var title = “Diving into ES2015 proxy performance in V8”;

```
var info = {
    name: “Maya Lekova”,
    email: “lekova.maya@gmail.com”
};
```

jQuery SOFIA

Sofia js.talks();

Nov 18, 2017

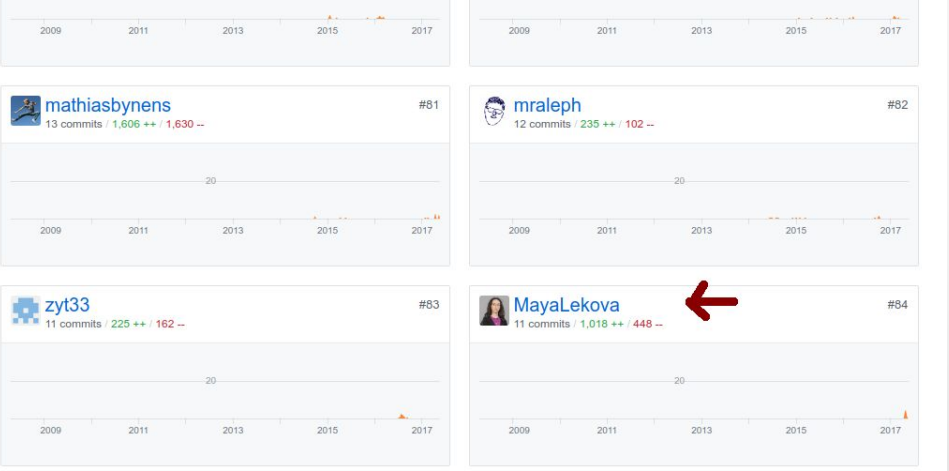
agenda();

- Who am I
- What are ES2015 proxies?
- Why should you use proxies?
- Proxies' implementation in V8
- Performance optimizations
- Conclusion & future plans

Speaker

Maya Lekova

- Ex-intern on the V8 team at Google
- Master student in E-Learning
- Interested in C++ and JS
- Background in the game industry
- You can find me
  - on Twitter **@MayaLekova** <or>
  - at the bottom of V8's contributors



js.talks();

Nov 18, 2017

What are ES2015 proxies?

Definition of “proxy”

“ES2015 Proxies provide JavaScript with an intercession API, enabling us to trap or intercept all of the operations on a target object and modify how this target operates” – Addy Osmani

Definition of “trap”

“The methods that provide property access. This is analogous to the concept of traps in operating systems.” *

- According to MDN’s page on Proxies

js.talks();

General syntax

```
const target = { /* some properties */ };
 const handler = { /* trap functions */ };
 const proxy = new Proxy(target, handler);
```

Available trap functions

- apply()
- construct()
- defineProperty()
- deleteProperty()
- get()
- getOwnPropertyDescriptor()
- getPrototypeOf()
- has()
- isExtensible()
- ownKeys()
- preventExtensions()
- set()
- setPrototypeOf()

When should you use proxies?

Possible use cases

- Interception (property observation)
- Object virtualization
- Resource management
- Profiling or logging for debugging (object extension)
- Validation, security and access control (membranes, think of access to file system from a web page)
- Contracts for object use, schema validation

Property observation

```
const target = {};
const call_tracer = new Proxy(target, {
   get: (target, name, receiver) => {
       console.log(`get was called for:
 ${name}`);
       return target[name];
   }
});

call_tracer.property = 'value';
console.log(call_tracer.property);
// get was called for: property
// 'value'
```

Library nx-js/observer-util

- https://github.com/nx-js/observer-util
- Usage:

```
import { observable, observe } from
 '@nx-js/observer-util';
 const person = observable({ name: 'Bob', age:
 25 });
 observe(() => console.log(`${person.name} is
 ${person.age}`));

// this logs 'John is 25' to the console
 setTimeout(() => { person.name = 'John'; });
```

Object virtualization

- “Virtual objects are proxies that emulate other objects without those objects needing to be in the same address space.”
- Remote objects (emulating objects in other spaces)
- Transparent futures (emulating results that are not yet computed)

