

Need to Node: Profiling Node.js Applications

Patrick Mueller



NODESOURCE

January 17, 2016

Questions during the Need to Node webinar?

Post a question to Twitter
with the hashtag:
#needtonode



NodeSource is *the* Enterprise Node.js company offering a suite of products and professional services explicitly focused on the needs of Enterprise users of Node.js.



NODESOURCE™



Introduction to profiling Node.js applications

- Explain what profiling applications means, and what kind of profiling is available for Node.js
- Show what insights are provided when profiling Node.js applications
- Demonstrate using the Node.js profiler

What is profiling?



What is profiling?

Profiling is:

- capturing statistics while a program is running
- displaying those statistics with useful visualizations

Provides deep view into application performance

- no more guessing at what your program is doing



What kind of profiling can I do with Node.js?

Profiling Node.js applications

- **V8 heap snapshot profiler** - measures memory usage
 - find memory leaks
 - optimize memory use by your program
- **V8 CPU profiler** - measures function execution time
 - find bottlenecks in your application
 - optimize the performance of your program



What is profiling?

What is the real value in profiling?

- Run your programs faster
- Run your programs with less RAM
- Find difficult to diagnose problems - leaking memory and code bottlenecks

Save \$\$\$!!!



What kind of insights does the heap snapshot profiler provide?



What does the heap snapshot profiler do?

- While your program is running, generates a JSON-able description of all JavaScript objects allocated in your program at a specific point in time, and the references between those objects.
- This description is quite large; plan on 2x RAM usage of your application.
- Visualization shows object counts/sizes by constructor (class), references to and from individual objects.

Heap Snapshot insights - counts/sizes per class (N|Solid)

Constructors				DONE X
FILTER	e.g. Array	OBJECTS ▼	SHALLOW SIZE	RETAINED SIZE
▶ Point2D		41710	1.91 MB	1.91 MB
▶ (system)		9855	426.81 KB	588.29 KB
▶ (array)		5450	2.52 MB	4.63 MB
▶ (string)		4719	1.37 MB	1.37 MB
▶ (compiled code)		4120	1.24 MB	2.41 MB
▶ (closure)		2327	163.62 KB	1.4 MB
▶ Object		829	42.97 KB	7.72 MB



Heap Snapshot insights

- **shallow size vs retained size**
- **shallow size** - amount of memory this object uses just by itself; typically only relevant for Strings and Arrays
- **retained size** - amount of memory this object is referencing that would be garbage collected (GC'd) if this object was garbage collected
- Heap Snapshots typically grouped by constructor name, so literal objects are all lumped into **Object**



Heap Snapshot insights - snapshot diff (Chrome Dev Tools)

Constructor	# New	# Deleted	# Delta	Alloc. Size	Freed Size	Size Delta
▶ Point2D	1 156	0	+1 156	55 488	0	+55 488
▶ (compiled code)	178	598	-420	34 112	352 128	-318 016
▶ (array)	264	821	-557	19 368	81 784	-62 416
▶ (system)	97	626	-529	2 328	17 552	-15 224
▶ (string)	1	185	-184	40	7 688	-7 648
▶ Array	1	1	0	32	32	0
▶ (concatenated string)	0	23	-23	0	920	-920

Object	Distance	Shallow Size	Retained Size

From the time the first snapshot was taken, till the second snapshot was taken, 1156 new Point2D objects were created, and none were garbage collected

Heap Snapshot insights - which objects reference selected object (Chrome Dev Tools)

Constructor	Distance	Objects Count	Shallow Size	Retained Size
► (array)	-	6 177	8%	3 055 784
► system / Context	3	191	0%	20 432
► Map	5	4	0%	128
► (compiled code)	3	4 394	6%	1 412 640
▼ Point2D	7	41 710	57%	2 002 056
► Point2D @8443	11	48	0%	48
► Point2D @8443	11	48	0%	48
► Point2D @8451	11	48	0%	48
► Point2D @8453	11	48	0%	48
► Point2D @8455	11	48	0%	48
Retainers				
Object				
▼ 148280 in [] @139625				
▼ table in Map @130623				
▼ LeakyCache in system / Context @138559				
▼ context in <i>function arunction()</i> @138561				
▼ _repeat in Timeout @137575				
▼ _idlePrev in Timer @793				
▼ [10] in @135185				
▼ lists in system / Context @31049				
▼ context in <i>function ()</i> @21255				
► setInterval in @583				
► value in system / PropertyCell @31131				
► setInterval in @28109				
► 10 in <i>(map descriptors)[1]</i> @132080				

