UDC 004.942

I.O. Zolotareva, O.O. Knyga

Simon Kuznets Kharkiv National University of Economics, Kharkiv

MODERN JAVASCRIPT PROJECT OPTIMIZERS

This article is devoted to diffrent methods of minification of single JavaScript files, entire projects, and their concatenation. It analyzes minificators JSMin, YUI Compressor, UglifyJS and compilators Google Closure and Kjscompiler, describes their benefits and presents comparative descriptions.

Keywords: *JavaScript, Kjscompiler, JSMin, YUI Compressor, UglifyJS, Google Closure, TypeScript, ECMAScript, data compression, coding optimization, compilation, ECMAScript, minification, filters.*

Introduction

JavaScript is one of the most popular dynamic computer Netscape-developed object scripting programming language used in millions of web pages and server applications worldwide. Netscape's JavaScript is a superset of the ECMA-262 Edition 3 (ECMAScript) standard scripting language, with only mild differences from the published standard. Virtually every personal computer in the world has at least one JavaScript interpreter installed on it and in active use. JavaScript's popularity is due entirely to its role as the scripting language of the WWW. It allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed [1]. It is also being used in server-side programming, game development and the creation of desktop and mobile applications. This language has strong object-oriented programming capabilities, even though some debates have taken place due to the differences in object-oriented JavaScript compared to other languages.

While often derided as a toy, beneath its deceptive simplicity lie some powerful language features, one that is now used by an incredible number of high-profile applications, showing that deeper knowledge of this technology is an important skill for any web or mobile developer [6]. JavaScript, like any other object oriented programming languages, has types and operators, core objects, and methods. Its syntax comes from the Java and C languages, so many structures from those languages apply to JavaScript as well. One of the key differences is that JavaScript does not have classes (this will be fixed in newer ECMAScript versions); instead, the class functionality is accomplished by object prototypes. The other main difference is that functions are objects, giving functions the capacity to hold executable code and be passed around like any other object.

Developers often see that JavaScript is prototype-based language, but not object oriented. Prototype-based programming is a style of object-oriented programming in which classes are not present, and behavior reuse, known as inheritance in class-based languages, is accomplished through a process of decorating existing objects which serve as prototypes. This model is also known as class-less, prototype-oriented, or instance-based programming.

The original (and most canonical) example of a prototype-based language is the programming language Self developed by David Ungar and Randall Smith. However, the class-less programming style has recently grown increasingly popular, and has been adopted for programming languages such as JavaScript, Cecil, NewtonScript, Io, MOO, REBOL, Kevo, Squeak (when using the Viewer framework to manipulate Morphic components), and several others.

The application of JavaScript to use outside of web pages – for example, in PDF documents, site-specific browsers, and desktop widgets – is also significant. Newer and faster JavaScript VMs and platforms built upon them (notably Node.js) have also increased the popularity of JavaScript for server-side web applications. On the client side, JavaScript was traditionally implemented as an interpreted language but just-in-time compilation is now performed by recent browsers.

Most of JavaScript’s features are common to all conforming ECMAScript implementations, unless explicitly specified otherwise.

JavaScript supports much of the structured programming syntax from C (e.g., if statements, while loops, switch statements, etc.). One partial exception is scoping: C-style block scoping is not supported. Instead, JavaScript has function scoping (although, block scoping using the let keyword was added in JavaScript 1.7). Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion, which allows the semicolons that would normally terminate statements to be omitted [1]. According to the coding conventions, it’s better to keep one entity per one file to make structure more simple and understandable. In the web it causes network problems, because to make system ready to run it is necessary to download all entity on what project depends just to the current execution position.

As were mentioned before, it is important to keep project ready for changes and easy to understand, so developers should have a bunch of files with code. Each file makes request to a server, that slow down page loading speed. It would be great to download all content that we need right now in a compressed form per one request to a server. And fortunately, there are few solutions that may help with minification: YUI Compressor, Google Closure, JSMin, the Dojo compressor and Dean Edwards'

Packer. Each of these tools, however, has drawbacks. JSMIN, for example, does not yield optimal savings, due to its simple algorithm, it must leave many line feed characters in the code in order not to introduce any new bugs.

