Understanding the Dynamics of JavaScript

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

Purdue University

July 6, 2009
Introduction
Introduction — The Significance of JavaScript

- 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)
Objects have a prototype, which is another object
Field lookup looks in the object itself, then its prototype
Prototype chains act like subtype relationships
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; }
List.prototype.map = function(f) {
    return new List(f(this.v),
    this.n ? this.n.map(f) : null);
}
var l = new List(1, null);
delete(List.prototype.map);
```
function List(v, n) { this.v = v; this.n = n; }
function List(v, n) { this.v = v; this.n = n; }
List.prototype.map = function(f) {
    return new List(f(this.v),
                    this.n ? this.n.map(f) : null);
}
function List(v, n) { this.v = v; this.n = n; }
List.prototype.map = function(f) {
    return new List(f(this.v),
        this.n ? this.n.map(f) : null);
}
var l = new List(1, null);
function List(v, n) { this.v = v; this.n = n; }
List.prototype.map = function(f) {
  return new List(f(this.v),
                  this.n ? this.n.map(f) : null);
}
var l = new List(1, null);

```javascript
// Diagram of prototype chain:
List
  --prototype
    --l
      map
        List.prototype
          List.prototype.map
```
function List(v, n) { this.v = v; this.n = n; }
List.prototype.map = function(f) {
    return new List(f(this.v),
        this.n ? this.n.map(f) : null);
}
var l = new List(1, null);
delete(List.prototype.map);
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?
How often do prototypes change after first instantiation?
Questions

- How often do prototypes change after first instantiation?
- How often do prototype chains change after first instantiation?
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?
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?
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?
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?
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?
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
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
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

<table>
<thead>
<tr>
<th></th>
<th>Regular Objects</th>
<th>Prototypes</th>
<th>Constructors</th>
<th>Other Functions</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>3d-cube</td>
<td>20491 (0.02%)</td>
<td>23 (0%)</td>
<td>1 (0%)</td>
<td>14 (0%)</td>
<td>20529 (0.02%)</td>
</tr>
<tr>
<td>3d-raytrace</td>
<td>10742 (0.61%)</td>
<td>36 (0%)</td>
<td>3 (0%)</td>
<td>25 (0%)</td>
<td>10806 (0.61%)</td>
</tr>
<tr>
<td>binary-trees</td>
<td>42072 (0%)</td>
<td>11 (0%)</td>
<td>1 (0%)</td>
<td>2 (0%)</td>
<td>42086 (0%)</td>
</tr>
<tr>
<td>v8-crypto</td>
<td>1076 (44%)</td>
<td>140 (2.86%)</td>
<td>5 (20%)</td>
<td>127 (0%)</td>
<td>1348 (35.5%)</td>
</tr>
<tr>
<td>v8-deltablue</td>
<td>22856 (0%)</td>
<td>87 (1.15%)</td>
