October 4-6, 2017 | Vancouver, BC

new debugger features

Yang Guo, Software Engineer, *Google*

@hashseed

node JS Interactive
October 4-6, 2017 | Vancouver, BC

DevTools protocol

code coverage

type profile

debug-evaluate

node Interactive

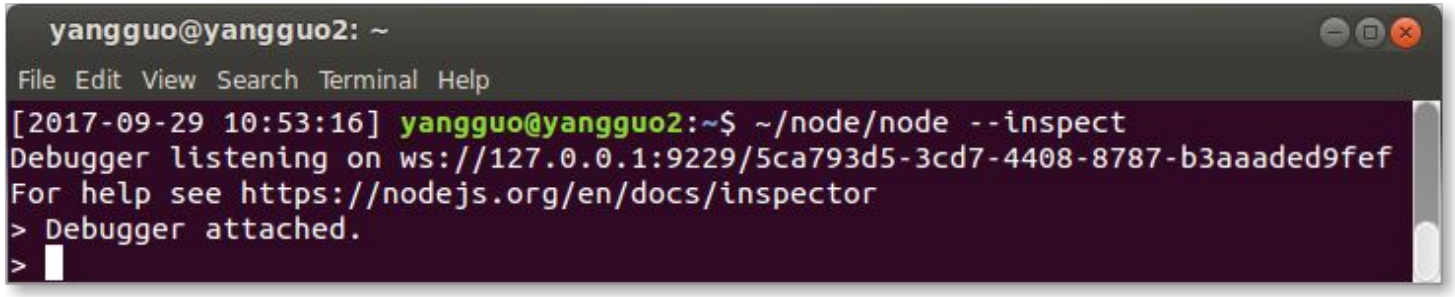
October 4-6, 2017 | Vancouver, BC

LEGACY V8 DEBUG API

JSON message API	DebugCommandProcessor
debug context JavaScript API	--expose-debug-as, vm.runInDebugContext
C++ API	v8/include/v8-debug.h

CURRENT V8 DEBUG API

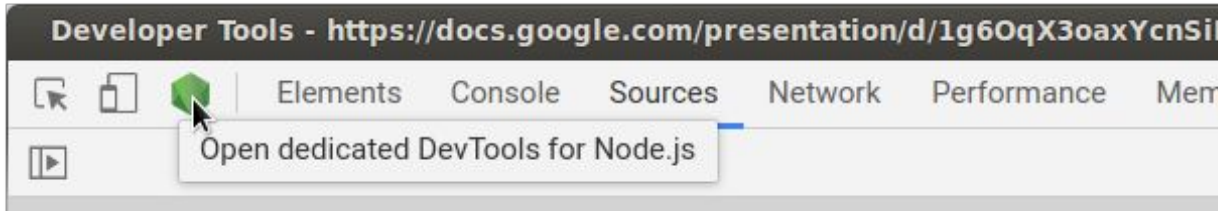
JSON message API	DebugCommandProcessor
debug context JavaScript API	--expose-debug-as, vm.runInDebugContext
C++ API	v8/include/v8-debug.h
DevTools protocol	--inspect



node Interactive
October 4-6, 2017 | Vancouver, BC

DEVTOOLS PROTOCOL IN V8

Chrome with Chrome DevTools	F12, CTRL+SHIFT+J
Node.js with Chrome DevTools	--inspect
Chrome with Node.js	github.com/GoogleChrome/puppeteer
Node.js with Node.s	require('inspector')
Node.js with telnet	;-)




```
{
  "id": 1,
  "method": "Debugger.setBreakpoint",
  "params": {
    "location": {
      "scriptId": "16",
      "lineNumber": 25,
      "columnNumber": 0
    }
  }
}
```

