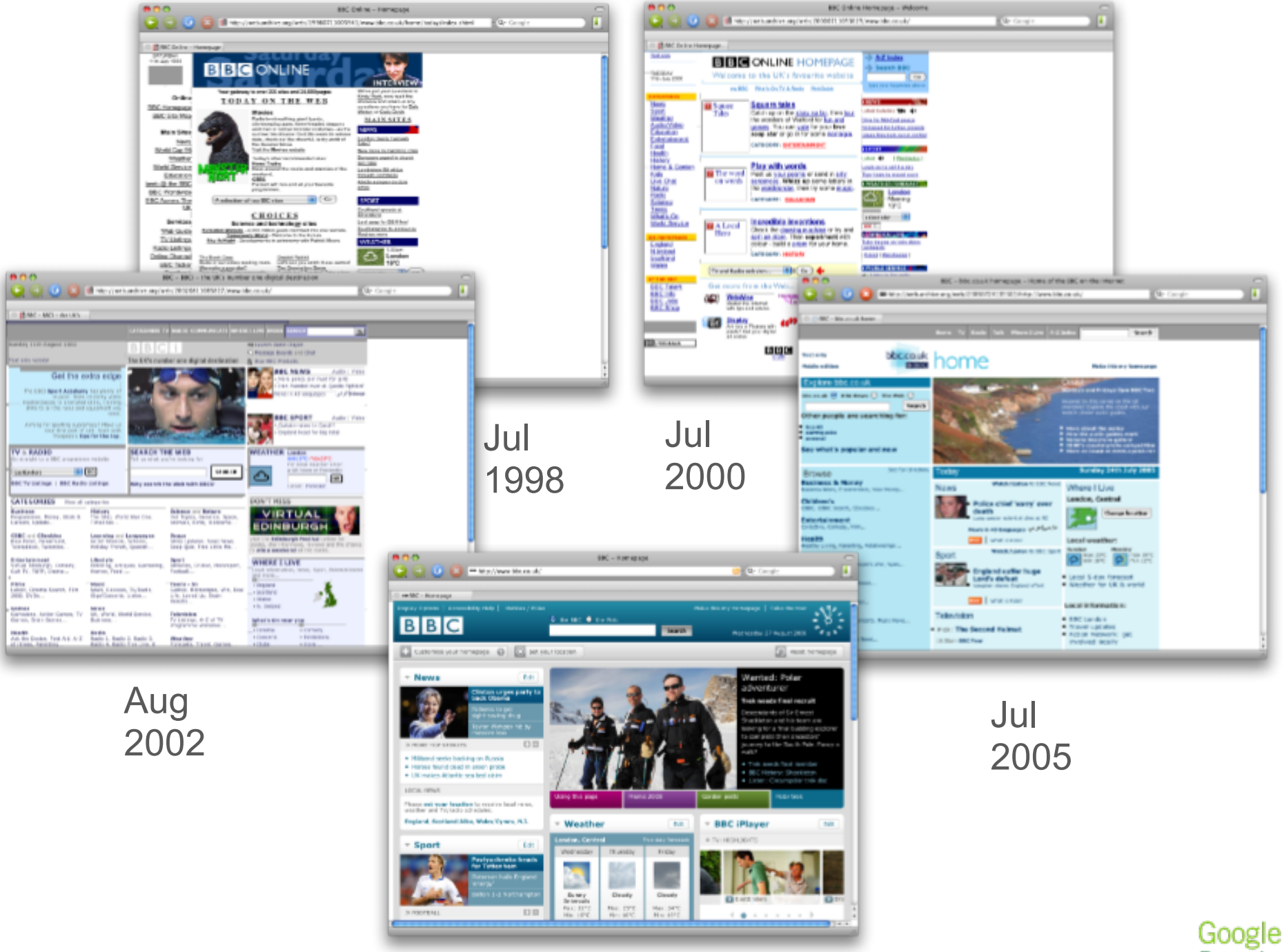

Google
Developer
Day2009

V8 JavaScript 引擎内部构造

Feng Qian（钱锋）
June 2009

Original presentation by Mads Ager

Google Developer Day 2009



Google Developer Day2009

为什么开发一个新的JS引擎？

- 在开发V8前，已有的JS引擎速度达不到新应用的要求：
  - 解释器（interpreters）解释执行AST或bytecodes
  - 简单的内存管理技术带来很大的GC停顿
- 高性能的JavaScript引擎是Web应用持续创新的关键
- 最佳的方式是从头开发
- 目的是要把JavaScript性能推向一个新的高度

挑战

- JavaScript是一个高度动态的语言
- 对象只是哈希表（hash map）
- 可以随时添加，修改，删除对象属性（property）
- 执行时可以修改原型链（prototype chain）
- “eval” 能改变调用的上下文（context）
- “with” 能动态的添加对象到作用域链（scope chain）里

概要

- V8 设计
- V8 内核
  - 隐藏类（hidden classes）
  - 内联缓存（inline caching）
  - 精确的分代垃圾收集(Precise generational garbage collection)
- Irregexp: 一个新的正则表达式处理器
- 未来发展

V8 设计

设计目的

- 让大的面向对象的程序很好的运转
- 快速访问对象属性
- 快速功能调用
- 快速和扩展的内存管理

V8的关键部件

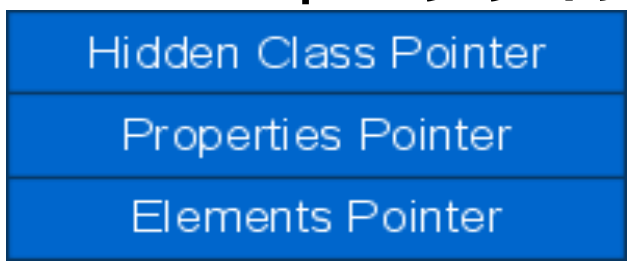
- 隐藏类型（hidden classes）和类型转换（class transitions）
- 编译成机器代码
- 内联缓存（inline cache）
- 高效的分代垃圾回收技术（generational garbage collection）

V8内核

V8内存模型(Memory Model)

