

最大限度利用 [JavaScript](#) 和 [Ajax](#) 性能

用 *Firebug*、*Safari Web Inspector*、*YUI Profiler* 和 *YSlow* 测量性能

简介： 随着 [Ajax](#) 和 [JavaScript](#) 在现代 [web](#) 应用程序中作用越来越突出，将 [JavaScript](#) 代码和 [Ajax](#) 请求适当调优以便能发挥最佳性能也日益重要。在本文中，您将了解编写 [JavaScript](#) 代码和执行 [Ajax](#) 请求的最佳实践。了解如何测试已有应用程序的性能，以及如何确定代码中的一般瓶颈。最后，了解如何解决性能相关的问题，以及如何根据您的具体情况使用各种工具，以使您的应用程序的运行速度达到最快。

[标记本文！](#)

发布日期： 2011 年 8 月 08 日

级别： 中级

原创语言： [英文](#)

简介

在 [web](#) 早期，优化 [web](#) 页面的性能通常意味着避免了使用不必要的 [HTML](#) 标记，将 [JavaScript](#) 代码量控制到最小，并尽量减小所有图片文件大小，否则上网冲浪者会走开去泡杯咖啡来等待页面加载。

[web](#) 各个方面的改进也意味着我们现在面临新的性能考虑。尽管 [DSL](#) 和宽带为很多人提供了对 [Internet](#) 上更多内容的高速访问，我们对加载时间和响应的期望也在发展，希望在页面上执行操作后能立刻得到结果。[Asynchronous JavaScript and XML \(Ajax\)](#) 的出现让开发人员能在 [web](#) 应用程序中提供像桌面程序一样的体验，不再需要响应事件前加载整个页面。其优势非常明显，但这也让普通的 [web](#) 用户希望在所有 [web](#) 应用程序中获得这样的响应。随着近期移动 [web](#) 的兴起，出现了新的挑战，满足现代 [web](#) 用户的期望，在有着更小屏幕、更低电量、更慢网速的目标设备上实现这些效果。

本文重点是告诉您在最大限度利用 [JavaScript](#) 和 [Ajax web](#) 应用程序时应考虑的问题。本文提供了以最佳方式处理代码的指导原则，无论是新应用程序还是已有程序中的。您还将了解各种工具和技术，用来测量应用程序的性能。最后，您将了解一些无需更改现有代码就可提升性能的方法。

[回页首](#)

[JavaScript](#) 和 [Ajax](#) 开发的最佳实践

[JavaScript](#) 开发的问题之一就是大部分编写 [JavaScript](#) 的开发人员和 [web](#) 设计师并未从基础学习 [JavaScript](#)。关于此语言的知识是长期通过添加执行某一特定功能的代码片段积攒起来的。他们通常知道如何声明变量、编写条件语句、执行运算，但他们从未静下心来全面系统地从头学起。现在，这些问题依然存在，而开发人员也急于使用库和框架（如 [jQuery](#) 和 [YUI](#)）来使开发变得更为容易。

使用 [JavaScript](#) 库其实也没有错（实际上，我也是其忠实粉丝）。在现代的开发人员中有这样一种趋势，是成为所选择的 [JavaScript](#) 框架的专家，而不是 [JavaScript](#) 本身的专家。这样带来的问题是，您会发现这些开发人员使用的是效率低下的编程实践，有时候做一件事时，使用普通的原始的 [JavaScript](#) 都会比使用某个框架特性要快得多。

这一章中，您将会了解一些 [JavaScript](#) 和 [Ajax](#) 开发最佳实践，特别是容易被非 [JavaScript](#) 开发人员忽略的方面。

[使用外部 \[JavaScript\]\(#\) 文件](#)

最大化 JavaScript 应用程序的金科玉律是尽可能使用外部 JavaScript 文件，而不是直接将 JavaScript 代码包含在 HTML 文件中。这么做不仅仅意味着不用在多个文件之间复制 JavaScript 代码，而且 JavaScript 代码会被 web 浏览器缓存，不必在每个子页面加载时重复加载一遍。第一个页面加载时特别慢，因为外部文件需要发送额外的 HTTP 请求到服务器。尽管如此，大多数应用程序中，第一次加载损失的性能远比子页面加载节省下的性能小得多。

有个例外情况是大多数的访问者只访问一个页面。这里最好使用内联 JavaScript；或者还有一种情况，就是您希望第一个页面与子页面加载一样快，或者比子页面更快。在 *High Performance Web Sites*（见 [参考资料](#) 中的链接）一书中，Steve Sounders 提出 *Post-Onload Download* 的概念，即将第一页的 JavaScript 代码放在其 HTML 文件中，然后在子页面完全加载后动态加载所需的外部 JavaScript 文件。尽管如此，大多数情况下，简单使用外部 JavaScript 文件已经足够。

何时使用 JavaScript 框架和库

我完全支持使用 JavaScript 框架和库。它们不仅能解决很多跨浏览器兼容性问题，而且，如果使用合适，可大大提高 web 应用程序开发速度。有这样的说法，使用这些工具要特别注意，因为它们其中多数都非常庞大，可能降低应用程序性能。

您要问自己的第一件事是：我确实需要使用框架吗？我初次接触 JavaScript 框架是在几年前，那时我需要在我开发的 web 应用程序中使用 Ajax。我没有自己编写 XMLHttpRequest 函数，而是决定使用 Prototype 框架使开发更容易。程序只用到框架的执行 Ajax 请求和处理服务器响应的功能，但我还是决定使用。幸运的是，我开发的应用程序相对较小，只在内部使用，性能不是很重要，但几年之后，我逐渐知道最好还是使用只提供 Ajax 功能的更轻量级的解决方案。

