What does “use strict” do in JavaScript, and what is the reasoning behind it?

Asked 9 years, 11 months ago Active 12 days ago Viewed 1.0m times

7210

1476

Recently, I ran some of my JavaScript code through Crockford's JSLint, and it gave the following error:

> Problem at line 1 character 1: Missing "use strict" statement.

Doing some searching, I realized that some people add `"use strict";` into their JavaScript code. Once I added the statement, the error stopped appearing. Unfortunately, Google did not reveal much of the history behind this string statement. Certainly it must have something to do with how the JavaScript is interpreted by the browser, but I have no idea what the effect would be.

So what is `"use strict";` all about, what does it imply, and is it still relevant?

Do any of the current browsers respond to the `"use strict";` string or is it for future use?

javascript syntax jslint use-strict

edited Jan 23 '17 at 1:30

asked Aug 26 '09 at 16:10

Mark Rogers

42.7k 17 71 119

27 Answers

4736

This article about Javascript Strict Mode might interest you: John Resig - ECMAScript 5 Strict Mode, JSON, and More

To quote some interesting parts:

> Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a "strict" operating context. This strict context prevents certain actions from being taken and throws more exceptions.

And:

> Strict mode helps out in a couple ways:
>
> - It catches some common coding bloopers, throwing exceptions.
> - It prevents, or throws errors, when relatively "unsafe" actions are taken (such as gaining access to the global object).
> - It disables features that are confusing or poorly thought out.

Also note you can apply "strict mode" to the whole file... Or you can use it only for a specific function *(still quoting from John Resig's article)*:

```
// Non-strict code...

(function(){
  "use strict";

  // Define your library strictly...
})();

// Non-strict code...
```

Which might be helpful if you have to mix old and new code ;-)

So, I suppose it's a bit like the `"use strict"` you can use in Perl *(hence the name?)*: it helps you make fewer errors, by detecting more things that could lead to breakages.

Strict mode is now supported by all major browsers.

Inside native ECMAScript modules (with `import` and `export` statements) and ES6 classes, strict mode is always enabled and cannot be disabled.

edited Jul 12 at 10:12

FZs
2,929 7 17 32

answered Aug 26 '09 at 16:15

Pascal MARTIN
347k 61 598 622

88 Changing the default after so many years ? Too late for that : it would break so many existing sites/scripts/applications... The only possible thing is to help make things better, for the future. – Pascal MARTIN Mar 4 '10 at 21:54

14 I tried a small code snippet that would be invalid when using `"use strict"` in Firefox 3.6, Safari 5, Chrome 7 and Opera 10.6 (all Mac). No errors whatsoever, so i guess 'use strict' is not supported in any browser yet. Didn't test in IE9 though ;) – Husky Nov 10 '10 at 9:54

10 Quick update: Firefox 4 has complete support for strict mode, and as far as I can tell, no other browser does. Safari and Chrome have "partial"

support, but I don't really know what that means. – Sasha Chedygov Feb 8 '11 at 2:02

27 Chrome 11 seems to pass all of these tests as does IE10 ie.microsoft.com/testdrive/HTML5/TryStrict/Default.html# – gman May 13 '11 at 17:27

11 @Julius - This couldn't have been implemented using a reserved keyword, because then code trying to trigger strict mode would break in old browsers. Adding a "random" string literal doesn't break anything. – nnnnnn Mar 5 '14 at 11:22

1193

It's a new feature of ECMAScript 5. John Resig wrote up a nice summary of it.

It's just a string you put in your JavaScript files (either at the top of your file or inside of a function) that looks like this:

```
"use strict";
```

Putting it in your code now shouldn't cause any problems with current browsers as it's just a string. It may cause problems with your code in the future if your code violates the pragma. For instance, if you currently have `foo = "bar"` without defining `foo` first, your code will start failing...which is a good thing in my opinion.

edited Oct 26 '14 at 10:31 — Peter Mortensen — 14.3k 19 88 116

answered Aug 26 '09 at 16:14 — seth — 31.7k 7 54 57

310 Fail fast and fail loudly. – Niels Bom Jan 29 '13 at 22:20

29 If you are writing Javascript inline in HTML files, start each new block with `<script>"use strict";` . The flag only applies to the block in which it is included. – nobar Oct 5 '13 at 18:50

6 It's funny, this resulted in strings must have single quotes. So write `'use strict';` instead – nilsi Jan 30 '15 at 8:49

1 then what would happen to the hoisting concept of javascript ? – Sunil Sharma May 14 '15 at 8:33

1 @SunilSharma If you try to hoist, but it fails because the variable isn't defined, at the moment it will add it to the global object. With `"use strict";` , it will fail instead. This makes more sense, because if it's adding it to the global object that means that it might not work the next time you run the function / do something else that resets the block, as it will be in the highest block (global). – wizzwizz4 Feb 16 '17 at 21:15

606

The statement `"use strict";` instructs the browser to use the Strict mode, which is a reduced and safer feature set of JavaScript.

