A Lightweight JavaScript Engine for Mobile Devices

Ryan H. Choi * Youngil Choi
Software R&D Center
Samsung Electronics, Republic of Korea
{ryan.choi,duddlf.choi}@samsung.com

Abstract

We present Typed JS, a subset of JavaScript that supports AOT compilation by utilizing type-decorated syntax. Typed JS is designed for mobile devices with goals of having smaller memory footprint while achieving high-performance, which is accomplished by having static types and AOT compilable architecture. Experiments show that Typed JS requires significantly much less memory usage while performing better than industry-leading JavaScript engines on a mobile platform.

Categories and Subject Descriptors D.3.2 [*Programming Languages*]: Language Classifications—Very high-level languages; D.3.4 [*Programming Languages*]: Processors—Code generation, Compilers

Keywords JavaScript, Static Typing, Mobile

1. Introduction

In Web application framework, JavaScript brings web application interactive by implementing client-side interaction and business logic. However, due to unsatisfactory performance of JavaScript, improving the performance and memory usage of JavaScript has been an active research and engineering problem [1, 3]. One research direction is to propose a variant of JavaScript that restricts current JavaScript's dynamicity by adding static types. Noticeable work include TypeScript[1] and Flow[2], which both extend JavaScript to accept type-decorated syntax. The aim of these work is to utilize many already-integrated optimization techniques in JavaScript engines by providing extra type information. Ahead-of-time (AOT) compilation can generally optimize better for performance, but it is not suitable for JavaScript due to its untyped dynamicity design. asm.js[3] attempts to integrate AOT compilation into JavaScript by translating C++ code into non-dynamic JavaScript code that runs faster. However, the use of asm.js is limited, as it cannot support JavaScript core design such as objects, prototype, etc.

In this paper, we propose Typed JavaScript (Typed JS), a subset of JavaScript that supports AOT compilation by utilizing type-decorated syntax. Unlike Google's V8 and Apple's JSC, Typed JS is designed to reduce runtime memory footprint and binary size when run on mobile devices. Unlike asm.js, Typed JS supports most of JavaScript core design such as object model, prototype, functions and closures, and garbage collection. By utilizing type-decoration and fixed object layout, object lookups are removed, which results in achieving high performance. Also by supporting AOT compilation, JavaScript VM is replaced by much compact native runtime library, which significantly reduced memory footprint. Typed JS provides additional advantages. Rigorous type checking during compilation allows us to early detect errors caused by type-mismatching. Any applications written in Typed JS runs on any platform, when recompiled, without modifying the source code. Furthermore, only binary files can be distributed, if one wants to prevent from unauthorized code modification, which often is an important requirement in industry. Finally, Typed JS can be used to easily implement mobile applications. A binding API between Typed JS and EFL[4] graphics library on Tizen[5] mobile platform is implemented, and we successfully implemented mobile applications in Typed JS which were originally written in C++. Our recent work [2] describes Typed JS in detail.

Organization: Section 2 presents design principles of Typed JS. Section 3 gives design, model, and implementation details of Typed JS. Section 4 shows experimental results. Lastly, we conclude in Section 5.

2. Design of Typed JS

Typed JS enforces type annotations, allows both dynamic and static features, and supports AOT compilation for performance and smaller memory footprint on mobile platforms. The design principles are as follows.

* Corresponding author

[1] http://www.typescriptlang.org

[2] https://flowtype.org

[3] http://asmjs.org

[4] https://www.enlightenment.org

[5] https://developer.tizen.org

MobileDeLi'15, October 26, 2015, Pittsburgh, PA, USA
© 2015 ACM. 978-1-4503-3906-3/15/10...$15.00
http://dx.doi.org/10.1145/2846661.2846662

```
var hanoi =
  function(disc: int,
           src: string,
           aux: string,
           dst: string):
           void {
    if(disc > 0) {
      hanoi(disc-1, src,
            dst, aux);
      console.log(
        "Move disc " +
         disc +
        " from " + src +
        " to " + dst);
      hanoi(disc-1, aux,
            src, dst);
    }
  }
hanoi(5, "src",
      "aux", "dst");
```

Figure 1. Tower of Hanoi in Typed JS

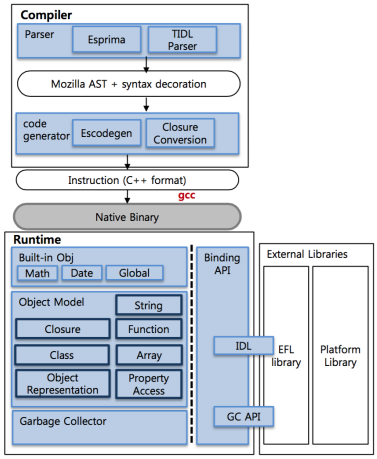


Figure 2. Architecture of Typed JS

Type Annotation: Type checking for dynamic objects in runtime is one of the major performance bottlenecks in JavaScript. Typed JS extends JavaScript such that, types of objects must be supplied when objects are declared. Figure 1 shows an example of a function written in Typed JS.

