Fast JavaScript in V8

Erik Corry
Google

The V8 JavaScript Engine

- A from-scratch reimplementation of the ECMAScript 3 language
- An Open Source project from Google, primarily developed here in Århus
- A real JavaScript "VM" with a JIT compiler, accurate garbage collection etc.
- Embedded in Google Chrome, Android 2.2, node.js, HP WebOS
- See more at `http://code.google.com/p/v8/`
- Kickstarted the JavaScript performance wars, resulting in better JS performance on all browsers
- My passion: "A rising tide lifts all boats"

Handout note: If you found the Rx, Erjang or akka talks interesting then check out node.js.

...well almost all boats.

Image credit: Jim Champion http://www.flickr.com/photos/treehouse1977/967186270/ Attribution-ShareAlike 2.0 Generic

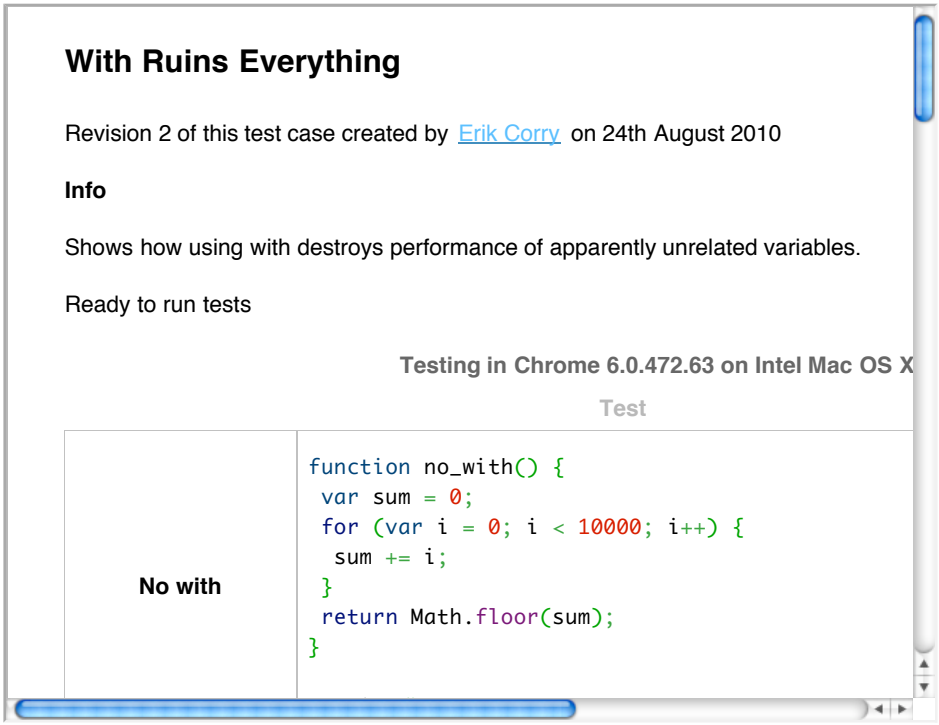
Never use `with`

```
function with_with() {
 with(Math) {
  var sum = 0;
  for (var i = 0; i < 10000; i++) {
   sum += i;
  }
  return floor(sum); // Note this is outside the loop!
 }
}

with_with();
```

Never use `with` part 2

`http://jsperf.com/with-ruins-everything/2`



With Ruins Everything

Revision 2 of this test case created by Erik Corry on 24th August 2010

Info

Shows how using with destroys performance of apparently unrelated variables.