List of features (non-exhaustive)

1. Disallows global variables. (Catches missing `var` declarations and typos in variable names)
2. Silent failing assignments will throw error in strict mode (assigning `NaN = 5;`)
3. Attempts to delete undeletable properties will throw (`delete Object.prototype`)
4. Requires all property names in an object literal to be unique (`var x = {x1: "1", x1: "2"}`)
5. Function parameter names must be unique (`function sum (x, x) {...}`)
6. Forbids octal syntax (`var x = 023;` some devs assume wrongly that a preceding zero does nothing to change the number.)
7. Forbids the `with` keyword
8. `eval` in strict mode does not introduce new variables
9. Forbids deleting plain names (`delete x;`)
10. Forbids binding or assignment of the names `eval` and `arguments` in any form
11. Strict mode does not alias properties of the `arguments` object with the formal parameters. (i.e. in `function sum (a,b) { return arguments[0] + b;}` This works because `arguments[0]` is bound to `a` and so on.)
12. `arguments.callee` is not supported

[Ref: Strict mode, *Mozilla Developer Network*]

edited Jan 22 '16 at 18:30

Will
114k 47 288 377

answered Nov 24 '14 at 21:22

gprasant
11.1k 4 38 54

34 Nit: global variables are allowed, just have to be explicit (e.g. `window.foo = bar`). – gcampbell Jun 20 '16 at 9:41

1 Requires all property names in an object literal to be unique (var x = {x1: "1", x1: "2"}) is this valid – Arun Killu Jul 29 '17 at 8:26

1 Your example in 11 is missing a modification of a (otherwise it makes no sense). I. e. function sum(a,b) { a = 0; return arguments[0] + b; } alert(sum(1, 2)) will return 3 with strict mode and 2 without strict mode, due to aliasing. – David Gausmann Mar 14 '18 at 11:04

393

If people are worried about using `use strict` it might be worth checking out this article:

ECMAScript 5 'Strict mode' support in browsers. What does this mean?
NovoGeek.com - Krishna's weblog

It talks about browser support, but more importantly how to deal with it safely:

```
function isStrictMode(){
    return !this;
}
/*
   returns false, since 'this' refers to global object and
   '!this' becomes false
*/

function isStrictMode(){
    "use strict";
    return !this;
}
/*
   returns true, since in strict mode the keyword 'this'
   does not refer to global object, unlike traditional JS.
   So here, 'this' is 'undefined' and '!this' becomes true.
*/
```

edited Jan 22 '18 at 11:13 — Community ♦ — 1 1

answered Jul 15 '12 at 23:25 — Jamie Hutber — 14.1k 25 104 182

113 I disagree. I think this shows why its very useful. In essence it means that this returns its function and not the `window` – Jamie Hutber Feb 26 '13 at 15:25

33 when do you ever want the window with `this` that you can't target with `window` ? – Jamie Hutber Jul 18 '13 at 8:34

12 It refers to itself. `this` belongs to its own function and not the global window – Jamie Hutber Jul 28 '13 at 21:17

24 In the second one `this` one is actually `undefined` . – Broxzier Aug 14 '13 at 11:40

11 The point is that your JS program will start failing due to accessing a property of an undefined, instead of silently doing the wrong thing on the global object. Makes tracking down subtle bugs much easier. – Stephen Chung Jul 31 '15 at 5:47

198

A word of caution, all you hard-charging programmers: applying `"use strict"` to existing code can be hazardous! This thing is not some feel-good, happy-face sticker that you can slap on the code to make it 'better'. With the `"use strict"` pragma, the browser will suddenly THROW exceptions in random places that it never threw before just because at that spot you are doing something that default/loose JavaScript happily allows but strict JavaScript abhors! You may have strictness violations hiding in seldom used calls in your code that will only throw an exception when they do eventually get run - say, in the production environment that your paying customers use!

If you are going to take the plunge, it is a good idea to apply `"use strict"` alongside comprehensive unit tests and a strictly configured JSHint build task that will give you some confidence that there is no dark corner of your module that will blow up horribly just because you've turned on Strict Mode. Or, hey, here's another option: just don't add `"use strict"` to any of your legacy code, it's probably safer that way, honestly. **DEFINITELY DO NOT** add `"use strict"` to any modules you do not own or maintain, like third party modules.

I think even though it is a deadly caged animal, `"use strict"` can be good stuff, but you have to do it right. The best time to go strict is when your project is greenfield and you are starting from scratch. Configure `JSHint/JSLint` with all the warnings and options cranked up as tight as your team can stomach, get a good build/test/assert system du jour rigged like `Grunt+Karma+Chai` , and only THEN start marking all your new modules as `"use strict"` . Be prepared to cure lots of niggly errors and warnings. Make sure everyone understands the gravity by configuring the build to FAIL if `JSHint/JSLint` produces any violations.

