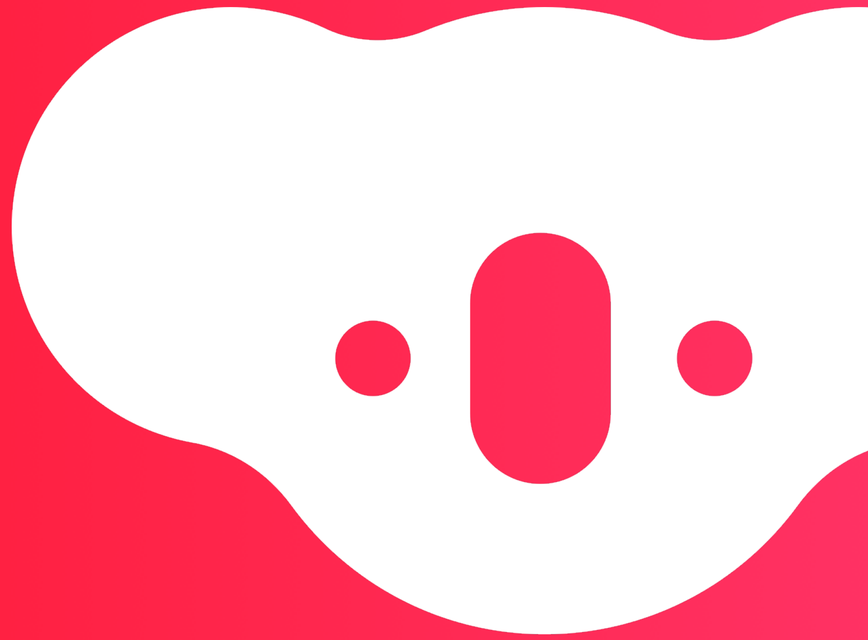
NETEASE
Kaola Center Training



浅析V8聊聊JS性能

主讲人：黄加樑

时　间：2019年8月1日

网易考拉

V8是如何被玩坏的

目录

网易考拉

V8 背景介绍

网易考拉

浏览器内核（渲染引擎）：Blink、WebKit、Gecko

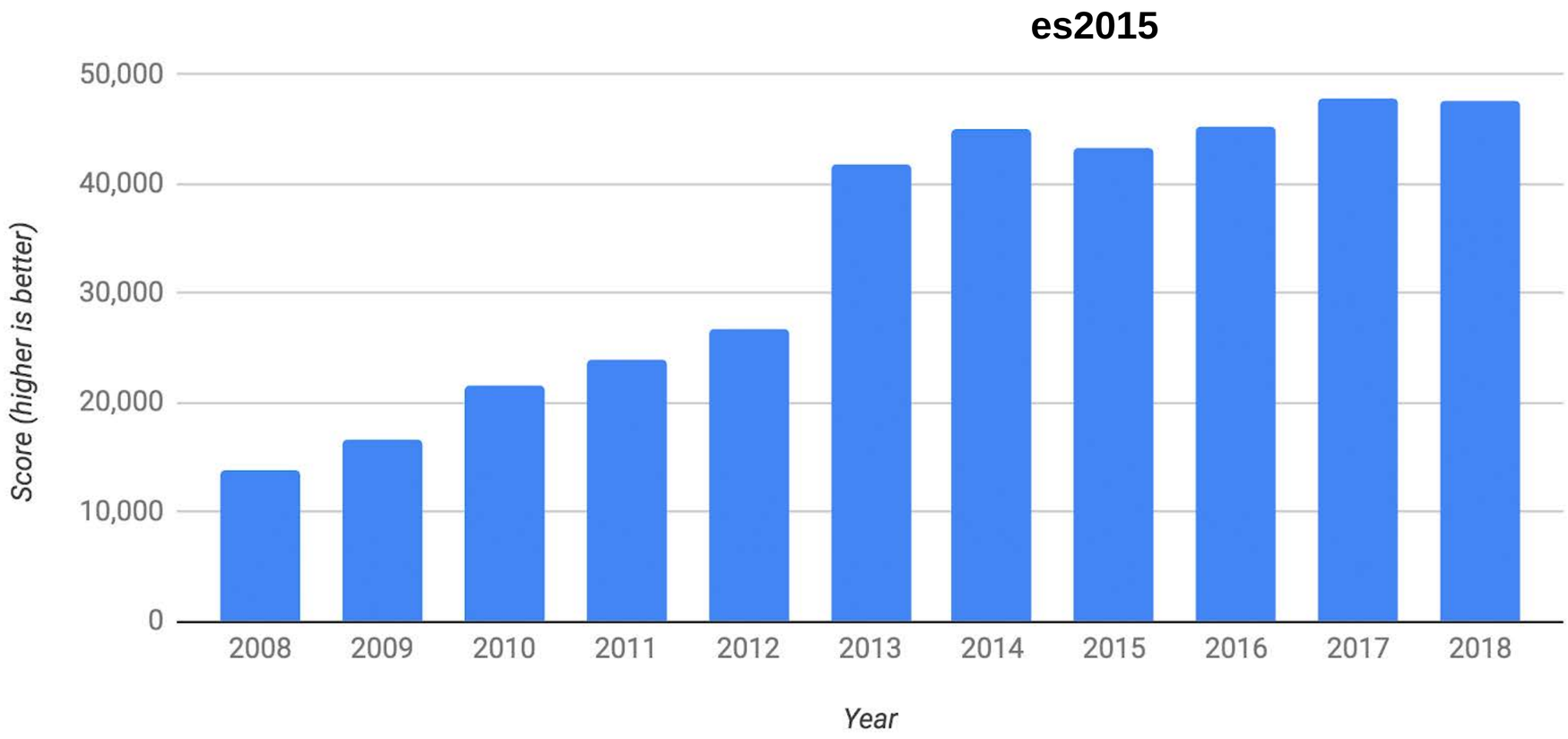
JS 引擎：V8、JavaScriptCore、SpiderMonkey

V8 是一个由 Google 开源的高性能 JavaScript 和 WebAssembly 引擎，其源代码使用 C++ 编写。V8 被用于 Google 的开源浏览器 Chrome 中，同时也被用于 Node.js。