- 32位的标记指针
- 对象大小4字节对齐，地址的最低两位可以用做标记（tag）
- 31位带符号整数可以直接用标记和指针区分开

- 一个基本的JavaScript对象有3个字（word）

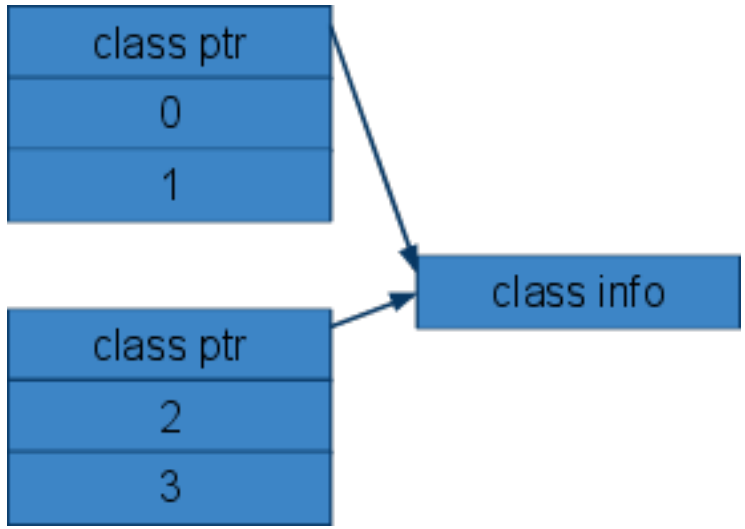


隐藏类型 Hidden Classes

- 充分利用静态类型面向对象语言的优化技术
- 引入隐藏类型的概念
- 隐藏类型把具有相同结构的对象组合到同一类型

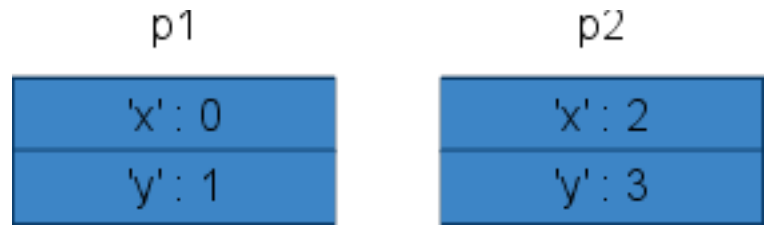
Java:

```
class Point {
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
private:
  int x;
  int y;
}
Point p1 = new Point(0, 1);
Point p2 = new Point(2, 3);
```



JavaScript:

