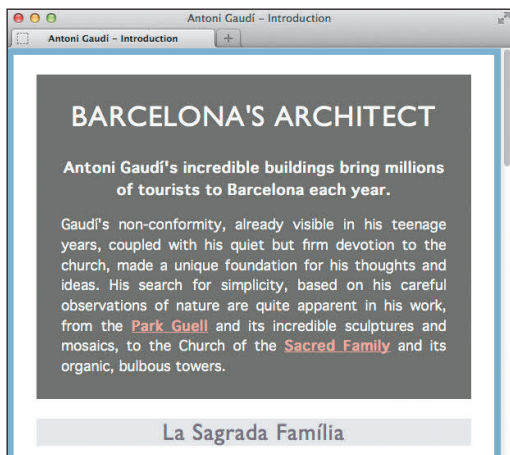


图 10.14.2 现在，一级标题真的很醒目，二级标题没有改变

修改文本大小写的步骤

(1) 输入 `text-transform:`。

(2) 在冒号 (:) 后输入 `capitalize` 让每个单词的首字母大写；

或者输入 `uppercase` 让所有字母大写；

或者输入 `lowercase` 让所有字母小写；

或者输入 `none` 让文本保持本来的样子(可以用来取消继承的值)。

提示 `capitalize` 属性值有它的局限性。它并不了解一门语言里按照惯例首字母不应该大写的单词，它只是简单地让每个单词首字母大写。因此，HTML 中的文本 Jim Rice enters the Hall of Fame 将显示为 Jim Rice Enters The Hall Of Fame。

提示 既然可以改变 HTML 里的文本，为什么还要用 `text-transform` 呢？这是因为，有时，内容是你无法控制的。例如，内容可能存储在数据库里，或者来自另一个网站的新闻源。在这些情况下，只能通过 CSS 控制文本的大小写。搜索引擎通常是按它在 HTML 里输入的样子索引文本的，在搜索结果里显示标准的大小写会更容易阅读。

提示 `lowercase` 属性值可以用来创建特殊的标题样式(想象自己是诗人 e.e. cummings^①)。

提示 `text-transform` 属性是继承的。

10.15 使用小型大写字母

很多字体都有对应的小型大写字母变体，其中一些字母是大写的，但缩小到了小写字母的大小。可以使用 `font-variant` 调用小型大写字母变体(参见图 10.15.1)。

① 美国诗人 E. E. Cummings 的一些作品全部使用小写字母署名，即 e.e. cummings。——译者注

```

...
h1,
h2 {
  font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
  font-weight: normal;
  letter-spacing: 1px;
  text-align: center;
}

h1 {
  font-size: 2.1875em;
  text-transform: uppercase;
}

h2 {
  background-color: #eaebef;
  color: #7d717c;
  font-size: 1.75em;
  font-variant: small-caps;
}

... 其余的CSS ...

```

10.15.1 这里为 h2 设置了 small-caps，如图 10.15.2 所示。不要忘记 font-variant 和 small-caps 中的连字符 (-)



图 10.15.2 现在可以看到 h2 中每个字母的小型大写版本。小型大写字母在不同浏览器中的显示效果稍有不同

1. 使用小型大写字母的方法

输入 font-variant: small-caps。

2. 取消小型大写字母的方法

输入 font-variant: none。

提示 与简单地缩小字号的大写字母相比，小型大写字母显得更为轻巧。

提示 并非所有的字体都有对应的小型大写字母设计。如果浏览器没有找到这种设计，那么它有几种选择：可以简单地缩小大写字母的尺寸来模拟小型大写字母（这会使它们显得有点矮胖），也可以完全忽略小型大写字母，并将它们显示为全部大写（就像前面描述的 text-transform: uppercase 那样）。

提示 font-variant 属性是继承的。

10.16 装饰文本

可以使用 text-decoration 属性装饰文本，如添加下划线或者其他类型的线条。目前，最常用到这个属性的情况是为链接状态添加样式，参见图 10.16.1 和图 10.16.2。

1. 装饰文本的步骤

(1) 输入 text-decoration:。

(2) 在冒号(:)后输入 underline 以添加下划线；

或者输入 overline 以添加上划线；

或者输入 line-through 以添加删除线。

2. 取消文本装饰的方法

输入 text-decoration: none;。

提示 除了 a 元素外，也可以为其他元素应用 text-decoration 属性。

提示 对于正常情况下有装饰的元素(如 a、del、ins)以及从父元素继承了装饰样式的元素，可以取消它们的装饰。

(接左栏)

```
body {
  background-color: #88b2d2;
  font: 100% Geneva, Tahoma, Verdana,
  → sans-serif;
}

h1, h2 {
  font-family: "Gill Sans", "Gill Sans MT",
  → Calibri, sans-serif;
  font-weight: normal;
  letter-spacing: 1px;
  text-align: center;
}

h1 {
  font-size: 2.1875em; /* 35px/16px */
  text-transform: uppercase;
}

h2 {
  background-color: #eaebef;
  color: #7d717c;
  font-size: 1.75em; /* 28px/16px */
  font-variant: small-caps;
}

p { text-align: justify; }

em,
a:link,
.intro .subhead {
  font-weight: bold;
}

.architect {
  background-color: #fff;
  padding: 1.5em 1.75em;
}
```

```
.intro {
  background-color: #686a63;
  color: #fff;
  line-height: 1.45;
  padding: 1px 1.875em .7em;
}

.intro .subhead {
  font-size: 1.125em; /* 18px/16px */
  text-align: center;
}

.intro p {
  font-size: 1.0625em; /* 17px/16px */
}

.project p {
  text-indent: 2em;
  font-size: .9375em; /* 15px/16px */
  line-height: 1.65;
}

a:link {
  color: #e10000;
  text-decoration: none;
}

a:visited { color: #b44f4f; }

a:hover {
  color: #f00;
  text-decoration: underline;
}

.intro a { color: #fdb09d; }
.intro a:hover { color: #fec4b6; }
```

图 10.16.1 这是本章所使用的完整的样式表，包括改变链接样式的 text-decoration，参见图 10.16.2。配套网站上的代码版本包含了更多注释

on his careful observations of nature are quit
from the **Park Guell** and its incredible sculptu
e Church of the **Sacred Family** and its organic,

on his careful observations of nature are quit
from the **Park Guell** and its incredible sculptu
e Church of the **Sacred Family** and its organic,

The Park Guell always ren
The Fountainhead. Gaudí's
community whose residents w

The Park Guell always ren
The Fountainhead. Gaudí's
community whose residents w

图 10.16.2 在最上面的图中，默认状态下所有链接都没有下划线。鼠标悬停在链接上时又会出现下划线，且链接文本的颜色会更亮一些，它们提醒访问者可以进一步采取行动

提示 将链接的下划线去掉固然很好，但是必须用别的方式将链接与周围的内容进行区分（参见图 10.16.1 和图 10.16.2），否则访问者就无法知道它们是可激活的链接。

10.17 设置空白属性

默认情况下，HTML 文档里的多个空格和回车会显示为一个空格，或者被忽略。如果要让浏览器显示这些额外的空格，可以使用 `white-space` 属性，参见图 10.17.1 和图 10.17.2。

```
...

.intro {
  background-color: #686a63;
  color: #fff;
  line-height: 1.45;
  padding: 1px 1.875em .7em;
}

.intro .subhead {
  font-size: 1.125em;

  /* 出于演示目的进行的临时设置 */
  color: lime;
  text-shadow: 3px 2px 2px black;
  white-space: nowrap;
}

... 其余的CSS ...
```

图 10.17.1 这里阻止了 `.subhead` 段落文本断行，是为了突出 `nowrap` 在深色和浅色背景下的影响，参见图 10.17.2。14.5 节会讲解 `text-shadow` 属性

设置空白属性的步骤

- (1) 输入 `white-space:`;
- (2) 输入 `pre`，以让浏览器显示原文本中所有的空格和回车；

或者输入 `nowrap`，确保所有空格不断行，也就是文本全部显示在一行；

或者输入 `normal`，按正常方式处理空格。



图 10.17.2 `.subhead` 段落文本不会断行，即使浏览器窗口太窄不足以显示整行文本。这样，页面会添加一个水平滚动条，以便用户能通过滚动查看其余文本

提示 `white-space` 属性的 `pre` 值的名称来源于 `pre` 元素（参见第 4 章），该元素以等宽字体显示其包含的文本，并保留所有的空格和回车。不过，`white-space: pre;` 不会用等宽字体显示文本。

提示 如果为图 10.17.1 中的 `.intro.subhead` 规则添加 `overflow: hidden; text-overflow: ellipses;`，超出元素盒的文本会显示为省略号。

提示 在具有 `white-space: nowrap` 样式的元素中，可以使用 `br` 元素手动创建换行。尽管如此，还是要尽量避免使用 `br`，除非没有更好的选择，因为这样做会在 HTML 中混合表现，而不是让 CSS 控制它。关于 `br` 元素的更多信息参见 4.16 节。

本章内容

- 开始布局的注意事项
- 建立页面结构
- 在旧版浏览器上为 HTML5 元素添加样式
- 对默认样式进行重置或标准化
- 盒模型
- 控制显示类型和元素的可见性
- 设置元素的高度和宽度
- 添加元素周围的内边距
- 设置元素的边框
- 设置元素周围的外边距
- 使元素浮动
- 控制元素浮动的位置
- 对元素进行相对定位
- 对元素进行绝对定位
- 在栈中定位元素
- 处理溢出
- 垂直对齐元素
- 修改鼠标指针

可以使用 CSS 创建各种各样的布局。本章将演示如何创建一种常见的布局：顶部有一个报头，中间是两栏内容，底部有一个页脚，如图 11.0.1 所示。不过，使用即将讲到的 CSS 属性，还可以创建截然不同的布局。

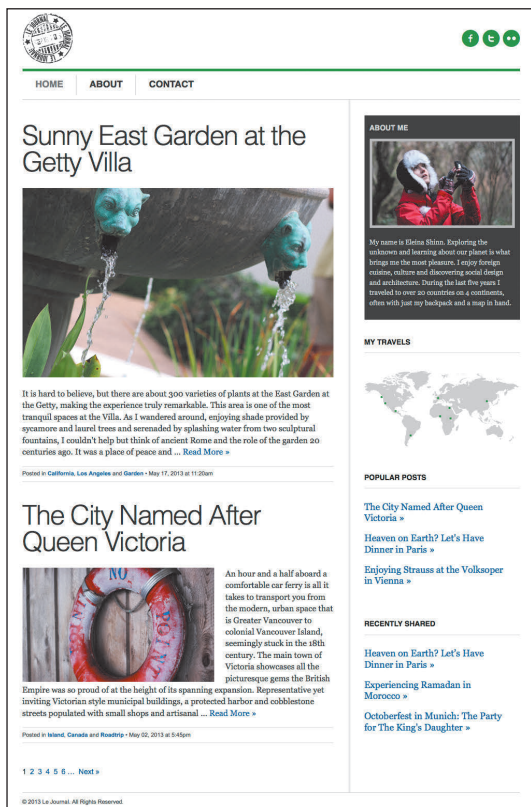


图 11.0.1 这个页面包含两栏、一个页眉和一个页脚，它是用 CSS 进行布局的。CSS 具有强大的改变页面外观的功能。这个页面的设计相当简单，以内容为重点。本章一步步地介绍了如何构建该布局，我们还会在第 12 章将这个页面做成响应式 Web 页面

本章和下一章是相互关联的。为了缓解学习曲线，本章演示了创建固定宽度布局的方法，不过你学到的大部分知识也适用于其他布局。在第 12 章中，我们会学习如何将这个页面做成响应式 Web 页面。

在本章中，不会将 CSS 的每一行都显示出来。例如，大部分文本格式化都在前面完成了。完整的代码见 www.htmlcssvqs.com/8ed/11/finished-page。所有这些文件（尤其是样式表）中都包含了大量的注释，用以对代码进行解释。

11.1 开始布局的注意事项

下面列举的一些要点有助于对网站进行布局，并在发布之前对网站进行调整。

1. 内容与显示分离

之前的章节已经提到过了，不过，由于这是构建网页的根本，因此有必要再次强调一下。作为最佳实践，应始终保持内容（HTML）与显示（CSS）分离。第 8 章介绍了如何通过外部样式表实现这一点。如果对所有的页面都这样做，就可以共享相同的布局和整体样式，而只在具体页面设置差异即可。

2. 布局方法

网站设计主要有两大类型：固定宽度和响应式。

□ 对于固定（fixed）布局，整个页面和每一栏都有基于像素的宽度。顾名思义，无论是使用移动电话和平板电脑等较小的设备查看页面，还是使用桌面浏览器并对窗口进行缩小，它的宽度都不会改变。要知道，你访问过的大部分网站都使用固定布局。在引入响应式 Web 设计之前，这是大多数网

站选用的布局方式。尽管使用固定布局的网站的比例在减少，但这种布局仍很常见，尤其是公司网站和大牌网站。此外，固定布局也是学习 CSS 时最容易掌握的布局方式，这也是本章演示这种布局的原因。

□ 响应式页面也称为流式（fluid 或 liquid）页面，它使用百分数定义宽度，允许页面随显示环境的改变进行放大或缩小。除了具有流动栏，响应式页面还可以根据屏幕尺寸以特定方式调整其设计。例如，可以更改图像大小或者调整每一栏，使其大小更合适。这就可以在使用相同 HTML 的情况下，为移动用户、平板电脑用户和桌面用户定制单独的体验，而不是提供三个独立的网站。

没有一种布局方式可以适用于所有的情景。不过，随着智能手机和平板电脑的广泛出现（更别提我们目前还不知道，但未来一定会出现的各种不同尺寸的设备），有必要将网站做成响应式布局。这也是每天都有大量响应式网站出现的原因。上文已经提到，我们将在下一章学习如何创建响应式网站。

3. 浏览器注意事项

并非所有的访问者都使用同样的浏览器，同样的操作系统，甚至同样的设备访问你的网站。因此，大多数情况下，在将网站放到服务器上发布之前，通常需要在很多浏览器上对页面进行测试。推荐在开发过程中就用几个浏览器对页面定期进行测试，这样，在最后进行全面测试时，碰到的问题就会少一些。关于如何对页面进行测试，以及测试用浏览器的有关信息，参见 20.2 节。

响应式 Web 设计的起源

Ethan Marcotte 创建了术语“响应式 Web 设计”（responsive Web design），并向大家介绍了创建响应式网站的技术。人们首次广泛关注这种方法始于他发表在 A List Apart 上的文章（www.alistapart.com/articles/responsive-web-design/）。他在 *Responsive Web Design* 一书中对此作了更为深入细致的探讨，强烈推荐读者去读一读。

本书第 12 章讲解了 Ethan Marcotte 普及的这种技术，以及 Web 社区其他贡献者对响应式 Web 设计的改进和补充。

11.2 构建页面

CSS 让页面内容富有活力，开发人员或者设计人员的设计技巧大部分都要靠它来体现。可以将 CSS 应用于第 3 章讲到的表现页面主体结构元素的内容容器，以及其中的内容（第 4 章、第 5 章、第 15 ~ 18 章）。但首先我们应该知道，高效网页的核心是结构良好、语义化的 HTML，参见图 11.2.1。

构建页面

(1) 恰当地使用 `article`、`aside`、`nav`、`section`、`header`、`footer` 和 `div` 等元素将页面划分成不同的逻辑区块。根据需要，对它们应用 ARIA 地标角色。上述两点的详细说明见第 3 章。图 11.2.1 创建了一个名为 *Le Journal* 的虚构博客，它的页面结构如下所示。

- 三个 `div`，其中一个将整个页面包起来，另外两个将两部分主体内容区域包起来以便应用样式设计。
- 用作报头的 `header`，包括标识、社交媒体网站链接和主导航。

- 划分为多个博客条目 `section` 元素的 `main` 元素，其中每个 `section` 元素都有自己的页脚。

- 附注栏 `div`（同时使用了 `article` 和 `aside`），提供关于博客作者和右栏（应用 CSS 之后就有了）博客条目的链接。

- 页面级 `footer` 元素，包含版权信息等内容。

(2) 按照一定的顺序放置内容，确保页面在不使用 CSS 的情况下也是合理的（参见图 11.2.2）。例如，首先是报头，接着是主体内容，接着是一个或多个附注栏，最后是页面级的页脚。将最重要的内容放在最上面，对于智能手机和平板电脑等小屏幕用户来说，不用滚动太远就能获取主体内容。此外，搜索引擎“看到”的页面也类似于未应用 CSS 的页面，因此，如果将主体内容提前，搜索引擎就能更好地对网站进行索引。这同样也会让屏幕阅读器用户受益。

(3) 以一致的方式使用标题元素（`h1` ~ `h6`），从而明确地标识页面上这些区块的信息，并对它们按优先级排序。

(4) 使用合适的语义标记剩余的内容，如段落、图和列表。

(5) 如果有必要，使用注释来标识页面上不同的区域及其内容，如图 11.2.1 所示。从代码中可以看出，笔者喜欢使用一种不同的注释格式标记区块的开始（而非结束）。

提示 不一定要在应用 CSS 之前就标记好整个页面。实践中，很少先将一个区块的 HTML 写好，再为其编写一些或全部的 CSS，然后再对下一个区块重复这一过程，等等。处理方式取决于个人习惯。

```

...
<body>
<div class="page">
  <!-- ==== 开始报头 ==== -->
  <header class="masthead"
    → role="banner">
    <p class="logo"><a href="/"><img ...
      → /></a></p>

    <ul class="social-sites">
      ... [社交图片链接] ...
    </ul>

    <nav role="navigation">
      ... [主导航链接列表] ...
    </nav>
  </header>
  <!-- 结束报头 -->

  <div class="container">
    <!-- ==== 开始主体内容 ==== -->
    <main role="main">
      <section class="post">
        <h1>Sunny East Garden at the
          → Getty Villa</h1>

        <img ... class="post-photo-full" />

        <div class="post-blurb">
          <p>It is hard to believe ...</p>
        </div>

        <footer class="footer">
          ... [博客条目页脚] ...
        </footer>
      </section>

      <section class="post">
        <h1>The City Named After Queen
          → Victoria</h1>

        <img ... class="post-photo" />

        <div class="post-blurb">
          <p>An hour and a half aboard
            → ...</p>
        </div>

        <footer class="footer">
          ... [博客条目页脚] ...

```

```

    </footer>
  </section>
  <nav role="navigation">
    <ol class="pagination">
      ... [链接列表项] ...
    </ol>
  </nav>
</main>
<!-- 结束主体内容 -->

<!-- ==== 开始附注栏 ==== -->
<div class="sidebar">
  <article class="about">
    <h2>About Me</h2>
    ...
  </article>

  <div class="mod">
    <h2>My Travels</h2>
    ... [映射图像] ...
  </div>

  <aside class="mod">
    <h2>Popular Posts</h2>
    <ul class="links">
      ... [链接列表项] ...
    </ul>
  </aside>

  <aside class="mod">
    <h2>Recently Shared</h2>
    <ul class="links">
      ... [链接列表项] ...
    </ul>
  </aside>
</div>
<!-- 结束附注栏 -->
</div>
<!-- 结束容器 -->

<!-- ==== 开始页脚 ==== -->
<footer role="contentinfo" class=
  → "footer">
  <p class="legal"><small>&copy; 2013 Le
    → Journal ...</small></p>
</footer>
<!-- 结束页脚 -->
</div>
<!-- 结束页面 -->
</body>
</html>

```

图 11.2.1 这是贯穿本章和下一章的 HTML 页面，这个页面中有四个主要区块（masthead、main、sidebar 和页面 footer）。主体内容和附注栏区域包含在 class="container" 的 div 中。整个页面包含在 class="page" 的 div 中。完整的页面内容请参见 www.htmlcssvqs.com/8ed/11/finished-page，默认情况下页面很普通，但功能完整，参见图 11.2.2



图 11.2.2 这是示例在浏览器中显示的样子，除了浏览器默认样式以外并没有设置其他样式。这个页面是一栏的。由于它的语义结构非常好，页面是完全可用且可理解的，只是有一点朴素

关于语义的注意事项

你可能已经注意到，我在图 11.2.1 中使用了 `section` 元素来标记每个包含部分博客的条目（图 11.2.2 展示了一个）。如果它们是完整的博客条目，我就会使用 `article` 标记它们，就像单独的、完整的博客文章页面那样。对它们使用 `article` 替代 `section` 也没有错，只是代表这一段代码足以成为独立的内容。第 3 章包含了各种使用 `article` 和 `section` 的例子。

11.3 在旧版浏览器中为 HTML5 元素添加样式

我们已经知道，HTML5 引入了一些新的语义化元素，其中大多数在第 3 章和第 4 章作了讲解。在大多数情况下，现代浏览器原生支持这些元素。从样式的角度来说，这意味着浏览器将为这些新的元素应用默认样式，就像它们对待早在这门语言诞生之际就存在的元素那样。例如，`article`、`footer`、`header`、`nav` 以及其他一些元素显示为单独的行，就像 `div`、`blockquote`、`p` 以及其他的在 HTML5 之前的 HTML 版本中称做块级元素的元素。

你可能会想：“对于旧版浏览器呢？这些浏览器出现时还不存在 HTML5 的新元素，我该如何使用这些元素呢？”

值得庆幸的是大多数浏览器允许对它们并不原生支持的元素添加样式。Internet Explorer 8 及以下版本是个例外，不过第 (2) 步中描述了一个简单的解决办法。因此，跟着下面三个简单的步骤，就可以开始为包含 HTML5 元素的页面添加样式了。

针对全部浏览器为 HTML5 新元素添加样式

(1) 将图 11.3.1 中的代码添加到网站的主样式表文件（即所有页面都用到的样式表文件）。注意：如果使用 CSS 重置或 `normalize.css`（参见下一节）可以跳过这一步，它们会包含这里的代码。

(2) 有两种方式可以实现在 Internet Explorer 9 之前的 IE 中为新的 HTML5 元素设置样式。（只能选择一种，不能两种都用。）它们都使用了 HTML5 shiv 这个 JavaScript 文件（参见补充材料“关于 HTML5 shiv”）。

在这两种方法中，相对简单的一种是在每个页面的 `head` 元素（注意不是 `header` 元素）中添加图 11.3.2 中突出显示的代码。这

段代码会从外部网站（googlecode.com）加载该 JavaScript 文件。

更好的方式是将其 JavaScript 文件放在你自己的网站上。这是推荐的做法。这样可以避免在 googlecode.com 上的文件出问题（尽管可能性不大，但仍有可能），网站不会在旧版本的 IE 中崩溃。可以通过 <https://github.com/aFarkas/html5shiv/> 下载 HTML5 shiv 的 ZIP 文件（如图 11.3.3 所示），该地址是存放和维护 HTML5 shiv 代码的地方，再将它放入你的网站（如图 11.3.4 和图 11.3.5 所示）。

```
article, aside, figcaption, figure, footer,
→ header, main, nav, section {
    display: block;
}
```

图 11.3.1 大多数浏览器默认将它们无法识别的元素作为行内元素处理。因此这一小段 CSS 将强制 HTML5 新语义元素显示在单独的行。各浏览器内置的默认样式表均对 div、blockquote、p 等元素声明了 display: block;。更多信息参见 11.6 节

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>Le Journal</title>
<link rel="stylesheet"
→ href="css/lejournal.css" />
<!--[if lt IE 9]>
    <script src="http://html5shiv.
→ googlecode.com/svn/trunk/
→ html5.js"></script>
<![endif]-->
</head>
<body>
...
```

图 11.3.2 script 元素里面是一段所谓的条件注释（conditional comment）。将它放在引用样式表的 link 元素后面。[if lt IE 9] 开头的这段代码表示只有 Internet Explorer 9 以下版本的 IE 才加载这个 JavaScript 文件。通常，最好在页面结束的位置加载 JavaScript（原因见第 19 章）。不过，HTML5 shiv 是少有的必须在 head 中加载 JavaScript 的情况之一。否则，它就不起作用。

(3) 现在，可以随意使用 CSS 添加样式了！

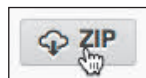


图 11.3.3 将 HTML5 shiv 的 ZIP 文件保存到你的电脑，然后解压缩，得到将会使用的 JavaScript 文件

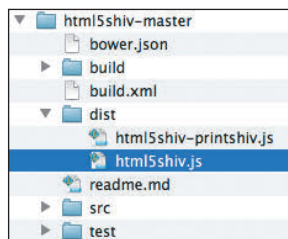


图 11.3.4 在电脑上打开该 ZIP 文件时，会发现其中包含了几个文件夹（如图所示）。你只需要关注 dist 这个文件夹，其中包含了将要用到的 html5shiv.js。将该文件复制到你网站的某个子目录（如名为 js 的目录）里面。此外，还可以使用 html5shiv-printshiv.js。它与 html5shiv.js 基本相同，只是它还可以让 IE8 及更早版本的 IE 在打印时能够正确地处理新的 HTML5 元素

```
...
<link rel="stylesheet"
→ href="css/lejournal.css" />
<!--[if lt IE 9]>
    <script src="js/html5shiv.js">
→ </script>
<![endif]-->
</head>
<body>
...
```

图 11.3.5 突出显示的代码显示在网页的 head 中，引用样式表的 link 元素之后。这里与图 11.3.2 的唯一区别是 src 的值。该值应该引用你复制到网站里的 JavaScript 文件的位置。这段代码假定你将该文件放置在 HTML 页面所在文件夹下的 js 文件夹里（关于 URL，参见 1.7 节）。可以根据需要修改 src 里的路径，从而与个人网站的实际情况相符

关于 HTML5 shiv

与其他主流浏览器不同, Internet Explorer 8 及之前的版本会忽略它们不原生支持的元素的 CSS。HTML5 shiv 是专门用于解决这一问题的一段 JavaScript。(有时也称作 HTML5 shim。)Sjoerd Visscher 发起了一个研究 IE 特性的 Web 社区, HTML5 shiv 就是该社区的几个成员创建的。

HTML5 剃刀也集成进了其他一些 JavaScript 库, 如 Modernizr (www.modernizr.com/)。因此, 如果你在页面中使用了 Modernizr, 就不需要单独加载 HTML5 剃刀了。顺便说一下, 可以跳过前面步骤中的第 (2) 步。Modernizr 是一个非常方便的库, 它可以检测浏览器是否支持 HTML5 和 CSS3 的各种特性。

11.4 对默认样式进行重置或标准化

前面提到, 每个浏览器都有内置的默认样式表。HTML 会遵照该样式表显示, 除非你自己编写的 CSS 覆盖了它们。整体上, 不同浏览器提供的默认样式表是相似的, 但也存在一定的差异。为此, 开发人员在应用他们自己的 CSS 之前, 常常需要抹平这些差异。

抹平差异的方法主要有两种(只使用其中一种即可)。

- ❑ 使用 CSS 重置 (reset) 开始主样式表, 如 Eric Meyer 创建的 Meyer 重置 (<http://meyerweb.com/eric/tools/css/reset/>)。另外还有其他的一些重置样式表。
- ❑ 使用 Nicolas Gallagher 和 Jonathan Neal 创建的 normalize.css 开始主样式表。该样式表位于 <http://necolas.github.com/normalize.css/> (最新版本使用 Download 按钮)。

从上面合适的 URL 中复制 CSS, 并粘贴

到你自己的样式表中。

CSS 重置可以有效地将所有默认样式都设为“零”, 如图 11.4.1 所示。第二种方法, 即 normalize.css, 则采取了不同的方式。它并非对所有样式进行重置, 而是对默认样式进行微调, 这样它们在不同的浏览器中具有相似的外观, 参见图 11.4.2。(重要提醒: 我删除了前面两个图中的大尺寸图像, 以便你能看到更多文本, 重置或者 normalize.css 都不会隐藏图像。)

并非一定要用到这两种方法中的一种。保留浏览器的默认样式, 并自己编写相应的 CSS 也不错。

如果确实要使用 normalize.css 或 CSS 重置, 也不一定要保留它们提供的所有 CSS。它们的样式规则中, 有一些所针对的 HTML 元素是你的网站并未使用的。这时, 就不必在你的样式表中包含这些多余的 CSS。

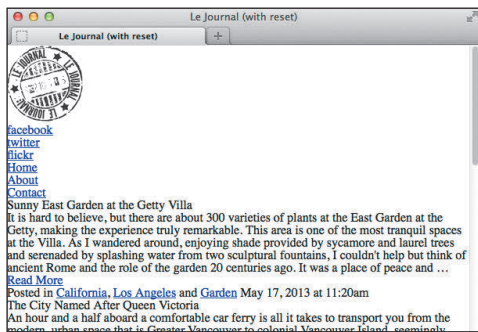


图 11.4.1 这是应用了重置的示例页面。最大的变化是所有的字体大小都变成一样的了, 文本边框和列表样式没了, 所有的外边距和内边距都设为了 0

在本章中, 我使用了 normalize.css 中的一部分代码(如图 11.4.3 所示), 并对文本添加了一些样式, 形成一个初始的页面。在根据本章的介绍继续添加样式之前, 页面的样子如图 11.4.4 所示。可以通过 www.htmlcssvqs.com/8ed/11 找到图 11.4.1、图 11.4.2 和图 11.4.4 对应的完整页面。

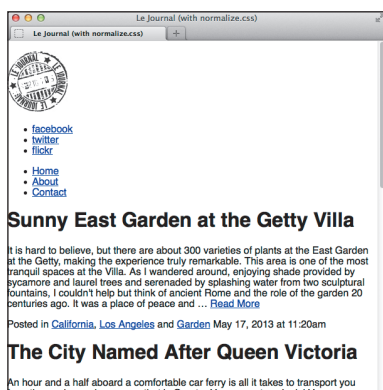


图 11.4.2 这是使用了 normalize.css（而不是重置）的示例页面。它与无样式的默认显示效果很相似，但也有差异。更重要的是，这个版本与使用当今常见的浏览器查看的效果是极为相似的