V8 JavaScript 引擎（解释、编译、执行）

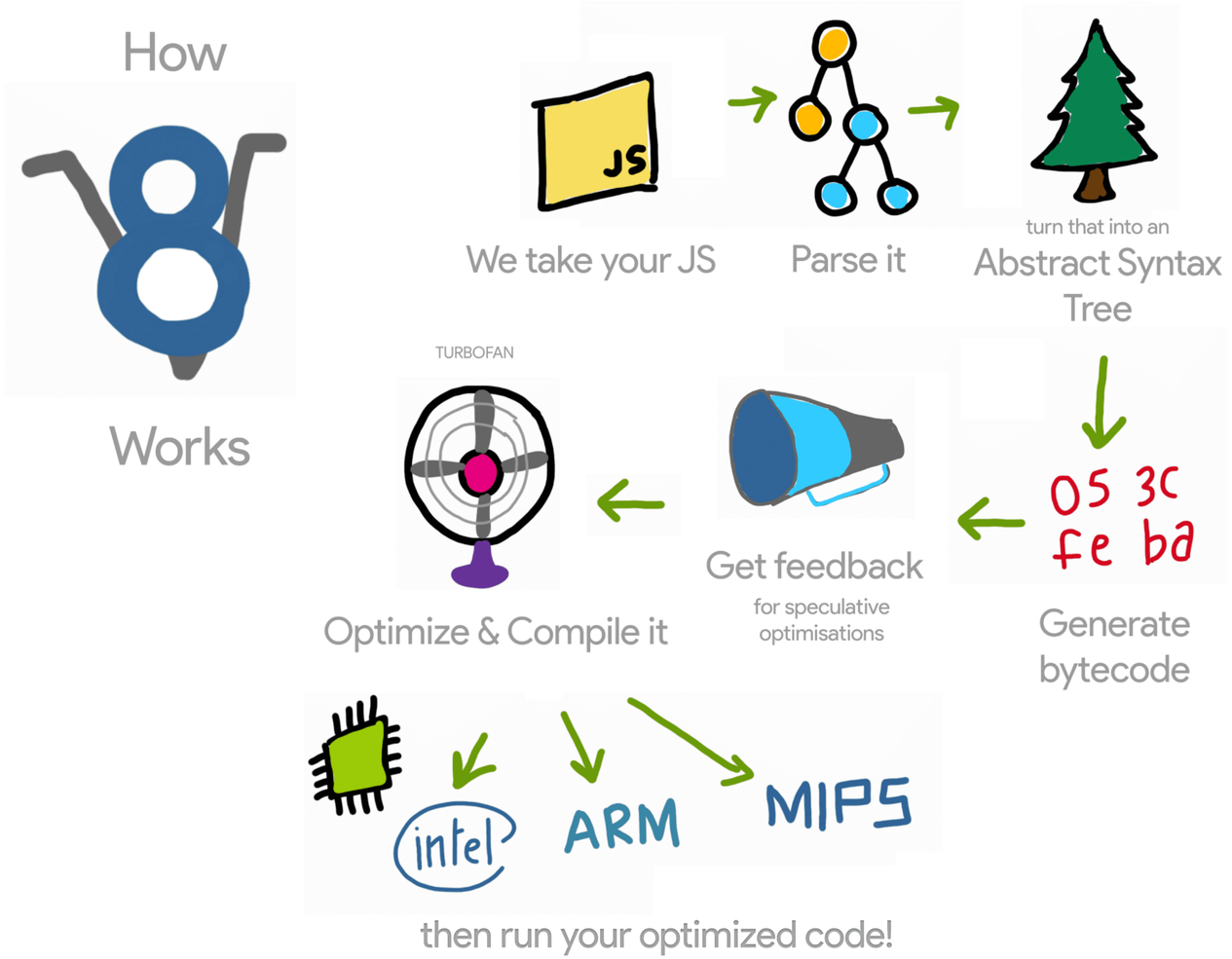
V8 背景介绍 – 概述

网易考拉



V8 背景介绍 – 执行过程

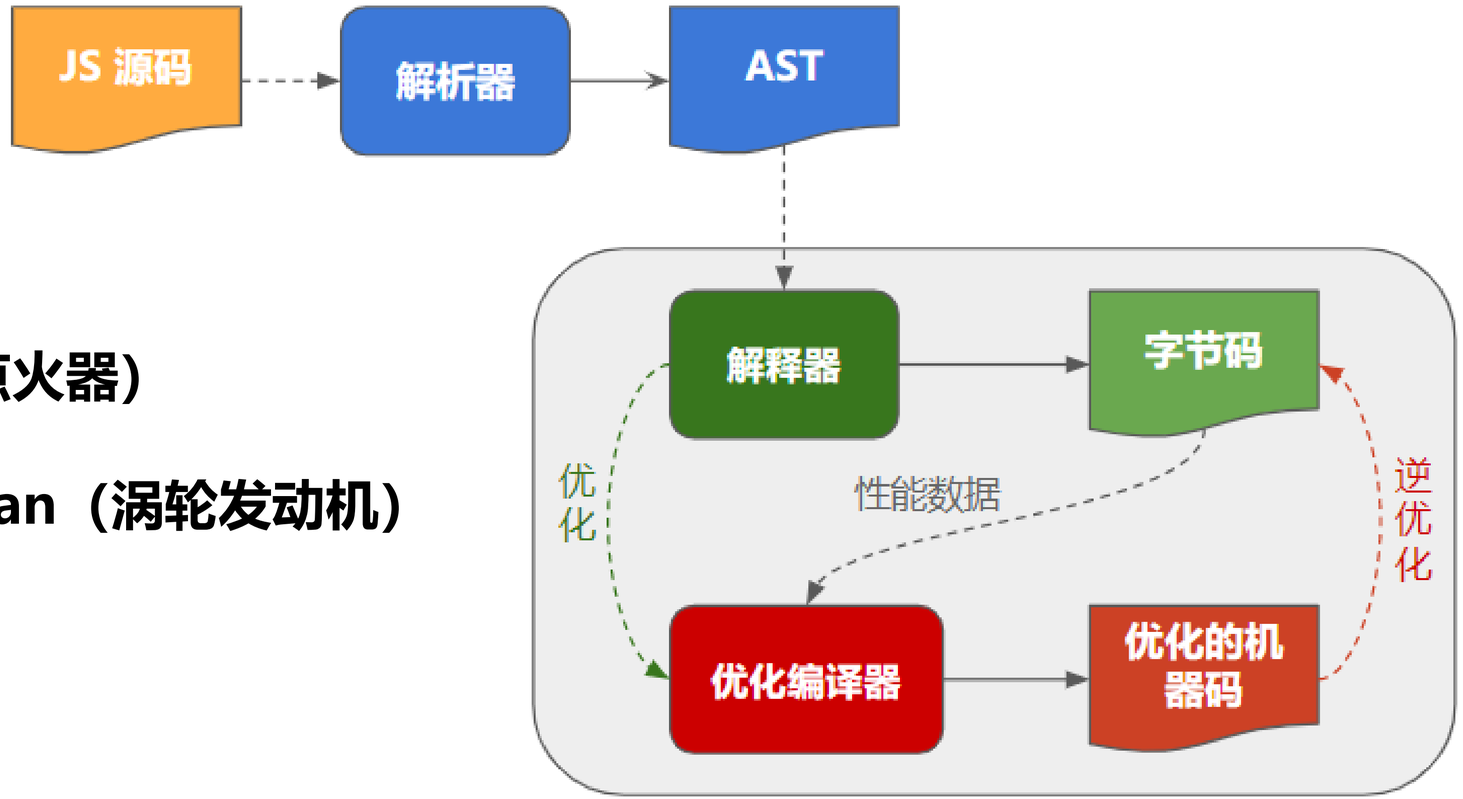
网易考拉



By @addyosmani

V8 背景介绍 – Pipeline

网易考拉



V8 解释器：Ignition（点火器）

V8 优化编译器：TurboFan（涡轮发动机）

所有场景都能优化吗，限制？

环境

网易考拉

Node 10.16.0
V8 6.8.275.32-node.52

Turbofan 优化限制 – 从现象开始

网易考拉

1. Node --opt / --no-opt　　开启 / 关闭 Turbofan 优化（默认开启）
2. Node --trace-opt　　跟踪Turbofan的优化过程
3. 函数生成

```
function generate(n) {
    let s = '(function func' + n + '(x) { return 0';
    for (let i = 0; i < n; ++i) {
        s += '+x';
    }
    s += '; })';
    return eval(s);
}

//generate(10)： function func10(x) { return 0+x+x+x+x+x+x+x+x+x+x; }
```

Turbofan 优化限制 – 从现象开始

网易考拉

case-func10.js

```
const func10 = generate(10);
console.time('measure');
let result = 0;
for (let i = 0; i < 5e6; ++i) {
    result += func10(i);
}
console.timeEnd('measure');
```

case-func10000.js

```
const func10000 = generate(10000);
console.time('measure');
let result = 0;
for (let i = 0; i < 50000; ++i) {
    result += func10000(i);
}
console.timeEnd('measure');
```

Turbofan 优化限制 – 从现象开始

网易考拉

$ node ./src/case-func10.js

measure: 212.007ms

$ node --no-opt ./src/case-func10.js

measure: 671.125ms

$ node ./src/case-func10000.js

measure: 2715.201ms

$ node --no-opt ./src/case-func10000.js

measure: 2675.396ms

Turbofan 优化限制 – 从现象开始

网易考拉

$ node --trace-opt ./src/case-func10.js

[marking 0x0010e43ed249 <JSFunction func10 (sfi = 000003E6086D7061)> for optimized recompilation, reason: small function, ICs with typeinfo: 10/10 (100%), generic ICs: 0/10 (0%)]

[compiling method 0x0010e43ed249 <JSFunction func10 (sfi = 000003E6086D7061)> using TurboFan]

[optimizing 0x0010e43ed249 <JSFunction func10 (sfi = 000003E6086D7061)> - took 4.108, 0.787, 0.078 ms]

[completed optimizing 0x0010e43ed249 <JSFunction func10 (sfi = 000003E6086D7061)>]

$ node --trace-opt ./src/case-func10000.js

Nothing

Turbofan 优化限制 – 原因

网易考拉

```
OptimizationReason RuntimeProfiler::ShouldOptimize(JSFunction* function,
  // 生成的bytecode大于60kb，跳出
 if (shared->GetBytecodeArray()->length() > kMaxBytecodeSizeForOpt //60kb) {
      return OptimizationReason::kDoNotOptimize;
  }
}
```

deps/v8/src/runtime-profiler.cc#L200