This Point2D object is being referenced by this LeakyCache module variable

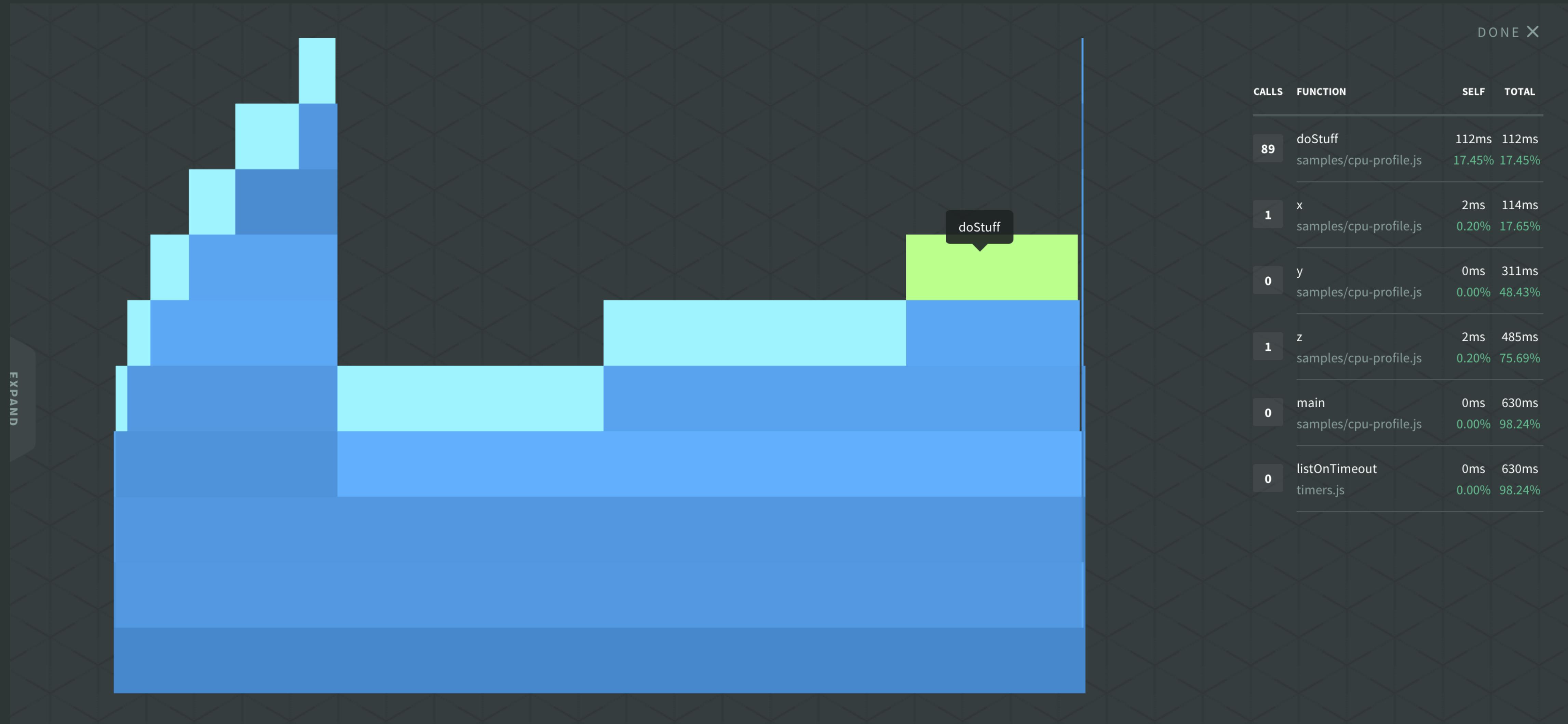
What kind of insights does CPU profiling provide?



What does the CPU profiler do?

- While your program is running, you can start the profiler, let it run for some number of seconds, and then stop the profiler.
- While the profiler is running, it collects the stack of functions being executed, at a sub-millisecond interval.
- When the profiler is stopped, that data is aggregated into a JSON-able object.
- Visualizations show object function call stacks, function execution time, and aggregate function call times.