According to Yahoo!'s Exceptional Performance Team, 40% to 60% of Yahoo!'s users have an empty cache experience and about 20% of all page views are done with an empty cache. This fact outlines the importance of keeping web pages as lightweight as possible. Improving the engineering design of a page or a web application usually yields the biggest savings and that should always be a primary strategy. With the right design in place, there are many secondary strategies for improving performance such as minification of the code, HTTP compression, etc. [7].

The goal of JavaScript minification is always to preserve the operational qualities of the code while reducing its overall byte footprint (both in raw terms and after gzipping, as most JavaScript and CSS served from production web servers is gzipped as part of the HTTP protocol). One more problem that we would face, it is concatenation order. Structure of projects may be extremely complex. Component could depend on another component, which depends on another and so on… It means that order of concatenation makes a huge difference. We should always load independent components first and depended on them components after. **Purpose of the article** – to analyze existing JavaScript minification tools, concatenation and compilation technics.

The main part

JSMin

JSMin is a filter which removes comments and unnecessary whitespace from JavaScript files. It typically reduces filesize by half, resulting in faster downloads. It also encourages a more expressive programming style because it eliminates the download cost of clean, literate self-documentation. This does not change the behavior of the program that it is minifying. The result may be harder to debug. It will definitely be harder to read [4].

Libary first replaces carriage returns ('\r') with linefeeds ('\n'). It replaces all other control characters (including tab) with spaces. It replaces comments in the // form with linefeeds. It replaces comments in the /* */ form with spaces. All runs of spaces are replaced with a single space. All runs of linefeeds are replaced with a single linefeed. It omits spaces except when a space is preceded and followed by a non-ASCII character or by an ASCII letter or digit, or by one of these characters: \, $, _. It is more conservative in omitting linefeeds, because linefeeds are sometimes treated as semicolons. A linefeed is not omitted if it precedes a non-ASCII character or an ASCII letter or digit or one of these characters: \, $, _, {, [, (, +, -. and if it follows a non-ASCII character or an ASCII letter or digit or one of these characters: \, $, _, },],), +, -, ", '. No other characters are omitted or modified. JSMin knows to not modify quoted strings and regular expression literals. It does not obfuscate, but it does uglify.

YUI Compressor

The YUI Compressor is a JavaScript compressor which, in addition to removing comments and white-spaces, obfuscates local variables using the smallest possible variable name. This obfuscation is safe, even when using constructs such as 'eval' or 'with' (although the compression is not optimal is those cases) Compared to jsmin, the average savings is around 20% [4].

The YUI Compressor is also able to safely compress CSS files. The decision on which compressor is being used is made on the file extension (js or css) Compressor is JavaScript minifier designed to be 100% safe and yield a higher compression ratio than most other tools. Tests on the YUI Library have shown savings of over 20% compared to JSMin (becoming 10% after HTTP compression). The YUI Compressor is also able to compress CSS files by using a port of Isaac Schlueter's regular-expression-based CSS minifier.

The YUI Compressor is written in Java and relies on Rhino to tokenize the source JavaScript file. It starts by analyzing the source JavaScript file to understand how it is structured. It then prints out the token stream, omitting as many white space characters as possible, and replacing all local symbols by a 1 (or 2, or 3) letter symbol wherever such a substitution is appropriate in the face of evil features such as eval or with, the YUI Compressor takes a defensive approach by not obfuscating any of the scopes containing the evil statement. The CSS compression algorithm uses a set of finely tuned regular expressions to compress the source CSS file. The YUI Compressor is open-source, so don't hesitate to look at the code to understand exactly how it works.