V8 的限制 — 60KB（注意是字节码）

Turbofan 优化限制 – Bytecode

网易考拉

func10

74 Bytes

Node --print-bytecode

```
[generated bytecode for function: func10]
Parameter count 2
Register count 2
Frame size 16
   16 E> 0000021FC005FC06 @    0 : a5                StackCheck
   38 S> 0000021FC005FC07 @    1 : 0c 03             LdaSmi [3]
         0000021FC005FC09 @    3 : 26 fb             Star r0
   40 S> 0000021FC005FC0B @    5 : 25 02             Ldar a0
   56 E> 0000021FC005FC0D @    7 : 34 fb 00          Add r0, [0]
         0000021FC005FC10 @   10 : 26 fa             Star r1
         0000021FC005FC12 @   12 : 25 02             Ldar a0
   58 E> 0000021FC005FC14 @   14 : 34 fa 01          Add r1, [1]
         0000021FC005FC17 @   17 : 26 fa             Star r1
         0000021FC005FC19 @   19 : 25 02             Ldar a0
……
   74 E> 0000021FC005FC4C @   70 : 34 fa 09          Add r1, [9]
   77 S> 0000021FC005FC4F @   73 : a9                Return
Constant pool (size = 0)
Handler Table (size = 0)
```

Turbofan 优化限制 – Bytecode

网易考拉

func10000

99235 Bytes（超过60k）

Node --print-bytecode

```
[generated bytecode for function: func10000]
Parameter count 2
Frame size 8
   19 E> 000003CADAC5C3E2 @    0 : a0                StackCheck
   25 S> 000003CADAC5C3E3 @    1 : 0b                LdaZero
         000003CADAC5C3E4 @    2 : 26 fb             Star r0
         000003CADAC5C3E6 @    4 : 25 02             Ldar a0
   33 E> 000003CADAC5C3E8 @    6 : 32 fb 00          Add r0, [0]
         000003CADAC5C3EB @    9 : 26 fb             Star r0
         000003CADAC5C3ED @   11 : 25 02             Ldar a0
   35 E> 000003CADAC5C3EF @   13 : 32 fb 01          Add r0, [1]
         000003CADAC5C3F2 @   16 : 26 fb             Star r0
         000003CADAC5C3F4 @   18 : 25 02             Ldar a0

         000003CADAC74772 @ 99216 : 25 02              Ldar a0
20029 E> 000003CADAC74774 @ 99218 : 00 32 fb ff 0e 27 Add.Wide r0, [9998]
         000003CADAC7477A @ 99224 : 26 fb             Star r0
         000003CADAC7477C @ 99226 : 25 02             Ldar a0
20031 E> 000003CADAC7477E @ 99228 : 00 32 fb ff 0f 27 Add.Wide r0, [9999]
20034 S> 000003CADAC74784 @ 99234 : a4                Return
Constant pool (size = 0)
Handler Table (size = 0)
```