```

/*! normalize.css v2.1.2 | MIT License |
→ git.io/normalize */

article, aside, figcaption, figure, footer,
→ header, main, nav, section {
    display: block;
}

html {
    -ms-text-size-adjust: 100%;
    -webkit-text-size-adjust: 100%;
}

body {
    margin: 0;
}

a:focus {
    outline: thin dotted;
}

a:active,
a:hover {
    outline: 0;
}

small {
    font-size: 80%;
}

img {
    border: 0;
}

```

图 11.4.3 这是将 normalize.css 中本站不需要的样式规则移除后的一个版本



图 11.4.4 这是经过重置并定义好了文本格式后的示例页面的样子。在本章中，你将以此为起点对页面其余的样式进行定义

11.5 盒模型

CSS 处理网页时，它认为每个元素都包含在一个不可见的盒子里。这就是众所周知的盒模型，这里的盒子由内容区域、内容区域周围的空间（内边距，padding）、内边距的外边缘（边框，border）和边框外面将元素与相邻元素隔开的不可见区域（外边距，margin）构成，参见图 11.5.1。这类似于挂在墙上的带框架的画，其中图画是内容，衬边是内边距，框架是边框，而该画框与相邻画框之间的距离则是外边距。

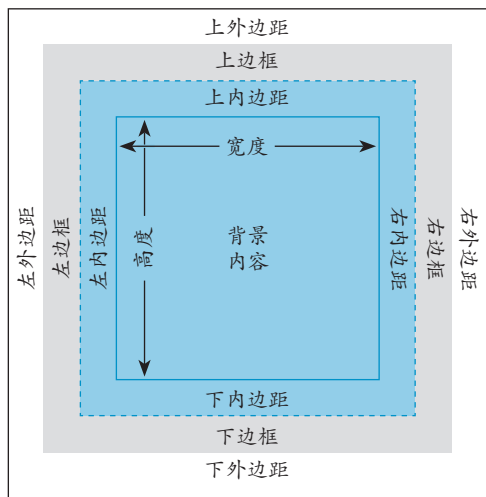


图 11.5.1 每个元素的盒子都有一些决定该元素所占空间及其外观的要素。可以使用 CSS 分别控制各个要素。注意，在默认情况下，宽度和高度仅定义内容区域的尺寸。背景（蓝色区域）会延伸到边框的后面，因此，通常情况下，它仅在内边距所延伸到的区域可见，除非边框是透明的或半透明的

可以使用 CSS 确定每个元素的盒子的外观和位置，并由此控制网页的布局，参见图 11.5.2。本章会深入讲解 `width`、`padding`、`border`、`margin` 和控制盒子外观的其他属性。但首先，理解盒模型的基础知识非常重要。

宽度、高度和盒模型

关于 CSS 的 `width` 属性对元素所显示宽度的影响，有两种处理方式。（不包含任何将其与邻近元素隔开的外边距。）

默认的处理方式（如图 11.5.1 所示）实际上有些有悖于常理。浏览器中元素的宽度与其 `width` 属性值并不一致（除非它没有内边距和边框）。CSS 中的宽度指示的是内边距里内容区域的宽度，如图 11.5.1 表示的那样。而元素在浏览器中的显示宽度则是内容宽度、左右内边距和左右边框的总和。显示高度与之类似，只不过计算的是上下内边距和边框值。



图 11.5.2 示例页面中附注栏的盒模型。记住，每个元素都有自己的盒子。例如，这里显示的 `img` 元素有边框，而它左右的空间则是包含该图像的 `article` 元素的内边距

对大多数代码编写人员来说,另一种方式则更为直观。使用这种方式的话,元素的显示宽度就等于 `width` 属性的值。内容宽度、内边距和边框都包含在里面(如图 11.5.3 所示)。`height` 属性与之类似。要使用这种模式,仅需对元素设置 `box-sizing: border-box;`。

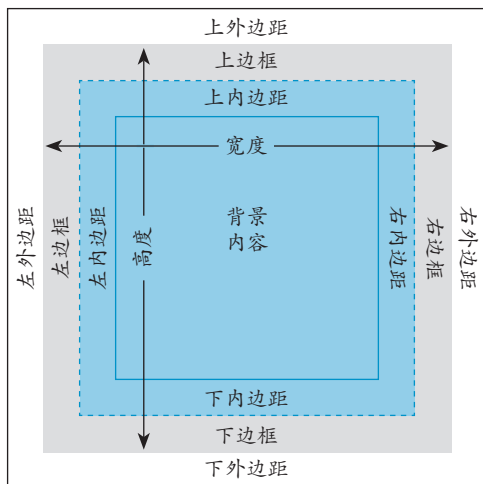


图 11.5.3 如果设置了 `box-sizing: border-box;`, 则宽度和高度就包含了除外边距以外的所有要素。如果不设置这条规则, 则表现为图 11.5.1 所示的方式

以上讲述都比较抽象, 因此不妨看看分别使用两种方式的一个实例(如图 11.5.4 所示)。注意这里显示的 `padding` 和 `border-width` 值在每个边都起作用。例如, `padding: 15px;` 会在水平方向总共产生 30 像素的内边距(左 15 像素, 右 15 像素), 垂直方向也是 30 像素(上 15 元素, 下 15 像素)。

提示 图 11.5.1 是根据 Rich Hauck 的盒模型图绘制的(该图本身是根据 CSS 规范中的插图绘制的)。

提示 关于 `box-sizing: border-box;`、通常不设置 `height` 值的原因等主题, 参见 11.7 节。

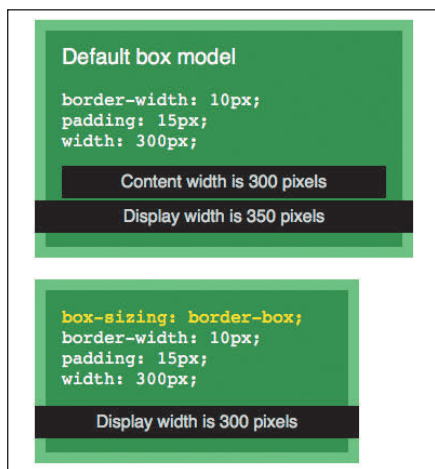


图 11.5.4 两个 `div` 的 `border-width` (浅绿色)、`padding` 和 `width` 值都相同。上图是以默认盒模型(如图 11.5.1 所示)显示的, 下图则是以设置了 `box-sizing: border-box;` 的盒模型(如图 11.5.3 所示)显示的, 其显示宽度与 `width` 属性值相同(都是 300 像素)。两个盒子都设置了 `height: 170px;`(这里没有显示出来), 不过第一个要更高一些, 这是由于上下内边距和边框尺寸增加了显示高度(注意, 黑色的横条不是这些 `div` 的一部分, 因此它们不会影响 `div` 的高度)

11.6 控制元素的显示类型和可见性

正如第 1 章讲到的, 每个元素在默认情况下要么显示在单独的行(如 `h1 ~ h6`、`p` 等), 要么显示在行内(如 `em`、`strong`、`cite` 等), 如图 11.6.1 所示。第 1 章还提到, 前一种元素称为块级元素, 后一种称为行内元素(这是一种非官方的叫法)。

造成这种情况的本质是它们的 `display` 属性, 即块级元素被设置为 `display: block` (对于 `li` 元素为 `display: list-item`), 而行内元素被设置为 `display: inline`。

当然, 使用 CSS 可以改变元素的默认显示类型, 例如将 `block` 改为 `inline` (如

图 11.6.2 所示)，或者反过来（如图 11.6.3 所示）。还有一种混合显示方式称为 `inline-block`，让元素与其他内容出现在同一行，同时具有块级元素的属性（如图 11.6.4 ~ 图 11.6.6 所示）。还可以通过设置 `display` 属性让元素完全不在页面上显示（如图 11.6.7 所示）。

```
<p>This is the <em>first</em> paragraph.</p>
<p>This is the <em>second</em> paragraph.</p>
```

Browser default

This is the *first* paragraph.

This is the *second* paragraph.

图 11.6.1 这些简单嵌套 `em` 元素的段落演示了 `display: block` 和 `display: inline` 的区别

```
p {
    display: inline;
}
```

Paragraphs set to `display: inline`

This is the *first* paragraph. This is the *second* paragraph.

图 11.6.2 两个段落看起来像一个段落

```
em {
    display: block;
}
```

`em` elements set to `display: block`

This is the
first
paragraph.

This is the
second
paragraph.

图 11.6.3 `em` 元素显示在单独的行，就像段落一样

```
em {
    background: lightgreen;
    display: inline-block;
    width: 300px;
}
```

`em` elements set to `display: inline-block`

This is the *first* paragraph.

This is the *second* paragraph.

图 11.6.4 设置为 `inline` 的元素会忽略任何 `width`、`height`、`margin-top` 和 `margin-bottom` 设置。不过，设置为 `inline-block` 的元素可以使用这些属性，如这里的 `em` 可以设置 `width`。因此，如果本例对 `em` 设置 `display: inline`，则会像图 11.6.1 那样显示（除了背景色），无论是否对它指定 `width`（所有的 `display` 类型都接受 `background` 样式。这里将 `em` 的背景设为浅绿色，是为了让 `em` 的宽度变得明显）

```
.social-sites li {
    display: inline;
}

.nav-main li {
    display: inline-block;
}

.nav-main a {
    color: #292929;
    display: inline-block;
    font-family: 'Open Sans', sans-serif;
    font-size: 1.125em;
    font-weight: 700;
    text-transform: uppercase;
}
```

图 11.6.5 这里将主导航链接及其父元素列表项目设置为 `inline-block`，从而方便为其添加上下内边距（参见 11.8 节）。社交网站图标（指这里的 `.social-sites li` 选择的元素）不需要添加块级元素才具有的样式，因此将它们设置为 `inline` 就够了

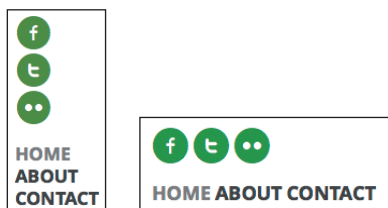


图 11.6.6 这里是一个演示 `display` 属性功能的实例。社交网站图标与主导航都包含在列表项目 (`li`) 里, 因此, 在默认情况下, 它们从上往下排列 (如左图所示)。将它们默认的 `display: list-item`; 覆盖后 (如图 11.6.5 所示), 每个链接列表中的元素都显示在一行内

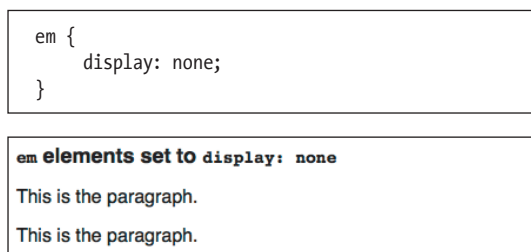


图 11.6.7 `em` 文字既不显示, 也不占据任何视觉空间。在 HTML 中紧跟在它后面的文字会向左移动, 完全无法察觉到 `em` 文字曾占据那个位置

文档流

默认情况下, 元素会按照它们在 HTML 中自上而下出现的次序显示 (这称为文档流, document flow), 并在每个非行内元素的开头和结尾处换行。对于这一点, 图 11.2.1 和图 11.2.2 提供了一个比上面这些例子更长的例子。

1. 指定元素的显示方式

(1) 在样式表规则中, 输入 `display:`。

(2) 输入 `block`, 让元素显示为块级元素 (就像开始新的段落), 参见图 11.6.3;

或者输入 `inline`, 让元素显示为行内元素 (不同于开始新的段落), 参见图 11.6.2;

或者输入 `inline-block`, 让元素显示为行内元素, 同时具有块级元素的特征, 即可以为元素的四条边设置 `width`、`height`、`margin`、`padding` 等属性, 参见图 11.6.4 ~ 图 11.6.6;

或者输入 `none`, 隐藏元素, 并将其从文档流中完全移除 (参见图 11.6.7)。

关于 `display` 属性的其他值, 参见提示。

2. 控制元素可见性

`visibility` 属性的主要目的是控制元素是否可见。与 `display` 属性不同的是, 使用 `visibility` 隐藏元素时, 元素及其内容应该出现的位置会留下一片空白区域 (如图 11.6.8 所示)。隐藏元素的空白区域仍然会在文档流中占据位置。

(1) 在样式表规则中, 输入 `visibility:`。

(2) 输入 `hidden`, 让元素不可见, 但在文档流中保留它 (如图 11.6.8 所示);

或者输入 `visible`, 让元素显示出来。

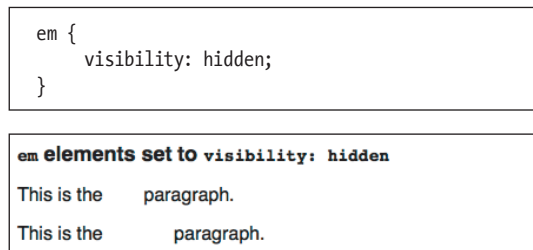


图 11.6.8 隐藏的 `em` 文字所在的位置出现了一块空白, 就像是 `em` 文字设成了白色。空白的尺寸与文本 (或者其他任何隐藏的内容) 的大小完全一致

提示 设置为 `display: inline;` 的元素不接受 `padding` 的设置, 但 `padding-top` 和 `padding-bottom` 会越界进入相邻元素的区域, 而不是局限于该元素本身的空间。可以对元素设置 `background: red;` 以作测试。

提示 第 12 章演示了一个针对窄屏（如手机屏幕）将行内元素设置为 `display: block;`，针对宽屏（如桌面计算机的屏幕）改回 `display: inline;` 的实例。

提示 设置为 `display: none;` 或 `visibility: hidden;` 的元素里的所有内容（包括所有的后代）也会受到影响。例如，如果将一个包含一些 `p`、`figure` 和 `img` 元素的 `article` 元素设置为 `display: none;`，则这些元素都不会显示出来。如果将 `article` 元素设置为 `visibility: hidden;`，就会出现一片空白（很有可能是一大块空白）。

提示 `display` 属性还有其他几个值，但浏览器对它们的支持程度是有差异的。例如，`grid` 和 `flex` 属性提供了一些额外的布局技术，但它们还没有进入最终的规范。更多信息参见 <http://developer.mozilla.org/CSS/display>。

提示 除了 `inherit`，`visibility` 属性还有一个值：`collapse`，用于 `table` 元素的部分内容。关于 `collapse` 的更多信息，参见 <http://developer.mozilla.org/CSS/visibility>。

11.7 设置元素的高度和宽度

可以为很多元素设置高度和宽度，如分块内容、段落、列表项、`div`、图像、`video`、表单元素等（参见图 11.7.1 ~ 图 11.7.3）。

同时，可以为短语内容元素（默认以行内方式显示）设置 `display: block;` 或 `display: inline-block;`，再对它们设置宽度或高度。（关于 `display` 属性的更多信息，参见 11.6 节。）

```
main {
    width: 600px; /* 62.5% = 600px/960px */
}

.sidebar {
    width: 300px; /* 31.25% = 300px/960px */
}
```

图 11.7.1 最终，页面会有两栏，一栏是 `main` 里的内容，一栏是 `class="sidebar"` 的 `div` 里的内容。这些规则分别对它们设置了固定的宽度（如图 11.7.2 和图 11.7.3 所示）

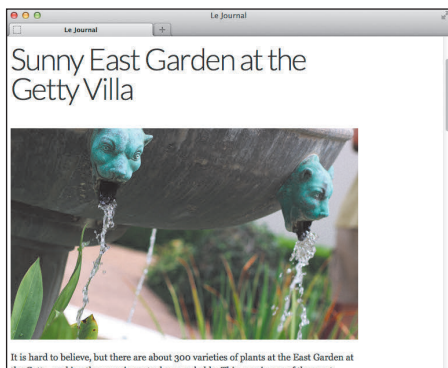


图 11.7.2 这里将 `main` 的宽度设置为 600px 以适应图像的宽度。现在，无论浏览器窗口的宽度是多少，页面主区域的宽度会始终保持 600px 不变。文字会自动折行，不会超过图像的宽度

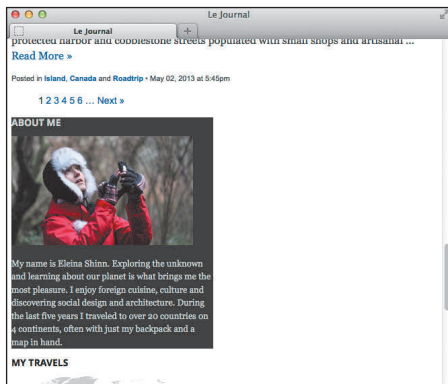


图 11.7.3 侧边栏以 `About Me` 模块开头。目前，我们的页面只有一列，可以看到 `main` 内容的尾部位于侧边栏的上方。还可以看到，侧边栏的宽度（目前设置为 300px）只有 `main` 的宽度的一半

1. 设置元素高度或宽度的步骤

(1) 输入 `width:  $w$` , 其中的 w 是元素内容区域的宽度, 可以表示为长度 (带单位, 如 `px`、`em` 等) 或父元素的百分数。或者使用 `auto` 让浏览器计算宽度 (这是默认值)。

(2) 输入 `height:  $h$` , 其中的 h 是元素内容区域的高度, 只能表示为长度 (带单位, 如 `px`、`em` 等)。或者使用 `auto` 让浏览器计算高度 (这是默认值)。

对定宽页面使用像素, 对响应式 Web 设计使用百分数

本章在介绍网页布局方法时, 对宽度使用像素为单位。在代码注释里, 说明了与像素宽度等价的百分数, 以及如何算出该数值的 (如图 11.7.1)。在第 12 章介绍响应式布局时, 则会使用百分数, 而不是像素。

提示 内边距、边框和外边距都不包含在宽度或高度的数值里 (参见 11.5 节及本节后面的“宽度、外边距和 `auto`”部分)。11.5 节还介绍了对元素设置 `box-sizing: border-box`; (如图 11.7.4 所示), 让宽度和高度包含内边距和边框的方法。

```
* {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
```

图 11.7.4 可以使用 `*` 通配符对所有元素应用 `border-box` 的规则。当然, 也可以单独对元素应用该规则 (用你需要的选择器替换 `*` 即可)。带有 `-webkit-` 和 `-moz-` 这些奇怪前缀的属性可以让这些规则在旧的 Android 和 iOS 设备上起作用, 同时在 Firefox 上也能正常工作。

提示 百分数是相对于父元素的宽度的, 而不是相对于元素本身的原始宽度的。例如, 假设 `div` 宽度为 100 像素, 将其内部的某个元素的宽度设为 70%, 则这个子元素的宽度为 70 像素, 你在下一章还能看到这方面的例子。

提示 不能为显示为行内元素的元素 (如短语内容) 设置高度或宽度, 除非为它们设置 `display: inline-block` 或 `display: block`。关于 `display` 属性的更多信息, 参见 11.6 节。

提示 还有 `min-width`、`min-height`、`max-width` 和 `max-height` 属性用来设置最小和最大尺寸。`max-width` 属性是为流式布局 (下一章会讲到一个流式布局的例子) 设置宽度限制的绝佳工具。在那个例子中, 对包围整个页面的 `.page div` 设置了 `max-width: 960px`;。同时, `main` 和 `.sidebar` 均使用百分数宽度, 因此页面在小一些的屏幕上会较窄, 在大一些的屏幕上则较宽, 但不会超过 960 像素, 哪怕屏幕非常大。另外, 参考补充材料“为什么 `min-height` 通常比 `height` 更适用”。

提示 `width` 和 `height` 不是继承的。

2. 宽度、外边距和 `auto`

如果不显式设置宽度和高度, 浏览器就会使用 `auto`。对于大多数默认显示为块级元素的元素, `width` 的 `auto` 值是由包含块的宽度减去元素的内边距、边框和外边距计算出来的, 参见图 11.7.5 和图 11.7.6。简单说来, 包含块的宽度指的是父元素给元素留出的宽度。

```
div { /* 包含块 */
    background: khaki; /* tan */
    width: 300px;
}

p,
.second {
    background: white;
    border: 3px solid royalblue;
    margin: 10px;
    padding: 8px;
}

.second { /* 第二个段落 */
    border-color: orangered;
    width: 200px;
}
```

图 11.7.5 在这个例子中，将父元素 div 的 width 设为 300 像素。这将是我们的包含块。然后，两个段落各个边上都有 10 像素的外边距、8 像素的内边距及 3 像素宽的边框。第一个段落的宽度是自动设置的，因为 auto 是 width 的默认值（除非指定其他的值）。第二个段落（在 HTML 中带有 class="second"）设为 200px

例如，位于 200 像素宽的空间里的 p 元素的 auto 宽度为 200 像素。但如果为其设置 padding-right: 10px，则 auto 宽度为 190 像素。

对于图像这样的元素，auto 宽度等于它们固有的宽度，即外部文件的实际尺寸（如本章示例页面中最大的图像，大小为 600×365）。行内元素会完全忽略 width 属性（即不能为 em、cite 等元素设置宽度，除非将它们设置为 display: inline-block 或 display: block）。关于 display，参见 11.6 节。

如果手动设置 width、margin-left 和 margin-right 值，但这些值加上边框和内边距的值不等于包含块的大小，有的属性就需要作出让步。实际上，浏览器会优先考虑 width，确保右边的间距大于你设置的 margin-right（参见图 11.7.6 中底部的段落）。

如果手动设置 width，但将某个外边距设为 auto，那么这个外边距将进行伸缩以弥补不足的部分。

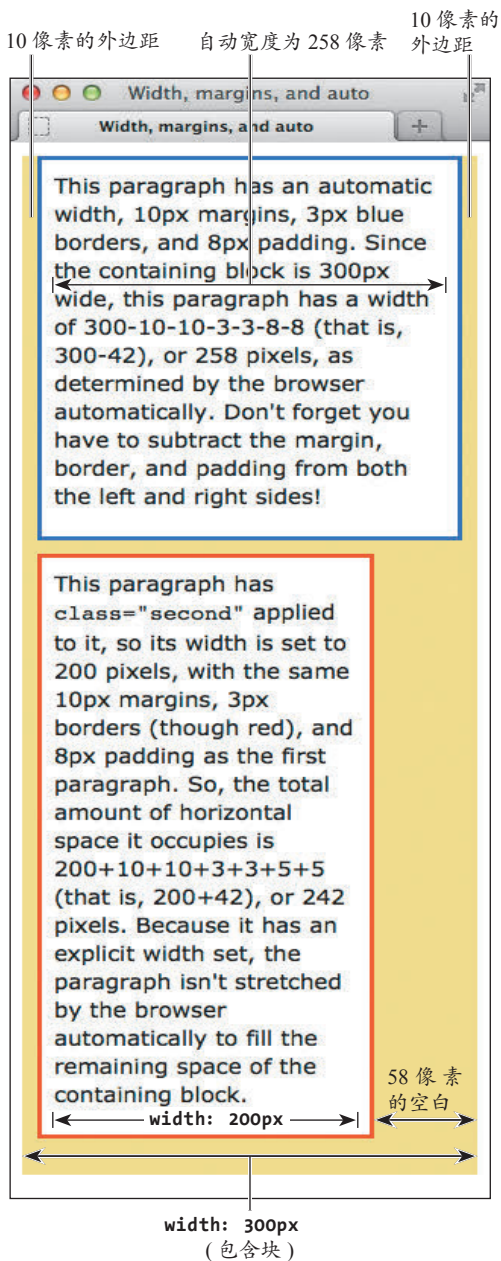


图 11.7.6 如果 width 是 auto（就像第一个段落那样），它的值就是由包含块（黄褐色区域）的宽度减去元素自身的外边距、内边距和边框计算出来的。如果 width 是手动设置的（就像最后一个段落那样），则右边边距常常会进行调整以填补不足的空间

不过,如果手动设置 `width`,并将左右外边距都设为 `auto`,那么两个外边距就将设为相等的最大值,这样会导致元素居中。例如, `.page { margin: 0 auto; }` 会使对应的元素在页面居中显示。这也是我对示例页面所做的设置,参见图 11.7.6 和图 11.10.7。

为什么 `min-height` 通常比 `height` 更适用

除非你确定元素的内容不会变得更高,最好避免在样式表中指定高度。在大多数情况下,可以让内容和浏览器自动控制高度。这可以让内容在浏览器和设备中根据需要进行流动。

如果设置了高度,随着内容变多,它们有可能撑破元素的盒子,这可能是你预期之外的。在这种情况下,符合标准的浏览器不会自动扩大高度,它们在你指定高度时采用了这个指令,并一直坚持下去。(了解相关示例请参考 11.16 节。)

不过,如果希望元素至少具有某一特定高度,可以设置 `min-height`。如果内容日后变多,元素的高度会自动按需增长。这是 `height` 与 `min-height` 的区别, `width` 和 `min-width` 也是类似的。

或许你还有所疑虑,实际上,很多原因都会导致内容增长。你的内容或许来自数据库、第三方源或由用户生成的内容。同时,访问者可能增大其浏览器的字体大小,覆盖你指定的样式。

11.8 在元素周围添加内边距

顾名思义,内边距就是元素内容周围、边框以内的空间。根据上文的类比,内边距就像是图画(内容)与画框(边框)之间的衬边。可以改变内边距的厚度(如图 11.8.1

到图 11.8.4 所示),不能改变它的颜色或纹理。显示在内边距区域的颜色和纹理是元素的背景,是通过 `background`、`background-color` (参见图 11.8.1) 或者 `background-image` 设置的。

```
.about {
  background-color: #2b2b2b;
  padding: .3125em .625em .625em;
}

/*
跟padding: 5px 10px 10px;
是一样的,因为字体大小是16px,且
.3125 = 5px/16px
.625 = 10px/16px
*/
```

图 11.8.1 同外边距类似,如果为 `padding` 设置四个值,那么它们分别表示上、右、下、左(按时钟顺序)内边距。因此在这里四个边都有内边距(实现结果如图 11.8.2 所示)



图 11.8.2 如果不设置内边距(左),内容就会贴到边界。添加内边距后,元素里的内容外围就会产生一些空白(右)。同时,在内边距的区域会显示背景色。(如果添加 `.about { border: 1px solid red; }`,则整个盒子的外围会显示一条红色的边框,边框位于内边距的外面。)可以看到,这个模块的上边和下边的空白要大于内边距指定的大小。这是由于浏览器为标题(ABOUT ME)设置的默认上外边距和为段落(如图像下面的段落)设置的默认下外边距造成的。它们外围的 `.about` 盒子的内边距让这些外边距在盒子内部生效

