# 11. PERFORMANCE IN JAVASCRIPT

## Caching

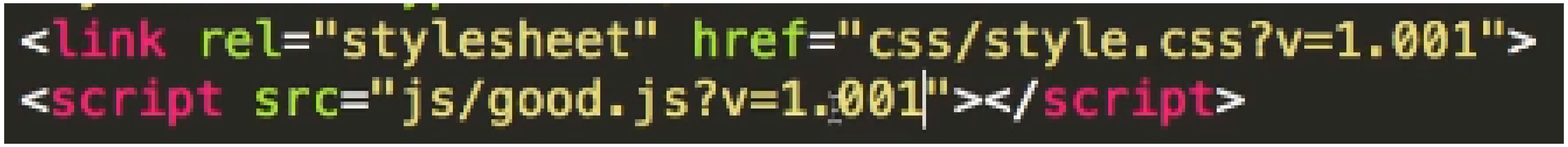
- Caching the scripts at the server side and in the browser
- Using Consistent URLs –In case things like code, script, css or images are used at different places, make sure to refer to the path correctly
- Content Delivery Network
  - Stores static assets to third-party server
  - Helps to avoid overloading and crashing of website.

# ETAGS and Expiry Rules

- ETAG assigns a unique ID to every file
- Expiry rule states for how long a file will remain unchanged so that the site doesnot have to download it till that time.
- If you are not much familiar with the backend configurations then turn ETAGS off and set **explicity expirty rules.**
- **.htaccess.txt**

## Using Version Control for caching

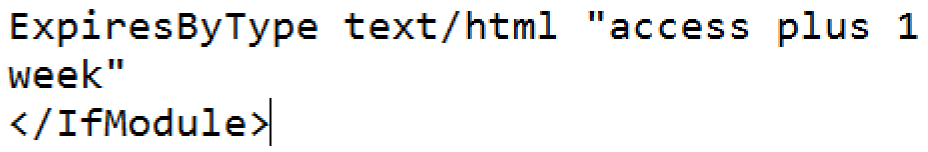
- Cache Controller
  - Versioning your scripts, to control the caching



```
<link rel="stylesheet" href="css/style.css?v=1.001">
<script src="js/good.js?v=1.001"></script>
```

- For frequently updated HTML files set the **ExpiresByType** to short interval, say 1 week.

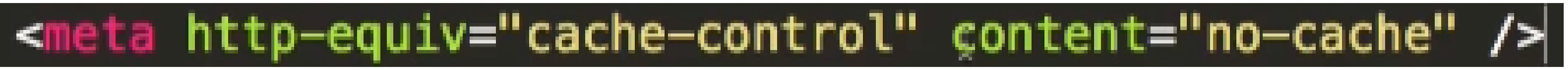
.htaccess.txt



```
ExpiresByType text/html "access plus 1
week"
</IfModule>
```

## Enabling No-cache

- Html meta-data include the following



```
<meta http-equiv="cache-control" content="no-cache" />
```

- Secure information should not be cached
- One day expiry for the html in **.htaccess.txt**

## Minify our code

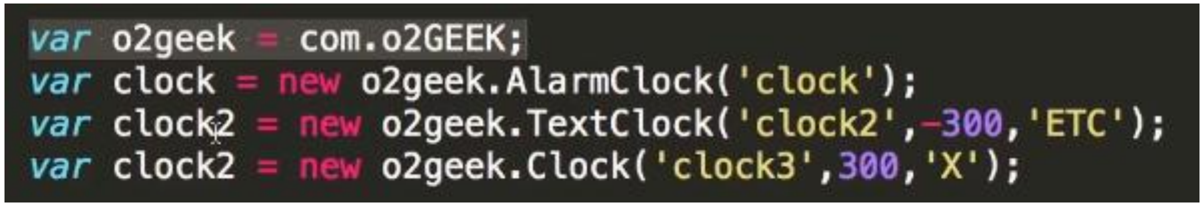
- Minification of your css and javascript code to load your files quickly
- http://tools.w3clubs.com/cssmin/

## Variable scope

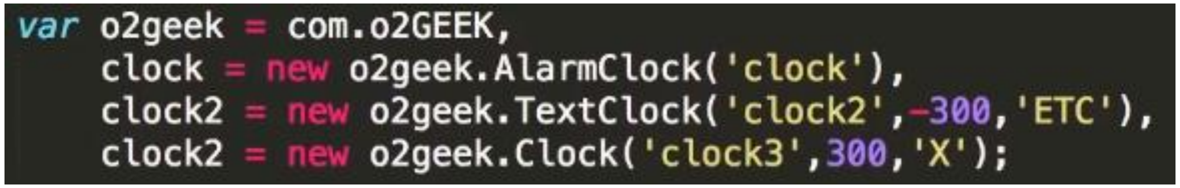
- Limit your variable scope within the local scope.
- Try not to create global scope variables
- Create local variables for references being used more often to avoid frequent call backs. Thus, we can drastically improve the performance.