Turbofan 优化限制 – Machine Code

网易考拉

func10经过优化编译后的机器码

356 Bytes！！(接近5倍)

Node --print-opt-code

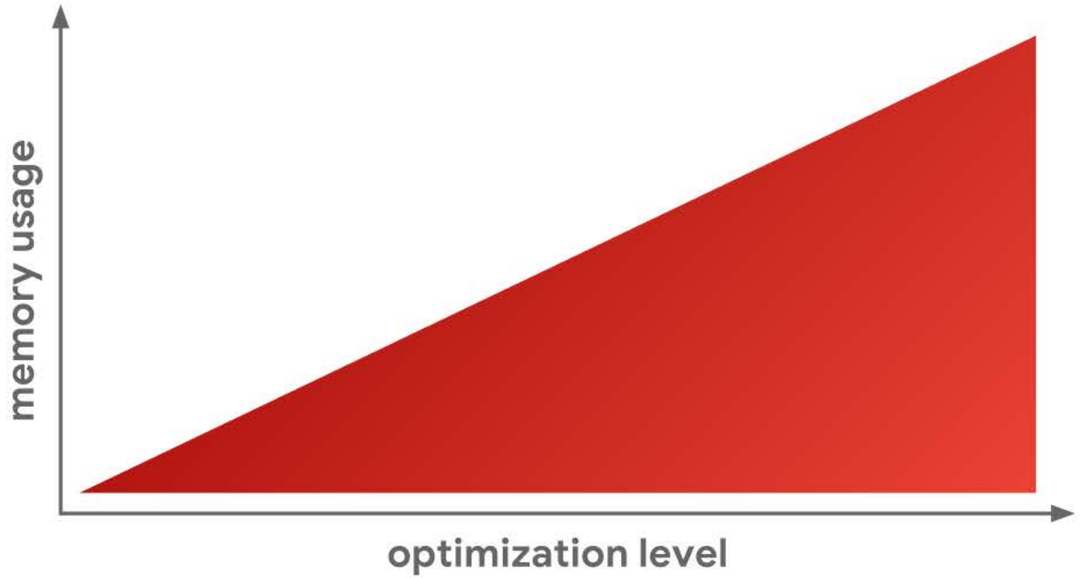
```
Instructions (size = 356)
000002F8C17C4320     0  488b59e0        REX.W movq rbx,[rcx-0x20]
000002F8C17C4324     4  f6430f01        testb [rbx+0xf],0x1
000002F8C17C4328     8  740d            jz 000002F8C17C4337  <+0x17>
000002F8C17C432A     a  49ba90e55ed3f67f0000 REX.W movq
r10,00007FF6D35EE590  (CompileLazyDeoptimizedCode)
000002F8C17C4334    14  41ffe2          jmp r10
000002F8C17C4337    17  55              push rbp
000002F8C17C4338    18  4889e5          REX.W movq rbp,rsp
000002F8C17C433B    1b  56              push rsi
000002F8C17C433C    1c  57              push rdi
000002F8C17C433D    1d  4883ec08        REX.W subq rsp,0x8
000002F8C17C4341    21  488975e8        REX.W movq [rbp-0x18],rsi
……
000002F8C17C447C   15c  e8bfdb0700      call 000002F8C1842040    ;; lazy
deoptimization bailout
000002F8C17C4481   161  0f1f00          nop
```

Turbofan 优化限制 – 猜测

网易考拉

优化产生的机器码，会增大内存的占用，JS引擎需要平衡性能优化和内存使用

猜测：
1. 需要控制内存的使用
2. 生成的机器码在不同指令集上有限制



Inline cache 机制 – 从现象开始

网易考拉

1. Node --use-ic / --no-use-ic　　开启 / 关闭 Inline cache 优化（默认开启）
2. 测试函数

```
function getX(obj) {
    return obj.x;
}
```

Inline cache 机制 – 从现象开始

网易考拉

case-ic-opt.js

```
console.time('measure');
for (var i = 0; i <= 5e6; i++) {
    getX({ x: i });
}
console.timeEnd('measure');
```

case-ic-no-opt.js

```
console.time('measure');
for (var i = 0; i <= 2e6; i++) {
    getX({[`a${i}`]: i});
}
console.timeEnd('measure');
```