```
.nav-main a {
  color: #292929;
  display: inline-block;
  font-family: 'Open Sans', sans-serif;
  font-size: 1.125em; /* 18px/16px */
  font-weight: 700;
  padding: .5em 1.15em .5em 1.4em;
  text-transform: uppercase;
}

/*
跟padding: 9px 20.7px 9px 25.2px;
一样, 因为字体大小为18px, 且
.5   = 9px/18px
1.15 = 20.7px/18px
1.4   = 25.2px/18px
*/
```

图 11.8.3 现在, 我们为主导航里的每个链接都添加了内边距。在下一部分为其添加边框之后, 上下内边距会更加明显

HOME ABOUT CONTACT

HOME ABOUT CONTACT

图 11.8.4 这些链接之前都挤在一起(上), 添加了内边距后, 它们就有“呼吸”的空间了(下)

padding 的简记法

同 border 和 margin 属性一样, padding 也可以使用简记法, 从而不必使用 padding-top、padding-right 等属性为每个边都单独设定内边距。

- padding: 5px;——使用一个值, 这个值就会应用于全部四个边。
- padding: 5px 9px;——使用两个值, 则前一个值会应用于上下两边, 后一个值会应用于左右两边。

□ padding: 5px 9px 11px;——使用三个值, 则第一个值会应用于上边, 第二个值会应用于左右两边, 第三个值会应用于下边(如图 11.8.1 和图 11.8.2 所示)。

□ padding: 5px 9px 11px 0;——使用四个值, 它们会按照时钟顺序, 依次应用于上、右、下、左四个边(如图 11.8.3 和图 11.8.4 所示)。

内边距的值可以使用像素、百分数、em 或 rem 的组合。

在元素周围添加内边距的方法

输入 padding: x;, 这里的 x 是要添加的空间量, 表示为带单位(通常为 em 或像素)的长度或父元素宽度的百分数(如 20%)。

也可以输入 padding-top: x;、padding-right: x;、padding-bottom: x; 或者 padding-left: x; 单独为一个边添加内边距, 参见图 11.8.5 和图 11.8.6。

```
.links {
  padding-left: 0;
  width: 270px; /* 90% = 270px/300px */
}
```

图 11.8.5 侧边栏包含两组链接, 每组都包含在一个带 class="links" 的无序列表(ul)里。这里将内边距设为 0, 以覆盖浏览器默认的 40px(如图 11.8.6 所示)。宽度的样式能确保链接在距侧边栏(先前设定为 300 像素宽)右侧 30 像素的地方折行(关于列表, 参见第 15 章; 关于列表的默认左内边距, 重点参考 15.3 节)

提示 关于如何确定图 11.8.1 和图 11.8.3 中的 em 值的信息参见 11.10 节补充材料。

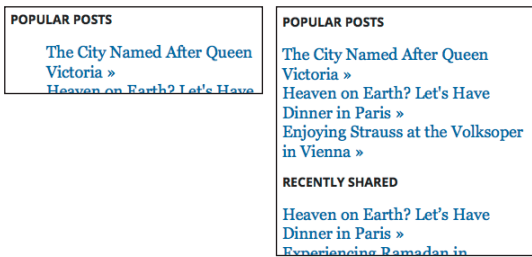


图 11.8.6 列表默认的缩进样式很难看（上），现在，由于取消了内边距，链接与标题对齐了（下）

提示 默认情况下，元素的 width 和 height 不包含 padding 的大小。参见前一节的“宽度、外边距和 auto”部分及 11.5 节（其中还讲解了如何覆盖默认的盒模型设置）。

提示 关于背景样式的设置，参见 10.10 节、14.6 节和 14.7 节。

提示 内边距不是继承的。

11.9 设置边框

可以在元素周围创建边框（参见图 11.9.1 和图 11.9.2），或针对元素的某一边设置边框（参见图 11.9.3 ~ 图 11.9.10），并设置它的厚度、风格和颜色。如果指定了内边距（参见 11.8 节），边框将包围在元素内容和内边距的外面（参见图 11.9.5 ~ 图 11.9.10）。图 11.9.11 和图 11.9.12 展示了我们可用的所有边框样式。

```
.about img {  
    border: 5px solid #bebebe;  
}
```

图 11.9.1 边框可以应用于任何元素，包括图像。如果希望对一个以上的元素设置相同的边框样式，最好引入一个类，从而可以复用，例如 .frame { border: 5px solid #bebebe; }

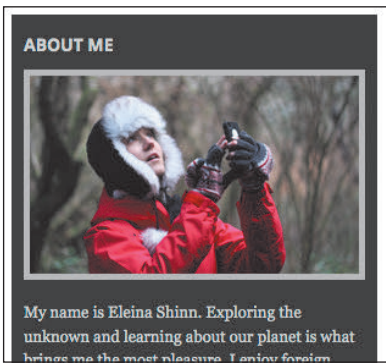


图 11.9.2 边框将图像很好地框起来了

```
.nav-main {  
    /* 绿色 */  
    border-top: 5px solid #019443;  
    /* 灰色 */  
    border-bottom: 1px solid #c8c8c8;  
}  
  
.nav-main li {  
    border-left: 1px solid #c8c8c8;  
    display: inline-block;  
}  
  
.nav-main li:first-child {  
    border-left: none;  
}
```

图 11.9.3 这些简单的边框对页面的主导航和头部起到了下定义的作用（如图 11.9.4 所示）。作为一种常见的做法，每个导航链接都包含在一个列表项目（li）里。注意，这里对第一个 li 使用了 border-left: none;，以取消其左边框（即改回至它的默认样式）

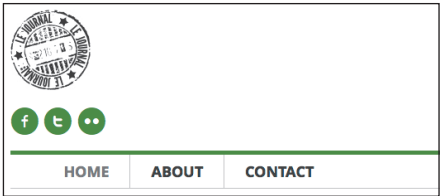


图 11.9.4 边框让主导航更加突出，让导航链接彼此分开，还能有效地区分页面头部和它下面的内容（这里没有显示出来）。水平的边框会延伸至页面的整个宽度

```
.nav-main {
    padding: .45em 0 .5em; /* 7px 0 8px */
}
```

图 11.9.5 导航容器顶部和底部的一点点儿内边距使格局发生了微妙的变化

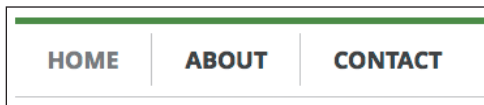


图 11.9.6 内边距将水平线和垂直线分开了

1. 定义边框风格

输入 `border-style: type`，其中的 *type* 可以是 `none`、`dotted`（点线）、`dashed`（虚线）、`solid`（实线）、`double`（双线）、`groove`（槽线）、`ridge`（脊线）、`inset`（凹边）或 `outset`（凸边）。

2. 设置边框宽度

输入 `border-width: n`，这里的 *n* 是需要的宽度（含单位，如 `4px`）。

3. 设置边框颜色

输入 `border-color: color`，这里的 *color* 是颜色名称、十六进制值或 RGB、HSL、RGBA、HSLA 颜色（参见 7.5 节的“CSS 颜色”）。

4. 使用简记法同时设置多个边框属性

(1) 输入 `border`。

(2) 如果需要，输入 `-top`、`-right`、`-bottom` 或 `-left` 将边框效果限制在某一条边上。

(3) 如果需要，输入 `-property`，这里的 *property* 可以是 `style`（风格）、`width`（宽度）或 `color`（颜色），仅使用单个属性设置边框。

(4) 输入冒号（`:`）。

(5) 输入恰当的值（如前面介绍的那样）。如果跳过了第 (3) 步，则可以指定所有三种

边框属性（如 `border: 1px solid` 和 `border-right: 2px dashed green;`）。如果在第 (3) 步中指定了一种属性，则只能使用这种属性可以接受的值（如 `border-right-style: dotted;`）。

提示 默认情况下，Internet Explorer 会对作为链接的图像添加一个蓝色的边框。为了取消这一样式，可以在样式表中添加 `img { border: none; }`。这里，示例页面也加入了这一规则，因为网站的标识就位于一个指向主页的链接里面。（这个例子是主页，但整个网站的页面都使用相同的标识代码。）

提示 `border` 简写属性及各个边框属性（`border-width`、`border-style` 和 `border-color`）均可接受一至四个值。如果使用一个值，那么它会应用于全部四个边。如果使用两个值，那么前一个值会应用于上下两边，后一个值会应用于左右两边。如果使用三个值，那么第一个值会应用于上边，第二个值会应用于左右两边，第三个值会应用于下边。如果使用四个值，那么它们会按照时钟顺序，依次应用于上、右、下、左四个边。

提示 要让边框显示出来，至少必须定义边框样式。如果没有定义样式，就不会显示边框。边框样式的默认值是 `border-style: none`。

提示 如果使用简记法，如 `border` 或 `border-left` 等，则未提供的属性将显示其默认值。因此，`border: 1px black;` 相当于 `border: 1px black none;`，这意味着不会显示边框（即便在之前使用 `border-style` 指定了边框样式）。

提示 边框的默认颜色是元素的 `color` 属性的值（参见 10.9 节）。

提示 `border` 属性可用于表格及其单元格，相关示例参见 18.1 节。

提示 默认情况下，元素的宽度和高度设置不包括边框大小。11.7 节第一条提示解释了如何覆盖默认设置。

提示 CSS3 引入了 `border-image` 属性。除 Internet Explorer 以外，浏览器对它的支持程度较好（参见 <http://caniuse.com/#search=border-image>）。关于 `border-image`，参见 www.sitepoint.com/css3-border-image 和 css-tricks.com/。

提示 边框不是继承的。

```
.about h2,
.mod h2 {
    font-size: 0.875em;
}

.about h2,
.mod h2,
.nav-main a {
    font-family: 'Open Sans', sans-serif;
    font-weight: 700;
    text-transform: uppercase;
}

.mod h2 {
    border-bottom: 1px solid #dbdbdb;
    padding-bottom: 0.75em;
}
```

图 11.9.7 我在一条声明中为多个标题添加了边框，效果参见图 11.9.8（名为 `mod` 的类代表了页面中的一个通用模块）

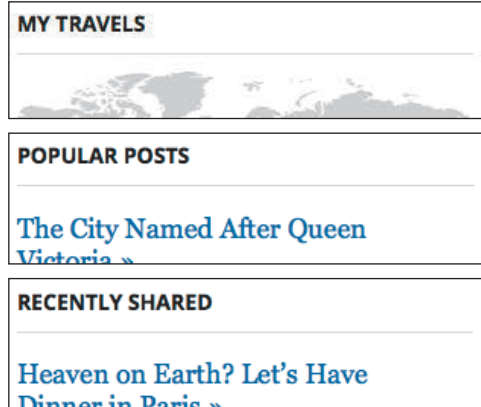


图 11.9.8 我们对附注栏中的三个标题也应用了同样的样式，底部的内边距确保了边框和标题文本之间拉开了一点距离

```
.footer p {
    font-size: .6875em; /* 11px/16px */
}

.post-footer {
    border-bottom: 1px solid #dbdbdb;
    border-top: 1px solid #dbdbdb;
    padding-bottom: .7em; /* 7.7px/11px */
    padding-top: .7em;
}
```

图 11.9.9 除了都有一个上边框，这里还对每篇博客文章下方的页脚应用了相似的效果（如图 11.9.8 所示）。每个这样的页脚都是一个带 `class="post-footer"` 的段落

Posted in [California](#), [Los Angeles](#) and [Garden](#) • May 17, 2013 at 11:20am

Posted in [California](#), [Los Angeles](#) and [Garden](#) • May 17, 2013 at 11:20am

图 11.9.10 如果没有上下内边框（如图 11.9.7 所示），则边框几乎会贴着文字（上）。添加了内边框后，效果看起来好多了（下）

```

p {
    border: 10px solid red;
    padding: 15px;
}

.ddd {
    border-width: 4px;
    border-style: dotted dashed double;
}

.ridge {
    border-style: ridge;
    border-color: orange;
}

.groove {
    border-style: groove;
    border-color: purple;
}

.inset {
    border-style: inset;
    border-color: blue;
}

.outset {
    border-style: outset;
    border-color: green;
}

```

图 11.9.11 在这个例子中，我为每个段落设置了内边距和默认边框。然后，对第一个段落设置了应用于四个边的边框宽度，并为每个边设置了不同的样式。对于余下的四个段落，使用一条声明并在其中重复 10px 比通过两个属性分开设置样式和颜色要方便一些

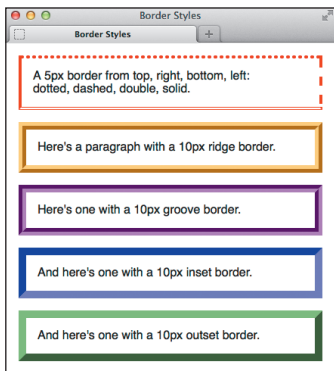


图 11.9.12 各浏览器对边框样式的处理方式并不完全相同。这是在 Firefox 中显示的不同边框样式，从中可以了解到不同样式的差异。另见彩插

11.10 设置元素周围的外边距

外边距是元素与相邻元素之间的透明空间（参见图 11.10.1 和图 11.10.2）。关于它与元素边框和内边距的关系，参见 11.5 节。

```

h1 {
    color: #212121;
    font-family: "Lato", sans-serif;
    font-size: 3.25em; /* 36px/16px */
    font-weight: 300;
    letter-spacing: -2px;
    line-height: .975;
    margin-bottom: .4125em; /* 21.45px */
}

```

图 11.10.1 我用自己的设置覆盖了默认的下外边距设置，关于使用 em 设置外边距的说明参见补充材料

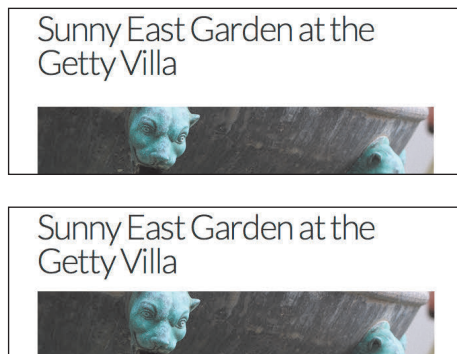


图 11.10.2 博客条目标题和图像之间的默认空间太大了（上图），重新设置后两者更为紧凑了，看起来好很多（下图）

设置元素外边距的方法

输入 `margin: x`，其中的 `x` 是要添加的空间量，可以表示为长度、相对父元素宽度的百分数或 `auto`。

也可以输入 `margin-top: x; margin-right: x; margin-bottom: x; margin-left: x`；（参见图 11.10.1 和图 11.10.3）或者 `margin-left: x`；为元素的一条边应用外边距。

```
.map {
  margin: 1.4375em 0 .8125em;
  /* 23px 0 13px */
}

.links {
  margin: 1.5em 0 4.125em;
  /* 24px 0 66px */
  padding: 0;
}

.links li {
  margin-bottom: 1.1em;
}
```

图 11.10.3 以下几条样式规则处理几个元素之间的间距，第一条规则为地图周围留出了一些空间，如图 11.10.4 所示，后面两条规则影响了附注栏中的链接，如图 11.10.5 所示

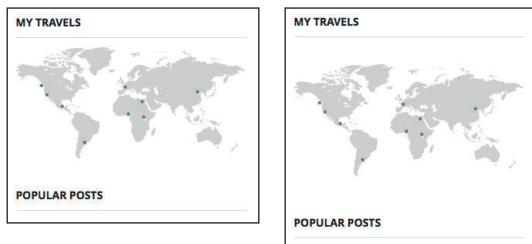


图 11.10.4 地图和两个标题之间显得很挤(左图)，外边距解决了这一问题(右图)

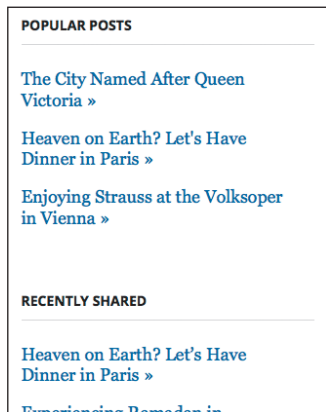


图 11.10.5 每组链接之间的空隙更大了，不同链接之间的空隙也变得更大(关于之前的样子，参见 11.8 节的图 11.8.6)

提示 如果对 margin 使用一个值，这个值就会应用于全部四个边。如果使用两个值(参见图 11.10.6)，那么前一个值会应用于上下两边，后一个值会应用于左右两边。如果使用三个值(参见图 11.10.3)，那么第一个值会应用于上边，第二个值会应用于左右两边，第三个值会应用于下边。如果使用四个值，那么它们会按照时钟顺序，依次应用于上、右、下、左四个边。

提示 margin 属性的 auto 值依赖于 width 属性的值(如图 11.10.6 所示)，参见 11.7 节。

提示 如果元素位于另一个元素的上面，对于相互接触的两个 margin(即元素相互接触的下外边距和上外边距)，仅使用其中较大的一个，另一个外边距会被叠加。左右外边距不叠加。

提示 确定 em 值时，可能需要在手头准备一个计算器。此外，还可以使用 <http://pxtoem.com> 这样的工具，它们可以帮你计算 em 值。

提示 外边距不是继承的。

```
.page {
  border: 1px solid red; /* 临时设置 */
  margin: 0 auto;
  width: 960px;
}
```

图 11.10.6 让网页的主体内容在浏览器里水平居中显示是一种常见的样式，要实现这种样式其实并不难。首先，对内容的容器添加 width(也可以用 max-width)。然后将其左右外边距都设为 auto。这样做的话，浏览器会基于浏览器宽度和容器宽度之差计算这些外边距。在本例中，带 class="page" 的 div 包含了整个页面。

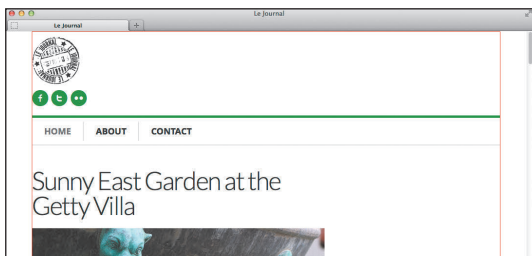


图 11.10.7 页面内的内容可能是左对齐的，但页面本身却位于浏览器窗口的中间。这里临时为 .page div 添加了一个红色的边框，好让效果更为明显。这里还将该 div 的宽度设为 960px，从而为侧边框腾出空间。我们将在下一节学习如果将附注栏放到那个位置

理解内边距和外边距的 em 值

当 em 值用于内边距和外边距时，它的值是相对于元素的字体大小的，而不是相对于父元素的字体大小的（你可能会这样认为），具体参见 10.6 节。计算外边距或内边距的 em 值的公式为：

要指定的字体大小 / 元素的字体大小 = 值

设想一个简单的例子，其中的段落定义了 `p { font-size: 14px; padding: .5em; }`。那么，它在四个边的内边距都是 7 像素，因为 $7/14 = 0.5$ 。如果将字体大小改为 20px，内边距就会自动变成 10 像素（ $10/20 = 0.5$ ）。这就是 em 这样的相对单位的作用——随着布局放大或缩小，布局的比例始终不变。

外边距也是类似的。如图 11.10.1 所示的，h1 的字体大小大约是 36 像素（ $3.25em = 36px/16px$ ）。因此， $0.4125em$ 的下外边距大约为 21.45 像素（ $21.45/36 = 0.4125$ ）。

像 em、百分数以及 rem 这样的相对单位在响应式网页中的优势更为明显（响应式 Web 设计将在下一章进行介绍）。不过它们在固定宽度的布局中也有作用。Trent Walton 介绍了一个强大的使用相对单位的例子，见 <http://trentwalton.com/2013/01/07/flexible-foundations>。

对于内边距和外边距的设置，建议使用相对单位，就像本书的例子中演示的那样。我知道，在刚刚开始学习网站构建的时候，需要一段时间去习惯这种做法。这也是我既提供 em 值又提供像素值的原因。刚开始的时候，你可能会感觉对内边距和外边距使用像素更舒服一些。这样做没有问题。你可以在自己认为合适的时候转向 em。

这里需要指出一条简短的说明：如果要对内边距和外边距使用百分数，通常不会将它们用于上下边距的值，因为这样的值是基于其包含块的宽度的。

11.11 使元素浮动

可以通过 float 属性使元素浮动在文本或其他元素上。可以使用这种技术让文本环绕在图像（参见图 11.11.1 ~ 图 11.11.3）或者

其他元素周围，也可以使用这种技术创建多栏布局等，如图 11.11.4 和图 11.11.5 所示。



An hour and a half aboard a comfortable car ferry is all it takes to transport you from the modern, urban space that is Greater Vancouver to colonial Vancouver Island, seemingly stuck in the 18th century. The main town of Victoria showcases all the

图 11.11.1 这是一篇博客文章的片段。默认情况下，段落显示在图像的下面。段落没有设置宽度，文字折行是因为段落（以及图像）位于 main 元素中，而该元素在先前设置了 600 像素的宽度（又如图 11.11.3 所示）。图像的宽度（370 像素）是在 HTML 中指定的。如何让文字围绕在图像周围呢？

```
.post-photo {
  float: left;
  margin-bottom: 2px;
  margin-right: 22px;
}
```

图 11.11.2 这很简单！只要让一个设置了宽度的元素浮动，在正常情况下显示在它下面的内容就会流动到它的周围（前提是这些内容没有设置宽度）。这里的图像（带有 class="post-photo"）和段落就满足这些条件，因此会形成如图 11.11.3 所示的效果。外边距的设置让图像和文字之间有了一些空隙

The City Named After Queen Victoria



An hour and a half aboard a comfortable car ferry is all it takes to transport you from the modern, urban space that is Greater Vancouver to colonial Vancouver Island, seemingly stuck in the 18th century. The main town of Victoria showcases all the picturesque gems the British

Empire was so proud of at the height of its spanning expansion. Representative yet inviting Victorian style municipal buildings, a protected harbor and cobblestone streets populated with small shops and artisanal ... [Read More »](#)

Posted in [Island, Canada](#) and [Roadtrip](#) • May 02, 2013 at 5:45pm

图 11.11.3 图像向左浮动时，如果文本的高度大于图像的高度，文本会环绕在图像周围。这个例子使用的是图像，但你也可以使用同样的方法让 figure、aside 或其他元素浮动

```
main {
  float: left;
  width: 600px; /* 62.5% = 600px/960px */
}

.sidebar {
  float: right;
  margin-top: 1.875em; /* 30px/16px */
  width: 300px; /* 31.25% = 300px/960px */
}
```

图 11.11.4 可以用类似的方法让两个元素并排在一起，如主要内容和附注栏。它们都指定了宽度，因此都可以进行浮动

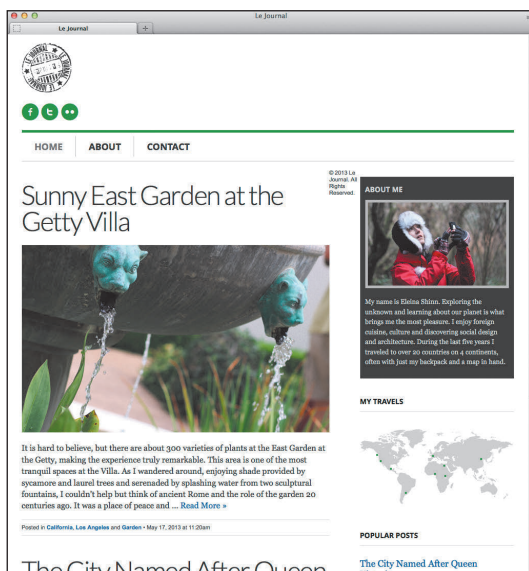


图 11.11.5 瞧，一个两栏布局形成了！不过，两栏之间的东西是什么呢？让我们仔细看看（如图 11.11.6 所示）……

让文本环绕元素

(1) 输入 float:。

(2) 输入 left，让元素浮动到左边，其他内容围绕在它右边；

或者输入 right，让元素浮动到右边，其他内容围绕在它左边；

或者输入 none，让元素完全不浮动。（none

是默认值，因此只有在特定情况下需要覆盖某个浮动元素的样式规则时才需要用到它。）

(3) 使用 `width` 属性显式地设置元素的宽度（参见 11.7 节），以便有空间放置围绕它的内容。



图 11.11.6 这是页脚！它怎么会在那里呢？首先，记住我们的页面是 960 像素宽，`main` 是 600 像素宽，而附注栏是 300 像素宽。两栏向相反的方向浮动，中间就会留下 60 像素的空隙。其次，如果你看看 HTML，你会发现 `main` 的代码就在附注栏的前面，而附注栏就在页脚前面。由于两栏都是浮动的，因此页脚的内容就会流动到它们之间，就像图 11.11.3 中的段落文字环绕图像一样。后面将介绍如何让页脚回到下面的位置

提示 记住，你选择的方向是应用于需要浮动的元素的，而不是应用于环绕它的元素的。在使用 `float:left` 时，页面的其他部分围绕在右边，反之亦然。

提示 `float` 属性不是继承的。

11.12 控制元素浮动的位置

我们可以控制浮动什么时候停止，不过在介绍这些内容之前，需要进一步理解 `float` 属性是如何影响页面的。

考虑图 11.12.1 中的 HTML。这是一段博客文章的介绍（不过没有标题）。可以看

到，`section` 元素包含了所有相关的内容。假设该元素的样式与之前的一样，只是添加了一个花哨的背景色，从而突出其高度（如图 11.12.2 所示）。

```
<section class="post">
  <img ... class="post-photo" />

  <div class="post-blurb">
    <p>An hour and a half ...</p>
  </div>

  <footer class="footer">
    <p class="post-footer">Posted ...</p>
  </footer>
</section>
```

图 11.12.1 在这个简单的结构中，博客文章的所有介绍性内容都放在一个 `section` 中。在贯穿本章的例子中，`<h1>The City Named After Queen Victoria</h1>` 位于 `section` 中 `img` 前的位置

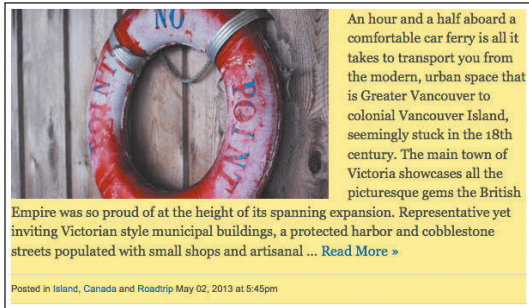


图 11.12.2 图像应用了 `float: left;`，这样文字就环绕在了它旁边。`section` 元素（黄色区域）的高度看起来好像受到了图像的影响，但事实并非如此

浮动的元素对文档流的影响与非浮动元素的影响是不同的。如图 11.12.2 所示，图像等浮动元素会让接下来的内容环绕在它周围。不过，它并不会影响父元素或其他祖先元素的高度，因此从这一点来说，它不属于文档流的一部分。

为了演示这一点，我移除了博客摘要中的一部分文字（如图 11.12.3 所示）。你或许

会对此感到惊讶——尽管图像是 `section` 的子元素，但 `section` 却比图像更矮一些！只有普通文档流中的内容会影响父元素的高度。在这个例子中，这样的内容包括博客摘要文字和博客文章的页脚，但不包括图像。



图 11.12.3 现在你可以清晰地看到浮动元素是如何影响文档流的了——它完全不影响父元素的高度

页脚为什么会显示在这里呢？它之前的内容变短之后，它移到了图像的旁边。不过，这显然不是我们希望看到的效果。毕竟，你的博客文章的摘要可能很短。如何改变这样的样式呢？

幸好可以使用 `clear` 属性清除浮动效果（如图 11.12.4 和图 11.12.5 所示）。如果对某个元素使用该属性，该元素和它后面的元素就会显示在浮动元素的下面。

```
.post-footer {
    clear: left;
}
```

图 11.12.4 使用 `clear: left;`，让内容显示在浮动元素的后面

1. 让浮动元素的父元素“自清除”

回过头来看 11.11 节（参见图 11.11.6），我们还有一个布局问题有待解决。尽管在 HTML 中页面的页脚在 `main` 和附注栏的后面（如图 11.12.6 所示），但是它并没有显示在 `main` 和附注栏的后面。

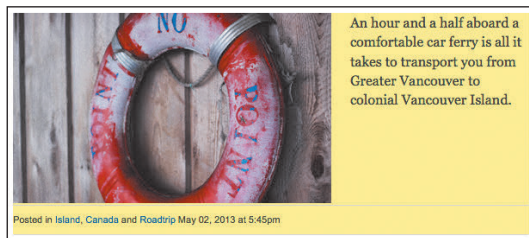


图 11.12.5 现在，博客文章的页脚不会显示在图像的旁边了，无论它前面的文本多短。父元素的高度也会相应增加。可以为页脚添加 `margin-top` 属性，这样它不会紧贴着图像。一旦清除浮动，浮动效果就没有了，因此不需要为图像后面的所有元素都应用这条规则

```
...
<body>
<div class="page">
    ... [masthead] ...