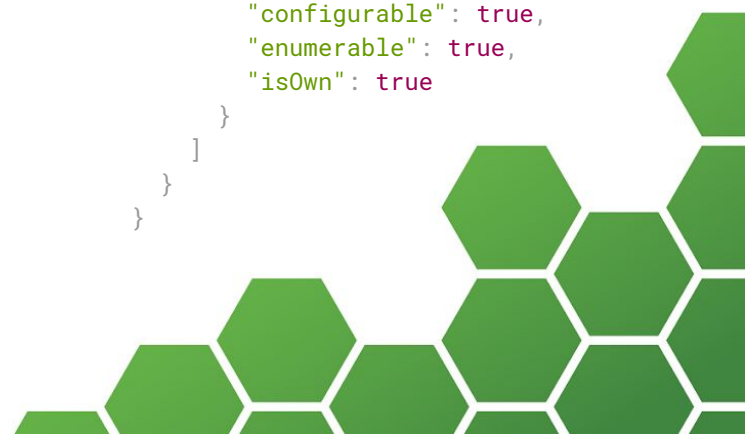
```
{
  "id": 1,
  "result": {
    "breakpointId": "16:25:0",
    "actualLocation": {
      "scriptId": "16",
      "lineNumber": 25,
      "columnNumber": 2
    }
  }
}
```

```
{
  "id": 0,
  "method": "Debugger.enable"
}
```

```
{
  "method": "Debugger.scriptParsed",
  "params": {
    "scriptId": "14",
    "url": "test/mjsunit/mjsunit.js",
    "startLine": 0,
    "startColumn": 0,
    "endLine": 826,
    "endColumn": 0,
    "executionContextId": 1,
    "hash":
"8A11A530A7BFCC85AF929B31548DDBF56FF75175
    "isLiveEdit": false,
    "sourceMapURL": "",
    "hasSourceURL": false,
    "isModule": false,
    "length": 27412,
    "stackTrace": {
      "callFrames": [
        {
          "functionName": "sendMessage",
          "scriptId": "15",
          "url":
"test/debugger/test-api.js",
          "lineNumber": 360,
          "columnNumber": 4
        }
      ]
    }
  }
}
```

```
{
  "method": "Debugger.paused",
  "params": {
    "callFrames": [
      {
        "callFrameId":
"{\"ordinal\":0,\"injectedScriptId\":1}",
        "functionName": "f",
        "functionLocation": {
          "scriptId": "16",
          "lineNumber": 24,
          "columnNumber": 10
        },
        "location": {
          "scriptId": "16",
          "lineNumber": 25,
          "columnNumber": 2
        },
        "url": "test.js",
        "scopeChain": [
          {
            "type": "local",
            "object": {
              "type": "object",
              "className": "Object",
              "description": "Object",
              "objectId": "{\"injectedScriptId\ ...,  ,:1}"
            },
            "name": "f",
            "startLocation": {
              "scriptId": "16",
              "lineNumber": 24,
              "columnNumber": 10
            },
            "endLocation": {
              "scriptId": "16",
              "lineNumber": 26,
              "columnNumber": 1
            }
          },
          {
```

```
{
  "id": 3,
  "result": {
    "result": [
      {
        "name": "x",
        "value": {
          "type": "number",
          "value": 1,
          "description": "1"
        },
        "writable": true,
        "configurable": true,
        "enumerable": true,
        "isOwn": true
      },
      {
        "name": "y",
        "value": {
          "type": "undefined"
        },
        "writable": true,
        "configurable": true,
        "enumerable": true,
        "isOwn": true
      }
    ]
  }
}
```



