JavaScript: The Used Parts

Sharath Chowdary Gude
Auburn University
szg0033@tigermail.auburn.edu

Categories and Subject Descriptors D.2.8 [*Software Engineering*]: Metrics; D.3.3 [*Programming features*]: Language constructs and features

General Terms Languages, Measurement

Keywords JavaScript, Empirical Study, Variable Scope

Abstract

1. Introduction

JavaScript is a language that is evolving. Some features proposed for JS arguably made it more complex and had to be abandoned (features in ECMAScript 4). Other efforts to evolve the language defined a subset of language that provides rigorous error checking and avoid error-prone constructs (ECMAScript 5). However, most of these attempts to include or exclude JS features were done without a scientific study of how the language features are used by developers.

Advances in JavaScript have not been backed by large-scale empirical studies on how people use JS features and how people react to the changes in JavaScript standards. Without this information, it is hard to determine how to evolve the language, which IDE features and refactorings to develop, how to optimize programs, how to teach developers good programming practices, etc.

We perform an empirical study on a huge corpus of JS programs. The test corpus consists of more than a million scripts from well-known and not-so-well-known webpages, and from Firefox Addons. We studied various JS features. This paper describes two JS features: the way JS variables are declared in block scope and function scope, and the way `for..in` statements are used in programs.

2. Research design

JavaScript is the most widely wide used language in the Internet and its in our best interest to collect JS programs coded by wide assorted set of programmers. Programs in the test corpus were collected from 3 sources.

(1) To collect JS programs from the wild, we used a web spider to crawl through the Internet and generate a unique list of 78644 URLs. These URLs reflect webpages from relatively remote websites. We loaded the URLs on an instrumented Google Chromium browser [2], that extracted the JavaScript programs from the URLs. In total, 1,064,793 separate JS programs were extracted.

(2) To collect JS programs that probably have better coding standards, we extracted JS programs from the top 100 websites (as per Alexa [1]). We visited and browsed these websites meaningfully to extract ubiquitously used JS functions in the web.

(3) The Mozilla Firefox Addons constitute the third source of corpus which include JS programs with most recent improvements in the languages. The top 40 Firefox Addons based on popularity are chosen for our research.

The test programs are then analyzed using an instrumented V8 engine. V8 is Google's open source JavaScript engine. It is written in C++ and is used in Google Chrome. We implemented visitors [4] that visited abstract syntax trees generated from JS programs. When V8 loads a JS program, the corresponding visitors were invoked and they stored the results in files. The individual results were aggregated later.

3. Results

The poster will describe several empirical study results. This document describes the results regarding two JS features: how variables are declared in JS, and how `for..in` statements are used to enumerate through an object's properties.

3.1 Variable Declaration in JS

We studied the scope of variable declarations inside functions. JavaScript does not have block scopes: Anything declared within a code block inside a function applies applies throughout the function. Crockford [3] considers this a bad feature of JS, and suggests that variables declared in block scope should be hoisted to the top of a function, so that they are in function scope. But do people follow the advice of experts? Are variables typically used outside their declaration scopes? Our study not only demonstrates the way people de-

[Copyright notice will appear here once 'preprint' option is removed.]

clare variables, but also justifies the need for a construct (let construct proposed in ES 6) for variables with local scope.

Of the 1,064,793 scripts in the corpus from spidered websites, 407,123 have a variable declaration. In total, 1,271,664 variables were declared: 1,241,344 were declared in functional scope, while the rest, i.e., 40,320, were in block scope. Of the 1,24,1344 variables in function scope, 787,502 were used in the function scope, 117,161 were declared within a function scope but used only within a block scope, and 346,781 were declared within a function scope but used both within block scope and function scope. Among the variables declared in block scope, 40,028 were limited within the block scope, but 282 escaped the block scope and were used within the function scope.

