SAMPLE CHAPTER

FAST ASP.NET WEBSITES

Dean Alan Hume

MANNING

Fast ASP.NET Websites

by Dean Alan Hume

Chapter 2

brief contents

PART 1 DEFINING PERFORMANCE.....1

1 ■ High-speed websites 3

2 ■ First steps toward a faster website 11

PART 2 GENERAL PERFORMANCE BEST PRACTICES 27

3 ■ Compression 29

4 ■ Caching: The sell-by date 43

5 ■ Minifying and bundling static files 59

6 ■ HTML optimization tips 75

7 ■ Image optimization 99

8 ■ ETags 117

9 ■ Content Delivery Networks 125

PART 3 ASP.NET-SPECIFIC TECHNIQUES 137

10 ■ Tweaking ASP.NET MVC performance 139

11 ■ Tweaking ASP.NET Web Forms performance 155

12 ■ Data caching 171

2 First steps toward a faster website

This chapter covers

- The basics of HTTP
- Empty cache versus primed cache
- Tips and tools for interpreting performance charts
- What does it all mean?
- Performance rules to live by

This chapter runs through the basic tools and skills that you need to know in order to start analyzing your site. You're going to start by learning the basics of HTTP and understanding HTTP requests and responses. You'll also run through performance charts and the tools you can use to create them. By the time you're finished with this chapter, you'll be able to dive straight into coding.

2.1 The basics of HTTP

HTTP, the foundation of all communication on the web, allows browsers and servers to communicate with each other using a request-and-response communication system. HTTP, in the most simple terms, is like a conversation: one person is the

browser requesting information, and the other is the server, responding with a result. You (the browser) then interpret the response and act accordingly.

In general, the client always initiates the conversation and the server replies. These HTTP requests and responses contain data that is readable to the human eye, which makes it easy to follow and understand. Most modern browsers come with a set of free tools that enable you to monitor these messages easily.

HTTP messages are made up of a header and a body. The HTTP header contains important data about the client browser, the requested page, and more. It's transmitted in a key/value pair format and is the core part of an HTTP message. Then, in the most basic type of HTTP request, the HTTP message body will contain data being sent to the server. In a request, this is where user-entered data or uploaded files are sent to the server.

Each HTTP request also contains an HTTP verb that tells the server what to do with the data being sent across. You may be familiar with the two most common verbs—GET and POST. They often appear in the HTML action form attributes.

GET is used to request a resource without expecting to change that resource (for example, loading a website's homepage). POST is used to submit data to the resource, which is then updated (for example, submitting your details when changing your user preferences on a website).

2.1.1 Understanding an HTTP GET request

The most common type of HTTP request is GET. Every time you type a URL in your browser and hit return, the action fires off a GET request. Figure 2.1 shows a typical GET request. I have used one of the built-in browser tools that we're going to cover later in this chapter to view its internal contents.

The information inside an HTTP request is full of useful details; it's up to you to understand exactly what is happening.

Figure 2.1 An HTTP GET request

In figure 2.1, you'll notice a typical HTTP request to www.mozilla.org.

- The Request Method is listed as a GET, and the Status Code is 200, which means it was successful.
- The Accept header field tells the server which content types are acceptable. In this case, the browser is accepting HTML. The browser also tells the server it supports other content types in case the server doesn't support the first one it asks for. The string containing multiple content types is chained together for efficiency, meaning that the browser doesn't have to request multiple content types one at a time if the first request fails. The Accept-Charset tells the server which character encoding is acceptable (such as ASCII, UTF-8, etc.) For this request, it is ISO-8859-1, UTF-8.
- In the field Accept-Encoding, the browser is letting the server know that it supports Gzip, Deflate, and SDCH compression types. If the data the server sends back is compressed, it will understand how to decompress it and display it to the user. We'll cover compression in chapter 3.
- The browser uses the Accept-Language field to tell the server which languages it can use to respond. In this case it can respond in en-GB and en-US.
- The Connection field tells the server what type of connection the user-agent would prefer. In this request, the browser has asked for a keep-alive connection type.
- The User-Agent field is a text string that browsers and devices use to identify themselves for tracking and other purposes.

2.1.2 Understanding an HTTP GET response

After you've made the HTTP request to www.mozilla.org as shown in figure 2.1, the server replies with an HTTP response.

From the request shown in figure 2.2, you can see the following:

- The server tells the browser that this component can be cached. In this case it's using the Cache-Control field and notifying the browser that it can be cached for 600 seconds. All Cache-Control timings are measured in seconds.

