第 1 章

规则 1——减少 HTTP 请求

Rule 1: Make Fewer HTTP Requests

绪言 A 中介绍的**性能黄金法则**（*Performance Golden Rule*）揭示了只有 10%~20%的最终用户响应时间花在接收所请求的 HTML 文档上。剩下的 80%~90%时间花在为 HTML 文档所引用的所有组件（图片、脚本、样式表、Flash 等）进行的 HTTP 请求上。因此，改善响应时间的最简单途径就是减少组件的数量，并由此减少 HTTP 请求的数量。

从页面中移除组件的想法会引发性能和产品设计之间的矛盾。在这一章中，我将介绍的技术既可以减少 HTTP 请求，又能避免在性能和设计之间进行艰难的抉择。这些技术包括图片地图、CSS Sprites、内联图片和脚本、样式表的合并。运用这些技术在示例页面上估计响应时间减少到 50%左右。

图片地图

Image Maps

在一个最简单的转简中，超链接带有一些文本，并被关联到目标 URL 上。一种更为美观的选择是将超链接关联到图片上，例如在导航栏或按钮中。如果是以这种形式关联多个带有超链接的图片，使用图片地图这种方式就既能减少 HTTP 请求，又无需改变页面外观感受。**图片地图**（*Image Map*）允许你在一个图片上关联多个 URL。目标 URL 的选择取决于用户单击了图片上的哪个位置。

图 1-1 给出了一个示例，在一个导航栏上有五幅图片。单击一个图片会将你带到与之相关的链接。这可以通过五个分开的超链接、使用五个分开的图片来实现。然而，如果使用一个图片地图则可以更有效率，因为五个 HTTP 请求被减少为只有一个 HTTP 请求。响应时间将会降低，因为减少了 HTTP 开销。

你可以通过访问下面的 URL 来自己试验一下。单击每个链接来看一下获取时间。

图 1-1：图片地图示例

无图片地图的示例
http://stevesouders.com/hpws/imagemap-no.php

图片地图的示例
http://stevesouders.com/hpws/imagemap.php

当在 DSL（900Kbps）上使用 Internet Explorer 6.0 时，获取图片地图的时间比获取为每个超链接使用分离图片的导航条的时间**快** 56%（354ms : 799ms）。这是因为图片地图减少了四个 HTTP 请求。

图片地图有两种类型。**服务器端图片地图**（*Server-side image maps*）将所有点击提交到同一个目标 URL，向其传递用户单击的 x、y 坐标。Web 应用程序将该 x、y 坐标映射为适当的操作。**客户端图片地图**（*Client-side image maps*）更加典型，因为它可以将用户的点击映射到一个操作，而无需向后端应用程序发送请求。映射通过 HTML 的 `MAP` 标签实现。下面的 HTML 将图 1-1 中的导航栏转换成了图片地图，并展示了如何使用 `MAP` 标签。

```
<img usemap="#map1" border=0 src="/images/imagemap.gif?t=1196816255">
<map name="map1">
  <area shape="rect" coords="0,0,31,31" href="home.html" title="Home">
  <area shape="rect" coords="36,0,66,31" href="gifts.html" title="Gifts">
  <area shape="rect" coords="71,0,101,31" href="cart" title="Cart">
  <area shape="rect" coords="106,0,136,31" href="settings.html" title="Settings">
  <area shape="rect" coords="141,0,171,31" href="help.html" title="Help">
</map>
```

使用图片地图也有缺点。在定义图片地图上的区域坐标时，如果采取手工的方式则很难完成且容易出错，而且除了矩形之外几乎无法定义其他形状。通过 DHTML 创建的图片地图则在 Internet Explorer 中无法工作。

如果你正在导航栏或其他超链接中使用多个图片，将它们转换为图片地图是加速页面的最简单的方式。

CSS Sprites

和图片地图一样，*CSS Sprites* 也可以合并图片，但更为灵活。这个概念就像是使用“显灵板（Ouija Board）”一样，占卜写板（Planchette，由所有参与者一起托着的一个观察用具）不停地移动，停留在不同的字母上。要使用 CSS Sprites，需要将多个图片合并到一个单独的图片中，就像图 1-2 所展示的那样。这就是“显灵板”。

