Need to Node: Profiling Node.js Applications

Patrick Mueller
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.
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 profilers
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?
Heap Snapshot profiler

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), and references to and from individual objects.
Heap Snapshot insights - counts/sizes per class (N|Solid)

<table>
<thead>
<tr>
<th>Class</th>
<th>Objects</th>
<th>Shallow Size</th>
<th>Retained Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point2D</td>
<td>41710</td>
<td>1.91 MB</td>
<td>1.91 MB</td>
</tr>
<tr>
<td>(system)</td>
<td>9855</td>
<td>426.81 KB</td>
<td>588.29 KB</td>
</tr>
<tr>
<td>(array)</td>
<td>5450</td>
<td>2.52 MB</td>
<td>4.63 MB</td>
</tr>
<tr>
<td>(string)</td>
<td>4719</td>
<td>1.37 MB</td>
<td>1.37 MB</td>
</tr>
<tr>
<td>(compiled code)</td>
<td>4120</td>
<td>1.24 MB</td>
<td>2.41 MB</td>
</tr>
<tr>
<td>(closure)</td>
<td>2327</td>
<td>163.62 KB</td>
<td>1.4 MB</td>
</tr>
<tr>
<td>Object</td>
<td>829</td>
<td>42.97 KB</td>
<td>7.72 MB</td>
</tr>
</tbody>
</table>
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** - total amount of memory this object is referencing that would be garbage collected (GC’d) if this object was garbage collected - usually the more interesting number

- Heap Snapshots are typically grouped by constructor name, so literal objects are all lumped into **Object**
Heap Snapshot insights - snapshot diff (Chrome Dev Tools)

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

Retainers

| 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)

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Distance</th>
<th>Objects Count</th>
<th>Shallow Size</th>
<th>Retained Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>(array)</td>
<td></td>
<td>6177</td>
<td>3055784</td>
<td>5294736</td>
</tr>
<tr>
<td>system / Context</td>
<td>3</td>
<td>191</td>
<td>20432</td>
<td>3957092</td>
</tr>
<tr>
<td>Map</td>
<td>5</td>
<td>4</td>
<td>128</td>
<td>3837664</td>
</tr>
<tr>
<td>(compiled code)</td>
<td>3</td>
<td>4394</td>
<td>1412640</td>
<td>2635360</td>
</tr>
<tr>
<td>Point2D</td>
<td>7</td>
<td>41710</td>
<td>2002056</td>
<td>2002056</td>
</tr>
<tr>
<td>▶ Point2D @8443</td>
<td>11</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
</tbody>
</table>

Retainers

<table>
<thead>
<tr>
<th>Object</th>
<th>Distance</th>
<th>Count</th>
<th>Shallow Size</th>
<th>Retained Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>▼148280 in [] @139625</td>
<td>11</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>▼LeakyCache in system / Context @138559</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>▼context in function () @21255</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>▼setInterval in @583</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>▼value in system / PropertyCell @31131</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>▼setInterval in @28109</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>▼10 in (map descriptors)[] @132202</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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)

**CALLS** | **FUNCTION**          | **SELF** | **TOTAL** |
---|-----------------------|----------|-----------|
89  | doStuff samples/cpu-profile.js | 112ms    | 112ms     |
1   | x samples/cpu-profile.js     | 2ms      | 114ms     |
0   | y samples/cpu-profile.js     | 0ms      | 311ms     |
1   | z samples/cpu-profile.js     | 2ms      | 485ms     |
0   | main samples/cpu-profile.js  | 0ms      | 630ms     |
0   | listOnTimeout timers.js      | 0ms      | 630ms     |
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 - tabular view of functions (Chrome Dev Tools)

<table>
<thead>
<tr>
<th>Self</th>
<th>Total</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>4364.6 ms</td>
<td>630.0 ms</td>
<td>(idle)</td>
</tr>
<tr>
<td>0 ms</td>
<td>0 %</td>
<td>listOnTimeout</td>
</tr>
<tr>
<td>0 ms</td>
<td>0 %</td>
<td>main</td>
</tr>
<tr>
<td>626.2 ms</td>
<td>97.65 %</td>
<td>doStuff</td>
</tr>
<tr>
<td>1.3 ms</td>
<td>485.4 ms</td>
<td>Z</td>
</tr>
<tr>
<td>0 ms</td>
<td>0 %</td>
<td>y</td>
</tr>
<tr>
<td>0 ms</td>
<td>0 %</td>
<td>a</td>
</tr>
<tr>
<td>0 ms</td>
<td>0 %</td>
<td>b</td>
</tr>
<tr>
<td>0 ms</td>
<td>0 %</td>
<td>c</td>
</tr>
<tr>
<td>1.3 ms</td>
<td>113.2 ms</td>
<td>x</td>
</tr>
<tr>
<td>0 ms</td>
<td>0 %</td>
<td>d</td>
</tr>
<tr>
<td>0 ms</td>
<td>0 %</td>
<td>e</td>
</tr>
<tr>
<td>1.3 ms</td>
<td>25.1 ms</td>
<td>f</td>
</tr>
</tbody>
</table>

5.0 ms  | 5.0 ms  | (program) |
2.5 ms  | 2.5 ms  | (garbage collector) |
0 ms    | 0 %    | readableAddChunk |
0 ms    | 0 %    | processStats |
0 ms    | 0 %    | ChildProcess.spawn |
0 ms    | 0 %    | exports.stat |
1.3 ms  | 1.3 ms  | spawn |
1.3 ms  | 1.3 ms  | stopProfiling |
0 ms    | 0 %    | exports.spawn |
1.3 ms  | 1.3 ms  | Readable.read |
0 ms    | 0 %    | exports.executeFile |
0 ms    | 0 %    | exports.exec |
0 ms    | 0 %    | Readable.push |
0 ms    | 0 %    | start |
0 ms    | 0 %    | profiler.startProfiling |
0 ms    | 0 %    | stats.ps |
0 ms    | 0 %    | (anonymous function) |
0 ms    | 0 %    | onread |

(program): -1
	timers.js: 55
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:14
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:30
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:26
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:27
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:19
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:20
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:21
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:28
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:22
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:23
/Users/pmueller/Projects/slides/2016/01-intro-to-profiling/samples/cpu-profile.js:24

(program): -1
	internal/child_process.js:247

nslok.js:247

nslok.js:251

nslok.js:1165

nslok.js:117

child_process.js:355

stream_readable.js:249

child_process.js:117

child_process.js:108

stream_readable.js:98

net.js:500
How do you get profiling information from Node.js applications?
Profiling tools

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
Profiling with v8-profiler

**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
- load JSON files into the Profiles tab of Chrome Dev Tools
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 the N|Solid Console, and the data is available to download for further analysis in Chrome Dev Tools
Demo time!

sample applications being profiled:
https://gist.github.com/pmueller/2c7e9c7b95352d1b33e0
Get involved in building new profiling tools!

**follow the Node.js Tracing Work Group**

- [https://github.com/nodejs/tracing-wg/issues/38](https://github.com/nodejs/tracing-wg/issues/38)

**write your own profiling visualizers**

- CPU profiles and heap snapshots are just JSON!
N|Solid references

- **Download N|Solid - free for development:**
  https://nodesource.com/products/nsolid

- **N|Solid documentation:**
  https://docs.nodesource.com/

- **Getting Started with N|Solid:**

- **Getting Started with the N|Solid Console:**
  https://nodesource.com/blog/getting-started-with-the-nsolid-console/
V8 profiling 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

down these slides at pmuellr.github.io/slides
Thank you.

Patrick Mueller
pmuellr@nodesource.com
@pmuellr