Prototype 框架最新的未精简未压缩版本，是 141KB。而与我的应用程序相关的代码部分不到 2KB，剩下 139KB JavaScript 代码对我的应用程序毫无用处。这不仅增加由文件大小造成的加载时间，还增加了在浏览器中执行 JavaScript 代码的执行时间。

概括来说，现代 JavaScript 框架和库，如 Prototype、jQuery、Dojo、MooTools、YUI、ExtJS 等等，包含一大堆您可能用到或用不到的特性。如果您只用到其中一小部分，您最好寻求更轻量级的解决方案。例如，YUI 库，可让您默认加载最小的空的框架，然后由您选择在此基础上加载哪些库。

脚本放置和加载

当您阅读 HTML 书籍时，它很可能建议您将 <script> 标记放在页面的 <head> 元素内部。如果这就是您目前对 <script> 标记放置的认识，赶紧把它忘了吧！将 <script> 标记放在 HTML 页面顶部将会使页面直到 JavaScript 代码完全加载和执行后才能呈现。如果将它们放在 <head> 标记内部，只有到脚本加载并执行后，页面主体才会呈现，从而让用户觉得页面加载很慢。

为了最优化页面的性能，应该将 JavaScript 代码放在页面底部，如果可能，就在 标记之前。这样，web 页面其他部分（HTML、[CSS](#)、图片、Flash 内容等等）将会在脚本加载和执行前下载，从而会让用户觉得加载要快一些。

如果您的 web 页面或应用程序需要很多 JavaScript，将所有代码放在一个单独文件中可能会造成下载和执行时长时间等待。这些情况下，最好将 JavaScript 代码分别放到多个文件中，当页面加载完成后在需要时动态加载。

LazyLoad JavaScript 库旨在提供动态脚本加载，并考虑到有关脚本执行顺序在跨浏览器时的不一致性。更多关于 LazyLoad 库的信息，见 [参考资料](#)。

最小化 Ajax 请求

Ajax 请求彻底改变了传统 web 应用程序的样子，它让 JavaScript 开发人员能创建高度动态化、交互性强、响应迅速的应用程序，就像在桌面应用程序中体验到的那样。结果，在现代的 web 应用程序中，Ajax 请求随处可见。有时候很容易忘记这点，尽管用户看不到页面加载，但 Ajax 请求执行的是完整的 HTTP 请求，它与常规页面加载一样。因此，应该多加关注，减少所使用的 Ajax 请求的数量。

这方面的一个例子是搜索结果分页。我经常看到在应用程序中，用一个 Ajax 请求以 JSON 数组形式返回搜索结果，再用一个请求返回数据库中结果条数，用于分页逻辑。[清单 1](#) 和 [清单 2](#) 显示的是这两个请求的基本样例（使用 Prototype 框架）。

清单 1. 第一个请求：获取表记录

```
var url = "get_data.php";
var options = {
    method: "post",
    parameters: {"page":1,"rows":5},
    onSuccess: firstCallbackFunction,
    onFailure: firstCallbackFunction
}
new Ajax.Request(url, options);
```

[清单 2](#) 显示的是第二个获取总记录数的请求。

清单 2. 第二个请求：获取总记录数

```
var url = "get_count.php";
var options = {
    method: "post",
    parameters: {},
    onSuccess: secondCallbackFunction,
    onFailure: secondCallbackFunction
}
new Ajax.Request(url, options);
```

[清单 3](#) 和 [清单 4](#) 显示的是对应的 JSON 格式的 HTTP 请求。

清单 3. 第一个响应：记录数组

```
{
    "records": [
        {"id":1,"name":"John","email":"john@example.com"},
        {"id":2,"name":"Mary","email":"mary@example.com"},
        {"id":3,"name":"Tony","email":"tony@example.com"},
        {"id":4,"name":"Emma","email":"emma@example.com"},
        {"id":5,"name":"Alan","email":"alan@example.com"}
    ]
}
```

```
    ]
}
```

[清单 4](#) 显示的是报告总记录数的响应。

清单 4. 第二个响应：总记录数

```
{"total_records": 95}
```

将这两个 Ajax 请求分开是浪费资源，它们可以合并到一个请求中，并生成以下 [清单 5](#) 的响应。

清单 5. 高效响应：记录总数和数组

```
{
  "total_records": 95,
  "records": [
    {"id":1,"name":"John","email":"john@example.com"},
    {"id":2,"name":"Mary","email":"mary@example.com"},
    {"id":3,"name":"Tony","email":"tony@example.com"},
    {"id":4,"name":"Emma","email":"emma@example.com"},
    {"id":5,"name":"Alan","email":"alan@example.com"}
  ]
}
```

这样做不仅使所需的 HTTP 请求和响应更少，而且也减少了用于响应 Ajax 请求的服务器端脚本。

本例演示非常简单 — 应用程序越复杂，减少所用的 Ajax 请求数量就越重要。

（适当）使用变量

为了尽量减少代码行，很多开发人员常忽略了变量的使用，很多情况下他们其实能显著提高某段代码的执行速度。例如，在以下代码中，对一个元素应用了多种样式：

清单 6. 对一个元素应用多种样式（低效）

```
document.getElementById("myField").style.backgroundColor = "#CCC";
document.getElementById("myField").style.color = "#FF0000";
document.getElementById("myField").style.fontWeight = "bold";
```

以上每行代码中，浏览器都要在 DOM 中搜索 ID 为 myField 的元素。其实无需像这样执行三次，可以通过将 document.getElementById("myField") 结果赋给一个变量并在每行使用变量来提高效率。如 [清单 7](#) 所示。

清单 7. 对一个元素应用样式（高效）

```
var myField = document.getElementById("myField");
myField.style.backgroundColor = "#CCC";
```

```
myField.style.color = "#FF0000";  
myField.style.fontWeight = "bold";
```