Transparent futures

```
const handler = {
  get: (target, name, receiver) => {
   const p = this;
   if (name == 'authenticated_user')
  {
    target.get_user.then((user) => {
      p.authenticated_user = user;
    }).catch((error) => {
      console.error('Network error');
       throw error;
    });
    return this.authenticated_user;
  } else {
    return target[name];
  }}};
```

```
const dog_user = {
  name: 'Dog'
};
const network = {
  get_user: new
 Promise((resolve, reject)
 => {
    // Simulate async
 operation
    setTimeout(() => {
      resolve(dog_user);
    }, 500);
  })
};
```

Transparent futures (continued)

```
const proxy = new Proxy(network,
                        handler);
(function poll_for_user() {
  const t = setInterval(() => {
   const user =
  proxy.authenticated_user;
   console.log('Polling…');
   if (user !== undefined) {
     clearInterval(t);
     console.log('Got authenticated
  user; name:', user.name);
     // Continue work
   }
  }, 100);
})();
```

```
// Output
Polling…
Polling…
Polling…
Polling…
Polling…
Got authenticated
  user; name: Dog
```

js.talks();

Property validation

```
const validator = {
  set: (obj, prop, value) => {
    if (prop === 'month') {
      if
  (!Number.isInteger(value)) {
        throw new TypeError(...);
      }

      if (value < 0 || value > 11)
  {
        throw new RangeError(...);
      }
    }

    obj[prop] = value;
  }
};
```

```
const date = new Proxy({},
  validator);

date.month = 6;
console.log(date.month); // 6
date.month = 'January'; //
  Throws an exception
date.month = 25; // Throws an
  exception
```

js.talks();

Access control

```
function
  create_simple_membrane(target) {
    let enabled = true;
   function wrap(obj) {
      if (obj !== Object(obj))
  return obj;
      const handler = new Proxy({},
  {get: (_, key) => {
        if (!enabled) throw new
  Error('disabled');
        switch (key) {
        case 'apply': {...}
        case 'construct': {...}
        default:
          return (_, ...args) => {
            try {
              return
  wrap(Reflect[key](obj,
  ...(args.map(wrap))));
```

```
            } catch (exception) {
                throw
  wrap(exception);
      }}}}});
      return new Proxy(obj,
  handler);
    }
   const gate = Object.freeze({
      enable: () => enabled =
  true,
      disable: () => enabled =
  false
    });
   return Object.freeze({
      wrapper: wrap(target),
      gate: gate
    });
  }
```

Access control – usage

```
// File system API
  const fs = {
    read_file(name) {
      console.log('Reading
  file', name);
    }
  }
const membrane =
  create_simple_membrane(fs);

  // Using the API from
  unsafe code
  membrane.wrapper.read_file(
  'foo.txt');

  // Reading file foo.txt
```

```
membrane.gate.disable();

  try {

  membrane.wrapper.read_file('bar.
  txt');
  } catch(err) {
    console.error('Error while
  reading file;', err);
  }
// Error while reading file;
  Error: disabled
```

js.talks();

Contracts

```
// contract wrapper
  implementation
  function
  check_predicate(pred) {
    return {
      set: (target, prop, val)
  => {
        if (!pred (val)) {
  throw new
  ContractException(); };
        target[prop] = val;
      }}};

  function
  assert_contract(target, pred)
  {
    return new Proxy(target,
      check_predicate(pred));
  }
```

```
// application code
  function modify(acc1, acc2,
  amount) {
    acc1.balance += amount;
    acc2.balance += amount;
  }
let account = { balance: 10 };
  let restricted =
  assert_contract(account, (x) =>
  x >= 0);
  modify(restricted, account, 40);
  // += 80
  console.log(account.balance); //
  90
  modify(restricted, account,
  -80); // -= 160
   // ContractException
```

Proxies' implementation in V8

What is V8?

