Best Practices for Speeding Up Your Web Site

The Exceptional Performance team has identified a number of best practices for making web pages fast. The list includes 35 best practices divided into 7 categories.

Minimize HTTP Requests

tag: content

80% of the end-user response time is spent on the front-end. Most of this time is tied up in downloading all the components in the page: images, stylesheets, scripts, Flash, etc. Reducing the number of components in turn reduces the number of HTTP requests required to render the page. This is the key to faster pages.

One way to reduce the number of components in the page is to simplify the page's design. But is there a way to build pages with richer content while also achieving fast response times? Here are some techniques for reducing the number of HTTP requests, while still supporting rich page designs.

Combined files are a way to reduce the number of HTTP requests by combining all scripts into a single script, and similarly combining all CSS into a single stylesheet. Combining files is more challenging when the scripts and stylesheets vary from page to page, but making this part of your release process improves response times.

CSS Sprites are the preferred method for reducing the number of image requests. Combine your background images into a single image and use the CSS `background-image` and `background-position` properties to display the desired image segment.

Image maps combine multiple images into a single image. The overall size is about the same, but reducing the number of HTTP requests speeds up the page. Image maps only work if the images are contiguous in the page, such as a navigation bar. Defining the coordinates of image maps can be tedious and error prone. Using image maps for navigation is not accessible too, so it's not recommended.

Inline images use the `data: URL scheme` to embed the image data in the actual page. This can increase the size of your HTML document. Combining inline images into your (cached) stylesheets is a way to reduce HTTP requests and avoid increasing the size of your pages. Inline images are not yet supported across all major browsers.

Reducing the number of HTTP requests in your page is the place to start. This is the most important guideline for improving performance for first time visitors. As described in Tenni Theurer's blog post Browser Cache Usage - Exposed!, 40-60% of daily visitors to your site come in with an empty cache. Making your page fast for these first time visitors is key to a better user experience.

top | discuss this rule

Use a Content Delivery Network

tag: server

The user's proximity to your web server has an impact on response times. Deploying your content across multiple, geographically dispersed servers will make your pages load faster from the user's perspective. But where should you start?

As a first step to implementing geographically dispersed content, don't attempt to redesign your web application to work in a distributed architecture. Depending on the application, changing the architecture could include daunting tasks such as synchronizing session state and replicating database transactions across server locations. Attempts to reduce the distance between users and your content could be delayed by, or never pass, this application architecture step.

Remember that 80-90% of the end-user response time is spent downloading all the components in the page: images, stylesheets, scripts, Flash, etc. This is the *Performance Golden Rule*. Rather than starting with the difficult task of redesigning your application architecture, it's better to first disperse your static content. This not only achieves a bigger reduction in response times, but it's easier thanks to content delivery networks.

A content delivery network (CDN) is a collection of web servers distributed across multiple locations to deliver content more efficiently to users. The server selected for delivering content to a specific user is typically based on a measure of network proximity. For example, the server with the fewest network hops or the server with the quickest response time is chosen.

Some large Internet companies own their own CDN, but it's cost-effective to use a CDN service provider, such as Akamai Technologies, EdgeCast, or level3. For start-up companies and private web sites, the cost of a CDN service can be prohibitive, but as your target audience grows larger and becomes more global, a CDN is necessary to achieve fast response times. At Yahoo!, properties that moved static content off their application web servers to a CDN (both 3rd party as mentioned above as well as Yahoo's own CDN) improved end-user response times by 20% or more. Switching to a CDN is a relatively easy code change that will dramatically improve the speed of your web site.

top | discuss this rule

Add an Expires or a Cache-Control Header

tag: server

There are two aspects to this rule:

- For static components: implement "Never expire" policy by setting far future `Expires` header
- For dynamic components: use an appropriate `Cache-Control` header to help the browser with conditional requests

Web page designs are getting richer and richer, which means more scripts, stylesheets, images, and Flash in the page. A first-time visitor to your page may have to make several HTTP requests, but by using the Expires header you make those components cacheable. This avoids unnecessary HTTP requests on subsequent page views. Expires headers are most often used with images, but they should be used on *all* components including scripts, stylesheets, and Flash components.

Browsers (and proxies) use a cache to reduce the number and size of HTTP requests, making web pages load faster. A web server uses the Expires header in the HTTP response to tell the client how long a component can be cached. This is a far future Expires header, telling the browser that this response won't be stale until April 15, 2010.

```
Expires: Thu, 15 Apr 2010 20:00:00 GMT
```

If your server is Apache, use the ExpiresDefault directive to set an expiration date relative to the current date. This example of the ExpiresDefault directive sets the Expires date 10 years out from the time of the request.

```
ExpiresDefault "access plus 10 years"
```

Keep in mind, if you use a far future Expires header you have to change the component's filename whenever the component changes. At Yahoo! we often make this step part of the build process: a version number is embedded in the component's filename, for example, yahoo_2.0.6.js.

