

JavaScript Performance

MYTHBUSTERS



Crowdsourcing the results

- no dedicated test resources
- project runs in perpetuity
- real world test conditions
- aggregating results reduces bias
- new browsers show up immediately

Interpreting JSPerf results

- More is better
- Diminishing value with large numbers
 - 1M ops/sec vs 8M ops/sec is kinda meaningless ;)

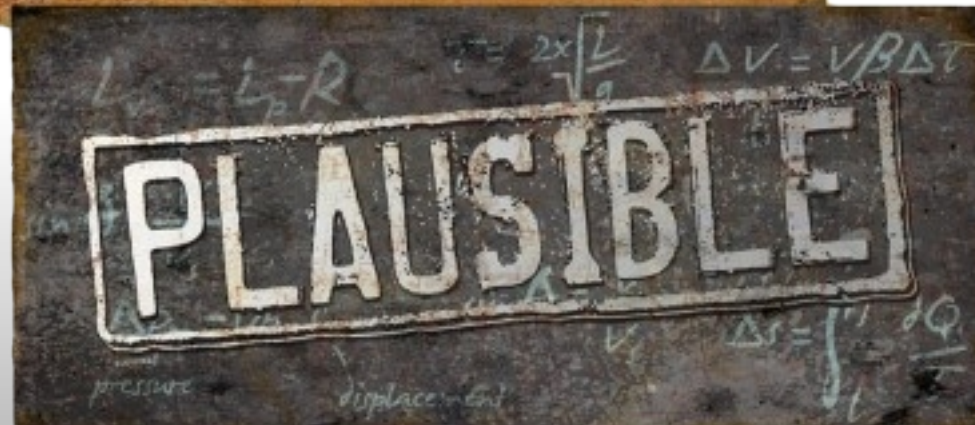


John-David Dalton (revision owner) commented on 17th March 2012: ∞

The point of this test is to show that even though there are micro differences there is no real world performance concern for using `void 0` over `undefined` or `typeof` as most of lowest ops/sec are still in the millions of operations a second.

- Don't forget about hardware bias

Test Time!





String concatenation is slow,
use `Array.join()` instead.

jsperf.com/string-concatenation-vs-array-join-2/8

Results table



MYTH

String concatenation is slow,
use `Array.join()` instead.



BUSTED



PLAUSIBLE

IE ≤ 7



Operations which require an implicit primitive-to-object cast/conversion will be slower.

udw.getti.fi/jsperf/object-casting



Operations which require an implicit primitive-to-object cast/conversion will be slower.





localStorage is too slow to use

jsperf.com/localstorage-vs-objects/19

Results table

There is no simple solution for local storage

on March 5, 2012 by [Chris Heilmann](#)

 99 comments

in [Featured Article](#) [HTML5](#) [IndexedDB](#) [localStorage](#) [Performance](#) [Research](#)

TL;DR: we have to stop advocating `localStorage` as a great opportunity for storing data as it performs badly. Sadly enough the alternatives are not nearly as supported or simple to implement.

When it comes to web development you will always encounter things that sound too good to be true. Sometimes they are good, and all that stops us from using them is our notion of being conspicuous about *everything* as developers. In a lot of cases, however, they really are not as good as they seem but we only find out after using them for a while that we are actually "doing it wrong".

One such case is local storage. There is a [storage specification](#) (falsely attributed to [HTML5](#) in a lot of examples) with an incredibly simple API that was heralded as the cookie killer when it came out. All you have to do to store content on the user's machine is to access the `navigator.localStorage` (or `sessionStorage` if you don't need the data to be stored longer than the current browser session):

ABOUT THE AUTHOR

Chris Heilmann

Principal Evangelist at Mozilla
for HTML5 and open web.
Let's fix this!



[Read more articles by Chris Heilmann...](#)

ARTICLES BY CATEGORY

[35 Days](#) (45)

[Identity](#) (7)

[@font-face](#) (9)

[Images](#) (4)

[about:hacks](#) (2)

[IndexedDB](#) (9)

[Add-ons](#) (11)

[JägerMonkey](#) (5)

[Animations](#) (3)

[JavaScript](#) (73)

[Apps](#) (19)

[Labs](#) (2)



localStorage is too slow to use



ish



"eval is evil", or in other words, eval is too slow and quirky to be considered useful.

jsperf.com/more-practical-eval/3

jsperf.com/eval-kills



"eval is evil", or in other words, eval is too slow and quirky to be considered useful.





You can rely on feature detection.

(aka hashbang routing is lame,
use `history.pushState!`)

jquery.bassistance.de/historytest/
HTML5 History test results

danwebb.net/2011/5/28/it-is-about-the-hashbangs



You can rely on feature detection.