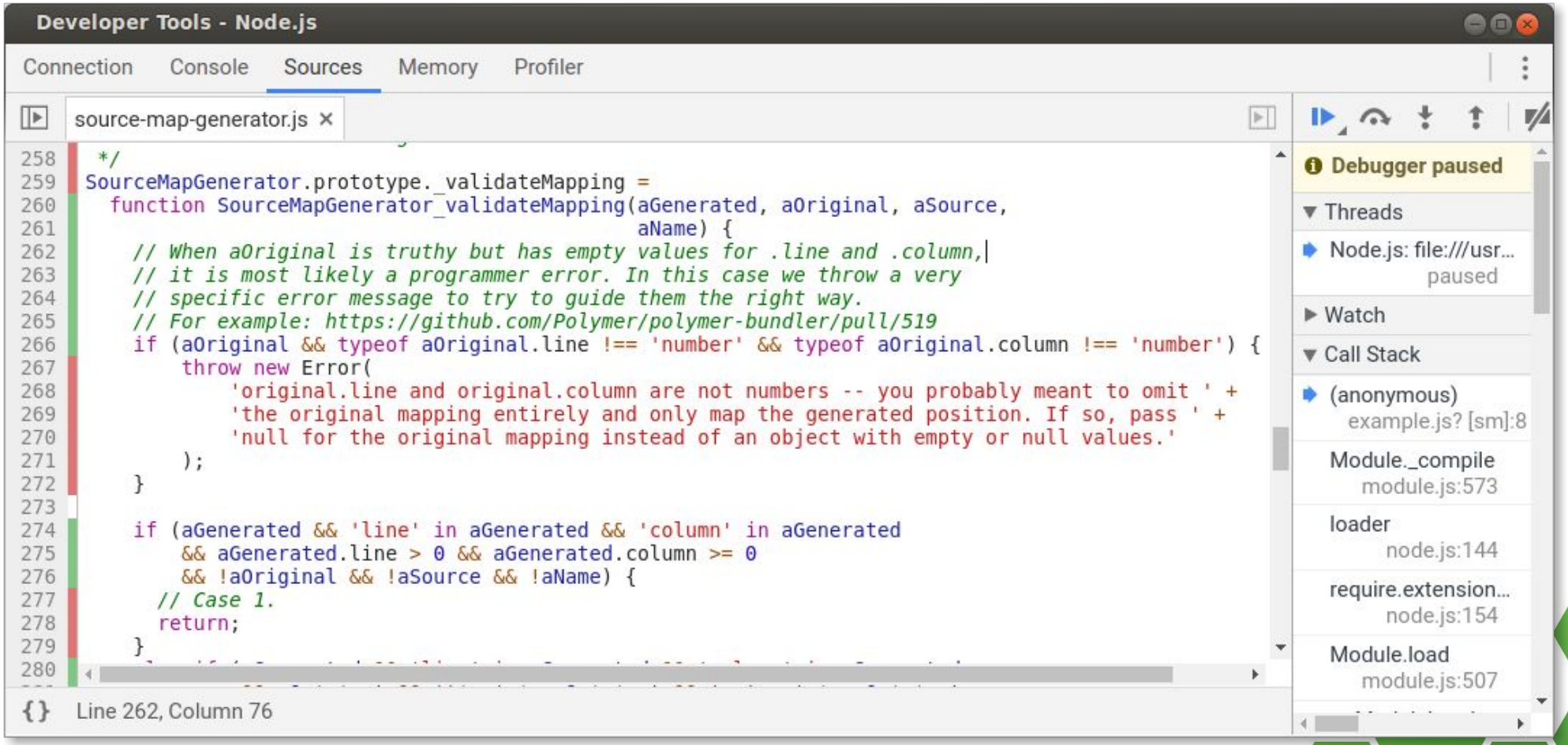
INSPECTOR IN NODE

```
const inspector = require('inspector');
const session = new inspector.Session();
session.connect();
session.post(command, arguments, callback);
session.disconnect();
```

node Interactive

October 4-6, 2017 | Vancouver, BC

CODE COVERAGE



CODE COVERAGE

branch-based code coverage

accurate call counts

accessible via DevTools protocol

can be translated to common coverage formats, e.g. LCOV

	Line data	Source code
1	1 :	function thrower() {
2	0 :	throw new RangeError();
3	1 :	}
4	177 :	function fib(n) {
5	177 :	if (n == 0) {
6	34 :	return 0;
7	177 :	} else if (n == 1) {
8	55 :	return 1;
9	143 :	} else if (n > 1) {
10	88 :	return fib(n - 1) + fib(n - 2);
11	0 :	} else {
12	0 :	thrower();
13	177 :	}
14	177 :	}
15	1 :	console.log("fib(10):", fib(10));
16	1 :	

Coverage report generated by LCOV

CODE COVERAGE

script

```
function fib(x) {
  if (x < 2) {
    return 1;
  }
  return fib(x-1) + fib(x-2);
}

function dead() {
  unreachable;
}

var failed = false;
try {
  fib(8);
} catch (e) {
  failed = true;
}
```

☑ Call count ☑ Detailed coverage

obtain coverage

coverage

```
function fib(x) {
  if (x < 2) {
    return 1;
  }
  return fib(x-1) + fib(x-2);
}

function dead() {
  unreachable;
}

var failed = false;
try {
  fib(8);
} catch (e) {
  failed = true;
}
```