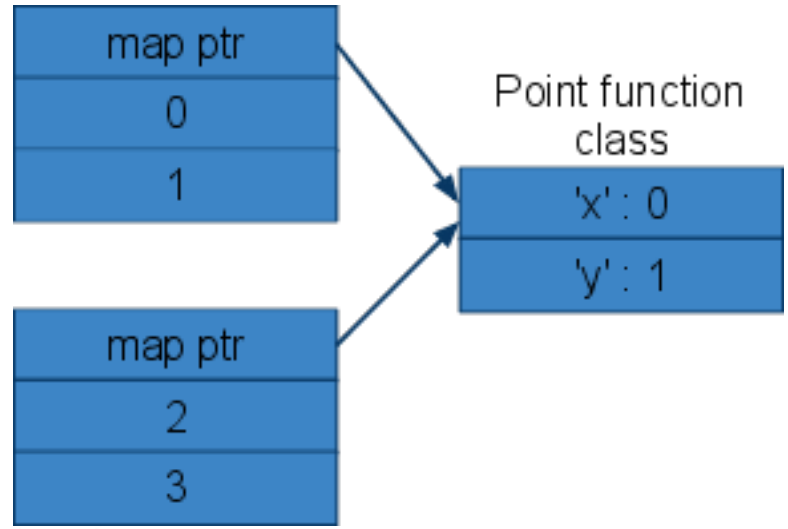
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0, 1);
var p2 = new Point(2, 3);
```



隐藏类型举例

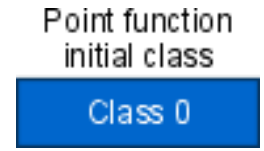
- 用相同方式构造的JavaScript对象有相同的隐藏类型

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



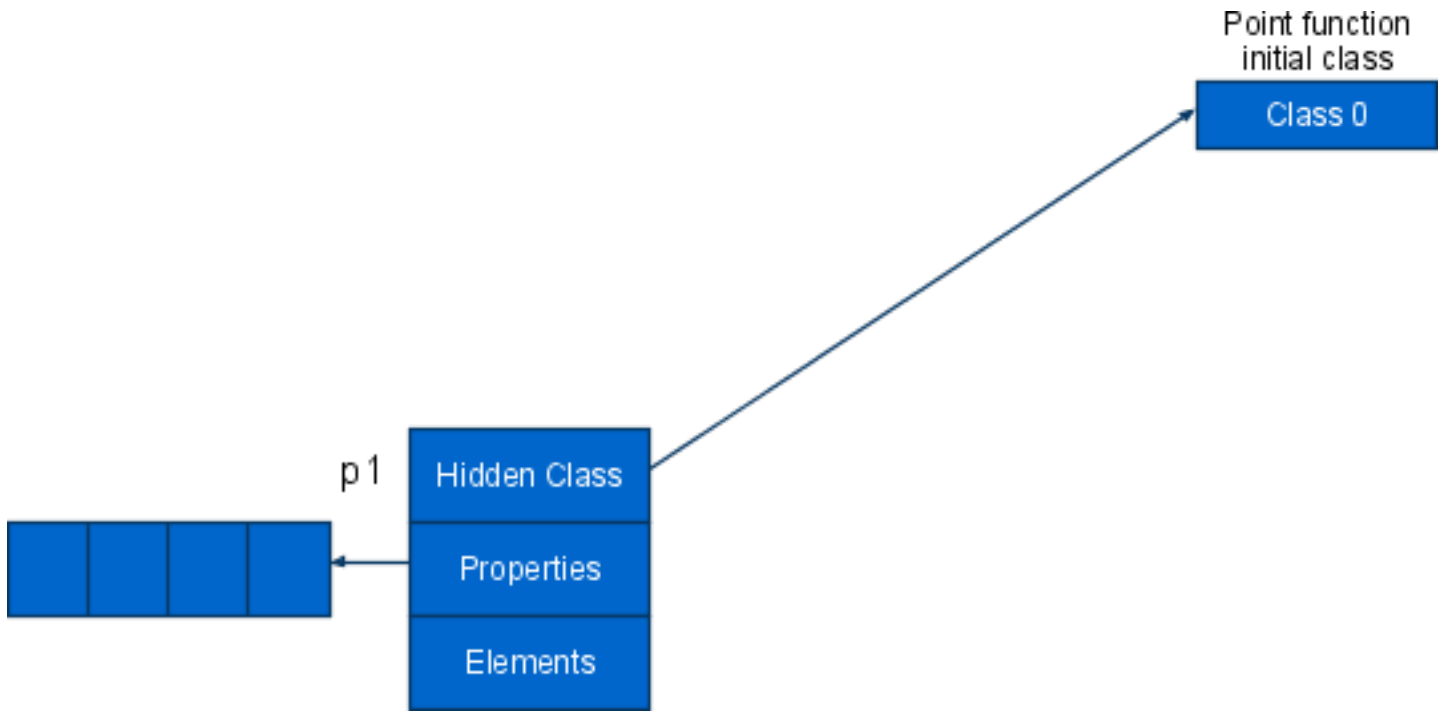
隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