- Open-source JavaScript engine
- Google Chrome and Node.js run on top of it
- Written in C++ and JavaScript
- More than 1 million lines of code
- Implements ECMAScript as specified in ECMA-262
- Runs on
  - Windows 7 or later
  - macOS 10.5+
  - Linux systems that use IA-32, ARM or MIPS processors
- https://github.com/v8/v8

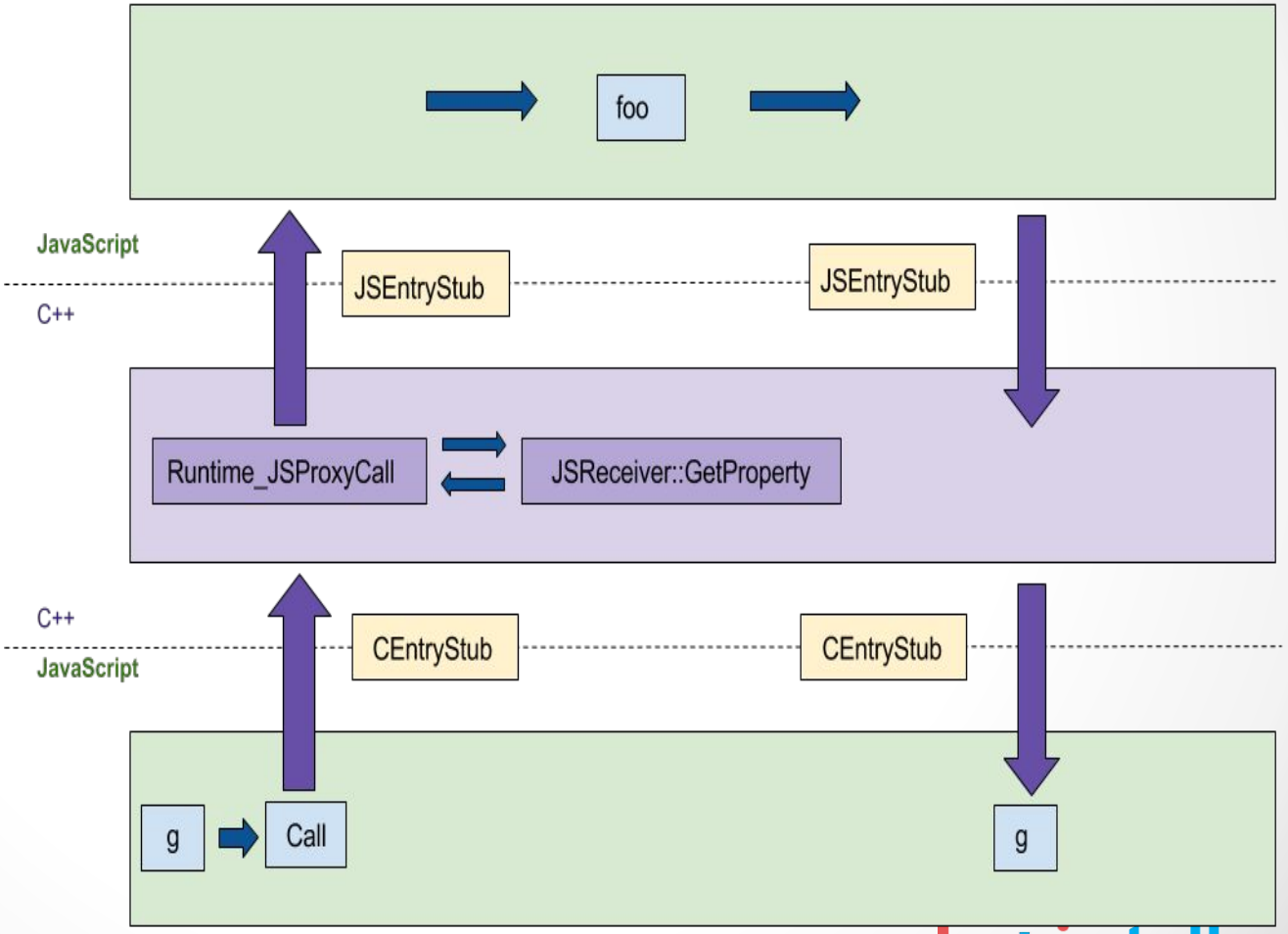
The two worlds

- Wrapping in proxies is generally an overhead
- C++ Runtime
- CSA (Code Stub Assembler - platform-agnostic macro assembler), executed in the JS runtime
- Expensive jumps between them

```
function foo(...) {...}

g = new Proxy({...}, {
  apply: foo
});
g(1, 2);
```

Proxy's [[Call]] internal method (before)