(aka hashbang routing is lame,
use `history.pushState!`)



* but you should still prefer it



Frameworks/libraries (jQuery, etc) are performance-tuned *enough* by js gurus.

4vk.geti.fi/jsperf/jquery-this



Frameworks/libraries (jQuery, etc) are performance-tuned *enough* by js gurus.





querySelectorAll() is faster
than
getElementsByTagName()

[jsperf.com/queryselectorall-vs-
getelementsbytagname](http://jsperf.com/queryselectorall-vs-getelementsbytagname)

Results table



querySelectorAll() is faster
than
getElementsByTagName()





Converting a NodeList to an array is expensive, so you shouldn't do it.

jsperf.com/lists-of-nodes



Converting a NodeList to an array is expensive, so you shouldn't do it.





Script concatenation and/or
<script defer> is ***all*** you need
to load JS performantly.

(aka "Issue 28")

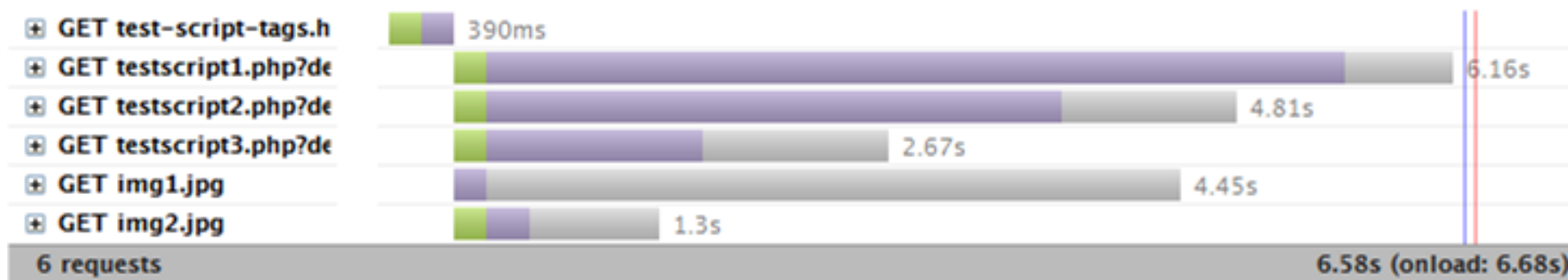
github.com/h5bp/html5-boilerplate/issues/28

<script defer>

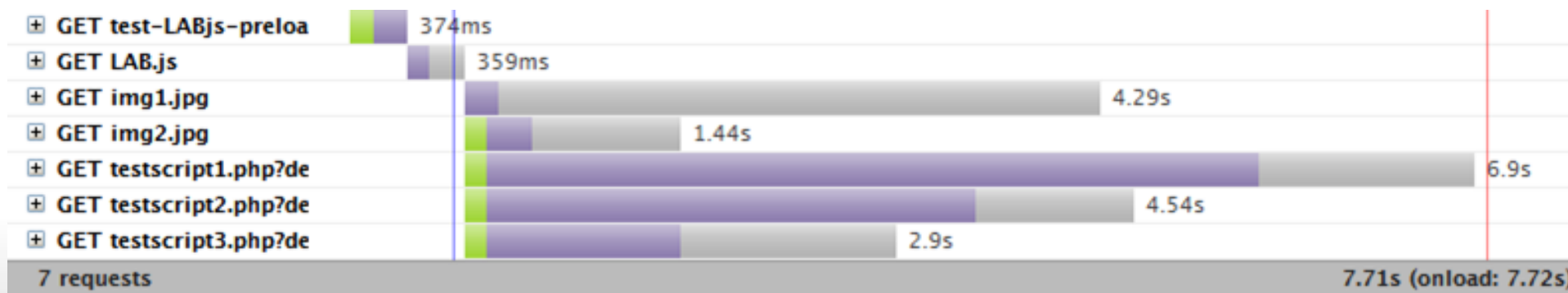
- horribly buggy across browsers (esp IE<=9!)

github.com/paulirish/lazyweb-requests/issues/42

- still would delay the DOM-ready event (bad!)...and btw, this is also buggy cross-browser!



<script> or <script defer>



dynamic script loading

<script "one-big-file.js">

- at some point, "big" is big enough to overcome HTTP connection overhead, meaning a second parallel request for half the file will be faster.

NOTE: **do** concat your files first, **then** (if size $\geq 100k$) chunk into 2-3 ~equal-sized parts.