My project was not a greenfield project when I adopted `"use strict"` . As a result, my IDE is full of red marks because I don't have `"use strict"` on half my modules, and JSHint complains about that. It's a reminder to me about what refactoring I should do in the future. My goal is to be red mark free due to all of my missing `"use strict"` statements, but that is years away now.

edited Oct 16 '16 at 9:10

Ataur Rahman Munna
3,196 1 16 28

answered Mar 3 '14 at 7:37

DWoldrich
2,922 1 14 18

23 WHY are devs in this thread so cavalier about "use strict"?? It THROWS EXCEPTIONS in *otherwise working JavaScript*, for goodness sakes! Just sprinkle it on the code like sugar on Corn Flakes, eh? NO! BAD! "use strict" should be used cautiously, preferably only in code you control that has unit tests that pass against all major browsers and that exercise all code paths. You got tests? Okay, "use strict" is fine for you, knock yourselves out. – DWoldrich May 8 '14 at 8:38

53 Yes. Obviously "use strict" can break seemingly valid javascript which hasn't broken before. But the code not having broken before is not equal to the code being correct and doing what it's supposed to. Usually referencing undeclared variables signals a typo, etc. Use strict allows you to catch these kinds of errors, and hopefully before you ship production code. – Jostein Kjønigsen Mar 9 '15 at 7:46

3 ... or just apply "use strict" as part of a last pass over your code, fix all the obvious problems, shrug, say "good enough," then take it out for production :) – Wolfie Inu Nov 10 '15 at 8:49

12 Personally, I never/very rarely *add* `"use strict";` to existing code. That being said, I'll almost always use it when I'm writing new code from scratch – Martin Mar 25 '16 at 20:48

3 If you're already using JSLint, you've probably fixed most of the places where "use strict" would break things, though. – Jonathan Cast Dec 4 '17 at 15:22

Using `'use strict';` does not suddenly make your code better.

151

The JavaScript strict mode is a feature in ECMAScript 5. You can enable the strict mode by declaring this in the top of your script/function.

```
'use strict';
```

When a JavaScript engine sees this *directive*, it will start to interpret the code in a special mode. In this mode, errors are thrown up when certain coding practices that could end up being potential bugs are detected (which is the reasoning behind the strict mode).

Consider this example:

```
var a = 365;
var b = 030;
```

In their obsession to line up the numeric literals, the developer has inadvertently initialized variable `b` with an octal literal. Non-strict mode will interpret this as a numeric literal with value `24` (in base 10). However, strict mode will throw an error.

For a non-exhaustive list of specialties in strict mode, see this answer.

Where should I use `'use strict';` ?

- In my *new* JavaScript application: **Absolutely!** Strict mode can be used as a whistleblower when you are doing something stupid with your code.
- In my *existing* JavaScript code: **Probably not!** If your existing JavaScript code has statements that are prohibited in strict-mode, the application will simply break. If you want strict mode, you should be prepared to debug and correct your existing code. This is why *using `'use strict';` does not suddenly make your code better*.

How do I use strict mode?

1. Insert a `'use strict';` statement on top of your script:

```
// File: myscript.js

'use strict';
var a = 2;
....
```

Note that everything in the file `myscript.js` will be interpreted in strict mode.

2. Or, insert a `'use strict';` statement on top of your function body:

```
function doSomething() {
    'use strict';
    ...
}
```

Everything in the *lexical scope* of function `doSomething` will be interpreted in strict mode. The word *lexical scope* is important here. See this answer for a better explanation.

What things are prohibited in strict mode?

I found a nice article describing several things that are prohibited in strict mode (note that this is not an exclusive list):

> ### Scope
>
> Historically, JavaScript has been confused about how functions are scoped. Sometimes they seem to be statically scoped, but some features make them behave like they are dynamically scoped. This is confusing, making programs difficult to read and understand. Misunderstanding causes bugs. It also is a problem for performance. Static scoping would permit variable binding to happen at compile time, but the requirement for dynamic scope means the binding must be deferred to runtime, which comes with a significant performance penalty.
>
> Strict mode requires that all variable binding be done statically. That means that the features that previously required dynamic binding must be eliminated or modified. Specifically, the with statement is eliminated, and the eval function's ability to tamper with the environment of its caller is severely restricted.
>
> One of the benefits of strict code is that tools like YUI Compressor can do a better job when processing it.
>
> ### Implied Global Variables
>
> JavaScript has implied global variables. If you do not explicitly declare a variable, a global variable is implicitly declared for you. This makes programming easier for beginners because they can neglect some of their basic housekeeping chores. But it makes the management of larger programs much more difficult and it significantly degrades reliability. So in strict mode, implied global variables are no longer created. You should explicitly declare all of your variables.
>
> ### Global Leakage