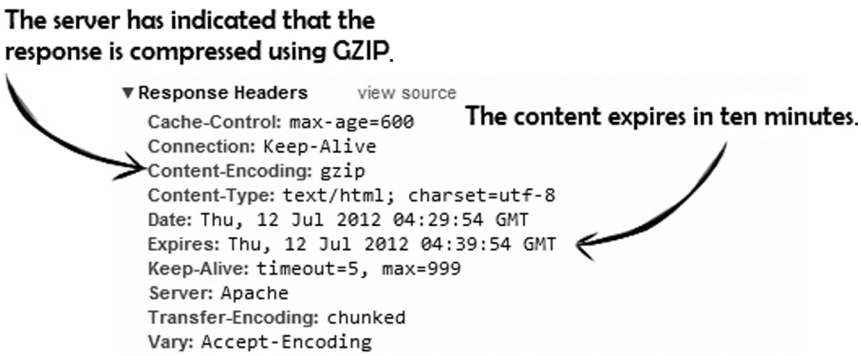


Figure 2.2 An HTTP GET response

- The Connection field shows Keep-Alive is supported.
- The Keep-Alive field tells the browser which connection limits it supports (timeouts and max connection time).
- The Content-Encoding field tells the browser that the server is sending data back that's encoded with Gzip. Now the browser can decompress that data with the appropriate coding. You'll look at compression more closely in chapter 3.
- The component's Content-Type is text/html.
- The Date field tells the browser the date it was processed so it can cache it if necessary.
- The Expires field tells the browser how long it's allowed to keep the component. The Expires field is a date and time in the future. If it's far enough into the future, the browser may choose to cache the component. This is a good thing, because it saves the browser from having to request that component again from the server for a specified period of time. This speeds up the download because there is one less request to make. You'll look into Expires headers and how to apply them in chapter 4.
- Transfer-Encoding is the type of encoding used to send data back to the browser. In this case it was *chunked*, which means that the data was sent over by the server in a series of chunks instead of all at once.
- The Vary field instructs the proxies to cache two versions of the resource: one compressed and one uncompressed. Without Vary, a server may mistakenly send users the incorrect cached version of an HTML page instead of the correct one for its encoding type.

In chapter 1, you learned that multiple components are downloaded when you request a single URL. These HTTP requests and responses are made for each component in the HTML document and they all have similar fields in the headers. If you refer to the request for www.mozilla.org, you can see that multiple requests are made when you enter the URL. When you request the URL, the HTML document is downloaded, and as the browser parses its contents, it starts to download the additional components it finds inside the HTML.

From the chart in figure 2.3, you can see the multiple GET requests downloaded when you accessed www.mozilla.org. As the browser parsed and located additional components inside the HTML document, it started to download them. Each component is one round-trip that the browser has to make to the server, meaning sending a request and waiting for a response takes time. As a developer, you can find ways to reduce the number of server requests that the browser makes. You can also influence how the browser behaves by telling it to cache the information it downloads. This is what web performance is all about—reducing the number of HTTP requests sent to the server.

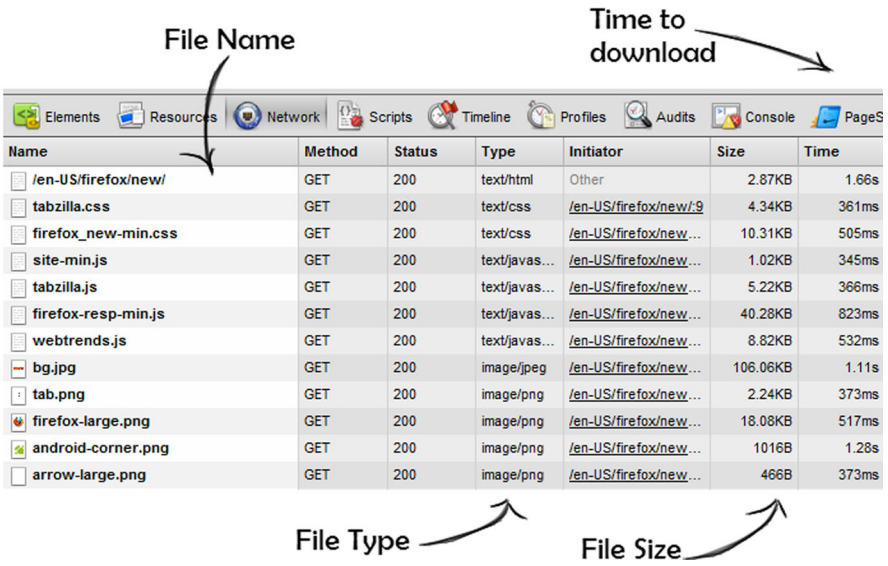


Figure 2.3 Multiple requests to www.mozilla.org

2.1.3 Understanding HTTP status codes

When a request is made to your server for a component on your website, the server returns an HTTP status code, as shown in table 2.1. The status code tells the browser if it retrieved the component and it tells the browser how to react. Status codes can be quite handy when you're debugging an application.

Table 2.1 An HTTP status code list

HTTP status code range	Description
100–1xx	Informational: Request received, continuing process.
200–2xx	Success: The action requested by the client was received, understood, accepted, and processed successfully.
300–3xx	Redirection: The client must take additional action to complete the request.
400–4xx	Client Error: This status code is intended for cases in which the client seems to have made an error.
500–5xx	Server Error: The server failed to fulfill a valid request.

For a full list of status codes, visit http://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

2.2 Empty cache vs. primed cache

A web browser stores certain downloaded items for future use in a browser cache folder. Images, JavaScript, CSS, and even entire web pages are examples of cache items. Before you start looking at tools that allow you to profile a website, it's important to understand the differences between an empty cache and a primed cache. The first time you visit a website, you won't have a cache of that site. But as you continue to browse the website, the browser cleverly stores the components you download in this temporary folder cache. The next time you visit the same website, you'll have a primed cache that contains the website's cached items. The browser does this so that on subsequent visits to the same website, it can easily retrieve the components, which speeds up your download time.