CEntryStub, JSEntryStub – jumps between languages

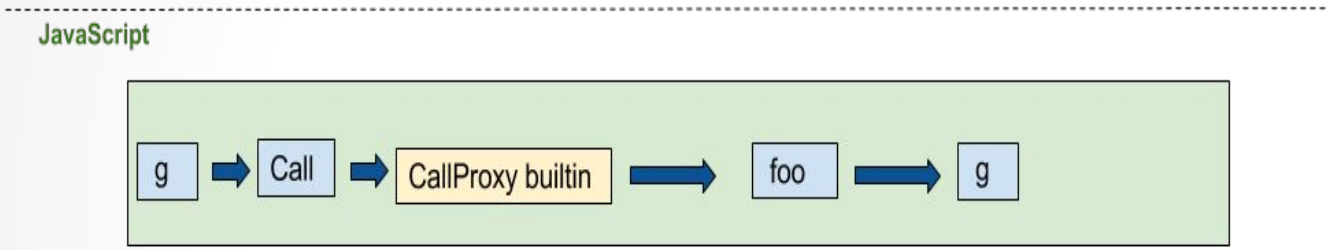


Performance optimizations

Goal & procedure

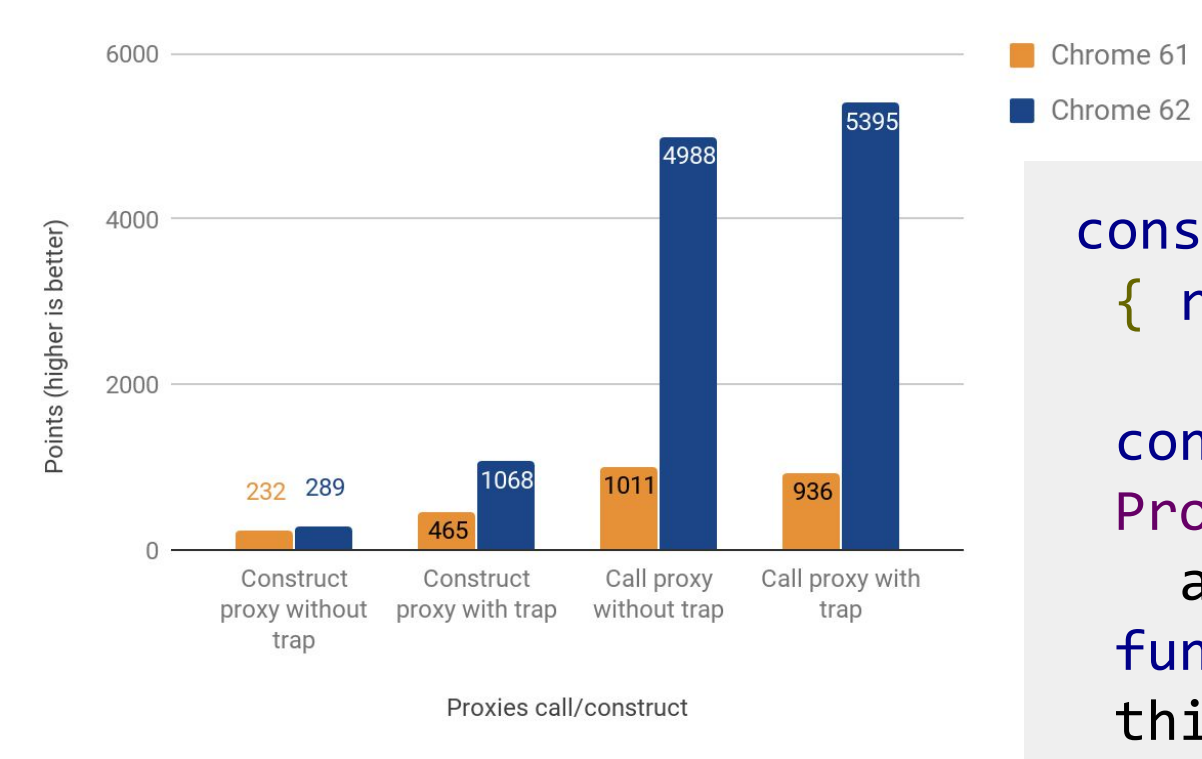
- Goal - To decrease the number of jumps between languages
- How? - By porting code from C++ to CSA
- Procedure:
  1. Write performance tests
  2. Write (add more) correctness tests
  3. Port code from C++ to CSA
  4. Measure performance again
  5. *(optional)* Re-think the feedback and iterate back to point 3