> There are a number of situations that could cause `this` to be bound to the global object. For example, if you forget to provide the `new` prefix when calling a constructor function, the constructor's `this` will be bound unexpectedly to the global object, so instead of initializing a new object, it will instead be silently tampering with global variables. In these situations, strict mode will instead bind `this` to `undefined`, which will cause the constructor to throw an exception instead, allowing the error to be detected much sooner.
>
> ## Noisy Failure
>
> JavaScript has always had read-only properties, but you could not create them yourself until ES5's `Object.createProperty` function exposed that capability. If you attempted to assign a value to a read-only property, it would fail silently. The assignment would not change the property's value, but your program would proceed as though it had. This is an integrity hazard that can cause programs to go into an inconsistent state. In strict mode, attempting to change a read-only property will throw an exception.
>
> ## Octal
>
> The octal (or base 8) representation of numbers was extremely useful when doing machine-level programming on machines whose word sizes were a multiple of 3. You needed octal when working with the CDC 6600 mainframe, which had a word size of 60 bits. If you could read octal, you could look at a word as 20 digits. Two digits represented the op code, and one digit identified one of 8 registers. During the slow transition from machine codes to high level languages, it was thought to be useful to provide octal forms in programming languages.
>
> In C, an extremely unfortunate representation of octalness was selected: Leading zero. So in C, `0100` means 64, not 100, and `08` is an error, not 8. Even more unfortunately, this anachronism has been copied into nearly all modern languages, including JavaScript, where it is only used to create errors. It has no other purpose. So in strict mode, octal forms are no longer allowed.
>
> ## Et cetera
>
> The arguments pseudo array becomes a little bit more array-like in ES5. In strict mode, it loses its `callee` and `caller` properties. This makes it possible to pass your `arguments` to untrusted code without giving up a lot of confidential context. Also, the `arguments` property of functions is eliminated.
>
> In strict mode, duplicate keys in a function literal will produce a syntax error. A function can't have two parameters with the same name. A function can't have a variable with the same name as one of its parameters. A function can't `delete` its own variables. An attempt to `delete` a non-configurable property now throws an exception. Primitive values are not implicitly wrapped.

Reserved words for future JavaScript versions

ECMAScript 5 adds a list of reserved words. If you use them as variables or arguments, strict mode will throw an error. The reserved words are:

> `implements` , `interface` , `let` , `package` , `private` , `protected` , `public` , `static` , and `yield`

Further Reading

- Strict Mode - JavaScript | MDN
- Browser support for strict mode
- Transitioning to strict mode

edited May 23 '17 at 12:02
Community ♦
1 1

answered Jan 29 '16 at 11:35

sampathsris
14.3k 8 46 79

1 it is very nice explanation. However, I have one doubt that can I use "strict" mode in conjunction with other java script libraries, like Angular js? – UVM Aug 5 '16 at 3:39

2 @UVM: The strict-mode directive affects only lexical scope. i.e. only the file/function that it is declared. If you have another file/function that does not have the `'use strict'` directive, they will be executed in non-strict mode, even when called from a function running in strict mode. See this asnwer for an explanation. – sampathsris Aug 5 '16 at 4:11

This is not entirely correct. 'use strict' does change the way code is executed. – CyberEd Oct 22 '16 at 19:04

2 On second look, you're right. I thought you meant it only threw exceptions, but didn't change the way the code worked (like changing `this`). Now I see you were referring to calling other functions. – CyberEd Oct 23 '16 at 3:55

2 There are some cases where octal is useful. The C syntax for it is horrible, but I would have liked to have seen languages add a new octal syntax which could then allow the leading-zero form to be deprecated. Of course, for Javascript to have supported the leading-zero form was just silly. – supercat Feb 9 '17 at 21:08

131

I strongly recommend every developer to start using strict mode now. There are enough browsers supporting it that strict mode will legitimately help save us from errors we didn't even know were in your code.

Apparently, at the initial stage there will be errors we have never encountered before. To get the full benefit, we need to do proper testing after switching to strict mode to make sure we have caught everything. Definitely we don't just throw `use strict` in our code and assume there are no errors. So the churn is that it's time to start using this incredibly useful language feature to write better code.

For example,

```
var person = {
    name : 'xyz',
    position : 'abc',
    fullname : function () {  "use strict"; return this.name; }
};
```

JSLint is a debugger written by Douglas Crockford. Simply paste in your script, and it'll quickly scan for any noticeable issues and errors in your code.

edited Feb 25 '16 at 15:54

Willi Mentzel
11.9k 11 57 75

answered Jul 5 '13 at 19:38

Pank
6,858 10 27 44

5 @JamieHutber : Please visit this link caniuse.com/use-strict AND kangax.github.io/es5-compat-table. It will give exact idea for all browser. – Pank Jan 18 '14 at 13:21

92

I would like to offer a somewhat more founded answer complementing the other answers. I was hoping to edit the most popular answer, but failed. I tried to make it as comprehensive and complete as I could.

You can refer to the MDN documentation for more information.

`"use strict"` a directive introduced in ECMAScript 5.

Directives are similar to statements, yet different.

- `use strict` does not contain key words: The directive is a simple expression statement, which consists of a special string literal (in single or double quotes). JavaScript engines, that do not implement ECMAScript 5, merely see an expression statement without side effects. It is expected that future versions of ECMAScript standards introduce `use` as a real key word; the quotes would thereby become obsolete.
- `use strict` can be used only at the beginning of a script or of a function, i.e. it must precede every other (real) statement. It does not have to be the first instruction in a script of function: it can be preceded by other statement expressions that consist of string literals (and JavaScript implementations can treat them as implementation specific directives). String literals statements, which follow a first real statement (in a script or function) are simple expression statements. Interpreters must not interpret them as directives and they have no effect.