Using a far future Expires header affects page views only after a user has already visited your site. It has no effect on the number of HTTP requests when a user visits your site for the first time and the browser's cache is empty. Therefore the impact of this performance improvement depends on how often users hit your pages with a primed cache. (A "primed cache" already contains all of the components in the page.) We measured this at Yahoo! and found the number of page views with a primed cache is 75-85%. By using a far future Expires header, you increase the number of components that are cached by the browser and re-used on subsequent page views without sending a single byte over the user's Internet connection.

top | discuss this rule

Gzip Components

tag: server

The time it takes to transfer an HTTP request and response across the network can be significantly reduced by decisions made by front-end engineers. It's true that the end-user's bandwidth speed, Internet service provider, proximity to peering exchange points, etc. are beyond the control of the development team. But there are other variables that affect response times. Compression reduces response times by reducing the size of the HTTP response.

Starting with HTTP/1.1, web clients indicate support for compression with the Accept-Encoding header in the HTTP request.

```
Accept-Encoding: gzip, deflate
```

If the web server sees this header in the request, it may compress the response using one of the methods listed by the client. The web server notifies the web client of this via the Content-Encoding header in the response.

```
Content-Encoding: gzip
```

Gzip is the most popular and effective compression method at this time. It was developed by the GNU project and standardized by RFC 1952. The only other compression format you're likely to see is deflate, but it's less effective and less popular.

Gzipping generally reduces the response size by about 70%. Approximately 90% of today's Internet traffic travels through browsers that claim to support gzip. If you use Apache, the module configuring gzip depends on your version: Apache 1.3 uses mod_gzip while Apache 2.x uses mod_deflate.

There are known issues with browsers and proxies that may cause a mismatch in what the browser expects and what it receives with regard to compressed content. Fortunately, these edge cases are dwindling as the use of older browsers drops off. The Apache modules help out by adding appropriate Vary response headers automatically.

Servers choose what to gzip based on file type, but are typically too limited in what they decide to compress. Most web sites gzip their HTML documents. It's also worthwhile to gzip your scripts and stylesheets, but many web sites miss this opportunity. In fact, it's worthwhile to compress any text response including XML and JSON. Image and PDF files should not be gzipped because they are already compressed. Trying to gzip them not only wastes CPU but can potentially increase file sizes.

Gzipping as many file types as possible is an easy way to reduce page weight and accelerate the user experience.

top | discuss this rule

Put Stylesheets at the Top

tag: css

While researching performance at Yahoo!, we discovered that moving stylesheets to the document HEAD makes pages *appear* to be loading faster. This is because putting stylesheets in the HEAD allows the page to render progressively.

Front-end engineers that care about performance want a page to load progressively; that is, we want the browser to display whatever content it has as soon as possible. This is especially important for pages with a lot of content and for users on slower Internet connections. The importance of giving users visual feedback, such as progress indicators, has been well researched and documented. In our case the HTML page is the progress indicator! When the browser loads the page progressively the header, the navigation bar, the logo at the top, etc. all serve as visual feedback for the user who is waiting for the page. This improves the overall user experience.

The problem with putting stylesheets near the bottom of the document is that it prohibits progressive rendering in many browsers, including Internet Explorer. These browsers block rendering to avoid having to redraw elements of the page if their styles change. The user is stuck viewing a blank white page.

The HTML specification clearly states that stylesheets are to be included in the HEAD of the page: "Unlike A, [LINK] may only appear in the HEAD section of a document, although it may appear any number of times." Neither of the alternatives, the blank white screen or flash of unstyled content, are worth the risk. The optimal solution is to follow the HTML specification and load your stylesheets in the document HEAD.

top | discuss this rule

Put Scripts at the Bottom

tag: javascript

The problem caused by scripts is that they block parallel downloads. The HTTP/1.1 specification suggests that browsers download no more than two components in parallel per hostname. If you serve your images from multiple hostnames, you can get more than two downloads to occur in parallel. While a script is downloading, however, the browser won't start any other downloads, even on different hostnames.

In some situations it's not easy to move scripts to the bottom. If, for example, the script uses `document.write` to insert part of the page's content, it can't be moved lower in the page. There might also be scoping issues. In many cases, there are ways to workaround these situations.

An alternative suggestion that often comes up is to use deferred scripts. The `DEFER` attribute indicates that the script does not contain document.write, and is a clue to browsers that they can continue rendering. Unfortunately, Firefox doesn't support the `DEFER` attribute. In Internet Explorer, the script may be deferred, but not as much as desired. If a script can be deferred, it can also be moved to the bottom of the page. That will make your web pages load faster.

top | discuss this rule

Avoid CSS Expressions

tag: css

CSS expressions are a powerful (and dangerous) way to set CSS properties dynamically. They were supported in Internet Explorer starting with version 5, but were deprecated starting with IE8. As an example, the background color could be set to alternate every hour using CSS expressions:

```
background-color: expression( (new Date()).getHours()%2 ? "#B8D4FF" : "#F08A00" );
```

As shown here, the `expression` method accepts a JavaScript expression. The CSS property is set to the result of evaluating the JavaScript expression. The `expression` method is ignored by other browsers, so it is useful for setting properties in Internet Explorer needed to create a consistent experience across browsers.