Inline cache 机制 – 从现象开始

网易考拉

$ node ./src/case-ic-opt.js

measure: 6.850ms

$ node --no-use-ic ./src/case-ic-opt.js

measure: 367.637ms

$ node src/case-ic-no-opt.js

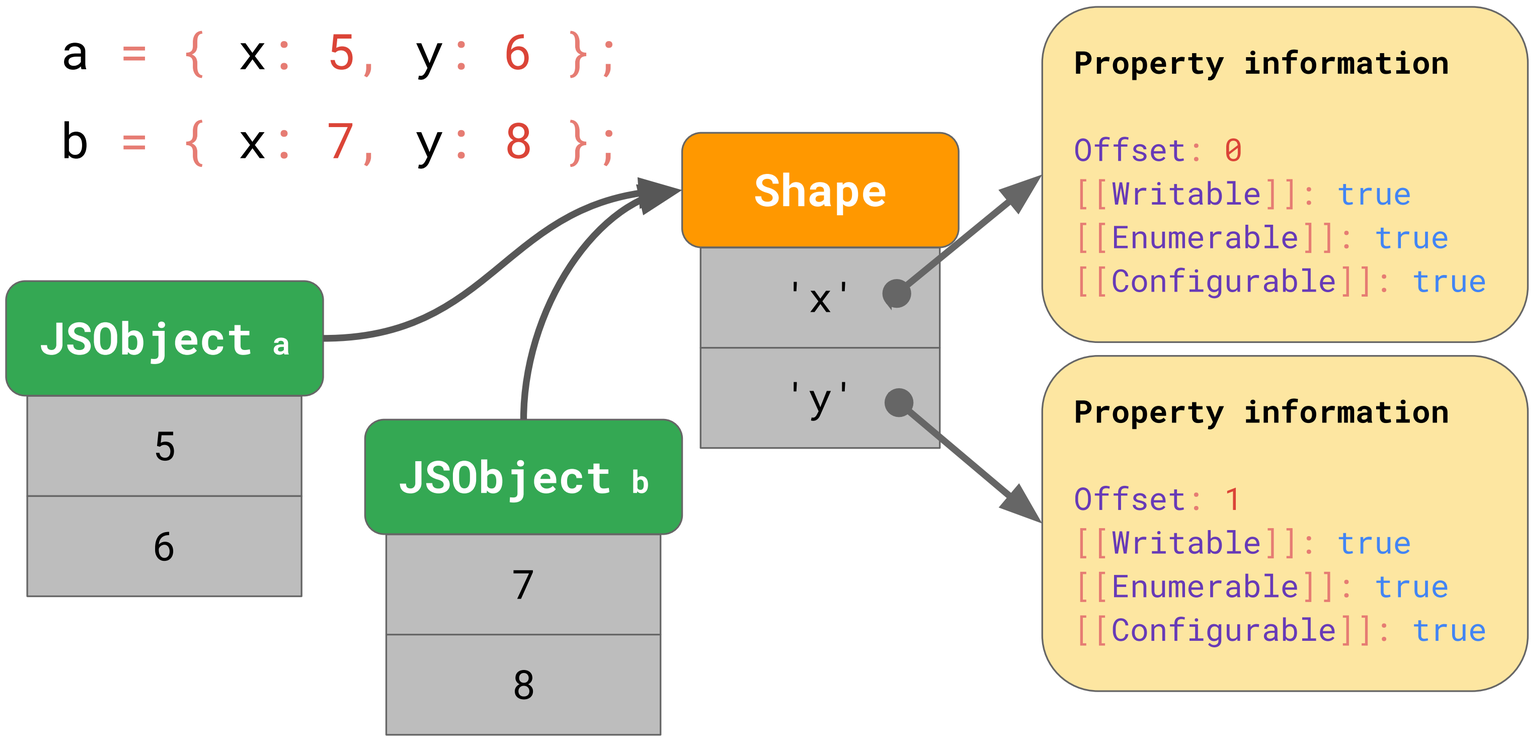
measure: 2411.771ms

$ node -- no-use-ic ./src/case-ic-no-opt.js

measure: 2183.535ms

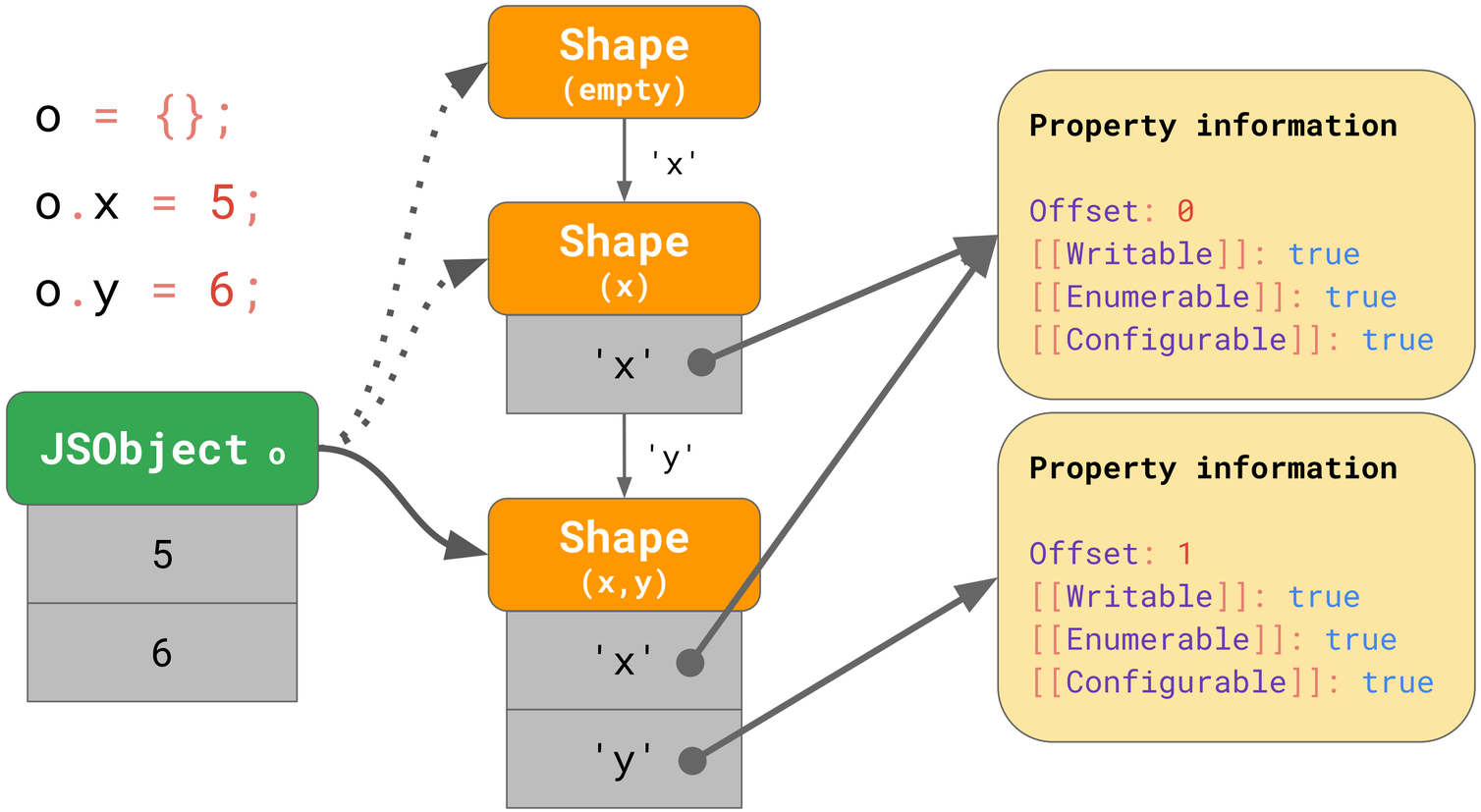
Inline cache 机制 – Shape（形状或隐藏类）

网易考拉



Inline cache 机制 – Shape（形状或隐藏类）

网易考拉



Inline cache 机制 – Shape（形状）

网易考拉

```
var a = { x: 1, y: 2, z: 3 };
var b = { x: 4, y: 5, z: 6 };
// 0x03fb308ff1b9 <Object map = 0000015B2B2F9489>
// 0x03fb308ff2c1 <Object map = 0000015B2B2F9489>
// a, b shape相同

var a = { x: 1, y: 2, z: 3 };
var b = { y: 3, x: 2, z: 4 };
// 0x03a7049bf1b1 <Object map = 0000000D753F9489>
// 0x03a7049bf2b9 <Object map = 0000000D753F9579>
// a, b shape不同
```

