Minimize payload size

The amount of data sent in each server response can add significant latency to your application, especially in areas where bandwidth is constrained. In addition to the network cost of the actual bytes transmitted, there is also a penalty incurred for crossing an IP packet boundary. (The maximum packet size, or Maximum Transmission Unit (MTU), is 1500 bytes on an Ethernet network, but varies on other types of networks.) Unfortunately, since it's difficult to know which bytes will cross a packet boundary, the best practice is to simply reduce the number of packets your server transmits, and strive to keep them under 1500 bytes wherever possible.

Minimizing the payload size of both dynamic and static resources can reduce network latency significantly. In addition, for scripts that are cached, cutting down their byte size speeds up the time the browser takes to parse and execute code needed to render the page.

Enable compression

Overview

Compressing resources with `gzip` or `deflate` can reduce the number of bytes sent over the network.

Details

Most modern browsers support data compression for HTML, CSS, and JavaScript files. This allows content to be sent over the network in more compact form and can result in a dramatic reduction in download time.

Many web servers can compress files in gzip format before sending them for download, either by calling a third-party module or using built-in routines. To enable compression, configure your web server to set the `Content-Encoding` header to `gzip` format for all compressible resources. You can also use `deflate`, which uses the same compression algorithms, but it is not widely used, so we recommend gzip. If streaming compression imposes too much load on your server, you can usually configure it to pre-compress files and cache them for future downloading.

Note that gzipping is only beneficial for larger resources. Due to the overhead and latency of compression and decompression, you should only gzip files above a certain size threshold; we recommend a minimum range between 150 and 1000 bytes. Gzipping files below 150 bytes can actually make them larger.

Recommendations

Write your web page content to make compression most effective.

To ensure that your content compresses well, do the following:

- Ensure consistency in HTML and CSS code. To achieve consistency:
  - Specify CSS key-value pairs in the same order where possible, i.e. alphabetize them.
  - Specify HTML attributes in the same order , i.e. alphabetize them. Put `href` first for links (since it is most common), then alphabetize the rest. For example, on Google's search results page, when HTML attributes were alphabetized, a 1.5% reduction in the size of the gzipped output resulted.
  - Use consistent casing, i.e. use lowercase wherever possible.
  - Use consistent quoting for HTML tag attributes, i.e. always single quote, always double quote, or no quoting at all where possible.
- Minify JavaScript and CSS. Minifying JavaScript and CSS can enhance compression both for external JS and CSS files and for HTML pages containing inlined JS code and style blocks.

Don't use gzip for image or other binary files.

Image file formats supported by the web, as well as videos, PDFs and other binary formats, are already compressed; using gzip on them won't provide any additional benefit, and can actually make them larger. To compress images, see Optimize images.

Additional resources

Back to top

Remove unused CSS

Overview

Removing or deferring style rules that are not used by a document avoid downloads unnecessary bytes and allow the browser to start rendering sooner.

Details

Before a browser can begin to render a web page, it must download and parse any stylesheets that are required to lay out the page. Even if a stylesheet is in an external file that is cached, rendering is blocked until the browser loads the stylesheet from disk. In addition, once the stylesheet is loaded, the browser's CSS engine has to evaluate every rule contained in the file to see if the rule applies to the current page. Often, many web sites reuse the same external CSS file for all of their pages, even if many of the rules defined in it don't apply to the current page.

The best way to minimize the latency caused by stylesheet loading and rendering time is to cut down on the CSS footprint; an obvious way to do this is to remove or defer CSS rules that aren't actually used by the current page.

Tip: When you run Page Speed against a page referencing CSS files, it identifies all CSS rules that don't apply to that page.

Recommendations

- Remove any inline style blocks containing CSS that is not used by the current page.
- Minify CSS.
- If your site uses external CSS files shared among multiple pages, consider splitting them into smaller files containing rules for specific pages.
- If a page references style rules that are not needed right at startup, put them in a separate .css file and defer loading of the file until the `onload` event is fired.
- If you use JavaScript to generate styles, be sure that those functions aren't called from pages that don't use those styles. This may require some refactoring of JS code.

Back to top

Minify JavaScript

Overview

Compacting JavaScript code can save many bytes of data and speed up downloading, parsing, and execution time.

Details

"Minifying" code refers to eliminating unnecessary bytes, such as extra spaces, line breaks, and indentation. Keeping JavaScript code compact has a number of benefits. First, for inline

JavaScript and external files that you don't want cached, the smaller file size reduces the network latency incurred each time the page is downloaded. Secondly, minification can further enhance compression of external JS files and of HTML files in which the JS code is inlined. Thirdly, smaller files can be loaded and run more quickly by web browsers.