CPU Profiling insights - flame graph (N|Solid)

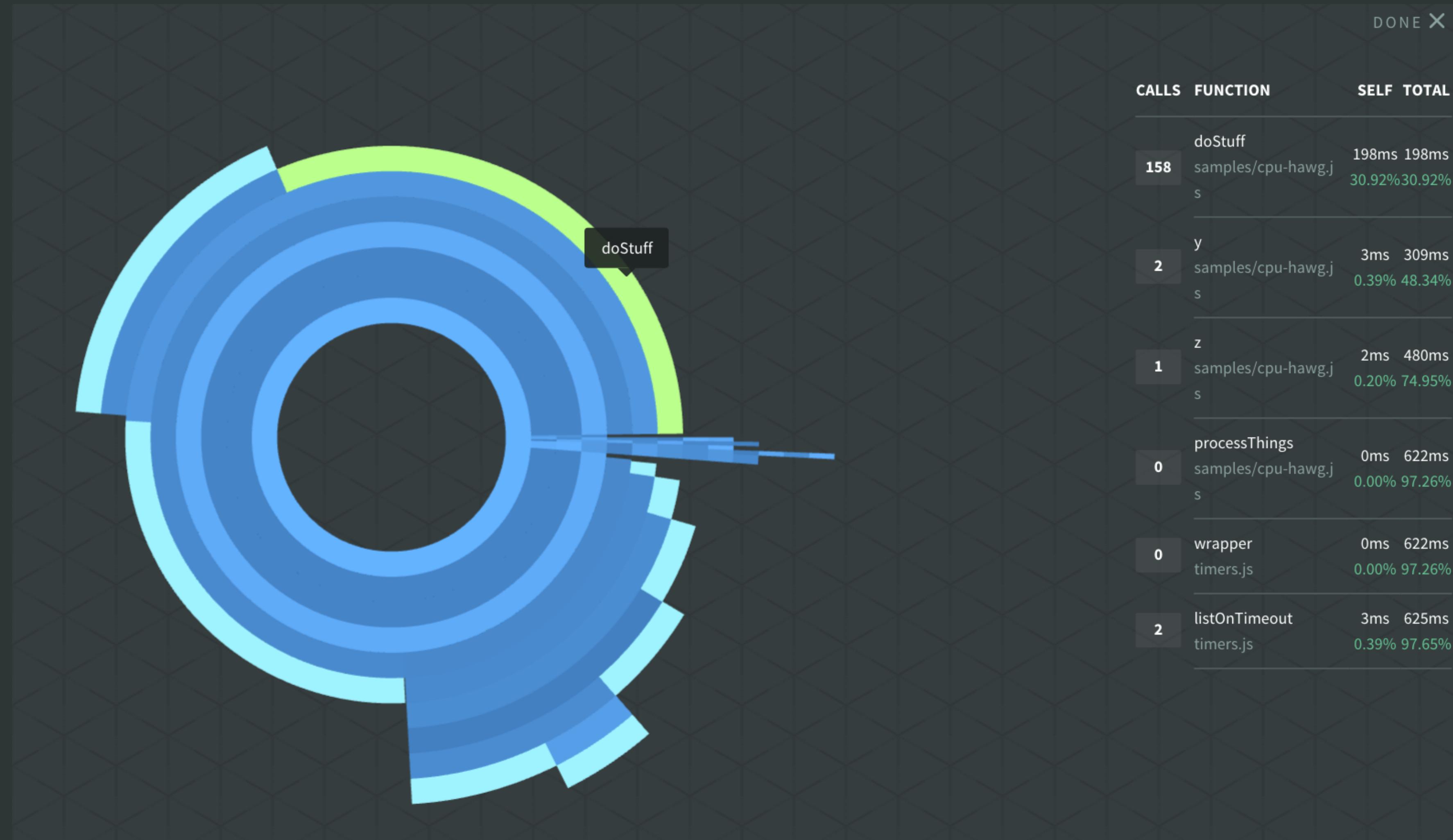


CPU Profiling insights

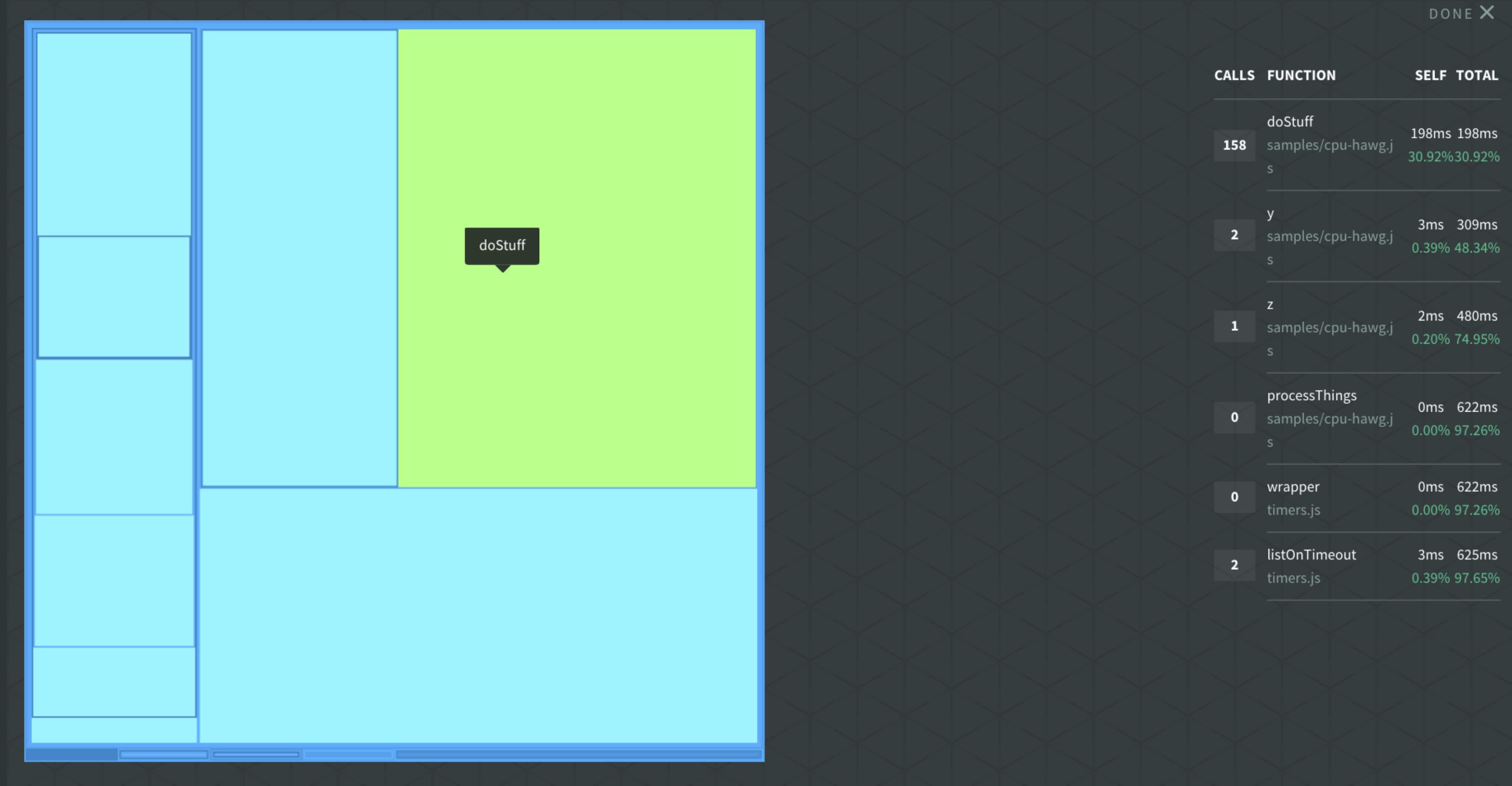
- **total time** vs **self time**
 - **total time** - total elapsed time spent in a function
 - **self time** - total elapsed time spent in a function, minus total time spent in functions called from this function
- **source location** of functions provided in profiling data
- **name your functions**, lest you see lots of functions named “**(anonymous function)**”



CPU Profiling insights - sunburst (N|Solid)



CPU Profiling insights - treemap (N|Solid)



CPU Profiling insights- tablular view of functions (Chrome Dev Tools)