<td>19 (73.33%)</td>
<td>64 (3.12%)</td>
<td>23020 (0.07%)</td>
</tr>
<tr>
<td>v8-raytrace</td>
<td>399685 (0.02%)</td>
<td>361 (1.94%)</td>
<td>18 (100%)</td>
<td>335 (5.37%)</td>
<td>400399 (0.03%)</td>
</tr>
<tr>
<td>v8-richards</td>
<td>3000 (0%)</td>
<td>46 (0%)</td>
<td>7 (0%)</td>
<td>31 (0%)</td>
<td>3084 (0%)</td>
</tr>
<tr>
<td>amazon</td>
<td>31589 (21.6%)</td>
<td>20909 (0%)</td>
<td>1 (100%)</td>
<td>20750 (4.9%)</td>
<td>73249 (10.7%)</td>
</tr>
<tr>
<td>basecamp</td>
<td>4381 (5.2%)</td>
<td>1796 (0.4%)</td>
<td>0 (0%)</td>
<td>1663 (0%)</td>
<td>7840 (3%)</td>
</tr>
<tr>
<td>facebook</td>
<td>72231 (28.5%)</td>
<td>22692 (1.4%)</td>
<td>167 (69.5%)</td>
<td>22407 (13.1%)</td>
<td>117497 (20.4%)</td>
</tr>
<tr>
<td>gmail</td>
<td>55833 (24%)</td>
<td>11140 (0.06%)</td>
<td>1899 (98.8%)</td>
<td>8338 (40.4%)</td>
<td>77210 (24.1%)</td>
</tr>
<tr>
<td>livelykernel</td>
<td>316717 (7.1%)</td>
<td>70036 (0%)</td>
<td>1 (0%)</td>
<td>69845 (0.01%)</td>
<td>456599 (4.9%)</td>
</tr>
<tr>
<td>nasa</td>
<td>93989 (3.1%)</td>
<td>13853 (2.7%)</td>
<td>24 (4.2%)</td>
<td>13392 (0.06%)</td>
<td>121218 (2.7%)</td>
</tr>
<tr>
<td>random</td>
<td>47921 (16.4%)</td>
<td>12680 (0.15%)</td>
<td>189 (55%)</td>
<td>12338 (0.4%)</td>
<td>73128 (11%)</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th></th>
<th>Regular Objects</th>
<th>Prototypes</th>
<th>Constructors</th>
<th>Other Functions</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>3d-cube</td>
<td>20491 (0.02%)</td>
<td>23 (0%)</td>
<td>1 (0%)</td>
<td>14 (0%)</td>
<td>20529 (0.02%)</td>
</tr>
<tr>
<td>3d-raytrace</td>
<td>10742 (0.61%)</td>
<td>36 (0%)</td>
<td>3 (0%)</td>
<td>25 (0%)</td>
<td>10806 (0.61%)</td>
</tr>
<tr>
<td>binary-trees</td>
<td>42072 (0%)</td>
<td>11 (0%)</td>
<td>1 (0%)</td>
<td>2 (0%)</td>
<td>42086 (0%)</td>
</tr>
<tr>
<td>v8-crypto</td>
<td>1076 (44%)</td>
<td>140 (2.86%)</td>
<td>5 (20%)</td>
<td>127 (0%)</td>
<td>1348 (35.5%)</td>
</tr>
<tr>
<td>v8-deltablue</td>
<td>22854 (0%)</td>
<td>87 (1.15%)</td>
<td>15 (73.33%)</td>
<td>64 (3.12%)</td>
<td>23020 (0.07%)</td>
</tr>
<tr>
<td>v8-raytrace</td>
<td>399685 (0.02%)</td>
<td>361 (1.94%)</td>
<td>18 (100%)</td>
<td>335 (5.37%)</td>
<td>400399 (0.03%)</td>
</tr>
<tr>
<td>v8-richards</td>
<td>3000 (0%)</td>
<td>46 (0%)</td>
<td>7 (0%)</td>
<td>31 (0%)</td>
<td>3084 (0%)</td>
</tr>
<tr>
<td>amazon</td>
<td>31589 (21.6%)</td>
<td>20999 (0%)</td>
<td>1 (100%)</td>
<td>20750 (4.9%)</td>
<td>73249 (10.7%)</td>
</tr>
<tr>
<td>basecamp</td>
<td>4381 (5.2%)</td>
<td>1796 (0.4%)</td>
<td>0 (0%)</td>
<td>1663 (0%)</td>
<td>7840 (3%)</td>
</tr>
<tr>
<td>facebook</td>
<td>72231 (28.5%)</td>
<td>22692 (1.4%)</td>
<td>167 (69.5%)</td>
<td>22407 (13.1%)</td>
<td>117497 (20.4%)</td>
</tr>
<tr>
<td>gmail</td>
<td>55833 (24%)</td>
<td>11140 (0.06%)</td>
<td>1899 (98.8%)</td>
<td>8336 (40.4%)</td>
<td>77210 (24.1%)</td>
</tr>
<tr>
<td>livelykernel</td>
<td>316717 (7.1%)</td>
<td>70036 (0%)</td>
<td>1 (0%)</td>