Ready to run tests

Testing in Chrome 6.0.472.63 on Intel Mac OS X

Test

No with

```
function no_with() {
 var sum = 0;
 for (var i = 0; i < 10000; i++) {
  sum += i;
 }
 return Math.floor(sum);
}
```

`eval` can be like `with`

```
(function() {
 var sum;

 function bench() {
  sum = 0;
  for (var i = -1000; i < 1000; i++) {
   sum += i;
  }
  sum += eval("42");
  return sum;
 }

 bench();
})();
```

`eval` can be like `with` part 2

```
(function() {
 var sum;

 function bench() {
  sum = 0;
  for (var i = -1000; i < 1000; i++) {
   sum += i;  // which sum?
  }
  sum += eval("var sum;");
  return sum;
 }

 bench();
})();
```

`eval` can be like `with` part 3

- Solution: use eval.call instead.
- `eval.call(null, "42");`
- See `http://jsperf.com/eval-done-right`

It should be slow to use `parseInt`

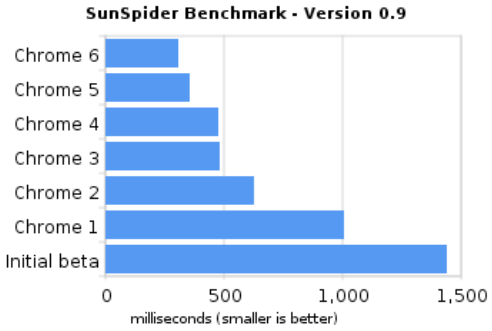
- Instead of `Math.floor` people use `parseInt`
- This converts your number to a string
- Then it parses it as an integer
- When it gets to a decimal point it stops parsing
- That's slow, but...

But `parseInt` has friends

- ... it's fast.
- Dean Edwards' JavaScript packer uses parseInt
- SunSpider uses packer
- So everyone is fast at `parseInt` on floating point numbers

We are not SunSpider fans

... but we are fast at it.



I actually do love jsnes

What `parseInt` used to look like

```
// ECMA-262 - 15.1.2.2
function GlobalParseInt(string, radix) {
 if (IS_UNDEFINED(radix)) {
  radix = 0;
 } else {
  radix = TO_INT32(radix);
```

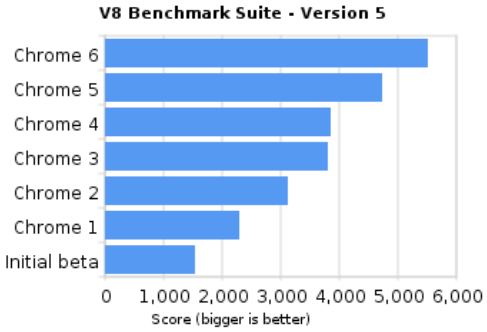
```
  if (!(radix == 0 || (2 <= radix && radix <= 36))) return $NaN;
 }
 string = TO_STRING(string);
 return %StringParseInt(string, radix);
}
```

What `parseInt` looks like now

```
// ECMA-262 - 15.1.2.2
function GlobalParseInt(string, radix) {
 if (IS_UNDEFINED(radix)) {
  if (%_IsSmi(string)) return string;
  if (IS_NUMBER(string) && ((0.01 < string && string < 1e9) || (-1e9 < string && string < -0.01))) {
   // Truncate number.
   return string | 0;
  }
  radix = 0;
 } else {
  radix = TO_INT32(radix);
  if (!(radix == 0 || (2 <= radix && radix <= 36))) return $NaN;
 }
 string = TO_STRING_INLINE(string);
 if (%_HasCachedArrayIndex(string) && (radix == 0 || radix == 10)) {
  return %_GetCachedArrayIndex(string);
 }
 return %StringParseInt(string, radix);
}
```