另一个应该用变量而没有用的地方是遍历数组的 `for` 循环。请看 [清单 8](#) 的例子。

清单 8. 用 `for` 循环遍历数组（低效）

```
for(var i=0; i < myArray.length; i++) {  
    //do something  
}
```

这段代码效率很低，因为每次循环迭代时都要计算 `myArray` 数组的长度。这对小数组无关痛痒，但对大数组影响巨大。提高循环性能的方法很简单，甚至不需多加一行代码（见 [清单 9](#)）。

清单 9. 用 `for` 循环遍历数组（高效）

```
for(var i=0, arrayLength=myArray.length; i < arrayLength; i++) {  
    //do something  
}
```

清单 9 中，我们把对 `myArray.length` 的引用移到了语句的初始化部分，将其赋给了一个变量。这样，它只在循环第一个迭代时执行，从而在以后每次迭代时节省了宝贵的时间。

使用 DOM

完全遍历和操作 DOM 会给 web 应用程序造成很大性能负担。问题是，与 DOM 的交互必不可少，能给应用程序带来快速响应和丰富接口。DOM 操作的一个特别问题是浏览器需要在屏幕上回流和重绘（主要是重新呈现）元素。因此，减少代码中回流和重绘的次数就尤为重要。

再看看前面 [清单 7](#) 中的例子，我们用一个 ID 为 `myField` 的元素，对其应用三个不同的样式属性。这将导致三次独立的回流（`re-flow`）和重绘（`re-paint`），效率很低，因此我们可以做些小更改。这些语句效率低下，可以考虑将这些操作合并成一句。[清单 10](#) 显示的是效率更高的样例。

清单 10. 通过合并修改内容来提高 DOM 操作的性能

```
var myField = document.getElementById("myField");  
myField.style.cssText = "background-color: #CCC; color: #FF0000; font-weight: bold";
```

通过将样式修改合并起来，结果是只有一次回流和重绘，而且，响应速度和性能也得到提升。

本章包含了可在应用程序中发挥最佳性能的 JavaScript 和 Ajax 开发最佳实践。遗漏在所难免。请查阅 [参考资料](#)，其中提供了重点介绍高性能 JavaScript 最佳实践的优秀文章、教程、书籍的链接。

测量性能

当谈到测量 web 应用程序的性能，有一系列的工具可用于在 web 浏览器内部分析应用程序的速度和加载时间。这些工具一般是预先设置一段 JavaScript 代码来计算执行时间。还提供了分析页面加载产生的网络流量的方法，其中有产生的 HPPT 请求数量、加载的静态资源（图像、外部样式表、JavaScript 文件）的大小，等等。

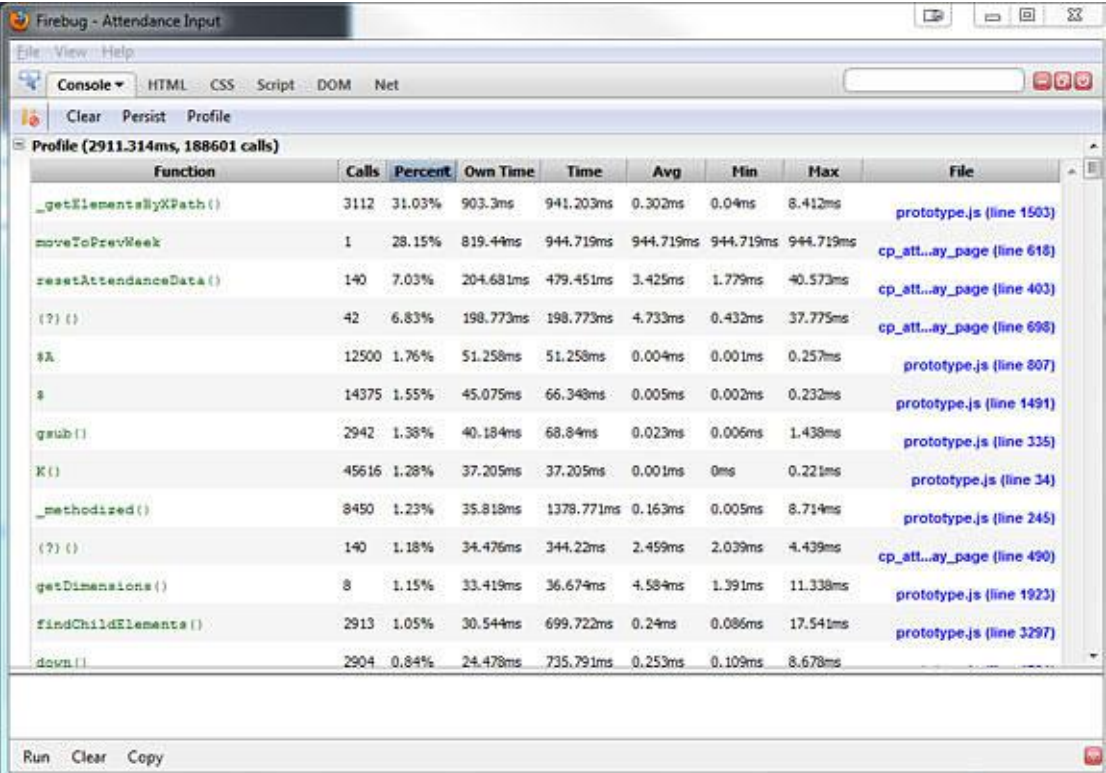
这一章中，您将了解各种工具，可根据具体情况预先设置并分析 JavaScript 应用程序中的网络活动。

Firebug

Firebug 是 Mozilla® Firefox® 的浏览器扩展，它为 web 开发人员提供了大量功能。这些功能包括 JavaScript 控制端、DOM 操作和选择、脚本调试、DOM 源浏览、分析、网络分析，等等。