The problem with expressions is that they are evaluated more frequently than most people expect. Not only are they evaluated when the page is rendered and resized, but also when the page is scrolled and even when the user moves the mouse over the page. Adding a counter to the CSS expression allows us to keep track of when and how often a CSS expression is evaluated. Moving the mouse around the page can easily generate more than 10,000 evaluations.

One way to reduce the number of times your CSS expression is evaluated is to use one-time expressions, where the first time the expression is evaluated it sets the style property to an explicit value, which replaces the CSS expression. If the style property must be set dynamically throughout the life of the page, using event handlers instead of CSS expressions is an alternative approach. If you must use CSS expressions, remember that they may be evaluated thousands of times and could affect the performance of your page.

top | discuss this rule

Make JavaScript and CSS External

tag: javascript, css

Many of these performance rules deal with how external components are managed. However, before these considerations arise you should ask a more basic question: Should JavaScript and CSS be contained in external files, or inlined in the page itself?

Using external files in the real world generally produces faster pages because the JavaScript and CSS files are cached by the browser. JavaScript and CSS that are inlined in HTML documents get downloaded every time the HTML document is requested. This reduces the number of HTTP requests that are needed, but increases the size of the HTML document. On the other hand, if the JavaScript and CSS are in external files cached by the browser, the size of the HTML document is reduced without increasing the number of HTTP requests.

The key factor, then, is the frequency with which external JavaScript and CSS components are cached relative to the number of HTML documents requested. This factor, although difficult to quantify, can be gauged using various metrics. If users on your site have multiple page views per session and many of your pages re-use the same scripts and stylesheets, there is a greater potential benefit from cached external files.

Many web sites fall in the middle of these metrics. For these sites, the best solution generally is to deploy the JavaScript and CSS as external files. The only exception where inlining is preferable is with home pages, such as Yahoo!'s front page and My Yahoo!. Home pages that have few (perhaps only one) page view per session may find that inlining JavaScript and CSS results in faster end-user response times.

For front pages that are typically the first of many page views, there are techniques that leverage the reduction of HTTP requests that inlining provides, as well as the caching benefits achieved through using external files. One such technique is to inline JavaScript and CSS in the front page, but dynamically download the external files after the page has finished loading. Subsequent pages would reference the external files that should already be in the browser's cache.

top | discuss this rule

Reduce DNS Lookups

tag: content

The Domain Name System (DNS) maps hostnames to IP addresses, just as phonebooks map people's names to their phone numbers. When you type www.yahoo.com into your browser, a DNS resolver contacted by the browser returns that server's IP address. DNS has a cost. It typically takes 20-120 milliseconds for DNS to lookup the IP address for a given hostname. The browser can't download anything from this hostname until the DNS lookup is completed.

DNS lookups are cached for better performance. This caching can occur on a special caching server, maintained by the user's ISP or local area network, but there is also caching that occurs on the individual user's computer. The DNS information remains in the operating system's DNS cache (the "DNS Client service" on Microsoft Windows). Most browsers have their own caches, separate from the operating system's cache. As long as the browser keeps a DNS record in its own cache, it doesn't bother the operating system with a request for the record.

Internet Explorer caches DNS lookups for 30 minutes by default, as specified by the `DnsCacheTimeout` registry setting. Firefox caches DNS lookups for 1 minute, controlled by the `network.dnsCacheExpiration` configuration setting. (Fasterfox changes this to 1 hour.)

When the client's DNS cache is empty (for both the browser and the operating system), the number of DNS lookups is equal to the number of unique hostnames in the web page. This includes the hostnames used in the page's URL, images, script files, stylesheets, Flash objects, etc. Reducing the number of unique hostnames reduces the number of DNS lookups.

Reducing the number of unique hostnames has the potential to reduce the amount of parallel downloading that takes place in the page. Avoiding DNS lookups cuts response times, but reducing parallel downloads may increase response times. My guideline is to split these components across at least two but no more than four hostnames. This results in a good compromise between reducing DNS lookups and allowing a high degree of parallel downloads.

top | discuss this rule

Minify JavaScript and CSS

tag: javascript, css

Minification is the practice of removing unnecessary characters from code to reduce its size thereby improving load times. When code is minified all comments are removed, as well as unneeded white space characters (space, newline, and tab). In the case of JavaScript, this improves response time performance because the size of the downloaded file is reduced. Two popular tools for minifying JavaScript code are JSMin and YUI Compressor. The YUI compressor can also minify CSS.