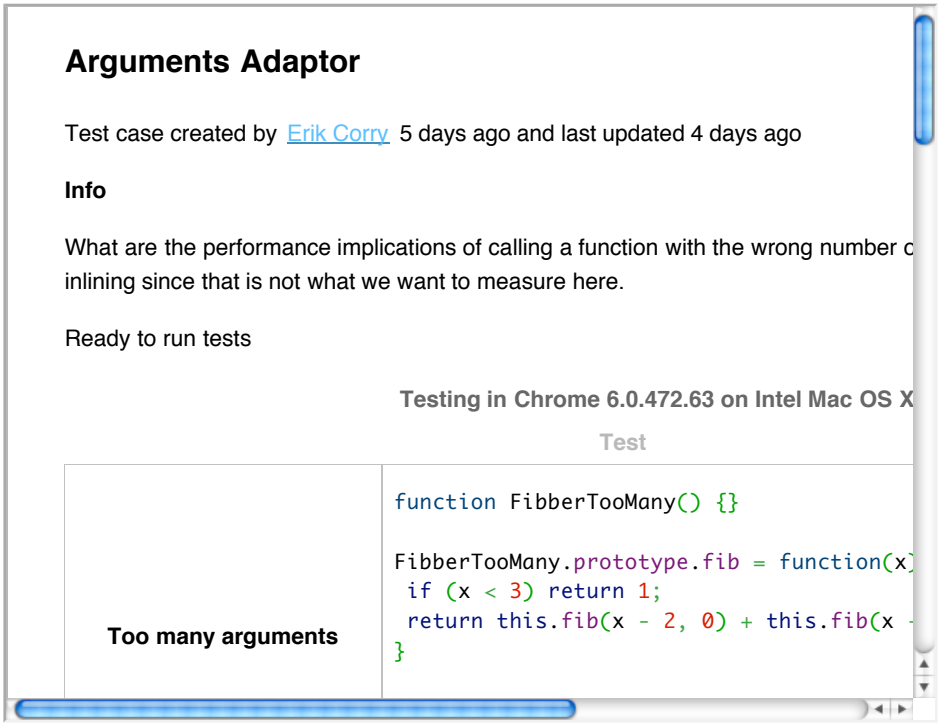
Keeping method calls fast

- Calling methods is a fundamental operation in object-oriented programs
- In JavaScript, methods are usually properties on the prototype of an object
- It's not a huge effect, but V8 and Safari like for the number of arguments to match up at the call site and the function definition.
- There are a lot of method calls in the V8 benchmark suite



Goldilocks method calls

`http://jsperf.com/arguments-adaptor`



Arguments Adaptor

Test case created by Erik Corry 5 days ago and last updated 4 days ago

Info

What are the performance implications of calling a function with the wrong number c
inlining since that is not what we want to measure here.

Ready to run tests

Testing in Chrome 6.0.472.63 on Intel Mac OS X

Test

Too many arguments	`function FibberTooMany() {}`
	
	`FibberTooMany.prototype.fib = function(x`
	`  if (x < 3) return 1;`
	`  return this.fib(x - 2, 0) + this.fib(x -`
	`}`