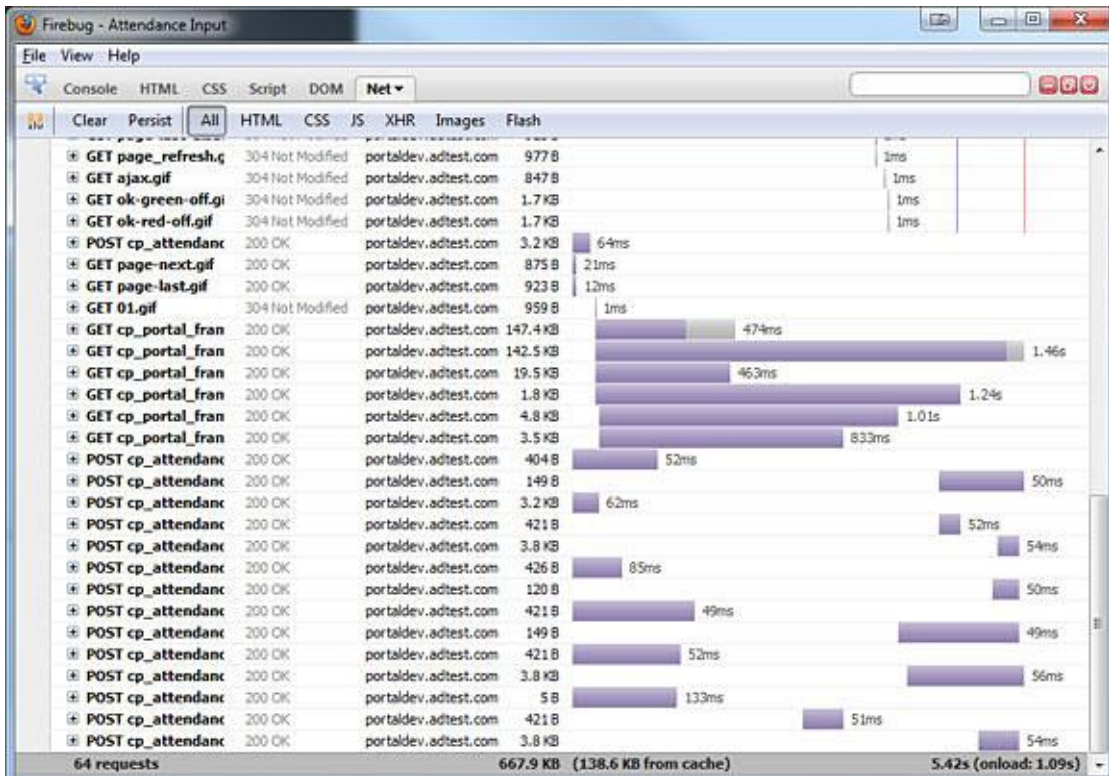
要执行 Firebug 中的 JavaScript 分析，导航到 **Console** 选项卡，单击 **Profile** 按钮。现在可以与页面中各种启用 JavaScript/Ajax 的特性交互，Firebug 将计时。Firebug 还提供控制端 API 与分析器交互。[图 1](#) 显示的是 Firebug 分析器生成的报告样例屏幕截图。

图 1. Firebug 分析器报告样例



要使用 Firebug 的网络分析工具，只要单击 **Net** 选项卡。将会显示所有生成的 HTTP 请求、响应代码和接收的消息、接收来源域、文件大小、发送时间点。还可以深入这些请求，查看发送的 HTTP 头部、接收到的响应、所用文件相关缓存信息。[图 2](#) 显示的是 Firebug 关于网络流量输出报告的样例。