Obfuscation is an alternative optimization that can be applied to source code. It's more complex than minification and thus more likely to generate bugs as a result of the obfuscation step itself. In a survey of ten top U.S. web sites, minification achieved a 21% size reduction versus 25% for obfuscation. Although obfuscation has a higher size reduction, minifying JavaScript is less risky.

In addition to minifying external scripts and styles, inlined `<script>` and `<style>` blocks can and should also be minified. Even if you gzip your scripts and styles, minifying them will still reduce the size by 5% or more. As the use and size of JavaScript and CSS increases, so will the savings gained by minifying your code.

top | discuss this rule

Avoid Redirects

tag: content

Redirects are accomplished using the 301 and 302 status codes. Here's an example of the HTTP headers in a 301 response:

```
HTTP/1.1 301 Moved Permanently
Location: http://example.com/newuri
Content-Type: text/html
```

The browser automatically takes the user to the URL specified in the `Location` field. All the information necessary for a redirect is in the headers. The body of the response is typically empty. Despite their names, neither a 301 nor a 302 response is cached in practice unless additional headers, such as `Expires` or `Cache-Control`, indicate it should be. The meta refresh tag and JavaScript are other ways to direct users to a different URL, but if you must do a redirect, the preferred technique is to use the standard 3xx HTTP status codes, primarily to ensure the back button works correctly.

The main thing to remember is that redirects slow down the user experience. Inserting a redirect between the user and the HTML document delays everything in the page since nothing in the page can be rendered and no components can start being downloaded until the HTML document has arrived.

One of the most wasteful redirects happens frequently and web developers are generally not aware of it. It occurs when a trailing slash (/) is missing from a URL that should otherwise have one. For example, going to http://astrology.yahoo.com/astrology results in a 301 response containing a redirect to http://astrology.yahoo.com/astrology/ (notice the added trailing slash). This is fixed in Apache by using `Alias` or `mod_rewrite`, or the `DirectorySlash` directive if you're using Apache handlers.

Connecting an old web site to a new one is another common use for redirects. Others include connecting different parts of a website and directing the user based on certain conditions (type of browser, type of user account, etc.). Using a redirect to connect two web sites is simple and requires little additional coding. Although using redirects in these situations reduces the complexity for developers, it degrades the user experience. Alternatives for this use of redirects include using `Alias` and `mod_rewrite` if the two code paths are hosted on the same server. If a domain name change is the cause of using redirects, an alternative is to create a CNAME (a DNS record that creates an alias pointing from one domain name to another) in combination with `Alias` or `mod_rewrite`.

top | discuss this rule

Remove Duplicate Scripts

tag: javascript

It hurts performance to include the same JavaScript file twice in one page. This isn't as unusual as you might think. A review of the ten top U.S. web sites shows

It hurts performance to include the same JavaScript file twice in one page. This isn't as unusual as you might think. A review of the ten top U.S. web sites shows that two of them contain a duplicated script. Two main factors increase the odds of a script being duplicated in a single web page: team size and number of scripts. When it does happen, duplicate scripts hurt performance by creating unnecessary HTTP requests and wasted JavaScript execution.

Unnecessary HTTP requests happen in Internet Explorer, but not in Firefox. In Internet Explorer, if an external script is included twice and is not cacheable, it generates two HTTP requests during page loading. Even if the script is cacheable, extra HTTP requests occur when the user reloads the page.

In addition to generating wasteful HTTP requests, time is wasted evaluating the script multiple times. This redundant JavaScript execution happens in both Firefox and Internet Explorer, regardless of whether the script is cacheable.

One way to avoid accidentally including the same script twice is to implement a script management module in your templating system. The typical way to include a script is to use the SCRIPT tag in your HTML page.

```
<script type="text/javascript" src="menu_1.0.17.js"></script>
```

An alternative in PHP would be to create a function called `insertScript`.

```
<?php insertScript("menu.js") ?>
```

In addition to preventing the same script from being inserted multiple times, this function could handle other issues with scripts, such as dependency checking and adding version numbers to script filenames to support far future Expires headers.

top | discuss this rule

Configure ETags

tag: server

Entity tags (ETags) are a mechanism that web servers and browsers use to determine whether the component in the browser's cache matches the one on the origin server. (An "entity" is another word a "component": images, scripts, stylesheets, etc.) ETags were added to provide a mechanism for validating entities that is more flexible than the last-modified date. An ETag is a string that uniquely identifies a specific version of a component. The only format constraints are that the string be quoted. The origin server specifies the component's ETag using the `ETag` response header.

```
HTTP/1.1 200 OK
Last-Modified: Tue, 12 Dec 2006 03:03:59 GMT
ETag: "10c24bc-4ab-457e1c1f"
Content-Length: 12195
```

Later, if the browser has to validate a component, it uses the `If-None-Match` header to pass the ETag back to the origin server. If the ETags match, a 304 status code is returned reducing the response by 12195 bytes for this example.

```
GET /i/yahoo.gif HTTP/1.1
Host: us.yimg.com
If-Modified-Since: Tue, 12 Dec 2006 03:03:59 GMT
If-None-Match: "10c24bc-4ab-457e1c1f"
HTTP/1.1 304 Not Modified
```

The problem with ETags is that they typically are constructed using attributes that make them unique to a specific server hosting a site. ETags won't match when a browser gets the original component from one server and later tries to validate that component on a different server, a situation that is all too common on Web sites that use a cluster of servers to handle requests. By default, both Apache and IIS embed data in the ETag that dramatically reduces the odds of the validity test succeeding on web sites with multiple servers.