The `use strict` directive indicates that the following code (in a script or a function) is strict code. The code in the highest level of a script (code that is not in a function) is considered strict code when the script contains a `use strict` directive. The content of a function is considered strict code when the function itself is defined in a strict code or when the function contains a `use strict` directive. Code that is passed to an `eval()` method is considered strict code when `eval()` was called from a strict code or contains the `use strict` directive itself.

The strict mode of ECMAScript 5 is a restricted subset of the JavaScript language, which eliminates relevant deficits of the language and features more stringent error checking and higher security. The following lists the differences between strict mode and normal mode (of which the first three are particularly important):

- You cannot use the `with` -statement in strict mode.
- In strict mode all variables have to be declared: if you assign a value to an identifier that has not been declared as variable, function, function parameter, catch-clause parameter or property of the global `Object` , then you will get a `ReferenceError` . In normal mode the identifier is implicitly declared as a global variable (as a property of the global `Object`)
- In strict mode the keyword `this` has the value `undefined` in functions that were invoked as functions (not as methods). (In normal mode `this` always points to the global `Object`). This difference can be used to test if an implementation supports the strict mode:

```
var hasStrictMode = (function() { "use strict"; return this===undefined }());
```

- Also when a function is invoked with `call()` or `apply` in strict mode, then `this` is exactly the value of the first argument of the `call()` or `apply()` invocation. (In normal mode `null` and `undefined` are replaced by the global `Object` and values, which are not objects, are cast into objects.)
- In strict mode you will get a `TypeError` , when you try to assign to readonly properties or to define new properties for a non extensible object. (In normal mode both simply fail without error message.)
- In strict mode, when passing code to `eval()` , you cannot declare or define variables or functions in the scope of the caller (as you can do it in normal mode). Instead, a new scope is created for `eval()` and the variables and functions are within that scope. That scope is destroyed after `eval()` finishes execution.
- In strict mode the arguments-object of a function contains a static copy of the values, which are passed to that function. In normal mode the arguments-object has a somewhat "magical" behaviour: The elements of the array and the named function parameters reference both the same value.
- In strict mode you will get a `SyntaxError` when the `delete` operator is followed by a non qualified identifier (a variable, function or function parameter). In normal mode the `delete` expression would do nothing and is evaluated to `false` .
- In strict mode you will get a `TypeError` when you try to delete a non configurable property. (In normal mode the attempt simply fails and the `delete` expression is evaluated to `false`).

- In strict mode it is considered a syntactical error when you try to define several properties with the same name for an object literal. (In normal mode there is no error.)
- In strict mode it is considered a syntactical error when a function declaration has multiple parameters with the same name. (In normal mode there is no error.)
- In strict mode octal literals are not allowed (these are literals that start with `0x` . (In normal mode some implementations do allow octal literals.)
- In strict mode the identifiers `eval` and `arguments` are treated like keywords. You cannot change their value, cannot assign a value to them, and you cannot use them as names for variables, functions, function parameters or identifiers of a catch block.
- In strict mode are more restrictions on the possibilities to examine the call stack. `arguments.caller` and `arguments.callee` cause a `TypeError` in a function in strict mode. Furthermore, some caller- and arguments properties of functions in strict mode cause a `TypeError` when you try to read them.

edited Jul 12 '15 at 20:31

answered May 15 '15 at 6:58

Ely
8,129 3 33 55

3 "In strict mode octal literals are not allowed (these are literals that start with 0x ...)" octal literals start with a leading `0` . – Alex Gittemeier Aug 11 '16 at 19:52

80

My two cents:

One of the goals of strict mode is to allow for faster debugging of issues. It helps the developers by throwing exception when certain wrong things occur that can cause silent & strange behaviour of your webpage. The moment we use `use strict` , the code will throw out errors which helps developer to fix it in advance.

Few important things which I have learned after using `use strict` :

Prevents Global Variable Declaration:

```
var tree1Data = { name: 'Banana Tree',age: 100,leafCount: 100000};

function Tree(typeOfTree) {
    var age;
    var leafCount;

    age = typeOfTree.age;
    leafCount = typeOfTree.leafCount;
```

```
    nameoftree = typeOfTree.name;
};

var tree1 = new Tree(tree1Data);
console.log(window);
```

Now,this code creates `nameoftree` in global scope which could be accessed using `window.nameoftree` . When we implement `use strict` the code would throw error.

> Uncaught ReferenceError: nameoftree is not defined

Sample

Eliminates `with` statement :

`with` statements can't be minified using tools like uglify-js. They're also deprecated and removed from future JavaScript versions.

Sample

Prevents Duplicates :

When we have duplicate property, it throws an exception

> Uncaught SyntaxError: Duplicate data property in object literal not allowed in strict mode

```
"use strict";
var tree1Data = {
    name: 'Banana Tree',
    age: 100,
    leafCount: 100000,
    name:'Banana Tree'
};
```

There are few more but I need to gain more knowledge on that.

answered Oct 21 '14 at 13:31

community wiki
Shubh

59

If you use a browser released in the last year or so then it most likely supports JavaScript Strict mode. Only older browsers around before ECMAScript 5 became the current standard don't support it.