The empty cache shown in figure 2.4 represents the first time a user visits a site. Compare that to the primed cache for the same website. You can see the difference in both the number of HTTP requests and the total weight of the web pages. The total weight of the web page on the left is 130K; the total weight for the primed cache on the right is 39.8K. All the components that were saved from the first visit were retrieved from the cache, thus cutting the download time and weight drastically.

An important question to ask yourself when analyzing your website is "How many of my users are first-time visitors (who will have an empty cache) and how many are repeat visitors (who will have a primed cache)?" When you answer this question, based on statistics gathered while using a website analytics package, you'll understand where to focus while you're optimizing your website's performance.

If you don't use a website analytics package or you don't have enough data to determine visitor trends, it can be helpful to think about the domain of your website. Is it an intranet website that might expect a lot of repeat visits throughout the day? Is it a site expecting to attract a lot of new visits? This mindset allows you to put yourself in the shoes of the user so you can improve and enhance their site experience.

It's also important to note that both the primed and empty caches of a browser need to be taken into account when profiling, implementing, and monitoring a web

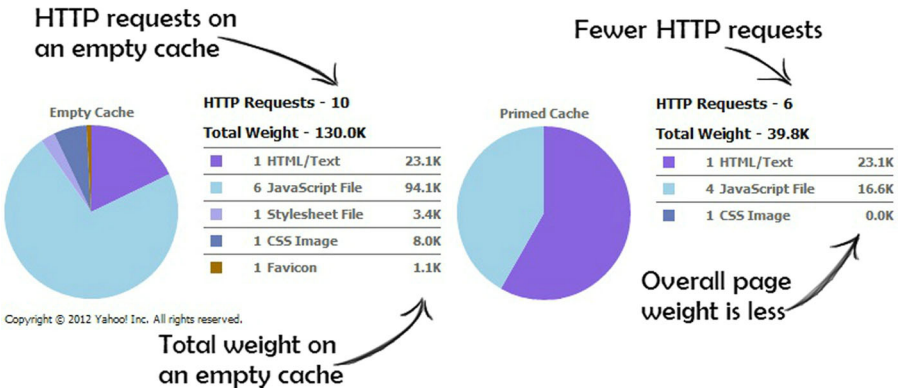


Figure 2.4 The difference between an empty cache and a primed cache. Notice the differences in the total weight and the number of HTTP requests.

page. Sometimes you may find yourself refreshing a web page and getting skewed results because the browser is actually retrieving the components from its cache instead of fetching a fresh version on an empty cache. Most browsers will allow you to refresh a page by hitting the F5 key, which might load the page from cache. But Ctrl-F5 forces a cache refresh, and will guarantee that you'll get the newest content. You may also find that some browsers allow you to force a cache refresh by holding Shift (or Ctrl) and clicking the refresh icon. Keep this in mind when you're profiling your site because you might be profiling a web page that's been updated on the server but isn't reflected in your browser because of caching.

2.3 Tips and tools for interpreting performance charts

Now that we've talked about what's going on under the hood when you make a browser request, we can start interpreting performance charts.

To understand how to improve your website's performance, it's vital that you learn how to read performance charts. The most typical charts you'll come across with today's profiling tools are *waterfall charts*, diagrams that show downloads in a linear progression, in a manner that looks like a waterfall.

Most modern browsers come with their own built-in developer tools and a version of a waterfall chart that can be easily accessed via the Tools menu. Most browsers also have a hotkey for developer tools and F12 seems to bring up the developer panel. You may need to check your browser settings first, as different browsers might organize their tools differently. In the next section, you'll go through a brief review of well-known developer tools you can use to produce performance charts; after that, you'll learn how to interpret the data from these charts in greater depth.

2.3.1 What does it all mean?

When you look at the waterfall chart in figure 2.5, you can see that it gives you so much more than a series of simple requests.

In a waterfall chart, the length of the bars shows how long a certain resource took to download. In figure 2.5 one bar is extremely long, highlighting an area to investigate. Is the image file size too large? Is the image the correct format? What is causing this image to download so slowly?

The green vertical line (at 1.2) running through all the requests indicates the `DOMContentLoaded` event. The `DOMContentLoaded` event is triggered when the page's Document Object Model (DOM) is ready, which means that the API for interacting with the content, style, and structure of a page is ready to receive requests from your code.

The blue vertical line (near 3.2) indicates the `Load` event being fired. The `Load` event is triggered when the entire page has loaded and is generally the moment when the loading icon in your browser's title bar stops spinning. When this has happened, all JavaScript and Cascading Style Sheets (CSS) have finished loading and have been executed, and all images have been downloaded and displayed. The `Load` event lines help you see how long it takes for pages to load and helps you understand when the