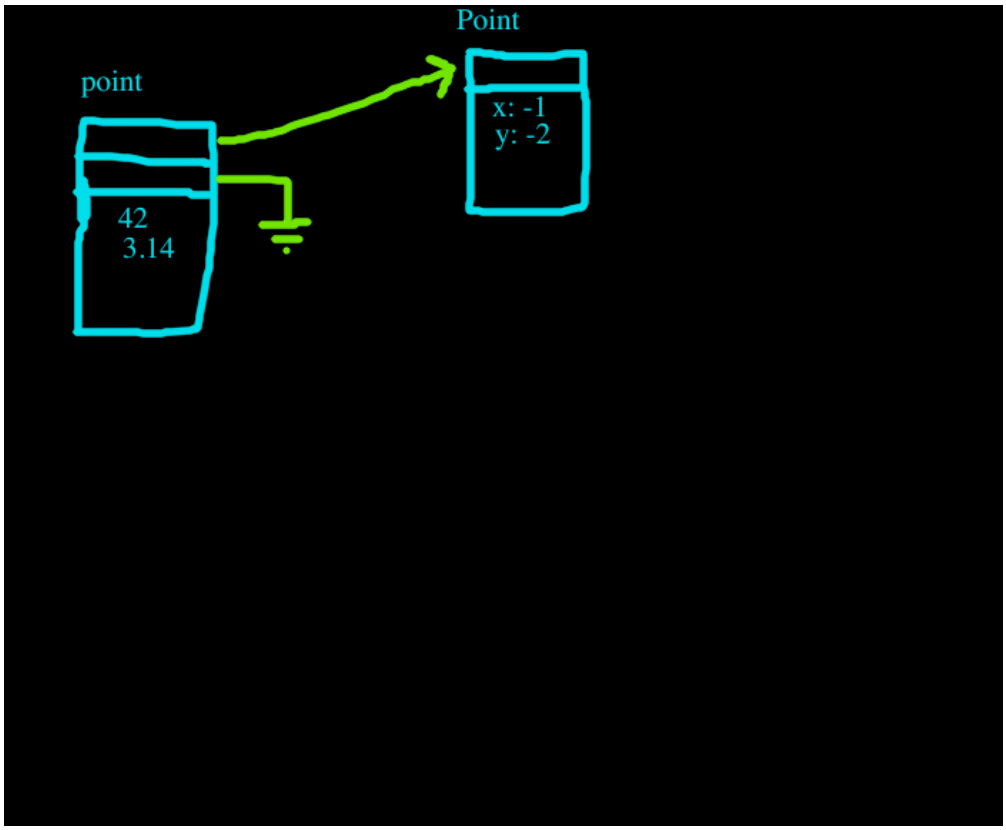
Keeping property accesses fast

- Accessing member variables on objects is another fundamental operation in object-oriented programs
- This applies to member variables on `this` too
- In JavaScript member variables are properties on an object
- Objects are rather like string-keyed hash maps
- So how does V8 represent these objects?

Maps in V8

- Each object in V8 has a map that describes its layout
- Many objects share a map

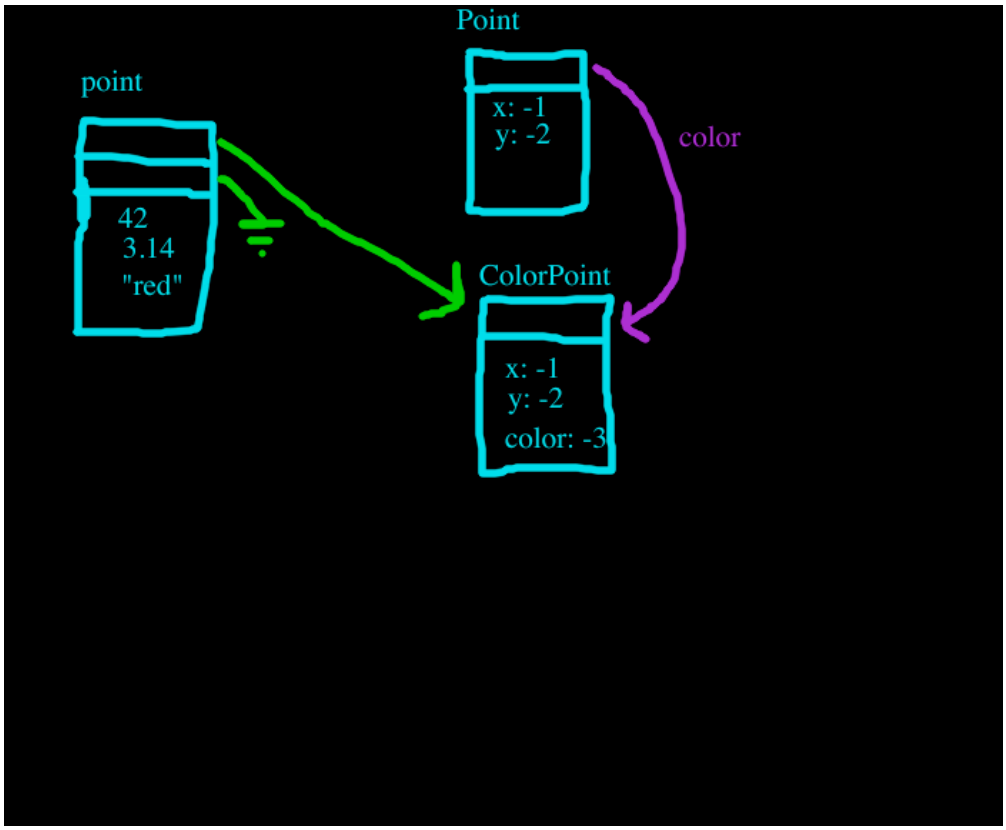
```
function Point(x, y) {
 this.x = x;
 this.y = y;
}
var point = new Point(42, 3.14);
```



