Understanding the Dynamics of JavaScript

Sylvain Lebresne, Gregor Richards, Johan Östlund, Tobias Wrigstad, Jan Vitek

Purdue University

July 6, 2009

Introduction

JavaScript

Measurements

Results

Conclusions

Introduction

Introduction — The Significance of JavaScript[1]

- Language of "Web 2.0"
- Dynamic language used for large, structured web programs
- Supplanting Java applets, Flash

[1] JavaScript is also known as ECMAScript, JScript

Introduction — Motivation

- Understand real-world patterns used in dynamic languages
- Do dynamic languages beget untypable code?
- Potential for type analysis of JavaScript
- What patterns in JavaScript could be recreated in a static context

Introduction — JavaScript and Types

- Extremely dynamic, flexible object system
- No static notion of type
- But is the dynamicity used?

JavaScript

JavaScript — The Language

- Imperative, object-oriented
- Minimalistic standard library
- 3rd-party libraries abstract the type system (Prototype.js, jQuery, Ext)

JavaScript — Prototypes

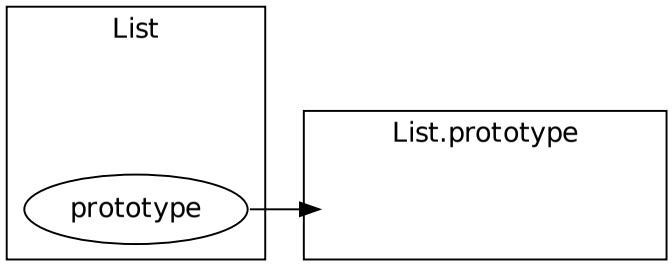
- Objects have a prototype, which is another object
- Field lookup looks in the object itself, then its prototype
- Prototype chains act like subtype relationships

JavaScript — Terminology

- Constructors have a `prototype` field, the prototype of objects created by the constructor: `X.prototype` is **not** the prototype of `X`, but the prototype of objects created by `X`
- The prototype of an object is accessible in many implementations by the field `__proto__`

JavaScript — Prototypes Example

```
function List(v, n) { this.v = v; this.n = n; }
```




```
    return new List(f(this.v),
                    this.n ? this.n.map(f) : null);
}
```


Questions

- How often do prototypes change after first instantiation?
- How often do prototype chains change after first instantiation?
- How often are entirely new fields or methods added to live objects?
- What is the object-to-prototype ratio?
- How complex/deep are prototype hierarchies?
- Do JavaScript programs make use of type introspection?
- What is the ratio of message sends and field updates?

List

Measurements

Measurements — Dirtiness

- Primary measurement is "dirtiness" of objects
- Dirtying actions:
  - Addition or deletion of a property
  - Update of a method
  - Update of the `prototype` field
- Intuition:
  - "Clean" objects are nearly statically typable
  - "Dirty" objects use dynamic features
- Update of a field explicitly ignored

Measurements — Test Cases

- SunSpider tests
  - Popular for benchmarking JavaScript implementations
  - "(...) avoids microbenchmarks, and tries to focus on the kinds of actual problems developers solve with JavaScript today, (...)"
- Real web pages
  - Amazon, Basecamp, Facebook, Gmail, LivelyKernel, NASA
  - Random walk (normal web surfing activity)

Results — Objects

- Results broken down by objects, in these categories:
  - *Regular objects*: objects created by `new` (and array literals.)
  - *Constructors*: functions used to create regular objects.
  - *Functions*: functions that are not used as a constructor.
  - *Prototypes*: objects created as prototypes of functions.

Results — Object Dirtiness

	Regular Objects	Prototypes	Constructors	Other Functions	Objects
3d-cube	20491 (0.02%)	23 (0%)	1 (0%)	14 (0%)	20529 (0.02%)
3d-raytrace	10742 (0.61%)	36 (0%)	3 (0%)	25 (0%)	10806 (0.61%)
binary-trees	42072 (0%)	11 (0%)	1 (0%)	2 (0%)	42086 (0%)
v8-crypto	1076 (44%)	140 (2.86%)	5 (20%)	127 (0%)	1348 (35.5%)
v8-deltablue	22854 (0%)	87 (1.15%)	15 (73.33%)	64 (3.12%)	23020 (0.07%)
v8-raytrace	399685 (0.02%)	361 (1.94%)	18 (100%)	335 (5.37%)	400399 (0.03%)
v8-richards	3000 (0%)	46 (0%)	7 (0%)	31 (0%)	3084 (0%)
amazon	31589 (21.6%)	20909 (0%)	1 (100%)	20750 (4.9%)	73249 (10.7%)
basecamp	4381 (5.2%)	1796 (0.4%)	0 (0%)	1663 (0%)	7840 (3%)
facebook	72231 (28.5%)	22692 (1.4%)	167 (69.5%)	22407 (13.1%)	117497 (20.4%)
gmail	55833 (24%)	11140 (0.06%)	1899 (98.8%)	8338 (40.4%)	77210 (24.1%)
livelykernel	316717 (7.1%)	70036 (0%)	1 (0%)	69845 (0.01%)	456599 (4.9%)
nasa	93989 (3.1%)	13853 (2.7%)	24 (4.2%)	13352 (0.06%)	121218 (2.7%)
random	47921 (16.4%)	12680 (0.15%)	189 (55%)	12338 (0.4%)	73128 (11%)