The quotes around the command make sure that the code will still work in older browsers as well (although the things that generate a syntax error in strict mode will generally just cause the script to malfunction in some hard to detect way in those older browsers).

answered Mar 10 '12 at 3:31

Stephen
685 5 2

11 Then what does it do? – Anish Gupta May 20 '12 at 14:47

6 ... this describes *in part* the compatibility, but not what it actually does. – courtsimas Jul 11 '12 at 16:04

56

When adding `"use strict";`, the following cases will throw a **SyntaxError** before the script is executing:

- *Paving the way for future ECMAScript versions*, using one of the newly reserved keywords (in prevision for *ECMAScript 6*): `implements`, `interface`, `let`, `package`, `private`, `protected`, `public`, `static`, and `yield`.
- Declaring function in blocks

```
if(a<b){ function f(){} }
```

- Octal syntax

```
var n = 023;
```

- `this` point to the global object.

```
function f() {
     "use strict";
     this.a = 1;
};
f();
```

- Declaring twice the same name for a property name in an object literal

```
{a: 1, b: 3, a: 7}
```

This is no longer the case in ECMAScript 6 (bug 1041128).

- Declaring two function arguments with the same name function

```
f(a, b, b){}
```

- Setting a value to an undeclared variable

```
function f(x){
   "use strict";
   var a = 12;
   b = a + x*35; // error!
}
f();
```

- Using `delete` on a variable name `delete myVariable;`
- Using `eval` or `arguments` as variable or function argument name

```
"use strict";
arguments++;
var obj = { set p(arguments) { } };
try { } catch (arguments) { }
function arguments() { }
```

Sources:

- Transitioning to strict mode on MDN
- Strict mode on MDN
- JavaScript's Strict Mode and Why You Should Use It on Colin J. Ihrig's blog (archived version)

edited Oct 24 '17 at 10:50 — Palec — 8,104 5 43 92

answered Dec 23 '15 at 3:10 — zangw — 25.1k 7 98 122

51

Strict mode makes several changes to normal JavaScript semantics:

- eliminates some JavaScript silent errors by changing them to throw errors.
- fixes mistakes that make it difficult for JavaScript engines to perform optimizations.

- prohibits some syntax likely to be defined in future versions of ECMAScript.

for more information vistit Strict Mode- Javascript

edited Oct 29 '14 at 17:34

alexyorke
2,166 3 28 53

answered Mar 27 '14 at 12:18

Renganathan M G
2,904 2 24 29

49

"Use Strict"; is an insurance that programmer will not use the loose or the bad properties of JavaScript. It is a guide, just like a ruler will help you make straight lines. "Use Strict" will help you do "Straight coding".

Those that prefer not to use rulers to do their lines straight usually end up in those pages asking for others to debug their code.

Believe me. The overhead is negligible compared to poorly designed code. Doug Crockford, who has been a senior JavaScript developer for several years, has a very interesting post here. Personally, I like to return to his site all the time to make sure I don't forget my good practice.

Modern JavaScript practice should always evoke the "Use Strict"; pragma. The only reason that the ECMA Group has made the "Strict" mode optional *is to permit less experienced coders access to JavaScript and give then time to adapt to the new and safer coding practices.*

edited Oct 26 '14 at 10:34

Peter Mortensen
14.3k 19 88 116

answered May 31 '13 at 18:29

user2436758
799 5 3

66 The reason strict mode is optional has nothing to do with what you've stated. The real reason *is to not break existing code that may not conform*. – George Jempty Oct 31 '13 at 13:34

17 Indeed, the less experienced coders ought to be the *first* ones to enable the "use strict"; – Antti Haapala Aug 19 '14 at 5:37

44

Including `use strict` in the beginning of your all sensitive JavaScript files from this point is a small way to be a better JavaScript programmer and avoid random variables becoming global and things change silently.

edited Feb 25 '16 at 15:55

Willi Mentzel
11.9k 11 57 75

answered Sep 5 '14 at 12:53

Placeholder
2,430 5 21 33

39

Quoting from w3schools:

> ## The "use strict" Directive
>
> The "use strict" directive is new in JavaScript 1.8.5 (ECMAScript version 5).
>
> It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
>
> The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
>
> With strict mode, you can not, for example, use undeclared variables.
>
> ## Why Strict Mode?
>
> Strict mode makes it easier to write "secure" JavaScript.
>
> Strict mode changes previously accepted "bad syntax" into real errors.
>
> As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.
>
> In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties.
>
> In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

Please refer to http://www.w3schools.com/js/js_strict.asp to know more

edited Aug 13 '18 at 3:15

Shog9 ♦
133k 32 211 228

answered Apr 29 '15 at 10:10

Heich-B
528 5 13

34

`"use strict"` makes JavaScript code to run in *strict mode*, which basically means everything needs to be defined before use. The main reason for using strict mode is to avoid accidental global uses of undefined methods.

Also in strict mode, things run faster, some warnings or silent warnings throw fatal errors, it's better to always use it to make a neater code.

`"use strict"` is widely needed to be used in ECMA5, **in ECMA6 it's part of JavaScript by default**, so it doesn't need to be added if you're using ES6.