The ETag format for Apache 1.3 and 2.x is `inode-size-timestamp`. Although a given file may reside in the same directory across multiple servers, and have the same file size, permissions, timestamp, etc., its inode is different from one server to the next.

IIS 5.0 and 6.0 have a similar issue with ETags. The format for ETags on IIS is `Filetimestamp:ChangeNumber`. A `ChangeNumber` is a counter used to track configuration changes to IIS. It's unlikely that the `ChangeNumber` is the same across all IIS servers behind a web site.

The end result is ETags generated by Apache and IIS for the exact same component won't match from one server to another. If the ETags don't match, the user doesn't receive the small, fast 304 response that ETags were designed for; instead, they'll get a normal 200 response along with all the data for the component. If you host your web site on just one server, this isn't a problem. But if you have multiple servers hosting your web site, and you're using Apache or IIS with the default ETag configuration, your users are getting slower pages, your servers have a higher load, you're consuming greater bandwidth, and proxies aren't caching your content efficiently. Even if your components have a far future `Expires` header, a conditional GET request is still made whenever the user hits Reload or Refresh.

If you're not taking advantage of the flexible validation model that ETags provide, it's better to just remove the ETag altogether. The `Last-Modified` header validates based on the component's timestamp. And removing the ETag reduces the size of the HTTP headers in both the response and subsequent requests. This Microsoft Support article describes how to remove ETags. In Apache, this is done by simply adding the following line to your Apache configuration file:

```
FileETag none
```

top | discuss this rule

Make Ajax Cacheable

tag: content

One of the cited benefits of Ajax is that it provides instantaneous feedback to the user because it requests information asynchronously from the backend web server. However, using Ajax is no guarantee that the user won't be twiddling his thumbs waiting for those asynchronous JavaScript and XML responses to return. In many applications, whether or not the user is kept waiting depends on how Ajax is used. For example, in a web-based email client the user will be kept waiting for

the results of an Ajax request to find all the email messages that match their search criteria. It's important to remember that "asynchronous" does not imply

the results of an Ajax request to find all the email messages that match their search criteria. It's important to remember that "asynchronous" does not imply "instantaneous".

To improve performance, it's important to optimize these Ajax responses. The most important way to improve the performance of Ajax is to make the responses cacheable, as discussed in Add an Expires or a Cache-Control Header. Some of the other rules also apply to Ajax:

- Avoid Redirects
- Configure ETags

Let's look at an example. A Web 2.0 email client might use Ajax to download the user's address book for autocompletion. If the user hasn't modified her address book since the last time she used the email web app, the previous address book response could be read from cache if that Ajax response was made cacheable with a future Expires or Cache-Control header. The browser must be informed when to use a previously cached address book response versus requesting a new one. This could be done by adding a timestamp to the address book Ajax URL indicating the last time the user modified her address book, for example, &t=1190241612. If the address book hasn't been modified since the last download, the timestamp will be the same and the address book will be read from the browser's cache eliminating an extra HTTP roundtrip. If the user has modified her address book, the timestamp ensures the new URL doesn't match the cached response, and the browser will request the updated address book entries.

Even though your Ajax responses are created dynamically, and might only be applicable to a single user, they can still be cached. Doing so will make your Web 2.0 apps faster.

top | discuss this rule

Flush the Buffer Early

tag: server

When users request a page, it can take anywhere from 200 to 500ms for the backend server to stitch together the HTML page. During this time, the browser is idle as it waits for the data to arrive. In PHP you have the function flush(). It allows you to send your partially ready HTML response to the browser so that the browser can start fetching components while your backend is busy with the rest of the HTML page. The benefit is mainly seen on busy backends or light frontends.

A good place to consider flushing is right after the HEAD because the HTML for the head is usually easier to produce and it allows you to include any CSS and JavaScript files for the browser to start fetching in parallel while the backend is still processing.

Example:

```
    ... <!-- css, js -->
</head>
<?php flush(); ?>
<body>
    ... <!-- content -->
```

Yahoo! search pioneered research and real user testing to prove the benefits of using this technique.

top

Use GET for AJAX Requests

tag: server

The Yahoo! Mail team found that when using XMLHttpRequest, POST is implemented in the browsers as a two-step process: sending the headers first, then sending data. So it's best to use GET, which only takes one TCP packet to send (unless you have a lot of cookies). The maximum URL length in IE is 2K, so if you send more than 2K data you might not be able to use GET.