    <div class="container">
        <main role="main">
            ...
        </main>

        <div class="sidebar">
            ...
        </div>
    </div> <!-- end container -->

    <footer role="contentinfo" ...>
        <p class="legal"><small>&copy;
2013
→ Le Journal. All Rights
Reserved.
→ </small></p>
    </footer>
</div>
</body>
</html>
```

图 11.12.6 示例页面的基本结构。带 `class="container"` 的 `div` 是 `main` 和附注栏的父元素。页面的页脚在它们的后面

根据先前定义的样式，`main` 元素有 `float: left;`，附注栏有 `float: right;`。现在你了解了，浮动的元素不会影响其父元素的高度，

这样你就能理解页脚显示在靠近页面顶端位置的原因了。两列内容父元素（作为容器的 `div`）的高度是 0！不信？看看图 11.12.7。



图 11.12.7 如果应用样式 `.container { border: 1px solid red; }`，你可能认为容器 `div` 的子元素外围出现一条边框，实际上不是，你会看到边框显示为一条线，位于容器 `div` 的顶端。这是因为它所有的内容要么是左浮动的，要么是右浮动的，因此它的高度为 0。这就是页面的页脚直接显示在容器下面的原因

可以通过为页脚设置 `clear: both;` 解决这个问题。这与图 11.12.4 中演示的方法类似，它可以清除浮动，既包括左浮动，也包括右浮动。

这样做可以让页脚显示在浮动的列的下方，但容器 `div` 的高度仍然是 0。清除浮动的 `.post-footer` 与浮动元素（图 11.12.1 中的 `section`，效果如图 11.12.5 所示）位于同一个父元素的里面，但页面的页脚却位于作为容器的父元素（如图 11.12.6 所示）的外面。这意味着它无法影响容器的高度、浮动的元素或不浮动的元素。

大多数时候，这样做没有问题。但如果希望为容器添加一个背景色，由于容器高度为 0，不会显示该背景色。解决的办法是让容器自身具有清除浮动的能力。

有很多方法可以解决这个问题，但最可靠的是使用所谓的 `clearfix` 方法。要使用这种方法，只需要在样式表中引入 `.clearfix` 的规则（如图 11.12.8 所示），然后为浮动元素的父元素（该元素为希望清除浮动的元素）添加 `clearfix` 类（如图 11.12.9 所示），从而可以支撑起父元素，并能够添加一些预期样式（如图 11.12.10 和图 11.12.11 所示）。

```
.clearfix:before,
.clearfix:after {
    content: " ";
    display: table;
}

.clearfix:after {
    clear: both;
}

.clearfix {
    *zoom: 1;
}
```

图 11.12.8 大量网站使用 `clearfix` 类或类似方法清除浮动。本书不会对所有这些代码进行详细解释，因为有些复杂。不过请将它常备手头，复制到每一个需要用到它的网站的样式表中去

```
...
<div class="container clearfix">
    <main role="main">
        ...
    </main>

    <div class="sidebar">
        ...
    </div>
</div>
...
```

图 11.12.9 对容器添加 `clearfix` 类会清除浮动的 `main` 和附注栏元素，从而让容器的高度等于两列中较高的那一个的高度

```
.container {
    background: url(../img/bg.png)
        repeat-y 629px 0;
    padding-bottom: 1.9375em;
}

footer[role="contentinfo"] {
    border-top: 1px solid #cacbc;
}
```

图 11.12.10 这里有一个简单的 5 像素 × 20 像素的背景图像，背景位置为距左侧 629 像素，垂直方向重复，从而为两栏之间提供了一个分隔线。另外，可以为页面的页脚加上一个上边框，将页脚与其他内容分隔开



图 11.12.11 页面的页脚的边框与两栏的分隔线是垂直的。为两栏的容器添加的下内边距让分页链接与页脚之间产生了一些空间

2. 控制元素浮动位置的步骤

(1) 输入 `clear:` (如图 11.12.4 所示)。

(2) 输入 `left`, 阻止元素浮动在该元素的左边;

或输入 `right`, 阻止元素浮动在该元素的右边;

或输入 `both`, 阻止元素浮动在该元素的左右两边;

或输入 `none` (默认值), 允许元素浮动在该元素的左右两边。

(3) 如果要想浮动元素的祖先元素在高度上包含浮动元素, 并消除浮动, 可以使用 `clearfix` (参见图 11.12.8 和图 11.12.9) 或 `overflow` 方法替代第 (1) 步和第 (2) 步。(参见补充材料“使用 `overflow` 创建自清除浮动元素”。)

提示 应该将 `clear` 属性添加到不希望环绕浮动对象的元素上 (参见图 11.12.4 ~ 图 11.12.6)。因此, 如果要想让一个元素在右侧没有浮动元素 (以及任何靠向右侧的元素) 之后才显示, 就为它添加 `clear: right;` (而不是为浮动的元素添加此样式规则)。而 `clearfix` 和 `overflow` 方法是应用于浮动元素的父元素或祖先元素的。

提示 过去的几年, 有很多人在不断地优化 `clearfix` 的代码。图 11.12.8 演示的代码来自 Nicolas Gallagher (见 <http://nicolasgallagher.com/micro-clearfix-hack/>, 不过注意这篇文章讲到的技术比较深), 并包含在出色的 HTML5 Boilerplate (www.h5bp.com) 项目中 (该项目是由 Paul Irish 发起的)。

提示 浮动元素的 `display` 属性会变成 `display: block;`, 哪怕将其设置为 `display: inline;` (无论是浏览器的默认样式还是手动显式设置), 该属性值依然为 `block`。

使用 `overflow` 创建自清除浮动元素

通常, 可以对浮动元素的父元素使用 `overflow` 属性以替代 `clearfix` 方法 (参见图 11.12.8 和图 11.12.9)。例如, 在示例页面中可以使用以下代码:

```
.container {
  overflow: hidden;
}
```

在某些情况下, `overflow: hidden;` 会将内容截断, 对此要多加注意。有时使用 `overflow: auto;` 也有效, 但这样做可能会出现一个滚动条, 这显然是我们不希望看到的。有的代码编写人员仅在 `overflow` 能解决 `float` 问题的情况下才使用 `overflow` 方法, 在其他情况下则使用 `clearfix` 方法。有的代码编写人员则始终使用 `clearfix` 方法。

顺便说一下, 可以将 `clearfix` 或 `overflow` 应用到浮动元素的任何一个非父元素的祖先元素。这样做不会让父元素变高, 但祖先元素的高度会包含浮动元素。

关于 `overflow` 属性的更多信息, 参见 11.16 节。

11.13 对元素进行相对定位

每个元素在页面的文档流中都有一个自然位置（参见图 11.13.1 和图 11.13.2）。相对于这个原始位置对元素进行移动就称为相对定位（如图 11.13.3 和图 11.13.4 所示）。周围的元素完全不受此影响（如图 11.13.4 所示）。

```
...
<h1>Relative Positioning</h1>

<p>When you position an element relatively,
→ you <span class="example">position it
→ </span> relative to its normal location.
→ </p>
...
```

图 11.13.1 一会儿要定位的 span 文本

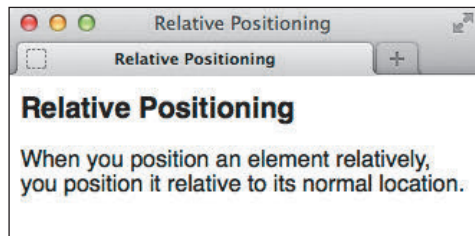


图 11.13.2 这是一个普通段落，这里的呈现没什么奇怪的

```
.example {
  position: relative;
  top: 35px;
  left: 100px;
}
```

图 11.13.3 记住，既要指明采用相对位置，还要给出偏移量。可以使用正数，也可以使用负数。为简化示例，这里使用的是像素，使用 em 会使偏移量的大小与文本字体大小成比例，即便文字变小或者变大都一样。由于 1em 等于元素的字体大小，默认情况下，文本通常为 16 像素，因此这个例子中的 top: -3em;，会使文本向下移动 48 像素

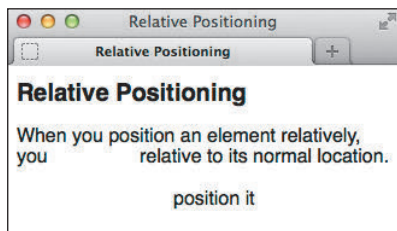


图 11.13.4 相较于先前的位置（现在显示为一块空白），文本与上边相距 35 像素，左边相距 100 像素。如果设置为 top: -55px;，它的位置就会在 h1 文本的上面

偏移自然流中元素的步骤

- (1) 输入 position: relative;。
- (2) 输入 top、right、bottom 或 left。

再输入 :d，这里的 d 是希望元素从它的自然位置偏移的距离，可以表示为绝对值（如 10px）、相对值（如 2em）或百分数。值可负可正。需要添加其他偏移量，可重复这一步。

提示 其他元素不会受到偏移的影响，仍然按照这个元素原来的盒子进行排列。设置了相对定位的内容可能与其他内容重叠，这取决于 top、right、bottom 和 left 的值。

提示 使用相对定位、绝对定位或固定定位时，对于相互重叠的元素，可以用 z-index 属性指定它们的叠放次序。详细信息参见 11.15 节。

提示 对元素设置 position: static，可以覆盖 position: relative 设置。static 是元素的默认值，这就是元素出现在常规文档流中的原因。有关示例参见 11.15 节。

提示 定位不是继承的。

11.14 对元素进行绝对定位

正如前面提到的，网页中的元素通常按照它们在 HTML 源代码中出现的次序进行排列（如图 11.14.1 和图 11.14.2 所示），除非你使用 CSS 规则改变前面的样式。也就是说，如果 `img` 元素出现在 `p` 元素之前，图像就出现在段落的前面。

通过对元素进行绝对定位，即指定它们相对于 `body`（参见图 11.14.3 和图 11.14.4）或最近的已定位祖先元素（参见图 11.14.5 和图 11.14.6）的精确位置，可以让元素脱离正常的文档流。

这与相对定位不同，绝对定位的元素不会在原先的位置留下空白。这与让元素浮动也不同。对于绝对定位的元素，其他元素不会环绕在它的周围。事实上，其他内容不知道它的存在，它也不知道其他内容的存在。

```
...
<header class="masthead" role="banner">
  <p class="logo"><a href="/"><img ... />
  → </a></p>

  <ul class="social-sites">
    ...
  </ul>

  <nav role="navigation">
    <ul class="nav-main">
      ...
    </ul>
  </nav>
</header>
...
```

图 11.14.1 页面的报头（或为页眉）依次包含了网站的标识、社交网站图标和主导航。它们的默认显示方式如图 11.14.2 所示。接下来你将看到如何通过两个突出显示的类，让社交网站图标在报头中进行绝对定位

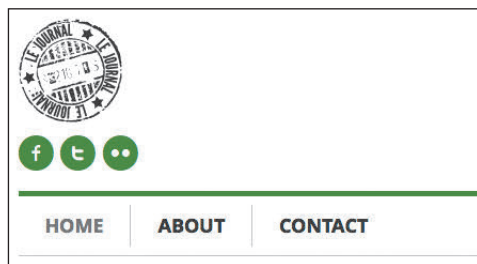


图 11.14.2 在默认文档流里，社交网站图标的列表位于网站标识和导航之间。我们希望它显示在包含它的报头里自上而下一半的位置，并让它显示在报头的右侧

```
.masthead .social-sites {
  position: absolute;
  top: 41px;
  right: 0;
}
```

图 11.14.3 通过对图标列表进行绝对定位，我们让它完全脱离了文档流。仅用这段代码还不能实现目的，因为在没有另外进行指定的情况下，设置 `position: absolute` 的元素是相对于 `body` 进行定位的，如图 11.14.4 所示

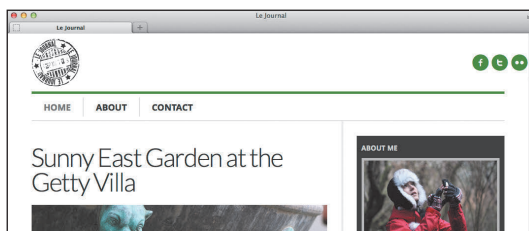


图 11.14.4 图标列表显示在距离 `body` 上边缘 41 像素、右边缘 0 像素的位置。很明显这种处理方法不好，因为用户将浏览器窗口放大得越宽，图标列表就跟页面中的其他内容距离越远

对元素进行绝对定位的步骤

- (1) 输入 `position: absolute;`。
- (2) 根据需要，输入 `top`、`right`、`bottom` 或 `left`。

再输入 `:d`，这里的 `d` 是希望元素相对于其祖先元素偏移的距离（如 `10px` 或 `2em`）或

相对于其祖先的百分数（有关解释参见第二条提示）。

```
.masthead {
    position: relative;
}

.masthead .social-sites {
    position: absolute;
    top: 41px;
    right: 0;
}
```

图 11.14.5 对包含图标列表（参见图 11.14.2）的页眉设置 `position: relative`，从而让这些图标可以相对页眉（而不是 `body` 元素）进行绝对定位。这样就可以让图标显示在我们希望它显示的位置，如图 11.14.6 所示



图 11.14.6 完美了

(3) 根据需要，对其他方向重复第 (2) 步。

(4) 根据需要，对希望成为绝对定位参照体的祖先元素添加 `position: relative`；（如图 11.14.5 和图 11.14.6 所示）。如果跳过这一步（如图 11.14.3 所示），将对元素相对于 `body` 计算偏移量（如图 11.14.4 所示）。

提示 由于绝对定位的元素脱离了文档流，因此它们可能会相互重叠，或与其他元素重叠（这不一定是坏事儿）。

提示 如果不为绝对定位的元素指定偏移量，这个元素将显示在它的自然位置上，但不会影响后续元素在文档流中的位置。

提示 还有一种定位类型称为固定定位。当访问者滚动浏览器窗口时，页面内容通常随之上下移动。如果对元素设置 `position: fixed`，它就会固定在浏览器窗口中。当访问者上下滚动浏览器窗口时，该元素不会随之移动，页面的其余部分仍照常滚动。这跟固定背景图像的工作原理类似（参见 10.10 节）。固定定位在很多移动浏览器中效果不佳，因此如果你希望自己的网页能很好地适应移动设备，最好不要使用固定定位。

提示 使用相对定位、绝对定位或固定定位时，可以使用 `z-index` 属性指定相互重叠的元素的叠放次序。详细说明参见 11.15 节。

提示 对元素设置 `position: static` 将覆盖 `position: absolute` 的设置。`static` 是元素定位的默认值，这也是元素为什么出现在常规文档流中的原因。有关实例参见 11.15 节。

提示 定位不是继承的。

11.15 在栈中定位元素

当你开始使用相对定位、绝对定位和固定定位以后，很可能发现元素相互重叠的情况，这时可以选择哪些元素应该出现在顶层（参见图 11.15.1、图 11.15.2 和图 11.15.3）。

```
<div class="box1"><p>Box 1</p></div>
<div class="box2"><p>Box 2</p></div>
<div class="box3"><p>Box 3</p></div>
<div class="box4"><p>Box 4</p></div>
```

图 11.15.1 HTML 中 div 的次序

```
div {
  border: 1px solid #666;
  height: 125px;
  position: absolute;
  width: 200px;
}

.box1 {
  background: pink;
  left: 110px;
  top: 50px;
  z-index: 120;
}

.box2 {
  background: yellow;
  left: 0;
  top: 130px;
  z-index: 530;
}

.box3 {
  background: #ccc;
  position: static;
  /* 静态的, 没有任何效果 */
  z-index: 1000;
}

.box4 {
  background: orange;
  left: 285px;
  top: 65px;
  z-index: 3;
}
```

图 11.15.2 对于定位元素, `z-index` 最高的数显示在最上面 (如图 11.15.3 所示), 不管该元素在 HTML 中出现的次序 (如图 11.15.1 所示)。第一条规则为所有四个 `div` 设置了 `position: absolute;`, 而定义 `.box3` 时又覆盖了这一规则, 让 `.box3` 回到默认的 `static`。因此, 尽管它的 `z-index` 值最高, 但这没有任何效果, 它总是位于最底层

在栈中定位元素

输入 `z-index: n`, 其中的 n 是表示元素在定位过的对象堆中的层级的数字。

提示 `z-index` 的值越大, 元素在堆中就越高 (参见图 11.15.2 和图 11.15.3)。

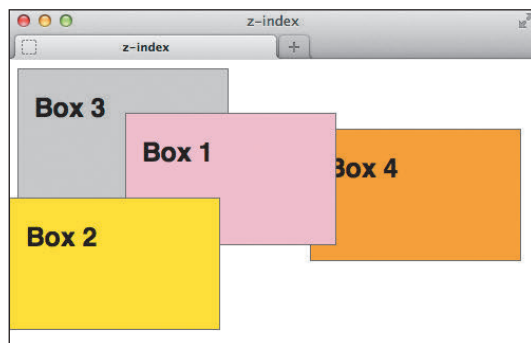


图 11.15.3 定位的盒子按照 `z-index` 由高到低的次序进行叠放。第三个盒子位于最底层, 因为它处于常规的文档流

提示 `z-index` 属性仅对定位过的元素 (即设为绝对定位、相对定位或固定定位的元素) 有效。图 11.15.1 仅包含绝对定位的元素, 但实际上可以对绝对定位、相对定位和固定定位的元素混合使用 `z-index`, `z-index` 会将它们作为整体进行安排, 而不是分别安排。

提示 `z-index` 属性不是继承的。

11.16 处理溢出

元素并不总是包含在它们自己的盒子里。这可能是因为盒子不够大, 例如, 图像比它的容器更宽就会溢出; 也可能是因为使用负的外边距或绝对定位让内容处于盒子的外面; 还有可能你对元素设置了显式高度, 它的内容太高而无法装入盒子内部。无论是哪种情况, 都可以使用 `overflow` 属性控制元素在盒子外面的部分, 参见图 11.16.1 和图 11.16.2。

```
div {
    background: #e0f7ac;
    border: 1px solid #666;
    height: 88px;
    width: 300px;
}

.example-hidden { /* div 2 */
    overflow: hidden;
}

.example-scroll { /* div 3 */
    overflow: scroll;
}

.example-auto { /* div 4 */
    overflow: auto;
}
```

图 11.16.1 这里对 div 设置了高度，不过通常最好不要对元素设置高度。这里为 div 添加了背景色和边框，很容易看出不设置高度的原因。三个类演示了控制内容溢出的三种方式

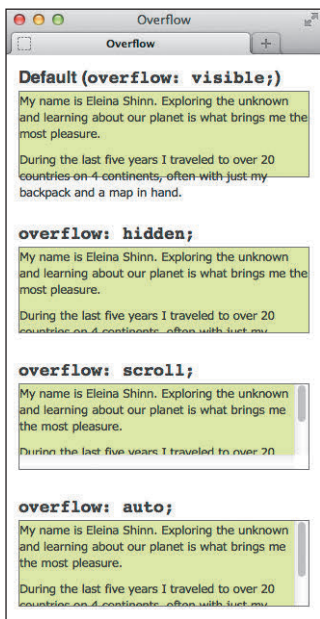


图 11.16.2 每个带边框的绿色区域都是一个包含两个段落的 div。浏览器会照顾对元素指定的高度。如果内容太高，就会穿过 div，透到外面来（就像第一个 div 那样），除非使用其他 overflow 设置，覆盖默认的风格

决定浏览器如何处理溢出的步骤

(1) 输入 overflow:。

(2) 输入 visible，让元素盒子中的所有内容可见，这是默认项；

或者输入 hidden，隐藏元素盒子容纳不了的内容；

或者输入 scroll，无论元素是否需要，都在元素上添加滚动条；

或者输入 auto，让滚动条仅在访问者访问溢出内容时出现。

提示 如果图 11.16.1 中的代码使用 min-height: 88px; 代替 height 88px;，图 11.16.2 中的 div 就会随着内容的增加而变大。如果内容的高度比 88 像素矮，div 仍为 88 像素高，因为 min-height 设置的是最小高度。

提示 overflow 属性还可以很方便地清除 float，参见 11.12 节的补充材料。

提示 overflow 属性不是继承的。

11.17 垂直对齐元素

可以使用除默认对齐方式以外的多种方式对齐元素，让它们在页面上显得较为整齐（如图 11.17.1 ~ 图 11.17.4 所示）。

```
...
<form action="results.php" method="get"
  → role="search">
  <label for="search">Search</label>
  <input type="search" id="search"
    → name="search" />
  <input type="image" src="img/btn-go.png"
    → alt="Submit search" />
</form>
...
```

图 11.17.1 这是一个简单的表单，只有一个文本标签、一个文本输入框和一个图像提交按钮。关于表单的更多信息，参见第 16 章

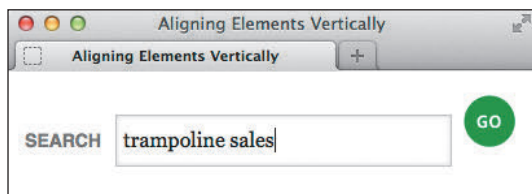


图 11.17.2 默认情况下，行内内容在垂直方向上与文本的基线对齐。这里在表单输入框里填入了一些文本，用以演示这一情况——标签文本（SEARCH）、输入框中的文字与图像的底端都与基线对齐

```
input[type="image"] {
    vertical-align: bottom;
}
```

图 11.17.3 每一行内容都有一个看不见的框（线框），它代表了行的高度。在这个例子中，文本输入字段就能指示框的下边缘，因为它是这一行最低的部分（如图 11.17.2 所示），图像会对齐线框的底部（如图 11.17.4 所示）

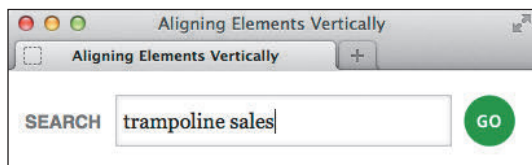


图 11.17.4 搞定！

使元素垂直对齐的步骤

- (1) 输入 `vertical-align:`。
- (2) 输入 `baseline`，使元素的基准线对齐父元素的基准线；

或者输入 `middle`，使元素位于父元素中央；

或者输入 `sub`，使元素成为父元素的下标；

或者输入 `super`，使元素成为父元素的上标；

或者输入 `text-top`，使元素的顶部对齐父元素的顶部；

或者输入 `text-bottom`，使元素的底部对齐父元素的底部；

或者输入 `top`，使元素的顶部对齐当前行里最高元素的顶部；

或者输入 `bottom`，使元素的底部对齐当前行里最低元素的底部；

或者输入元素行高的百分比，可以是正数，也可以是负数；

或者输入某个值（正负均可，单位为像素或 `em`）分别按照特定的值向上或者向下移动元素。

更多关于 `vertical-align` 的说明

可以使用 `vertical-align` 设置表格单元格中内容的对齐方式。通常，默认的模式是中间对齐，而不像表格以外的内容那样与基线对齐。我们将在第 18 章讲解表格。除了表格以外，`vertical-align` 属性仅适用于行内元素，不能应用于块级元素。更多信息参见 Chris Coyier 的解释（<http://css-tricks.com/what-is-vertical-align/>）。

11.18 修改鼠标指针

一般情况下，浏览器负责控制鼠标指针的形状。大多数时候使用箭头形状，指向链接时使用手指形状（参见图 11.18.1），等等。使用 CSS 可以修改指针的形状（如图 11.18.2 和图 11.18.3 所示）。

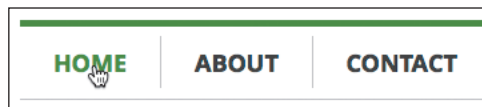


图 11.18.1 指向 Home 链接时，指针变成手指形状，同时链接高亮显示。对于其他链接也是这样