Heavy (Bottom Up) ▾			
Self	Total	Function	
4364.6 ms	4364.6 ms	(idle)	(program):-1
0 ms 0 %	630.0 ms 98.24 %	listOnTimeout	timers.js:55
0 ms 0 %	630.0 ms 98.24 %	▶ main	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:14
626.2 ms 97.65 %	626.2 ms 97.65 %	▶ doStuff	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:30
1.3 ms 0.20 %	485.4 ms 75.69 %	▶ z	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:26
0 ms 0 %	310.6 ms 48.43 %	▶ y	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:27
0 ms 0 %	144.6 ms 22.55 %	▶ a	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:19
0 ms 0 %	137.1 ms 21.37 %	▶ b	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:20
0 ms 0 %	122.0 ms 19.02 %	▶ c	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:21
1.3 ms 0.20 %	113.2 ms 17.65 %	▶ x	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:28
0 ms 0 %	96.8 ms 15.10 %	▶ d	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:22
0 ms 0 %	66.6 ms 10.39 %	▶ e	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:23
1.3 ms 0.20 %	25.1 ms 3.92 %	▶ f	/Users/pmuellr/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:24
5.0 ms 0.78 %	5.0 ms 0.78 %	(program)	(program):-1
2.5 ms 0.39 %	2.5 ms 0.39 %	(garbage collector)	(program):-1
0 ms 0 %	1.3 ms 0.20 %	▶ readableAddChunk	stream_readable.js:122
0 ms 0 %	1.3 ms 0.20 %	processStats	nsolid.js:247
0 ms 0 %	1.3 ms 0.20 %	▶ ChildProcess.spawn	internal/child_process.js:251
0 ms 0 %	1.3 ms 0.20 %	▶ exports.stat	nsolid.js:1165
1.3 ms 0.20 %	1.3 ms 0.20 %	▶ spawn	(program):-1
1.3 ms 0.20 %	1.3 ms 0.20 %	▶ stopProfiling	(program):-1
0 ms 0 %	1.3 ms 0.20 %	▶ exports.spawn	child_process.js:355
1.3 ms 0.20 %	1.3 ms 0.20 %	▶ Readable.read	stream_readable.js:249
0 ms 0 %	1.3 ms 0.20 %	▶ exports.execFile	child_process.js:117
0 ms 0 %	1.3 ms 0.20 %	▶ exports.exec	child_process.js:108
0 ms 0 %	1.3 ms 0.20 %	▶ Readable.push	stream_readable.js:98
0 ms 0 %	1.3 ms 0.20 %	start	nsolid.js:105
0 ms 0 %	1.3 ms 0.20 %	▶ profiler.startProfiling	profiler.js:174
0 ms 0 %	1.3 ms 0.20 %	▶ stats.ps	nsolid.js:1315
0 ms 0 %	1.3 ms 0.20 %	▶ (anonymous function)	nsolid.js:1136
0 ms 0 %	1.3 ms 0.20 %	onread	net.js:500

How do you get profiling information from Node.js applications?



v8-profiler package on npm

- manually instrument your application, load profile data into Chrome Dev Tools

N|Solid from NodeSource

- generate and display profiles at the click of a button

using v8-profiler from npm

- npm install v8-profiler
- add v8-profiler to your package.json dependencies if not running locally
- add code to your app for triggers for starting/stopping CPU profiles, and to generate heap snapshots
 - the triggers will call functions in v8-profiler
 - save results of v8-profiler function calls into JSON files
- run your app and trigger the profiles you want to generate JSON files
- load JSON files into Chrome Dev Tools profiles tab

what is N|Solid?

- N|Solid is a fully compatible Node.js v4.x LTS runtime that has been enhanced to provide additional runtime diagnostics.
- Provides a web-based console to:
 - monitor applications at scale, in production
 - obtain process- and system- specific statistics for an individual application instance
 - obtain and view CPU profiles and heap snapshots at the click of a button

using N|Solid

- Run your app with the N|Solid runtime
- Open the N|Solid Console in a browser, navigate to your application, press the buttons to generate a CPU profile or heap snapshot
- The results are shown in N|Solid Console, and the data available for download for further analysis in Chrome Dev Tools

Demo time!

<https://gist.github.com/pmuellr/2c7e9c7b95352d1b33e0>



Get Involved in Building New Profiling Tools!

follow the Node.js Tracing Work Group

- <https://github.com/nodejs/tracing-wg/issues/38>

write your own profiling visualizers

- CPU profiles and heap snapshots are just JSON!



Download N|Solid

nsrc.io/download-nsolid

free for development use

<https://nodesource.com/blog/getting-started-with-nsolid-at-the-command-line/>

<https://nodesource.com/blog/getting-started-with-the-nsolid-console/>



References

- **Google Developers - “How to Record Heap Snapshots”** - introduction to heap snapshots:
<https://developers.google.com/web/tools/chrome-devtools/profile/memory-problems/heap-snapshots>
- **Google Developers - “Speed Up JavaScript Execution”** - introduction to CPU profiles:
<https://developers.google.com/web/tools/chrome-devtools/profile/rendering-tools/js-execution>
- **v8-profiler package at npm** - open source package exposing V8’s profiling APIs:
<https://www.npmjs.com/package/v8-profiler>



Questions during the Need to Node webcast?

post a question to Twitter
with the hashtag

#needtonode



Thank you.

Patrick Mueller

pmuellr@nodesource.com

@pmuellr



NODESOURCE