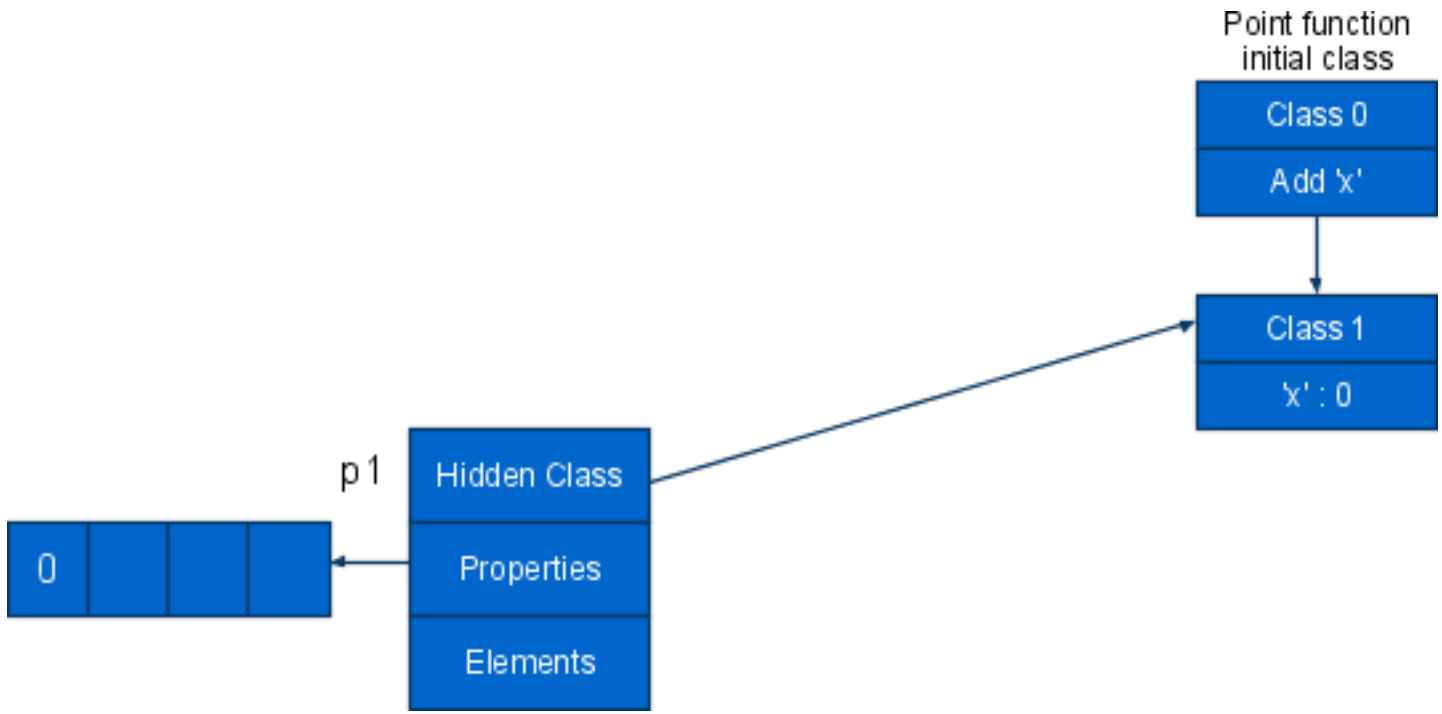
隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



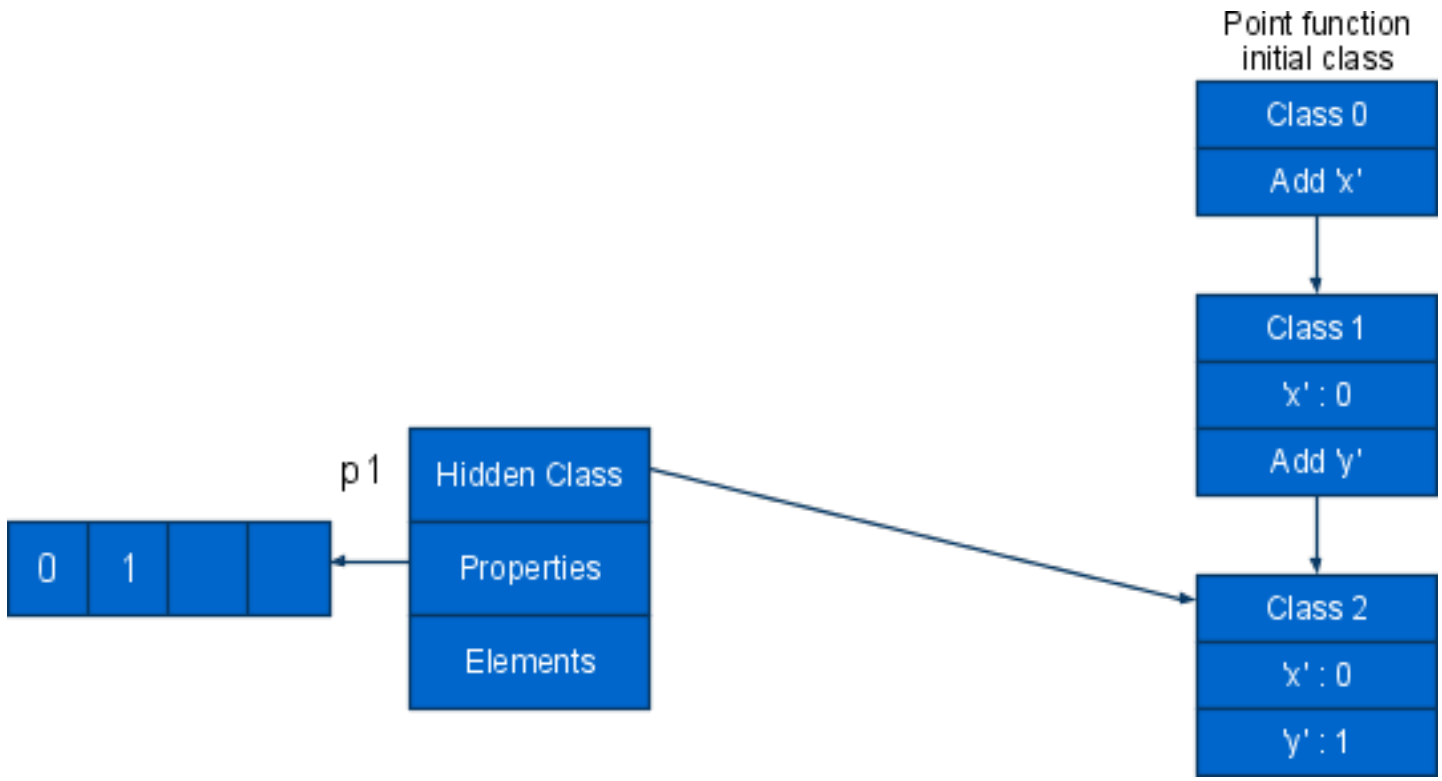
隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



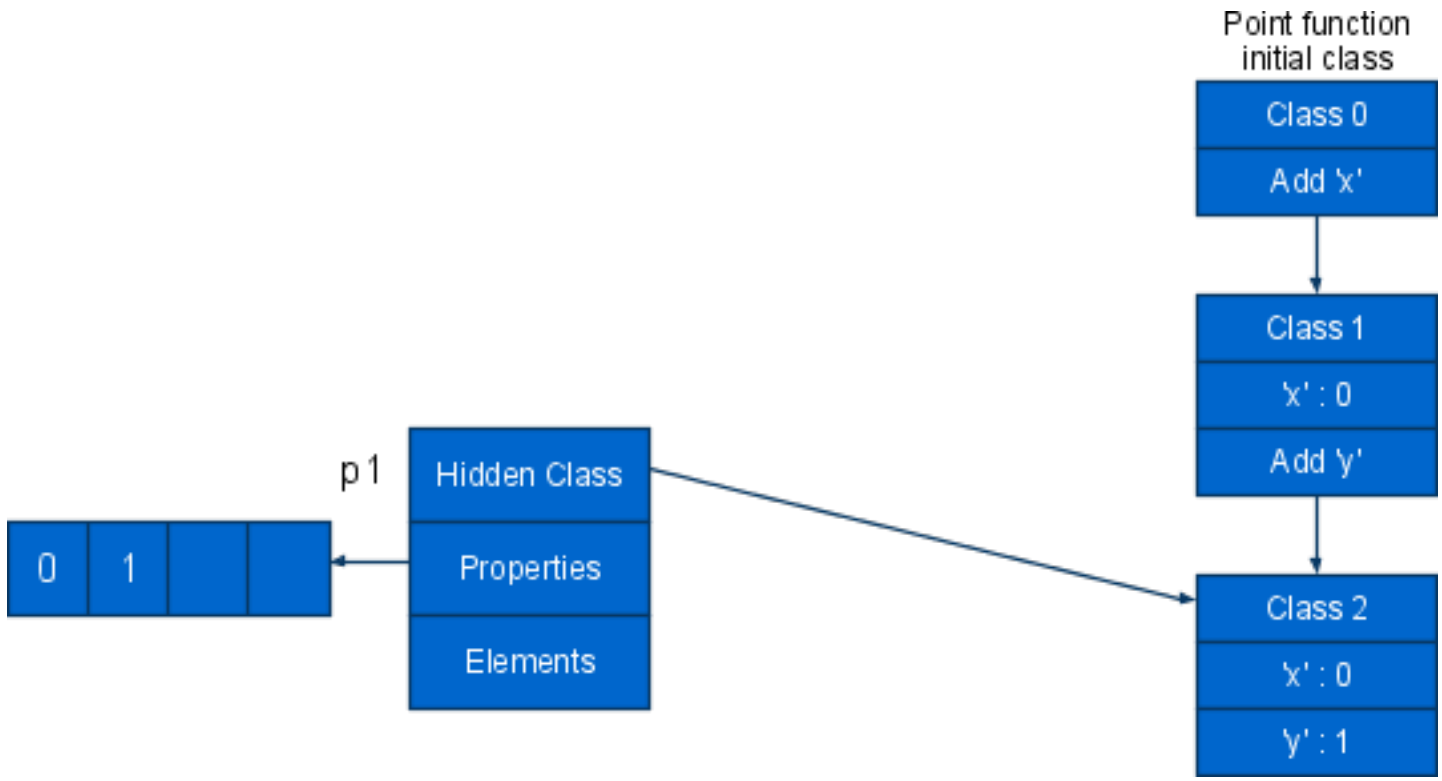
隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



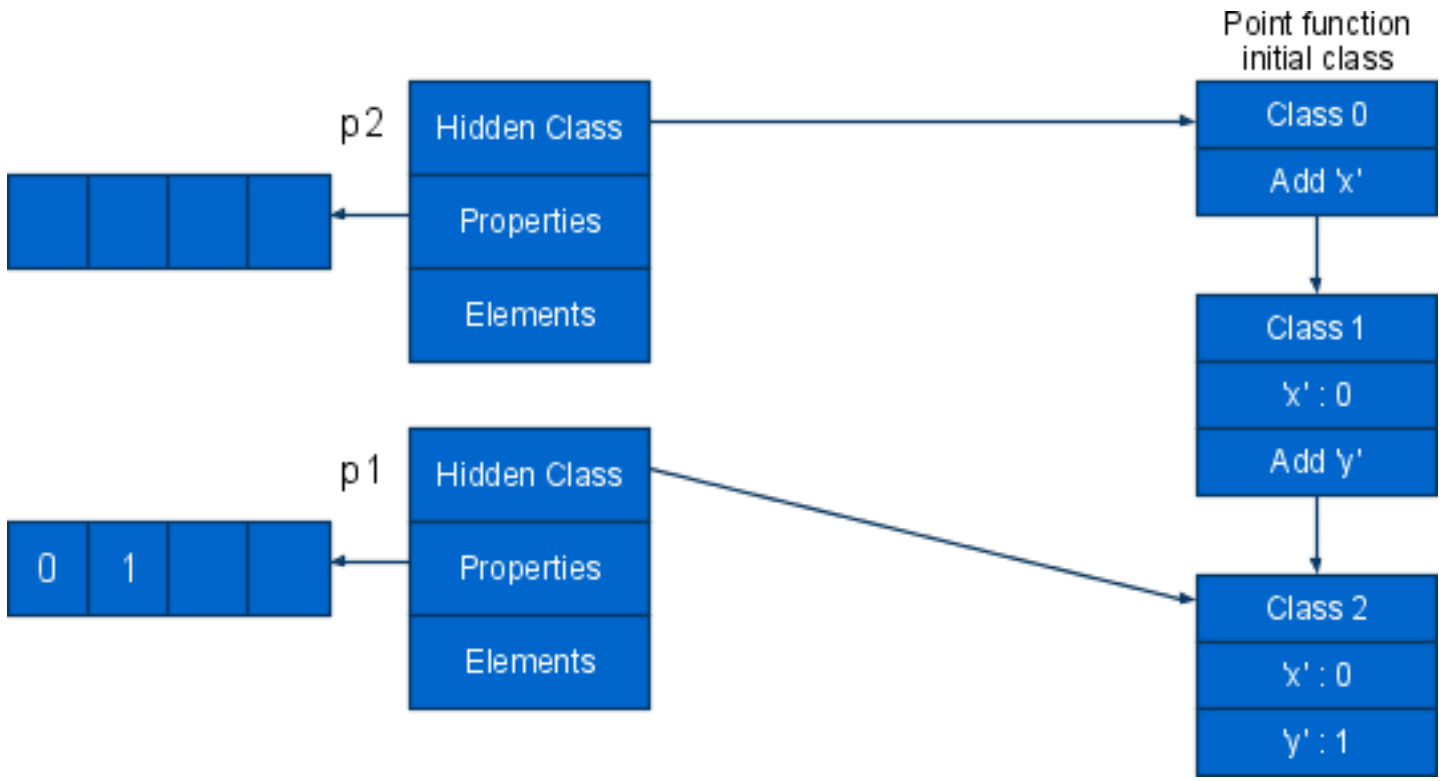
隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



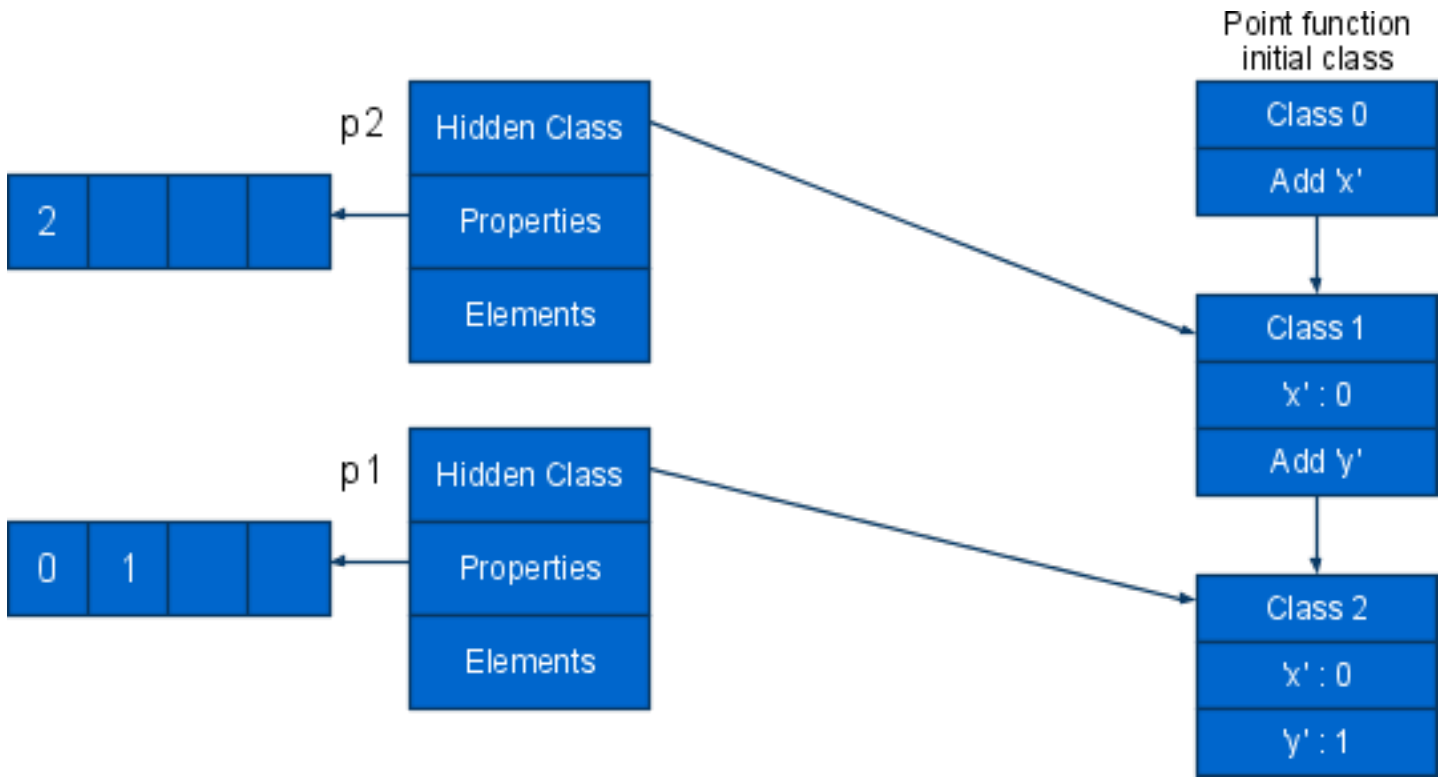
隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



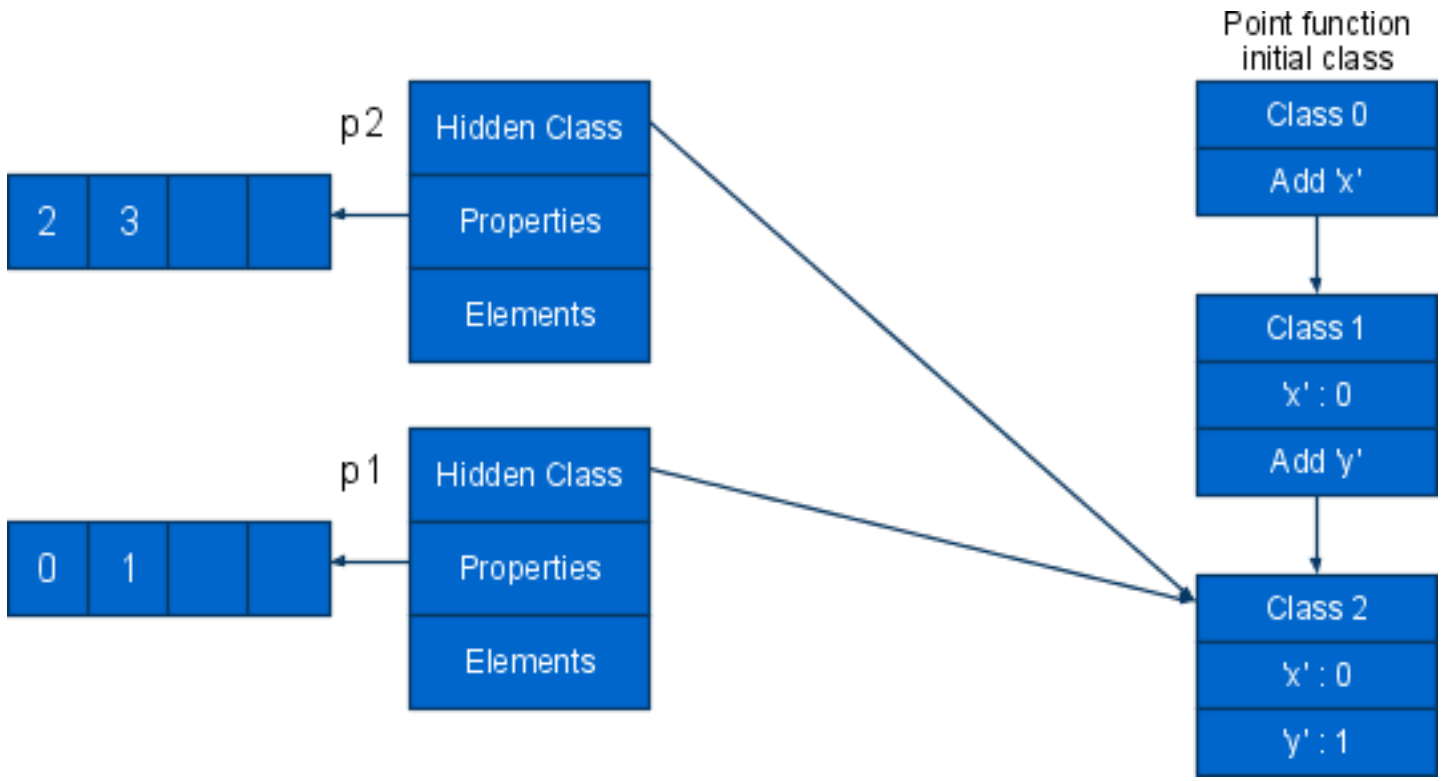
隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



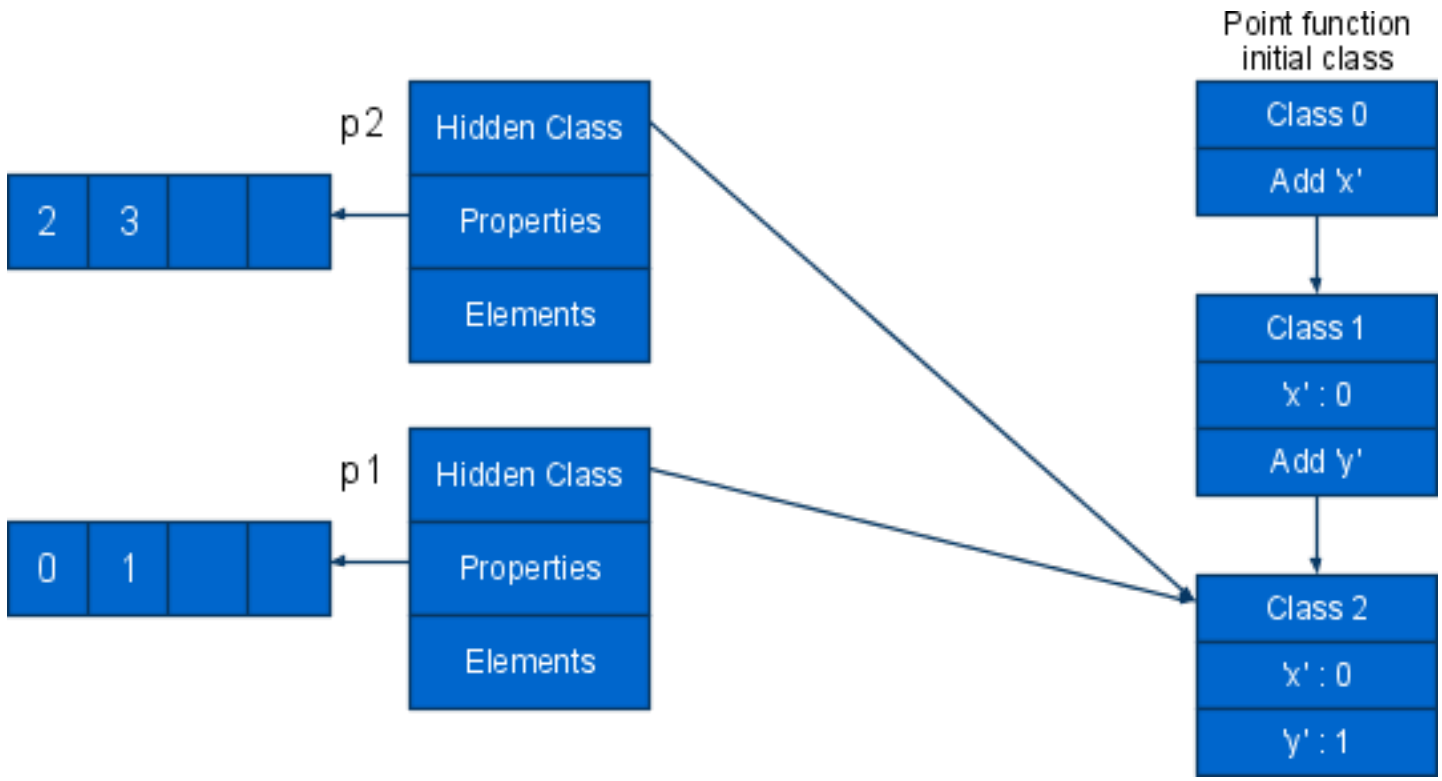
隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



隐藏类型举例

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



JavaScript的动态性

- 在90%的情况下，程序中属性访问点只见到具有相同类型的对象
- 隐藏类型为应用静态面向对象语言的优化技术提供了足够的结构
- 在编译时候并不知道隐藏类型
- 利用动态代码生成和内联缓存（inline caching）技术

嵌入缓存（Inline Caching）

... (p points to p1)

... = p.x 

...

Lookup 结果：

p的隐藏类是class 2，x 的 offset 是 0

动态生成一个特殊的代码段，修改p.x调用地址

特殊代码段

0xf7c0d32d (size = 37):