# Condensing var definitions

- Combining different variable definitions, within a function, together



```
var o2geek = com.o2GEEK;
var clock = new o2geek.AlarmClock('clock');
var clock2 = new o2geek.TextClock('clock2',-300,'ETC');
var clock2 = new o2geek.Clock('clock3',300,'X');
```



```
var o2geek = com.o2GEEK,
    clock = new o2geek.AlarmClock('clock'),
    clock2 = new o2geek.TextClock('clock2',-300,'ETC'),
    clock2 = new o2geek.Clock('clock3',300,'X');
```

## Strict Equality Check

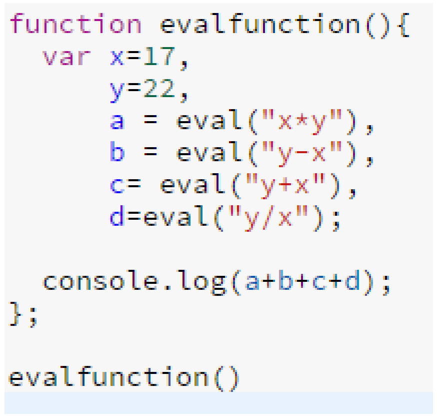
- Always perform **strict equality check** in your comparison
  **“===“**
  - Where ever possible!!!

## Strings

- The most cost effective way to building strings is with Arrays.
- Google recommends to use arrays to concatenate or copy strings using arrays.

# Avoiding **eval** for object references

- Analyze an **eval function**
- **Use an alternative to eval function**
- **What is eval function**
- Evaluate/Execute JavaScript code/expressions:



```
function evalfunction(){
  var x=17,
      y=22,
      a = eval("x*y"),
      b = eval("y-x"),
      c= eval("y+x"),
      d=eval("y/x");

  console.log(a+b+c+d);
};

evalfunction()
```

# Reducing Anonymous function

- It is a best practice to reduce / avoid the use of anonymous functions.
- Anonymous function is a function that is a standalone function, which may or may not be attached to a variable.
- Avoid creating functions/anoymous functions inside loops.

## Best Practice

- Data validation and checks to be done at client thoroughly prior to sending them to the server.
  - – Minimise the calls to the server
  - – Move data processing to the client-side
  - – What data types to use ?
    - JSON
    - XML
    - Web Services

It depends on the situation!

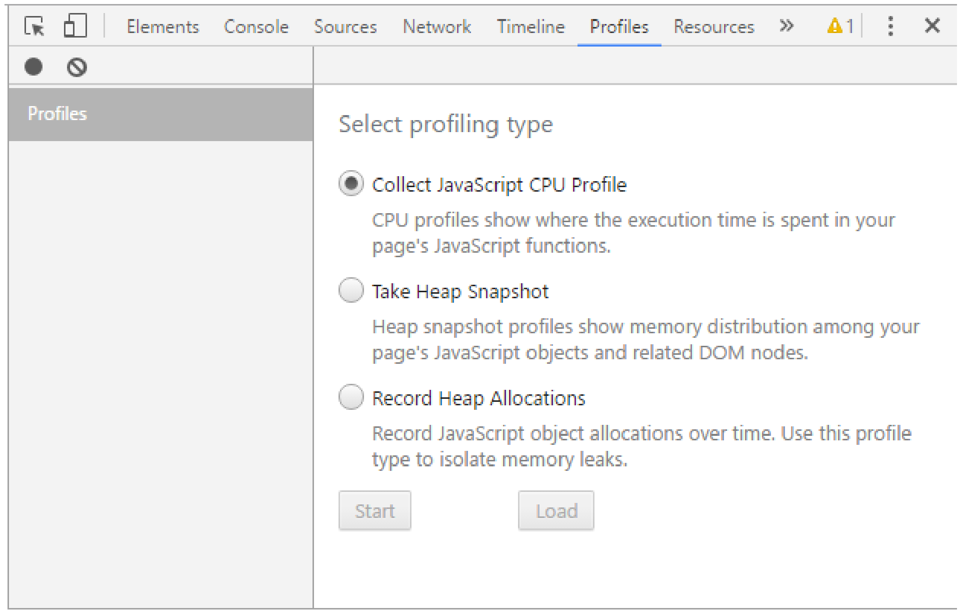
**JSON – JavaScript Object Notation**
- **Simple and lightweight**
- **Consumes less memory**

# Client-Server Communication

- While working with a client and server, some of the questions that are important to answer are:
  – How often do we need live data from our server
  – Do we want users to get this data instantly or after a particular time.
  – Can we create a local storage for this data for the user instead of communication between client and server to fetch the data every time?
  – What is the nature of security profile between the client and server?
  – Less the communication between client and server, the better the app performance.
  – Is session based storage required?

# JavaScript Profiling

- What functions are taking most of the processing time?
- How do we monitor in real time?
- What is the total amount of allocation of resources?
- HEAP Snapshots - show memory distribution among your page's JavaScript Objects and related DOM nodes.
- Record Heap Allocations – record JavaScript Object allocations over time. Used to isolate memory leaks.



Select profiling type

(●) Collect JavaScript CPU Profile

CPU profiles show where the execution time is spent in your page's JavaScript functions.