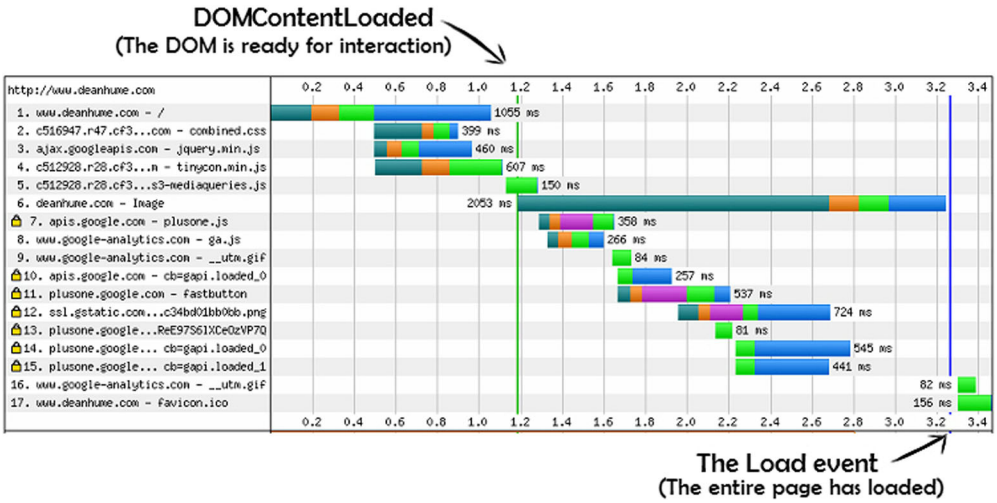


Figure 2.5 Waterfall chart for www.deanhume.com

browser is parsing and loading your website's components. So a green line indicates that the `DOMContentLoaded` event is triggered and the browser is ready to interact with requests from your code, and a blue line indicates that the `Load` event has been triggered and all JavaScript and CSS have finished executing. Different developer tools may provide a different color for each vertical line, but they generally mean the same thing.

If you refer to the line in figure 2.5 that was taking a long time to load, you can see that it's pushing the `Load` event out. There's a gap of whitespace between item 15 being finished loading and item 16 starting. It's as if the image were blocking progress and causing a slight delay, so this is definitely an area to investigate further.

The waterfall chart also shows three JavaScript files being downloaded every time you access the page. Are all three files necessary? Could you reduce these to one request? As you start optimizing your website and looking at ways to improve its performance, you'll need to keep asking yourself these sorts of questions.

Depending on your organization, your website may be image- or style-intensive. Figure 2.6 is the waterfall chart for a popular online clothing store in the UK. You'll see a large number of HTTP requests being made, so many that I had to crop the image.

Due to the nature of the business, high-quality images of the clothes are necessary, but this can have a very negative effect on web page performance. In chapter 7, you'll learn image optimization techniques that reduce the overall size of image files without reducing their quality.

Another area that could be addressed immediately is the number of JavaScript files on the page. Combining them into one file could drastically reduce HTTP requests. In chapter 5, you'll look at minifying and bundling static files, which will reduce the number of HTTP requests and the size of the requests that a browser needs to make.

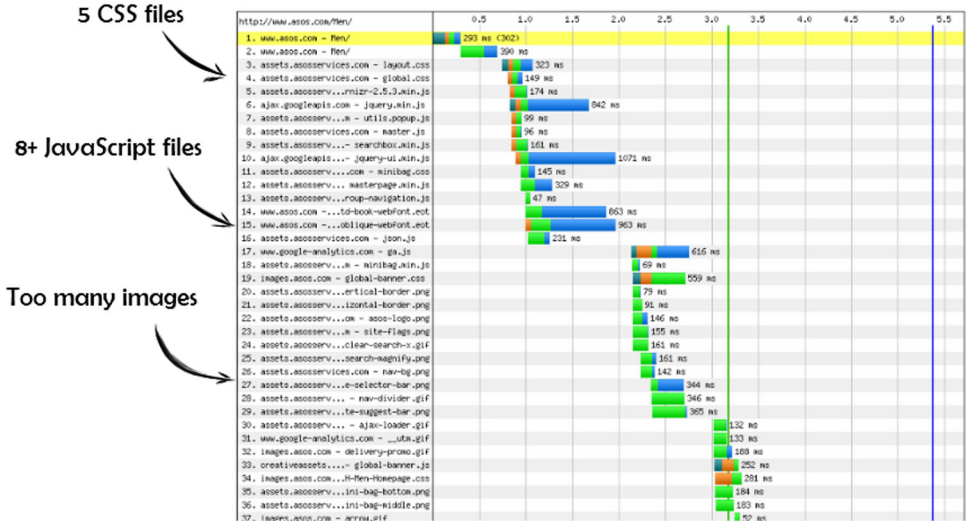


Figure 2.6 Waterfall chart for Asos.com

At first glance, you may notice obvious ways to make performance improvements, but the solution may not be glaringly obvious. If you keep the two main principles (reducing the number of requests and the size of the requests) in mind when profiling your site, finding the solution will be a lot easier.

Each profiling tool will produce waterfall charts with slightly different features. It's up to you to decide which tool and browser you prefer. As you become familiar with your chosen tool, you'll find it easier to read and spot areas for improvement.

2.3.2 Google Chrome developer tools

In figure 2.7, I use Chrome developer tools to show you how to produce a waterfall chart. To access Chrome's developer tools, navigate to the Settings menu and bring up the developer tools in the Tools menu; you could also hit F12.

If you bring up the Network panel and navigate to my website (www.deanhume.com), you can see the components being downloaded as you reach the home page as shown in figure 2.8. The developer tools also show you the waterfall chart and the order in which the components were downloaded.

The waterfall chart also takes latency into account. *Latency* is the amount of time it takes to open a connection to the server and is associated with the round-trip time that it takes for a request to reach the server and return to the user. The amount of latency depends largely on how far away the user is from the server. It's shown in the lighter shade within each bar.

The chart is color coded, with each hue representing a content type, such as JavaScript files, CSS, or images. This helps you see and visually group the different content types.

