Deep Dive into Browser Performance

Ilia Alshanetsky

@iliaa
Me, Myself and I

PHP Core Developer

Author of
Guide to PHP Security

CIO at Centah Inc.
Why Browser Performance Matters?

PHPConference.com - Total Page Load - 3.47s
- Browser rendering 3.342s
- Back-end Processing 0.128s

PHP.net - Total Page Load - 1.45s
- Browser rendering 1.347s
- Back-end Processing 0.103s

Github - Total Page Load - 1.43s
- Browser rendering 1.43s
- Back-end Processing 0.058.6
What Takes All This Time?

1. DNS
2. HTTP + SSL Negotiation
3. JavaScript Processing
4. CSS Rendering
5. Image Rendering
6. DOM Rendering
DNS

DNS may take up-to 20% of 1st page load!
DNS Prefetching

Instruct the browser to pre-resolve (sub)domain from which you intend to retrieve resources.

<link rel="dns-prefetch" href="//mydomain.tld" />

Supported By:

Firefox 3.5+
Chrome
Safari 5+
IE 9+
DNS Based Optimizations

1. Use Embedded images via data:image

2. Defer loading of external resources

3. Avoid multi-domain CDNs

4. Sub-domains still need to be resolved
Use Sprites!

40 images = 1 Request
17.5% reduction in size (74kb saving)

http://www.spritebox.net/
http://spritepad.wearekiss.com/
Profile Page Loading

• Use Your Browser
  ✴ Developer Tools or Equivalent

• Do Remote Tests
  ✴ http://www.webpagetest.org/
  ✴ https://developers.google.com/speed/pagespeed/
  ✴ https://www.modern.ie/en-us

• Actual User Profiling
  ✴ http://www.lognormal.com/boomerang/doc/
  ✴ Use Web-Timing API directly
Compression For The Win!

Use gzip compression
965.8 KB total in compressible text, savings = 695.2 KB

Compress Images
171.6 KB total in images, savings = 51.8 KB

1,394 KB
59 requests,
4.63 seconds to load

Compression Reduces data-size by >50% and makes page load in 2.1 seconds!
<table>
<thead>
<tr>
<th></th>
<th>Load Time</th>
<th>First Byte</th>
<th>Start Render</th>
<th>DOM Elements</th>
<th>Document Complete</th>
<th>Fully Loaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>First View</td>
<td>3.467s</td>
<td>0.314s</td>
<td>2.280s</td>
<td>1985</td>
<td>3.467s</td>
<td>3.547s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>711 KB</td>
<td>712 KB</td>
</tr>
<tr>
<td>Repeat View</td>
<td>1.641s</td>
<td>0.630s</td>
<td>1.262s</td>
<td>1985</td>
<td>1.641s</td>
<td>1.641s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>145 KB</td>
<td>145 KB</td>
</tr>
</tbody>
</table>
Cache, Cache, Cache

Set **max-age** or **expires** headers

Value should be at least **30 days**

To prevent stale content, **use unique file names** on new deployments for changed files.

Your goal is that 2nd page load only queries the server for the dynamic content!
JavaScript is loaded synchronously, so compact your files into a single compressed file!
JavaScript

Combination & minifying of JS files is best achieved with:

- Closure Compiler
  - http://goo.gl/8MVOIJ

- YUI Compressor
  - http://refresh-sf.com/yui/
  - http://yui.github.io/yuicompressor/

- PHP Based
  - https://github.com/tedious/JShrink
JavaScript

Don’t over-do combining of JS Files!

- Unnecessary data loading
- Decompression Overhead
- Extra JS Compilation
Micro-Case Study: SlashDot.org

One “BIG” JavaScript file

71kb compressed, 251kb actual size

199ms to receive

37ms to process

21.3% of total page load, 16% of total page size

< 10% of loaded JS code is executed
JavaScript

Only load up-front what you absolutely need

Defer loading of everything else via RequireJS

```html
<head>
  <script src="scripts/require.js"></script>
</head>

require.config({
  baseUrl: 'js/lib',
  paths: { jquery: 'jquery-1.11.1' }
});

define(['lib/jquery'], function ($) {...});
```

http://requirejs.org/
If you can’t win, cheat!

$(document).ready(function() {
  setTimeout(function() {
    $.get( "your-file.js" );
  }, 2000);
});
General JS Tips

1. Avoid Xpath, reference/search by ID

2. Setup events pre-load as opposed to post-load
   onkeyup="js_function()" vs $("input").each(function() {});

3. For Grids only load the data to be displayed

4. innerHTML is not always faster than DOM
   http://jsperf.com/dom-vs-innerhtml/37
General JS Tips

• Most browsers leak memory with JS, avoid the common culprits:
  ✦ Use closures
  ✦ Avoid passing objects (can result in circular references)
  ✦ Avoid global variables
General JS Tips

- Help browser to make use of multiple CPUs by using iframes to embed complex components such as grids.

YO DAWG WE HEARD YOU LIKE IFRAMES

SO WE PUT AN IFRAME IN YOUR IFRAME SO YOU CAN LOAD PAGES WHILE YOU LOAD PAGES WHILE YOU LOAD PAGES
* Minimize
* Combine
* Compress

Don’t fear inlined (<style>) CSS
Avoid Repaints & Reflows

- Changes to DOM nodes
- Hiding DOM nodes
- Actions that extend the page (causes scroll)
- Changes to colour, background and outline properties
Merge Style Changes

// slowest
el.style.left  = "10px";
el.style.top   = "10px";

// getting better
el.className += " top-left-class";

// best
el.style.cssText += "; left: 10px; top: 10px;";
Peekaboo Trick

```
var me = $('#el');
me.hide();

// make various changes to DOM/Content

me.show();
```
Dolly Trick

```javascript
var $dolly = el.clone();

// make changes to the copy

el.replaceWith($dolly);
```
Don’t Abuse Computed Styles

```javascript
// nein, nein, nein!!!
for (var i = 0; i < 100; i++) {
  el[i].style.left = el.offsetTopLeft + "10px";
  el[i].style.top = el.offsetTopTop + "10px";
}

// Wunderbar
for (var left = el.offsetLeft, top = el.offsetTopTop, i = 0; i < 100; i++, top+=10, left+=10) {
  el[i].style.cssText += "; left: " + left + "px; top: " + top + "px;";
}
```
Good Reference Points

http://www.phpied.com/rendering-repaint-reflowrelayout-restyle/


https://developers.google.com/speed/articles/reflow
More CSSery

- Reference by element ID
- Be specific, but avoid child selectors
- Avoid @import()
- Avoid multi-class css rule (.foo.bar.baz)
More CSSery

- Pseudo selectors are slow
- Name space attribute selectors (type="…" vs input[type="…"])
- Eliminate un-used rules
- Avoid browser specific extensions (-webkit, -opera, -moz, etc...)
Micro-Case Study: SlashDot.org

Optimize the order of styles and scripts (9)
Put CSS in the document head (5)
Remove unused CSS rules (1986)
1986 rules (81%) of CSS not used by the current page.
- 69% is not used by the current page.
- classic.css: 79% is not used by the current page.
- providers.css: 97% is not used by the current page.
- Inline block #1: 69% is not used by the current page.
- Inline block #2: 81% is not used by the current page.
Use normal CSS property names instead of vendor-prefixed ones (251)
CSS Tools

https://github.com/Cerdic/CSSTidy  PHP

http://devilo.us/  Web-based
THANK YOU FOR LISTENING

Slides: http://ilia.ws
@iliaa