An interesting side affect is that POST without actually posting any data behaves like GET. Based on the HTTP specs, GET is meant for retrieving information, so it makes sense (semantically) to use GET when you're only requesting data, as opposed to sending data to be stored server-side.

top

Post-load Components

tag: content

You can take a closer look at your page and ask yourself: "What's absolutely required in order to render the page initially?". The rest of the content and components can wait.

JavaScript is an ideal candidate for splitting before and after the onload event. For example if you have JavaScript code and libraries that do drag and drop and animations, those can wait, because dragging elements on the page comes after the initial rendering. Other places to look for candidates for post-loading include hidden content (content that appears after a user action) and images below the fold.

Tools to help you out in your effort: YUI Image Loader allows you to delay images below the fold and the YUI Get utility is an easy way to include JS and CSS on the fly. For an example in the wild take a look at Yahoo! Home Page with Firebug's Net Panel turned on.

It's good when the performance goals are inline with other web development best practices. In this case, the idea of progressive enhancement tells us that

JavaScript, when supported, can improve the user experience but you have to make sure the page works even without JavaScript. So after you've made sure the

JavaScript, when supported, can improve the user experience but you have to make sure the page works even without JavaScript. So after you've made sure the page works fine, you can enhance it with some post-loaded scripts that give you more bells and whistles such as drag and drop and animations.

top

Preload Components

tag: content

Preload may look like the opposite of post-load, but it actually has a different goal. By preloading components you can take advantage of the time the browser is idle and request components (like images, styles and scripts) you'll need in the future. This way when the user visits the next page, you could have most of the components already in the cache and your page will load much faster for the user.

There are actually several types of preloading:

- *Unconditional* preload - as soon as onload fires, you go ahead and fetch some extra components. Check google.com for an example of how a sprite image is requested onload. This sprite image is not needed on the google.com homepage, but it is needed on the consecutive search result page.
- *Conditional* preload - based on a user action you make an educated guess where the user is headed next and preload accordingly. On search.yahoo.com you can see how some extra components are requested after you start typing in the input box.
- *Anticipated* preload - preload in advance before launching a redesign. It often happens after a redesign that you hear: "The new site is cool, but it's slower than before". Part of the problem could be that the users were visiting your old site with a full cache, but the new one is always an empty cache experience. You can mitigate this side effect by preloading some components before you even launched the redesign. Your old site can use the time the browser is idle and request images and scripts that will be used by the new site

top

Reduce the Number of DOM Elements

tag: content

A complex page means more bytes to download and it also means slower DOM access in JavaScript. It makes a difference if you loop through 500 or 5000 DOM elements on the page when you want to add an event handler for example.

A high number of DOM elements can be a symptom that there's something that should be improved with the markup of the page without necessarily removing content. Are you using nested tables for layout purposes? Are you throwing in more `<div>`s only to fix layout issues? Maybe there's a better and more semantically correct way to do your markup.

A great help with layouts are the YUI CSS utilities: grids.css can help you with the overall layout, fonts.css and reset.css can help you strip away the browser's defaults formatting. This is a chance to start fresh and think about your markup, for example use `<div>`s only when it makes sense semantically, and not because it renders a new line.

The number of DOM elements is easy to test, just type in Firebug's console:
`document.getElementsByTagName('*').length`

And how many DOM elements are too many? Check other similar pages that have good markup. For example the Yahoo! Home Page is a pretty busy page and still under 700 elements (HTML tags).

top

Split Components Across Domains

tag: content

Splitting components allows you to maximize parallel downloads. Make sure you're using not more than 2-4 domains because of the DNS lookup penalty. For example, you can host your HTML and dynamic content on `www.example.org` and split static components between `static1.example.org` and `static2.example.org`

For more information check "Maximizing Parallel Downloads in the Carpool Lane" by Tenni Theurer and Patty Chi.

top

Minimize the Number of iframes

tag: content

Iframes allow an HTML document to be inserted in the parent document. It's important to understand how iframes work so they can be used effectively.

`<iframe>` pros:

- Helps with slow third-party content like badges and ads
- Security sandbox
- Download scripts in parallel

`<iframe>` cons:

- Costly even if blank
- Blocks page onload
- Non-semantic

top

No 404s

tag: content

HTTP requests are expensive so making an HTTP request and getting a useless response (i.e. 404 Not Found) is totally unnecessary and will slow down the user experience without any benefit.

Some sites have helpful 404s "Did you mean X?", which is great for the user experience but also wastes server resources (like database, etc). Particularly bad is when the link to an external JavaScript is wrong and the result is a 404. First, this download will block parallel downloads. Next the browser may try to parse the 404 response body as if it were JavaScript code, trying to find something usable in it.

