RubyJS

“Efficient” Ruby to Javascript Compilation

RubyConf 2007, Charlotte
Michael Neumann
mneumann@ntecs.de

Agenda

- Why RubyJS?
- How is it implemented?
  - Design decisions
  - Difficulties
- Applications
  - Ruby Web Toolkit

Example #1

```
# hw.rb
require 'rwt/DOM'

class HelloWorld
  def self.main
    out = DOM.getElementById('out')
    DOM.setInnerText(out, 'hello world')
  end
end
```

```
> rubyjs_gen --main HelloWorld hw.rb > hw.js
```

```
<html>
  <body>
    <script src=”hw.js”></script>
    <a href=”#” onclick=”main()”>say hello</a>
    <div id=”out”/>
  </body>
</html>
```

Example #1

Example #1

```
# hw.rb
require 'rwt/DOM'

class HelloWorld
  def self.main
    out = DOM.getElementById('out')
    DOM.setInnerText(out, 'hello world')
  end
end
```

```
> rubyjs_gen
```

```
/* HelloWorld.main */
function(){var self,_a,_b;
_b=nil;
self=this;
_a=$w.$aH(nil,"out");
_b=$w.$bA(nil,_a,"hello world");
return _b}
```

Example #2

```
require 'rwt/DOM'
require 'rwt/HTTPRequest'
require 'json'

class JsonTest
  def self.main
    out = DOM.getElementById('out')
    HTTPRequest.asyncGet('/json') do |resp|
      DOM.setInnerText(out, JSON.load(resp).inspect)
    end
  end
end
```

```
> rubyjs_gen --main JsonTest jsontest.rb > jsontest.js
```

Part I

Why RubyJS?

Why not stick with Javascript?

It's all about *Applications*
not Scripts

Why not stick with Javascript?

(More) Error prone

Local variable declaration

```ruby
$i = 0

def m
  for i in 0..9 do ...
end
```

```javascript
var i = 0;

function m() {
  for (i=0; i<10; i++) ...
}
```

Ooops, i is
a global

Why not stick with Javascript?

Readability

Line Noise

```
3.times { @i += 1 }
```

```
(3).times(function() { self.i += 1; });
```

No argument parsing

```
def test(i, j=”blah”, *args)
```

```
function test(i, j) {
  var args = [];
  if (j===undefined) j=”blah”;
  for (var i=2; i < arguments.length; i++)
    args.push(arguments[i]);
}
```

OO?

```ruby
class Animal
  def say_hello() end
end

class Cat < Animal; end
```

```javascript
// using prototype.js
var Animal = Class.create(Object, {
  say_hello: function() { ... }
});

var Cat = Class.create(Animal, {
  ...
});
```

Why not stick with Javascript?

Missing abstractions

not Everything.is_a?(Object)

- `null` is special in that it has no properties and no prototype!

```
var a = [1, null, 2];

a.each(function(e) {
  alert(e.toString());
});
```

e has no properties

No method_missing

```
aProxy = Proxy.new(obj)
aProxy.say_hello(“Michael”)
```

```
aProxy = new Proxy(obj);
aProxy.send(“say_hello”, “Michael”);
```

Harder to Deploy

- No `require` for explicit dependency declaration (unless you use *Dojo*).
- A lot <script src=”/xxx/t.js”> lines in HTML. Slower loading than one big file.

Why not stick with Javascript?

Large Javascript *applications* tend to become **unmaintainable** over time.

Why not stick with Javascript?

Coding is not fun!

Am I totally wrong?

Part II

How is it implemented?

Primary Design Goal of RubyJS

Generate "***efficient***" Javascript code
without sacrifying Ruby's beauty!!!