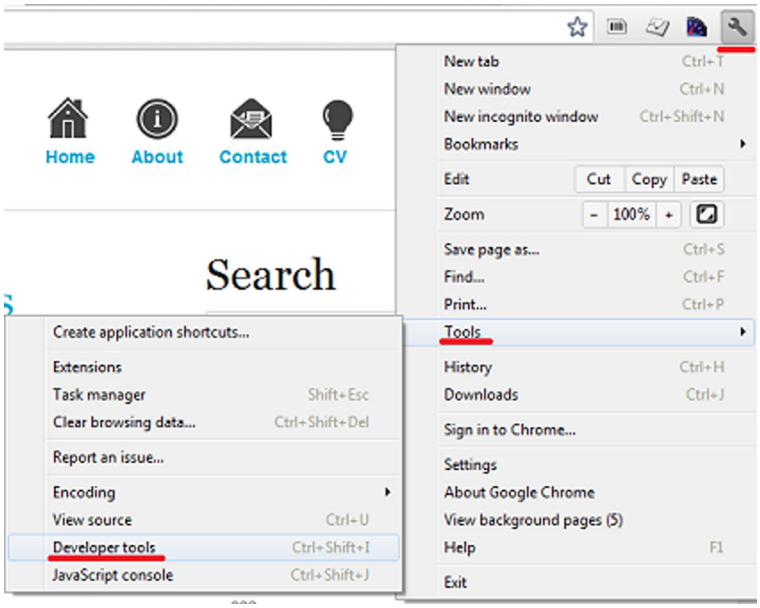
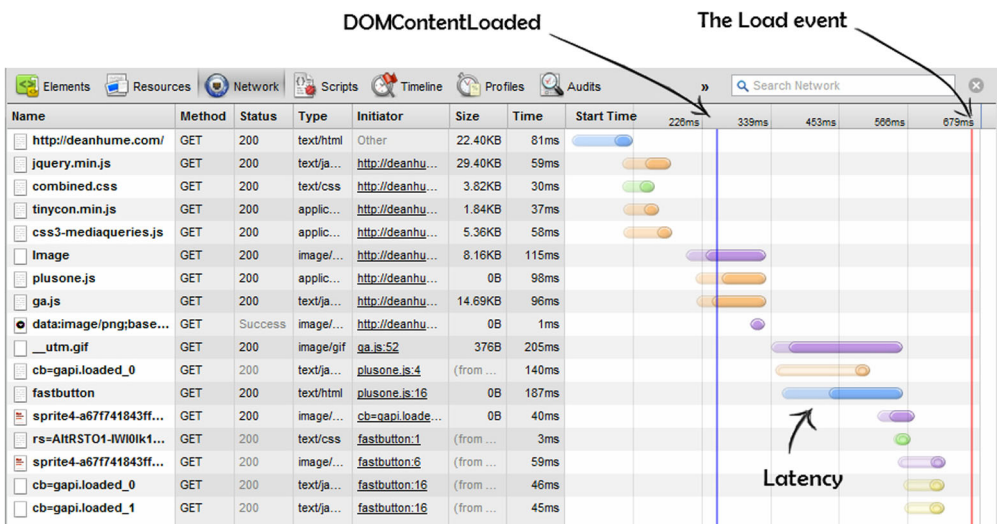


Figure 2.7 Accessing Chrome developer tools

The Chrome developer tools also allow you to edit HTML and CSS on the fly, which is a handy way to develop quickly and make small changes without having to reload the page. Next, you'll run through a few other browser developer tools and show the differences between waterfall charts and other profiling tools.



Name	Method	Status	Type	Initiator	Size	Time	Start Time
http://deanhume.com/	GET	200	text/html	Other	22.40KB	81ms	
jquery.min.js	GET	200	text/ja...	http://deanhu...	29.40KB	59ms	
combined.css	GET	200	text/css	http://deanhu...	3.82KB	30ms	
tinycon.min.js	GET	200	applic...	http://deanhu...	1.84KB	37ms	
css3-mediaqueries.js	GET	200	applic...	http://deanhu...	5.36KB	58ms	
Image	GET	200	image/...	http://deanhu...	8.16KB	115ms	
plusone.js	GET	200	applic...	http://deanhu...	0B	98ms	
ga.js	GET	200	text/ja...	http://deanhu...	14.69KB	96ms	
data:image/png;base...	GET	Success	image/...	http://deanhu...	0B	1ms	
__utm.gif	GET	200	image/gif	ga.js:52	376B	205ms	
cb=gapi.loaded_0	GET	200	text/ja...	plusone.js:4	(from ...	140ms	
fastbutton	GET	200	text/html	plusone.js:16	0B	187ms	
sprite4-a67f741843ff...	GET	200	image/...	cb=gapi.loade...	0B	40ms	
rs=AltRSTO1-IWI0Ik1...	GET	200	text/css	fastbutton:1	(from ...	3ms	
sprite4-a67f741843ff...	GET	200	image/...	fastbutton:6	(from ...	59ms	
cb=gapi.loaded_0	GET	200	text/ja...	fastbutton:16	(from ...	46ms	
cb=gapi.loaded_1	GET	200	text/ja...	fastbutton:16	(from ...	45ms	

Figure 2.8 Waterfall chart for www.deanhume.com using Chrome developer tools. The figure also shows the latency that can sometimes be associated with the download time of a component.

