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 — The Significance of JavaScript

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

1JavaScript 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 — 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

```javascript
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);
}
```

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 — 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</td>
<td>18</td>
<td>11</td>
<td>20750</td>
</tr>
<tr>
<td>binary-trees</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v8-crypto</td>
<td>1076 (44%)</td>
<td>42086</td>
<td>399685</td>
<td>20491</td>
<td>42086</td>
</tr>
<tr>
<td>v8-raytrace</td>
<td>399685 (0.02%)</td>
<td>20491</td>
<td>20750</td>
<td>11140</td>
<td>20491</td>
</tr>
<tr>
<td>v8-richards</td>
<td>18</td>
<td>11140</td>
<td>20491</td>
<td>11140</td>
<td>20491</td>
</tr>
<tr>
<td>amazon</td>
<td>31589 (21.6%)</td>
<td>316717</td>
<td>335 (5.37%)</td>
<td>316717</td>
<td>335</td>
</tr>
<tr>
<td>facebook</td>
<td>72231 (38.5%)</td>
<td>22407</td>
<td>140 (2.86%)</td>
<td>22407</td>
<td>140</td>
</tr>
<tr>
<td>gmail</td>
<td>55833 (24%)</td>
<td>1899</td>
<td>10806</td>
<td>1899</td>
<td>1899</td>
</tr>
<tr>
<td>livelykernel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>random</td>
<td>121218 (2.7%)</td>
<td>73128</td>
<td>7840</td>
<td>73128</td>
<td>7840</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

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);

- 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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Meth. add</th>
<th>Meth. upd</th>
<th>Field add</th>
<th>Proto. upd</th>
<th>Deletions</th>
<th>Avg (Obj./Tot.)</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?