```
.nav-main .current-page,  
.nav-main .current-page:hover {  
    color: #747474;  
    cursor: default;  
}
```

图 11.18.2 当访问者位于主页时，我为 HOME 链接添加了 `class="current-page"`。这样就修改了链接文本的颜色、鼠标停留状态的颜色和指针形状，让它看起来不像是链接（另外，在这个例子中，也可以在导航中去掉 HOME 链接周围的 `a` 元素）

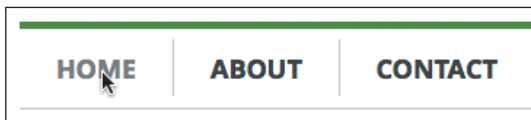


图 11.18.3 尽管它仍然是真正的链接，但它看起来已不像链接。访问者正位于这个链接指向的页面，因此这样做是有意义的

修改指针形状的步骤

(1) 输入 `cursor:`。

(2) 输入 `pointer` 表示停留在链接上时通常显示的指针形状(☞)或 `default` 表示箭头(☞)，或者输入 `crosshair` (+)、`move` (☞)、`wait` (⌚)、`help` (☞?)、`text` (I) 或 `progress` (☞)；

或者输入 `auto` 显示特定情形下通常使用的指针形状；

或者输入 `x-resize` 显示双向箭头，这里的 `x` 是其中一个箭头需要指向的方向，可以是 `n` (北)、`nw` (西北)、`e` (东)，等等。例如，`e-resize` 指针显示成☞。

提示 不同浏览器、不同系统的指针形状可能有细微的差异。

第 12 章

构建响应式网站

本章内容

- 响应式 Web 设计：概述
- 创建可伸缩图像
- 创建弹性布局网格
- 理解和实现媒体查询
- 汇总
- 兼容旧版 IE

最后一分钟决定要不要去看电影；就安道尔的官方语言跟人打赌；查找已经迟到 15 分钟的会议所在公司的电话号码；查看去这家公司的地图（因为迟到的原因是找不到这家公司）……

我们想要及时地获取信息，而随着各种形状和尺寸的移动设备的剧增，万维网可以出现在你的口袋里、钱包里、背包里，就像它在书桌、餐桌和卧室一样方便。

现在，网站的建设者需要让访问者能够通过移动电话、智能手机、平板电脑、笔记本电脑、台式机、游戏机、电视机以及未来任何可以上网的设备获取信息。响应式 Web 设计就是为此诞生的。

在本章中，你将了解到如何构建在各种设备上都能正常工作的网站，它能根据设备的功能和特征对布局进行调整。

12.1 响应式 Web 设计：概述

以前，为了满足移动用户的需求，需要额外建立一个专门为移动设备设计的网站。不久之前这种做法还比较流行，但是目前已经越来越少了。不过，现在也还有一些公司在这样做。有的还为平板电脑建立了第三个网站。

不存在可以适应所有情形的某种正确方法。每周都有新的设备投放到市场上，而且毋庸置疑的是，各家公司都在设计新的设备类型。那么，构建和维护多个网站现实吗？甚至说，有必要吗？我们无法预知未来的情形。

幸好，我们有办法构建一个既可以运行在现在的设备，也可以运行在未来设备上的网站。更棒的是，可以让它在较小的屏幕、较大的屏幕以及介于两者之间的屏幕上显示不同的效果（如图 12.1.1 ~ 图 12.1.3 所示）。

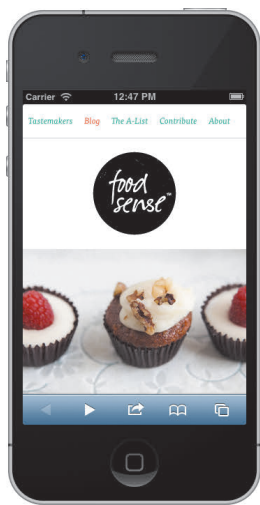


图 12.1.1 不管你信不信，这里以及图 12.1.2 和图 12.1.3 显示的 Food Sense 主页（www.foodsense.is）都来自同一个网站，而不是各自具有单独 URL 的不同网站。这个网站采用了响应式 Web 设计方法，因此它的布局可以根据不同的查看环境进行改变。iPhone（如本图所示）及其他屏幕大小相似的设备会根据特定的 CSS 规则显示布局。对于更大的浏览器窗口（参见图 12.1.2 和图 12.1.3），则有另外一些 CSS 规则控制相应的布局

响应式页面的组成

Ethan Marcotte 为我们提供了如何实现上述效果的蓝图，并将相应的方法命名为响应式 Web 设计（responsive Web design）。关于响应式设计的历史参见 11.1 节。他的方法植根于以下三点。

- ❑ 灵活的图像和媒体。图像和媒体资源的尺寸是用百分数定义的，从而可以根据环境进行缩放。
- ❑ 灵活的、基于网格的布局，也就是流式布局。对于响应式网站，所有的 width 属性都用百分数设定，因此所有的布局成分都是相对的。其他水平属性通常也会使用相对单位（em、百分数和 rem 等）。

- ❑ 媒体查询。使用这项技术，可以根据媒体特征（如浏览器可视页面区域的宽度）对设计进行调整。

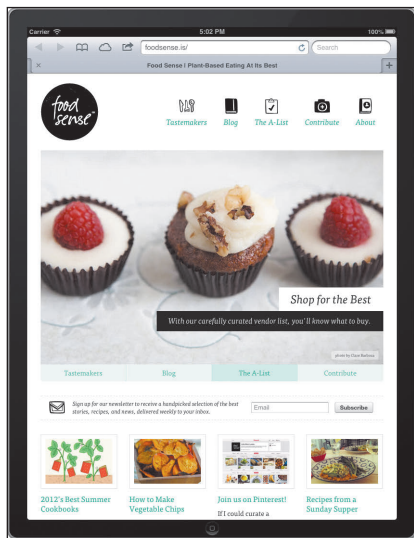


图 12.1.2 这是使用 iPad 及其他屏幕大小相似的设备查看 Food Sense 主页的样子。由于浏览器有更多空间显示内容，因此这套样式的 CSS 对标识和导航都进行了调整

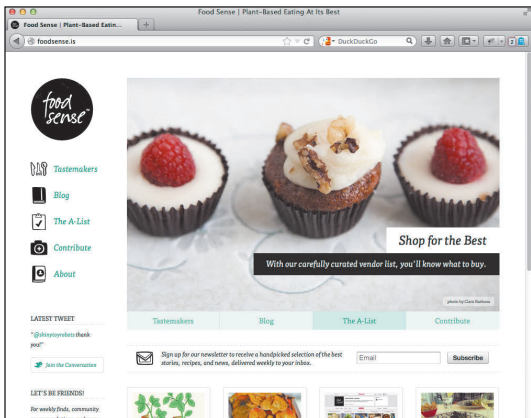


图 12.1.3 这是 Food Sense 主页显示在桌面浏览器上的样子，也是最宽的显示效果。该站还有另外两个布局没有展示出来。你可以在计算机上访问 www.foodsense.is，再拖动浏览器的一角让浏览器窗口变窄或变宽并查看效果

我们将在接下来的几章逐一介绍这些主题。随后，你将一步步学习如何构建第 11 章示例的响应式版本。

提示 John Allsopp 的文章“Web 设计之道”（“A Dao of Web Design”，<http://alistapart.com/article/dao>）写于 2000 年，该文讨论了设计和构建灵活的网站的方法。这篇文章是响应式 Web 设计的先驱，Marcotte 以及很多其他的作者都引用过这篇文章，且认为它影响巨大。这篇文章很值得一读。

提示 Jeremy Keith 在题为“One Web”的演讲中归纳了“一个网站适应所有设备”的方法（www.vimeo.com/27484362/）。讲稿全文参见 www.adactio.com/articles/4938/。

12.2 创建可伸缩图像

正如你在第 5 章学到的，默认情况下，图像显示的尺寸是 HTML 中指定的 `width` 和 `height` 属性值（参见图 12.2.1）。如果不指定这些属性值，图像就会自动按照其原始尺寸显示。此外，你还可以通过 CSS 以像素为单位设置 `width` 和 `height`。

显然，当屏幕宽度有限的时候，按原始尺寸显示图像就不一定合适了（如图 12.2.2 所示）。使用可伸缩图像技术（如图 12.2.3 和图 12.2.4 所示），就可以让图像在可用空间内缩放，但不会超过其本来的宽度（如图 12.2.5 和图 12.2.6 所示）。可用空间是由包含图像的元素决定的（参见最后一条提示）。在图 12.2.5 中，可用空间是由 `body` 元素决定的，在图 12.2.6 中则是 `main` 元素。

```
...
<meta name="viewport"
  → content="width=device-width,
  → initial-scale=1" />
</head>
<body>



</body>
</html>
```

图 12.2.1 一些典型的指定了 `width` 和 `height` 的 `img` 标签（参见图 12.2.2）。这些代码出现在第 11 章的代码中（注意，这里省略了 `alt` 中的文本，是为了让代码更简短）

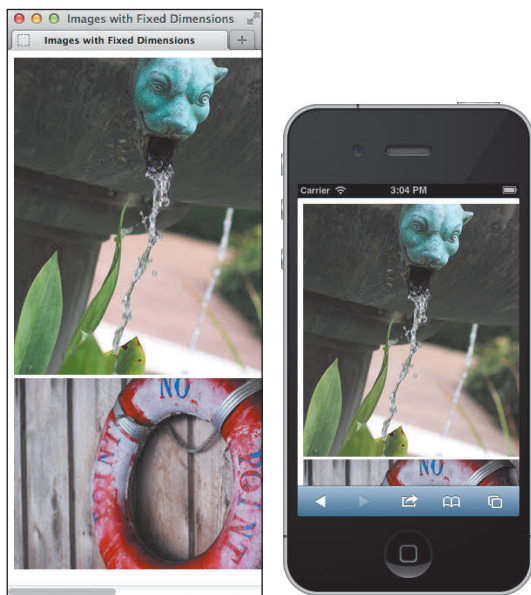


图 12.2.2 使用固定的 `width` 和 `height` 值时，即便可视区域变小，图像也依然显示其原始尺寸。桌面浏览器会显示滚动条（左），而在手机上（右），图像占据了整个屏幕

```
...



...
```

图 12.2.3 只有 HTML 中不包含 width 和 height 属性，图像才有可能变成可伸缩的图像。你还可以为图像添加用于添加样式的类

```
.post-photo,
.post-photo-full {
    max-width: 100%;
}
```

图 12.2.4 让图像变成可伸缩图像的一小段魔法 CSS

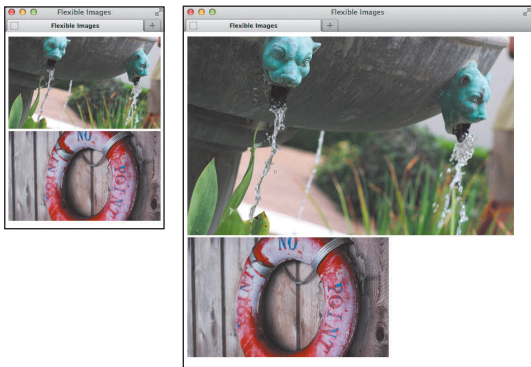


图 12.2.5 可伸缩图像可以根据包含它们的元素（在这个例子中为 body）的尺寸按比例缩放。它们不会比其本来的宽度更宽，就像下面的图像所演示的那样。在这个例子中，上面的图像的最大宽度为 600 像素，下面的为 370 像素

制作可伸缩图像的步骤

(1) 对任何想做成可伸缩图像的图像，在 HTML 的 img 标签中省略 width 和 height 属性（如图 12.2.3 所示）。

(2) 在样式表中，为每个想做成可伸缩图像的图像应用 max-width: 100%（如图 12.2.4 所示）。



图 12.2.6 我们在本章将要创建的响应式页面在 320 像素宽的区域（左）和 480 像素宽的区域（右）中显示的样子。图像会缩放以适应容器！

提示 图像缩放的可用空间是由其父元素建立的内容区域。如果父元素有水平方向上的内边距，可用空间就会相应减小。关于内容区域的尺寸，参见 11.7 节。

提示 一定要使用 max-width: 100% 而不是 width: 100%。它们都能让图像在容器内缩放，不过，width: 100% 会让图像尽可能地填充容器，如果容器的宽度比图像宽，图像就会放大到超过其本来尺寸，有可能会显得较为难看。

提示 图标字体和 SVG 可以创建无损缩放的图形。这两个主题分别参见 13.2 节和 5.1 节。

提示 不要忘了对图像进行优化，让文件大小尽可能小，参见 5.4 节。

提示 这项技术对已经为 Retina 显示屏扩大到双倍大小的图像也适用。当然，双倍分辨率的图像的文件大小也会大很多。关于如何创建为 Retina 显示屏准备的图像，参见 5.8 节（其中还包含了一条帮助减小文件尺寸的提示）。

提示 可以使用 `background-size` 属性对背景图像进行缩放（对 IE8 无效）。更多信息参见 www.css3.info/preview/background-size/。

提示 还可以使用 `video`, `embed`, `object` { `max-width: 100%;` } 让 HTML5 视频及其他媒体变成可伸缩的（同样也不要再在 HTML 中为它们指定 `width` 和 `height`）。对于视频无法按照预期进行缩放的情况，参见 Jonathan Nicol 的文章（<http://f6design.com/journal/2011/10/18/responsive-elements-that-retain-their-aspect-ratio/>）。

响应式设计 with 图像之谜

可伸缩图像很棒，对吧？“不过，等等，”你可能会说，“加载一个比在页面上需要显示的图像更大的图像，不是很不好吗？尤其是在连接速度较慢的移动设备上？”

你说得没错，这也是人们试着处理响应式 Web 设计时遇到的最大的问题。

理想情况下，我们需要的是在特定设备上刚好符合尺寸要求的图像。移动设备上用小图像，平板电脑上用大一些的，等等。这能让文件大小变小，同时减小设备用于显示图像的内存消耗（这对手机来说尤其重要）。特别是，这样做能加快页面加载速度。这对那些需要为移动数据计划付费的用户来说也相当友好（移动数据计划限制了用户每个月可以下载的数据大小）。

此外，有时你可能希望在移动设备上显示不一样的图像，而不仅仅是缩小的图像。一个常见的例子是人的照片。小的屏幕只显示人脸的部分，而大的屏幕则显示全身照。

很多人致力于解决这一问题，但到本书写作之际，还没有一种简单的解决办法。

有两个建议为 HTML 增加功能的提案。一个建议是为 `img` 增加一个名为 `srcset` 的属性，另一个建议是增加一个新的名为 `picture` 的元素（参见 <http://www.w3.org/community/resping/>）。它们解决的是相似的问题，但并非同一个问题。因此这两个提议有同时变为现实的可能。敬请期待！

同时，在页面中引入图像的时候也要聪明一些。我们的示例页面中最大的图像为 600 像素宽（如图 12.2.6 所示），比智能手机需要的尺寸（大多为 320 像素宽）要大一些，因此访问者有可能察觉到需要较长时间才能加载。幸好页面上只需要显示这一个图像。至于为 Retina 显示屏准备图像，参见倒数第三条提示。

12.3 创建弹性布局网格

拥有定宽容器的网页显得有些死板。前一章创建的例子就是这样（参见图 12.3.1 ~

图 12.3.3）。如果桌面浏览器的宽度小于页面宽度，就会出现横向滚动条（如图 12.3.4 所示）。

```
...
<body>
<div class="page">
  <header class="masthead" role="banner">
    ...
  </header>

  <div class="container">
    <main role="main">
      ...
    </main>

    <div class="sidebar">
      ...
    </div>
  </div>

  <footer role="contentinfo"
  → class="footer">
    ...
  </footer>
</div>
</body>
</html>
```

图 12.3.1 在第 11 章创建的页面中，所有的内容都处于一个 page 类的 div 里。其中的基本组成包括报头（页眉）、包含 main 和附注栏区域的 div（带有 container 类），以及页面的页脚

```
.page {
  width: 960px;
}

main {
  width: 600px;
}

.sidebar {
  width: 300px;
}
```

图 12.3.2 这些是为第 11 章创建的示例页面（参见图 12.3.1）设置的固定宽度。我们没有为页眉、页脚以及包含 main 和附注栏的 container div 设置宽度，而是让它们使用浏览器的默认值，即 width: auto;，因此它们会跟其父元素一样宽。它们的父元素是 page div（参见图 12.3.1），因此它们也是 960 像素宽（如图 12.3.3 所示）

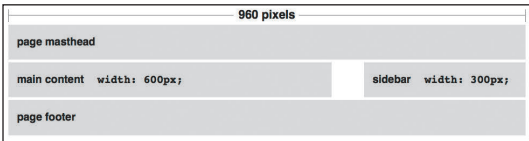


图 12.3.3 最起码，页面要包含四个主要内容区域。page div 和 container div 没有显示在这里，是因为它们本身并不包含任何内容。container div、报头和页脚没有指定宽度（参见图 12.3.2），它们的宽度自动与 page div 相同。（注意：图 12.3.2 和图 12.3.6 中的 CSS 不会让 main 和附注栏左右并排显示。要知道如何实现这种效果，参见 11.11 节）

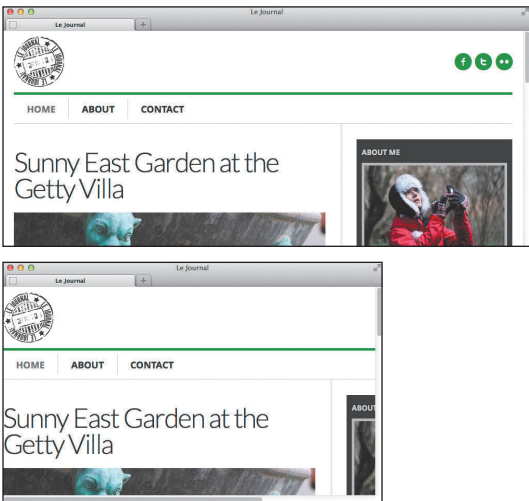


图 12.3.4 即便让浏览器窗口变窄，宽度仍然保持不变

移动浏览器会显示完整的宽度，但页面会显得特别小，这并不太友好（如图 12.3.5 所示）。

这对响应式页面来说并不合适。我们希望页面能进行缩放，并正好适应浏览器视觉区域大小，就像可伸缩图像一样。流式布局（又称弹性布局）便可以做到这一点。

一旦掌握了弹性布局方法，创建弹性布局就很容易了。创建弹性布局需要使用百分数宽度，并将它们应用于页面里的主要区域

(参见图 12.3.1 和图 12.3.6)。这需要一些数学知识, 不过不必担心, 它不会比你小学数学学到的东西复杂。如果你怀疑这一点, 可以在手头准备一个计算器!

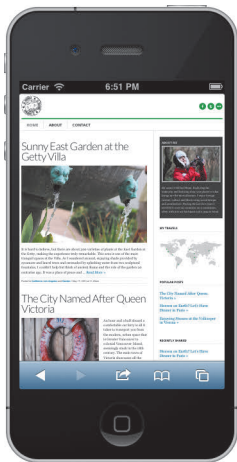


图 12.3.5 像 iPhone 这样的手机会缩小页面 (如果你希望的话), 但访问者需要放大才能阅读里面的内容

```
.page {
  max-width: 960px;
}

main {
  /* 要求宽度/包含块宽度使用600px/960px */
  width: 62.5%;
}

.sidebar {
  width: 31.25%; /* 300px/960px */
}
```

图 12.3.6 这是与图 12.3.2 中定宽布局对等的流式布局样式。为 `.page` `div` 设置的死板的 `width: 960px`; 被替换为 `max-width: 960px`;。这让该元素可以进行缩放, 但不会超过 960 像素。我用先前提供的公式来确定 `main` 和 `.sidebar` 的百分数宽度

那么我们如何知道需要使用多大的百分数呢? 实际上, 元素的百分数宽度基于其父

元素 (即包含该元素的容器) 提供的可用空间。这些说法似乎有些熟悉? 这是因为创建弹性布局的原理与创建可伸缩图像的原理是相同的。可以使用下面的公式计算需要使用的百分数:

要指定的宽度 (以像素为单位) / 容器宽度 (以像素为单位) = 值

这与 Ethan Marcotte 的公式 (目标 / 环境 = 值) 是一致的, 只是解释得更明了。如果创建了定宽布局, 那么就将它们带到公式里算一下。

1. 一个真实的例子

让我们考虑一个简单的页面布局 (将在本章后面构建该页面)。我将演示如何算出 `main` 的 `width` 为 62.5% (参见图 12.3.6)。使用公式, 不难得出:

要指定的宽度 (600px) / 容器宽度 (960px) = 值 (.625)

然后将值转化为百分数 (将小数点向右移动两位), 得到 62.5%。将这个数值作为 `width` 的值 (参见图 12.3.6)。完成!

如果你对这一数字还是感到困惑, 我会详细解释。布局最宽为 960 像素, 这是由 `.page` 的样式规则确定的 (参见图 12.3.6)。如果不对 `main` 的父元素 `.container` `div` 指定宽度 (参见图 12.3.1), 其宽度就自动为 960 像素。 (`.container` `div` 的父元素是 `.page` `div`。)

如果我希望 `main` 的宽度最多占据 960 像素中的 600 像素。使用先前给出的公式, $600/960=0.625$ 。使用百分数, 则为 62.5%。对 `.sidebar` 也可以使用同样的方法, 只不过我希望其宽度不超过 300 像素。因此使用 $300/960$ 计算, 得 0.3125, 即 31.25% (参见图 12.3.6)。

结合使用可伸缩图像和弹性布局, 便可

以让整个页面变得可以缩放（如图 12.3.7 和图 12.3.8 所示）。



图 12.3.7 无论页面的视觉区域有多宽，两栏始终保持各自所占的比例。如果可视区域宽度超过 960 像素，页面就停止变宽，如最上面的图所示

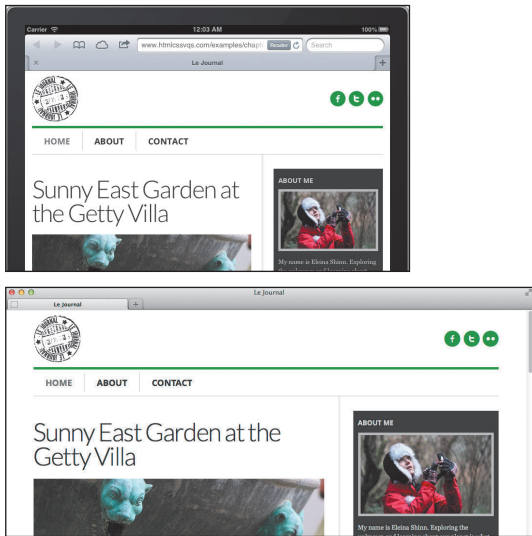


图 12.3.8 应用到本章后面将要建立的响应式页面，图 12.3.6 中给出的样式会让布局可以缩放，不过永远不会超过 960 像素，无论浏览器窗口有多宽，如下面的图所示

2. 创建弹性布局的步骤

对于需要某个宽度实现预期布局的元素，设置 `width: percentage;`，其中 `percentage` 表示你希望元素在水平方向上占据容器空间的比例（参见图 12.3.6）。通常说来，不必设置 `width: 100%;`，因为默认设置为 `display: block;` 的元素（如 `p` 以及其他很多元素）或手动设置为 `display: block;` 的元素在默认情况下会占据整个可用空间。

作为可选的一步，对包含整个页面内容的元素设置 `max-width: value;`，其中 `value` 表示你希望页面最多可增长到的最大宽度（参见图 12.3.6、图 12.3.7 和图 12.3.8）。通常，`value` 以像素为单位，不过也可以使用百分数、`em` 值或其他单位的值。

提示 如果父元素有水平方向上的内边距，它为子元素建立的容器就会变小。要回顾这段内容，请参见 11.7 节。

提示 还可以对元素设置基于百分数的 `margin` 和 `padding` 值。在示例页面中，我对这些属性使用的是 `em` 值，这是一种常见的做法。内边距和外边距的 `em` 值是相对于元素的 `font-size` 的，而基于百分数的值则是相对于包含元素的容器的。

提示 对于设置了 `body { font-size: 100%; }` 的页面，对 `font-size`、`margin`、`padding` 和 `max-width` 使用 `em` 值还有一个好处——如果用户更改了浏览器默认字体大小，那么页面也会跟着变大或变小。例如，在 Firefox 中，可以在 Preferences 中的 Content 标签页中更改默认字体大小。不妨打开 www.htmlcssvqs.com/8ed/examples/12/finished-page.html 试试这一特性。

提示 在使用公式计算元素的百分数宽度时，不要忘了容纳它的容器来自它最直接的祖先（即上下文）。

提示 将 `box-sizing` 属性设置为 `border-box`，就可以很方便地对拥有水平方向内边距（使用 `em` 或其他的单位）的元素定义宽度，而不必进行复杂的数学计算来找出百分数的值。这对响应式页面来说很方便。更多信息参见 11.5 节。

设置相对的 `max-width`

包含整个页面的 `.page div`（参见图 12.3.1）设置了以像素为单位的 `max-width`（960），参见图 12.3.6。这个值能否使用灵活的相对单位呢？其实，设置 `.page { max-width: 60em; }` 就可以了。下面是具体的解释：

`em` 宽度是基于元素字体大小的。例如，如果其字体大小等价于 14 像素，则 `width: 10em;` 会将宽度设置为 140 像素。

`.page div` 没有指定 `font-size`，因此它从父元素（`body` 元素）继承。我们知道，`body` 的默认字体大小通常等于 16 像素。因此，如果希望将 `.page` 的最大宽度设为 960 像素， $960/16=60em$ 。

设置 `.page { max-width: 60em; }` 与使用 960px 是相似的，但它们有一个显著的不同：前者会根据浏览器默认字体大小进行缩放。第三条提示里讲了如何看到这一效果。

12.4 理解和实现媒体查询

我们在 8.6 节中学习过，可以使用两种

方式针对特定的媒体类型定位 CSS。（还有第三种方式，即使用 `@import` 规则，我们不讨论这种方法，因为它会影响性能。）回顾一下，第一种方式是使用 `link` 元素的 `media` 属性，位于 `head` 内，参见图 12.4.1。第二种方式是在样式表中使用 `@media` 规则，参见图 12.4.2。

```
...
<head>
    ...
    <link rel="stylesheet" href=
        → "your-styles.css" media="screen" />
</head>
...
```

图 12.4.1 在屏幕上查看页面时会应用 `your-styles.css` 中的样式

```
/* 屏幕和打印共用的样式 */
...

/* 只用于打印的样式 */
@media print {
    header[role="banner"] nav,
        .ad {
            display: none;
        }
}
```

图 12.4.2 通过 `@media print` 规则可以创建专门为打印浏览器里的页面定义的样式。也可以将它们与为其他媒体定义的样式放在一起

媒体查询增强了媒体类型方法，允许根据特定的设备特性定位样式（参见图 12.4.3）。要调整网站的呈现样式，让其适应不同的屏幕尺寸，采用媒体查询特别方便。下面列出了可以包含在媒体查询里的媒体特性。

- ❑ `width`（宽度）
- ❑ `height`（高度）
- ❑ `device-width`（设备宽度）
- ❑ `device-height`（设备高度）
- ❑ `orientation`（方向）

- ❑ aspect-ratio (高宽比)
- ❑ device-aspect-ratio (设备高宽比)
- ❑ color (颜色)
- ❑ color-index (颜色数)
- ❑ monochrome (单色)
- ❑ resolution (分辨率)
- ❑ scan (扫描)
- ❑ grid (栅格)

还有一些非标准的媒体特性, 如

- ❑ -webkit-device-pixel-ratio (WebKit^① 设备像素比)
- ❑ -moz-device-pixel-ratio (Mozilla^② 设备像素比)

除了 orientation、scan 和 grid 以外, 上述属性均可添加 min- 和 max- 前缀。min- 前缀定位的是“大于等于”对应值的目标, 而 max- 前缀定位的则是“小于等于”对应值的目标。

在本章里, 我们将着重介绍 min-width 和 max-width, 因为它们是制作响应式页面时反复用到的两个媒体特性。

现代桌面浏览器和移动电话浏览器对媒体查询的支持程度很高。不过, Internet Explorer 8 及以下版本并不支持媒体查询, 相关解决方案请参见 12.6 节。

1. 媒体查询语法和示例

Peter Gasston 的 *The Book of CSS3* (No Starch Press, 2011) 一书写得很棒。该书对媒体查询的语法做了很好的归纳。以下是媒体查询的基本语法。

- ❑ 指向外部样式表的链接:

```
<link rel="stylesheet" media="logic
→ type and (feature: value)"
→ href="your-stylesheet.css" />
```

- ❑ 位于样式表中的媒体查询:

```
@media logic type and (feature:
→ value) {
/* 目标CSS样式规则写在这里 */
}
```

我将在随后解释有关的语法, 不过几个简单的例子 (参见图 12.4.3 和图 12.4.4) 有助于在上下文中理解这些语句。示例中的查询是相同的, 但它们呈现样式的方式却是不同的。图 12.2.3 中的示例可以翻译为“仅当媒体类型为 screen 且视觉区域最小宽度为 480 像素时, 加载并使用 styles-480.css 中的样式规则”。图 12.4.4 中的示例可以翻译为“仅当媒体类型为 screen 且视觉区域最小宽度为 480 像素时, 使用下面的样式规则”。(关于视觉区域的含义, 参见本节最后的补充材料。)对于响应式页面, 大多数情况下我们需要将媒体查询放到样式表中。

```
...
<head>
  <meta charset="utf-8" />
  <title>Media query in link</title>
  <meta name="viewport" content="width=
→ device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="base.css"
→ media="all" />

  <!--
  The logic is only.
  The type is screen.
  The feature: value is min-width: 480px.
  -->
  <link rel="stylesheet" media="only
→ screen and (min-width: 480px)"
→ href="styles-480.css" />
</head>
...
```

图 12.4.3 base.css 中的样式用于所有的输出设备。styles-480.css 中的样式则仅用于支持媒体查询且视觉区域宽度不低于 480 像素的浏览器

① WebKit 是 Chrome、Safari 等浏览器使用的网页引擎和 JavaScript 引擎开源程序。——译者注

② Mozilla 是 Firefox 等浏览器的基础程序。——译者注