NOTE Remember that certain components on a web page may be cached, which can affect your waterfall chart's accuracy. Run the tool and produce charts for both primed and empty caches in order to get a more complete picture.

2.3.3 Internet Explorer developer tools

The developer tools in Internet Explorer have been around since IE 6 and have evolved with each new version. In IE 9, the tools allow you to debug JavaScript, HTML, and CSS as well as profile the performance of a web page using the handy profiling reports. They can be easily accessed by hitting the F12 key, as seen in figure 2.9.

2.3.4 Firebug

Firebug is a free and open source tool that was originally built as an extension for Firefox. It has been around since 2006 and is a solid and proven tool. Firebug was one of the first developer tools to produce a waterfall chart, and most other developer tools have produced similar waterfall charts based on this original style. Much like Chrome's developer tools, it allows you to edit HTML and CSS on the fly.

Using the Net tab allows you to easily view a waterfall chart, and by expanding on the individual nodes, you can view the HTTP requests and responses. Although Firebug was originally intended for Firefox, it's also available as a plugin for Chrome. For more information, visit getfirebug.com.

2.3.5 Safari Web Inspector

If you develop for Mac users or just prefer to use Safari, it also has a free tool, called Web Inspector, which allows you to inspect network traffic. You may also notice that the layout and design are very similar to the Chrome developer tools. Safari and Chrome are powered by WebKit.

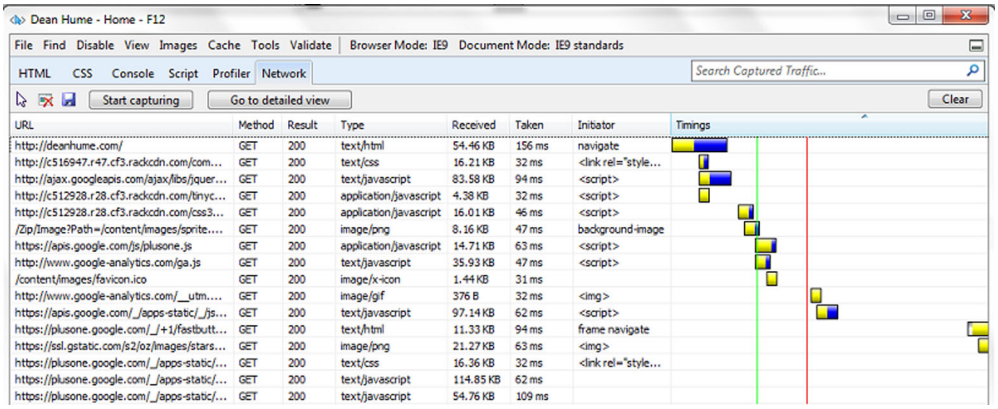


Figure 2.9 Waterfall chart for www.deanhume.com using IE developer tools

2.3.6 HTTPWatch

HTTPWatch is an integrated HTTP sniffer for IE and Firefox that allows you to watch and "sniff" the HTTP traffic coming to and from your website. It provides a great set of tools that allow you to easily profile your site's performance, as well. It doesn't come built into any browsers, but a free basic version and an advanced version with more features can be purchased and downloaded from www.httpwatch.com.

2.3.7 WebPagetest

You can find an extremely handy tool to profile your site at www.webpagetest.org. It isn't built into any browser, but it can provide a wealth of information about any website (figure 2.10). Simply visit the site and enter the URL you wish to profile.

WebPagetest is an open source project that's developed and supported primarily by Google. Many partners also work with WebPagetest and provide a test location for you to run your site against. I really like the way you can profile your site against a location from almost anywhere in the world, using multiple browsers. It's especially handy if you need to see what your users would see if they accessed your site from halfway across the world. These test locations provide useful insight into the round-trips the browser will make to download the required components. There is even an option to record video of the page as it loads, which can be very useful to compare and review page rendering.

WebPagetest provides a breakdown of the first view and the repeat view, allowing you to see how many requests you saved by using caching, Expires headers, and so on. You can also experiment with different advanced features if your website has a complex setup. Throughout this book, I will refer to www.webpagetest.org because it provides a great set of charts that give us an in-depth look at a website's performance.

2.3.8 Fiddler

Another fantastic free tool is Fiddler. This web debugging proxy logs all network traffic between your computer and the internet and lets you inspect traffic, set

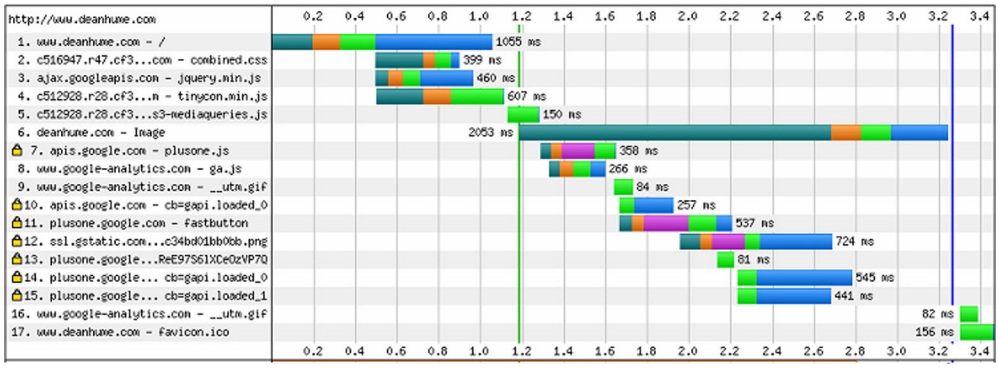


Figure 2.10 Waterfall chart for www.deanhume.com using www.webpagetest.org

breakpoints, and "fiddle" with incoming or outgoing data. It can be quite interesting to fire up Fiddler and watch the requests coming and going from your PC, let alone the website that you are profiling!

