re-evaluating
front-end performance
best practices
@bentlegen
car · go cult
cargo cult
web performance
how does it happen?
deprecated ...

- books
- blog posts
- best practice guides
- StackOverflow answers
- performance analysis tools
- practices on live websites
- conference talks like this one
the agenda

• hostname sharding
• for-loop array length caching
• dynamic script insertion
hostname sharding
in the beginning (HTTP 1.0)
with more connections
with hostname sharding

www

www1, www2

www1
www2
www3
www4
still common

<table>
<thead>
<tr>
<th>Website</th>
<th># Static Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>plus.google.com</td>
<td>4</td>
</tr>
<tr>
<td>tumblr.com</td>
<td>4</td>
</tr>
<tr>
<td>alibaba.com</td>
<td>4</td>
</tr>
<tr>
<td>theverge.com</td>
<td>4</td>
</tr>
<tr>
<td>ebay.com</td>
<td>6</td>
</tr>
<tr>
<td>businessinsider.com</td>
<td>6</td>
</tr>
<tr>
<td>netflix.com</td>
<td>11!</td>
</tr>
</tbody>
</table>
Spreading static files over hostnames

As part of my routine scan through things like PageSpeed Insights, I decided to put my images on separate hosts to improve parallel downloads. Images are now served from a subdomain like this:

```
http://n26eh5.i.example.com/img/something.png
```

Where that `n36eh5` is the file's modification time, fantastic for caching since it will change immediately when the file does, automatically. Two birds with one stone, right?

Sure enough, I ended up scoring full points on the parallel downloads front.

Today I ran a test on another site. It's now telling me that I have too many DNS lookups.

Now...I have `*.i.example.com` set up as a wildcard vhost, but is that DNS lookup going to be an issue? With DNS caching, is it even a problem? After all, it will only be fetching that image the first time it gets requested, loading it from cache every time afterwards.

Should I look for a balance, or continue using the wildcard subdomain as I am now?

Tags: performance, http, dns, pagespeed
not a big deal anymore
## browser connections/origin

<table>
<thead>
<tr>
<th>Browser</th>
<th># HTTP / origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome 42</td>
<td>6</td>
</tr>
<tr>
<td>Firefox 37</td>
<td>8</td>
</tr>
<tr>
<td>Safari 7</td>
<td>6</td>
</tr>
<tr>
<td>IE 8, 9</td>
<td>6</td>
</tr>
<tr>
<td>IE 10</td>
<td>8</td>
</tr>
<tr>
<td>IE 11</td>
<td>13</td>
</tr>
</tbody>
</table>

chrome caps parallel image requests at 10 anyways

bit.ly/rprf-bscope
rise of https

- stalled/proxy negotiation
- connection + tls/ssl
- time-to-first-byte
- download

- dns lookup
- tls/ssl handshake
and http/2 will make all this irrelevant anyways
“2 domains for non-SPDY modern browsers”
– Souders in 2013
etsy case study

- 4 image domains → 2
- 50-80 ms faster for image heavy pages
- 30-50 ms faster overall
- up to 500ms faster on mobile

bit.ly/rprf-etsy
the web’s moving on

<table>
<thead>
<tr>
<th>Website</th>
<th># Static Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>netflix.com*</td>
<td>1</td>
</tr>
<tr>
<td>nytimes.com</td>
<td>1</td>
</tr>
<tr>
<td>youtube.com</td>
<td>2</td>
</tr>
<tr>
<td>twitter.com</td>
<td>2</td>
</tr>
<tr>
<td>facebook.com</td>
<td>2</td>
</tr>
<tr>
<td>pinterest.com</td>
<td>2</td>
</tr>
<tr>
<td>bbc.co.uk</td>
<td>2</td>
</tr>
<tr>
<td>etsy.com</td>
<td>3</td>
</tr>
</tbody>
</table>
looping
specifically array length caching in for loops
for (var i = 0; i < arr.length; i++) {
    // do stuff
}

vs

for (var i = 0, len = arr.length; i < len; i++) {
    // do stuff
}
Google Closure: How not to write JavaScript

Kevin Yank

Published November 12, 2009

At the Edge of the Web conference in Perth last week I got to catch up with Dmitry Baranovskiy, the creator of the Raphaël and gRaphaël JavaScript libraries. Perhaps the most important thing these libraries do is make sophisticated vector graphics possible in Internet Explorer, where JavaScript performance is relatively poor. Dmitry, therefore, has little patience for poorly-written JavaScript like the code he found in Google’s just-released Closure Library.