Proxy's [[Call]] internal method (after)



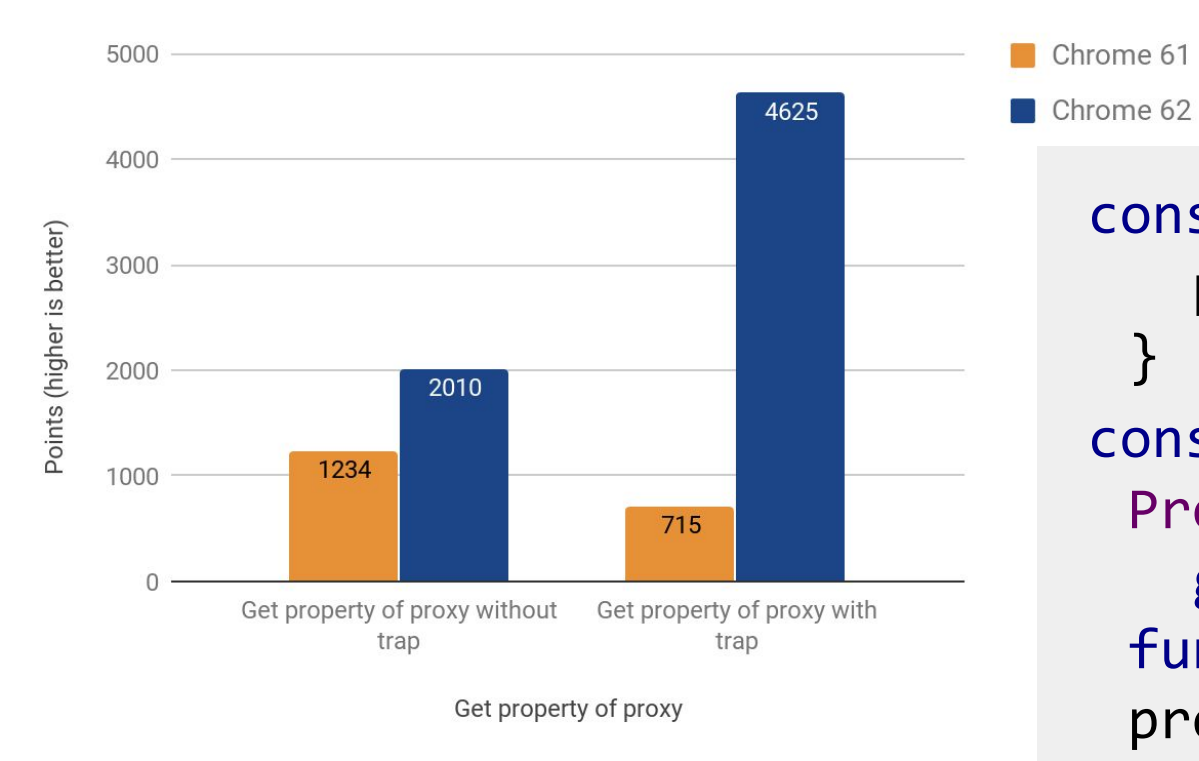
⇒ A lot simpler — 0 language barrier crossings instead of 4 or 8 (in the case of handler being is a proxy itself - example with membrane)!

Results (Proxies construct & call)