The YUI Compressor yielded exceptional results, however it was missing one thing. Integration in to my build and deployment process. In IdeaPipe I use a MSBuild script to compile, manipulate, and prepare for publishing. So naturally I built a MSBuild Task to minimize my JavaScript and CSS files. So your should write ant tasks or msbuild.

UglifyJS

This package implements a general-purpose JavaScript parser, compressor, beautifier toolkit. It is developed on NodeJS, but it should work on any JavaScript platform supporting the CommonJS module system. If platform of choice doesn't support CommonJS, developer can easily implement it, or discard the exports.* lines from UglifyJS sources [7].

The tokenizer, parser generates an abstract syntax tree from JS code. It can then traversed by the AST. The second part of this package inspects and manipulates the AST generated by the parser. It provides ability to re-generate JavaScript code from the AST. Optionally indented – you can use this if you want to "beautify" a program that has been compressed, so that you can inspect the source. But you can also run our code generator to print out an AST without any whitespace, so you achieve compression as well.

Another compression tool is shortification of variable names, usually to single characters. UglifyJS's mangler will analyze the code and generate proper variable names, depending on scope and usage, and is smart enough to deal with globals defined elsewhere, or with eval calls or with statements. In short, if eval or with are used in some scope, then all variables in that scope and any variables in the parent scopes will remain unmangled, and any references to such variables remain unmangled as well. Libary joins consecutive var declarations, resolves simple constant expressions. We only do the replacement if the result occupies less bytes; for example 1/3 would translate to 0.333333333333, so in this case we don't replace it. It transforms consecutive statements in blocks are merged into a sequence; in many cases, this leaves blocks with a single statement, so then we can remove the block brackets. UglifyJS removes some unreachable code and warn about it (code that follows a return, throw, break or continue statement, except function/variable declarations). But it should be mentioned, that current minifier has special feature that doesn't relate to minification: beautifier. The beautifier can be used as a general purpose indentation tool. It's useful when you want to make a minified file readable. One limitation, though, is that it discards all comments, so you don't really want to use it to reformat your code, unless you don't have, or don't care about, comments. In fact it's not the beautifier who discards comments – they are dumped at the parsing stage, when we build the initial AST. Comments don't really make sense in the AST, and while we could add nodes for them, it would be inconvenient because we'd have to add special rules to ignore them at all the processing stages. UglifyJS package is used by default in r.js compiler for compiling require.js projects, that combines related scripts together into build layers and minifies them. R.js also Optimizes CSS by inlining CSS files referenced by @import and removing comments [6].

Google Closure Tools

Google Closure Tools is a set of tools to help developers build rich web applications with JavaScript. It was developed by Google for use in their web applications such as Gmail, Google Docs and Google Maps [3]. The Closure tools help developers to build rich web applications with web development tools that are both powerful and efficient. The Closure tools include.

Main module of Closure Compiler is JavaScript minificator, that translates JavaScript code into compact, high-performance code. The compiler removes dead code and rewrites and minimizes what's left so that it downloads and runs quickly. It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls. These checks and optimizations help you write apps that are less buggy and easier to maintain [9]. It also has a comprehensive JavaScript library, that helps in developing JavaScript project with huge trees of dependencies. Developer can just pull what he needs from a large set of reusable UI widgets and controls, and from lower-level utilities for DOM manipulation, server communication, animation, data structures, unit testing, rich-text editing, and more. The Closure Library is server-agnostic, and is intended for use with the Closure Compiler. Closure Templates simplify the task of dynamically generating HTML. They have a simple syntax that is natural for programmers. In contrast to traditional templating systems, in which you use one big template per page, you can think of Closure Templates as small components. The Closure Linter enforces the guidelines set by the Google JavaScript Style Guide. The linter handles style issues so that developer can focus on coding.

Closure Code optimizer supports different types of optimization: ADVANCED_OPTIMIZATIONS, SIMPLE_OPTIMIZATION and WHITESPACE_ONLY mode.