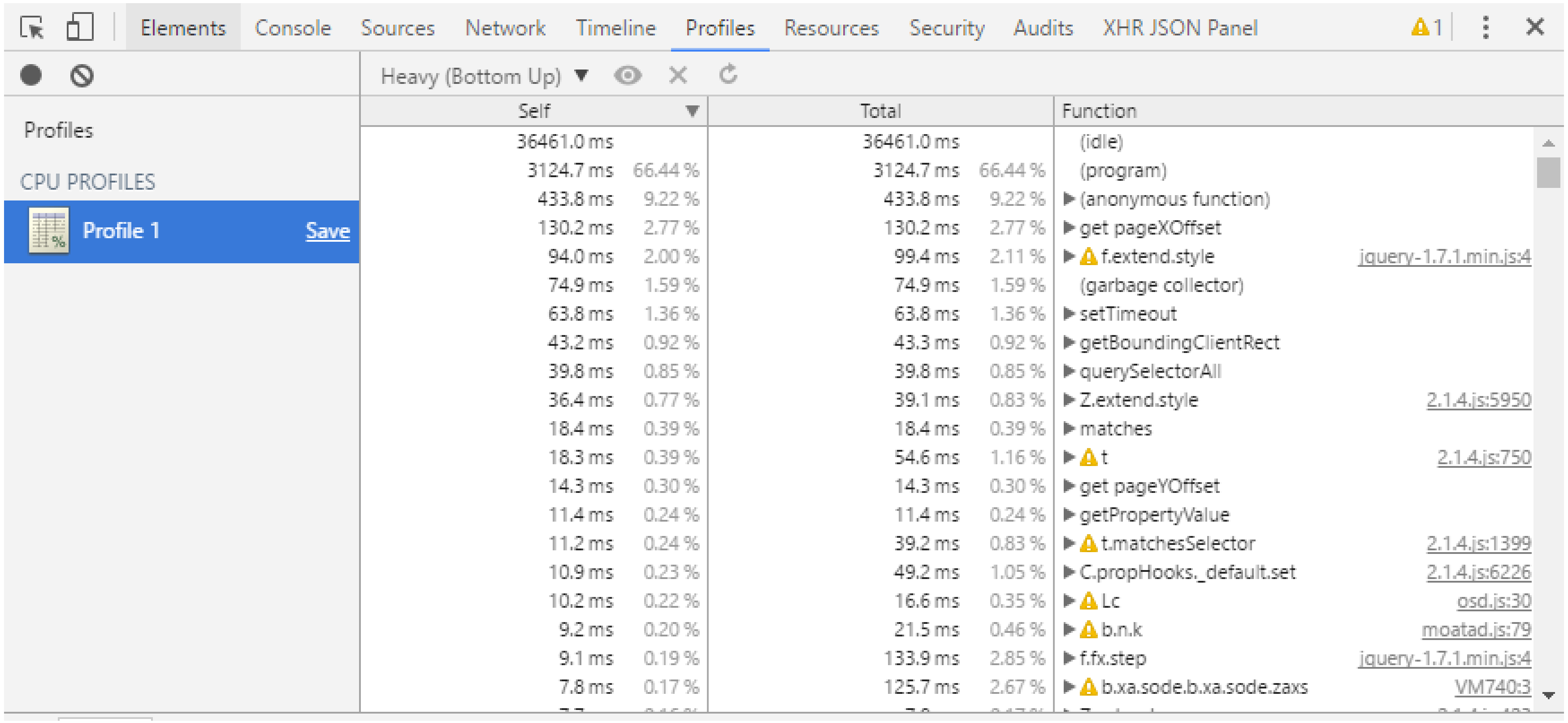
( ) Take Heap Snapshot

Heap snapshot profiles show memory distribution among your page's JavaScript objects and related DOM nodes.

( ) Record Heap Allocations

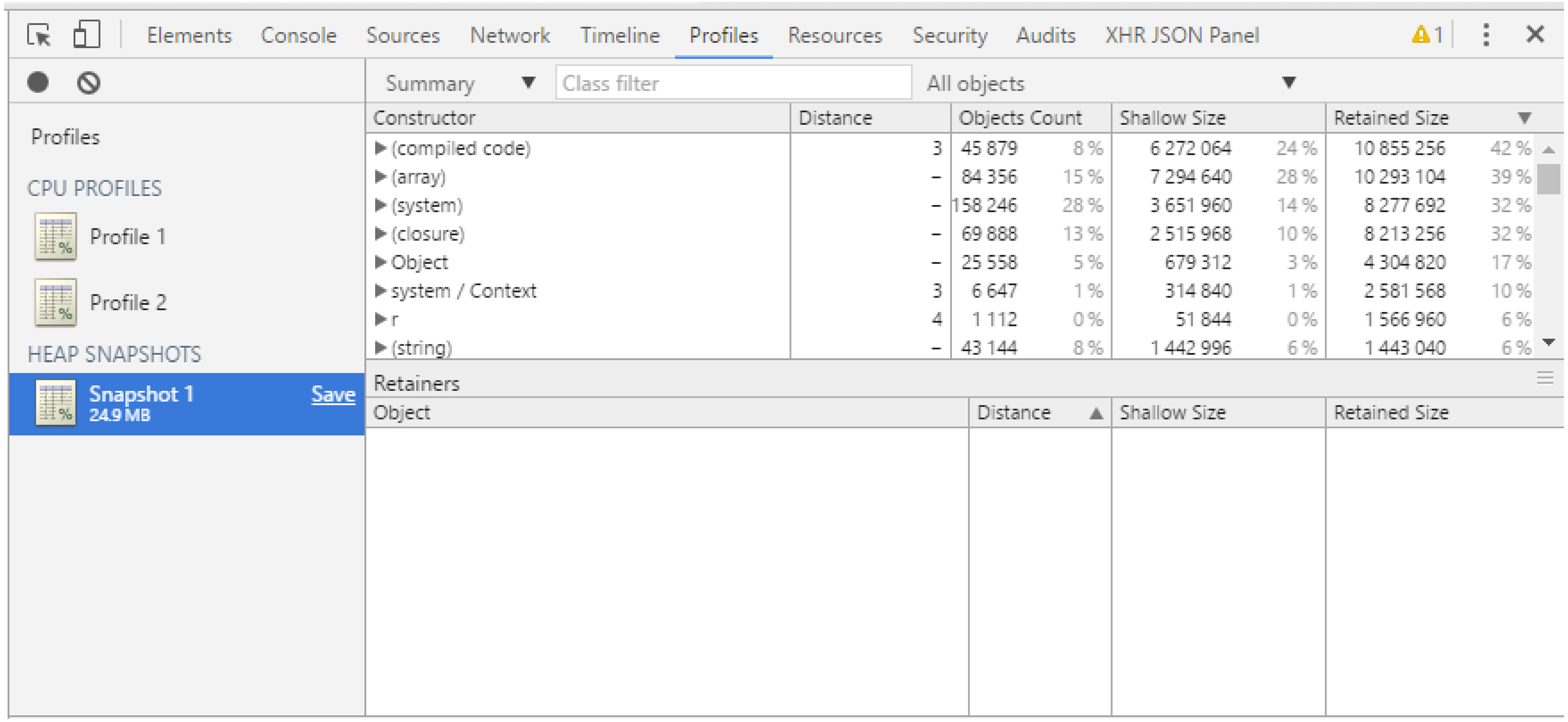
Record JavaScript object allocations over time. Use this profile type to isolate memory leaks.

