Optimize browser rendering

Once resources have been downloaded to the client, the browser still needs to load, interpret, and render HTML, CSS, and JavaScript
your code and pages in ways that exploit the characteristics of current browsers, you can enhance performance on the client side.

Use efficient CSS selectors

Overview

Avoiding inefficient key selectors that match large numbers of elements can speed up page rendering.

Details

As the browser parses HTML, it constructs an internal document tree representing all the elements to be displayed. It then matches
various stylesheets, according to the standard CSS cascade, inheritance, and ordering rules. In Mozilla's implementation (and probal
element, the CSS engine searches through style rules to find a match. The engine evaluates each rule from right to left, starting from
the "key") and moving through each selector until it finds a match or discards the rule. (The "selector" is the document element to w

According to this system, the fewer rules the engine has to evaluate the better. So, of course, removing unused CSS is an important
performance. After that, for pages that contain large numbers of elements and/or large numbers of CSS rules, optimizing the definiti
can enhance performance as well. The key to optimizing rules lies in defining rules that are as specific as possible and that avoid unn
the style engine to quickly find matches without spending time evaluating rules that don't apply.

The following categories of rules are considered to be inefficient:

Rules with descendant selectors

For example:

Rules with the universal selector as the key

```
body * {...}
.hide-scrollbars * {...}
```

Rules with a tag selector as the key

```
ul li a {...}
#footer h3 {...}
* html #atticPromo ul li a {...]
```

Descendant selectors are inefficient because, for each element that matches the key, the browser must also traverse up the DOM
ancestor element until it finds a match or reaches the root element. The less specific the key, the greater the number of nodes th

Rules with child or adjacent selectors

For example:

Rules with the universal selector as the key

```
body > * {...}
```

```
.hide-scrollbars > * {...}
```

Rules with a tag selector as the key

```
ul > li > a {...}
#footer > h3 {...}
```

Child and adjacent selectors are inefficient because, for each matching element, the browser has to evaluate another node. It bec
each child selector in the rule. Again, the less specific the key, the greater the number of nodes that need to be evaluated. Howev
still preferable to descendant selectors in terms of performance.

Rules with overly qualified selectors

For example:

```
ul#top_blue_nav {...}
form#UserLogin {...}
```

ID selectors are unique by definition. Including tag or class qualifiers just adds redundant information that needs to be evaluatec

Rules that apply the `:hover` pseudo-selector to non-link elements

For example:

```
h3:hover {...}
.foo:hover {...}
#foo:hover {...}
div.faa :hover {...}
```

The `:hover` pseudo-selector on non-anchor elements is known to make IE7 and IE8 slow in some cases*. When a strict doctype
ignore `:hover` on any element other than anchors. When a strict doctype is used, `:hover` on non-anchors may cause performa

* See a bug report at http://connect.microsoft.com/IE/feedback/ViewFeedback.aspx?FeedbackID=391387.

Recommendations

Avoid a universal key selector.

Allow elements to inherit from ancestors, or use a class to apply a style to multiple elements.

Make your rules as specific as possible.

Prefer class and ID selectors over tag selectors.

Remove redundant qualifiers.

These qualifiers are redundant:

- ID selectors qualified by class and/or tag selectors
- Class selectors qualified by tag selectors (when a class is only used for one tag, which is a good design practice anyway).

Avoid using descendant selectors, especially those that specify redundant ancestors.

For example, the rule `body ul li a {...}` specifies a redundant `body` selector, since all elements are descendants of the `boc`

Use class selectors instead of descendant selectors.

For example, if you need two different styles for an ordered list item and an ordered list item, instead of using two rules:

```
ul li {color: blue;}
ol li {color: red;}
```

You could encode the styles into two class names and use those in your rules; e.g:

```
.unordered-list-item {color: blue;}
.ordered-list-item {color: red;}
```

If you must use descendant selectors, prefer child selectors, which at least only require evaluation of one additional node, not all
an ancestor.

Avoid the `:hover` pseudo-selector for non-link elements for IE clients.

If you use `:hover` on non-anchor elements, test the page in IE7 and IE8 to be sure your page is usable. If you find that `:hover` consider conditionally using a JavaScript `onmouseover` event handler for IE clients.