Compilation with ADVANCED_OPTIMIZATIONS achieves extra compression by being more aggressive in the ways that it transforms code and renames symbols. However, this more aggressive approach means that you must take greater care when you use ADVANCED_OPTIMIZATIONS to ensure that the output code works the same way as the input code.

For today it is the most complete tool for JavaScript developers. But to get all the features from Google Closure tool developer should include one more dependency in the code – Google Closure itself. Otherwise, it is not possible to concatenate all files in the right order and to minify them in the right way.

Kjscompiler

Kjscompiler makes compilation of multiple JavaScript files with Google Closure Compiler application in right order [11].

As were described in the block about Google Closure Tools, googles solution has one disadvantage – lack of the possibility to compile whole project without including one more dependency in the code, dependency on Google Closure Compiler. Why is it bad solution? Despite additional functionality provided by that tool, developers may give their preferences to another libraries, to another solutions to create connection between components, developers may do not have any need of linter tool, they may have desire to chose CommonJS or AMD ecosystem themselves.

Kjscompiler eliminates this disadvantage and provides solution based on annotations. With another words, Kjscompiler is some kind of wrapper around Google Closure Compiler, because it includes it and provides additional functionality.

This compiler compiles entire project directory, that contains JavaScript files. It scans project and each file, then creates chain of files for concatenation.

Task of computing optimal concatenation order of files soles via algorithm, that represents each file as a vertex on a graph and each dependency as edge. On a first step algorithm sorts all edges with an order when edges with lowest number of dependencies goes first. On the second step we add edges to a ordered list. To do so, algorithm scans each edge and adds it to a list only in a case if each of its parents (dependencies) is already in the list or if there is not parents for current edge. To deal with cycle dependencies algorithm just adds edges after second step to a ordered list according to ordered after sorting. With a configuration file developer may specify compilation settings, like level of compilation (ADVANCED_OPTIMIZATIONS, SIMPLE_OPTIMIZATION and WHITESPACE_ONLY), regex pattern to select files, that should be concatenated and compressed, basic directory and output file path.

Kjscompiler and TypeScript

TypeScript is a free and open source programming language developed by Microsoft. It is a strict superset of JavaScript, so any existing JavaScript programs are also valid TypeScript programs [5]. TypeScript is designed for development of large applications and compiles down to JavaScript. TypeScript supports header files which can contain type information of existing JavaScript libraries, enabling other programs to use objects defined in the header files as if they were strongly typed TypeScript objects. There are third-party header files for popular libraries like jQuery, MongoDB, Node.js, and D3.js.

For today, there is no solution, that would compile TypeScript project into the one minified file according to the TypeScript file annotations. Another good thing about Kjscompiler is that it can compile produced by TypeScript to Javascript translator JavaScript files. Solution is not perfect, because TypeScipt code requires additional annotations for Kjscompiler, that would duplicate TypeScript references, and that developer is obligated to use translator.

Conclusions

Minificaton is really important technic for a modern web, because about 20 % of all page views are done with and empty cache. his fact outlines the importance of keeping web pages as lightweight as possible. Concatenation represents one of the primary strategies of reducing number of request on a server. Speed of the project in the web relies on its optimality. And to build best solution developer should always ask himself, does he can do better. Concatenation of the files is the most obvious way to reduce number of request on the server. This will be effective for both, client and server sides. As we already discussed servers benefits, let's look at the client side. Modern browsers have powerful caching technics that reduces the latency of file loading. As a result, site is loading faster. With concatenation we could skip unnecessary file checking and reduce its number to one file. Compilation, in the current context, is a technic that does both concatenation and minification. Today we have few solutions that may minify JavaScript code: YUI Compressor, Google Closure, JSMin, the Dojo compressor and Dean Edwards' Packer. In the current article we have analyzed only first three of them. According to the test done with compilation of jquery-1.6.2, Google Closure in basic mode and UglifyJS give the best results: 61,15 % compression for Google Closure and 61,11% compression for UglifyJS. With the advanced mode the results of Google Closure could be much better, but this mod could only be used in the project context. Despite the 0,04 % extra compression of Google Closure it is 5 times slower than UglifyJS. Kjscompiler makes compilation of multiple JavaScript files with Google Closure Compiler application in right order, that removes Google Closure Compiler disadvantage, based on need to include one more dependency to the code, and provides solution based on annotations. With another words, Kjscompiler is some kind of wrapper around Google Closure Compiler, because it includes it and provides additional functionality. With its flexibility and annotation system it is the best solution for modern project compilation. And only Kjscompiler among described compilers could be used with the TypeScript.

