The Future of Source Maps

June 3rd, 2024
This talk

1. History of source maps
2. Anatomy of a source map
3. What is (and isn’t encoded)
4. How do debuggers work (regex, babel)
5. New proposals
   a. Scopes proposal
   b. Debug IDs
   c. Range mappings
6. Get involved!
About me

- Jon Kuperman
- Engineer at Bloomberg
- TC39
- Co-convenor of the source map task group

Sorry, this is a million years old!
When were source maps created?

The official Google Code blog
Get the latest updates on Google APIs and developer tools.

Thursday, November 05, 2009

Introducing Closure Tools

Millions of Google users worldwide use JavaScript-intensive applications such as Gmail, Google Docs, and Google Maps. Like developers everywhere, Googlers want great web apps to be easier to create, so we've built many tools to help us develop these (and many other) apps. We're happy to announce the open sourcing of these tools, and proud to make them available to the web development community.
Why do we need source maps?
What are source maps?

- JSON objects
- Link between generated code and source code
- Created by “generators” (esbuild, webpack, SWC)
- Used by “consumers” (Chrome, Firefox, Replay.io)
- Also used by “error monitoring tools” (Sentry)
Anatomy of a source map

```json
{
    "version": 3,
    "sources": ["index.js"],
    "sourcesContent": [
        "function greet(name) {
            let message = "Hello, \" + name;\n            console.log(message);
        };
greet("World");"
    ],
    "mappings": "AAAA,SAASA,IAAK,GAAG;AACH,MAAMC,GAAI,GAAG,SAASC,GAAI;AACxB,C,YAAY,CAACC,GAAG,CAAC;AAGtB,CAAC,
    }
```
How do generators work?
export function add(first, second) {
    return first + second;
}
import { add } from "./file1.js";

add();

debugger;
(()=>{function r(d,e){return d+e}r();debugger;})();
//# sourceMappingURL=bundle.js.map
{
  "version": 3,
  "sources": ["file1.js", "index.js"],
  "sourcesContent": ["export function add(first, second) {
  return first +
  second;\n}\n", "import { add } from "./file1.js";\n\nadd();\n"
],
  "mappings": "MAAO,SAASA,EAAIC,EAAOC,EAAQ,CACjC,OAAOD,EAAQC,CACjB,CCAAC,EAAI,EACJ",
  "names": ["add", "first", "second", "add"]
}
Mappings

**VLQ encoding** is used in source maps to efficiently encode integers. Each segment in the mappings string is VLQ-encoded. Here's a quick rundown of how it works:

- For each token in the generated file
- Capture its line and column (relative to the last generated token)
- Capture the source files index in the sources array
- Capture the line and column for the matching source file (relative to the last source token)
- A => 0 - The 0th column offset from the last generated token (aka this is the first mapping!)
- A => 0 - The 0th file in the `sources` array
- A => 0 - The 0th line offset from the last processed source token
- A => 0 - The 0th column offset from the last processed source token
How do debuggers work?

```javascript
export function add(first, second) {
    debugger;
    return first + second;
}
```
How do debuggers work?

```javascript
export function multiply(first, second) {
  return first * second;
}
```
How do debuggers work?

```javascript
import { add } from './file1.js';
import { multiply } from './file2.js';

add(2, 2);
multiply(3, 3);
```
How do debuggers work?

```javascript
const esbuild = require("esbuild");

esbuild
  .build({
    entryPoints: ["index.js"],
    bundle: true,
    minify: true,
    outfile: "bundle.js",
  })
  .catch(() => process.exit(1));
```
How do debuggers work?

```javascript
(()=>{function o(r,t){debugger;return r+t}function u(r,t){return r*t}o(2,2);u(3,3);})();
```
function o(r, t) {
  debugger; return r + t
}

function u(r, t) {
  return r * t
}

o(2, 2);
u(3, 3);

();

Line 1, Column 23
How do debuggers work?
How do debuggers work?

```javascript
(()=>{function o(r,t){debugger;return r+t}function u(r,t){return r*t}o(2,2);u(3,3);})();
//# sourceMappingURL=bundle.js.map
```
How do debuggers work?

```javascript
{  
  "version": 3,  
  "sources": ["file1.js", "file2.js", "index.js"],  
  "sourcesContent": ["export function add(first, second) {\n    debugger;\n    return first + second;\n}\n", "import { add } from "/file1.js";\nimport { multiply } from  
  "mappings":  
  "MAAO,SAASA,EAAIC,EAAOC,EAAQ,CACjC,SACA,OAAOD,EAAQC,CACjB,CCHO,SAASC,EAASC,EAAOC,EAAQ,CACtC,0  
  "names": ["add", "first", "second", "multiply", "first", "second", "add", "multiply"]
}
```
```javascript
export function add(first, second) {
  debugger;
  return first + second;
}
```
```javascript
export function add(first, second) {
  debugger;
  return first + second;
}
```
Consumers try to offer great experiences

- Chrome Devtools uses Regular Expressions
- Firefox runs Babel on the source files
- Scopes are hard to implement!
Stack traces are even harder…
Bloomberg’s Pasta Source Maps

```javascript
// sample.js
const penne = () => { throw Error(); }
const spaghetti = () => penne();
const orzo = () => spaghetti();
orzo();
```

```javascript
// **original** output
Error
  at penne (sample.js:2:33)
  at spaghetti (sample.js:3:25)
  at orzo (sample.js:4:25)

// **compiled** output
Error
  at r (out.js:1:82)
  vs
  at o (out.js:1:97)
  at n (out.js:1:107)
```
It’s not just names

```javascript
// edit tser options
{
  module: true,
  compress: {},
  mangle: {},
  output: {},
  parse: {},
}

function foo() {
  bar();
}

function bar() {
  baz();
}

function baz() {
  console.log(10);
  debugger;
}

foo();
```
Difficult to move forward

Add the `x_google_ignoreList` source map extension

Goals
1. This will add clarity to the Chrome console.
2. It will be easier to debug the application.
3. If this gets added, the issue within the code will be easier to track.

Non-Goals

The ignoreList source map extension

Improve debugging experience in Chrome DevTools with the `ignoreList` source map extension.
The Future of Source Maps

- Formed a task group underneath the TC39 umbrella
- Improve the specification
- Embed scope information
- Embeds function and variable names
- Add Debug IDs
- Add Range Mappings
Scopes Proposal

- **Inline Functions**: Reconstruct and step through inlined functions.
- **Variable Mapping**: Map renamed/erased variables back to original names.
- **Scope Reconstruction**: Rebuild original and hidden scopes.
- **New Fields**:
  - `originalScopes`: Describes original code scopes.
  - `generatedRanges`: Describes generated code scopes and bindings.
Debug IDs

- Multiple source maps
- Outdated source maps
- Stack traces

```javascript
TypeError: Cannot read property 'length' of undefined
at app.min.js:2:4567
debugId: 85314830-023f-4cf1-a267-535f4e37bb17
```
Range Mappings

Current participants

- Bloomberg
- Google
- JetBrains
- Meta
- Microsoft
- Mozilla
- Replay.io
- Sentry
- And more!
Come get involved!

- **Already a TC39 member?**
  - Find our events on the TC39 calendar
  - Read out CONTRIBUTING guide

- **Not a TC39 member yet?**
  - Join our matrix chat and message me (jkup) - I’ll help you get involved!
Thank you!