# JavaScript CPU Profile



| Self | | Total | | Function | |
|---|---|---|---|---|---|
| 36461.0 ms | | 36461.0 ms | | (idle) | |
| 3124.7 ms | 66.44 % | 3124.7 ms | 66.44 % | (program) | |
| 433.8 ms | 9.22 % | 433.8 ms | 9.22 % | ▶ (anonymous function) | |
| 130.2 ms | 2.77 % | 130.2 ms | 2.77 % | ▶ get pageXOffset | |
| 94.0 ms | 2.00 % | 99.4 ms | 2.11 % | ▶ ⚠ f.extend.style | jquery-1.7.1.min.js:4 |
| 74.9 ms | 1.59 % | 74.9 ms | 1.59 % | (garbage collector) | |
| 63.8 ms | 1.36 % | 63.8 ms | 1.36 % | ▶ setTimeout | |
| 43.2 ms | 0.92 % | 43.3 ms | 0.92 % | ▶ getBoundingClientRect | |
| 39.8 ms | 0.85 % | 39.8 ms | 0.85 % | ▶ querySelectorAll | |
| 36.4 ms | 0.77 % | 39.1 ms | 0.83 % | ▶ Z.extend.style | 2.1.4.js:5950 |
| 18.4 ms | 0.39 % | 18.4 ms | 0.39 % | ▶ matches | |
| 18.3 ms | 0.39 % | 54.6 ms | 1.16 % | ▶ ⚠ t | 2.1.4.js:750 |
| 14.3 ms | 0.30 % | 14.3 ms | 0.30 % | ▶ get pageYOffset | |
| 11.4 ms | 0.24 % | 11.4 ms | 0.24 % | ▶ getPropertyValue | |
| 11.2 ms | 0.24 % | 39.2 ms | 0.83 % | ▶ ⚠ t.matchesSelector | 2.1.4.js:1399 |
| 10.9 ms | 0.23 % | 49.2 ms | 1.05 % | ▶ C.propHooks._default.set | 2.1.4.js:6226 |
| 10.2 ms | 0.22 % | 16.6 ms | 0.35 % | ▶ ⚠ Lc | osd.js:30 |
| 9.2 ms | 0.20 % | 21.5 ms | 0.46 % | ▶ ⚠ b.n.k | moatad.js:79 |
| 9.1 ms | 0.19 % | 133.9 ms | 2.85 % | ▶ f.fx.step | jquery-1.7.1.min.js:4 |
| 7.8 ms | 0.17 % | 125.7 ms | 2.67 % | ▶ ⚠ b.xa.sode.b.xa.sode.zaxs | VM740:3 |