Map Transitions in V8

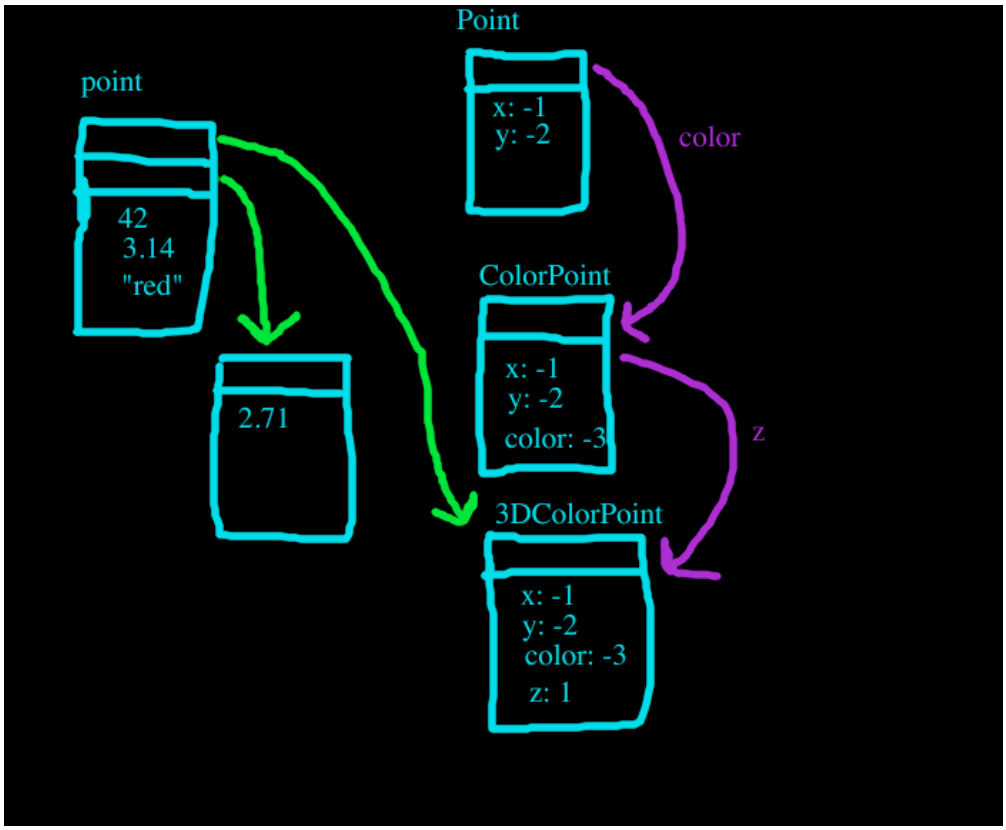
- If you add a property to an object it transitions to a new map

```
point.color = "red";
```



Out-of-object properties

```
point.z = 2.71;
```



Load of an in-object property

```
return this.x;

17 8b4508         mov eax,[ebp+0x8] ;load this from stack
20 a801           test al,0x1       ;is this an object
22 0f841b000000   jz 55  (0xf54905f7)
28 8178ff21a049f5 cmp [eax+0xff],0xf549a021 ;check map
35 0f850e000000   jnz 55  (0xf54905f7)
41 8b98feffff7f   mov ebx,[eax+0x7ffffffe] ;load in-object
47 89d8           mov eax,ebx               ;return in eax
49 8be5           mov esp,ebp               ;js return
51 5d             pop ebp
52 c20400         ret 0x4
; out-of-line code
55 b97d514af5     mov ecx,0xf54a517d        ;"x"
60 e89ff8feff     call LoadIC_Initialize    ;load
65 a9dbffffff     test eax,0xffffffdb       ;offset
70 89c3           mov ebx,eax               ;restore regs
72 8b7df8         mov edi,[ebp+0xf8]
75 8b4508         mov eax,[ebp+0x8]
78 ebdf           jmp 47  (0xf54905ef)      ;to fast case
```