Inline cache 机制 – Shape（形状或隐藏类）

网易考拉

住户	房屋
张三	房子1
李四	房子2

单元号	业主
1#101	张三
2#101	李四

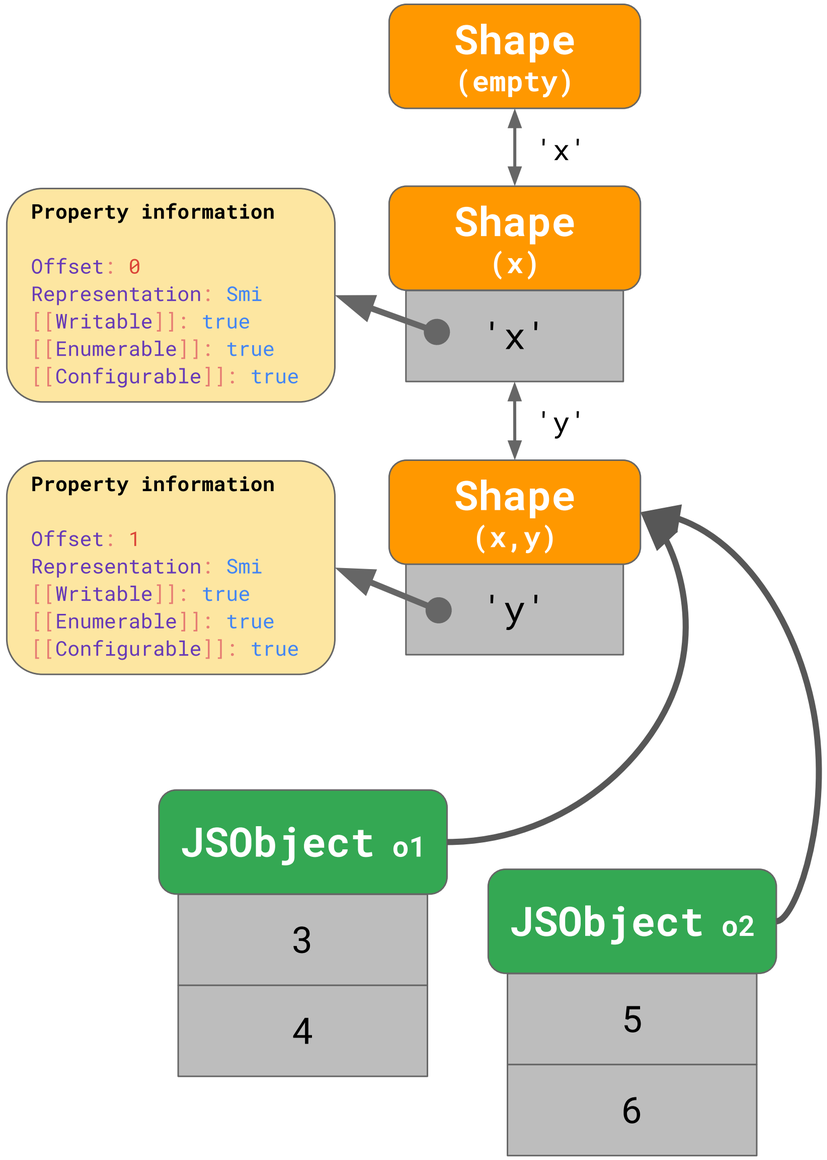
Js Object

单元号	房屋
1#101	房子1
2#101	房子2

Shape

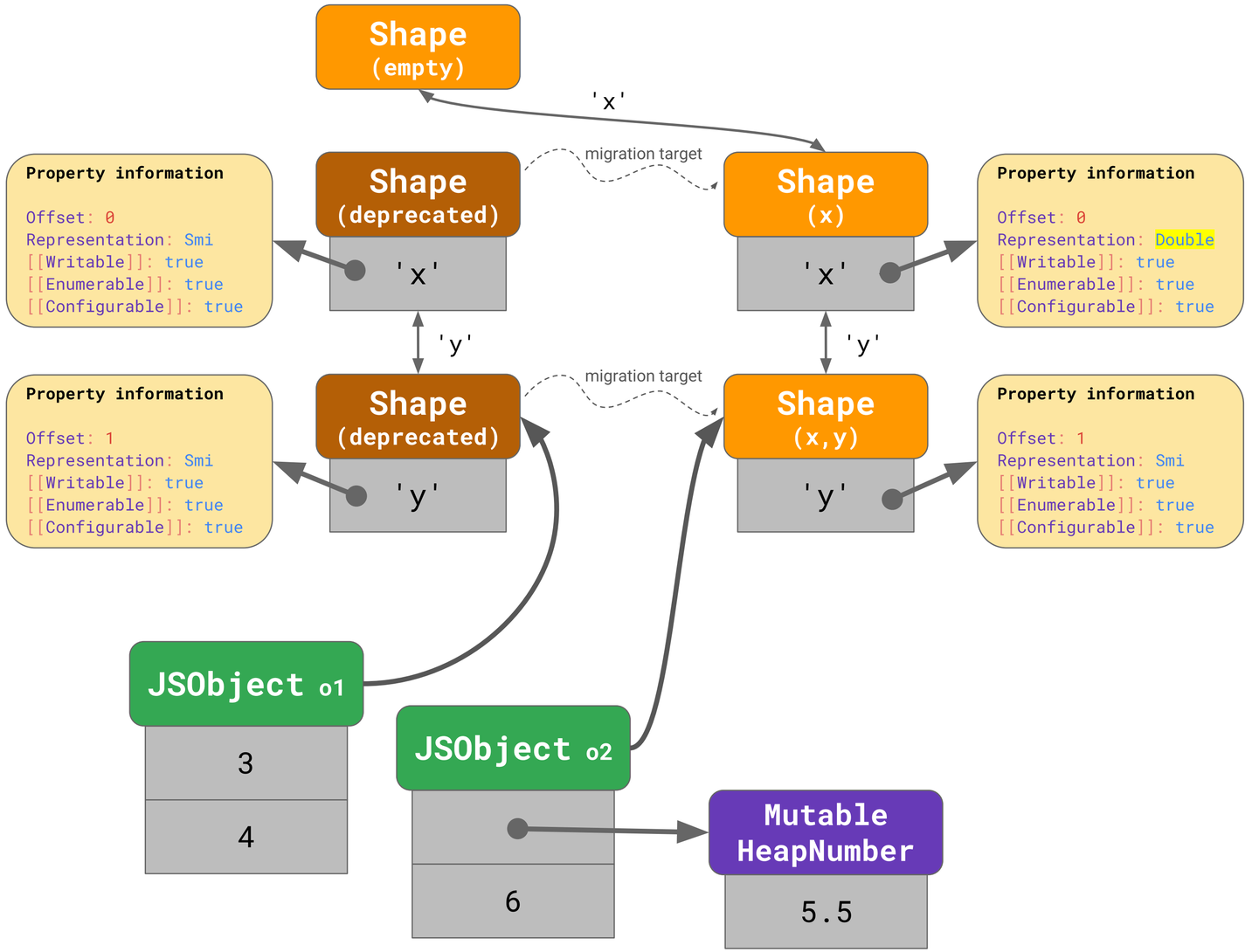