Ruby's Beauty #1

Meta-programming &
“`eval`”

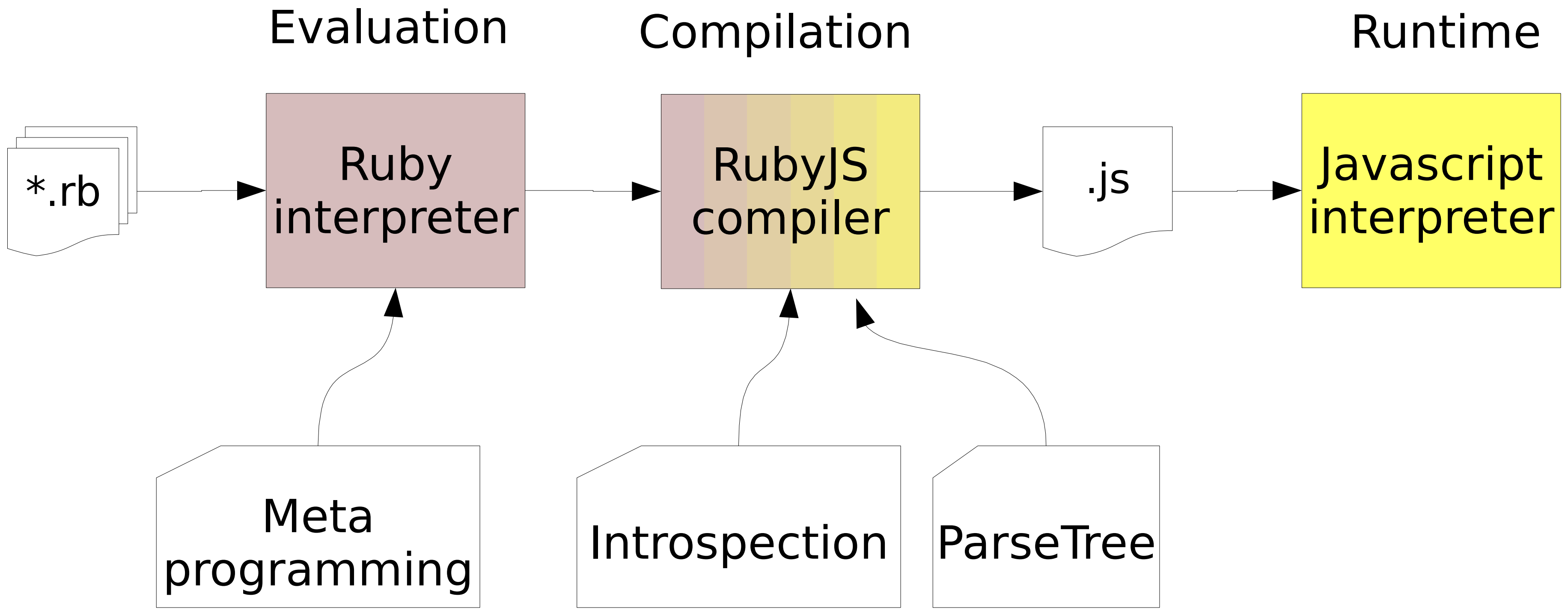
Problem #1

`eval` doesn't work efficiently in browser!

Solution #1

Use `eval` before compilation.

The Translation Process



Implications: Static Ruby

- no eval at runtime
- method/classes fixed at compile-time
  - allows static checks
  - allows efficient method_missing

Ruby's Beauty #2

Everything is an object

Problem #2

Not in Javascript: `null`

Solution #2.1

Map `nil` to `null`

Dispatch function

Dispatch Function

```
a = nil
a.nil?
a.to_s
```

```
a = null;
rjs_send(a, 'nil?');
rjs_send(a, 'to_s');
```

```
function rjs_send(recv, meth, args)
{
  if (typeof(recv)=='null')
  ...
  else if ...
}
```

Dispatch Function Implications

Slooooooooow!

Solution #2.2

Do not map `nil` to `null`

direct method calls

Direct method calls

```
a = nil
a.nil?
a.to_s
```

```
a = nil;
a.nil$q();
a.to_s();
```

```
nil = new NilClass();
```

Implications: Uninitialized ivars

- Instance variables are implemented as attributes.
  - `null` before usage
  - We expect `nil`
  - Solution: Initialize to `nil` before usage.

```
def inc_a
  @a = @a + 1
  return @a
end
```