- Regular object dirtiness usually due to "optional" fields or object literals
  - *v8-crypto*: bignums constructor does not always create some fields; created instead by later functions such as `fromInt`
  - *v8-raytrace*: optional `shader` function added to some (but not all) objects

Results — Object Dirtiness

	Regular Objects	Prototypes	Constructors	Other Functions	Objects
3d-cube	20491 (0.02%)	23 (0%)	1 (0%)	14 (0%)	20529 (0.02%)
3d-raytrace	10742 (0.61%)	36 (0%)	3 (0%)	25 (0%)	10806 (0.61%)
binary-trees	42072 (0%)	11 (0%)	1 (0%)	2 (0%)	42086 (0%)
v8-crypto	1076 (44%)	140 (2.86%)	5 (20%)	127 (0%)	1348 (35.5%)
v8-deltablue	22854 (0%)	87 (1.15%)	15 (73.33%)	64 (3.12%)	23020 (0.07%)
v8-raytrace	399685 (0.02%)	361 (1.94%)	18 (100%)	335 (5.37%)	400399 (0.03%)
v8-richards	3000 (0%)	46 (0%)	7 (0%)	31 (0%)	3084 (0%)
amazon	31589 (21.6%)	20909 (0%)	1 (100%)	20750 (4.9%)	73249 (10.7%)
basecamp	4381 (5.2%)	1796 (0.4%)	0 (0%)	1663 (0%)	7840 (3%)
facebook	72231 (28.5%)	22692 (1.4%)	167 (69.5%)	22407 (13.1%)	117497 (20.4%)
gmail	55833 (24%)	11140 (0.06%)	1899 (98.8%)	8338 (40.4%)	77210 (24.1%)
livelykernel	316717 (7.1%)	70036 (0%)	1 (0%)	69845 (0.01%)	456599 (4.9%)
nasa	93989 (3.1%)	13853 (2.7%)	24 (4.2%)	13352 (0.06%)	121218 (2.7%)
random	47921 (16.4%)	12680 (0.15%)	189 (55%)	12338 (0.4%)	73128 (11%)

- Prototypes are dirty if modified *after* the first instance is created
  - Adding fields to `Object.prototype` and `String.prototype`
  - *v8-crypto*: "static" fields `zero` and `one`

Results — Object Dirtiness

```
function BigNum(val) { ... }
BigNum.ZERO = new BigNum(0);
BigNum.ONE = new BigNum(1);
BigNum.prototype.add = function(to) { ... }
```

```
function BigNum(val) { ... }
BigNum.prototype.add = function(to) { ... }
BigNum.ZERO = new BigNum(0);
BigNum.ONE = new BigNum(1);
```


Results — Object Dirtiness

	Regular Objects	Prototypes	Constructors	Other Functions	Objects
3d-cube	20491 (0.02%)	23 (0%)	1 (0%)	14 (0%)	20529 (0.02%)
3d-raytrace	10742 (0.61%)	36 (0%)	3 (0%)	25 (0%)	10806 (0.61%)
binary-trees	42072 (0%)	11 (0%)	1 (0%)	2 (0%)	42086 (0%)
v8-crypto	1076 (44%)	140 (2.86%)	5 (20%)	127 (0%)	1348 (35.5%)
v8-deltablue	22854 (0%)	87 (1.15%)	15 (73.33%)	64 (3.12%)	23020 (0.07%)
v8-raytrace	399685 (0.02%)	361 (1.94%)	18 (100%)	335 (5.37%)	400399 (0.03%)
v8-richards	3000 (0%)	46 (0%)	7 (0%)	31 (0%)	3084 (0%)
amazon	31589 (21.6%)	20909 (0%)	1 (100%)	20750 (4.9%)	73249 (10.7%)
basecamp	4381 (5.2%)	1796 (0.4%)	0 (0%)	1663 (0%)	7840 (3%)
facebook	72231 (28.5%)	22692 (1.4%)	167 (69.5%)	22407 (13.1%)	117497 (20.4%)
gmail	55833 (24%)	11140 (0.06%)	1899 (98.8%)	8338 (40.4%)	77210 (24.1%)
livelykernel	316717 (7.1%)	70036 (0%)	1 (0%)	69845 (0.01%)	456599 (4.9%)
nasa	93989 (3.1%)	13853 (2.7%)	24 (4.2%)	13352 (0.06%)	121218 (2.7%)
random	47921 (16.4%)	12680 (0.15%)	189 (55%)	12338 (0.4%)	73128 (11%)