Inline cache 机制 – Shape 迁移

网易考拉



Inline cache 机制 – Shape 迁移

网易考拉



Inline cache 机制 – Shape 迁移

网易考拉

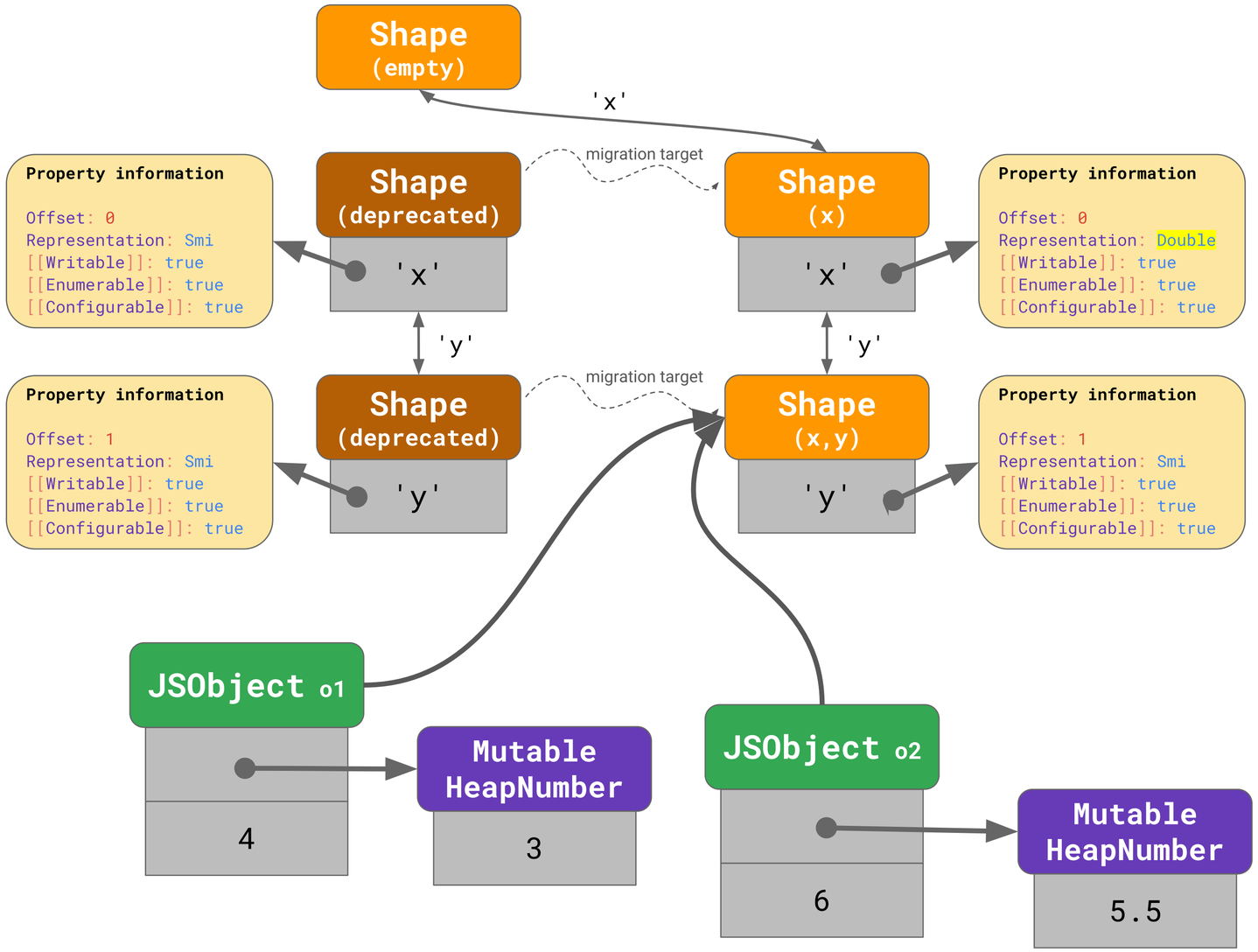
https://bugs.chromium.org/p/v8/issues/detail?id=8538