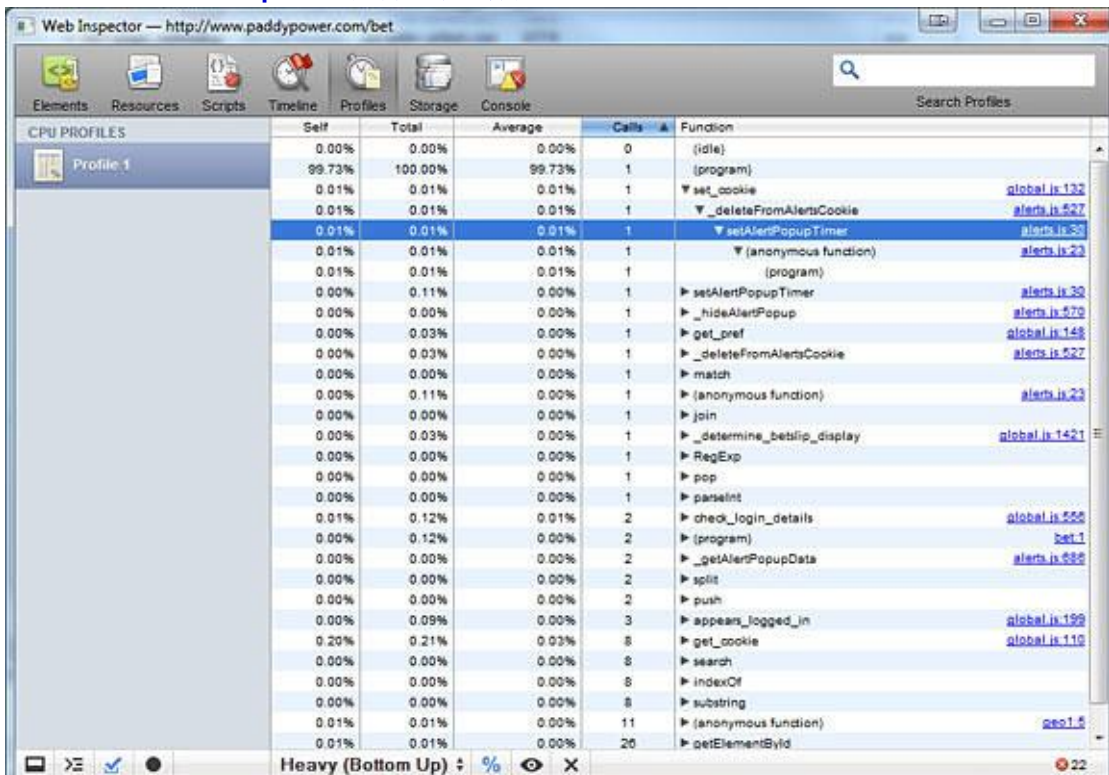
图 2. Firebug Net 面板报告样例



Safari Web Inspector 和 Chrome Developer Tools

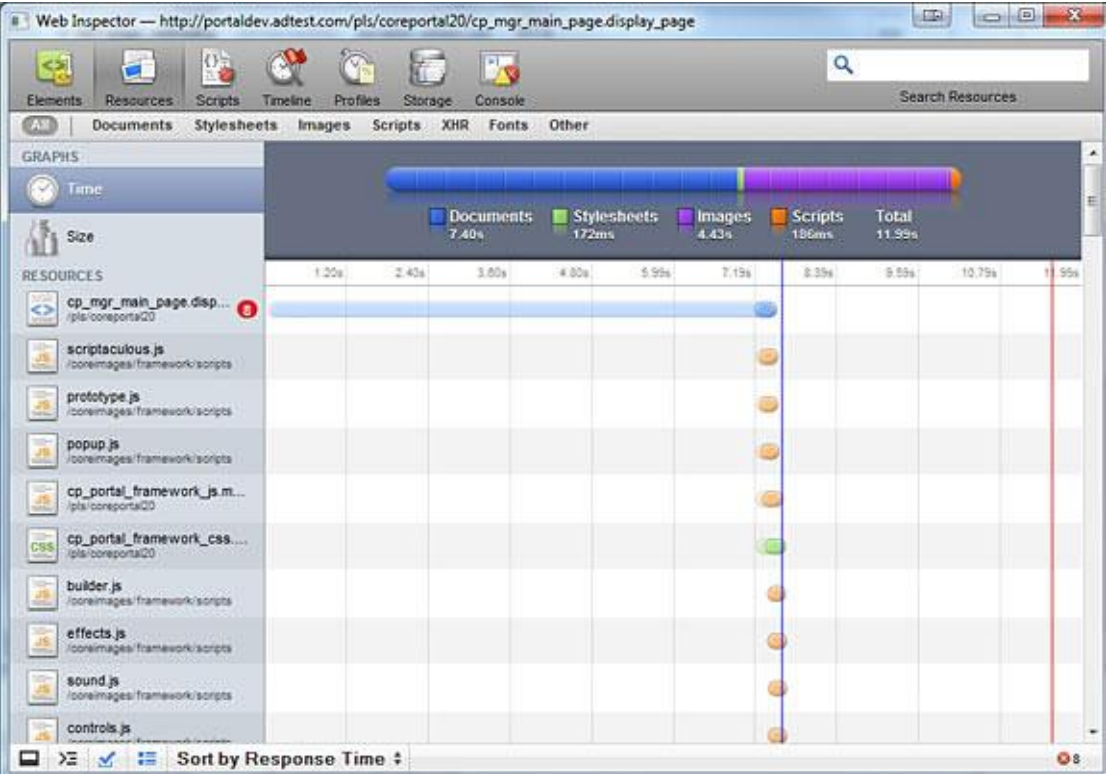
Safari® Web Inspector 和 Chrome® Developer Tools 提供了与 Firebug 相似的功能，他们看上去更漂亮。在 Safari 中，进入工具栏的 Develop 菜单（需要在 Safari 参数中启用）并选择 **Show Web Inspector**，打开 Web Inspector。单击检查器窗口顶部的 **Profile** 选项卡，点击底部左侧的圆形记录图标来启动分析。再次单击图标，以停止分析并生成报告。图 3 显示了报告样例。

图 3. Safari Web Inspector 分析器报告样例



要分析应用程序的加载时间和 HTTP 请求，单击 Inspector 窗口顶部的 **Resources** 按钮。然后您可以选择查看所使用的网络资源的时间和大小的图片。[图 4](#) 中显示的是漂亮的输出结果样例。