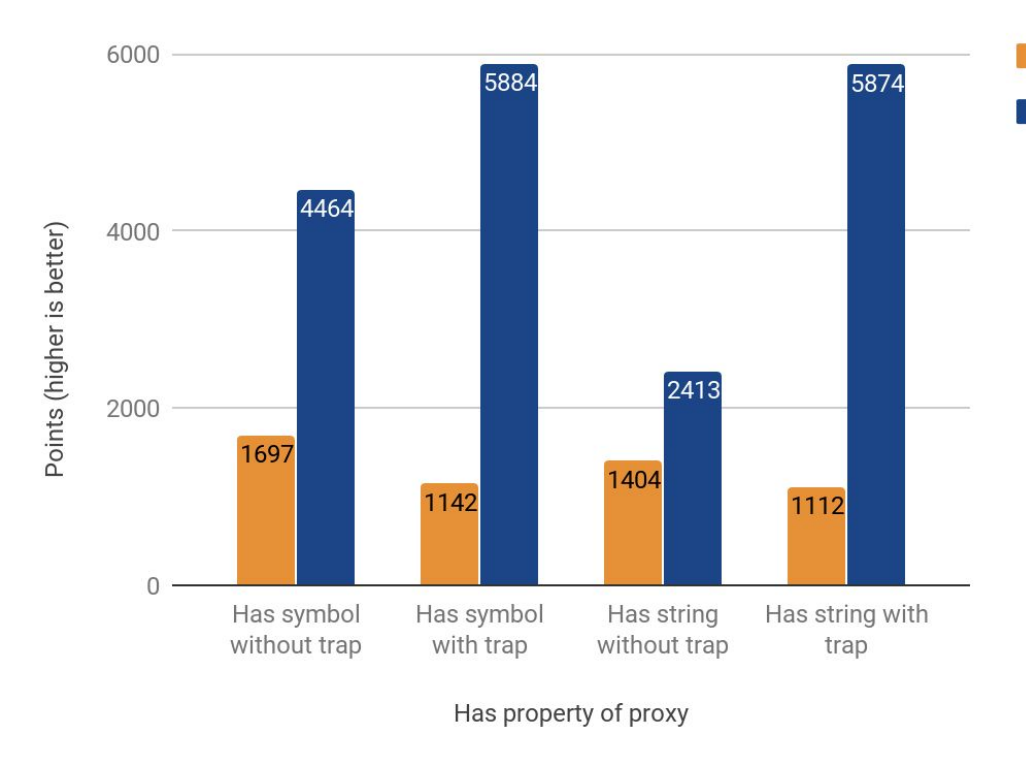

```
const target = () =>
  { return 42; };

  const p = new
  Proxy(target, {
    apply:
  function(target,
  thisArg,
  argumentsList) {
      return 1337;
    }
  });
p();
```

Results (Get property of proxy)



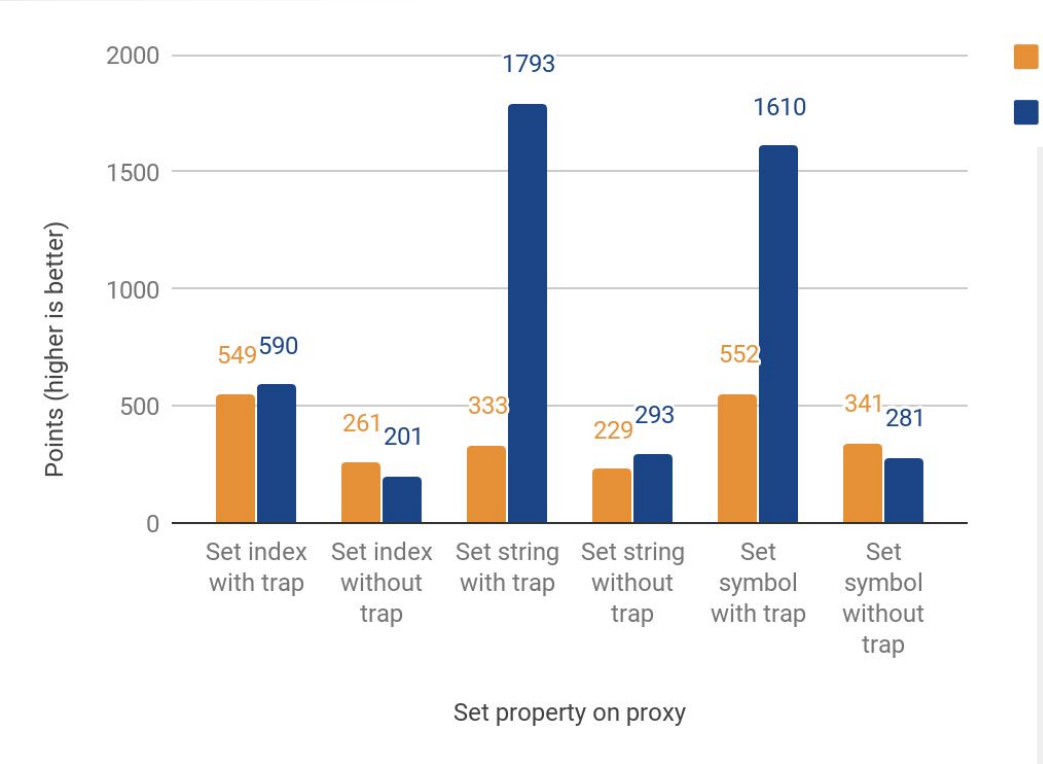

```
const obj = {
    prop: 42
  }
const p = new
  Proxy(obj, {
    get:
  function(target,
  propertyKey, receiver)
  {
      return 1337;
    }
  });
p.prop;
```