Look at these statements and examples from MDN:

> **The "use strict" Directive**
> The "use strict" directive is new in JavaScript 1.8.5 (ECMAScript version 5). It is not a statement, but a literal expression, ignored by earlier versions of JavaScript. The purpose of "use strict" is to indicate that the code should be executed in "strict mode". With strict mode, you can not, for example, use undeclared variables.
>
> **Examples of using "use strict":**
> Strict mode for functions: Likewise, to invoke strict mode for a function, put the exact statement "use strict"; (or 'use strict';) in the function's body before any other statements.

1) strict mode in functions

```
function strict() {
    // Function-level strict mode syntax
    'use strict';
    function nested() { return 'And so am I!'; }
    return "Hi!  I'm a strict mode function!  " + nested();
}
function notStrict() { return "I'm not strict."; }

console.log(strict(), notStrict());
```

2) whole-script strict mode

```
'use strict';
var v = "Hi! I'm a strict mode script!";
console.log(v);
```

3) Assignment to a non-writable global

```
'use strict';

// Assignment to a non-writable global
```

```
var undefined = 5; // throws a TypeError
var Infinity = 5; // throws a TypeError

// Assignment to a non-writable property
var obj1 = {};
Object.defineProperty(obj1, 'x', { value: 42, writable: false });
obj1.x = 9; // throws a TypeError

// Assignment to a getter-only property
var obj2 = { get x() { return 17; } };
obj2.x = 5; // throws a TypeError

// Assignment to a new property on a non-extensible object.
var fixed = {};
Object.preventExtensions(fixed);
fixed.newProp = 'ohai'; // throws a TypeError
```

You can read more on MDN.

edited Aug 30 '18 at 15:16
Palec
8,104 5 43 92

answered May 22 '17 at 12:38
Alireza
59.4k 14 195 127

29

There's a good talk by some people who were on the ECMAScript committee: Changes to JavaScript, Part 1: ECMAScript 5" about how incremental use of the `"use strict"` switch allows JavaScript implementers to clean up a lot of the dangerous features of JavaScript without suddenly breaking every website in the world.

Of course it also talks about just what a lot of those misfeatures are (were) and how ECMAScript 5 fixes them.

edited Mar 29 '14 at 19:39

answered Mar 29 '14 at 0:47
FutureNerd
669 6 9

21

Small examples to compare:

Non-strict mode:

```
for (i of [1,2,3]) console.log(i)

// output:
// 1
// 2
// 3
```

Run code snippet Expand snippet

Strict mode:

```
'use strict';
for (i of [1,2,3]) console.log(i)

// output:
// Uncaught ReferenceError: i is not defined
```

Run code snippet Expand snippet

Non-strict mode:

```
String.prototype.test = function () {
  console.log(typeof this === 'string');
};

'a'.test();

// output
// false
```

Run code snippet Expand snippet

```
String.prototype.test = function () {
  'use strict';
```

```
  console.log(typeof this === 'string');
};

'a'.test();

// output
// true
```

Run code snippet Expand snippet

edited Nov 18 '18 at 16:31

answered Aug 21 '16 at 21:43

Foo

1

2 Notice that the code above will add the **i** variable to the global scope (generally this not a best practice and **strict mode** helps to avoid that). – michael Feb 18 '17 at 12:41

Can somebody explain the second example? I don't get it. Shouldn't `this === 'a'` in both examples? – MaximeW Feb 6 at 9:24

18

Note that `use strict` was introduced in ECmaScript 5 and was kept since then.

Below are the conditions to trigger strict mode in ES6 and ES7:

- Global code is strict mode code if it begins with a Directive Prologue that contains a Use Strict Directive (see 14.1.1).
- Module code is always strict mode code.
- All parts of a *ClassDeclaration* or a *ClassExpression* are strict mode code.
- Eval code is strict mode code if it begins with a Directive Prologue that contains a Use Strict Directive or if the call to eval is a direct eval (see 12.3.4.1) that is contained in strict mode code.
- Function code is strict mode code if the associated *FunctionDeclaration, FunctionExpression, GeneratorDeclaration, GeneratorExpression, MethodDefinition, or ArrowFunction* is contained in strict mode code or if the code that produces the value of the function's [[ECMAScriptCode]] internal slot begins with a Directive Prologue that contains a Use Strict Directive.
- Function code that is supplied as the arguments to the built-in Function and Generator constructors is strict mode code if the last argument is a String that when processed is a *FunctionBody* that begins with a Directive Prologue that contains a Use Strict Directive.

answered Apr 12 '16 at 0:25

Oriol
4,375 1 25 33

12

The main reasons why developers should use `"use strict"` are:

1. Prevents accidental declaration of global variables.Using `"use strict()"` will make sure that variables are declared with `var` before use. Eg:

```
function useStrictDemo(){
 'use strict';
 //works fine
 var a = 'No Problem';

 //does not work fine and throws error
 k = "problem"

 //even this will throw error
 someObject = {'problem': 'lot of problem'};
}
```

2. N.B: The `"use strict"` directive is only recognized at the beginning of a script or a function.
3. The string `"arguments"` cannot be used as a variable:

```
"use strict";
var arguments = 3.14;    // This will cause an error
```

4. Will restrict uses of keywords as variables. Trying to use them will throw errors.

In short will make your code less error prone and in turn will make you write good code.