Making properties slow: Out of object

```
function OutOfObject() {
 this.initialize();
}

OutOfObject.prototype.initialize = function() {
 this.foo = null;
 this.bar = null;
 this.color = "transparent";
 this.that = "bla";
 this.the_other = "bla";
 this.x = 0;
 this.y = 0;
}
```

Making properties slow: `delete`

```
function Deleted() {
 this.foo = null;
 this.bar = null;
```

```
  this.color = "transparent";
  this.that = "bla";
  this.the_other = "bla";
  this.x = 0;
  this.y = 0;
  delete this.foo;
 }
```

Making properties slow: ECMAScript 5

- This can probably be improved
- But right now using these ECMAScript 5 functions will slow down property access:
  - `Object.freeze();`
  - `Object.seal();`
  - `Object.preventExtensions()`

Compare ways to slow down JS

`http://jsperf.com/making-property-access-slow`

Making property access slow

Test case created by Erik Corry 1 week ago

Info

Investigates some ways to accidentally slow down property access.

Preparation code

```
<script>
 function InObject() {
  this.foo = null;
  this.bar = null;
  this.color = "transparent";
  this.that = "bla";
  this.the_other = "bla";
  this.x = 0;
  this.y = 0;
 }

 function OutOfObject() {
```

The silly one: `indexOf`

- Some like to use indexOf to test whether a string starts with something.

```
if (hayStack.indexOf("fish") == 0) {
```

- You can use lastIndexOf similarly
- Also applies to arrays
- What if the string you are looking at is big?
- For `indexOf` use `/^fish/`
- For `lastIndexOf` I'd like to suggest `/fish$/`
- Unfortunately that's not optimized...

Instead of `lastIndexOf`

```
String.prototype.EndsWith = function(needle) {
 var len = needle.length;
 if (len > this.length) return false;
 var offset = this.length - len;
 for (var i = 0; i < len; i++) {
  if (needle.charCodeAt(i) !==
      this.charCodeAt(offset + i)) return false;
 }
 return true;
}
```

`http://jsperf.com/careful-with-that-indexof-eugene`

How to iterate over an array

- The best way is `for (var i = 0; i < data.length; i++) {`
- You can cache the length in a variable, but even on the empty loop it's only worth <20%
- This is 20 times slower: `for (var i in data) {`
- It also breaks down if someone adds an enumerable property to `Array.prototype`
- If you have a sparse array then you have to use `for in`. Perhaps one day `forEach` will optimize for this
- Even with the function call overhead `array.forEach()` is still 4 times faster than `for in`
- `http://jsperf.com/js-forin-vs-classicfor/2`

The Dreaded Miscellaneous

- DOM operations are slow. Cache the results.
- Local variables in functions are faster and cleaner than variables on the window/global object
- Regular expressions can do catastrophic backtracking
- Premature optimization is the root of all evil.
- There's a profiler built into Google Chrome.

Summary

- O(n) → O(1) — Avoid using `indexOf` to test what a string starts with
- Factor 200 — Avoid `with`
- Factor 20 — Use `for` loop instead of `for in` on arrays.
- Factor 6 — Use `eval.call(null, arg)`
- Factor 3 — Use `x.foo = void 0` instead of `delete`
- Factor 3 (right now) — Avoid `Object.freeze` etc.
- Factor 2 — Keep properties in-object
- 20% — Call functions with the right number of arguments
- 20% — Cache array.length in an empty loop