JavaScript and V8
A functional-ish language and implementation in the mainstream
Andreas Rossberg, Google Munich
rossberg@google.com
CEFP 2011, Budapest
Outline

- JavaScript
  - Background
  - Functional concepts
- V8
  - Inline Caching
  - Crankshaft
JavaScript

- Originally just a language for scripting Netscape
- Now the **lingua franca** of the web
- Increasingly used on the **server-side** as well
  - cf. the hype around *node.js*
- Dozens of **compilers** have JS back-ends
- More applications written in JS these days than in any other language!
JavaScript Applications

- Large applications are written in JavaScript
- GMail consists of ~1Mloc
- Performance-relevant ones, too!
  - *Angry Birds*, now natively in JS/HTML5 :-)
  - *Mandreel*, a game development framework that compiles existing C++/OpenGL code to JS/WebGL
  - *Emscripten*, LLVM to JS compiler that can run Doom
  - *jslinux*, a software x86 PC emulator that can boot Linux
First-class functions

function adder(x) {
    return function(y) { return x + y }
}

function makeCounter() {
    var i = 0
    return function() { return ++i }
}

function curry2(f) {
    return function(x) { return function(y) { return f(x, y) } }
}
Higher-order functions

```javascript
var a = [3, 0, -4, 5, 10]

var sum = a.reduce(function(x, y) { return x + y }, 0)

var b = a.filter(function(x) { return x >= 0 })
  .map(Math.sqrt)
  .sort (function(x, y) { return y - x })

b.forEach(print)
```
function Point(x, y) {
    this.getX = function() { return x }
    this.getY = function() { return y }
    this.move = function(dx, dy) { x += dx; y += dy }
}

var p = new Point(3, 4)
p.move(-5, +1)
Functional Encapsulation

(function() {
// do or compute stuff...
return result
})()
Continuation-passing

```javascript
function post(url, data, callback) {
    var client = http.createClient(url.port, url.hostname)
    var request = client.request("POST", url.pathname, ...)
    request.write(data)
    request.end()
    request.on("response", function(response) {
        var body = ""
        request.on("data", function(chunk) { body += chunk })
        request.on("end", function() {
            callback(response.statusCode, body)
        })
    })
}
```
Continuation-passing

```javascript
ioThis(theseArgs, function(thisResult) {
  ioThat(thoseArgs, function(thatResult) {
    ioYonder(yonderArgs, function(yonderResult) {
      // ...
    })
  })
})
```
The Not-so-functional

- Statements vs expressions, C-like syntax
- Everything is mutable for everybody
- Emphasis on object-oriented programming
- Weak type structure
- Parts were “designed” in a semantics-free space
- But things will improve significantly with EcmaScript 6 (due 2013)
The Bad: Scoping

```javascript
function f(n) {
    for (i = 0; i < n; ++i) {
        // oops, introduces global variable!
        // do something...
    }
}

function g() {
    for (i = 0; i < 100; ++i) {
        // double oops, never terminates!
        f(10)
    }
}
```
The Bad: “this”

```javascript
var myObject = {
    value: 5,
    output: function() { print(this.value) }
}

myObject.output()
myObject.output()
myObject.output()
```
The Bad: “this”

```javascript
var myObject = {
    value: 5,
    output: function() { print(this.value) }
}

var f = myObject.output
f(); f(); f() // oops, prints “undefined”!?!?!
function f(x) {
    g()
    return x
}

f(42)

function g() {
    f.arguments[0] = 666
    // wtf?!!
}
The Stupefying: "arguments", "caller", "callee"
V8

- Google’s open source virtual machine for JavaScript
- used in Chrome and other environments (e.g. Android, node.js)
- no interpreter, just-in-time compilation through-out
- two compilers: “full” and optimizing (aka Crankshaft)
- adaptive optimization in Crankshaft
V8 Performance (early days)

![JavaScript Benchmark Component Scores Graph](image-url)
The Problem

- JavaScript is a “dynamic” language
- ...which means that every primitive operation is expensive
- For example: field lookup is (very!) expensive, due to
  - untyped objects, dynamic field names, dynamic field addition & removal, mutable inheritance chain, arrays as objects, interceptors...
- Likewise, many other operations:
  - field access, global variable access, array access, stores, function calls, overloaded operators, conversions, other builtin functions...
Inline Caching

- Solution: cache lookup path for given object type
- In fact, do it inline in generated code
- Self-modifying code!
- Originally invented in the context of Smalltalk [Deutsch & Schiffmann POPL’84]
- V8 was first JavaScript VM to employ it
- Requires efficient structural type checks ("maps")
Maps

> function Point(x, y) {
>   this.x = x
>   this.y = y
> }

> var p = new Point(3, 4)
> p.x
Maps

function Point(x, y) {
  this.x = x
  this.y = y
}

var p = new Point(3, 4)
p.y

var q = new Point(1, 1)
Inline Caching

Runtime::Load:
1. looks up p.y in a slow but generic manner,
2. generates a lookup stub specialized for y and map2,
3. patches its caller’s call instruction to target the stub.

Source
...
p.y
...

Machine code
...
mov eax, p
mov ebx, y
call Runtime::Load
...

Stub (cached)
cmp [eax], map2
jne Runtime::Load
mov eax, [eax + 4]
mov eax, [eax + 8]
ret
Runtime::Load:

1. looks up p.y in a slow but generic manner,
2. generates a lookup stub specialized for y and map2,
3. patches its caller’s call instruction to target the stub.
Crankshaft

- Separate, optimizing backend for V8 (Chrome 10+)
- Used only for hot paths through hot functions
- Based on statistical profiling and type feedback
- Type information harvested directly from stub code!
- Inlines IC code stubs
- …which enables a lot of additional optimizations
- Deoptimization upon type miss
## Inline Caching

<table>
<thead>
<tr>
<th>Source</th>
<th>Machine code</th>
<th>Stub (cached)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>cmp [eax], map2</td>
</tr>
<tr>
<td>p.y</td>
<td>mov eax, p</td>
<td>jne Runtime::Load</td>
</tr>
<tr>
<td></td>
<td>mov ebx, y</td>
<td>mov eax, [eax + 4]</td>
</tr>
<tr>
<td></td>
<td>call stub</td>
<td>mov eax, [eax + 8]</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>ret</td>
</tr>
</tbody>
</table>
Optimized Code

Source
...
    p.y
...

Optimized machine code
...
    mov eax, p
    cmp [eax], map2
    jne Runtime::Deopt
    mov eax, [eax + 4]
    mov eax, [eax + 8]
...

Stub (cached)
    cmp [eax], map2
    jne Runtime::Load
    mov eax, [eax + 4]
    mov eax, [eax + 8]
    ret

Runtime::Deopt:
1. discards optimized code for entire function,
2. (lazily) rewrites all active stack frames for that function,
3. jumps to corresponding point in unoptimized code.
IC States

- uninitialized
  - first execution
  - initialized
    - miss
      - monomorphomorphic
        - 2..M different misses
          - polymorphomorphic
            - more than M misses
              - megamorphomorphic
The Big Picture

- Source function
- Unoptimized machine code
- Optimized machine code
- IC stub
- C++ runtime code
- JS source code
- Generated machine code
Summary

- JavaScript is the most used functional-ish language
- Vital to web infrastructure, increasing server-side use
- V8 repeatedly raised bar for JS performance
- Uses advanced dynamic compilation techniques

http://code.google.com/p/v8