```
 0 mov eax,[esp+0x4]
 4 test al,0x1
 6 jz 32
12 cmp [eax+0xff],0xf78fab81
19 jnz 32
25 mov ebx,[eax+0x3]
28 mov eax,[ebx+0x7]
31 ret
32 jmp LoadIC_Miss
```

特殊代码段

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]        ; receiver load
 4 test al,0x1              ; object check
 6 jz 32
12 cmp [eax+0xff],0xf78fab81
19 jnz 32
25 mov ebx,[eax+0x3]
28 mov eax,[ebx+0x7]
31 ret
32 jmp LoadIC_Miss
```

特殊代码段

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]         ; receiver load
 4 test al,0x1               ; object check
 6 jz 32
12 cmp [eax+0xff],0xf78fab81 ; class check
19 jnz 32
25 mov ebx,[eax+0x3]
28 mov eax,[ebx+0x7]
31 ret
32 jmp LoadIC_Miss
```

特殊代码段

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]         ; receiver load
 4 test al,0x1               ; object check
 6 jz 32
12 cmp [eax+0xff],0xf78fab81 ; class check
19 jnz 32
25 mov ebx,[eax+0x3]         ; load properties
28 mov eax,[ebx+0x7]         ; load property
31 ret
32 jmp LoadIC_Miss
```