Reduce Cookie Size

tag: cookie

HTTP cookies are used for a variety of reasons such as authentication and personalization. Information about cookies is exchanged in the HTTP headers between web servers and browsers. It's important to keep the size of cookies as low as possible to minimize the impact on the user's response time.

For more information check "When the Cookie Crumbles" by Tenni Theurer and Patty Chi. The take-home of this research:

- Eliminate unnecessary cookies
- Keep cookie sizes as low as possible to minimize the impact on the user response time
- Be mindful of setting cookies at the appropriate domain level so other sub-domains are not affected
- Set an Expires date appropriately. An earlier Expires date or none removes the cookie sooner, improving the user response time

Use Cookie-free Domains for Components

tag: cookie

When the browser makes a request for a static image and sends cookies together with the request, the server doesn't have any use for those cookies. So they only create network traffic for no good reason. You should make sure static components are requested with cookie-free requests. Create a subdomain and host all your static components there.

If your domain is `www.example.org`, you can host your static components on `static.example.org`. However, if you've already set cookies on the top-level domain `example.org` as opposed to `www.example.org`, then all the requests to `static.example.org` will include those cookies. In this case, you can buy a whole new domain, host your static components there, and keep this domain cookie-free. Yahoo! uses `yimg.com`, YouTube uses `ytimg.com`, Amazon uses `images-amazon.com` and so on.

Another benefit of hosting static components on a cookie-free domain is that some proxies might refuse to cache the components that are requested with cookies. On a related note, if you wonder if you should use example.org or www.example.org for your home page, consider the cookie impact. Omitting www leaves you no choice but to write cookies to `*.example.org`, so for performance reasons it's best to use the www subdomain and write the cookies to that subdomain.

Minimize DOM Access

tag: javascript

Accessing DOM elements with JavaScript is slow so in order to have a more responsive page, you should:

- Cache references to accessed elements
- Update nodes "offline" and then add them to the tree
- Avoid fixing layout with JavaScript

For more information check the YUI theatre's "High Performance Ajax Applications" by Julien Lecomte.

Develop Smart Event Handlers

tag: javascript

Sometimes pages feel less responsive because of too many event handlers attached to different elements of the DOM tree which are then executed too often. That's why using *event delegation* is a good approach. If you have 10 buttons inside a `div`, attach only one event handler to the div wrapper, instead of one handler for each button. Events bubble up so you'll be able to catch the event and figure out which button it originated from.

You also don't need to wait for the onload event in order to start doing something with the DOM tree. Often all you need is the element you want to access to be available in the tree. You don't have to wait for all images to be downloaded. `DOMContentLoaded` is the event you might consider using instead of onload, but until it's available in all browsers, you can use the YUI Event utility, which has an `onAvailable` method.

For more information check the YUI theatre's "High Performance Ajax Applications" by Julien Lecomte.

For more information check the YUI theatre's "High Performance Ajax Applications" by Julien Lecomte.

Choose <link> over @import

tag: css

One of the previous best practices states that CSS should be at the top in order to allow for progressive rendering.

In IE `@import` behaves the same as using `<link>` at the bottom of the page, so it's best not to use it.

Avoid Filters

tag: css

The IE-proprietary `AlphaImageLoader` filter aims to fix a problem with semi-transparent true color PNGs in IE versions < 7. The problem with this filter is that it blocks rendering and freezes the browser while the image is being downloaded. It also increases memory consumption and is applied per element, not per image, so the problem is multiplied.

The best approach is to avoid `AlphaImageLoader` completely and use gracefully degrading PNG8 instead, which are fine in IE. If you absolutely need `AlphaImageLoader`, use the underscore hack `_filter` as to not penalize your IE7+ users.

Optimize Images

tag: images

After a designer is done with creating the images for your web page, there are still some things you can try before you FTP those images to your web server.

- You can check the GIFs and see if they are using a palette size corresponding to the number of colors in the image. Using imagemagick it's easy to check using
  ```
  identify -verbose image.gif
  ```
  When you see an image using 4 colors and a 256 color "slots" in the palette, there is room for improvement.
- Try converting GIFs to PNGs and see if there is a saving. More often than not, there is. Developers often hesitate to use PNGs due to the limited support in browsers, but this is now a thing of the past. The only real problem is alpha-transparency in true color PNGs, but then again, GIFs are not true color and don't support variable transparency either. So anything a GIF can do, a palette PNG (PNG8) can do too (except for animations). This simple imagemagick command results in totally safe-to-use PNGs:
  ```
  convert image.gif image.png
  ```
  "All we are saying is: Give PiNG a Chance!"
- Run pngcrush (or any other PNG optimizer tool) on all your PNGs. Example:
  ```
  pngcrush image.png -rem alla -reduce -brute result.png
  ```
- Run jpegtran on all your JPEGs. This tool does lossless JPEG operations such as rotation and can also be used to optimize and remove comments and other useless information (such as EXIF information) from your images.
  ```
  jpegtran -copy none -optimize -perfect src.jpg dest.jpg
  ```