图 4. Safari Web Inspector 资源报告样例

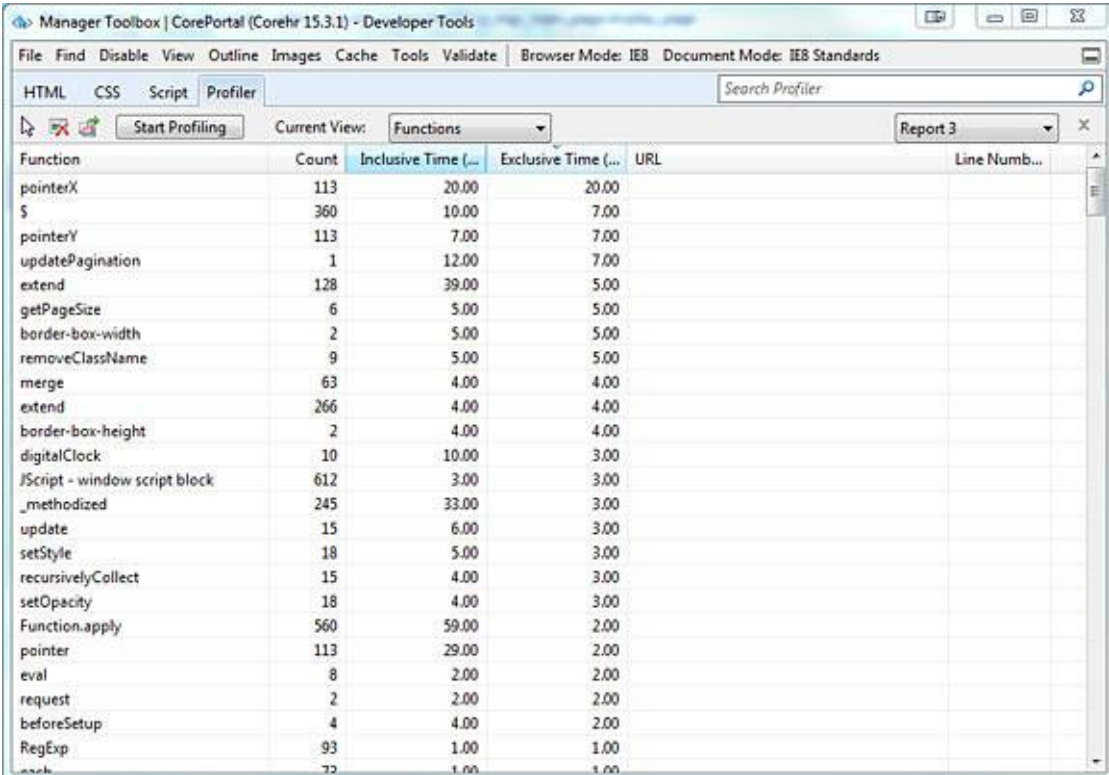


Google Chrome 的 Developer Tools 与 Safari 一样（都是基于浏览器的 WebKit），可以在在 Developer 菜单中看到 Developer Tools。

Internet Explorer Developer Tools

Internet Explorer® 8 也有一组 Developer Tools。单击 F12 启动 Developer Tools 窗口。Internet Explorer 的工具提供一组与 Firebug 相似的特性，而它提供了一个更强大的测试 JavaScript 函数性能的分析器。在 Developer Tools 中，单击 **Profiler** 选项卡并单击 **Start Profiling** 按钮开始分析应用程序。让应用程序执行想要测试的函数然后单击 **Stop Profiling** 生成报告。生成报告应与 [图 5](#) 类似。

图 5. Internet Explorer Developer Tools 分析器报告样例



The screenshot shows the 'Profiler' tab in the Internet Explorer Developer Tools. The 'Current View' is set to 'Functions'. A table lists various JavaScript functions along with their execution counts, inclusive and exclusive times, and the URLs where they were executed. The functions listed include 'pointerX', '\$', 'pointerY', 'updatePagination', 'extend', 'getPageSize', 'border-box-width', 'removeClassName', 'merge', 'extend', 'border-box-height', 'digitalClock', 'JScript - window script block', '_methodized', 'update', 'setStyle', 'recursivelyCollect', 'setOpacity', 'Function.apply', 'pointer', 'eval', 'request', 'beforeSetup', 'RegExp', and 'push'.

Function	Count	Inclusive Time (...)	Exclusive Time (...)	URL	Line Numb...
pointerX	113	20.00	20.00		
\$	360	10.00	7.00		
pointerY	113	7.00	7.00		
updatePagination	1	12.00	7.00		
extend	128	39.00	5.00		
getPageSize	6	5.00	5.00		
border-box-width	2	5.00	5.00		
removeClassName	9	5.00	5.00		
merge	63	4.00	4.00		
extend	266	4.00	4.00		
border-box-height	2	4.00	4.00		
digitalClock	10	10.00	3.00		
JScript - window script block	612	3.00	3.00		
_methodized	245	33.00	3.00		
update	15	6.00	3.00		
setStyle	18	5.00	3.00		
recursivelyCollect	15	4.00	3.00		
setOpacity	18	4.00	3.00		
Function.apply	560	59.00	2.00		
pointer	113	29.00	2.00		
eval	8	2.00	2.00		
request	2	2.00	2.00		
beforeSetup	4	4.00	2.00		
RegExp	93	1.00	1.00		
push	72	1.00	1.00		

不巧的是，Internet Explorer 的 Developer Tools 不包含网络分析器。如果您想要用 Internet Explorer 分析应用程序的网络流量，可以使用 Fiddler 工具。它可用于发出 HTTP 请求的所有应用程序，包括 Internet Explorer。想要了解更多关于 Fiddler 的信息，见 [参考资料](#)。

YUI Profiler

YUI Profiler 是用 JavaScript 编写的代码分析器。与包含在 web 浏览器中的分析器工具（或者是像 Firebug 一样的浏览器扩展）不同，YUI 分析器不可见，而且必须包含在 web 页面中。使用它的好处是可以明确定位到应用程序中需要分析的部分（而不只是执行的函数）。YUI Profiler 可以注册要测量的函数、类构造函数和对象。您可以使用分析器生成结果报告，它提供了一个过滤函数，可以精确定位到您希望测量的目标区域。配置文件报告是 JSON 格式的，可以轻松将其导入您自己的应用程序中，然后您可以生成表形和基于图表的数据视图。

YUI Profiler 一个主要优点是它不针对某个特定浏览器。它甚至可在有 A 级浏览器的移动设备上运行，例如 Apple® iPhone®、Android® 设备及 Nokia® N95 上的基于 WebKit 的浏览器。