特殊代码段

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]         ; receiver load
 4 test al,0x1               ; object check
 6 jz 32
12 cmp [eax+0xff],0xf78fab81 ; class check
19 jnz 32
25 mov ebx,[eax+0x3]         ; load properties
28 mov eax,[ebx+0x7]         ; load property
31 ret
32 jmp LoadIC_Miss           ; fallback to
                             ; generic lookup
```

内联缓存

... (p points to p2)

... = p.x;

...

内联缓存状态

- 缓存的三个状态
  - 未初始化(Uninitialized)
  - 单态（Monomorphic）
  - 多态（Megamorphic）
- 程序可以自己修改机器代码
- 在全局GC的时候，内置缓存被清空
  - 可以删除没用的代码段
  - 所有嵌入缓存有机会重新回到单态

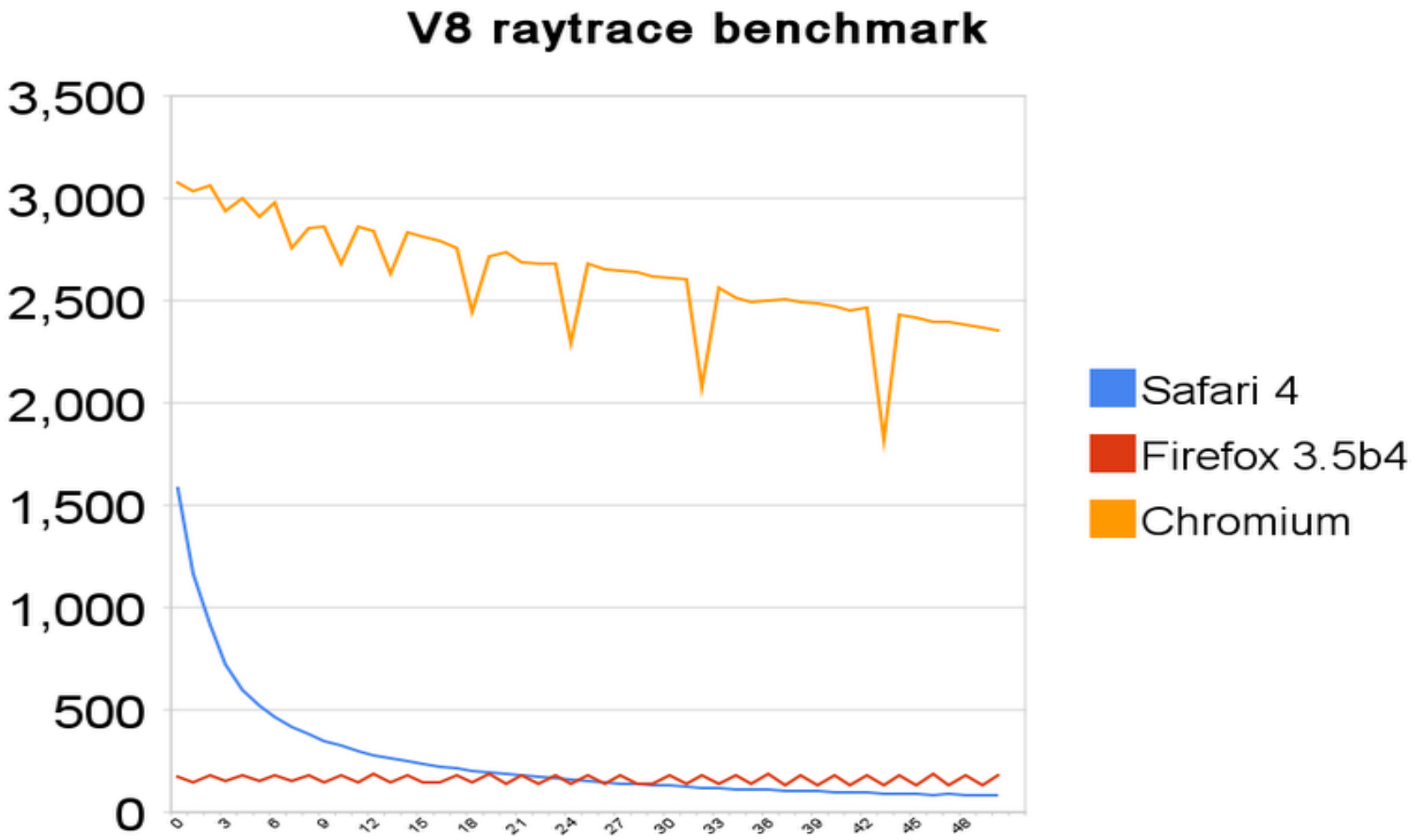
高效内存管理

- 精确分代垃圾收集技术（Precise Generational GC）
- 对分为两代：
  - 年轻代是一个较小的连续地址空间，经常性的收集
  - 年长代又被分成几个空间，不经常收集
    - 可执行的代码空间
    - 隐射表空间（存放隐藏类）
    - 大的对象空间（>8K）
    - 数据对象空间（存放不含指针的对象）
    - 普通对象空间
- 先在年轻代里创立对象，没有被收集掉的对象被移到年长代

垃圾收集技术

- 清除收集(Scavenge)
  - 只在年轻代使用复制收集算法
  - 间歇时间 ~2ms
- 全堆非压缩收集（Full non-compacting collection）
  - 对两代使用标记－清除收集算法（Mark-Sweep）
  - 空闲内存加入空闲列表
  - 可能产生碎片
  - 间歇时间 ~50ms
- 全堆压缩收集（Full compacting collection）
  - 对两代使用标记－压缩收集算法（Mark-Compact）
  - 间歇时间 ~100ms

可扩展性实验－运行速度



Bigger is better!

Irregexp: 新的正则表达式引擎

Irregexp 内核

- 从输入的正则表达式构造一个自动机
- 对自动机进行分析和优化
- 生成机器代码
- 使用一些技巧来避免回溯(backtracking)
- 重新编排操作顺序，优先执行快速的操作

Irregexp举例

- 先用蒙板(masks)匹配可选项（alternatives）的公共部分
- 要匹配
  /Sun|Mon/
- 先找
  /??n/
- 避免许多不必要的回溯

Irregexp 举例

- 对 ASCII 码字符串一次匹配4个字节
- 对

  /foobar/
- 查找

  0x666f6f62 0x6172

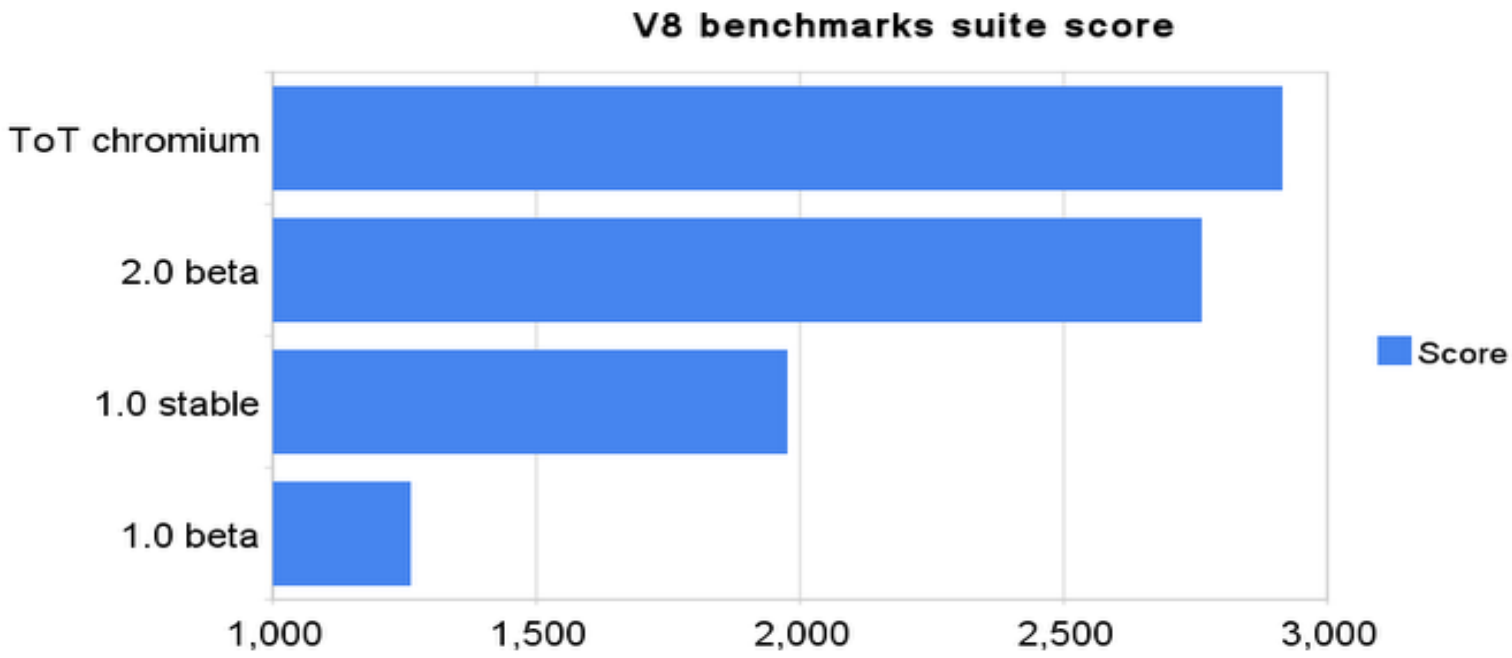
Irregexp 举例

- 先运行快捷操作
- 对
  /([fF]oo[bB]ar)/

- 运行下列动作
  - 在位置1和4匹配 **oo** 和 **ar**
  - 在0位置匹配 **[fF]**
  - 在3位置匹配 **[bB]**
  - 实现 `capture`

总结

- V8设计目标是速度和可伸展性
- 把JavaScript性能推向一个新的高度



未来工作

- 寄存器分配和代码优化
- 只生成一个通用的机器代码
  - 可以利用许多执行时的动态信息和假设来生成优化的机器代码，当假设不成立时能够回到通用代码
- 嵌入缓存是基于代码段的调用
  - 直接嵌入快速常见的代码段，避免调用
- 缓慢的 write barrier
  - 正在实验不同的实现
- 对JavaScript Global Object没有特殊处理
  - 可以生成和上下文(Context)专门的代码，读取属性可以变的很快


Google
Developer
Day2009