Optimize CSS Sprites

tag: images

- Arranging the images in the sprite horizontally as opposed to vertically usually results in a smaller file size.
- Combining similar colors in a sprite helps you keep the color count low, ideally under 256 colors so to fit in a PNG8.
- "Be mobile-friendly" and don't leave big gaps between the images in a sprite. This doesn't affect the file size as much but requires less memory for the user agent to decompress the image into a pixel map. 100x100 image is 10 thousand pixels, where 1000x1000 is 1 million pixels

Don't Scale Images in HTML

tag: images

Don't use a bigger image than you need just because you can set the width and height in HTML. If you need
```
<img width="100" height="100" src="mycat.jpg" alt="My Cat" />
```
then your image (mycat.jpg) should be 100x100px rather than a scaled down 500x500px image.

Make favicon.ico Small and Cacheable

tag: images

The favicon.ico is an image that stays in the root of your server. It's a necessary evil because even if you don't care about it the browser will still request it, so it's better not to respond with a 404 Not Found. Also since it's on the same server, cookies are sent every time it's requested. This image also interferes with the

better not to respond with a 404 Not Found. Also since it's on the same server, cookies are sent every time it's requested. This image also interferes with the download sequence, for example in IE when you request extra components in the onload, the favicon will be downloaded before these extra components.

So to mitigate the drawbacks of having a favicon.ico make sure:

- It's small, preferably under 1K.
- Set Expires header with what you feel comfortable (since you cannot rename it if you decide to change it). You can probably safely set the Expires header a few months in the future. You can check the last modified date of your current favicon.ico to make an informed decision.

Imagemagick can help you create small favicons

top

Keep Components under 25K

tag: mobile

This restriction is related to the fact that iPhone won't cache components bigger than 25K. Note that this is the *uncompressed* size. This is where minification is important because gzip alone may not be sufficient.

For more information check "Performance Research, Part 5: iPhone Cacheability - Making it Stick" by Wayne Shea and Tenni Theurer.

top

Pack Components into a Multipart Document

tag: mobile

Packing components into a multipart document is like an email with attachments, it helps you fetch several components with one HTTP request (remember: HTTP requests are expensive). When you use this technique, first check if the user agent supports it (iPhone does not).

Avoid Empty Image src

tag: server

Image with empty string **src** attribute occurs more than one will expect. It appears in two form:

1. straight HTML
       <img src="">

2. JavaScript
       var img = new Image();
       img.src = "";

Both forms cause the same effect: browser makes another request to your server.

- **Internet Explorer** makes a request to the directory in which the page is located.
- **Safari and Chrome** make a request to the actual page itself.
- **Firefox** 3 and earlier versions behave the same as Safari and Chrome, but version 3.5 addressed this issue[bug 444931] and no longer sends a request.
- **Opera** does not do anything when an empty image src is encountered.

Why is this behavior bad?

1. Cripple your servers by sending a large amount of unexpected traffic, especially for pages that get millions of page views per day.
2. Waste server computing cycles generating a page that will never be viewed.
3. Possibly corrupt user data. If you are tracking state in the request, either by cookies or in another way, you have the possibility of destroying data. Even though the image request does not return an image, all of the headers are read and accepted by the browser, including all cookies. While the rest of the response is thrown away, the damage may already be done.

The root cause of this behavior is the way that URI resolution is performed in browsers. This behavior is defined in RFC 3986 - Uniform Resource Identifiers. When an empty string is encountered as a URI, it is considered a relative URI and is resolved according to the algorithm defined in section 5.2. This specific example, an empty string, is listed in section 5.4. Firefox, Safari, and Chrome are all resolving an empty string correctly per the specification, while Internet Explorer is resolving it incorrectly, apparently in line with an earlier version of the specification, RFC 2396 - Uniform Resource Identifiers (this was obsoleted by RFC 3986). So technically, the browsers are doing what they are supposed to do to resolve relative URIs. The problem is that in this context, the empty string is clearly unintentional.

HTML5 adds to the description of the tag's src attribute to instruct browsers not to make an additional request in section 4.8.2:

The src attribute must be present, and must contain a valid URL referencing a non-interactive, optionally animated, image resource that is neither paged nor scripted. If the base URI of the element is the same as the document's address, then the src attribute's value must not be the empty string.

Hopefully, browsers will not have this problem in the future. Unfortunately, there is no such clause for <script src=""> and <link href="">. Maybe there is still time to make that adjustment to ensure browsers don't accidentally implement this behavior.

This rule was inspired by Yahoo!'s JavaScript guru Nicolas C. Zakas. For more information check out his article "Empty image src can destroy your site".

top