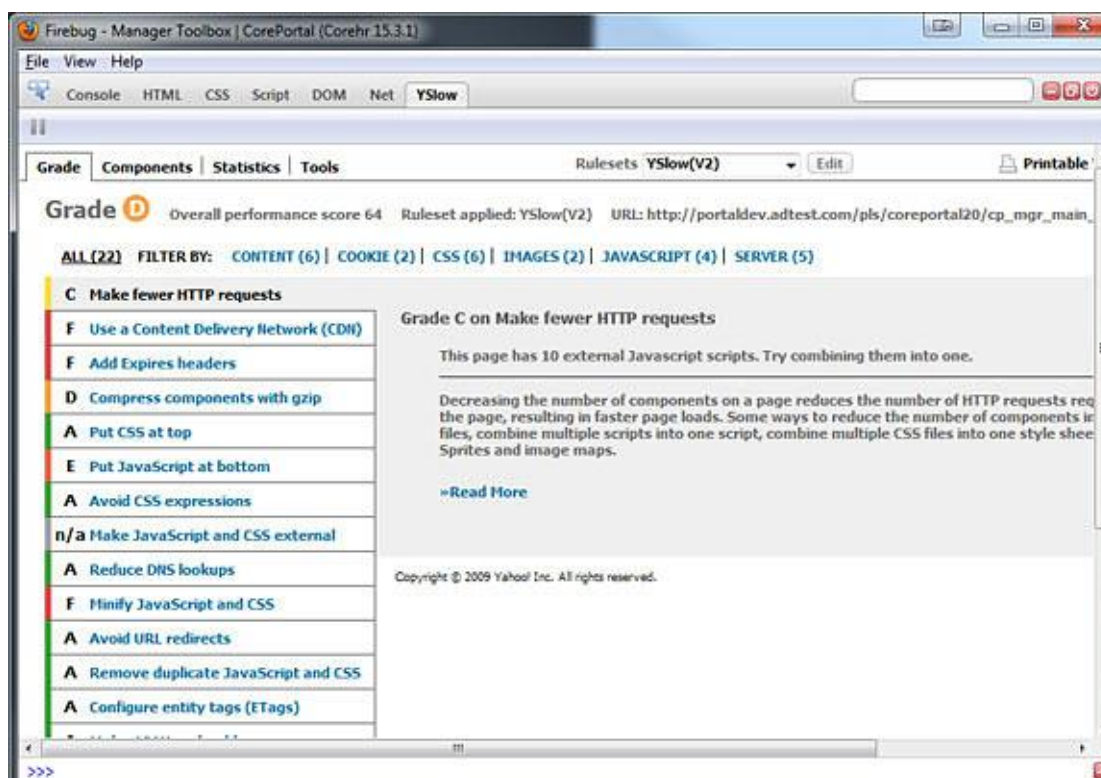
更多关于 YUI Profiler 的信息，见 [参考资料](#)。

YSlow

YSlow 是添加到 Firebug 扩展中的 Firefox 扩展，提供对 web 页面加载和执行的计分功能。它提供对 JavaScript 和 web 页面性能各方面的总体分级和详细分析。在我的测试中，我在发出 91 个 HTTP 请求的页面中运行 YSlow，评级为 C。它给出一个有用的提示，页面有 10 个外部 JavaScript 脚本，建议我将其合并成一个。

评级的其他部分还包括添加超期头部、使用 gzip、将 JavaScript 放在页面底部、减少 JavaScript 和 CSS、删除重复的 JavaScript 和 CSS 代码，等等。[图 6](#) 显示的是 YSlow 输出样例。

图 6. YSlow 报告样例



[回页首](#)

无需改变代码，提升性能

现在您已测量了应用程序的性能，并决定了让性能显著提升，您可能在考虑深入探究并修改程序代码。是的，某些方面需要这么做，不过还有一些方法可以在完全不改变应用程序代码的情况下提升 JavaScript 和 Ajax 性能。

合并 JavaScript 源文件

如果是一个大型的 JavaScript 应用程序，可能会向页面中加载多个外部 JavaScript 源文件，以提供各种功能。也许您正使用像 jQuery 这样的框架，并加入了提供各种额外特性的插件。很可能还有您自己的应用程序底层 JavaScript 代码，这将程序分割成多个文件。

尽管将 JavaScript 分成几个文件有其优势，但却对性能有严重影响。对于加载到 web 应用程序中的每个外部 JavaScript 文件，都要执行 HTTP 请求，这极大增加了脚本的加载和运行时间。为了减少请求数，强烈建议您尽可能合并 JavaScript 源文件。尤其是，如果一个插件只用于一个页面，就不要将它放在单独文件中。实际上，是否应该将其作为内联脚本，还有争论，这取决与它是否会被其他页面使用。也不能绝对地将所有东西都放入一个文件中，不过应用程序发出的 HTTP 请求数越少，应用程序加载越快。

缩减 JavaScript 源文件

好的编程实践指导原则讲到，编写应用程序代码时，应当遵循一致的风格，这对 JavaScript 和其他任何语言都适用。问题是，加入的所有空格、分行符、注释都增加了脚本文件大小和应用程序的加载时间。

为了降低 JavaScript 文件大小，强烈建议您在将文件放入产品环境前缩减 文件大小。缩减一般包括取出代码额外内容以获得紧凑和更小的文件。对潜在节约举例说明，单缩减本身就使最新版本从编写时的 155KB 减小到 70.6KB。节省了约 55%。

当然, 建议您缩减源代码是没什么问题, 但您会怎么做呢? 聪明的人是不会手工缩减代码的, 幸运的是, 有很多工具能帮您完成这项工作。其中一些流行的工具有 Douglas Crockford 的 JSMIN、Dean Edwards 的 Packer, 以及 Dojo 工具中的压缩器。我个人最喜欢的是 Yahoo!® Inc 的 YUI Compressor。关于这些工具的更多信息, 见 [参考资料](#)。

用 gzip 压缩 JavaScript 源文件

在前一章中, 您了解到可以通过缩减 JavaScript 源文件来显著减小其大小。实际上, 可以通过使用 gzip 压缩这些文件来进一步减小其大小。最好的方法是让服务器将 JavaScript 文件发送到客户端之前就用 gip 压缩。如果您正使用 Apache 2 (包含 mod_deflate) 并且 JavaScript 文件存储在同一文件夹下, 那就可以在 JavaScript 文件所在位置简单地创建一个 htaccess 文件。该文件的内容很简单: SetOutputFilter DEFLATE。