To read more about it you can refer here.

edited Jun 18 '17 at 3:14

answered Nov 18 '16 at 9:53

Pritam Banerjee
11.5k 6 52 73

"use strict"; is the ECMA effort to make JavaScript a little bit more robust. It brings in JS an attempt to make it at least a little "strict"

11

(other languages implement strict rules since the 90s). It actually "forces" JavaScript developers to follow some sort of coding best practices. Still, JavaScript is very fragile. There is no such thing as typed variables, typed methods, etc. I strongly recommend JavaScript developers to learn a more robust language such as Java or ActionScript3, and implement the same best practices in your JavaScript code, it will work better and be easier to debug.

edited Jun 14 '16 at 23:18

hologram
421 1 5 19

answered May 3 '16 at 12:59

PippoApps.com
481 3 16

11

JavaScript “strict” mode was introduced in ECMAScript 5.

```
(function() {
  "use strict";
  your code...
})();
```

Writing `"use strict";` at the very top of your JS file turns on strict syntax checking. It does the following tasks for us:

1. shows an error if you try to assign to an undeclared variable
2. stops you from overwriting key JS system libraries
3. forbids some unsafe or error-prone language features

`use strict` also works inside of individual functions. It is always a better practice to include `use strict` in your code.

Browser compatibility issue: The "use" directives are meant to be backwards-compatible. Browsers that do not support them will just see a string literal that isn't referenced further. So, they will pass over it and move on.

edited Feb 6 at 9:40

jkdev
5,713 6 37 68

answered Nov 11 '16 at 5:40

Rabin Pantha
778 8 18

10

`use strict` is a way to make your code safer, cause you can't use dangerous features which can work not as you expect.And as was writed before it makes code more strict.

answered May 17 '16 at 22:31

Просто программист
132 1 9

10

Use Strict is used to show common and repeated errors so that it is handled differently , and changes the way java script runs , such changes are :

- Prevents accidental globals
- No duplicates
- Eliminates with
- Eliminates this coercion
- Safer eval()
- Errors for immutables

you can also read this article for the details

edited Oct 17 '16 at 14:09

answered Oct 17 '16 at 13:59

Wesam
701 2 13 25

10

Normally, JavaScript does not follow strict rules, hence increasing chances of errors. After using `"use strict"` , the JavaScript code should follow strict set of rules as in other programming languages such as use of terminators, declaration before initialization, etc.

If `"use strict"` is used, the code should be written by following a strict set of rules, hence decreasing the chances of errors and ambiguities.

edited Jan 4 at 2:04

Pang
7,153 16 68 108

answered Nov 20 '16 at 16:23

Bikash Chapagain
109 1 8

4

"use strict"; Defines that JavaScript code should be executed in "strict mode".

- The "use strict" directive was new in ECMAScript version 5.

- It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
- The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
- With strict mode, you can not, for example, use undeclared variables.

All modern browsers support "use strict" except *Internet Explorer 9 and lower*.

Disadvantage

If a developer used a library that was in strict mode, but the developer was used to working in normal mode, they might call some actions on the library that wouldn't work as expected.

Worse, since the developer is in normal mode, they don't have the advantages of extra errors being thrown, so the error might fail silently.

Also, as listed above, strict mode stops you from doing certain things.

People generally think that you shouldn't use those things in the first place, but some developers don't like the constraint and want to use all the features of the language.

- **For basic example and for reference go through :**

  https://www.tutorialsteacher.com/javascript/javascript-strict

edited Jul 19 at 12:01

answered Jan 28 at 10:42

Ashish

561 1 12

1

Strict mode can prevent memory leaks.

Please check the function below written in non-strict mode:

```
function getname(){
    name = "Stack Overflow"; // Not using var keyword
    return name;
}
getname();
console.log(name); // Stack Overflow
```

In this function, we are using a variable called `name` inside the function. Internally, the compiler will first check if there is any variable declared with that particular name in that particular function scope. Since the compiler understood that there is no such variable, it will check in the outer scope. In our case, it is the global scope. Again, the compiler understood that there is also no variable declared in the global space with that name, so it creates such a variable for us in the global space. Conceptually, this variable will be created in the global scope and will be available in the entire application.

Another scenario is that, say, the variable is declared in a child function. In that case, the compiler checks the validity of that variable in the outer scope, i.e., the parent function. Only then it will check in the global space and create a variable for us there. That means additional checks need to be done. This will affect the performance of the application.

Now let's write the same function in strict mode.

```
"use strict"
function getname(){
    name = "Stack Overflow"; // Not using var keyword
    return name;
}
getname();
console.log(name);
```

We will get the following error.

```
Uncaught ReferenceError: name is not defined
at getname (<anonymous>:3:15)
at <anonymous>:6:5
```

Here, the compiler throws the reference error. In strict mode, the compiler does not allow us to use the variable without declaring it. So memory leaks can be prevented. In addition, we can write more optimized code.

edited Jul 9 at 6:01

Pang
7,153 16 68 108

answered Jul 8 at 8:46

Jerin K Alexander
178 5 14