The Fiddler dashboard gives you an in-depth look at the HTTP requests and allows you to create and test HTTP requests yourself. Fiddler also offers a whole host of other great features. For more information, point your browser at www.fiddler2.com/fiddler2.

2.4 Performance rules to live by

In 2007, Steve Souders, at the time Chief Performance Yahoo! at Yahoo!, created a set of 14 rules for faster front-end performance. These rules, shown in table 2.2, are outlined in his book, *High Performance Web Sites*, and every single one is widely accepted as best practice in web performance today.

Table 2.2 Steve Souders's rules for faster front-end performance

Rule Number	Description
1	Make fewer HTTP requests
2	Use a content delivery network
3	Add an Expires header
4	Compress components with Gzip
5	Put CSS at the top
6	Move JavaScript to the bottom
7	Avoid CSS expressions
8	Make JavaScript and CSS external
9	Reduce DNS lookups
10	Minify JavaScript
11	Avoid redirects
12	Remove duplicate scripts
13	Turn off ETags
14	Make AJAX cacheable and small

As the web has evolved, the number of rules has increased, but every core concept in this book is based on Souders's 14 rules. There may be newer browsers that can handle HTML5, but these original rules have been proven and tested, and they underpin everything that you are trying to achieve.

Many of these rules closely align with this book's table of contents. As you progress through the chapters, you'll learn about the performance rules, as well as some

of the newer concepts that have evolved with the introduction of HTML5 and advances in JavaScript.

There are a lot of performance techniques and methods that can be applied to your website and trying to remember them all when you're profiling your site can be quite daunting. This is where a performance-profiling tool can be very helpful. Instead of remembering each and every technique, these tools take the hard work out of profiling and provide a set of suggestions and best practices that can be applied to your website. Let's look at two such performance-analysis tools, Yahoo! YSlow and Google PageSpeed.

2.4.1 Yahoo! YSlow

YSlow is a great add-on for many browsers and it offers suggestions for improving a web page's performance. It's free and can be downloaded from http://developer.yahoo.com/yslow/ for Firefox, Chrome, Opera, and Safari. The tool runs against a set of 23 rules that affect web page performance. Throughout the remainder of this book, you'll come back to this tool to see how each improvement you make boosts your performance score.

YSlow provides a grade and overall performance score for your URL. It grades A as high performance and F as poor. You should, obviously, always aim for the highest grade you can obtain because each step closer to an A improves your web page performance. Figure 2.11 shows performance areas on my website that need to be improved.

In figure 2.11 you'll notice that the overall performance score for my website is quite high, but one area scored an E. Obviously I need to add an Expires header to

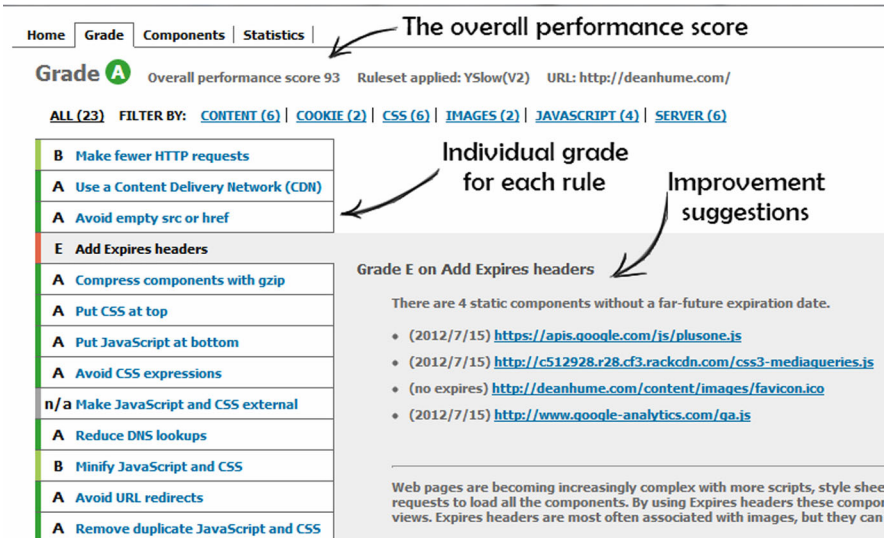


Figure 2.11 Yahoo! YSlow tool run against www.deanhume.com

certain components on the web page. As you recall from examples earlier in the chapter, these Expires headers let the browser know that the component doesn't need to be downloaded again because the content hasn't changed. It will only expire in the future—this saves the browser a round-trip to the server again, thus speeding up the load time. You'll look at code and the different ways in which you can add Expires headers to the components in your web page in chapter 4.