```

/* 常规样式写在这里。
   每个设备都能获取它们，
   除非被媒体查询中的样式规则覆盖 */
body {
    font: 200%/1.3 sans-serif;
}

p {
    color: green;
}

/*
The logic is only.
The type is screen.
The feature: value is min-width: 480px.
*/
@media only screen and (min-width:
→ 480px) {
    p {
        color: red;
        font-weight: bold;
    }
}

```

图 12.4.4 这个粗略的示例包含了默认的段落样式，接着是当媒体查询结果为真时对段落文本的修改。我将这份样式表保存为 basic-media-query.css，并在如图 12.4.5 所示的页面中加载该文件。结果如图 12.4.6、图 12.4.7 和图 12.4.8 所示

我创建了一个测试页（如图 12.4.5 所示），并指向包含图 12.4.4 中代码的样式表。可以查看该页面在 iPhone（图 12.4.6）、iPad（图 12.4.7）及窄的桌面浏览器（图 12.4.8）中显示的样子。

回到语法，让我们看看这些代码的组成部分。

- *logic*（逻辑）部分是可选的，其值可以是 *only* 或 *not*。*only* 关键字可以确保旧的浏览器不读取余下的媒体查询，同时一并忽略链接的样式表。*not* 关键字可以对媒体查询的结果求反，让其反面为真。例如，使用 *media="not screen"* 会在媒体类型为 *screen* 以外的任何类型时加载样式表。

- *type*（类型）部分是媒体类型，如 *screen*、*print* 等。
- *feature: value* 对是可选的，但一旦包含它们，它们必须用括号包围且前面要有 *and* 这个字。*feature* 是预定义的媒体特性，如 *min-width*、*max-width* 或者 *resolution*。对 *color*、*color-index* 和 *monochrome* 特性来说，*value* 是可选的。

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Basic media query example</title>
    <meta name="viewport" content="width=
→ device-width, initial-scale=1.0" />
    <link rel="stylesheet"
    href="assets/
→ css/basic-media-query.css" />
</head>
<body>
    <p>Hi, I'm a paragraph. By default, I'm
→ green and normal. But get me in a
→ viewport that's at least 480px wide,
→ and I get red and bold!</p>
</body>
</html>

```

图 12.4.5 这个样式表包含了需要加载的媒体查询，跟其他样式表一样

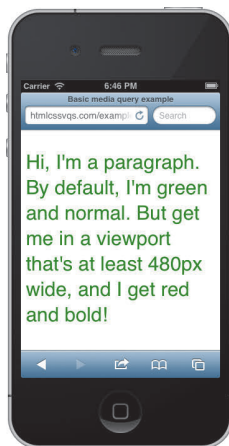


图 12.4.6 在 iPhone 纵向模式下，Mobile Safari 的视觉区域为 320 像素宽，因此文本仍为样式表中基准样式定义的绿色（它从浏览器的默认样式继承了 *font-weight* 的常规值）。不过，当页面显示在 iPad 中时……

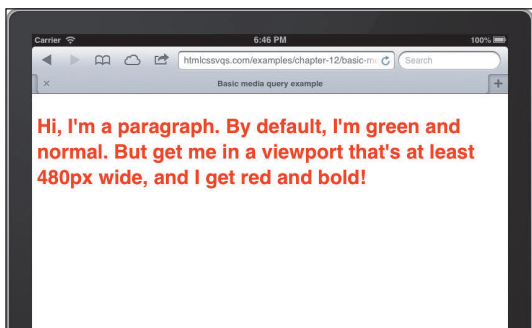


图 12.4.7 ……由于在 iPad 纵向模式下浏览器的视觉区域为 768 像素宽，而宽度大于等于 480 像素时会触发媒体查询，文本变成以红色粗体显示。在 iPhone 横向模式下该效果也会生效，因为该模式下视觉区域的宽度刚好为 480 像素

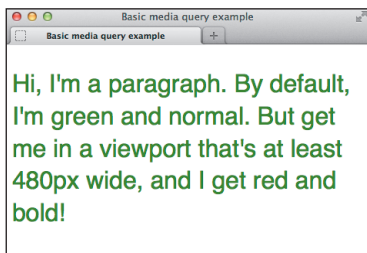


图 12.4.8 现代桌面浏览器也理解媒体查询。这是 Firefox，通过拖拽窗口的右下角使视觉区域的宽度小于 480 像素，因此文本显示为绿色，font-weight 为常规值。如果让窗口变大，使其宽度至少为 480 像素，文字将立即以红色粗体显示，无须刷新页面

可以使用 `and` 将多个特性和值的组合串接起来，还可以创建一系列媒体查询（使用逗号分隔每个媒体查询）。在用逗号分隔的媒体查询列表中，如果有一个媒体查询为真，则整个媒体查询列表为真。图 12.4.9 和图 12.4.10 显示了多种媒体查询。

2. 小结

通过媒体查询，可以根据设备的媒体属性应用不同的样式。尽管媒体查询包含了很多功能，但其中 `min-width` 和 `max-width` 是创

建响应式网页时用的最多的。

```
...
<link rel="stylesheet" media="only
→ screen and (min-width: 480px) and
→ (max-width: 767px)"
→ href="styles.css" />

<link rel="stylesheet" media="only
→ screen and (orientation:
→ landscape)" href="styles.css" />

<link rel="stylesheet" media="only print
→ and (color)" href="color-pages.css" />

<link rel="stylesheet" media="only print
→ and (monochrome)" href="monochrome-
→ pages.css" />

<link rel="stylesheet" media="only screen
→ and (color), projection and (color)"
→ href="styles.css" />
</head>
<body>
...
```

图 12.4.9 当媒体查询为真时加载外部样式表的示例

在媒体查询中使用 em

到目前为止，本书在媒体查询的示例（参见图 12.4.3、图 12.4.4、图 12.4.9 和图 12.4.10）中均使用像素作为 `min-width` 和 `max-width` 值的单位，这样做有助于帮助读者理解媒体查询的工作原理。不过在实践中，最好使用 `em`，因为媒体的触发条件往往与访问者浏览器中显示的字体大小有关。Lyza Gardner 对使用 `em` 的好处作了进一步说明，参见 <http://blog.cloudfour.com/the-ems-have-it-proportional-media-queries-ftw/>。

在下一部分构建响应式网页的示例页面时，媒体查询将使用 `em` 值。同时，注释中将说明它们产生的以像素为单位的近似值。

```

/* 基准样式
----- */

/* 针对所有设备的基准样式 */
...

/* 开始媒体查询
----- */
@media only screen and (min-width:
→ 480px) and (max-width: 767px) {
    /* 样式规则 */
}

@media only screen and (orientation:
→ landscape) {
    /* 样式规则 */
}

@media only print and (color) {
    /* 样式规则 */
}

@media only print and (monochrome) {
    /* 样式规则 */
}

@media only screen and (color), projection
→ and (color) {
    /* 样式规则 */
}

```

图 12.4.10 这些媒体查询与图 12.4.9 中的是相同的，只是直接出现在样式表中

提示 任何位于媒体查询以外的基准样式规则都会应用于所有的设备。可以使用媒体查询覆盖这些样式规则。需要说明的是，媒体查询样式规则声明仅在与常规样式冲突时进行覆盖，如图 12.4.3 中的 `color: green;`。如果媒体查询之前的 `p` 的样式规则包含 `font-style: italic;`，那么媒体查询为真时段落文本仍以斜体显示，因为媒体查询内的 `p` 的样式规则并未设置 `font-style`。

提示 如果你有 Mac，可以使用苹果推出的免费的 iOS Simulator（iOS 模拟器）测试示例页面在 iPhone 和 iPad 中的显示情况。参见 20.1 节。

提示 CSS3 媒体查询规范中列出了关于所有媒体特性的描述（www.w3.org/TR/css3-mediaqueries/#media1）。

理解视觉区域及使用视觉区域 meta 元素

视觉区域（viewport）指的是浏览器（包括桌面浏览器和移动浏览器）显示页面的区域。它不包含浏览器地址栏、按钮这样的东西，只是浏览区域。媒体查询的 `width` 特性对应的是视觉区域的宽度。不过，`device-width` 特性不是，它指的是屏幕的宽度。

在移动设备（如 iPhone）上，默认情况下这两个值通常不一样。Mobile Safari（iPhone 的浏览器）的视觉区域默认为 980 像素宽，但 iPhone 的屏幕只有 320 像素宽（iPhone 的 Retina 显示屏的屏幕分辨率有 640 像素宽，但它们是在相同的空间挤入两倍的像素，因此设备宽度仍为 320 像素）。

因此，iPhone 会像设为 980 像素宽的桌面浏览器那样显示页面，并将页面缩小以适应 320 像素的屏幕宽度（在纵向模式下），如图 12.4.11 所示。结果，当你在 Mobile Safari 中浏览大部分为桌面浏览器建立的网站时，会显示将这些网站缩小了的样子。在横向模式下也是这样处理的，只不过宽度为 480 像素（iPhone5 是 568 像素）。如图 12.4.12 所示，如果不进行放大，页面通常是难以阅读的（注意不同设备的默认视觉区域宽度并不相同）。

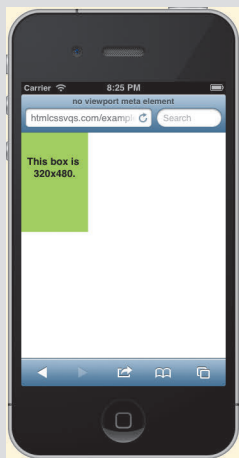


图 12.4.11 我的测试页面包含一个 320 像素 × 480 像素的绿色 div。Mobile Safari 的视觉区域默认为 980 像素宽，因此 iPhone 会将该 div 缩小，以在 320 像素宽的屏幕内显示它。这就是这个绿色盒子大约占据屏幕宽度的三分之一（即 320/980）的缘故

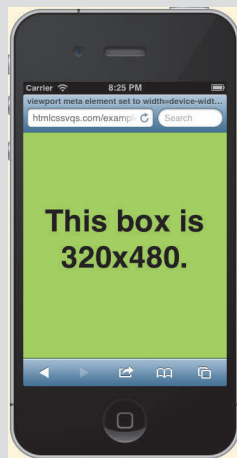


图 12.4.12 这个测试页面的代码与图 12.4.11 中页面的代码相比，除了包含设置了 `width=device-width` 的视觉区域 meta 元素以外，其他内容是完全相同的。如你所见，现在的视觉区域宽度与屏幕宽度是相同的

幸好，有一种快速解决方案。在页面的 head 部分添加视觉区域 meta 元素即可。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Your page title</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  ...
</head>
<body>
  ...
```

这段代码的重点是 `width=device-width`。有了这条语句，视觉区域的宽度会被设成与设备宽度（对 iPhone 来说为 320 像素）相同的值，因此在纵向模式下该宽度的页面内容会填充整个屏幕（如图 12.4.12 所示）。如果不包含这一语句，使用媒体查询的 `min-width` 和 `max-width` 特性将不会得到预期的结果。

代码中的 `initial-scale=1` 部分对 `width` 和 `device-width` 值没有影响，但通常会包含这一语句。它将页面的默认缩放级别设成了 100%，换成纵向模式也一样。（注意，iOS6 之前的版本有一个 bug，它会裁掉一部分内容，参见 <http://adactio.com/journal/5802/>。）如果不设置 `initial-scale=1`，在 iPhone 中，手机从纵向模式改为横向模式时，网页会被放大，从而让布局与纵向模式的一致。

12.5 组合使用

理解了可伸缩图像、弹性布局和媒体查询的知识之后，就可以将它们组合在一起，创建响应式网页。

本书不会演示应用于每个媒体查询块内的全部样式规则。本节将重点关注页面扩张或收缩时切换内容的显示方式所需要考虑的要点。重要的是了解如何建立响应式网站，以及用于实现响应式网站的媒体查询类型。完整的页面和代码见 www.htmlcssvqs.com/8ed/12。

需要说明的是，并不需要先做出一个定宽的设计，再将它转换成响应式的页面。第 11 章演示了如何构建定宽页面，本章（尤其是本节）演示的是如何从零开始构建响应式的页面，就像从未建立过定宽的页面似的。

1. 创建内容和 HTML

一切应该从可靠的、认真考虑过的内容开始。如果你试着用占位符文本（如没有任何含义的 *lorem ipsum*^①）设计和构建你的网站，当你填入真正的内容以后，你可能会发现形式与内容结合得不好。因此，应该尽可能地将内容采集工作提前，从而对设计和开发满足访问者（和你）需求的网站更有信心。

示例页面的底层 HTML 与第 11 章的页面的代码是相同的，只是我在 `head` 元素中添加了 `<meta name="viewport" content="width=device-width, initial-scale=1"/>`。关于这行代码的作用，参见上一节的补充材料“理解视觉区域及使用视觉区域 `meta` 元素”。

2. 移动优先方法

我会遵循移动优先为页面设计样式，推荐你在设计网页时也遵循这一点。下面是具体过程。

(1) 首先为所有的设备提供基准样式（如图 12.5.1 所示）。这同时也是旧版浏览器和功能比较简单的设备显示的内容。基准样式通常包括基本的文本样式（字体、颜色、大小），内边距、边框、外边距和背景（视情况而定），以及设置可伸缩图像的样式。通常，在这个阶段，需要避免让元素浮动，或对容器设定宽度，因为最小的屏幕并不够宽。内容将按照常规的文档流由上到下进行显示。网站的目标是在单列显示样式中是清晰的、中看的（如图 12.5.2 所示）。这样，网站对所有的设备（无论新旧）都具有可访问性。在不同设备下，外观可能有差异，不过这是在预期之内的，完全可以接受。

(2) 从这种样式开始，使用媒体查询逐渐为更大的屏幕（或其他媒体特性，如 *orientation*）定义样式。大多数时候，`min-width` 和 `max-width` 媒体查询特性是最主要的工具。

这是渐进增强在实战中的应用。（请回顾本书前言中的“渐进增强：一种最佳实践”部分。）处理能力较弱的（通常也是较旧的）设备和浏览器会根据它们能理解的 CSS 显示网站相对简单的版本。处理能力较强的设备和浏览器则显示增强的版本。所有人都能获得到网页的内容。

^① 在西方排版、设计领域常用一段以“*lorem ipsum*”开头的拉丁文作为占位符，以测试排版、设计的效果。这段文本是经过改造的、没有意义的拉丁文，常用“*lorem ipsum*”指代这段文本。——译者注

```

/* 基准样式
----- */
body {
    font: 100%/1.2 Georgia, "Times New
    → Roman", serif;
    margin: 0;
    ...
}

* { /* 参见第11章 */
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}

.page {
    margin: 0 auto;
    max-width: 60em; /* 960px */
}

h1 {
    font-family: "Lato", sans-serif;
    font-size: 2.25em; /* 36px/16px */
    font-weight: 300;
    ...
}

.about h2, .mod h2 {
    font-size: .875em; /* 15px/16px */
}

.logo,
.social-sites,
.nav-main li {
    text-align: center;
}

/* 创建可伸缩图像 */
.post-photo, .post-photo-full,
.about img, .map {
    max-width: 100%;
}
...

```

图 12.5.1 应用于所有视觉区域（小屏幕和大屏幕设备）的基准样式示例。这些样式规则与你在本章之前看到的其他代码是类似的，只是它们没有由媒体查询包围。注意我为整个页面设定了 60em 的最大宽度（通常等价于 960 像素），并使用 auto 外边距让其居中。我还让所有的元素使用 `box-sizing: border-box;`，将大多数图像设置为可伸缩图像

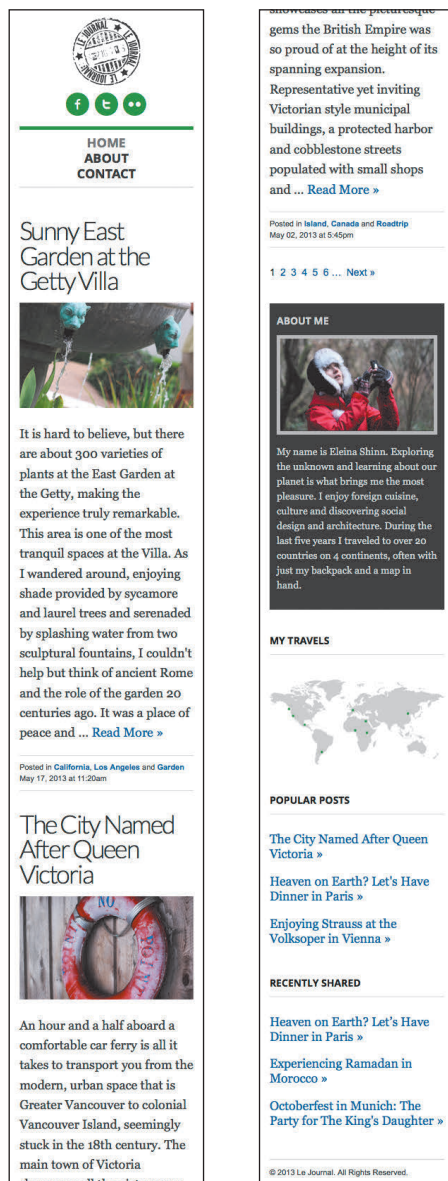


图 12.5.2 仅应用了基础样式的页面的开始点。这个页面在所有的浏览器中都是线性的（图中右侧的部分出现在左侧部分的下面。）此截图展示的是不支持媒体查询的旧浏览器中页面的显示效果。在这种状态下，依然保持了很高的可用性。由于没有设定容器宽度，因此在桌面浏览器中查看页面时，内容的宽度会延伸至整个浏览器窗口的宽度

3. 逐步完善布局

如果用响应式 Web 设计的术语，这一过程可以表述为，使用媒体查询为页面中的每个断点（breakpoint）定义样式。断点即内容需作适当调整的宽度。在本例中，应用基准样式规则后，我为下列断点创建了样式规则。记住，对于每个最小宽度（没有对应的最大宽度），样式定位的是所有宽度大于该 min-width 值的设备，包括台式机及更早的设备。

- ❑ 最小宽度为 20em，通常为 320 像素（如图 12.5.3 和图 12.5.4 所示）。定位纵向模式下的 iPhone、iPod touch、各种 Android 以及其他移动电话。

```
/* 基准样式
----- */
...

/* 20em (大于等于320px)
----- */
@media only screen and (min-width:
20em) {
    .nav-main li {
        border-left: 1px solid #c8c8c8;
        display: inline-block;
        text-align: left;
    }

    .nav-main li:first-child {
        border-left: none;
    }

    .nav-main a {
        display: inline-block;
        font-size: 1em;
        padding: .5em .9em .5em 1.15em;
    }
}
```

图 12.5.3 这里针对视觉区域不小于 20em 宽的浏览器修改了主导航的样式。（在假定 body 元素字体大小为 16 像素的情况下，20em 通常等价于 320 像素，因为 $20 \times 16 = 320$ 。）这样，链接会出现在单独的一行，而不是上下堆叠（如图 12.5.4 所示）。我没有将这些放到基础样式表中，因为有的手机屏幕比较窄，可能会让链接显得很局促，或者分两行显示

- ❑ 最小宽度为 30em，通常为 480 像素（如图 12.5.5 ~ 图 12.5.8 所示）。定位大一些的移动电话，以及横向模式下的大量 320 像素设备（iPhone、iPod touch 及某些 Android 机型）。
- ❑ 最小宽度介于 30em（通常为 480 像素）和 47.9375em（通常为 767 像素）之间，参见图 12.5.9 和图 12.5.10）。这适用于处于横向模式的手机、一些特定尺寸的平板电脑（如 Galaxy Tab 和 Kindle Fire），以及比通常情况更窄的桌面浏览器。
- ❑ 最小宽度为 48em，通常为 768 像素（如图 12.5.11 ~ 图 12.5.13 所示）。这适应于常见宽度及更宽的 iPad、其他平板电脑和台式机的浏览器。

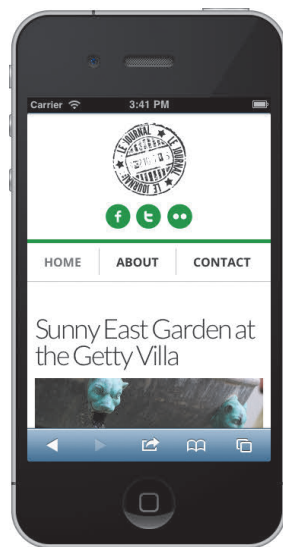


图 12.5.4 根据图 12.5.3 中的媒体查询，主导航显示为一行，每个链接之间由灰色的竖线分隔。这个样式会在 iPhone（以及很多其他的手机）中生效，因为它们在纵向模式下是 320 像素宽。如果你希望报头更矮一些，可以让标识居左，社交图标居右。不过我将这种样式用在下一个媒体查询中（参见图 12.5.5 和图 12.5.6）

```
/* 基准样式
----- */
...

/* 20em (大于等于320px)
----- */
@media only screen and (min-width: 20em) {
    ...
}

/* 30em (大于等于480px)
----- */
@media only screen and (min-width: 30em) {
    .masthead { position: relative; }

    .social-sites {
        position: absolute;
        right: -3px;
        top: 41px;
    }

    .logo {
        margin-bottom: 8px;
        text-align: left;
    }

    .nav-main {
        margin-top: 0;
    }
}
```

图 12.5.5 现在，样式表中有了定位视觉区域至少为 30em（通常为 480 像素）的设备的媒体查询。这样的设备包括屏幕更大的手机，以及横向模式下的 iPhone（参见图 12.5.6）。这些样式会再次调整报头。前两个规则跟第 11 章使用的规则一样，效果也一样

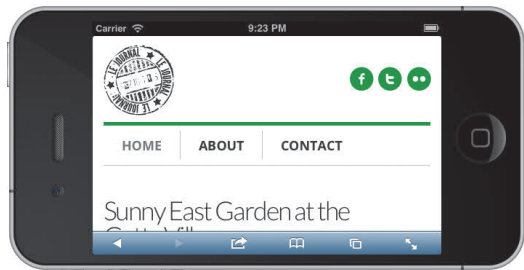


图 12.5.6 这是在 30em 宽的屏幕下显示页面顶部的样子，报头是完整的。在更大的视觉区域，报头宽度会自动调大

```
...

/* 30em (大于等于480px)
----- */
@media only screen and (min-width: 30em) {

    ... 报头样式 ...

    .post-photo {
        float: left;
        margin-bottom: 2px;
        margin-right: 22px;
        max-width: 61.667%;
    }

    .post-footer {
        clear: left;
    }

}
```

图 12.5.7 继续在同一个媒体查询块内添加样式，让图像向左浮动，并减少其 max-width，从而让更多的文字可以浮动到其右侧。11.12 节解释了为什么要在博客摘要下面的页脚上设置 clear: left;

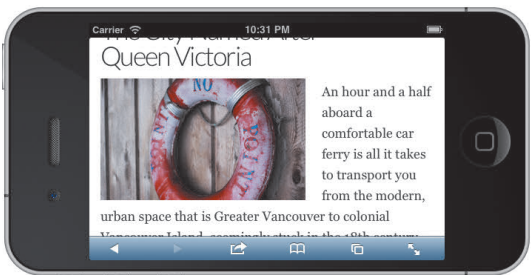


图 12.5.8 文本环绕在浮动图像周围

你的断点可能跟此处用的不同。这取决于哪些断点适合你的内容和设计。为适应更宽的视觉区域，一般不会创建超过 48em 的断点。你也不一定要严格按照设备视觉区域的宽度创建断点。如果一个基于 (min-width: 36em) 的断点非常适合你的内容，就可以大胆地使用这个断点。

```

/* 基准样式
----- */
...

/* 20em (大于等于320px)
----- */
@media only screen and (min-width: 20em) {
    ...
}

/* 30em (大于等于480px)
----- */
@media only screen and (min-width: 30em) {
    ...
}

/* 30em - 47.9375em
   (在480px和767px之间)
----- */
@media only screen and (min-width: 30em)
→ and (max-width: 47.9375em) {
    .about { /* self-clear float */
        overflow: hidden;
    }

    .about img {
        float: left;
        margin-right: 15px;
    }
}

```

图 12.5.9 现在，让 About Me 图像向左浮动。不过，这种样式仅当视觉区域的宽度在 30em（如图 12.5.10 所示）和 47.9375em 之间时才生效。超过这个宽度会让布局变成两列布局，About Me 文字会再次出现在图像的下面。这个随后实现



图 12.5.10 浮动的 About Me 图像已显示为其本来的尺寸（270 像素宽），它旁边的空间太小，无法很好地容纳文本。这就是之前减少其 max-width 的原因（如图 12.5.7 和图 12.5.8 所示）

```

/* 基准样式
----- */
...

/* 20em (大于等于320px)
----- */
@media only screen and (min-width: 20em) {
    ...
}

/* 30em (大于等于480px)
----- */
@media only screen and (min-width: 30em) {
    ...
}

/* 30em ~ 47.9375em
   (在480px和767px之间)
----- */
@media only screen and (min-width: 30em) and
→ (max-width: 47.9375em) {
    ...
}

/* 48em (大于等于768px)
----- */
@media only screen and (min-width: 48em) {
    .container {
        background: url(..img/bg.png)
        → repeat-y 65.9375% 0;
        padding-bottom: 1.875em;
    }

    main {
        float: left;
        width: 62.5%; /* 600px/960px */
    }

    .sidebar {
        float: right;
        margin-top: 1.875em;
        width: 31.25%; /* 300px/960px */
    }

    .nav-main { margin-bottom: 0; }
}

```

图 12.5.11 这是最终的媒体查询，定位至少有 48em 宽的视觉区域。该媒体查询对大多数桌面浏览器（参见图 12.5.13）来说都为真，除非用户让窗口变窄。它同时也适用于纵向模式下的 iPad 及其他一些平板电脑，如图 12.5.12 所示

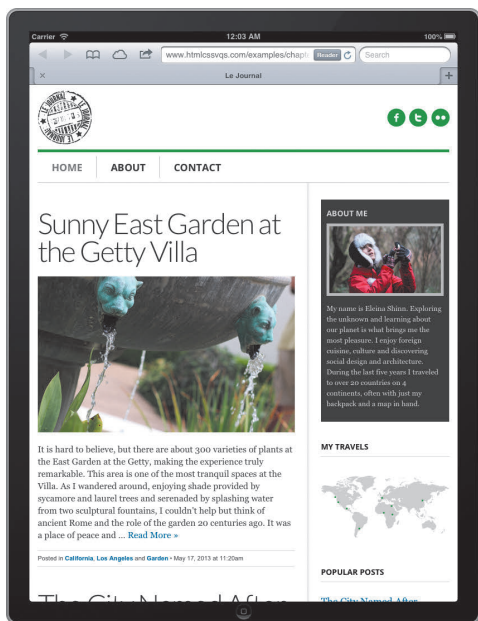


图 12.5.12 页面变得完整了。这里显示的是在 iPad 中页面显示的样子，在桌面浏览器中（尽管要宽一些）也是类似的。由于宽度是用百分数定义的，因此主体内容栏和附注栏会自动伸展

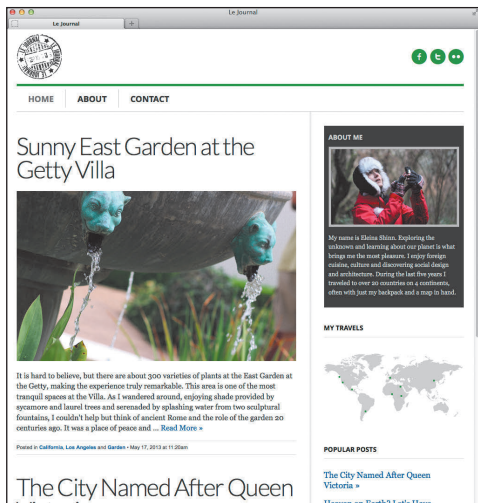


图 12.5.13 在一个至少有 960 像素宽的浏览器中的样子。看起来很像我们在第 11 章中创建的定宽布局。如果让浏览器的宽度变窄，布局就会自动变换为移动设备上应该显示的样子

4. 构建响应式 Web 页面

(1) 创建内容和 HTML。

(2) 在 HTML 页面的 head 元素中，输入 `<meta name="viewport" content="width=device-width" />` 或 `<meta name="viewport" content="width=device-width, initial-scale=1" />`。

(3) 创建适用于所有设备的基准样式（参见“移动优先方法”）。确保页面中的图像可伸缩，参见 12.2 节。

(4) 识别出适合你的内容的断点。创建相关的媒体查询，让布局适应从小屏幕到大屏幕的不同可视区域宽度。（参见“逐步完善布局”）。

(5) 如果需要为第 (4) 步中页面的一些内容指定宽度，使用百分数（参见 12.3 节）。

(6) 选择你希望的旧版 IE 显示页面的方式（参见 12.6 节）。

(7) 开始测试（参见“测试响应式网页”）。

(8) 根据需要，修改第 (3) ~ (5) 步中的 CSS，并进行测试，直到页面在各种设备下都呈现出预期的效果。

提示 Eivind Uggedal 的 <http://mediaqueri.es> 网站汇集了大量现实的响应式网站（数量还在增加），值得借鉴。

提示 Screen Sizes 网站（<http://screensiz.es>）提供了流行设备和显示屏的分辨率和设备宽度信息。使用媒体查询的时候，这些信息可能很有用。

提示 Maximiliano Firtman 维护了一个现代移动设备对 HTML5 和 CSS3 支持情况的表格，见 <http://mobilehtml5.org>。（其中大量信息属于 HTML5 高级特性，本书并未涉及。）

测试响应式页面

你可能希望在发布响应式页面之前先在移动设备和桌面浏览器上对其测试一遍。

构建响应式页面的时候，你可以放大或缩小桌面浏览器的窗口，模拟不同手机和平板电脑的视觉区域尺寸。然后再对样式进行相应的调整。这当然是一种不够精细的办法，但它确实有助于建立有效的样式，从而减少在真实设备上优化网站的时间。Malte Wasserman 提供的 Viewport Resizer (<http://lab.maltewassermann.com/viewport-resizer/>) 对这一工作很有帮助。

不过不要就此打住。对你的浏览器多进行几次放大或缩小，测试布局在桌面浏览器各种宽度下如何调整显示。你可能会找到其他需要引入额外媒体查询的断点。

20.2 节总结了对移动和桌面测试的方法。

对 Retina 及类似显示屏使用媒体查询

有时你可能希望为高像素密度设备设定样式（参见 5.8 节）。一种常见的用例是为这样的显示屏准备双倍尺寸（2x）的 sprite，从而让图像显得更锋利。（参见 14.10 节。）

假设你的 sprite 的原始尺寸是 200 像素 × 150 像素，其中每个图像都用 1 像素分隔。创建一个双倍大小的版本（400 像素 × 300 像素），每个图像之间就有 2 像素的间隔。同时，sprite 中的每个图像都是原始尺寸的 2 倍。针对高像素密度设备，可以使用下面的媒体查询：

```
@media (-o-min-device-pixel-ratio: 5/4),
(-webkit-min-device-pixel-ratio:1.25),
(min-resolution: 120dpi) {
  .your-class {
    background-image:url(sprite-2x.png);
    background-size: 200px 150px;
  }
}
```

注意 background-size 设置成了原始尺寸，而不是 400 像素 × 300 像素。这样会让图像缩小，为原始尺寸创建的样式对 2x 版本也有效。

12.6 兼容旧版 IE

对于移动优先方法，有一点需要注意，就是 Internet Explorer 8 及以下的版本不支持媒体查询。这意味着这些浏览器只会呈现媒体查询以外的样式，即基准样式。目前，在世界上大部分地区，使用 IE6 和 IE7 的用户已经非常少了。因此，真正费脑筋去考虑的是 IE8，它在全世界所占的份额不到

9%，且这个数字还在下降（详情参见 <http://gs.statcounter.com>）。

对于 IE8（及更早的版本），有三种解决方法。

- ❑ 什么都不做。让网站显示基本的版本。
- ❑ 为它们单独创建一个样式表，让它们显示网站最宽的版本（不会形成响应

式的网页)。一种做法是复制一份常规的样式表,将其命名为 old-ie.css 之类的文件名。将媒体查询语句去掉,但保留其中的样式规则。在 HTML 中添加条件注释,从而让不同的浏览器都能找到正确的样式表(参见图 12.6.1)。

- 如果希望页面有响应式的效果,就在页面中引入 respond.min.js (参见图 12.6.2)。Scott Jehl 创建了这段简短的代码,它让 min-width 和 max-width 媒体查询对旧版 IE 也有效。参见补充材料。

```
...
<head>
...