Additional resources

- For more details on efficient CSS rules with Mozilla, see Writing Efficient CSS for Use in the Mozilla UI.
- For complete information on CSS, see the Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. For information on C Chapter 5.

Back to top

Avoid CSS expressions

Overview

CSS expressions degrade rendering performance; replacing them with alternatives will improve browser rendering for IE users.

Note: This best practices in this section apply only to Internet Explorer 5 through 7, which support CSS expressions. CSS expression Explorer 8, and not supported by other browsers.

Details

Internet Explorer 5 introduced CSS expressions, or "dynamic properties", as a means of dynamically changing document properties They consist of JavaScript expressions embedded as the values of CSS properties in CSS declarations. For the most part, they are use

- To emulate standard CSS properties supported by other browsers but not yet implemented by IE.
- To provide dynamic styling and advanced event handling in a more compact and convenient way than writing full-blown JavaSc

Unfortunately, the performance penalty imposed by CSS expressions is considerable, as the browser reevaluates each expression w triggered, such as a window resize, a mouse movement and so on. The poor performance of CSS expressions is one of the reasons t 8. If you have used CSS expressions in your pages, you should make every effort to remove them and use other methods to achieve

Recommendations

Use standard CSS properties if possible.

IE 8 is fully CSS-standards-compliant; it supports CSS expressions only if run in "compatibility" mode, but it does not support the do not need to maintain backwards compatibility with older versions of IE, you should convert any instances of expressions use properties to their CSS standard counterparts. For a complete list of CSS properties and IE versions that support them, see the M you do need to support older versions of IE in which the desired CSS properties are not available, use JavaScript to achieve the e

Use JavaScript to script styles.

If you are using CSS expressions for dynamic styling, it makes sense to rewrite them as pure JavaScript to both improve performa of supporting the same functionality in other browsers at the same time. In this example given on the MSDN page on Dynamic P used to center an HTML block whose dimensions can change at runtime, and to re-center that block every time the window is re

```
<div id="oDiv" style="background-color: #CFCFCF; position: absolute;
left:expression(document.body.clientWidth/2-oDiv.offsetWidth/2);
 top:expression(document.body.clientHeight/2-oDiv.offsetHeight/2)">Example DIV</div>
```

Here's an equivalent example using JavaScript and standard CSS:

```
<style>
  #oDiv { position: absolute; background-color: #CFCFCF;}
</style>

<script type="text/javascript">
 // Check for browser support of event handling capability
  if (window.addEventListener) {
  window.addEventListener("load", centerDiv, false);
 window.addEventListener("resize", centerDiv, false);
  } else if (window.attachEvent) {
  window.attachEvent("onload", centerDiv);
  window.attachEvent("onresize", centerDiv);
  } else {
```

```
  window.onload = centerDiv;
  window.resize = centerDiv;
  }

  function centerDiv() {
  var myDiv = document.getElementById("oDiv");
  var myBody = document.body;
  var bodyWidth = myBody.offsetWidth;

  //Needed for Firefox, which doesn't support offsetHeight
  var bodyHeight;
 if (myBody.scrollHeight)
 bodyHeight = myBody.scrollHeight;
 else bodyHeight = myBody.offsetHeight;

  var divWidth = myDiv.offsetWidth;

  if (myDiv.scrollHeight)
   var divHeight = myDiv.scrollHeight;
   else var divHeight = myDiv.offsetHeight;

 myDiv.style.top = (bodyHeight - divHeight) / 2;
  myDiv.style.left = (bodyWidth - divWidth) / 2;
  }

</script>
```

If you are using CSS expressions to emulate CSS properties that aren't available in earlier versions of IE, you should provide JavaS
with a version test to disable it for browsers that do support CSS. For example, the `max-width` property, which forces text to w
number of pixels, was not supported until IE 7. As a workaround, this CSS expression provides that functionality for IE 5 and 6:

```
p { width: expression( document.body.clientWidth > 600 ? "600px" : "auto" ); }
```

To replace the CSS expression with equivalent JavaScript for the IE versions that don't support this property, you could use some

```
<style>
  p { max-width: 300px; }
</style>

<script type="text/javascript">

  if ((navigator.appName == "Microsoft Internet Explorer") && (parseInt(navigator.appVersion)
  window.attachEvent("onresize", setMaxWidth);

  function setMaxWidth() {
  var paragraphs = document.getElementsByTagName("p");
  for ( var i = 0; i < paragraphs.length; i++ )
  paragraphs[i].style.width = ( document.body.clientWidth > 300 ? "300px" : "auto" );

</script>
```