<td>69845 (0.01%)</td>
<td>456599 (4.9%)</td>
</tr>
<tr>
<td>nasa</td>
<td>93989 (3.1%)</td>
<td>13853 (2.7%)</td>
<td>24 (4.2%)</td>
<td>13352 (0.06%)</td>
<td>121218 (2.7%)</td>
</tr>
<tr>
<td>random</td>
<td>47921 (16.4%)</td>
<td>12680 (0.15%)</td>
<td>189 (55%)</td>
<td>12338 (0.4%)</td>
<td>73128 (11%)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Regular Objects</th>
<th>Prototypes</th>
<th>Constructors</th>
<th>Other Functions</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>3d-cube</td>
<td>20491 (0.02%)</td>
<td>23 (0%)</td>
<td>1 (0%)</td>
<td>14 (0%)</td>
<td>20529 (0.02%)</td>
</tr>
<tr>
<td>3d-raytrace</td>
<td>10742 (0.61%)</td>
<td>36 (0%)</td>
<td>3 (0%)</td>
<td>25 (0%)</td>
<td>10806 (0.61%)</td>
</tr>
<tr>
<td>binary-trees</td>
<td>42072 (0%)</td>
<td>11 (0%)</td>
<td>1 (0%)</td>
<td>2 (0%)</td>
<td>42086 (0%)</td>
</tr>
<tr>
<td>v8-crypto</td>
<td>1076 (44%)</td>
<td>140 (2.86%)</td>
<td>5 (20%)</td>
<td>127 (0%)</td>
<td>1346 (35.5%)</td>
</tr>
<tr>
<td>v8-deltablue</td>
<td>22854 (0%)</td>
<td>87 (1.15%)</td>
<td>15 (73.33%)</td>
<td>84 (3.12%)</td>
<td>23020 (0.07%)</td>
</tr>
<tr>
<td>v8-raytrace</td>
<td>399685 (0.02%)</td>
<td>361 (1.94%)</td>
<td>18 (100%)</td>
<td>335 (5.37%)</td>
<td>400399 (0.03%)</td>
</tr>
<tr>
<td>v8-richards</td>
<td>3000 (0%)</td>
<td>46 (0%)</td>
<td>7 (0%)</td>
<td>31 (0%)</td>
<td>3084 (0%)</td>
</tr>
<tr>
<td>amazon</td>
<td>31589 (21.6%)</td>
<td>20909 (0%)</td>
<td>1 (100%)</td>
<td>20750 (4.9%)</td>
<td>73249 (10.7%)</td>
</tr>
<tr>
<td>basecamp</td>
<td>4381 (5.2%)</td>
<td>1796 (0.4%)</td>
<td>0 (0%)</td>
<td>1663 (0%)</td>
<td>7840 (3%)</td>
</tr>
<tr>
<td>facebook</td>
<td>72231 (28.5%)</td>
<td>22692 (1.4%)</td>
<td>167 (69.5%)</td>
<td>22407 (13.1%)</td>
<td>117497 (20.4%)</td>
</tr>
<tr>
<td>gmail</td>
<td>55833 (24%)</td>
<td>11141 (0.06%)</td>
<td>1899 (98.8%)</td>
<td>8338 (40.4%)</td>
<td>77210 (24.1%)</td>
</tr>
<tr>
<td>livelykernel</td>
<td>316717 (7.1%)</td>
<td>70036 (0%)</td>
<td>1 (0%)</td>
<td>69845 (0.01%)</td>
<td>456599 (4.9%)</td>
</tr>
<tr>
<td>nasa</td>
<td>93989 (3.1%)</td>
<td>13855 (2.7%)</td>
<td>24 (4.2%)</td>
<td>13552 (0.06%)</td>
<td>121216 (2.7%)</td>
</tr>
<tr>
<td>random</td>
<td>47921 (16.4%)</td>
<td>12680 (0.15%)</td>
<td>189 (55%)</td>
<td>12338 (0.4%)</td>
<td>73128 (11%)</td>
</tr>
</tbody>
</table>

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

Deletions

- 3d-cube
- 3d-raytrace
- binary-trees
- v8-crypto
- v8-deltablue
- v8-raytrace
- v8-richards

- amazon
- basecamp
- facebook
- gmail
- livelykernel
- nasa
- random

Total deletions: 23
Results — Dirtiness Sources

- Meth. add.
- Field add.

- 3d-cube
- 3d-raytrace
- binary-trees
- v8-crypto
- v8-deltablue
- v8-raytrace
- v8-richards
- amazon
- basecamp
- facebook
- gmail
- livelykernel
- nasa
- random

0 5000 10000 15000 20000 25000
## Results — Dirtiness Sources

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

- 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?