Obfuscation, minification etc (UglifyJS, Google Closure Compiler, YUI Compressor etc)

Обфускация - это процесс, в результате которого код программы приобретает вид, трудный для анализа. Обфускация, как нетрудно догадаться, осуществляется с целью защиты программного кода и алгоритмов, которые он реализует, от чужих глаз

Существует несколько разных видов обфускации в зависимости от того, какой именно код запутывается при ней. Для тех языков программирования, для которых программа представляется прямо в виде исходных текстов, используются специальные методы, делающие код нечитабельным для человека, но приемлемым для интерпретатора. Среди них - запись программы в одну строку, замена имён переменных и функций, вставка ничего не значащих комментариев.

Обфускации может подвергаться не только программный код, но и другие виды данных: например, существуют специализированные методы обфускации HTML-страниц (особенно эффективно внедрение сценариев JavaScript'е, которые пишут текст динамически, и картинок с текстом). Обфускация HTML-страниц весьма полезна для спамеров, которые с её помощью обходят спам-фильтры.

Помимо защиты, обфускация для некоторых интерпретируемых языков увеличивает скорость выполнения программы - правда, это возможно только при применении оптимизирующих обфускаторов.

Но в большинстве случаев обфускация имеет массу побочных эффектов. В особо неприятных обстоятельствах обфускатор может сделать программу совершенно непригодной к выполнению, в менее тяжёлых случаях в программе могут появиться новые ошибки. Поэтому обфускацию необходимо применять с максимальной осторожностью.

* [Google Closure Compiler](https://developers.google.com/closure/compiler/)
* Убедиться, что стоит [Java](http://java.oracle.com/)
* Скачать и распаковать <http://closure-compiler.googlecode.com/files/compiler-latest.zip>, нам нужен файл compiler.jar.
* Сжать файл my.js: java -jar compiler.jar --charset UTF-8 --js my.js --js\_output\_file my.min.js

Для UglifyJS:

1. Убедиться, что стоит [Node.js](http://nodejs.org/)
2. Поставить npm install -g uglify-js.
3. Сжать файл my.js: uglifyjs my.js -o my.min.js

**[Что делает минификатор?](https://learn.javascript.ru/minification" \l "что-делает-минификатор)**

Все современные минификаторы работают следующим образом:

1. Разбирают JavaScript-код в синтаксическое дерево.

Также поступает любой интерпретатор JavaScript перед тем, как его выполнять. Но затем, вместо исполнения кода…

1. Бегают по этому дереву, анализируют и оптимизируют его.
2. Записывают из синтаксического дерева получившийся код.

## [Оптимизации](https://learn.javascript.ru/minification" \l "оптимизации)

Сжиматель бегает по дереву, ищет «паттерны» — известные ему структуры, которые он знает, как оптимизировать, и обновляет дерево.

В разных минификаторах реализован разный набор оптимизаций, сами оптимизации применяются в разном порядке, поэтому результаты работы могут отличаться. В примерах ниже даётся результат работы GCC.

**Объединение и сжатие констант**

До оптимизации:

function test(a, b) {

run(a, 'my' + 'string', 600 \* 600 \* 5, 1 && 0, b && 0)

}

После:

function test(a,b){run(a,"mystring",18E5,0,b&&0)};

* 'my' + 'string' → "mystring".
* 600 \* 600 \* 5 → 18E5 (научная форма числа, для краткости).
* 1 && 0 → 0.
* b && 0 → без изменений, т.к. результат зависит от b.

**Укорачивание локальных переменных**

До оптимизации:

function sayHi(name, message) {

alert(name +" сказал: " + message);

}

После оптимизации:

function sayHi(a,b){alert(a+" сказал: "+b)};

* Локальная переменная заведомо доступна только внутри функции, поэтому обычно её переименование безопасно (необычные случаи рассмотрим далее).
* Также переименовываются локальные функции.
* Вложенные функции обрабатываются корректно.

**Объединение и удаление локальных переменных**

До оптимизации:

function test(nodeId) {

var elem = document.getElementsById(nodeId);

var parent = elem.parentNode;

alert( parent );

}

После оптимизации GCC:

function test(a){a=document.getElementsById(a).parentNode;alert(a)};

* Локальные переменные были переименованы.
* Лишние переменные убраны. Для этого сжиматель создаёт вспомогательную внутреннюю структуру данных, в которой хранятся сведения о «пути использования» каждой переменной. Если одна переменная заканчивает свой путь и начинает другая, то вполне можно дать им одно имя.
* Кроме того, операции elem = getElementsById и elem.parentNode объединены, но это уже другая оптимизация.

**Уничтожение недостижимого кода, разворачивание if-веток**

До оптимизации:

function test(node) {

var parent = node.parentNode;

if (0) {

alert( "Привет с параллельной планеты" );

} else {

alert( "Останется только один" );

}

return;

alert( 1 );

}

После оптимизации:

function test(){alert("Останется только один")}

* Если переменная присваивается, но не используется, она может быть удалена. В примере выше эта оптимизация была применена к переменной parent, а затем и к параметру node.
* Заведомо ложная ветка if(0) { .. } убрана, заведомо истинная — оставлена.

То же самое будет с условиями в других конструкциях, например a = true ? c : d превратится вa = c.

* Код после return удалён как недостижимый.