Having delivered a talk on how to write your own JavaScript library (detailed notes) at the conference, Dmitry shared his thoughts on the new library over breakfast the next morning. “Just what the world needs
for (var i = 0, len = this.players.length; i < len; i++) {
  var player = this.players[i];
  
  return true;
}
After performing this test with most modern browsers...

http://jsperf.com/caching-array-length/4

Currently, the fastest form of loop (and in my opinion the most syntactically obvious).

a standard for loop with length caching

```
for (var i = 0, len = myArray.length; i < len; i++) {
}
```

I would say this is definitely a case where I applaud JavaScript engine developers. A run time should be optimized for **clarity**, not **cleverness**.

Interestingly, in IE9 this is faster: for (var i = 0, len = myArray.length; i < len; ++i) {} // prefix incr, instead of postfix – Christopher Bennage Oct 31 '11 at 17:38
does it still hold?
cache vs no cache

ops/second (normalized), bigger is better

Chrome 42
Firefox 37
Safari 7
IE9+
IE8

78%
V8 (and other browsers) recognize this pattern

bit.ly/rprf-v8opt
```plaintext
t3 Parameter 1  // var arr
s11 Constant 0
s36 Constant 1
    for (var i = 0; i < arr.length; i++) {
        s24 LoadNamedField t3.%length@12
        Goto B2 Tagged
B2
        s16 Phi [ s11 s37 ]  // var i
            for (var i = 0; i < arr.length; i++) {
                CompareNumericAndBranch LT s16 s24 goto (B3, B5)
B3
            // Loop body would normally be here, but alas it is empty.
            Goto B4
B4
        for (var i = 0; i < arr.length; i++) {
        s37 Add s16 s36
        Goto B2
```
```javascript
Parameter 1 // var arr
Constant 0
Constant 1
    for (var i = 0, len = arr.length; i < len; i++) {
        LoadNamedField t3.%length@12
        Goto B2 Tagged
    }

Phi [ s11 s40 ] // var i
    for (var i = 0, len = arr.length; i < len; i++) {
        CompareNumericAndBranch LT s21 s16 goto (B3, B5)
    }

// Loop body would normally be here, but alas it is empty.
Goto B4

for (var i = 0, len = arr.length; i < len; i++) {
    Add s21 s39
    Goto B2
```
“we should start assuming that our code is optimized”
- Vyacheslav Egorov, V8
if you can trivially optimize it, the browser (probably) can too
mobile disagrees, a little

ops / second (normalized), bigger is better
if you think you’re going to get performance gains from optimizing for loops

you’re gonna have a bad time
dynamic script insertion
<script src="/app.js"></script>

VS

<script>
    var script = document.createElement('script');
    script.src = '/app.js';
    document.getElementsByTagName('head')[0].appendChild(script);
</script>
The best way to load external JavaScript

Posted at July 28, 2009 09:00 am by Nicholas C. Zakas
Tags: Blocking, JavaScript, Performance

Not too long ago, I wrote about loading JavaScript without blocking by creating a dynamic `<script>` tag. When `<script>` tags are in the flow of an HTML document, the browser must stop rendering and wait for the script file to download and execute before continuing (example). Creating a new `<script>` tag via JavaScript avoids this issue because it’s out of the flow of the document, so the script file is downloaded and executed without waiting. The result: dynamically loading JavaScript files allows your page to render faster and therefore improve perceived performance.

The best technique

Steve Souders has explored several different ways to load JavaScript without blocking both on his blog and in his books. After thinking about it and experimenting, I’ve come to the conclusion that there’s just one best practice for loading JavaScript without blocking:

1. Create two JavaScript files. The first contains just the code necessary to load JavaScript dynamically, the second contains everything else that’s necessary for the initial level of
2.2.3 **Dynamic script insertion**

It turns out you can re-create the behavior achieved by the async attribute by dynamically creating a script DOM element in JavaScript and appending it to the publisher’s page. Because you can append this script element to an arbitrary DOM location, even one that has already been processed by the browser, browsers don’t preserve execution order for JavaScript loaded in this fashion. And because execution order isn’t preserved, the browser downloads these files in parallel. This is your path to asynchronous script loading in browsers old and new.