# HEAP SNAPSHOT

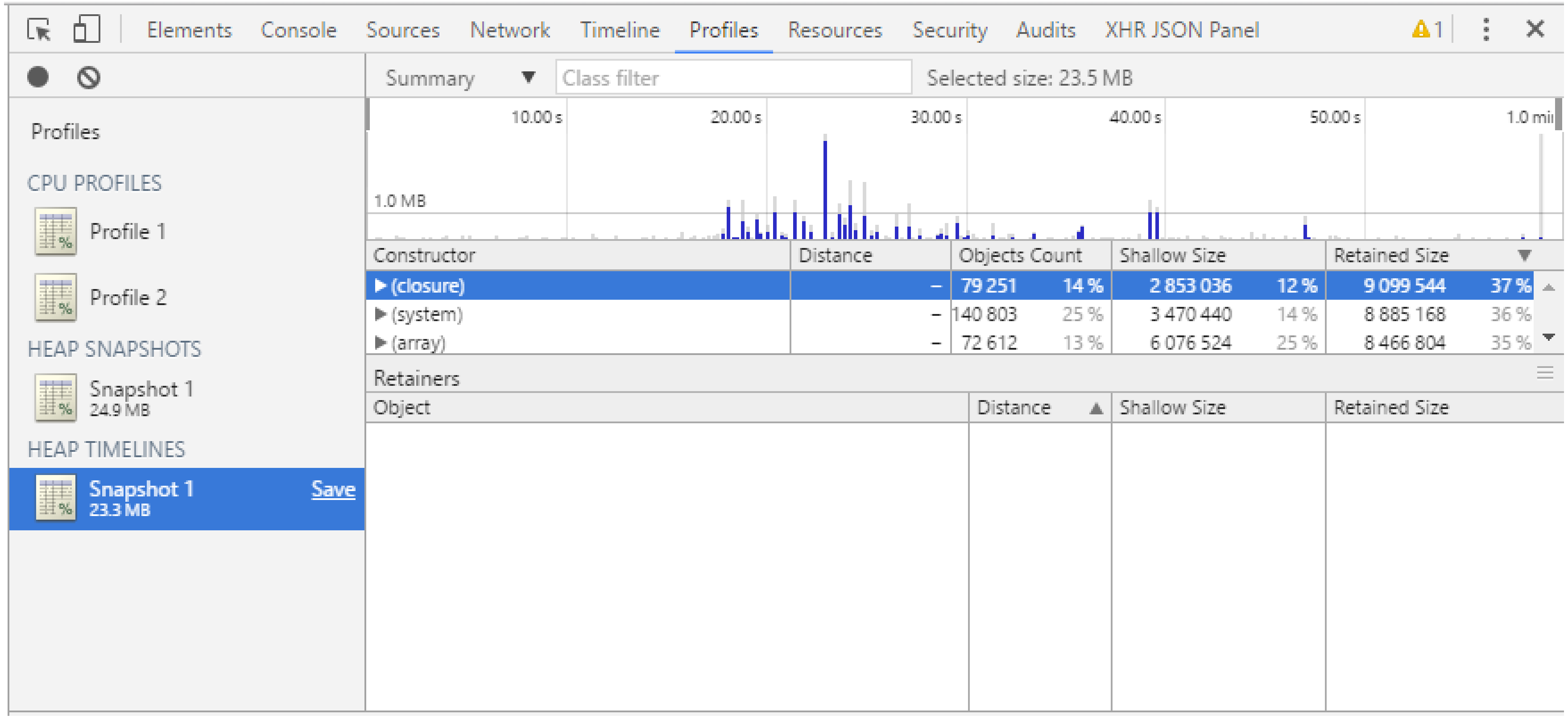


| Constructor | Distance | Objects Count | | Shallow Size | | Retained Size | |
|---|---|---|---|---|---|---|---|
| ▶ (compiled code) | 3 | 45 879 | 8 % | 6 272 064 | 24 % | 10 855 256 | 42 % |
| ▶ (array) | – | 84 356 | 15 % | 7 294 640 | 28 % | 10 293 104 | 39 % |
| ▶ (system) | – | 158 246 | 28 % | 3 651 960 | 14 % | 8 277 692 | 32 % |
| ▶ (closure) | – | 69 888 | 13 % | 2 515 968 | 10 % | 8 213 256 | 32 % |
| ▶ Object | – | 25 558 | 5 % | 679 312 | 3 % | 4 304 820 | 17 % |
| ▶ system / Context | 3 | 6 647 | 1 % | 314 840 | 1 % | 2 581 568 | 10 % |
| ▶ r | 4 | 1 112 | 0 % | 51 844 | 0 % | 1 566 960 | 6 % |
| ▶ (string) | – | 43 144 | 8 % | 1 442 996 | 6 % | 1 443 040 | 6 % |

