Sam Eubank

JavaScript Performance

Speed is a feature, and in the world of JavaScript, the quest for better performance has led to “browser wars” and “framework wars” which continue to rage on.

Let’s explore some JavaScript performance testing tools and learn some tips for crafting definitely faster experiences for our users

Qumulo

ElasticSearch?

**Web Application Performance**

* How to deliver the same or better UX in less time

[seubank@salesforce.com](mailto:seubank@salesforce.com)

*have it load a few millisecond faster, that’s a win.*

Agenda

1. Anatomy – where does work happen?
2. Measurement – how do we define “fast”?
3. Improvement – how can we get faster?

**Anatomy**

Server ==== Distance (Bandwidth) ==== Client

*How far away you are to the data*

*Distance matters, speed of light.*

*Load balance on multiple regions*

*Is like 2 speed of light hops*

**Server & Client**

Moore’s Law

* CPU power doubles every 2 years

**Bandwidth**

Nielsen’s Law

* High-end users’ bandwidth grows by 50% per year.

**Distance**

Eubank’s law (LOL) :)

* If the distance between Bellevue and San Francisco ever changes, we’ll have bigger problems than web performance.

*Web hops kills web site performance*

**Increasingly true as time rolls on…**

“Your computer can probably render a page in less time than it takes to ping the servers that its data come from.”

Solutions -> CDNs (content delivery network), caching, combining requests

*Not making a lot of request, can solve web performance*

*CDN is not dependent on what on it, it’s just a file server.*

*Most modern CDN has problematic capabilities.*

*Think of it like a static file server. React minified.*

**Measurement**

How do we define “fast”?

Why measure?

**Because Science**

Measurements are the only way to…

* Make provably significant improvements.
* Detect significant regressions.

*No measurements mean no proof of improvement – Amazon*

*Regressions = performance degradation.*

**Customer Focus**

Performance is the speed with which an application finishes doing what you asked it to do.

* For example: “**show** a list of **clickable** results”>

**window.performance.timings**

navigationStart – *click on something when you get to a new page. When you told your browser, hey I want you to load this thing.*

unloadEventStart – *browser is really doing work.*

unloadEventEnd

redirectStart

redirectEnd

fetechStart

domainLookupStart

domainLookupEnd

connectStart

connectEnd

secureConnectionStart

requestStart

responseStart

responseEnd

domLoading

domInteractive

domContentLoadedEventStart

domContentLoadedEventEnd

domComplete

loadEventStart

loadEventEnd

**Visibility**

Avoid measuring load times for things that are out of view

(i.e. below the fold behind a timer, or after an interaction).

Solutions -> Pixel onload, Video recoding

Example -> Amazon Above-the-Fold (ATF) pixel

*No point of loading 5 and on, until you scroll.*

*Care more about above and below the fold.*

*Was stuff visible before the window fired?*

**Single Page Applications**

When navigating between views in a SPA, navigationStart and loadEventEnd don’t re-trigger.

Solution -> construct your own

Example -> Salesforce Experienced Page Time (EPT)

*After page has loaded, the window.performance.timings doesn’t change.*

**How to measure**

**Problem: Noise**

Variance is an unavoidable reality, even in the case of simple benchmarks running simultaneously on identical hardware.

Solution -> ….

**Statistics**

A distribution of measurement samples can be understood using statistics, such as…

* Mean +/- Standard Deviation
* Median +/- Median Absolute Deviation
* Percentiles: p25, p50, p75, p90, p99

**Goal-Focused Statistics**

Owner: “I want **most** page loads to take less than 2 seconds.”

* Goal: median(loadEventEnd - navigationStart ) < 2000ms

Owner: “I want pages to load in under 5 seconds **90%** of the time.”

* Goal: p90(loadEventEnd – navigationStart) < 5000ms

**Avoid Means**

Performance sample distributions are typically right-skewed, so their medians tend to be lower than their means.

*Slow wifi, etc. can distort the graph/curve of request distribution.*

[Graph: normal distribution but skewed right and has a long tail]

*Mean will be affected by the long tail.*

**Wild West or Gated Community?**

Option 1: Real User Measurement (RUM)

Option 2: Lab Simulations

*Don’t want to run experiments in Salesforce, we don’t want to lose trust.*

**Real User Measurement (RUM)**

Advantages:

Measures the target audience

No infrastructure required

Problems:

Highly variable

Can’t run on demand

*Can’t run experiments on on-demand.*

**Lab Simulations**

Advantages:

More controllable

Can run specific benchmarks on demand

Problems:

Requires maintenance

Difficult to cover all scenarios

*We do a combination of both. Hybrid of two, gives the benefit of both.*

**Improvement**

How can we make stuff faster?

in general

reduce total number of…

* Transfer bytes
* CSS rules
* Function calls
* Loop iterations
* DOM nodes
* .. everything

*forEach is very expensive on performance*

**Server**

* Enable compression
* Enable streaming & send first bytes ASA
* Limits redirects
* Serve static assets from CDN
* Use high-concurrency technologies (e.g. non-blocking I/O)

**Transfer**

* Minify code & Optimize images
* Combine and parallelize requests
* Cache whenever possible
* Omit unused JS code & CSS rules
* Lazy-load content that is initially invisible
* Pre-cache likely next-page data/assets

*There is a way to check for unused JS codes*

*Lazy-load like amazon search pages*

*Pre-cache likely – predicting where user will click/go next*

*Hundred milliseconds for you actually click the mouse*

*When user hovers on the button, prefetch the JSON file / render page.*

*A tap is capture on touch-end. A middle-touch, might be a possible end-touch.*

**Client**

* Declare CSS first
* Avoid descendent CSS selectors
* Limit DOM depth
* Use fixed sizes when possible
* Don’t modify the DOM post-render

*Avoid descendent CSS selectors could be N squared look up time, cause it will go up to its parent to look for CSS.*

*Use fixed sizes when possible – if not, as data is loading, it is dynamically sizing, and it keeps on rendering.*

*CPU reloads the content on every flow (resize) in every 200 milliseconds*

**Page Speed Insights**

Tool in Chrome

Audits tab in Chrome

* Lighthouse

*You can emulate mobile experience, crappy 3g, full cache, clean cache but not all at once.*

Performance tab in Chrome

*You can record performance, CPU profiles and timelines.*

*Make sure not using different versions of jquery/react, but they do.*

*Istanbul for front-end tests?*

*Front-end JS is usually not well tested than back-end JS that use Mocha.*

**Firefox Performance Tab**

*Some of framework in Salesforce uses TypeScript*

**Exam**

Benchmark comparison tool

**Best (tool)**

*Compare two different commits and run the same benchmark. Checks the performance.*

**Pau (tool)**

*Pau will run headless in Chrome and Firefox. Interface in Safari. None yet in Opera.*

*Both Best and Pau from Salesforce might be open-sourced in the future.*

Join us: goo.gl/HCfFNt (link to Saleforce Job - UI Software Engineer, Performance)

*Double-click when making ad-decision, double hops.*

*Amazon I get what I get, Salesforce is not out-of-the-box*

*They more flexibility, the less control you have on how fast it is.*

*Experimentation on http2?*

*More people are looking at the bleeding edge.*