Results (Has property of proxy)




```
const obj = {};
const p = new
 Proxy(obj, {
  has:
 function(target,
 propertyKey) {
    return true;
  }
});
'prop' in p;
```

Results (Set property on proxy)




```
const obj = { prop:
 undefined; };

const p = new Proxy(obj,
 {

  set: function(target,
 propKey, value,
 receiver) {

    target[propKey] = 42;

  }

});

p.prop = 1337;
```

Results (Creating new proxies)


```
// With arrow function
  const proxy =
    new Proxy(() => {},
  {});
// With class
  class Class {};
  const proxy1 =
    new Proxy(Class,
  {});
// With object
  const proxy2 =
    new Proxy({}, {});
```

Conclusion & future plans

24%–546%

Improvement when creating new proxies and calling *construct*, *call*, *get*, *has* and *set* traps

js.talks();

Nov 18, 2017

Future improvements

- Porting to CSA *getOwnProperty* trap
- Porting all other possible traps
- Porting NewObject (for objects created through proxies) to CSA – this will improve the performance of "Construct proxy without trap" use case
- Further optimizations in TurboFan (V8's new optimizing compiler – on by default since Chrome M59)

Contributing

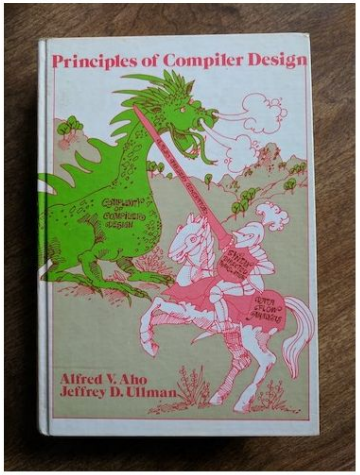
Medium

Franziska Hinkelmann Follow
Compiler Engineer at Google. Node.js Monkey Patcher.
Jul 30 · 4 min read

How do I get started with V8 development?

Are you interested in understanding more about compilers, virtual machines, JavaScript engines, and maybe even want to contribute to V8? Have you never taken a compiler course (maybe no formal CS course) or no experience in C++? Fear not, nobody was born with that knowledge. You don't need to understand all aspects of compilers to make a contribution. Here are some resources that might help you on the way.

There are very few compiler books, and I have not found one that covers modern optimizations, especially for JavaScript engines. If you want to learn the fundamentals, the Dragon Book (Compilers: Principles, Techniques, & Tools, 2nd Edition) is the book to read.



My professor's copy that he used in grad school. Get the 2nd edition from 2006.

- Contributing to an open-source project is great!

js.talks();

Nov 18, 2017

References (1)

Introductory

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy — reference
- https://developers.google.com/web/updates/2016/02/es2015-proxies — intro blog post
- http://soft.vub.ac.be/~tvcutsem/invokedynamic/proxies_tutorial — 2-year old tutorial on proxies + great number of articles and talks on their use, e.g. membranes

Observation

- https://www.youtube.com/watch?v=Oev8wv6S1wI — "Object.observe with ES6 Proxy"
- http://blog.revathskumar.com/2016/02/es6-observe-change-in-object-using-proxy.html — "ES6: observe the object change using Proxy"
- https://github.com/nx-js/observer-util — "An NX utility, responsible for powerful data observation with ES6 Proxies"

References (2)