PROTOCOL FOR COVERAGE

Profiler.startPreciseCoverage

Enable precise code coverage. Coverage data for JavaScript executed before enabling precise code coverage may be incomplete. Enabling prevents running optimized code and resets execution counters. EXPERIMENTAL

PARAMETERS

`callCount`	**boolean**	Collect accurate call counts beyond simple 'covered' or 'not covered'.
optional		
`detailed`	**boolean**	Collect block-based coverage.
optional		

Profiler.stopPreciseCoverage

Disable precise code coverage. Disabling releases unnecessary execution count records and allows executing optimized code. EXPERIMENTAL

Profiler.takePreciseCoverage

Collect coverage data for the current isolate, and resets execution counters. Precise code coverage needs to have started. EXPERIMENTAL

RETURN OBJECT

`result`	**array [** ScriptCoverage **]**	Coverage data for the current isolate.

PROTOCOL FOR COVERAGE

```
session.post("Profiler.enable", arguments, callback);
...
session.post("Profiler.startPreciseCoverage",
             { callCount: true, detailed: true},
             callback);
...
session.post("Profiler.takePreciseCoverage");
```

TYPE PROFILE

script

```
function add(left,
             right) {
  return left + right;
}

class Potato {
  valueOf() { return 3; }
}
class Tomato {
  toString() { return "T"; }
}

console.log(add(1, 2));
console.log(add(1, "2"));
console.log(add(new Potato, new Tomato));
```

obtain type profile

type profile

```
function add(number Potato left,
             number string Tomato right) {
  return left + right;
number string }

undefined class Potato {
  valueOf() { return 3; number }
}
undefined class Tomato {
  toString() { return "T"; string }
}

console.log(add(1, 2));
console.log(add(1, "2"));
console.log(add(new Potato, new Tomato));
```

PROTOCOL FOR TYPE PROFILE

Profiler.startTypeProfile

Enable type profile. EXPERIMENTAL

Profiler.stopTypeProfile

Disable type profile. Disabling releases type profile data collected so far. EXPERIMENTAL