Object Model: Two object models, dynamic object and sealed class models, cohesively exist. Dynamic object model is the prototypical model found in JavaScript, while sealed class model is the class-oriented model found in C++. Former is to be compatible with typical JavaScript, while latter is designed to give better performance. The differences between these two models are that, dynamically adding/removing properties is removed in the sealed class model, and all types are finalized in the compilation time.

AOT Compilation: Typed JS is compiled to a target-specific, optimized binary executable. Moreover, Typed JS utilizes modern compiler optimization techniques, as it annotates types and supports static classes. In our prototype, Typed JS compiler transpiles Typed JS source code into C++11, and it is natively compiled and optimized by g++.

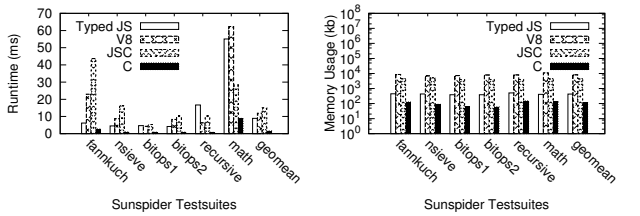
Robust and Secure: Typed JS follows the strict mode of JavaScript, and redefines a set of dynamic features that can be efficiently implemented. Also, Typed JS does not support evaluating source code during runtime, i.e., $eval()$, eliminating security holes.

3. Architecture of Typed JS

Figure 2 shows the architecture of Typed JS. It consists of compiler and runtime parts. The compiler part takes Typed JS source code as input, and generates C++11 code. The runtime part provides the implementation of the internal data structure and runtime library on which Typed JS depends.

Compilation is a 3-phase process. First, the parser generates an abstract syntax tree (AST) from Typed JS source code. The AST follows Mozilla JavaScript AST except it additionally contains type-specific information and Typed JS extensions such as type-annotated objects and sealed classes. Moreover, esprima,[6] a ECMAScript 5.1 parser, is extended

[6] http://esprima.org



(a) Runtime (b) Memory

Figure 3. Tizen Platform

to parse and validate Typed JS syntax. Furthermore, Tizen binding API written in Tizen IDL (TIDL) is parsed and validated, and added to the AST. Second, the code generator takes the AST as input, performs semantics validation, and generates C++11 code. We modified escodegen,[7] which originally generates JavaScript code from an AST, to generate C++11 code. Furthermore, we modified escodegen to perform type inference by deriving and applying a set of type inference rules during the semantics validation to deduce unknown variable types. Finally, the auto-generated C++ code is compiled and linked to Typed JS runtime library and optionally Tizen library, if required.

4. Experimental Results

We now present experimental results. Experiments were conducted on a preproduction, low-end Samsung Tizen mobile phone. Typed JS is compiled using Tizen SDK 2.3 and g++. Sunspider testsuites[8] were used for performance benchmark. Furthermore, the same testsuites were executed on V8 and JSC, and were also ported to C and executed.

Figure 3 shows the runtime performance and memory usage of Typed JS against V8, JSC, and C on the Tizen mobile phone. Typed JS outperforms V8 and JSC by up to 3.5x while consuming up to 20x less memory.

5. Conclusion

In this paper, we presented Typed JS, a memory efficient but yet high-performance JavaScript engine for mobile devices. By utilizing type-decoration, Typed JS can be compiled ahead-of-time, which results in achieving smaller memory footprint and high-performance. Experiments show that Typed JS is memory-efficient and achieves better performance than industry-leading JavaScript engines on Tizen.

As future work, we plan to update C++11 code generator to generate LLVM IR to give us more performance optimization opportunities.

References

[1] W. Ahn, J. Choi, T. Shull, M. J. Garzarán, and J. Torrellas. Improving javascript performance by deconstructing the type system. In *PLDI*, 2014.

[2] R. H. Choi and Y. Choi. Typed js: A lightweight typed javascript engine for mobile devices. In *MobiCASE*, 2015.

[3] T. Rompf, A. K. Sujeeth, K. J. Brown, H. Lee, H. Chafi, and K. Olukotun. Surgical precision JIT compilers. In *PLDI*, 2014.

[7] https://github.com/estools/escodegen

[8] http://www.webkit.org/perf/sunspider/sunspider.html