Retainers

| Object | Distance | Shallow Size | Retained Size |
|---|---|---|---|

# HEAP ALLOCATION TIMELINE



Selected size: 23.5 MB

Profiles

CPU PROFILES

- Profile 1
- Profile 2

HEAP SNAPSHOTS

- Snapshot 1 24.9 MB

HEAP TIMELINES

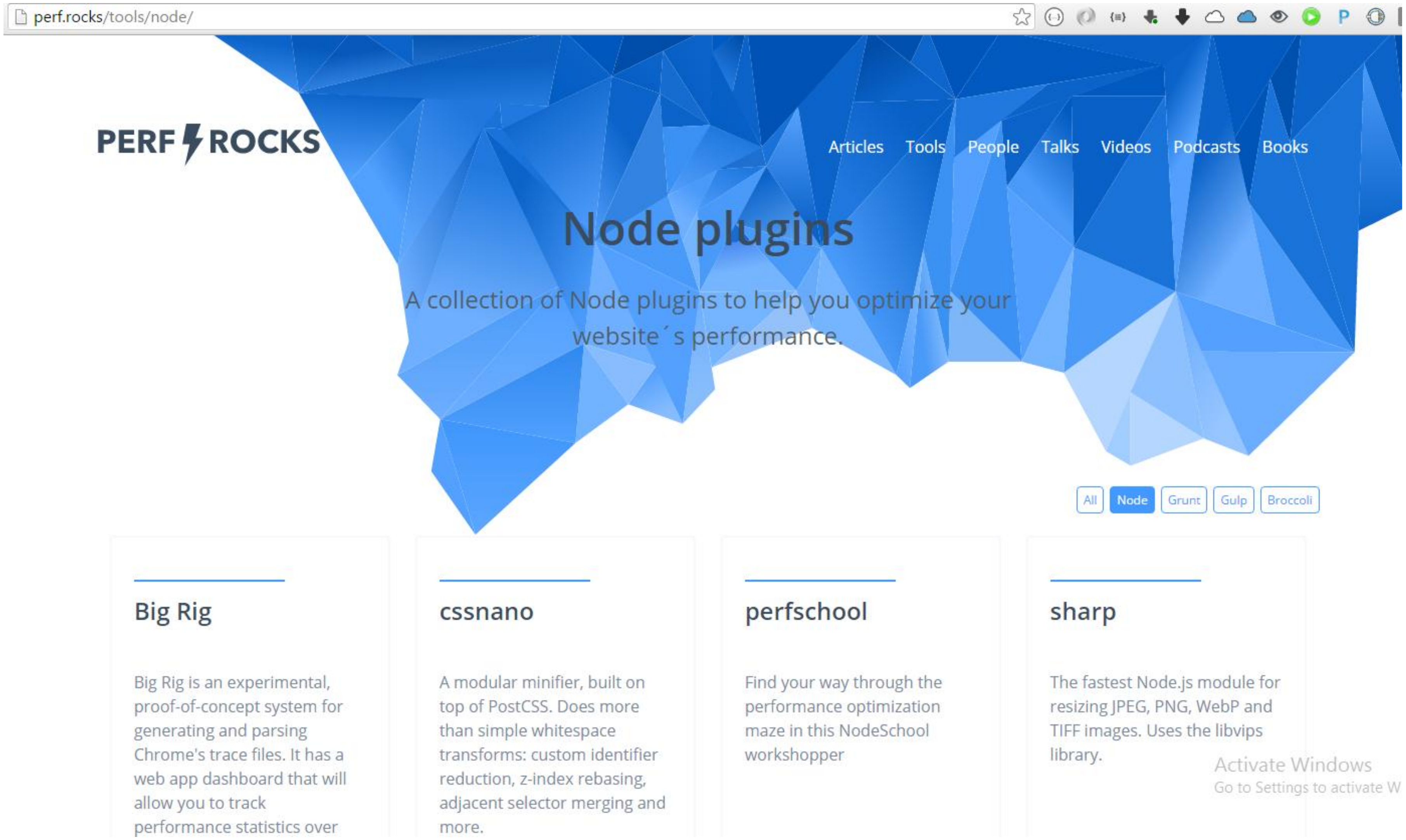
- Snapshot 1 23.3 MB Save

| Constructor | Distance | Objects Count | | Shallow Size | | Retained Size | |
|---|---|---|---|---|---|---|---|
| ▶ (closure) | – | 79 251 | 14 % | 2 853 036 | 12 % | 9 099 544 | 37 % |
| ▶ (system) | – | 140 803 | 25 % | 3 470 440 | 14 % | 8 885 168 | 36 % |
| ▶ (array) | – | 72 612 | 13 % | 6 076 524 | 25 % | 8 466 804 | 35 % |

Retainers

| Object | Distance ▲ | Shallow Size | Retained Size |
|---|---|---|---|

# Perf.rocks



perf.rocks/tools/node/

PERF ROCKS

Articles Tools People Talks Videos Podcasts Books

Node plugins

A collection of Node plugins to help you optimize your website´s performance.

All Node Grunt Gulp Broccoli

**Big Rig**

Big Rig is an experimental, proof-of-concept system for generating and parsing Chrome's trace files. It has a web app dashboard that will allow you to track performance statistics over

**cssnano**

A modular minifier, built on top of PostCSS. Does more than simple whitespace transforms: custom identifier reduction, z-index rebasing, adjacent selector merging and more.

**perfschool**

Find your way through the performance optimization maze in this NodeSchool workshopper