Profiler.takeTypeProfile

Collect type profile. EXPERIMENTAL

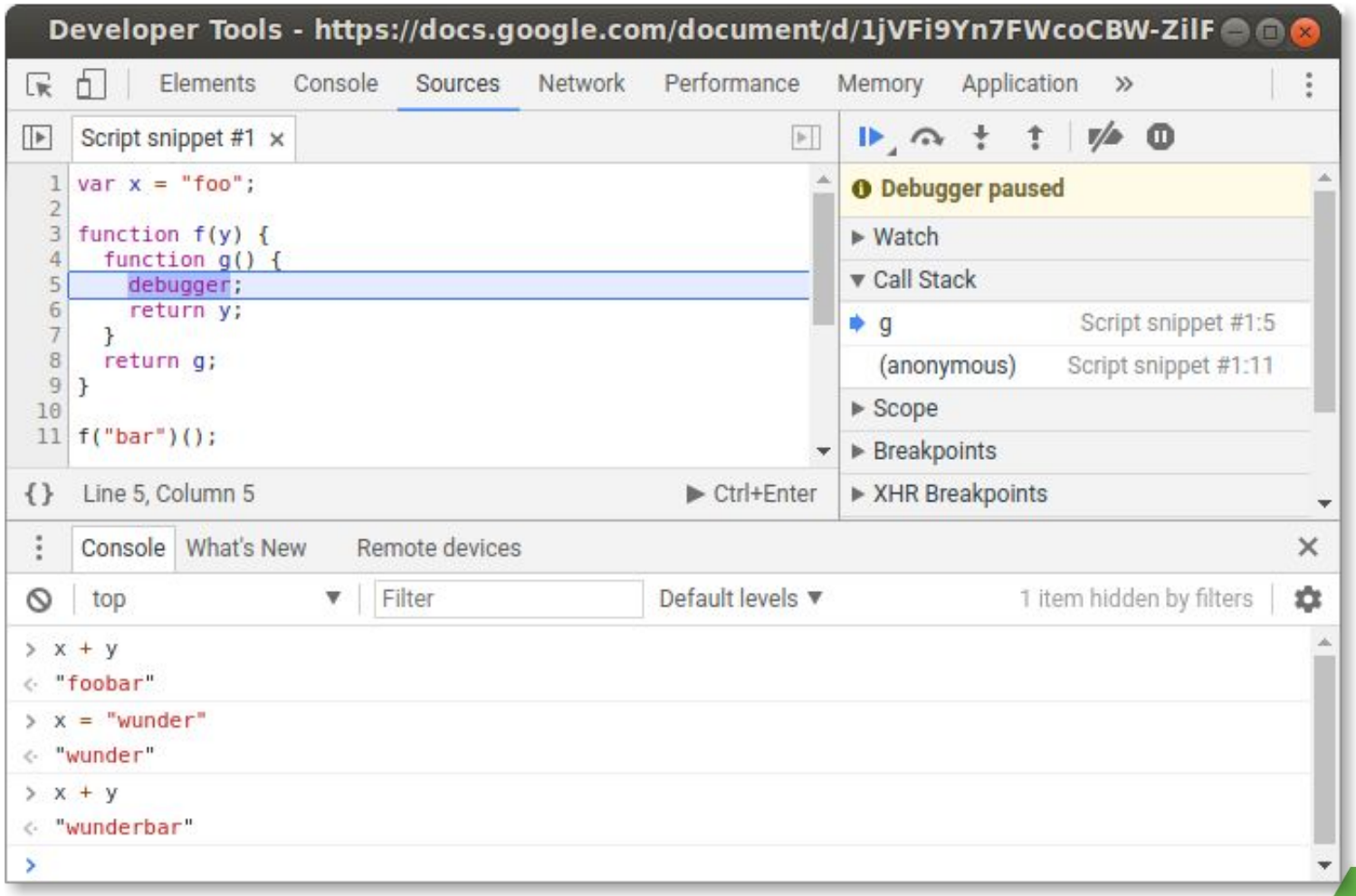
RETURN OBJECT

result	**array [ScriptTypeProfile]** Type profile for all scripts since startTypeProfile() was turned on.

PROTOCOL FOR TYPE PROFILE

```
session.post("Profiler.enable", arguments, callback);
...
session.post("Profiler.startTypeProfile", callback);
...
session.post("Profiler.takeTypeProfile");
```

DEBUG-EVALUATE



DEBUG-EVALUATE

script

```
function f(x) {
  return function g() {
    debugger;
    console.log(x);
    return x;
  }
}

f(2)();
```