Several tools are freely available to minify JavaScript, including the Closure Compiler, JSMin or the YUI Compressor. You can create a build process that uses these tools to minify and rename the development files and save them to a production directory. We recommend minifying any JS files that are 4096 bytes or larger in size. You should see a benefit for any file that can be reduced by 25 bytes or more (less than this will not result in any appreciable performance gain).

Tip: When you run Page Speed against a page referencing JS files, it automatically runs the Closure Compiler (if available) and JSMin (for inline blocks and if the compiler is not available) on the files and saves the minified output to a configurable directory.

Back to top

Minify CSS

Overview

Compacting CSS code can save many bytes of data and speed up downloading, parsing, and execution time.

Details

Minifying CSS has the same benefits as those for minifying JS: reducing network latency, enhancing compression, and faster browser loading and execution.

Several tools are freely available to minify JavaScript, including the YUI Compressor and cssmin.js.

Tip: When you run Page Speed against a page referencing CSS files, it automatically runs cssmin.js on the files and saves the minified output to a configurable directory.

Back to top

Minify HTML

Overview

Compacting HTML code, including any inline JavaScript and CSS contained in it, can save many bytes of data and speed up downloading, parsing, and execution time.

Details

Minifying HTML has the same benefits as those for minifying CSS and JS: reducing network latency, enhancing compression, and faster browser loading and execution. Moreover, HTML frequently contains inline JS code (in `<script>` tags) and inline CSS (in `<style>` tags), so it is useful to minify these as well.

Note: This rule is experimental and is currently focused on size reduction rather than strict HTML well-formedness. Future versions of the rule will also take into account correctness. For details on the current behavior, see the Page Speed wiki.

Tip: When you run Page Speed against a page referencing HTML files, it automatically runs the Page Speed HTML compactor (which will in turn apply JSMin and cssmin.js to any inline JavaScript and CSS) on the files and saves the minified output to a configurable directory.

Back to top

Defer loading of JavaScript

Overview

Deferring loading of JavaScript functions that are not called at startup reduces the initial download size, allowing other resources to be downloaded in parallel, and speeding up execution and rendering time.

Details

Like stylesheets, scripts must be downloaded, parsed, and executed before the browser can begin to render a web page. Again, even if a script is contained in an external file that is cached, processing of all elements below the script is blocked until the browser loads the code from disk and executes it. However, for some browsers, the situation is worse than for stylesheets: while JavaScript is being processed, the browser blocks all other resources from being downloaded. For AJAX-type applications that use many bytes of JavaScript code, this can add considerable latency.

For many script-intensive applications, the bulk of the JavaScript code handles user-initiated events, such as mouse-clicking and dragging, form entry and submission, hidden elements expansion, and so on. All of these user-triggered events occur *after* the page is loaded and the `onload` event is triggered. Therefore, much of the delay in the "critical path" (the time to load the main page at startup) could be avoided by deferring the loading of the JavaScript until it's actually needed. While this "lazy" method of loading doesn't reduce the total JS payload, it can significantly reduce the number of bytes needed to load the initial state of the page, and allows the remaining bytes to be loaded asynchronously in the background.

To use this technique, you should first identify all of the JavaScript functions that are not actually used by the document before the `onload` event. For any file containing more than 25 uncalled functions, move all of those functions to a separate, external JS file. This may require some refactoring of your code to work around dependencies between files. (For files containing fewer than 25 uncalled functions, it's not worth the effort of refactoring.)

Then, you insert a JavaScript event listener in the head of the containing document that forces the external file to be loaded after the `onload` event. You can do this by any of the usual scripting means, but we recommend a very simple scripted DOM element (to avoid cross-browser and same-domain policy issues). Here's an example (where "deferredfunctions.js" contains the functions to be lazily loaded):

```
<script type="text/javascript">

// Add a script element as a child of the body
function downloadJSAtOnload() {
var element = document.createElement("script");
element.src = "deferredfunctions.js";
document.body.appendChild(element);
}

// Check for browser support of event handling capability
if (window.addEventListener)
window.addEventListener("load", downloadJSAtOnload, false);
else if (window.attachEvent)
window.attachEvent("onload", downloadJSAtOnload);
else window.onload = downloadJSAtOnload;

</script>
```

Back to top

Optimize images

Overview

Properly formatting and compressing images can save many bytes of data.

Details

Images saved from programs like Fireworks can contain kilobytes of extra comments, and use too many colors, even though a reduction in the color palette may not perceptibly reduce image quality. Improperly optimized images can take up more space than they need to; for users on slow connections, it is especially important to keep image sizes to a minimum.