Back to top

Put CSS in the document head

Overview

Moving inline style blocks and `<link>` elements from the document body to the document head improves rendering performance

Details

Specifying external stylesheets and inline style blocks in the body of an HTML document can negatively affect the browser's renderi
block rendering a web page until all external stylesheets have been downloaded. Inline style blocks (specified with the `<style>` tag
shifting of content. Therefore, it's important to put references to external stylesheets, as well as inline style blocks, in the head of the

stylesheets are downloaded and parsed first, you can allow the browser to progressively render the page.

Recommendations

- As required by the HTML 4.01 Specification (section 12.3), always put external stylesheets in the `<head>` section using the in the `@import`. Also make sure that you specify the stylesheets in the correct order with respect to scripts.
- Put `<style>` blocks in the `<head>` section.

Back to top

Specify image dimensions

Overview

Specifying a width and height for all images allows for faster rendering by eliminating the need for unnecessary reflows and repaints.

Details

When the browser lays out the page, it needs to be able to flow around replaceable elements such as images. It can begin to render downloaded, provided that it knows the dimensions to wrap non-replaceable elements around. If no dimensions are specified in the the dimensions specified don't match those of the actual images, the browser will require a reflow and repaint once the images are reflows, specify the width and height of all images, either in the HTML `<img>` tag, or in CSS.

Recommendations

Specify dimensions that match those of the images themselves.
Don't use width and height specifications to scale images on the fly. If an image file is actually 60 x 60 pixels, don't set the dimensi CSS. If the image needs to be smaller, scale it in an image editor and set its dimensions to match (see Optimize images for details.

Be sure to specify dimensions on the image element or block-level parent
Be sure to set the dimensions on the `<img>` element itself, or a block-level parent. If the parent is not block-level, the dimensions dimensions on an ancestor that is not an immediate parent.

Back to top

Specify a character set

Overview

Specifying a character set in the HTTP response headers of your HTML documents allows the browser to begin parsing HTML and ex

Details

HTML documents are sent over the Internet as a sequence of bytes accompanied by character encoding information. Character enc in the HTTP response headers sent with the document, or in the HTML markup of the document itself. The browser uses the charact convert the stream of bytes into characters that it renders on-screen. Because a browser cannot correctly render a page without kn page's characters, most browsers buffer a certain number of bytes before executing any JavaScript or drawing the page, while they s information in the input. (A notable exception is Internet Explorer versions 6, 7, and 8.)

Browsers differ with the respect to the number of bytes buffered and the default encoding assumed if no character set is found. Hov buffered the requisite number of bytes and begun to render the page, if they encounter a character set specification that doesn't m to reparse the input and redraw the page. Sometimes, they may even have to rerequest resources, if the mismatch affects the URLs

To avoid these delays, you should always specify the character encoding in the HTTP response headers. Note that, while it is possibl using a meta http-equiv tag, doing so **disables the lookahead downloader** in Internet Explorer 8. Disabling the lookahead downlo increase the amount of time it takes to load your page. Microsoft notes: "we continue to strongly recommend that web developers HTTP Content-Type response header, as this ensures that the performance benefit of the Lookahead Downloader is realized".

Recommendations

Always specify a content type.

Before browsers can begin to check for a character set, they must first determine the content type of the document being proces
the HTTP header or the HTML meta tag, they will attempt to "sniff" the type, using various algorithms. This process can cause ad
representing a security vulnerability. For both performance and security reasons, you should always specify a content type for al
text/html).

Be sure to specify the correct character encoding.

It's important that the character set you specify in an HTTP header or HTML meta tag match the character encoding actually use
documents. If you specify a charset parameter in both the HTTP header and HTML meta tag, make sure they match each other. I
incorrect or mismatched encoding, it will render the page incorrectly and/or incur additional delays while it redraws the page. Fo
character sets, see Section 5.2, Character Encodings in the HTML 4.01 Specification.

Additional resources

For details on browser behavior with respect to the presence/absence of content-type and charset specifications, see:

- Page Speed wiki
- Browser Performance Issues with Charsets
- Performance Implications of "charset"

Back to top

Last updated March 28, 2012.