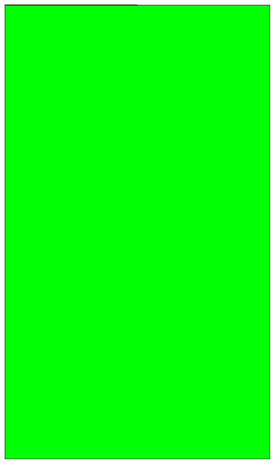
```
function a() {
  this.a = (this.a==null) ?
                 nil : this.a;

  this.a = this.a + 1;
  return this.a;
}
```

Design Decision #2

- Map `nil` to `null`
- Dispatch function
- Sloooooow
- Seamless interoperability

- Map `nil` to “nil”
- Direct method calls
- Fast
- Convert `nil` to `null`



Ruby's Beauty #3

Implicit “`return`”

Problem #3

Explicit `return` in Javascript

Solution #3

Convert last expression of last block into "return".

```
def fact(n)
  if n < 2
    1
  else
    n * fact(n-1)
  end
end
```

```
function fact(n) {
  var _res = nil;
  if (n<2)
    _res = 1;
  else
    _res = n * fact(n-1);
  return _res; }
```

Ruby's Beauty #4

`false` <=> `nil` or `false`

Problem #4

Javascript:

`false <=> 0` or `””` or `[]` or `null` or `false`

Solution #4

Convert into 2-ary expression

possible
side-effects!

```
if condition
```

```
if ((t=condition, t!==nil && t!==false))
```

Ruby's Beauty #5

Everything is an expression

Problem #5

Javascript: Statements != expressions

Solution #5

Translate into expressions where needed

Attempt 1a (#5)

Syntax error

```
result =
   if condition
      expr1
      expr2
   else
      expr3
   end
```

```
result =
   if (condition) {
      expr1;
      expr2;
   }
   else expr3;
```

Attempt 1b (#5)

```
result =
  if condition
    expr1
    expr2
  else
    expr3
  end
```

```
result = condition ?
  (expr1, expr2) :
  (expr3);
```

Attempt 2a (#5)

```
result =
   if condition
      return
   else
      2
   end
```

Syntax error

```
result = condition ?
   (return) :
   (2);
```

Attempt 2b (#5)

```
result =
  if condition
    return
  else
    2
  end
```

```
result = condition ?
  (rjs_return()) :
  (2);
```

```
function rjs_return() {
  throw new Return();
}
```

Translating Statements into Expression

- **if** -> cond **?** exp1 **:** exp2
- **while** -> very uncommon (solution: wrap in function).
- **return** -> use throw.
- **try/catch/finally** -> very uncommon
- **throw** -> call “rjs_throw”

Ruby's Beauty #6

```
def m(a, b=3, c=5, *args, &block)
```

Problem #6

Javascript: `function(a, b)`

Solution #6.1

Choose fixed-arity call convention

Fixed-arity call convention

```
def m(a, *args, &block)
end

m(1,2,3)

m(1) { ... }
```

```
function m(args, block)
{
}

m([1,2,3]);

m([1], function() {});
```

Solution #6.2

Choose variadic call convention

Variadic Call convention

```
def f()
```

```
function f()
```

```
def f(&block)
```

```
function f(block)
```

```
def f(x, y)
```

```
function f(_, x, y)
```

```
def f(x, &block)
```

```
function f(block, x)
```

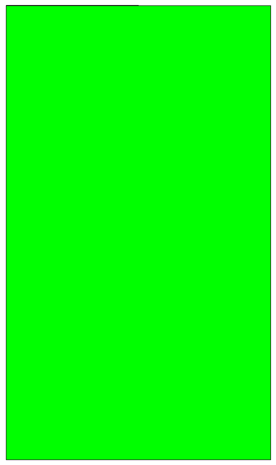
```
def f(x, y=5, *args, &block)
```

```
function f(block, x, y) {
  var args = [];
  if (y===undefined) y=5;
  for (var i=2; i < arguments.length; i++)
    args.push(arguments[i]);
```

Decision #6

- Fixed arity
- Array
- f([1,2,3])
- suffix block argument
- 2 x slower

- Variadic
- arguments
- f(nil, 1,2,3)
- prefix block argument



Ruby's Beauty #7

General purpose Hashes

Problem #7

Javascript:
Only strings as keys.