evaluate at debugger statement

```
x = 3
```

run ☑ allow side effect

result

3

console

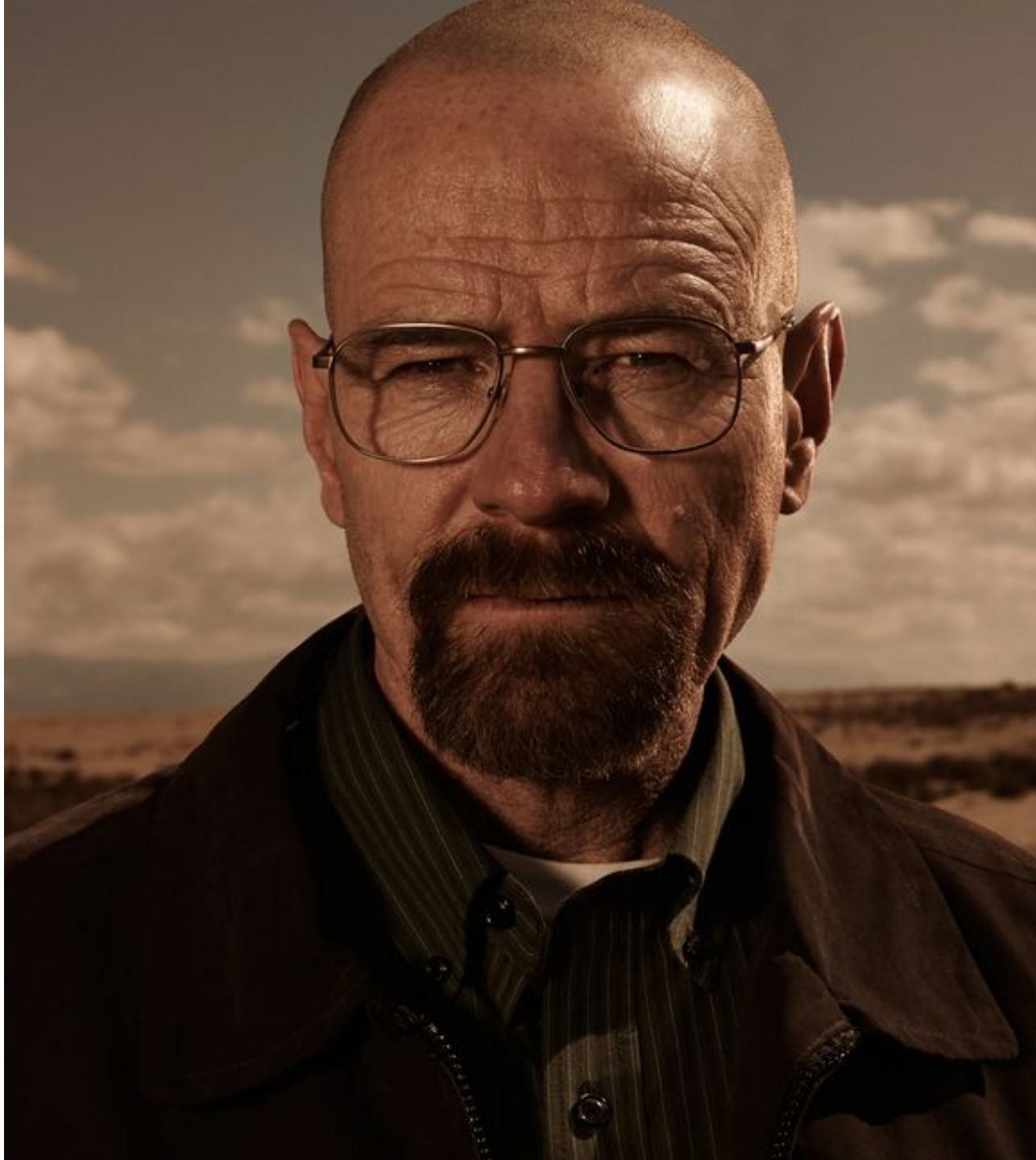
```
console.log: 3
```

node JS Interactive

October 4-6, 2017 | Vancouver, BC

"Looking at something changes it"

- Werner Heisenberg



DEBUG-EVALUATE W/O SIDE-EFFECTS

script

```
function f(x) {
  return function g() {
    debugger;
    console.log(x);
    return x;
  }
}

f(2)();
```

evaluate at debugger statement

```
x = 3
```

run ☐ allow side effect

result

[exception]

console

console.log: 2

PROTOCOL FOR DEBUG-EVALUATE

`Debugger.evaluateOnCallFrame`

Evaluates expression on a given call frame.

PARAMETERS

`callFrameId`	**CallFrameId** Call frame identifier to evaluate on.
`expression`	**string** Expression to evaluate.
`objectGroup` optional	**string** String object group name to put result into (allows rapid releasing resulting object handles using `releaseObjectGroup`).
`includeCommandLineAPI` optional	**boolean** Specifies whether command line API should be available to the evaluated expression, defaults to false.
`silent` optional	**boolean** In silent mode exceptions thrown during evaluation are not reported and do not pause execution. Overrides `setPauseOnException` state.
`returnByValue` optional	**boolean** Whether the result is expected to be a JSON object that should be sent by value.
`generatePreview` optional	**boolean** Whether preview should be generated for the result. EXPERIMENTAL
`throwOnSideEffect` optional	**boolean** Whether to throw an exception if side effect cannot be ruled out during evaluation. EXPERIMENTAL

RETURN OBJECT

`result`	**Runtime.RemoteObject** Object wrapper for the evaluation result.
`exceptionDetails` optional	**Runtime.ExceptionDetails** Exception details.

PROTOCOL FOR DEBUG-EVALUATE

`throwOnSideEffect` **boolean**
optional
Whether to throw an exception if side effect cannot be ruled out during evaluation. EXPERIMENTAL

```
session.post('Debugger.evaluateOnCallFrame',
             { callFrameId,
               expression,
               throwOnSideEffect },
             callback);
```

LINKS

- Node.js with newest V8
  https://github.com/v8/node/commits/vee-eight-lkgr
- DevTools protocol documentation
  https://chromedevtools.github.io/debugger-protocol-viewer/v8/
- demo files
  https://github.com/hashseed/node-coverage-demo