The scripts from Alexa top 100 websites shared similar pattern. 63,671 variable declarations were identified with declarations—61,236 in function scope and 2,345 in block scope. Of the function-scoped variables, 4,329 were used only within block scope, 15,564 were used within the function scope and 41,433 in both scopes. Among the block-scoped variables, only 15 escaped the scope.

Mozilla supports the `let` construct to declare block-scoped variables. 7 out of 40 Mozilla Addons from our corpus uses `let`—there were 1,800 variable declarations with `let`. As for regular variable declarations, there were 12,918—4,279 in block scope and 8,639 in function scope. Interestingly, all block-scoped variables were used within the scope. Among function-scoped variables, 1,034 were used only in block scope, 435 only in function scope, and the rest in both scopes.

Discussion A significant portion of the variables are declared in function scope and used in function scope. The most important observation is that the use of variables declared in block scope is limited to block scope. This implies that web programmers do follow Crockfordian ideas [3] about variable scope.

3.2 Property Enumeration with `for..in`

`for..in` is used for iterating through the properties of generic objects. `for..in` loops will not only enumerate every non-shadowed, user-defined properties, but also will enumerate the properties inherited from objects in the prototype chain. The `hasOwnProperty` construct can be used to distinguish an object's own and inherited properties. Our research focuses on the usage of `for..in` construct and whether they are used with `hasOwnProperty`.

There were 1,064,793 scripts extracted from the spidered websites. Among them 16725 instances of `for..in` were identified. Surprisingly, only 296 of them were refined with `hasOwnProperty` construct. The second source of repository are the scripts from Alexa top 100 sites which totaled 70,377 scripts. 777 of these functions contained `for..in` property and in those 17 instances were filtered with `hasOwnProperty`. The Firefox addons followed a similar pattern with 436 `for..in` functions and only 29 featuring `hasOwnproperty`.

Discussion The results indicate that the `hasOwnProperty` is seldom used with `for..in`. The above results evoked conjectures that the objects are stand-alone with no parents; it was verified to be true. Since most objects are created with null parents, it is not necessary to use `hasOwnProperty`.

Another interesting conclusion that can be drawn is that inheritance does not seem to be favored by web programmers with most of the objects being stand-alone. However, the addons have a higher percentage of `for..in`-s with `hasOwnProperty` because of the lengthier functions which often necessitate the need for inheritance.

4. Related Work

Empirical data on real-world usage of language features are available for popular languages, e.g., a study about preprocessor usage in C programsbut there has only been a few studies on JavaScript. JavaScript studies have focused on various dynamic features [6] [7], or the memory behavior [5], or security problems in JavaScript programs [8]. However, these studies do not target the language itself and how its features are used by developers. The test corpus that they use do not represent JS usage in the wild. Such studies are important for an evolving language to distinguish the used parts and focus on their evolution.

Acknowledgement

Allen Wirfs-Brock and Brendan Eich for their guidance.

References

[1] `alexa.com/Topsites`.

[2] `chromium.org`.

[3] D. Crockford. *JavaScript: the good parts*. O'Reilly Media, Inc., 2008.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1995.

[5] P. Ratanaworabhan, B. Livshits, and B. G. Zorn. Jsmeter: comparing the behavior of javascript benchmarks with real web applications. In *Proceedings of the 2010 USENIX conference on Web application development*, WebApps'10, Berkeley, CA, USA, 2010. USENIX Association.

[6] G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dynamic behavior of javascript programs. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010*, pages 1–12. ACM, 2010.

[7] G. Richards, C. Hammer, B. Burg, and J. Vitek. The eval that men do - A large-scale study of the use of eval in javascript applications. In *ECOOP 2011 - Object-Oriented Programming - 25th European Conference, Lancaster, UK, July 25-29, 2011 Proceedings*, volume 6813, pages 52–78. Springer, 2011.

[8] C. Yue and H. Wang. Characterizing insecure javascript practices on the web. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 961–970. ACM, 2009.