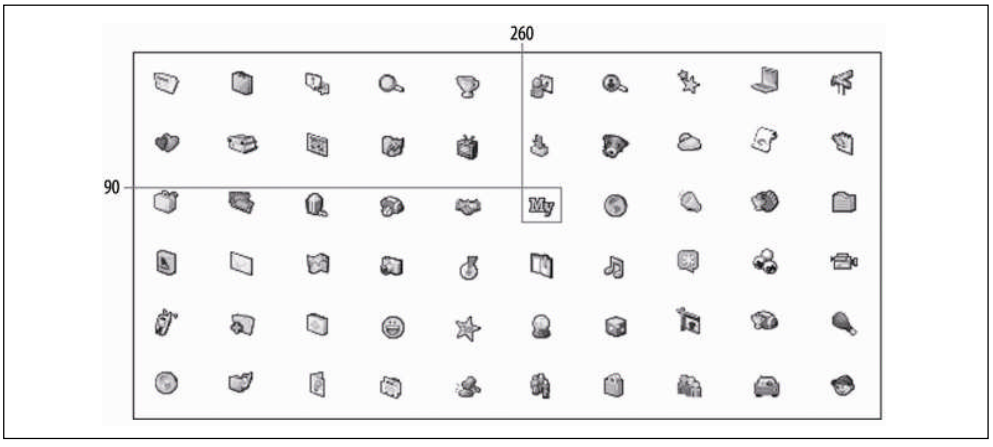

图 1-2：CSS Sprites 将多幅图片合并为一幅单独的图片

“占卜写板”是任何支持背景图片的HTML元素，如SPAN或DIV。使用CSS的background-position属性，可以将HTML元素放置到背景图片中期望的位置上。例如，像下面这样可以将“My”图标用作一个元素的背景图片：

```
<div style="background-image: url('a_lot_of_sprites.gif');
            background-position: -260px -90px;
            width: 26px; height: 24px;">
</div>
```

我使用CSS Sprites修改了前面的图片地图示例。一个名为navbar的DIV包含了五个链接。每个链接都被包围在一个SPAN中，它们使用同一个背景图片——spritebg.gif——定义在#navbar span规则中。每个SPAN都具有一个不同的类，通过background-position属性指定了CSS Sprites的偏移量：

```
<style>
#navbar span {
  width:31px;
  height:31px;
  display:inline;
  float:left;
  background-image:url(/images/spritebg.gif);
}
.home     { background-position:0 0; margin-right:4px; margin-left: 4px;}
.gifts    { background-position:-32px 0; margin-right:4px;}
.cart     { background-position:-64px 0; margin-right:4px;}
.settings { background-position:-96px 0; margin-right:4px;}
.help     { background-position:-128px 0; margin-right:0px;}
</style>

<div id="navbar" style="background-color: #F4F5EB; border: 2px ridge #333; width:
180px; height: 32px; padding: 4px 0 4px 0;">
```

```
    <a href="javascript:alert('Home')" title="Home"><span class="home"></span></a>
    <a href="javascript:alert('Gifts')" title="Gifts"><span class="gifts"></span></a>
    <a href="javascript:alert('Cart')" title="Cart"><span class="cart"></span></a>
    <a href="javascript:alert('Settings')" title="Settings"><span class="settings"></span></a>
    <a href="javascript:alert('Help')" title="Help"><span class="help"></span></a>
</div>
```

它和图片地图示例几乎一样快——两者分别是 342 毫秒和 354 毫秒，其间的差别微乎其微。重要的是，它比使用分离的图片**快** 57%。

CSS Sprites 示例

http://stevesouders.com/examples/sprites.php

但是，图片地图中的图片必须是连续的，而 CSS Sprites 则没有这个限制。关于 CSS Sprites 的赞成意见（与一些反对意见），Dave Shea 在其权威作品《CSS Sprites: Image Slicing's Kiss of Death》中给出了很好的解释。我在前面简要地提到了它的优点——通过合并图片减少 HTTP 请求，并且比图片地图更灵活。一个令人惊奇的优点是，它还降低了下载量。很多人会认为合并后的图片会比分离的图片的总和要大，因为合并的图片中包含有附加的空白区域。实际上，合并后的图片会比分离的图片的总和要小，这是因为它降低了图片自身的开销（颜色表、格式信息，等等）。