`<script "one-big-file.js">`

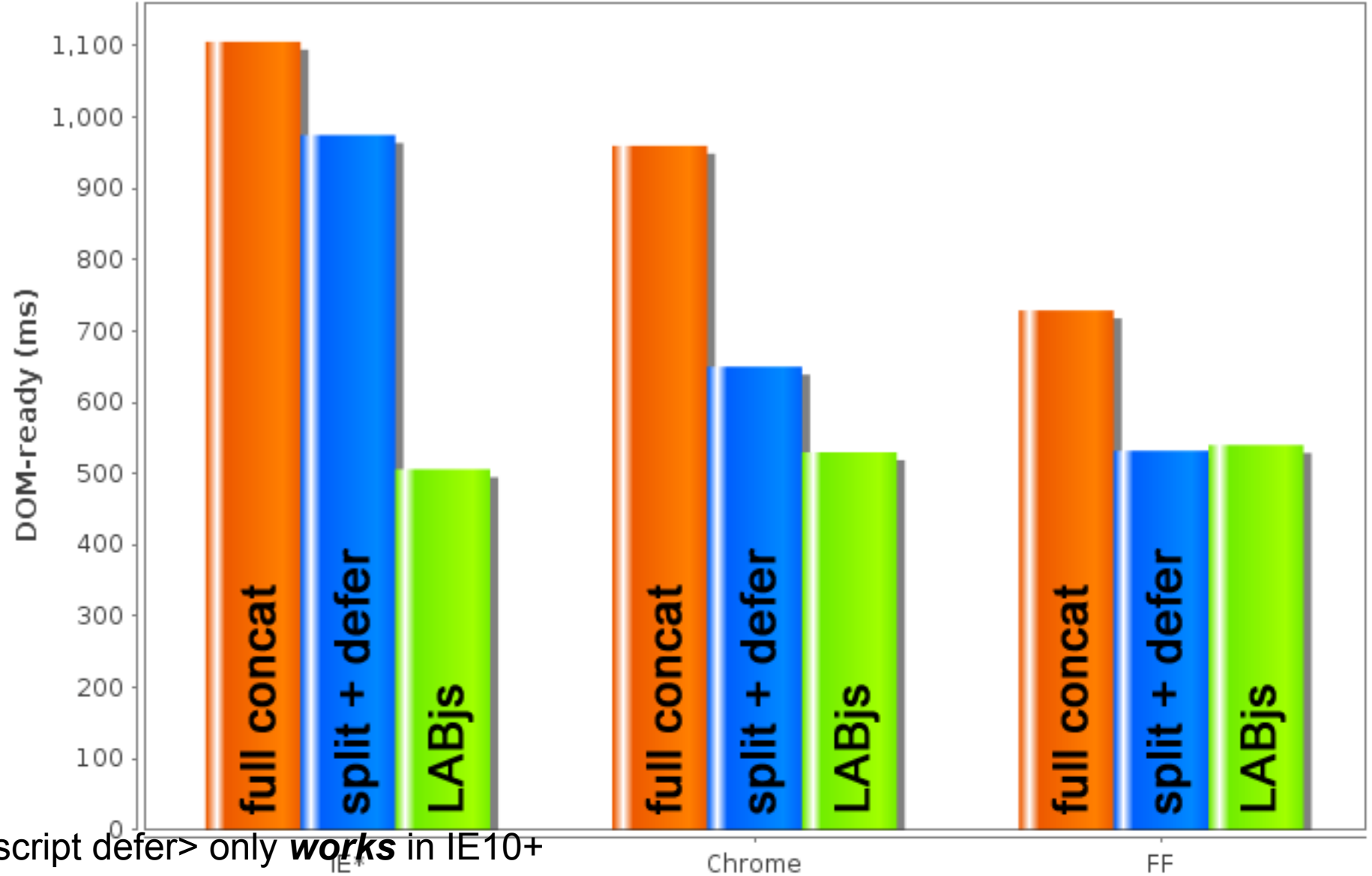
- chunking a big file into 2-3 parts, based on volatility of code (how often it changes) allows for different caching headers for each part (win!).

NOTE: jquery.js is far more stable than your site's code.

`<script "one-big-file.js">`

- chunking your concat'd code file allows you to load the chunks separately, like one before load, the next 100ms after DOM-ready, etc.

NOTE: lazy-loading is an established best-practice!



Experimenting with JS loading techniques on a real site (incl. jquery, jquery-ui, site code)



Script concatenation and/or
<script defer> is ***all*** you need
to load JS performantly.



(mostly)



Use native methods for better performance.

es5.github.com/#x15.4.4.18

jsperf.com/accessor-perf

jsperf.com/bind-vs-custom

jsperf.com/for-vs-array-foreach

MYTH

Use native methods for better performance.

Status:

BUSTED



The string method `indexOf` is much faster than using a regular expression to search a string.

imb.getti.fi/jsperf/regex/indexof



The string method `indexOf` is much faster than using a regular expression to search a string.