<!--[if gt IE 8]><!-->
<link rel="stylesheet"
→ href="css/styles.css" />
<!--<![endif]-->

<!--[if lt IE 9]>
<link rel="stylesheet"
→ href="css/old-ie.css" />
<![endif]-->
</head>
...
```

图 12.6.1 这里有很多看起来很怪异的代码。你可能不熟悉的代码是条件注释(参见 www.quirksmode.org/css/condcom.html)。第一个条件注释包含的是为 IE8 及更旧版本以外的所有浏览器准备的样式表。第二个包含的则是仅为 IE9 以下的 IE 准备的样式表

```
...
<head>
...
<link rel="stylesheet"
→ href="css/styles.css" />

<!--[if lt IE 9]>
<script src="js/respond.min.js">
→ </script>
<![endif]-->
</head>
...
```

图 12.6.2 将 src 值里的 js 部分替换为你网站中的地址(如果跟这里的不一样的话)。设置好以后,IE8 及以下版本会理解 min-width 和 max-width 媒体查询,并呈现相应的样式。这样做的话,就没有必要将 IE 样式表分离出来了。这个 script 元素外围的条件注释是可选的,不过如果包含的话,就只有 IE8 及以下版本会加载 respond.min.js

第二种方法更可靠,它让 IE8 用户也能看到网站的完整布局。

提示 如果使用 Sass、LESS 或 Stylus 这类 CSS 预处理器,就可以自动创建为旧版 IE 准备的样式表。Nicolas Gallagher 分享了一个使用 Sass 的方法(<http://nicolasgallagher.com/mobile-first-css-sass-and-ie/>)。

使用 Respond.js

可以访问 <https://github.com/scottjehl/Respond>, 点击 ZIP 按钮下载 Respond.js。下载到电脑后,打开该 zip 文件,然后将 respond.min.js 复制到你网站的一个文件夹里。如果这段脚本不起作用,不妨看看上述 GitHub 链接中的 Support & Caveats (支持与注意事项)部分。Respond.js 并不是一个适用于所有网站的完美方案。

本章内容

- 什么是 Web 字体
- 在哪里能找到 Web 字体
- 下载第一个 Web 字体
- 理解 @font-face 规则
- 为 Web 字体添加样式
- 为 Web 字体应用斜体和粗体
- 使用 Google Fonts 的 Web 字体

过去几年，我们看到了在万维网上使用字体的复兴。过去，我们通常只能在非常有限的字体集中进行选择。我们在第 10 章中提到过，最好使用那些用户很可能在自己的电脑上安装了字体。这也是大多数网站将 body 字体设为 Arial、Georgia、Verdana 或 Trebuchet MS 的原因。

现在有了 Web 字体，我们在开发 Web 项目时有了大量字体可供选择。因此，Web 设计师可以为访问者提供更为精细的体验。经历这种改变既让人着迷，又令人激动。

本章解释了 Web 字体的基础，如何使用自托管的 Web 字体（例如 Font Squirrel 提供的字体），如何使用 Google Font。学会如何使用自托管的字体需要花费较多的时间。如果你想在自己的网站上尽快用上 Web 字体，可以阅读 13.7 节。

13.1 什么是 Web 字体

CSS 规则 @font-face 为 Web 字体创造了可能，该样式规则允许 CSS 链接到服务器上的一种字体供网页使用。

很多人认为 Web 字体是新生事物。实际上，Web 字体大约在 1998 年就产生了。Netscape Navigator 4 和 Internet Explorer 4 均采用了这种技术，但它们的实现都不支持标准的字体文件格式，因此很少有人用到它们。直到将近十年以后，浏览器才开始采纳这种使用更为常见的字体文件格式的标准，Web 字体的使用才变得常见起来。

1. Web 字体文件格式和浏览器支持

Web 字体可以使用一系列文件类型。需要特别指出的是，下面介绍的前三种字体类型是如今经常使用的。

- 内嵌 OpenType (Embedded OpenType, .eot)。在使用 @font-face 时，Internet Explorer 8 及之前的版本仅支持内嵌 OpenType。内嵌 OpenType 是 Microsoft 的一项专有格式，它使用数字版权管理技术防止在未经许可的情况下使用字体。

- ❑ TrueType (.ttf) 和 OpenType (.otf)，台式机使用的标准字体文件类型。TrueType 和 OpenType 得到了 Mozilla Firefox (3.5 及之后的版本)、Opera (10 及之后的版本)、Safari (3.1 及之后的版本)、Mobile Safari (iOS 4.2 及之后的版本)、Google Chrome (4.0 及之后的版本) 及 Internet Explorer (9 及之后的版本) 的广泛支持。这些格式不使用数字版权管理。
 - ❑ Web 开放字体格式 (Web Open Font Format, .woff)。这种较新的标准是专为 Web 字体设计的。Web 开放字体格式的字体是经压缩的 TrueType 字体或 OpenType 字体。WOFF 格式还允许在文件上附加额外的元数据。字体设计人员或厂商可以利用这些元数据，在原字体信息的基础上，添加额外的许可证或其他信息。这些元数据不会以任何方式影响字体的表现，但经用户请求，这些元数据可以呈现出来。Mozilla Firefox (3.6 及之后的版本)、Opera (11.1 及之后的版本)、Safari (5.1 及之后的版本)、Google Chrome (6.0 及之后的版本) 及 Internet Explorer (9 及之后的版本) 均支持 Web 开放字体格式。
 - ❑ 可缩放矢量图形 (Scalable Vector Graphics, .svg)。简言之，应避免对 Web 字体文件使用 SVG。它更多地用于早期 Web 字体，因为它是 iOS 4.1 上移动 Safari 唯一支持的格式 (还有可能引起一些崩溃的情况)。从 iOS 4.2 (于 2011 年初即被广泛使用) 起，移动 Safari 开始支持 TrueType。
- 可以看到，Web 字体的跨平台支持程度

是很高的，因此没有任何理由不在今天的网站上使用它们。

2. 法律问题

从技术层面上说，字体都是小的软件。我认识以字体设计和制作为生的人，这是一种艰苦、细致的创造性活动，没有强大的心智是做不了这份工作的。

出于这一原因，即便 @font-face 功能从一开始就存在，它还是会激怒一些人也就不难理解了。毕竟，如果浏览器能链接并下载某种字体，意味着任何人都能下载并在自己的计算机上安装这种字体，无论他们是否购买了这种字体。

这就是为什么 Web 设计人员和开发人员必须确保网站上用到的任何字体都具有在万维网上使用的许可。大多数厂商和字体服务提供商通过将许可作为字体购买的一部分或菜单选项提供这种许可。

不管选用哪种方式，确保对项目中的使用的 Web 字体具有稳定的权利。为此，你可以查看购买的字体的许可证。有些字体可以在 Adobe Photoshop、Adobe Illustrator 等桌面应用程序中使用，但不能在网站上作为 Web 字体使用。你要购买字体的厂商的网站通常会提到这些信息。如果存在疑惑，可以跟厂商联系，了解哪些权利是允许的。

如果你购买了一个字体，并确切地知道可以将其用做 Web 字体，就可以使用 Font Squirrel 提供的免费 @font-face 生成器 (www.fontsquirrel.com/fontface/generator)。该工具可以将字体转化为在万维网上使用的所有 Web 字体文件类型。

真正的好消息是 Web 字体服务的大量涌现。通过它们，可以在网站上使用多达数千种的 Web 字体 (参见 13.2 节)，其中很多还是免费的。

3. 管理文件尺寸

使用 Web 字体，尤其是使用多种 Web 字体的一个潜在风险是它们可能会增加页面的“重量”。我在这里谈论的不是炸薯条和甜甜圈，我谈论的是千字节和兆字节。

对于所有这些字体来说，在它们显示到页面上之前，都需要下载到用户的计算机上。这会减慢网站加载的速度，尤其是对移动用户来说。我的建议是审慎地选择 Web 字体。如果你发现自己使用了好几个 Web 字体，就应该想办法对这些字体进行整合了。

◎ 构造子集

节省页面重量的一种方法是构造子集 (subsetting)。构造子集是通过仅包含确定使用的字符削减实际字体大小的方法。例如，如果对标题使用 League Gothic 字体，而网站的设计要求标题始终使用大写字母，没有使用小写字母的需要。通过构造子集，可以将小写字母从字体中移除，这样字体文件大小就会减小一些。

此外，对于很多字体，你还可以选择针对特定语言的子集。13.3 节解释了如何在 Font Squirrel 中采用这种方法。

解释子集构造的具体细节已经超出了本书的范围，不过 Font Squirrel 提供的 @font-face 生成器（之前提到过）可以帮助你进行专家级的子集构造。

13.2 在哪里能找到 Web 字体

在网站上使用 Web 字体有两种方式：自托管和 Web 字体服务。这两种方式都是完全有效的选项，不过二者差异很大，各有利弊。当你权衡这些利弊的时候，你会发现并非所有的 Web 字体都是随处可见的。你可能会发现，即便你想采用自托管的方式，你需要的

字体也只能来自 Web 字体服务。你还有可能会发现你希望使用的服务并未提供你想要的某种字体。这时，你可能需要寻找一种接近的替代方案，或者重新考虑你的方法。在作出决定之前，需要花费一些精力权衡所有的选项并灵活处理。

1. 自托管

自托管 Web 字体来源于你自己的服务器，就像其他的资源（如图像、CSS 文件）一样。如果需要与字体相关的花费，通常也是一次性的购买支出，是否将字体文件上传到网站上并将其包含到代码中取决于你自己。在接下来的小节，我们会讲解使用自托管服务（使用的是从 Font Squirrel 下载的一款免费 Web 字体）的详细步骤。

寻找可以自托管的字体是比较容易的，因为它们的数量很多。它们的质量和价格差异很大（有的甚至是免费的）。其中一些流行的自托管字体来源如下。

❑ Font Squirrel (www.fontsquirrel.com)，参见图 13.2.1

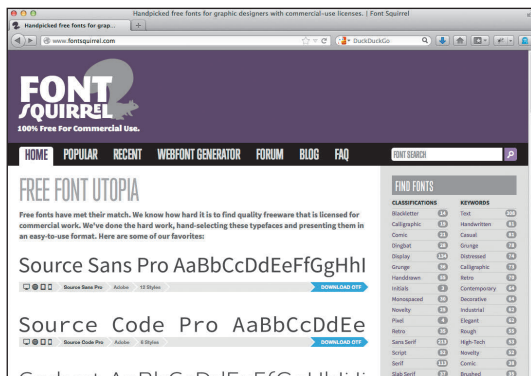


图 13.2.1 Font Squirrel 是寻找可以自行托管的免费 Web 字体的一个流行的网站。在本章接下来的小节里，你将学会如何使用它。你北非必须要了解使用 Google Fonts 这样的 Web 字体服务的全部细节，但了解它们有助于理解 Web 字体的工作原理，并了解一些常见的陷阱

- ❑ MyFonts (<http://myfonts.com>)
- ❑ The League of Moveable Type (www.theleagueofmoveabletype.com), 免费
- ❑ FontShop (www.fontshop.com)

2. Web 字体服务

Web 字体服务通常提供订购 Web 字体的方法。这种方式是指按月或按年支付使用字体的版权费用, 而不是彻底买断字体。Typekit 是这个领域的先驱, 现在已经存在几种服务了。

这些服务托管字体, 为用户提供一小段代码放在其网页中。这段代码可能是 JavaScript, 也可能是 CSS, 这取决于服务自身。它包含了从远程服务器获取字体文件并将其显示在网站上这一过程所需的全部代码。很多人主张使用这种方式, 因为这种方式通常比单独购买字体更便宜, 而且它让用户有机会尝试很多种不同的字体。

一些流行的 Web 字体服务包括:

- ❑ Cloud.typography (www.typography.com/cloud/welcome/)
- ❑ Edge Web Fonts (www.edgfonts.com), 免费
- ❑ Fontdeck (<http://fontdeck.com>)
- ❑ Fonts.com (www.fonts.com/web-fonts)
- ❑ Fontspring.com (www.fontspring.com)
- ❑ Google Fonts (www.google.com/fonts), 免费, 参见图 13.2.2
- ❑ Typekit (<https://typekit.com>)
- ❑ WebINK (www.webink.com)

按理说, Web 字体服务可以提供比自托管更多的特性。如果出现了更好的字体文件或代码, 服务便可以很容易地将它们提供给用户。例如, Typekit 中的一些字体使用基于 PostScript 的轮廓, 在 Windows 浏览器上显示这些字体时, Typekit 会提高它们的品质, 让它们的边缘看起来更平滑。

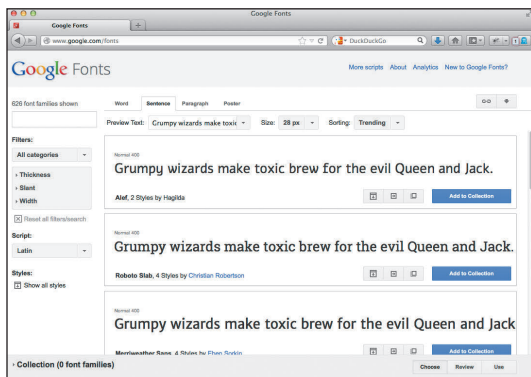


图 13.2.2 Google Fonts 是一种流行的免费 Web 字体服务。我们将在 13.7 节学习如何使用它

在这类服务中, 很多都使用 JavaScript 嵌入获取 Web 字体的代码。这样做有一些好处, 也有一些不好的地方。在这一过程中, JavaScript 做了不少工作, 包括微调 Web 字体显示设置, 为字体本身的加载提供额外的控制。这种控制可以产生更好的体验。

这种奢华的代价当然是你必须百分之百地依赖 JavaScript, 而 JavaScript 会影响页面的性能。在 Web 字体呈现在页面上之前, 用户必须等待 JavaScript 加载完成。这是你决定如何在网站上加入 Web 字体时需要记住的事情。不过别因此拒绝使用 Web 字体服务, 很多人, 包括我自己, 使用 Web 字体都很顺手。

3. Web 字体的质量与显示

不幸的是, 并非所有的 Web 字体都是一样的。不同浏览器中 Web 字体显示的效果具有很明显的差异。一些字体在 Internet Explorer 较早的版本中显示效果并不太好, 这一点尤为明显。

此外, 有些字体在某些字体大小下的显示效果要好于其他字体大小下的效果。它们有可能过于纤细, 不适合用来做正文字体, 或者用做标题时缺乏权重。

当你选择字体时,要尽量检查你即将选择的字体在各种浏览器中显示的效果。现在,由于很多 Web 字体公司提供 Web 字体的实时示例,有的公司提供字体在不同浏览器和平台中的截图,因此上述工作就变得容易多了。

如果你坚持自己做这些测试,可以试试 Web Font Specimen (<http://webfontspecimen.com>) 提供的资源。这是一个测试 Web 字体在各种

环境和字体大小下的显示效果的工具。

4. 那么,如何开始呢

本章接下来的章节将关注两个方面的内容:如何使用自托管的 Web 字体(参见图 13.2.1)和如何使用 Google Fonts(参见图 13.2.2)。其中,Google Fonts 更为简单直接,因此你可以先从那里开始。

图标字体以及如何获取

图标字体指的是包含图标而非字母、数字和标点符号的字体。你可以像对待文字那样对它们添加样式,如设置颜色。

图标字体最棒的一点是,无论使用什么样的字体大小,它们都会平滑缩放,边缘总是很锐利。因此对图标图像来说这是一个很好的选择,因为无需再为让它们在不同类型的显示屏(包括 Retina 显示屏)上或应用不同字体大小的响应式页面上都清晰显示而做额外的工作。

在第 11 章和第 12 章的示例页面的报头中,我使用 Fontfabric (www.fontfabric.com) 提供的 Socialico 创建了社交网站图标。通过将其上传到 Font Squirrel 的 Webfont Generator 创建了页面所需的 Web 字体格式。

Chris Coyier 收集了大量获取图标字体的资源(参见 <http://css-tricks.com/flat-icons-icon-fonts/>),还演示了如何使用 IcoMoon (<http://icomoon.io>) 创建开发人员自己的图标字体(参见 <http://css-tricks.com/video-screencasts/113-creating-and-using-a-custom-icon-font/>)。

13.3 下载第一个 Web 字体

在页面中使用自托管的 Web 字体为元素添加样式时,需要先将 Web 字体文件下载下来。下载免费的 Web 字体很快也很方便。我们将使用 Font Squirrel,不过使用其他字体服务提供商所需的操作与下面的步骤都是相似的。

1. 从 Font Squirrel 下载 Web 字体

(1) 访问 Font Squirrel (www.fontsquirrel.com),选择一种你希望使用的字体。有很多浏览字体的方式,如通过主页,通过 Popular(流行)或 Recent(最新)板块,或者通过搜索。这里选择了 PT Sans(如图 13.3.1 所示)。



图 13.3.1 通过点击字体的名称(如图所示)选择字体,而不要使用右边的蓝色下载按钮(图中未显示)。那个下载链接用于在你的系统中使用对应的字体,而非用于下载 Web 字体文件

(2) 在字体所在的页面选择 Webfont Kit(如图 13.3.2 所示)。

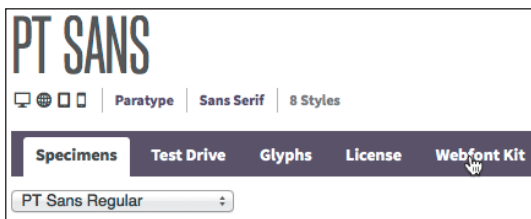


图 13.3.2 Specimens 部分让你可以看到字体在常规、加粗、斜体等文本样式(取决于字体提供的样式)下的字体效果。可以在 Test Drive 部分输入一些文本并查看效果。Glyphs 部分会显示字体中的每个字符,而 License 部分则会说明哪些使用情形是允许的。如果你看上了某款字体,请进入 Webfont Kit

(3) 在 Choose Font Formats (选择字体格式)中,取消 SVG 选项(如图 13.3.3 所示)。如果你需要让使用 Internet Explorer 8 (或更早版本)的访问者看到你的 Web 字体,则需要选择 EOT,否则不选择该项。



图 13.3.3 SVG 不再是推荐的 Web 字体格式,因此这里对它取消选择。如果你认为访问你的网站的 IE8 用户不多,则不必选择 EOT

(4) 选择字体的子集。通常,要么使用默认选项,要么选择网站内容所使用的语言(这里选择 English,如图 13.3.4 所示)。

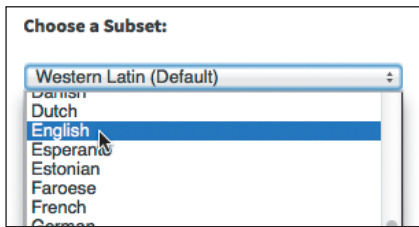


图 13.3.4 对一种 Web 字体构造子集可以减少其包含的字符数量,从而减小文件大小

(5) 点击 DOWNLOAD @FONT-FACE KIT

按钮(如图 13.3.5 所示),下载应该会立即开始。下载的文件为 ZIP 包。



图 13.3.5 将字体下载到你的电脑

(6) 下载完成后,打开压缩包,应该看到一个名为 web fonts 的文件里,里面至少有一个字体文件夹(如图 13.3.6 所示)。

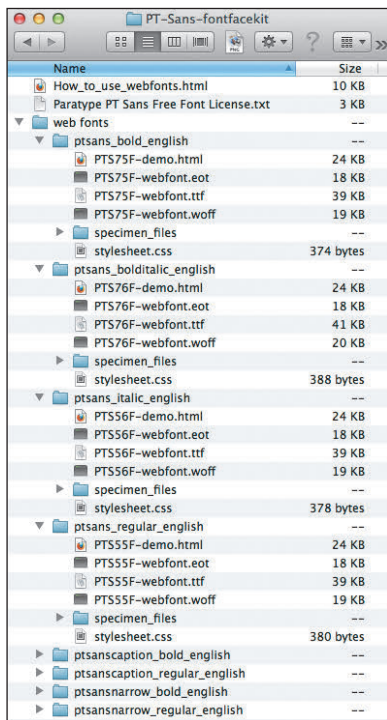


图 13.3.6 天哪,怎么会有这么多文件和文件夹!为什么有这么多呢?在第(3)步(如图 13.3.3 所示)中选择的字体不同,这里看到的文件和文件夹数量也不一样。如果在 Font Squirrel 的 PT Sans 页面中选择 Specimens 标签页(如图 13.3.2 所示),将会看到 8 种字体(如图 13.3.7 所示)。它们代表了 PT Sans 的不同样式和粗细版本。我们选择了三种格式,因此 ZIP 文件里就有八种字体的文件夹,每个文件夹里都有各自的三种格式的字体文件,以及示例 HTML 和 stylesheet.css 文件。在本章中,我仅使用图中显示的四个展开的文件夹中的字体文件以及 stylesheet.css 中的部分代码



图 13.3.7 PT Sans 有三个版本，共 8 种样式：PT Sans（常规、加粗、斜体和加粗斜体）、PT Sans Caption（常规、加粗）以及 PT Sans Narrow（常规、加粗）。本章不会用到 PT Sans Caption 或 PT Sans Narrow。有些字体只有很少的几种样式，有的甚至只有一种样式（如 Bebas Neue）

2. 在 demo.html 文件中查看所选字体

Web 字体文件夹中的每个文件夹都有一个演示该字体的 HTML 页面。其文件名均以 -demo.html 结尾。在浏览器中打开该文件，如图 13.3.8 所示（参见 2.7 节）。



图 13.3.8 从 Font Squirrel 下载的 PT Sans 字体包中的 PTS75F-demo.html 文件。这个文件位于 ptsans_bold_english 文件夹。如果要查看其他的 PT Sans 样式，就打开对应的文件夹中的演示文件（例如 ptsans_regular_english 文件夹中的 PTS55F-demo.html）。每个文件夹里都包含一个 stylesheet.css

演示文件显示 Web 字体是有效的。这很让人激动！在你欢庆胜利之前，我们将在下一节进一步探讨其工作原理，以及如何将它用到你自己的页面中去。

提示 通常，一个页面中不应使用两个（最多三个）以上的 Web 字体，因为引用的字体文件越多，访问者浏览器要下载的文件就越多。这样会减慢页面加载和呈现的速度，特别是对于使用智能手机等设备（意味着较慢的网络连接）的访问者，体验就更糟糕了。需要说明的是，这里说的数量指的是一种或多种字体的两种或三种样式或粗细版本。例如，PT Sans 常规、加粗和斜体算三种，因此如果你引入了另一种 Web 字体系列，则至少引入了四种字体。

提示 你开始下一个项目之前，想获得选择字体的灵感吗？Typekit 团队写了一篇很好的博客文章，这篇文章介绍了大量关于 Web 字体及一般性排版的有用信息。初学者可以尝试阅读 Sites we like（我们喜爱的网站）系列，参见 <http://blog.typekit.com/category/sites-we-like/>。

提示 需要在 Photoshop 中使用这些字体吗？你可以将 Web 字体包中的 TrueType (.tff) 字体装到你自己的计算机上。安装完成后就可以像使用其他字体一样使用该字体了。

13.4 理解 @font-face 规则

现在，是时候看看 Web 字体的工作原理了。让我们看看前四个 PT Sans 字体文件夹中的 stylesheet.css 文件。每个文件都有一条为一种 PT Sans 样式准备的规则，位于看起来有些奇怪的 @font-face { ... } 中（如图 13.4.1 所示）。

```

@font-face {
  font-family: 'pt_sansregular';
  src: url('PTS55F-webfont.eot');
  src: url('PTS55F-webfont.eot?#iefix')
    → format('embedded-opentype'),
    url('PTS55F-webfont.woff')
    → format('woff'),
    url('PTS55F-webfont.ttf')
    → format('truetype');
  font-weight: normal;
  font-style: normal;
}

@font-face {
  font-family: 'pt_sansitalic';
  src: url('PTS56F-webfont.eot');
  src: url('PTS56F-webfont.eot?#iefix')
    → format('embedded-opentype'),
    url('PTS56F-webfont.woff')
    → format('woff'),
    url('PTS56F-webfont.ttf')
    → format('truetype');
  font-weight: normal;
  font-style: normal;
}