如果需要在页面中为背景、按钮、导航栏、链接等提供大量图片，CSS Sprites 绝对是一种优秀的解决方案——干净的标签、很少的图片和很短的响应时间。

内联图片
Inline Images

通过使用 `data:` URL 模式可以在 Web 页面中包含图片但无需任何额外的 HTTP 请求。尽管 Internet Explorer 目前还不支持这种方式，但它能给其他浏览器带来的节省使得它值得关注。

我们都很熟悉包含 `http:`模式的 URL。其他类似的模式包括 `ftp:`、`file:`和 `mailto:`。但除此之外还有很多模式，如 `smtp:`、`pop:`、`dns:`、`whois:`、`finger:`、`daytime:`、`news:` 和 `urn:`。这其中有一些是官方注册的，还有一些由于广泛使用而被接受。

`data:` URL 模式在 1995 年被首次提议。规范（*http://tools.ietf.org/html/rfc2397*）对它的描述为：“允许将小块数据内联为‘立即（immediate）’数”。数据就在其 URL 自身之中，其格式如下：

```
data:[<mediatype>][;base64],<data>
```

一个红色五角星形状的内联图片可以定义为：

```
<IMG ALT="Red Star"
SRC="
lvrKy/FvcPewsO9VVfajo+w6O/zl5estLv/8/AAAAAAAAAAAAAAACH5BAEA
AAsALAAAAAAMAAwAAAQzcElZyryTEHyTUgknHd9xGV+qKsYirKkwDYiKDBia
tt2H1KBLQRFIJAIKywRgmhwAIlEEADs=">
```

我见过的 `data:` 都是用于内联图片的，但它可以用在任何需要指定 URL 的地方，包括 `SCRIPT` 和 `A` 标签。

`data:` URL 模式的主要缺陷在于不受 IE 的支持（直到包括版本 7 都是如此）。另一个缺陷是可能存在数据大小上的限制，但 Firefox 1.5 可以支持高达 100KB 的内联图片。Base64 编码会增加图片的大小，因此整体下载量会增加。

下面的示例将使用内联图片来实现前面的导航栏。

内联图片的示例

http://stevesouders.com/examples/inline-images.php

由于 `data:` URL 是内联在页面中的，在跨越不同页面时不会被缓存。不要去内联公司 Logo，因为编码过的 Logo 会导致页面变大。这种情况下，聪明的做法是使用 CSS 并将内联图片作为背景。将该 CSS 规则放在外部样式表中，这意味着数据可以缓存在样式表内部。在下面的例子中，导航栏中的每个链接所使用的背景图片都被实现为外部样式表中的内联图片。

内联 CSS 图片的示例

http://stevesouders.com/examples/inline-css-images.php

在外部样式表中，每个 `SPAN` 都有一个规则，其中包含了内联的背景图片：

```
.home { background-image: url(...);}
.gift { background-image: url(...);}
.cart { background-image: url(...);}
.settings { background-image: url(...);}
.help { background-image: url(...);}
```

PHP 函数 `file_get_contents` 可以很容易地通过从磁盘中读取图片并将其内容插入到页面中来创建内联图片。在这个示例中，外部样式表的 URL 指向一个 PHP 模板——*http://stevesouders.com/hpws/inline-css-images-css.php*。这个 PHP 模板展示了 `file_get_contents` 的使用，它生成了前面给出的样式表：

```
.home { background-image: url(
  <?php echo base64_encode(file_get_contents("../images/home.gif")) ?>);}
.gift { background-image: url(
  <?php echo base64_encode(file_get_contents("../images/gift.gif")) ?>);}
```

```
.cart { background-image: url(
  <?php echo base64_encode(file_get_contents("../images/cart.gif")) ?>);}
.settings { background-image: url(
  <?php echo base64_encode(file_get_contents("../images/settings.gif")) ?>);}
.help { background-image: url(
  <?php echo base64_encode(file_get_contents("../images/help.gif")) ?>);}
```

将这个例子与之前的示例进行比较，可以看到它和图片地图及 CSS Sprites 的响应时间几乎一样，也是比原来为每个链接使用单独的图片的方式快 50%以上。将内联图片放置在外部样式表中增加了一个额外的 HTTP 请求，但被缓存后可以得到额外的收获。