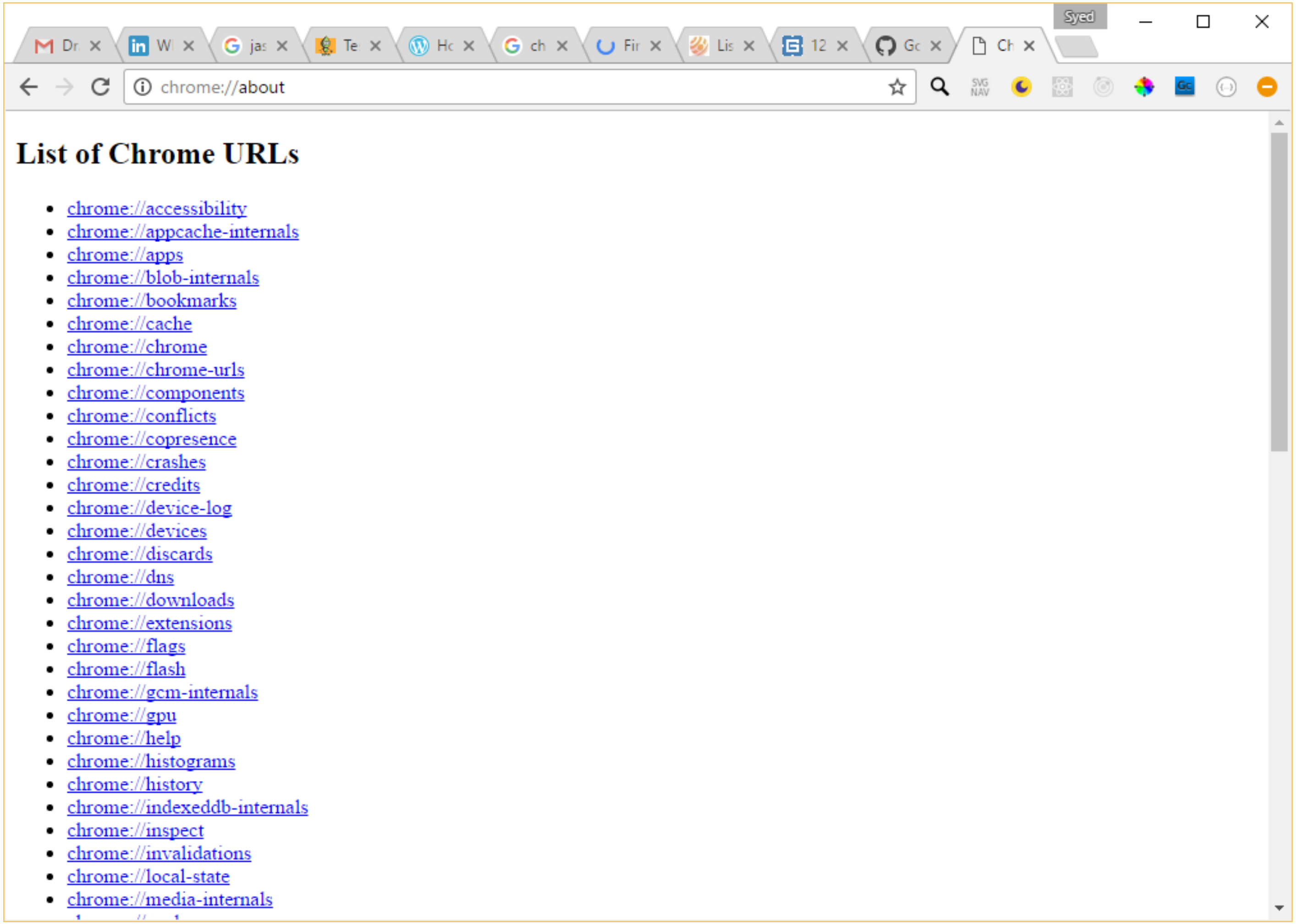
**sharp**

The fastest Node.js module for resizing JPEG, PNG, WebP and TIFF images. Uses the libvips library.