Meta programming

- http://exploringjs.com/es6/ch_proxies.html — "Metaprogramming with proxies"
- https://www.keithcirkel.co.uk/metaprogramming-in-es6-part-3-proxies/ — "Metaprogramming in ES6: Part 3 - Proxies"
- http://thecodebarbarian.com/thoughts-on-es6-proxies-performance — "Thoughts on ES6 Proxies Performance"

Contracts

- http://drops.dagstuhl.de/opus/volltexte/2015/5229/pdf/19.pdf — paper "Transparent Object Proxies for JavaScript"

References (3)

Membranes

- http://soft.vub.ac.be/Publications/2012/vub-soft-tr-12-03.pdf — membranes theory
- https://web.archive.org/web/20150905193543/http://wiki.ecmascript.org/doku.php?id=harmony:proxies#a_simple_membrane — membranes code adapted from here
- https://github.com/v8/v8/blob/master/test/mjsunit/es6/proxies-example-membrane.js — membranes example as one of V8's tests
- https://alexvincent.us/blog/?p=908 — modern membrane implementation (with link to library)

Books

- https://ponyfoo.com/books/practical-modern-javascript/chapters — Chapter 6 "Managing Property Access with Proxies"

ECMAScript proposals

- Optional chaining

```
obj?.prop        // optional static property access
obj?.[expr]      // optional dynamic property access
func?.(...args) // optional function or method call
```

- 'Do' expressions

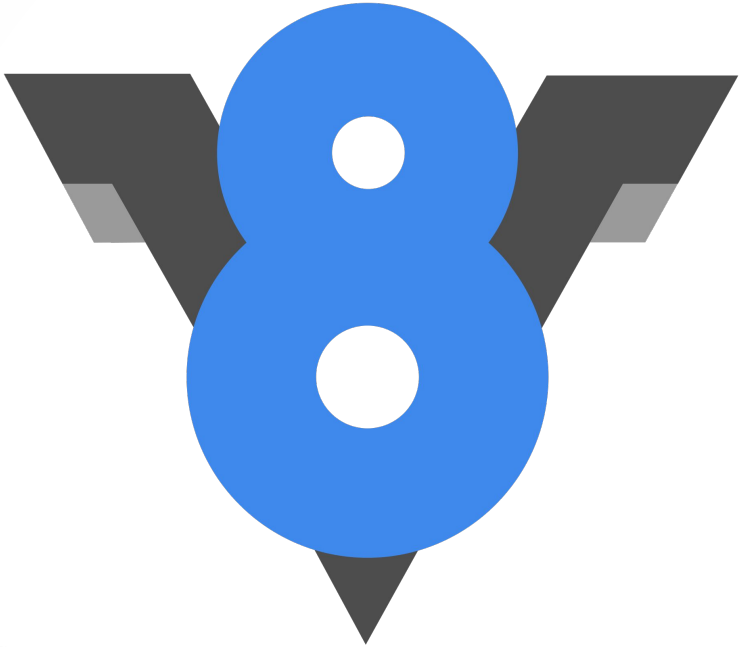
```
let x = do {
  let tmp = f();
  tmp * tmp + 1
};
```

js.talks();

ECMAScript proposals – what you can do today?

- Contributing – https://github.com/tc39/ecma262/blob/master/CONTRIBUTING.md (published yesterday)
- File issues
- Make PRs
- Talk, blog, tweet about proposals
- Write *test262* tests
- … and more!

Thanks to



and this awesome audience!

MUC JS
finest JavaScript since 2010

js.talks();

Nov 18, 2017

Thanks to our Sponsors:

General Sponsor: epam

Hosting Partner: SUPERHOSTING.BG

Gold Sponsors:

SBTech we know sports

fourth

Paysafe

Dutch Tech

Westernacher Solutions

Chaos Group

Anakatech

axway imagination takes shape

softserve experience matters

catenate

10 Years vmware Bulgaria

Silver Sponsors:

SafeCharge

Progress

Droxic

Nemetschek Bulgaria

Technological Partner: Microsoft

In-Kind Partner: Office relax 0888 20 70 81 ПОКАНЕТЕ НИ на безплатна демонстрация в Вашия офис

js.talks();

Nov 18, 2017