You should perform both basic and advanced optimization on all images. Basic optimization includes cropping unnecessary space, reducing color depth to the lowest acceptable level, removing image comments, and saving the image to an appropriate format. You can perform basic optimization with any image editing program, such as GIMP. Advanced optimization involves further (lossless) compression of JPEG and PNG files. You should see a benefit for any image file that can be reduced by 25 bytes or more (less than this will not result in any appreciable performance gain).

Recommendations

Choose an appropriate image file format.
The type of an image can have a drastic impact on the file size. Use these guidelines:

- PNGs are almost always superior to GIFs and are usually the best choice. IE 4.0b1+, Mac IE 5.0+, Opera 3.51+ and Netscape 4.04+ as well as all versions of Safari and Firefox fully support PNG, including transparency. IE versions 4 to 6 don't support alpha channel transparency (partial transparency) but they support 256-color-or-less PNGs with 1-bit transparency (the same that is supported for GIFs). IE 7 and 8 support alpha transparent PNGs except when an alpha opacity filter is applied to the element. You can generate or convert suitable PNGs with GIMP by using "Indexed" rather than "RGB" mode. If you must maintain compatibility with 3.x-level browsers, serve an alternate GIF to those browsers.
- Use GIFs for very small or simple graphics (e.g. less than 10x10 pixels, or a color palette of less than 3 colors) and for images which contain animation. If you think an image might compress better as a GIF, try it as a PNG and a GIF and pick the smaller.
- Use JPGs for all photographic-style images.
- Do not use BMPs or TIFFs.

Use an image compressor.
Several tools are available that perform further, lossless compression on JPEG and PNG files, with no effect on image quality. For JPEG, we recommend jpegtran or jpegoptim (available on Linux only; run with the `--strip-all` option). For PNG, we recommend OptiPNG or PNGOUT.

Tip: When you run Page Speed against a page referencing JPEG and PNG files, it automatically compresses the files and saves the output to a configurable directory.

Back to top

Serve scaled images

Overview

Properly sizing images can save many bytes of data.

Details

Sometimes you may want to display the same image in various sizes, so you will serve a single image resource and use HTML or CSS in the containing page to scale it. For example, you may have a 10 x 10 thumbnail version of a larger 250 x 250 image, and rather than forcing the user to download two separate files, you use markup to resize the thumbnail version. This makes sense if the actual image size matches at least one — the largest — of the instances in the page, in this case 250 x 250 pixels. However, if you serve an image that is larger than the dimensions used in all of the markup instances, you are sending unnecessary bytes over the wire. You should use an image editor to scale images to match the largest size needed in your page, and make sure that you specify those dimensions in the page as well.

Back to top

Serve resources from a consistent URL

Overview

It's important to serve a resource from a unique URL, to eliminate duplicate download bytes and additional RTTs.

Details

Sometimes it's necessary to reference the same resource from multiple places in a page — images are a typical example. Even more likely is that you share the same resources across multiple pages in a site such as .css and .js files. If your pages do need to include the same resource, the resource should always be served from a consistent URL. Ensuring that one resource is always assigned a single URL has a number of benefits. It reduces the overall payload size, as the browser does not need to download additional copies of the same bytes. Also, most browsers will not issue more than one HTTP request for a single URL in one session, whether or not the resource is cacheable, so you also save additional round-trip times. It's especially important to ensure that the same resource is not served from a different hostname, to avoid the performance penalty of additional DNS lookups.

Note that a relative URL and an absolute URL are consistent if the hostname of the absolute URL matches that of the containing document. For example, if the main page at www.example.com references resource /images/example.gif and www.example.com/images/example.gif, the URLs are consistent. However, if that page references /images/example.gif and mysite.example.com/images/example.gif, these URLs are not consistent.

Recommendations

Serve shared resources from a consistent URL across all pages in a site.

For resources that are shared across multiple pages, make sure that each reference to the same resource uses an identical URL. If a resource is shared by multiple pages/sites that link to each other, but are hosted on different domains or hostnames, it's better to serve the file from a single hostname than to re-serve it from the hostname of each parent document. In this case, the caching benefits may outweigh the DNS lookup overhead. For example, if both mysite.example.com and yoursite.example.com use the same JS file, and mysite.example.com links to yoursite.example.com (which will require a DNS lookup anyway), it makes sense to just serve the JS file from mysite.example.com. In this way, the file is likely to already be in the browser cache when the user goes to yoursite.example.com.

Back to top

Excerpted from *Minimize payload size*
http://code.google.com/speed/page-speed/docs/payload.html

Sign Up & Read Comfortably—Anytime, Anywhere

A subscription to Readability offers great features for mobile reading, saving articles for later and supporting the writers you enjoy. Learn More »