合并脚本和样式表

Combined Scripts and Stylesheets

今天的很多网站都使用了 JavaScript 和 CSS。前端工程师必须选择是对 JavaScript 和 CSS 进行“内联”（也就是将其嵌在 HTML 文档中）还是将其放在外部的脚本和样式表文件中。一般来说，使用外部脚本和样式表对性能更有利（这将在第 8 章中详细讨论）。然而，如果遵循软件工程师所推荐的方式和模块化的原则将代码分开放到多个小文件中，会降低性能，因为每个文件都会导致一个额外的 HTTP 请求。

表 1-1 表明，前十位的网站在其首页上平均使用六到七个脚本和一到二个样式表。绪言 A 中曾介绍过，这些网站选自 *http://www.alexa.com*。如果它们没有被缓存到用户的浏览器中，则每个文件都需要一个额外的 HTTP 请求。和图片地图及 CSS Sprites 的优点一样，将这些单独的文件合并到一个文件中，可以减少 HTTP 请求的数量并缩短最终用户响应时间。

表 1-1：十大网站的脚本和样式表的数量

网站	脚本	样式表
http://www.amazon.com	3	1
http://www.aol.com	18	1
http://www.cnn.com	11	2
http://www.ebay.com	7	2
http://froogle.google.com	1	1
http://www.msn.com	9	1
http://www.myspace.com	2	2
http://www.wikipedia.org	3	1
http://www.yahoo.com	4	1
http://www.youtube.com	7	3

为了清晰，我不建议将脚本和样式表合并在一起。但是多个脚本应该合并为一个脚本，多个样式表也应该合并为一个样式表。理想情况下，一个页面应该使用不多于一个的脚本和样式表。

下面的例子展示了合并脚本是如何缩短最终用户响应时间的。使用了合并脚本的页面在加载时**快**了 38%。合并样式表可以带来类似的性能改进。在这一部分的剩余内容中，我将只介绍脚本（因为其使用量很大），但所有这些讨论也都同样适用于样式表。

分离脚本的示例

http://stevesouders.com/examples/combo-none.php

合并脚本的示例

http://stevesouders.com/examples/combo.php

对于那些接受过编写模块化代码（不论是 JavaScript 还是其他编程语言）的开发者来说，将所有东西合并到一个单独的文件中看起来像是一种倒退，而且将所有的 JavaScript 合并为一个单独的文件在开发环境中很难完成。一个页面可能需要 `script1`、`script2` 和 `script3`，而另一个页面可能需要 `script1`、`script3`、`script4` 和 `script5`。解决的方法是遵守编译型语言的模式，保持 JavaScript 的模块化，而在生成过程中从一组特定的模块生成一个目标文件。

很容易想象包含合并脚本和样式表的生成过程——简单地将适当的文件连接为一个单独的文件。合并文件很容易。在这一步中还可以对文件进行精简（参见第 10 章）。难的是组合的数量的增长。如果有大量需要不同模块的页面，组合后的数量就会非常庞大。十个脚本就能产生上千种组合！更不要指望强制每个页面都使用每个模块，而不管它是否真的需要。以我的经验来看，拥有多个页面的网站肯定会有大量不同的模块组合。这值得花一些时间去分析一下你的页面，确保组合的数量是可管理的。

小结
Conclusion

这一章介绍了我们在 Yahoo!中使用的用于减少 Web 页面中 HTTP 请求数量，而又无需在页面设计上作出妥协的技术。后面将要介绍的规则也提到了有助于减少 HTTP 请求数量的方法，但它们主要关注的是后续的页面浏览。对于那些在最初呈现页面时并非必需的组件来说，第 8 章所介绍的**加载后下载**（*post-onload download*）技术有助于将这些 HTTP 请求推迟到页面加载完毕后进行。

本章的规则是在用户第一次访问你的网站时能更有效地减少 HTTP 请求的数量，这也是为什么我将它放在了第 1 章、为什么它是最重要的规则。遵守该规则可以同时改善首次浏览和后续浏览的网站响应时间。首次访问页面时的响应时间决定着用户是放弃你的网站还是不停地进行回访。

减少 HTTP 请求