压缩 JavaScript 文件节约显著。在前一章中, 您了解到缩减 jQuery 源代码让文件大小从 155KB 陡降到 70.6KB (减小 55%)。如果对缩减的文件再用 gzip 压缩, 文件进一步减小到 24KB。通过缩减和压缩, 文件大小总共减少约 85%。结果根据您的代码而不同, 但节约是显著的。

缓存 JavaScript 源文件

让 web 浏览器正确缓存内容很重要。例如, 如果您正使用 Apache (包含 mod_expires), 并且想要客户端将 JavaScript 文件缓存两天, 您可以将 [清单 11](#) 中的指令添加到 .htaccess 文件中保存 JavaScript 文件的位置 (假设有一个文件专门保存 JavaScript 文件; 如果没有, 就需要在 Apache 配置文件中设置)。

清单 11. 将 JavaScript 文件缓存两天的指令

```
ExpiresActive on
ExpiresDefault "access plus 2 days"
```

当然, 缓存 JavaScript 文件的问题是, 如果进行更改, 则使用缓存版本的用户在超时期限内再次访问将使用缓存的版本, 而非更新后的版本。幸运的是, 您可以通过向加载脚本的 <script> 标记中添加带版本号查询语句来强制用户获取最新版本。该查询语句对 JavaScript 代码没有影响, 但对于浏览器而言, 这是个完全独立的文件, 并会下载新版本。当然, 每次更改文件后就增加版本号很重要。在大型应用程序中, 应建立自动处理此过程的脚本, 以防此类问题发生。

[回页首](#)

结束语

本文中, 您了解到 JavaScript 和 Ajax 性能在现代网站和应用程序中的重要性。首先, 您了解好的 JavaScript 编程事件如何对应用程序的响应和加载时间做出巨大改变。然后, 您了解到如何通过使用分析和网络分析工具测量现有应用程序的性能。最后, 您了解到如何通过使用一些简单但强大的技术减小文件大小和应用程序发出的 HTTP 请求数来加速现有应用程序, 而无需修改应用程序源代码。

参考资料

学习

- | [掌握 Ajax, 第 2 部分: 使用 JavaScript 和 Ajax 进行异步请求](#) (Brett McLaughlin, developerWorks, 2006 年 1 月): 学习如何使用 Ajax 和 XMLHttpRequest 对象创建不让用户等待服务器响应的请求/响应模型。
- | [开发移动 Web Ajax 应用](#) (Michael Galpin, developerWorks, 2010 年 3 月): 学习如何使用 Ajax 创建跨浏览器的智能手机 web 应用程序。
- | [在应用程序中使用 Ajax 的时机](#) (Jesse Skinner, developerWorks, 2008 年 2 月): 学习如何使用 Ajax 改善网站, 同时也避免不好的用户体验。
- | [改善 Web 2.0 应用程序的性能](#) (Jian Qiao Sun 和 Hua Pin Shen, developerWorks, 2009 年 12 月): 探索不同浏览器端缓存机制。
- | [Ajax 性能分析](#) (Kristopher William Zyp, developerWorks, 2008 年 4 月): 查看能发现并解决包含 Ajax 的应用程序中的性能问题的工具。
- | [JavaScript 框架比较](#) (Joe Lennon, developerWorks, 2010 年 2 月): 简要了解能大大增强 JavaScript 开发的框架。
- | [High Performance Web Sites](#) Steve Souders 著 (O'Reilly Media, 2007 年 12 月): 了解能用来加速网站运行的 14 个具体工具。
- | [High Performance JavaScript](#) Nicholas Zakas 著 (O'Reilly Media, 2010 年 3 月): 学习能帮助您减少开发中瓶颈的技术和策略。
- | [Best Practices for Speeding Up Your Web Site](#): 这篇来源于 Yahoo! Developer Network 的文章包含 35 项使 web 页面快速运行的最佳实践。
- | 访问 [Mozilla Developer Network](#), 这是各种开发人员的社区网络, 涵盖 Web、 Mobile、 Add-ons、 Applications 和 Labs。
- | 在 [Introducing JSON](#) 一文中学习 JSON 语法。
- | [developerWorks 中国网站 Web 开发专区](#) 专门提供涵盖各种基于 web 解决方案的文章。
- | 要收听面向软件开发人员的有趣访谈和讨论, 请查看 [developerWorks 播客](#)。

获得产品和技术

- | [LazyLoad 库](#) 是一个小型 (只有 1,541 字节)、独立的 JavaScript 库, 可用来根据需要轻松加载 JavaScript 和 CSS 文件。
- | 获取 [Firebug](#) web 开发工具。
- | [YUI Profiler](#) 是一个用于 JavaScript 的简单的不可见的代码分析器。
- | [YSlow](#) 根据高性能 web 页面的规则分析 web 页面并给出提升页面性能的建议。

- 丨 [Fiddler](#) 是一个记录计算机与 Internet 之间所有 HTTP(S) 流量的 Web Debugging Proxy。
- 丨 用 [YUI Compressor](#) 缩减 JavaScript 代码。
- 丨 使用 [IBM 产品评估试用版软件](#) 改进您的下一个开发项目, 这些软件可以通过下载或从 DVD 获得。

讨论

- 丨 现在就创建 [developerWorks 社区配置文件](#), 建立 JavaScript 或 Ajax 的 [观察列表](#)。与 [developerWorks 社区](#) 取得联系, 保持联系。
- 丨 找到其他 [对 web 开发感兴趣的 developerWorks 成员](#)。
- 丨 分享您所知道的: [加入某一主题的 developerWorks 小组](#)。
- 丨 Roland Barcia 在博客中谈论 [Web 2.0 和中间件](#)。
- 丨 查看 developerWorks 成员的 [web 主题共享书签](#)。
- 丨 快速获取答案: 访问 [Web 2.0 Apps 论坛](#)。
- 丨 快速获取答案: 访问 [Ajax 论坛](#)。