Here’s how the script include snippet looks using dynamic `<script>` tag insertion.

**Listing 2.1**  Asynchronous script include

```javascript
<script>
(function() {
    var script = document.createElement('script');
    script.src = 'http://camerastork.com/widget.js?product=1234';
    script.async = true;

    var entry = document.getElementsByTagName('script')[0];
}
</script>
```

- Immediately-invoked function expression (IIFE) prevents declared variables from leaking into global scope.
- Set async property to true to support asynchronous loading in Opera and older versions of Firefox.
who does this?

• analytics: GA, Mixpanel, Chartbeat, Wordpress
• widgets: Disqus, Facebook Comments
• JS module loaders: RequireJS, LabJS
• script managers: Google Tag Manager, Segment
small problem: CSS Object Model
CSS Object Model (CCSOM)

```html
<link type="text/stylesheet" href="/main.css"/>
<script>
  window.getComputedStyle(document.body).margin;
</script>
```
CSSOM + dynamic script insertion

```html
<link type="text/stylesheet" href="/main.css"/>

<script>
  var script = document.createElement('script');
  script.src = '/app.js';
  document.getElementsByTagName('head')[0].appendChild(script);
</script>

<box>
can’t execute until CSS ready
</box>
```
dynamic script insertion

inline scripts can’t execute until CSSOM ready; downloading is delayed

scripts execute
blocking

1st script preloaded
1st script executes
2nd script downloads after 1st executes
“Have we been doing it all wrong?”
- Ilya Grigorik, 2014
3rd option: async attribute

<script async src="/app.js"></script>
ideal: async attribute

<table>
<thead>
<tr>
<th></th>
<th>Method</th>
<th>File</th>
<th>Domain</th>
<th>Type</th>
<th>Size</th>
<th>0 ms</th>
<th>640 ms</th>
<th>1.28 s</th>
<th>1.92 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>GET</td>
<td>quiet</td>
<td>jsbin.com</td>
<td>html</td>
<td>0.72 KB</td>
<td>→ 101 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>GET</td>
<td>style.css?rtt=2</td>
<td>udacity-crp.herokuapp.com</td>
<td>css</td>
<td>0.08 KB</td>
<td>→ 2160 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>GET</td>
<td>time.js?rtt=1&amp;a-async</td>
<td>udacity-crp.herokuapp.com</td>
<td>js</td>
<td>0.20 KB</td>
<td>→ 1135 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>GET</td>
<td>time.js?rtt=1&amp;b-async</td>
<td>udacity-crp.herokuapp.com</td>
<td>js</td>
<td>0.20 KB</td>
<td>→ 1135 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**All 3 resources download in parallel**

**Both scripts execute**
we should probably start using async
Simplify analytics snippet using `async` & `defer` #1696

mathiasbynens commented a day ago

With this snippet, modern browsers use `async`, older browsers (i.e. IE8 & IE9) use `defer`.

IE8 and IE9 lack `async` support but they have a broken implementation of `defer`. However, the brokenness doesn't apply in this scenario since no scripts depend on GA in the way jQuery UI depends on jQuery. `async` is also not supported by the Android 2.3 browser, but that browser does have a preload scanner to make up for it.

Once we drop support for IE8 and IE9, the `defer` attribute can be omitted.

The only downside is that the snippet is not a pure JavaScript solution anymore, meaning it cannot be moved or concatenated into a `.js` file. On the other hand, no one seemed to be doing that anyway; everyone just inlines the snippet into the HTML.

Ref. #1660 (comment)

paulirish commented 21 hours ago

love it.

👍
closing thoughts
don’t always believe what you read on the internet
benchmark your own stuff
benchmark your own stuff every year, apparently
always bet on browsers
(and JS engines)
thanks
acknowledgements

• performance research: Steve Souders, Ilya Grigorik, Guy Podjarny, Vyacheslav Egorov, Jonathan Klein, Paul Irish, Nicholas Zakas

• photos: Christian Junker, André Hofmeister, “My aim is true”

• me: Ben Vinegar (@bentlegen)

• office hours @ 1:30 PM today