Only strings as keys

```
a = {}

a[1] = 4
a[“1”] = 5

a[1] != 4
```

Solution #7.1

Map Hash in Ruby to Object in Javascript

Solution #7.2

Implement our own Hash class

Decision #7

- Map Hash (Ruby) to Object (JS)
- No memory or performance overhead
- Strings as keys
- Everything inherits from Object

- Implement custom Hash class in JS
- Slower, more memory
- General purpose

Do not sacrify Ruby's beauty!

Ruby's ~~Beauty~~ #8

Strings are mutable

Problem #8

Javascript Strings are immutable

Solution #8.1

Wrap Strings inside custom object

Solution #8.2

Map to immutable Strings

Decision #8

- Wrap
- Mutable Strings
- Huge Memory overhead

- Map
- Immutable Strings
- No memory overhead
- Interoperability
- No ! methods

Efficiency!!!

Ruby's Beauty #9

method_missing

Problem #9

Not in Javascript

Solution #9.1

Slooooow

Use Dispatch Function as in #2.1

Solution #9.2

Assign method stubs.

We know all called methods!

Ruby's Beauty #0

Ruby is complex!

Ruby is complex

- Meta-Classes. Brain still hurts ;-)
- Multi assignment
  - `a, b, *c = 1, *[1,2,3]`
- Different behaviour of yield/block.call
- `m { return }` vs. `m { break }`

Solution #0

Just do it ;-)

Part III

Applications

Ruby Web Toolkit

- Ongoing effort to port Google Web Toolkit (GWT)
  - Core classes ported (DOM, Events, AJAX, UIObject, Label, Widget)
  - 30% done

RubyJS/RWT vs. GWT

- Ruby vs. Java ;-)
- dynamic vs. static typed
- RubyJS generates less performant code
- GWT is probably much more mature!
- RubyJS: Meta-programming!!!
- Share code with server-side Ruby code

Browser-specific code

- GWT:
  - DOM.java (external interface)
  - DOMImpl.java (internal impl. interface)
  - DOMImplStandard.java (default impl.)
  - DOMImplOpera.java (specific impl.)
- RWT:
  - DOM.rb (default impl), DOM.Opera.rb (overrides)

Appendix

How you can help

- Donate to it! :)
- Use it!
- Extend it!
- Spread it!

Status of RubyJS

- Most of Ruby constructs compile
- Inheritance, Mixins, constants, class methods
- Exception handling
- method_missing, kind_of?, respond_to?
- yield, iterators
- String interpolation
- splat, multiple-assignment

RubyJS TODOs

- Recognize private/protected
- Inline private/protected methods.
- Undef methods
- Complete port of GWT/RWT
- Hashes use Javascript Object

Type mapping

`true, false`	`true, false`
`nil`	`nil (special object)`
`1, 1.2`	`1, 1.2 (Number)`
`“Hello”`	`“Hello” (immutable)`
`/[A-Z]\s+/`	`/[A-Z]\s+/`
`[1,2,3]`	`[1,2,3]`
`{|i| ... }`	`function(i) { ... }`
`{1 => 5, “1” => 4}`	`Custom Hash object`

Optimizations

- `rubyjs_gen --show-options`
  - NoArgumentArityChecks (Speed and Size)
    - `def a(i) ... end; a()`
    - will not give ArgumentError exception!!!!
  - OptimizeArithOps (Speed)
    - use native Javascript +, -, / and * operators instead of methods.
  - NoMethodMissing (Size)
    - 4 Kb code less (uncompressed)
    - small startup-speed improvement

Possible Future Optimizations

- Remove code of uncalled methods
  - Mutual exclusive with `method_missing`
- Inline private/protected methods.
  - `inline :this_method`

Inline Javascript

- Inside backticks
- RubyJS::inline “str”
- #<…> name munging
  - #<local_var>
  - #<Constant>
  - #<m:method>
  - #<@ivar>

this in nested functions

```ruby
def x
  @i = 0
  3.times { @i += 1 }
end
```

```javascript
function x() {
  var self = this;
  self.i = 0;
  (3).times(function() { self.i += 1 });
}
```

False Trueness

```
0 is false
1 is true

[] is false
“” is false
{} is true  // !!
```