**Переписывание синтаксических конструкций**

До оптимизации:

var i = 0;

while (i++ < 10) {

alert( i );

}

if (i) {

alert( i );

}

if (i == '1') {

alert( 1 );

} else if (i == '2') {

alert( 2 );

} else {

alert( i );

}

После оптимизации:

for(var i=0;10>i++;)alert(i);i&&alert(i);"1"==i?alert(1):"2"==i?alert(2):alert(i);

* Конструкция while переписана в for.
* Конструкция if (i) ... переписана в i&&....
* Конструкция if (cond) ... else ... была переписана в cond ? ... : ....

**Инлайнинг функций**

Инлайнинг функции — приём оптимизации, при котором функция заменяется на своё тело.

До оптимизации:

function sayHi(message) {

var elem = createMessage('div', message);

showElement(elem);

function createMessage(tagName, message) {

var el = document.createElement(tagName);

el.innerHTML = message;

return el;

}

function showElement(elem) {

document.body.appendChild(elem);

}

}

После оптимизации (переводы строк также будут убраны):

function sayHi(b) {

var a = document.createElement("div");

a.innerHTML = b;

document.body.appendChild(a)

};

* Вызовы функций createMessage и showElement заменены на тело функций. В данном случае это возможно, так как функции используются всего по разу.
* Эта оптимизация применяется не всегда. Если бы каждая функция использовалась много раз, то с точки зрения размера выгоднее оставить их «как есть».

**Инлайнинг переменных**

Переменные заменяются на значение, если оно заведомо известно.

До оптимизации:

(function() {

var isVisible = true;

var hi = "Привет вам из JavaScript";

window.sayHi = function() {

if (isVisible) {

alert( hi );

alert( hi );

alert( hi );

alert( hi );

alert( hi );

alert( hi );

alert( hi );

alert( hi );

alert( hi );

alert( hi );

alert( hi );

alert( hi );

}

}

})();

После оптимизации:

(function() {

window.sayHi = function() {

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

alert( "Привет вам из JavaScript" );

};

}

})();

* Переменная isVisible заменена на true, после чего if стало возможным убрать.
* Переменная hi заменена на строку.

Казалось бы — зачем менять hi на строку? Ведь код стал ощутимо длиннее!

…Но всё дело в том, что минификатор знает, что дальше код будет сжиматься при помощи gzip. Во всяком случае, все правильно настроенные сервера так делают.

[Алгоритм работы gzip](http://www.gzip.org/algorithm.txt) заключается в том, что он ищет повторы в данных и выносит их в специальный «словарь», заменяя на более короткий идентификатор. Архив как раз и состоит из словаря и данных, в которых дубликаты заменены на идентификаторы.

Если вынести строку обратно в переменную, то получится как раз частный случай такого сжатия — взяли "Привет вам из JavaScript" и заменили на идентификатор hi. Но gzip справляется с этим лучше, поэтому эффективнее будет оставить именно строку. Gzip сам найдёт дубликаты и сожмёт их.

Плюс такого подхода станет очевиден, если сжать gzip оба кода — до и после минификации. Минифицированный gzip-сжатый код в итоге даст меньший размер.

**Разные мелкие оптимизации**

Кроме основных оптимизаций, описанных выше, есть ещё много мелких:

* Убираются лишние кавычки у ключей

{"prop" : "val" } => {prop:"val"}

* Упрощаются простые вызовы Array/Object
* a = new Array() => a = []

o = new Object() => o = {}

Эта оптимизация предполагает, что Array и Object не переопределены программистом. Для включения её в UglifyJS нужен флаг --unsafe.

* …И еще некоторые другие мелкие изменения кода…

## What is the Closure Compiler?

The Closure Compiler is a tool for making JavaScript download and run faster. It is a true compiler for JavaScript. Instead of compiling from a source language to machine code, it compiles from JavaScript to better JavaScript. It parses your JavaScript, analyzes it, removes dead code and rewrites and minimizes what's left. It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls.

## How can I use the Closure Compiler?

You can use the Closure Compiler as:

 An open source Java application that you can run from the command line.

 A simple web application.

 A RESTful API.

To get started with the compiler, see "How do I start" below.

## What are the benefits of using Closure Compiler?

* **Efficiency.** The Closure Compiler reduces the size of your JavaScript files and makes them more efficient, helping your application to load faster and reducing your bandwidth needs.
* **Code checking.** The Closure Compiler provides warnings for illegal JavaScript and warnings for potentially dangerous operations, helping you to produce JavaScript that is less buggy and easier to maintain.
* Google Closure Compiler - уникальный инструмент, разработанный Google для сжатия и обфускации собственного javascript.
* У него есть ряд интересных особенностей, которые отличают его от прочих упаковщиков.
* Вместе с этим, инструмент это достаточно сложный. Основные его фичи скрываются в продвинутом режиме сжатия, для применения которого нужны соответствующие навыки и стиль программирования.
* Google Closure Compiler написан на языке Java, причем так, что его достаточно просто расширить - конечно, если вам знакомы понятия компилятора, синтаксического дерева и понятен Java.

## [YUI Compressor](http://yui.github.io/yuicompressor/)



Developed by Yahoo!, YUI Compressor is a command-line tool written in Java that allows you to minify your JavaScript files. It is 100% safe and yields a higher compression ratio than most other tools. The YUI Compressor is also able to compress CSS files.

<https://jsperf.com/closure-vs-yui/6>