Object.preventExtensions

Object.freeze

Object.seal

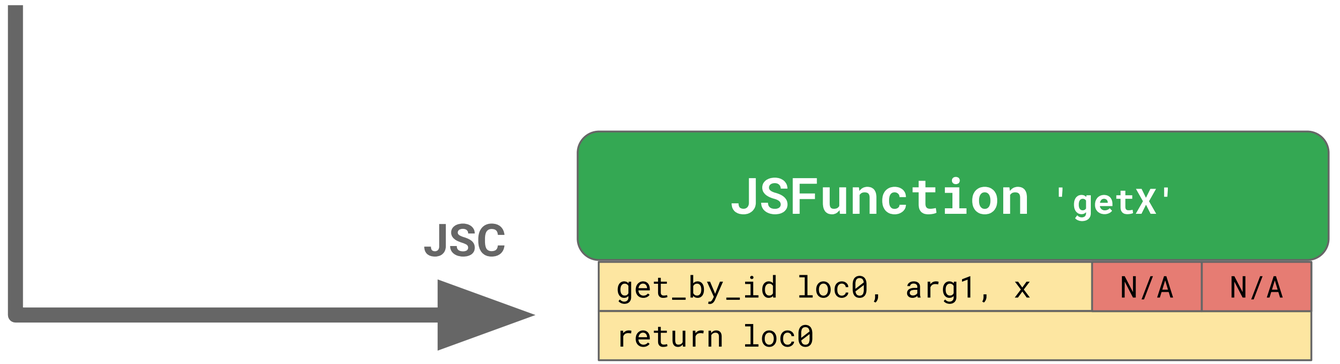
可能引起的性能问题



内联缓存（Inline Cache）

网易考拉

```
function getX(o) {
  return o.x;
}
```



内联缓存（Inline Cache）

网易考拉

1. 从对象的 shape 中获得全部keys
2. keys 遍历获取到key “x” 存储的位置。
3. 调用特定的方法读取属性值。

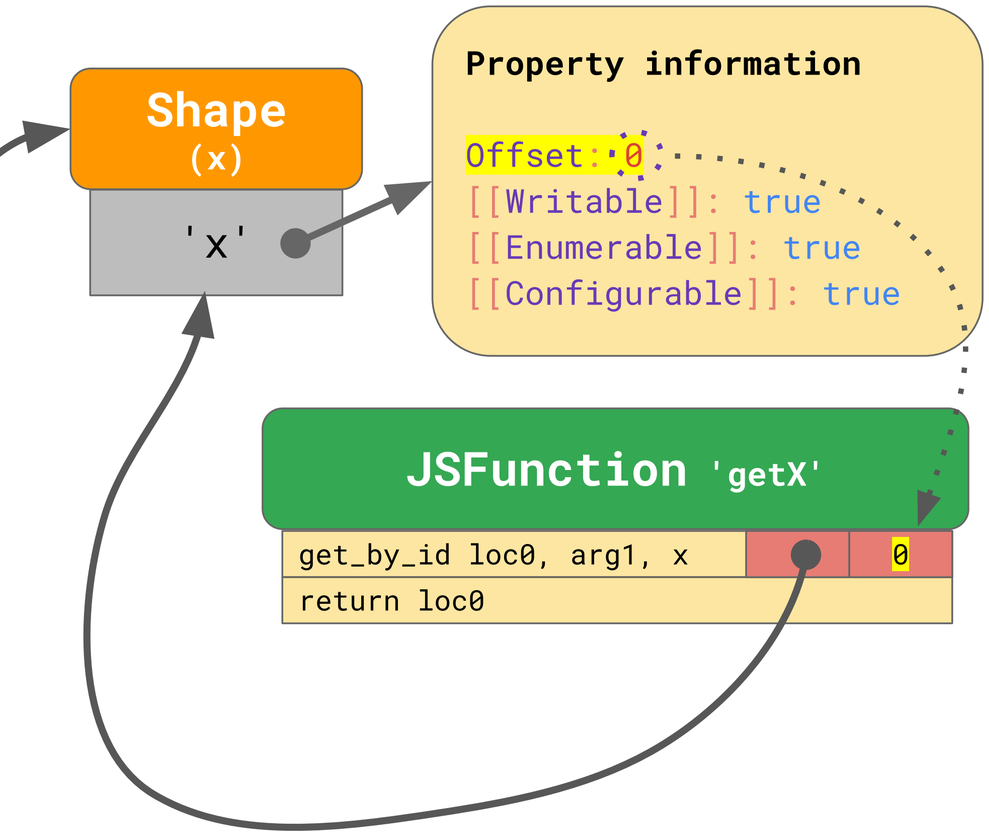
如果每次传入obj的shape都是一样的，此处可以优化

```
function getX(obj) {
    return getById(obj, "x");
}

function getById(obj, key) {
    // 获取对象的shape中的keys
  var keys = obj.shape.keys;
    var foundKeyIndex = -1;

    // 遍历shape中的key找到offset
    for (var i = 0; i < keys.length; i++) {
        if (keys[i] === key) {
            foundKeyIndex = i;
            break;
        }
    }

    //没找到，返回
    if (foundKeyIndex === -1) {
        return undefined;
    }

    var keyDesc = keys[foundKeyIndex];
    // 调用方法取出对应offset的value
    return obj.READ_FIELD(keyDesc.offset);
}
```

内联缓存（Inline Cache）

网易考拉

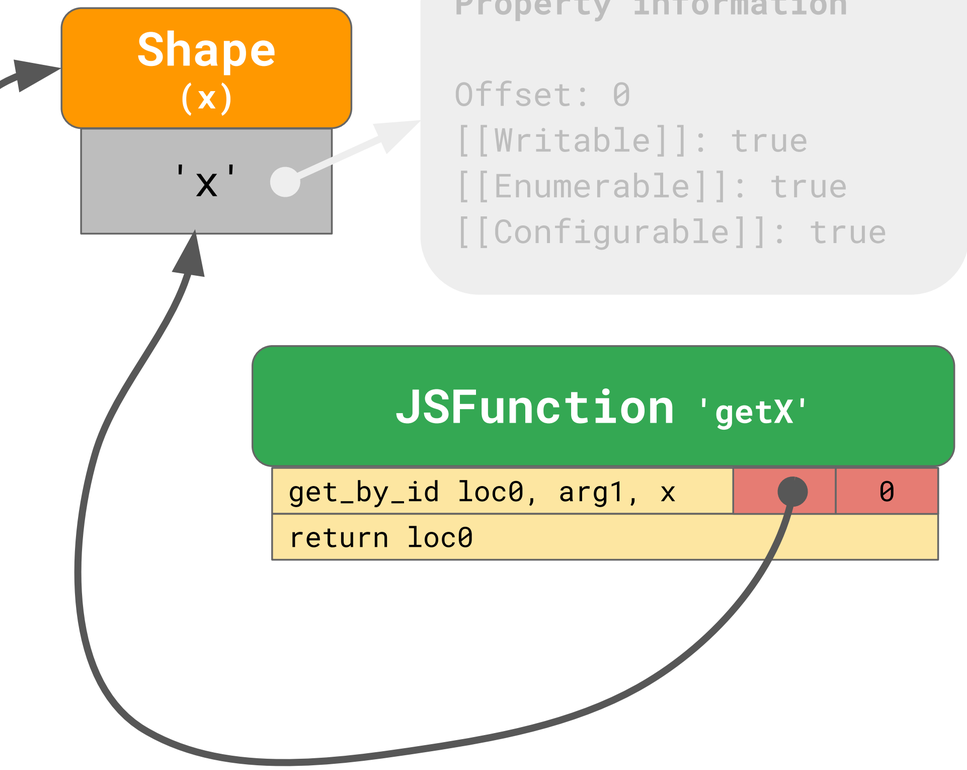
```
function getX(o) {
  return o.x;
}

getX({ x: 'a' });
```



内联缓存（Inline Cache）

网易考拉

```
function getX(o) {
  return o.x;
}

getX({ x: 'a' });
getX({ x: 'b' });
```



内联缓存（Inline Cache）

网易考拉

1. 判断对象的 shape 与之前缓存的是否相同
2. 如相同，直接从缓存的offset位置读取
3. 如不同，调用旧逻辑并把新的shape缓存下来

```
function getX(obj) {
    return getById_with_inline_cache(obj, "x");
}

const getById_with_inline_cache = (function () {
    var cache = {
          shape: null,
          offset: -1
     };
    return function (obj, key) {
        if (obj.shape === cache.shape) {
            // shape和缓存的shape一致，直接用缓存的offset
            if (cache.offset >= 0) {
                return obj.READ_FIELD(cache.offset);
            }
        } else {
            // 省略和之前的getById 逻辑一致
      // ...
            var keyDesc = keys[foundKeyIndex];
            cache = {
                shape: obj.shape,
                offset: keyDesc.offset
            };
            return obj.READ_FIELD(keyDesc.offset);
        }
    }
})()
```

内联缓存（Inline Cache）

网易考拉

```
console.time('measure');
for (var i = 0; i <= 2e6; i++) {
    getX({[`a${i}`]: i});
}
console.timeEnd('measure');
```

I = 0 => { a0 : 0 }
I = 1 => { a1 : 1 }
…

每次key不同，产生不同的shape导致inline cache不起作用

总结

网易考拉

1. 尽量写小而美的函数（60KB限制，用代码生成器可能会遇到）
2. 最好相同顺序初始化对象（减少不同类型shape的产生）
3. 函数的传入对象最好是固定的shape（利用inline cache）
4. 以上全错。性能优化的必要性（脱离实际场景聊性能都是伪科学）
5. 引擎还有很多无法优化的场景 deps/v8/src/bailout-reason.h

网易考拉

Q&A

NETEASE
Kaola Center Training

感谢聆听！