References

1. Flanagan, David, Ferguson, Paula. JavaScript: The Definitive Guide (5th ed.) // Flanagan, David, Ferguson, Paula – O'Reilly & Associates, 2006. – 626 p.

2. Andres Hejlsberg. Typescript Language Specification *// Andres Hejlsberg.* – Microsoft, 2012. – *143 p.*

3. Bolin, Michael. Closure: The Definitive Guide // Bolin, Michael. – O'Reilly Media Inc., 2010. – 331 p.

4. The JavaScript Minifier (Douglas Crockford) [Electronic resource]. - Access to resources: http://www.crockford.com/javascript/jsmin.html

5. Introduction to Object-Oriented JavaScript [Electronic resource]. - Access to resources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript

6. A re-introduction to JavaScript [Electronic resource]. - Access to resources: https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

7. YUI Compressor Introduction [Electronic resource]. - Access to resources: http://yui.github.io/yuicompressor/

8. Requirejs optimizer [Electronic resource]. - Access to resources: http://requirejs.org/docs/optimization.html

9. UglifyJS [Electronic resource]. - Access to resources: https://github.com/mishoo/UglifyJS

10. Google developers guide: closure tools [Electronic resource]. - Access to resources: https://developers.google.com/closure/

11. Kjscompiler [Electronic resource]. - Access to resources: https://github.com/knyga/kjscompiler

Reviewer: professor of the Department of Software Engineering Shubin Igor, Kharkiv National University of Radioelectronics

Author ZOLOTAREVA Irina
Kharkiv National University of Economics, professor, Ph.D. E-mail – izolotaryova@gmail.com

Author KNYGA Oleksandr
Kharkiv National University of Economics. E-mail – oleksandrknyga@gmail.com

СОВРЕМЕННАЯ ОПТИМИЗАЦИЯ ПРОЕКТОВ НА JAVASCRIPT

И.А. Золотарева, А.А. Книга

Данная статья посвящена различным способом минификации отдельных JavaScript файлов и целых проектов, а также их конкатенацие. В статье рассмотрены минификаторы JSMin, YUI Compressor, UglifyJS и компиляторы Google Closure и Kjscompiler, описаны их преимущества и приведены сравнительные характеристики.

Ключевые слова: *JavaScript, Kjscompiler, JSMin, YUI Compressor, UglifyJS, Google Closure, TypeScript, ECMAScript, сжатие данных, программирование, оптимизация, компиляция, ECMAScript, минификация, фильтры.*

СУЧАСНА ОПТИМІЗАЦИЯ ПРОЕКТІВ НА JAVASCRIPT

І.О. Золотарьова, О.О. Книга

Дана стаття присвячена різним способом мініфікації окремих JavaScript файлів і цілих проектів, а також їх конкатенації. У статті розглянуті мініфікатори JSMin, YUI Compressor, UglifyJS і компілятори Google Closure і Kjscompiler, описані їх переваги і наведені порівняльні характеристики.

Ключові слова: *JavaScript, Kjscompiler, JSMin, YUI Compressor, UglifyJS, Google Closure, TypeScript, ECMAScript, стиснення даних, програмування, оптимізація, компіляція, ECMAScript, мініфікація, фільтри.*