Tutorial: Optimizing JavaScript Code for V8

Florian Loitsch
Google Inc.
floitsch@google.com

http://floitsch.blogspot.com/2012/03/optimizing-for-v8-introduction.html

September 2012

The performance of programs running in JavaScript engines is notoriously difficult to predict. Indeed, JavaScript is a complex language and, due to time-constraints and limited engineering resources, all popular virtual machines only optimize a subset of the language. Code that runs outside this (non obvious) sweet spot can pay huge performance penalties.

JavaScript engines generally have at least two modes of operation: one non-optimized, and one optimized. Initially all functions are compiled to run in the non-optimized mode. Heuristics, like statistic profilers or invocation counters, then trigger the expensive recompilation of hot methods. Methods frequently see a performance improvement of an order of magnitude when they run in optimized mode. It is hence crucial that programs spend their time in optimized code.

There are several ways compilers can do this:

- avoid statements that cannot be optimized by the JIT. Indeed, V8 still cannot generate optimized code for all JavaScript constructs.
- avoid bailouts. Optimized code is generated under the assumption that the generated code will run with similar dynamic types as seen before. If that assumption fails, the optimized code must be thrown away.
- make code monomorphic. Optimized code is more efficient if it specializes for fewer dynamic types. Frequently it is possible to reduce the number of types by duplicating functions.

Knowing when and where to apply these tips is almost impossible without proper tool-support. In this tutorial I will discuss the listed optimization techniques and present the tools that allow the investigation of V8 generated

code. In particular, I will focus on V8s tracing flags, which report when methods are (de)optimized or inlined, Hydrogen traces, which represent V8s intermediate representation, and assembly dumps. During the talk I will concentrate on a Scheme-to-JavaScript compilation, but all talking-points will be of interest to any developer creating JavaScript code.