2.4.2 Google PageSpeed

Google also has a handy performance tool called PageSpeed, shown in figure 2.12, which can be added to both Firefox and Chrome. It's very similar to YSlow and was built using the same performance rules set out by the Yahoo! performance team. PageSpeed has grown to become a great tool that allows you to easily profile your site. If you would like to try the tool before adding it to your browser, Google offers you the ability to do so on the PageSpeed Insights web page (https://developers.google.com/speed/pagespeed/insights).

In figure 2.12, you can see the results of a test run against my site. Much like YSlow, it has given me a similar suggestion, letting me know that I need to use browser caching by setting an expiration date on some resources. The PageSpeed tool provides a very simple interface that suggests only the improvements you need to make. Unlike YSlow, it doesn't give you a breakdown of the components or the empty versus primed cache view. I find it very helpful to have the full dashboard that YSlow provides, but I also like to incorporate the performance results from Google PageSpeed into my overall performance profiling. Each tool provides its own rule set and logic to determine the score.

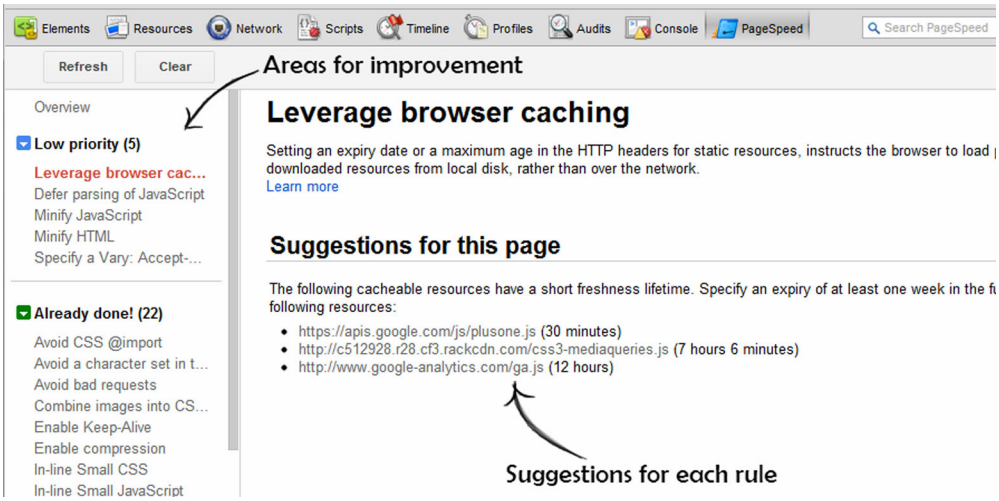


Figure 2.12 Google PageSpeed tool run against www.deanhume.com

NOTE Some of the performance profiling tools may offer a setting that enables you to autorun the tests every time a web page is loaded. Although this may be helpful during testing and development, remember that it needs to process a full set of rules and code, which may take time. It may feel as if a web page is running slowly, but actually it's the profiling tool running in the background. Don't forget to turn it off when you aren't using it!

2.5 Summary

In this chapter, you started off learning the basics of HTTP and understanding how the browser makes a request to the server and gets a response. Next, you had a brief summary of the tools that are freely available with most browsers. These developer tools will help you start profiling and analyzing HTTP requests and responses from your websites. The waterfall chart is the most widely used when it comes to profiling your site and it displays the component downloads over a timeline. You evaluated two different web pages with a waterfall chart and saw how you could reduce the overall number of HTTP requests. You also became familiar with two tools that help you measure your applications' performance, as well as suggest areas for improvement. You'll use these tools throughout the book as you work to improve the sample application's speed.

Now that you have a good grasp of the basics of web performance, it's time to start improving your website. In the next chapter you'll learn how to apply compression to your website and make significant speed gains.

ASP.NET/WEB DEVELOPMENT

FAST ASP.NET WEBSITES

Dean Alan Hume

There's a real cost to inefficient HTTP requests, overloaded data streams, and bulky scripts. Server throughput is a precious commodity, and seconds—even tiny fractions of a second—can seem like an eternity while a visitor waits for your site to load. As an ASP.NET developer, there are dozens of techniques you can apply immediately to make your sites and applications faster. You'll find them here.

Fast ASP.NET Websites delivers just what it promises—practical, hands-on guidance to create faster, more efficient ASP.NET sites and applications. This book offers step-by-step .NET-specific examples showing you how to apply classic page optimization tips, ASP.NET-specific techniques, and ways to leverage new HTML5 features.

What's Inside

- Drastically improved response times
- Tips for Webforms and ASP.NET MVC sites
- Optimizing existing pages
- .NET-specific examples

Readers should be familiar with basic HTML, CSS, and ASP.NET concepts.

Dean Hume is a software developer and blogger based in the U.K. A passionate techie, he created the ASP.NET HTML5 toolkit and blogs regularly about web performance at www.deanhume.com.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/FastASP.NETWebsites

"A clear and effective guide to the art of ASP.NET performance tuning."
—Bryn Keller, Jenkon

"Comprehensive, reader-friendly information on how to make your ASP.NET website fly."
—Danylo Kizyma
Advanced Utility Systems

"Demonstrates key concepts in clear detail."
—Michael Roberts, Sr.
Information Innovators

"An up-to-date guide ... focuses on client performance and user experience."
—Onofrio Panzarino
SBG Wolters Kluwe

ISBN 13: 978-1-617291-25-8
ISBN 10: 1-617291-25-0
53499
9 781617 291258

MANNING $34.99 / Can $36.99 [INCLUDING eBOOK]