- Emulation of class-based behavior common, further investigation needed

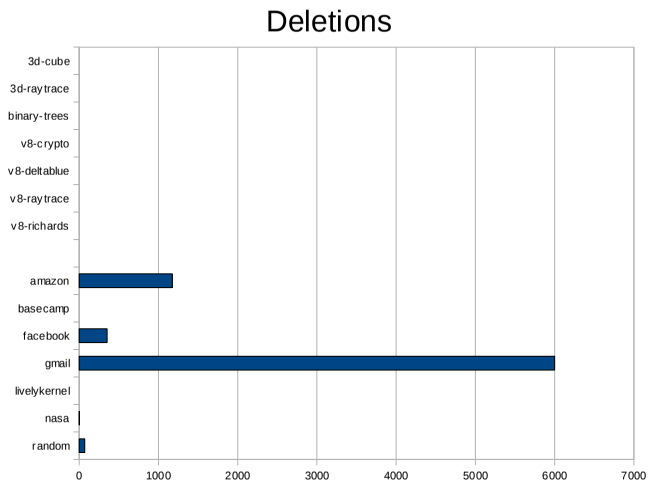
Results — Constructors and Functions

```
Object.prototype.inherits = function(shuper) {
    function Inheriter() {}
    Inheriter.prototype = shuper.prototype;
    this.prototype = new Inheriter();
    this.superConstructor = shuper;
}
function List(...) = { ... }
function ColorList(...) = {
    ColorList.superConstructor.call(this, ...);
    ...
}
ColorList.inherits(List);
```

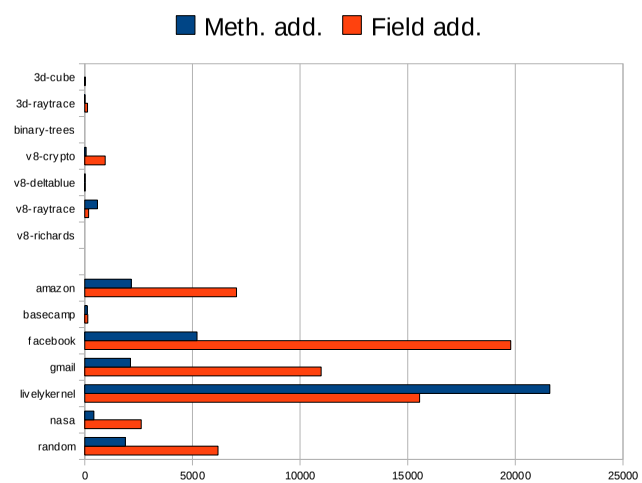

Results — Dirtiness Sources

- Results broken down by source of dirtiness
  - Method addition
  - Method update
  - Field addition
  - Prototype update
  - Deletion

Results — Dirtiness Sources



Results — Dirtiness Sources



Results — Dirtiness Sources

	Meth. add. Obj./Tot.	Meth. upd. Obj./Tot.	Field add. Obj./Tot.	Proto. upd. Obj./Tot.	Deletions Obj./Tot.	Avg. (Med.)
3d-cube	0/0	0/0	16/4	0/0	0/0	4.0 (2)
3d-raytrace	2/2	0/0	124/64	0/0	0/0	1.9 (2)
binary-trees	0/0	0/0	0/0	0/0	0/0	0.0 (0)
v8-crypto	61/4	0/0	950/475	0/0	0/0	2.1 (2)
v8-deltablue	11/8	0/0	10/2	12/12	0/0	2.2 (2)
v8-raytrace	587/77	10/5	180/36	33/33	0/0	6.4 (2)
v8-richards	0/0	0/0	0/0	0/0	0/0	0.0 (0)
amazon	2160/4198	39/67	7050/59769	2/2	1174/1896	8.4 (2)
basecamp	112/819	7/7	142/1883	0/0	0/0	11.6 (2)
facebook	5212/16432	256/648	19787/84912	72/72	352/727	4.3 (2)
gmail	2123/4258	68/180	10982/35783	1896/1896	6001/19972	3.3 (2)
livelykernel	21605/42346	0/0	15555/16584	0/0	0/0	2.6 (2)
nasa	421/2045	361/361	2621/6127	7/7	1/3	2.6 (1)
random	1885/4037	24/1563	6188/48988	121/121	69/173	6.8 (2)

- In Gmail, 1896 `prototype` field updates, 1899 updates to constructors; strongly suggests class emulation

Results — Other Results

- Ratio of message sends to field updates
  - The vast majority of programs have a low ratio; used imperatively, not functionally
- Length of prototype chains
  - Max length 10 (gmail)
  - SunSpider's chains were all short (≤ 4)
  - Real programs had greater max length (all ≥ 6)
  - All programs had ≈ 2 average
- Calls to `typeof`
  - Rare in SunSpider, common in real programs

Conclusions and Future Work

- Certain dynamic actions are common in JavaScript
- Many can be avoided by identifying patterns and refactoring
- Potential exists for static type analysis of JavaScript programs

Questions?