Activate Windows
Go to Settings to activate W

# List of Chrome URLS

Chrome://about

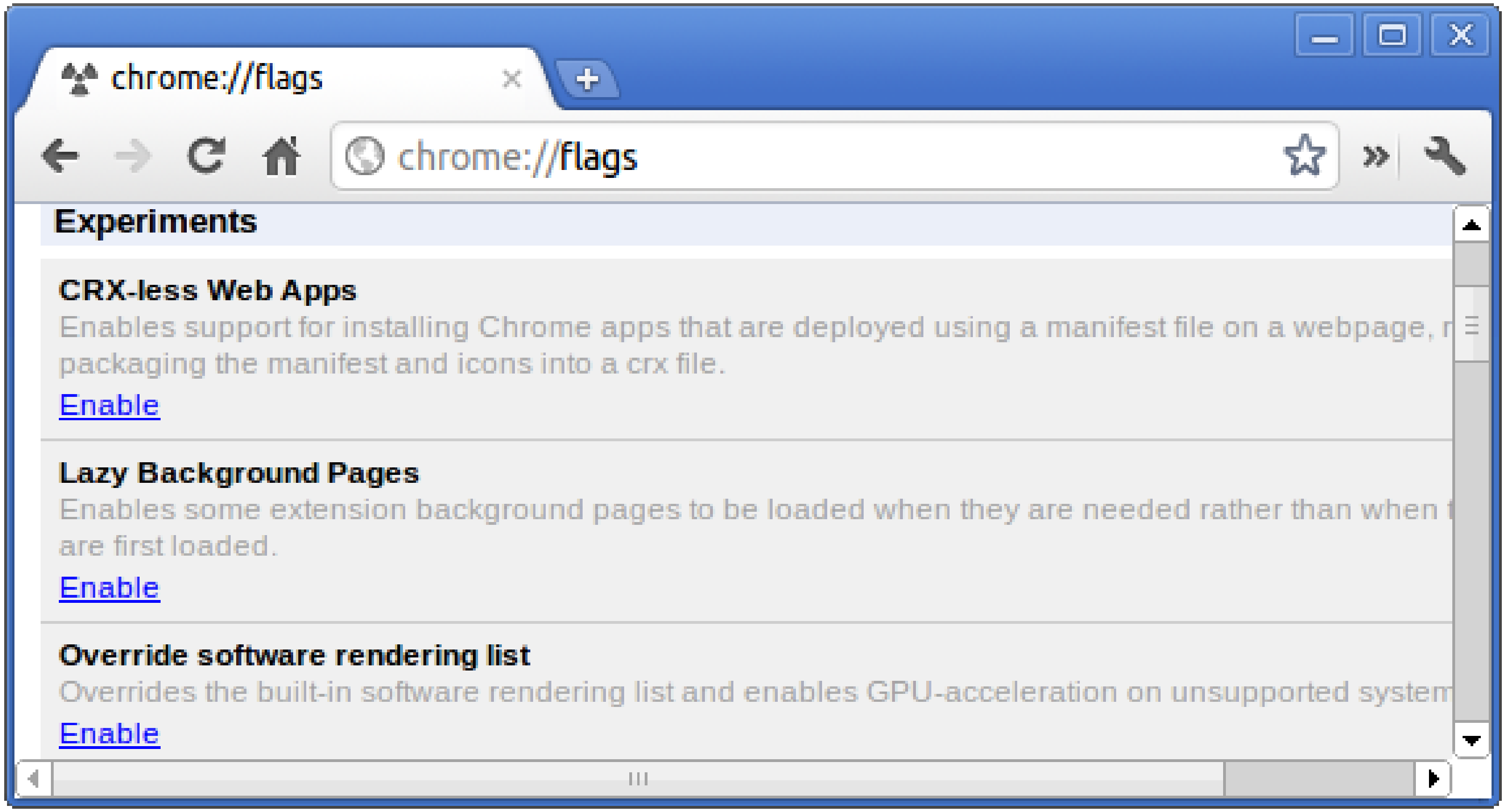


chrome://about

## List of Chrome URLs

- chrome://accessibility
- chrome://appcache-internals
- chrome://apps
- chrome://blob-internals
- chrome://bookmarks
- chrome://cache
- chrome://chrome
- chrome://chrome-urls
- chrome://components
- chrome://conflicts
- chrome://copresence
- chrome://crashes
- chrome://credits
- chrome://device-log
- chrome://devices
- chrome://discards
- chrome://dns
- chrome://downloads
- chrome://extensions
- chrome://flags
- chrome://flash
- chrome://gcm-internals
- chrome://gpu
- chrome://help
- chrome://histograms
- chrome://history
- chrome://indexeddb-internals
- chrome://inspect
- chrome://invalidations
- chrome://local-state
- chrome://media-internals

# Chrome://flags



**Experiments**

**CRX-less Web Apps**
Enables support for installing Chrome apps that are deployed using a manifest file on a webpage, r
packaging the manifest and icons into a crx file.
Enable

**Lazy Background Pages**
Enables some extension background pages to be loaded when they are needed rather than when t
are first loaded.
Enable

**Override software rendering list**
Overrides the built-in software rendering list and enables GPU-acceleration on unsupported system
Enable

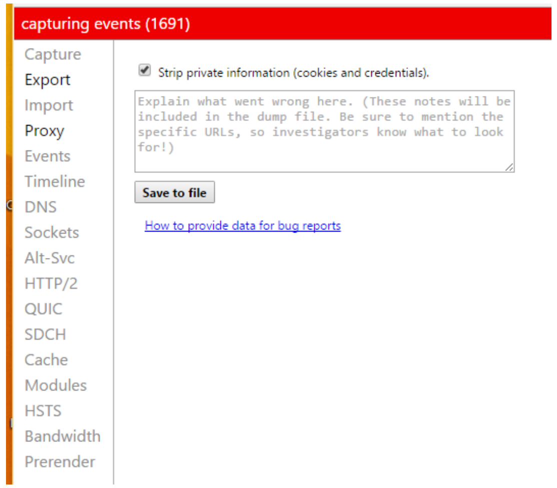
## Chrome(Debug)

```
chrome://crash
chrome://kill
chrome://hang
chrome://shorthang
chrome://gpuclean
chrome://gpucrash
chrome://gpuhang
```

## General Purpose

```
about:memory
about:stats
about:network
about:histograms
about:dns
about:cache
about:crash
about:plugins
about:version
```

## Chrome://net-internals


capturing events (1691)

Capture
Export
Import
Proxy
Events
Timeline
DNS
Sockets
Alt-Svc
HTTP/2
QUIC
SDCH
Cache
Modules
HSTS
Bandwidth
Prerender

✓ Strip private information (cookies and credentials).

```
Explain what went wrong here. (These notes will be included in the dump file. Be sure to mention the specific URLs, so investigators know what to look for!)
```

Save to file

How to provide data for bug reports

## Chrome://Sessions

## Chrome://dns

http://www.perf-tooling.today/tools

https://developers.google.com/web/tools/chrome-devtools/profile/rendering-tools/js-execution?hl=en

https://github.com/felixge/node-measured

https://github.com/mikejihbe/metrics

http://blog.3rd-eden.com/post/5809079469/theoretical-nodejs-real-time-performance

http://www.willvillanueva.com/the-node-js-profiling-guide-that-hasnt-existed-profiling-node-js-applications-part-1/

http://www.willvillanueva.com/the-node-js-profiling-guide-that-hasnt-existed-finding-a-potential-memory-leak-using-memwatch-part-2/

http://www.willvillanueva.com/the-node-js-profiling-guide-that-hasnt-existed-finding-the-cause-of-a-memory-leak-using-heap-snapshots-part-3/

https://www.adremsoft.com/netcrunch/