@font-face {
  font-family: 'pt_sansbold';
  src: url('PTS75F-webfont.eot');
  ...
}

@font-face {
  font-family: 'pt_sansbold_italic';
  src: url('PTS76F-webfont.eot');
  ...
}
...

```

图 13.4.1 这些样式来自四个 PT Sans 的 stylesheet.css 文件中的第一个。（我改变了样式规则的顺序，从而与本章介绍的顺序一致。）这些代码块都是一样的，只有突出显示的部分不一样。不一样的地方就是 font-family 值，即每种 Web 字体文件名的前缀（注意：你可能注意到在你的 stylesheet.css 中有一个额外的引用 SVG 的 URL。我在这里忽略了这一行代码，因为我没有下载该格式）

这段代码的语法与传统的 CSS 有些不同，因为 @font-face 部分并不是你要添加样

式的元素的选择器。因此，这段样式不会影响任何元素的样式，而是让样式表知道 Web 字体的存在，从而可以在其他的样式规则中用它对文本设置样式。（每个文件夹的演示 HTML 文件都会加载 stylesheet.css 并包含使用该字体的 CSS 规则。）

每条规则的第二行是为字体系列准备的，例如，font-family: 'pt_sansregular'; 或 font-family: 'pt_sansbold';。

这建立了使用某种特定的 Web 字体的名称——对元素设置字体时需要用到的名称，就像常规的字体一样。这个名称可以使用你选择的任何名称。除了可以使用 pt_sansregular 以外，你还可以选择 Banana 或 The Best Font Ever，具体情况取决于个人。事实上，我们在下一节为页面应用 PT Sans 字体时就会改变这个名称。

这条样式规则的下面几行告诉浏览器字体文件的位置。其中包括为所有支持 Web 字体的不同浏览器准备的字体文件格式。语法看起来有些吓人，但我们的目的并不是要完全理解它们。如果你想进一步深入研究，了解这样显示的原理，我向你推荐 Ethan Dunham 发表在 Fontspring 上的博文，参见 www.fontspring.com/blog/further-hardening-of-the-bulletproof-syntax。

13.5 使用 Web 字体设置文本样式

我们讲过了 @font-face 的语法，但还从来没有真正地将 Web 字体放到网页里去。尽管我们将要使用的是从 Font Squirrel 下载的字体，但这些方法也同样适用于其他任何自托管的字体。

在样式表中使用 Web 字体有多种方法。其中的一种便是 Font Squirrel 使用的方法，我

们可以通过查看 stylesheet.css 和演示 HTML 文件了解这种方法。这里将讲解另外一种方法。

没有哪种方法完全正确或完全错误，不过个人推荐我们即将介绍的方法。它更接近你所习惯的使用常规字体为文本设置样式的方式，也更方便管理（尤其是对于刚接触 Web 字体的人来说），而且还能提供 Web 字体无法加载时的备用方法。

幸好，第二种方法所需的 CSS 与第一种方法的差不多。因此，我们将在自己的样式表中使用 PT Sans stylesheet.css 中的第一个文件里的四个 @font-face 样式规则，并根据需要对其进行修改。现在，我们只需要用到字体的常规样式。下一节将用到字体的斜体、加粗以及加粗斜体等样式。

我们将使用图 13.5.1 中的 HTML，并将 CSS 放入 example.css（这个文件目前是空的）。尽管我们要使用的是 PT Sans，不过下面我们将按照一种通用的方式介绍各个步骤，从而可以很方便地将这些步骤应用到 Font Squirrel 中的任何字体。同样地，这种方法也适用于任何自托管的 Web 字体。

使用 Web 字体为普通文本设置字体样式的步骤

(1) 看看 13.3 节中介绍的从 Font Squirrel 下载的 ZIP 文件中的 web fonts 文件夹。如果你看到一个字体的常规文本版本的子文件夹，就打开该子文件中的 stylesheet.css。

(2) 将为常规文本准备的 @font-face 样式规则复制到你自己的样式表中（如图 13.5.2 所示）。

(3) 将字体文件（图 13.5.2 所示的样式表中引用的字体文件）从 Font Squirrel 文件夹复制到包含你的样式表的文件夹（如图 13.5.3 所示）。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Styling Text with a Web Font </title>
  <link rel="stylesheet"
    → href="example.css" />
</head>
<body>
<h1>Local Teen Prefers Vinyl Over Digital</h1>

<p>A local teenager has replaced all her
→ digital tracks with vinyl. "It's <em>
→ really</em> groovy," she said, on the
→ record. Without skipping a beat, she added,
→ "Besides, it's like going through a time
→ warp."</p>

<p>Some of her iPod-toting classmates aren't
→ as enthusiastic. "Yeah, they needle me
→ about it. What a bunch of ones and
→ zeros."</p>
</body>
</html>
```

图 13.5.1 我将从一个简单的 HTML 页面和一个空白的样式表开始。为了简化，两个文件都放在同一个目录。注意，该目录下并没有包含任何 PT Sans 字体文件

```
/* example.css */

@font-face {
  font-family: 'pt_sansregular';
  src: url('PTS55F-webfont.eot');
  src: url('PTS55F-webfont.eot?#iefix')
    → format('embedded-opentype'),
    url('PTS55F-webfont.woff')
    → format('woff'),
    url('PTS55F-webfont.ttf')
    → format('truetype');
  font-weight: normal;
  font-style: normal;
}
```

图 13.5.2 我更新了 example.css，从而让它包含 ptsans_regular_english 文件夹中 stylesheet.css 里的 @font-face 样式规则

(4) 对 font-family 值进行重命名，让该名称成为字体的代号。例如，使用字体的名

称、短横线和风格名称（如图 13.5.4 所示）。（你选择的名称并不重要，重要的是你需要在 font-family 中对该字体支持的所有风格（如常规、加粗、斜体、加粗斜体），都要使用同样的名称。当你通过下一节学会如何应用不同的风格以后，这样做会显得尤其重要。）

(5) 使用你最熟悉的方法，创建为文本元素添加样式的规则。为应用 Web 字体，在样式规则中输入 font-family: 'Web Font Name'; 其中的 Web Font Name 是第 (4) 步中创建的名称（如图 13.5.5 所示）。最后，一定要保存文件。

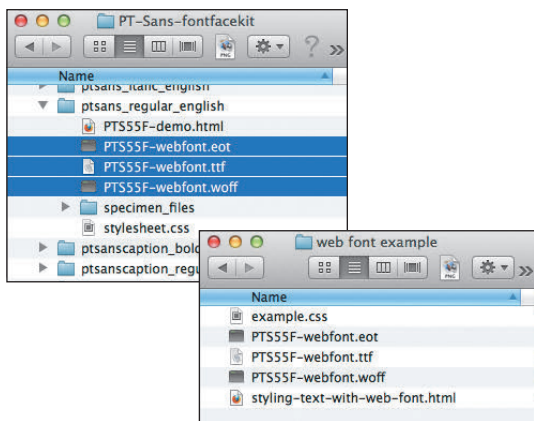


图 13.5.3 我将从 Font Squirrel 下载的文件夹中的三个 PT Sans Regular 字体格式文件（上）复制到我的示例文件夹（下）

```
/* example.css */

@font-face {
    font-family: 'PTSans';
    src: url('PT555F-webfont.eot');
    ...
}
```

图 13.5.4 对 example.css 所做的唯一改动是将 ptsansregular 替换为 PTSans，这样的名称更接近字体本身

```
/* example.css */

body {
    font-family: 'PTSans', sans-serif;
    font-size: 100%;
    line-height: 1.25;
}

@font-face {
    font-family: 'PTSans';
    src: url('PT555F-webfont.eot');
    ...
}
```

图 13.5.5 这是对 body 应用的样式，看起来跟常规的 CSS 没有两样。字体栈中的 font-family 名（在这个例子中为 PTSans）必须与 @font-face 样式规则中的名称一致。这里的 font-size 和 line-height 并不是让 Web 字体生效的必要样式规则

如果你刷新页面，就会看到页面已经能显示 Web 字体（如图 13.5.6 所示）。不过，我们还没有结束。第一眼看上去好像没有问题，但仔细看看就会发现，页面上呈现的粗体和斜体是伪粗体（标题）和伪斜体（单词 really）。我将在下一步介绍如何修正这一问题。



图 13.5.6 由于为 body 应用了 PT Sans 样式，因此整个页面的文字用的都是这种字体

提示 通常，我会将图 13.5.5 中的第一条规则写为 `body { font: 100%/1.25 'PTSans', sans-serif; }`，即使用 `font` 简记法。这里将规则分开来写，是为了更清楚地演示应用 Web 字体的方法本身。

提示 考虑到简化，上述步骤和示例中均假定 HTML、样式表和字体文件均位于同一个目录，但在实践中最好将它们分开。13.6 节的最后一条提示介绍了一种方法。

使用单引号还是双引号

Font Squirrel 提供的 `@font-face` 样式规则使用单引号而非双引号包围字体名称。本书的 CSS 示例中也都是这样做的。在 CSS 里，单引号和双引号是完全一样的，因此你可以根据自己的喜好挑选任何一种方式。

@font-face 方法

Roger Johansson 在他的文章中总结了在样式表中使用 Web 字体的两种方法。文章的 URL 很长，不过访问 www.htmlcssvqs.com/8ed/font-face，页面会跳转到他的网站 (www.456bereastreet.com) 上的那篇文章。他首先介绍了 Font Squirrel 的方法，接着介绍了本章演示的方法。

我在前面提到过，我倾向于使用第二种方法。不过这种方法也有一定的局限性：如果对一种 Web 字体（并非页面上所有的 Web 字体）使用四种样式和粗细版本，Internet Explorer 6 ~ 8 会将特殊的样式和粗细版本直接以常规样式显示。

实践中，这对绝大多数情形都没有影响。首先，我们通常不会对同一种字体引入四种不同的样式和粗细版本。要知道，使用那么多样式和粗细版本，就需要浏览器下载比正常情况更多或更大的文件。

更重要的是，在这三个 IE 版本中，只有 IE8 在世界上大多数地区拥有一定的用户基数，而且其用户份额还在不断下降。例如，截至本书写作之际，IE8 在欧洲的份额为 6.9%，在北美为 9.5%。一年前，它的份额还有 40% 甚至更高。

13.6 为 Web 字体应用斜体和粗体

如果你想对 Web 字体添加一些基本样式，这时 Web 字体用起来有些奇怪。需要记住的是，对于 Web 字体，每个字体只有一种粗细和一种风格。如果你想使用粗体或斜体，就需要为它们创建单独的样式规则，每条规则对应一个 Web 字体文件。否则，浏览器就会使用伪粗体或伪斜体（如图 13.6.1 所示），或者同时使用伪粗体和伪斜体（后面将会看到）。

Local Teen Prefers Vinyl Over Digital

"It's really groovy,"

图 13.6.1 在前一节的例子中，浏览器让常规文本变得稍微粗一点来模拟粗体，让常规文本稍微倾斜来模拟斜体。然而，我们需要的是专门为该字体设计的加粗或斜体版本。关于伪斜体和伪粗体，请回顾 10.4 节和 10.5 节

在前一节中，我们没有在样式表中为任何文本指定加粗或斜体的样式（参见图 13.6.2）。那么这些样式是怎么产生的呢？我想你一定能猜出来，即浏览器的默认样式。浏览器会对 `h1` 应用粗体，对 `em` 应用斜体（如图 13.6.3 所示），就像你应用 Web 字体之前那样。

```
body {
  font-family: 'PTSans', sans-serif;
  font-size: 100%;
  line-height: 1.25;
}

@font-face {
  font-family: 'PTSans';
  src: url('PTS55F-webfont.eot');
  ...
  font-weight: normal;
  font-style: normal;
}
```

图 13.6.2 我们不会在 `example.css` 中的任何位置使用 `font-weight: bold;` 或者 `font-weight: italic;`;

```
...
<h1>Local Teen Prefers Vinyl Over Digital
→ </h1>

<p>..."It's <em>really</em> groovy," she said,
→ on the record..."</p>
...
```

图 13.6.3 我们的标题是 `h1`，我们希望强调的文本放在 `em` 中

幸好要修复这一问题相当简单。步骤与对常规文本实施 Web 字体是相似的，只是 CSS 部分有一些变动。跟之前一样，我会以 PT Sans 为例进行讲解，不过这些步骤也适用于从 Font Squirrel 获取到的其他字体。

1. 为 Web 字体应用斜体的步骤

(1) 看看 13.3 节中介绍的从 Font Squirrel 下载的 ZIP 文件中的 `web fonts` 文件夹。如果

你看到一个字体的斜体版本的子文件夹，就打开该子文件中的 `stylesheet.css`。

(2) 将为常规文本准备的 `@font-face` 样式规则复制到你自己的样式表中（如图 13.6.4 所示）。

(3) 将斜体字体文件（图 13.6.4 所示的样式表中引用的字体文件）从 Font Squirrel 文件夹复制到包含你的样式表的文件夹（如图 13.6.5 所示）。

(4) 对 `font-family` 值进行重命名，让该名称与针对常规文本的样式规则中的名称保持一致（如图 13.6.6 所示）。

(5) 将斜体文本 `@font-face` 样式规则中的 `font-style` 值改为 `font-style: italic`（如图 13.6.6 所示）。最后，一定要保存文件。

```
body {
  font-family: 'PTSans', sans-serif;
  ...
}

/* 常规文本 */
@font-face {
  font-family: 'PTSans';
  src: url('PTS55F-webfont.eot');
  ...
}

/* 斜体文本 */
@font-face {
  font-family: 'pt_sansitalic';
  src: url('PTS56F-webfont.eot');
  src: url('PTS56F-webfont.eot?#iefix')
    → format('embedded-opentype'),
    url('PTS56F-webfont.woff')
    → format('woff'),
    url('PTS56F-webfont.ttf')
    → format('truetype');
  font-weight: normal;
  font-style: normal;
}
```

图 13.6.4 现在 `example.css` 包含了 PT Sans Italic 的 `@font-face` 规则

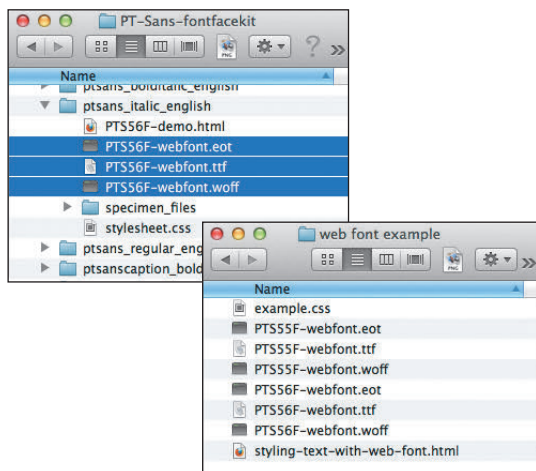


图 13.6.5 我将从 Font Squirrel 下载的文件夹中的三个 PT Sans Italic 字体格式文件（上）复制到我的示例文件夹（下）

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS55F-webfont.eot');
    ...
    font-weight: normal;
    font-style: normal;
}

/* 斜体文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS56F-webfont.eot');
    ...
    font-weight: normal;
    font-style: italic;
}
```

图 13.6.6 这里对代码做了两个重要的改动。将 `font-family: 'pt_sansitalic'` 改成了 `font-family: 'PTSans'`，从而与为常规文本 `@font-face` 样式规则中取的名称保持一致。然后在斜体文本 `@font-size` 样式规则中对 `font-style` 的值作了修改，以反映斜体样式

现在，只要 `font-style: italic;` 存在，无论是来自浏览器的默认样式（如图 13.6.7 所示）还是你自己的样式（如图 13.6.8 所示），都会显示斜体的 Web 字体。

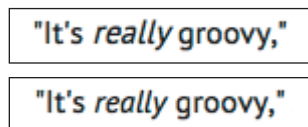


图 13.6.7 之前的伪斜体（上）与正确的版本（下）之间的差别在字体 a 和 e 上表现得最为明显。真正的斜体比伪斜体要小一些，与常规文本放在一起会显得更为协调

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

p {
    font-style: italic;
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS55F-webfont.eot');
    ...
    font-weight: normal;
    font-style: normal;
}

/* 斜体文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS56F-webfont.eot');
    ...
    font-weight: normal;
    font-style: italic;
}
```

图 13.6.8 出于演示需要，我将所有段落文本都以斜体显示。这一样式在接下来的示例中将被移除

2. 为 Web 字体应用粗体的步骤

(1) 重复“为 Web 字体应用斜体的步骤”中的第 (1) 步 ~ 第 (3) 步，但是要使用字体

的粗体版本对应的文件夹中的文件。将粗体文本的 `@font-face` 样式规则复制到样式表中（对应应用斜体样式步骤中的第 (2) 步，如图 13.6.9 所示），并将粗体字体文件复制到你的文件夹中（对应应用斜体样式步骤中的第 (3) 步，如图 13.6.10 所示）。

```
...

... 常规文本和斜体文本的@font-face规则 ...

/* 粗体文本 */
@font-face {
    font-family: 'pt_sansbold';
    src: url('PTS75F-webfont.eot');
    src: url('PTS75F-webfont.eot?#iefix')
        → format('embedded-opentype'),
        url('PTS75F-webfont.woff')
        → format('woff'),
        url('PTS75F-webfont.ttf')
        → format('truetype');
    font-weight: normal;
    font-style: normal;
}
```

图 13.6.9 现在 example.css 包含了 PT Sans Bold 的 `@font-face` 规则

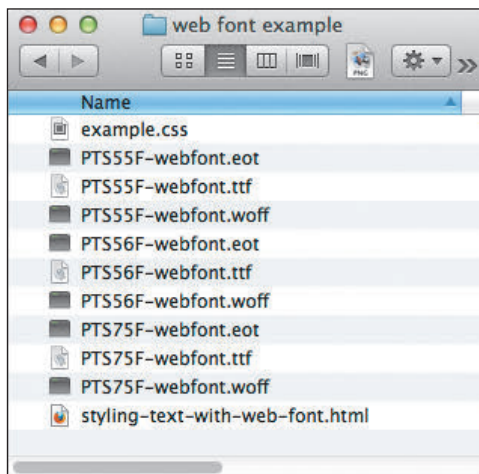


图 13.6.10 我将三个 PT Sans 粗体文件从 Font Squirrel 文件夹（未显示在图中）复制到示例文件夹。文件名均以 PTS75F 开头

(2) 对粗体文本 `@font-face` 样式规则中的 `font-family` 值进行重命名，使之与针对常规文本的样式规则中的名称保持一致（如图 13.6.11 所示）。

(3) 将粗体文本 `@font-face` 样式规则中的 `font-weight` 值改为 `font-weight: bold`（如图 13.6.11 所示）。最后，一定要保存文件。

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: normal;
    font-style: normal;
}

/* 斜体文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: normal;
    font-style: italic;
}

/* 粗体文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS75F-webfont.eot');
    ...
    font-weight: bold;
    font-style: normal;
}
```

图 13.6.11 我将 `font-family: 'pt_sansbold'` 改成了 `font-family: 'PTSans'`，从而与为常规文本 `@font-face` 样式规则中取的名称保持一致。然后在粗体文本 `@font-size` 样式规则中对 `font-weight` 的值作了修改，以反映粗体样式

现在，只要对文字设置加粗样式，无论是来自浏览器的默认样式（如图 13.6.12 中的 `h1`）还是你自己的样式，都会显示粗体的 Web 字体。例如，在图 13.6.11 中加入 `p`

{ font-weight: bold; }, 就会让所有的段落文本变成加粗样式（需要注意的是，这也影响了 em 元素）。



图 13.6.12 如果对伪粗体(上)和正确的版本(下)进行比较,会发现正确的版本中字母更清晰,字间距更合理。现在,我们的标题为正确的粗体,em 文本为正确的斜体,其他文本均为常规样式

3. 为 Web 字体应用粗斜体的步骤

前面,我们已经看到如何为文本应用常规、加粗和斜体的 Web 字体。这也为我们留下了一个疑问:如何让使用 Web 字体的文本既有加粗样式也有斜体样式。

记住,每个 Web 字体都只对应一种样式、一种粗细版本。在我们引入的三种 PT Sans 字体中(参见图 13.6.11),没有一种是为 font-weight: bold 和 font-style: italic 同时存在而准备的。要么是其中的一种,要么都没有。这就是图 13.6.13 中的代码不会照你设想的方式运行的原因。(这解释了设置 p { font-weight: bold; } 后 em 会使用伪斜体的原因,因为 em 继承了 p 的加粗样式。)

下面来为页面添加粗斜体样式。

重复“为 Web 字体应用斜体的步骤”中的第(1)步~第(3)步,但是要使用字体的粗斜体版本对应的文件夹中的文件。将粗斜体文本的 @font-face 样式规则复制到样式表中

(对应应用斜体样式步骤中的第(2)步,如图 13.6.14 所示),并将粗斜体字体文件复制到你的文件夹中(对应应用斜体样式步骤中的第(3)步,如图 13.6.15 所示)。

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

em {
    font-weight: bold;
}

... @font-face rules ...
```

"It's really groovy,"

图 13.6.13 我在图 13.6.11 中的样式表中添加了一条规则,尝试让 em 文本既是斜体(默认样式)又是粗体。但这里有个问题:我们没有加载 PT Sans 粗斜体字体,因此 em 有伪斜体样式(不过这有些难以观察)

```
...

em {
    font-weight: bold;
}

... 其他@font-face规则 ...

/* 粗斜体文本 */
@font-face {
    font-family: 'pt_sansbold_italic';
    src: url('PTS76F-webfont.eot');
    src: url('PTS76F-webfont.eot?#iefix')
        → format('embedded-opentype'),
        url('PTS76F-webfont.woff')
        → format('woff'),
        url('PTS76F-webfont.ttf')
        → format('truetype');
    font-weight: normal;
    font-style: normal;
}
```

图 13.6.14 使用图 13.6.13 的代码,现在 example.css 包含了为 PT Sans 粗斜体准备的 @font-face 样式规则

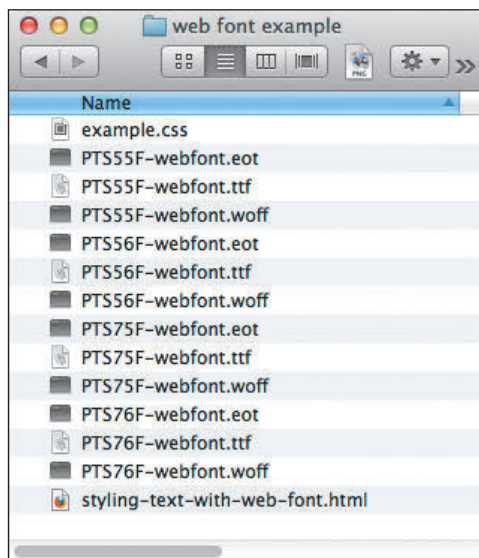


图 13.6.15 我将三个 PT Sans 粗斜体文件从 Font Squirrel 文件夹 (未显示在图中) 复制到示例文件夹。文件名均以 PTS76F 开头

对粗斜体文本 `@font-face` 样式规则中的 `font-family` 值进行重命名, 使之与针对常规文本的样式规则中的名称保持一致 (如图 13.6.16 所示)。

将粗斜体文本 `@font-face` 样式规则中的 `font-weight` 值改为 `font-weight: bold`, 同时将 `font-style` 值改为 `font-style: italic` (如图 13.6.16 所示)。最后, 一定要保存文件 (如图 13.6.17 所示)。

提示 如果某种 Web 字体没有粗体、斜体或粗斜体版本, 而我们又对文本添加了这些样式, 浏览器就会显示伪样式。

提示 有些字体 (如 ChunkFive) 本身就很粗, 没有粗体版本。应该对这类字体使用 `font-weight: normal`; (即其常规状态), 因为 `font-weight: bold`; 会让浏览器将其显示伪粗体, 这样它们的笔划就更粗了。

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

em {
    font-weight: bold;
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: normal;
    font-style: normal;
}

/* 斜体文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: normal;
    font-style: italic;
}

/* 粗体文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: bold;
    font-style: normal;
}

/* 粗斜体文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS76F-webfont.eot');
    ...
    font-weight: bold;
    font-style: italic;
}
```

图 13.6.16 我将 `font-family: 'pt_sansbold_italic'` 改成了 `font-family: 'PTSans'`, 从而与为常规文本 `@font-face` 样式规则中取的名称保持一致。然后对 `font-weight` 和 `font-style` 的值都作了修改, 以反映粗斜体样式

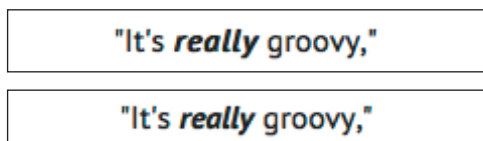


图 13.6.17 变化很微妙，伪效果（上）与正确的字体（下）相比，字母更粗一些，字间距也更大一些

提示 记住，每一个样式和粗细版本都会引入一个新的字体文件，从而增加浏览器要下载的文件大小，这会影响性能。出于这个原因，很多设计师仅对标题使用 Web 字体。

提示 由于引入了大量的文件，我们的文件夹看起来有些乱（如图 13.6.15 所示）。现在你应该能看出将字体文件与样式表放在不同文件夹的原因了！图 13.6.18 显示了一种组织这些文件的方式。你可能需要根据这种结构调整样式表中的每个 url。例如，url('font/PTS55F-webfont.woff')。同时，你还需要修改 HTML 中引入样式表的 link 元素以反映正确的位置：<link rel="stylesheet" href="css/example.css" />。

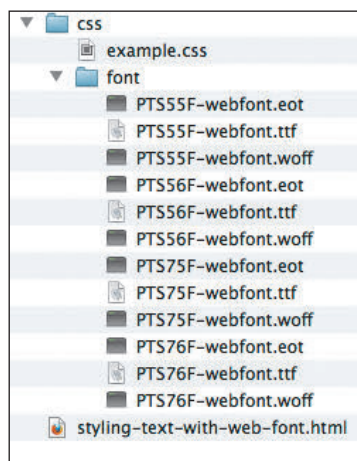


图 13.6.18 在这种结构中，字体文件都位于 font 文件夹，该文件夹又位于 css 文件夹中

13.7 使用 Google Fonts 的 Web 字体

前面的小节介绍了如何将自托管的 Web 字体添加到页面中去，例如使用 Font Squirrel 提供的字体。在众多托管 Web 字体的服务中，我们来看看 Google Fonts（参见 13.1 节）。

同其他字体服务一样，Google Fonts 也为我们省下了创建和修改 @font-face 规则的麻烦。这是自托管字体和使用 Google Fonts 这类服务之间的一个明显的区别（也可以说是 Google Fonts 这类服务的一个优势）。

不像其他的服务，Google Fonts 是免费的，它有几百种字体可供选择，而且使用它在页面中添加 Web 字体不到一分钟就能搞定，因此它已经变得越来越流行。一旦选中一种字体，省下我们要做的全部事情就是将一段 Google Fonts 代码添加到 HTML 页面中，然后就可以在 CSS 中为文本应用字体样式了。

使用 Google Fonts 的 Web 字体的步骤

(1) 访问 www.google.com/fonts。找到你喜欢的字体后，点击 Add to Collection（添加到集合），如图 13.7.1 所示。

(2) 将页面滚动到底部，点击 Use（使用），如图 13.7.2 所示。

(3) 仅选择将要使用的字体样式和粗细版本。这样做能让字体文件尽可能小（如图 13.7.3 所示）。

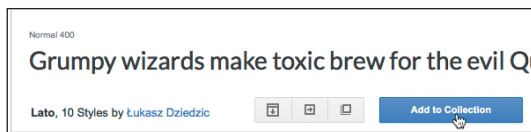


图 13.7.1 这里将 Lato 添加到了我的 Web 字体集合。此外还添加了 Open Sans（不过这里没有显示）。这是我为第 11 章和第 12 章中的大部分示例页面中的标题使用的字体