Use of [square brackets] around JavaScript variables

Asked 9 years, 10 months ago Active 3 months ago Viewed 55k times

47

We have received some JavaScript from an agency that looks wrong, but works.

For some reason they are adding [square brackets] around variables, thus:

```
var some_variable = 'to=' + [other_variable];
```

11

This works, but the square brackets seem completely superfluous.

Is there a purpose to this syntax or is it technically incorrect, but ignored by the browser?

javascript syntax

asked Oct 27 '09 at 9:05

Richard Everett

34k 50 169 264

It is well explained in MDN documentation inhere: developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/… – Tom Boja May 7 at 7:18

2 @TomBoja that MDN page applies to destructuring, which I don't think relates so this scenario. – Richard Everett May 8 at 0:29

7 Answers

43

Square brackets means new Array.

```
var ar=new Array("a","b");
var ar=["a","b"]; //Equal to the syntax above
```

in that situation there's no difference if you use square brackets or not because if it's an array it is converted to string, but if you delete the brackets it takes less time because it doesn't have to build a new array and convert it but it works with a simple string.

edited Jan 18 '13 at 13:15

Richard Everett
34k 50 169 264

answered Oct 27 '09 at 9:10

mck89
13.6k 15 72 98

1 So in my exampe a new array containing one element is getting created? – Richard Everett Oct 27 '09 at 9:13

1 yes. it means that there is one array element created. – Murvinlai Oct 27 '09 at 9:15

4 Yes and then it's converted to string – mck89 Oct 27 '09 at 9:15

Woah! Ten years ago, back when we were using `var` instead of `let` or `const` ! – colakollektiv Aug 15 at 15:50

37

Just in case anyone else arrives here while trying to find out what some weird/new syntax involving [square brackets] (seen in someone else's Javascript) might possibly be, like I was...

Nowadays, with ES6, we also have [] used on the left-hand side for destructuring arrays, e.g.

```
const names = ['Luke', 'Eva', 'Phil'];
const [first] = names;
console.log(first); // 'Luke'
const [first, second] = names;
console.log(first, second); // 'Luke' 'Eva'
```

For further info see http://www.deadcoderising.com/2017-03-28-es6-destructuring-an-elegant-way-of-extracting-data-from-arrays-and-objects-in-javascript/ or google 'es6 destructuring'.

answered Dec 17 '17 at 21:17

Jim Smart
381 3 2

Thanks, came here for this same reason. – inolasco Mar 7 at 16:54

thanks for using my name in the example. – Phillll_t Jul 12 at 0:36

Even without changing the Array prototype there are differences:

32

```
var other_variable;
var some_variable = 'to=' + [other_variable];
```

If other_variable is undefined, the return value with the array is 'to= ' ,

Without the array the return value is 'to= undefined' .

edited May 9 '13 at 4:04

answered Oct 27 '09 at 13:18

kennebec

82.7k 25 89 122

15

```
b = "bar" + ["foo"]
```

This is syntactically correct, but indeed very, very, superfluous. This is how it works:

```
["foo"]
```

JavaScript takes the string "foo" and converts it to an array with one element, "foo":

```
"bar" + ["foo"]
```

when `+` is used, and one of the operands is a string, "bar" in this case, JavaScript converts the second one to a string. Since operand two is an Array, the `Array.toString` method is called, which, by default, returns all elements joined by a comma. We have one element, and the result will be equal to this element, i.e., in this context `"foo"` is equivalent to `["foo"]` .

If you redefine `Array.toString` you can see better what's going on:

```
alert("bar" + ["foo"])
Array.prototype.toString = function() { return "???"; }
alert("bar" + ["foo"])
```

edited Apr 1 '13 at 19:45

the Tin Man

139k 27 181 260

answered Oct 27 '09 at 9:19

user187291

46.5k 17 80 126

9

I bet someone told that person:

- "Do the string concatenation with an array, it's faster!"

Meaning:

```
var some_variable = ['to=', other_variable].join("");
```

Which is apparently faster for lots of concatenations, but totally irrelevant, as that code will probably run just once anyway.
`Premature_optimization = sqrt(all_evil)` !

And the poor chap did that other irrelevant thing...

I love people.

answered Oct 27 '09 at 9:28

Victor
8,857 2 21 34

9

It's possible to construct a situation where this:

```
var some_variable = 'to=' + other_variable;
```

and this:

```
var some_variable = 'to=' + [other_variable];
```

produce different results. Specifically, if the `Array.prototype.toString()` method (or, I suppose, the `Array.prototype.join()` method) has been changed from its default, anything could happen. For example, extra functionality could be added to the `Array.prototype.toString()` method to generate logging information, do some lookups, or anything really.

The likelihood that this has been done is slim, I'd imagine, but it needs to be mentioned for completeness.

answered Oct 27 '09 at 10:20

Tim

8,079 24 45

6

May be this ..

Global Variable access with the Square Bracket Notation

The square bracket notation requires that there be some sort of object reference to the left of the brackets.

```
["document"] //Array literal, not a Property Accessor!
```

-will produce an error if an attempt is made to assign a value to it, as it will be treated as an Array literal, if an attempt to read from it is made the one element array containing the string within the brackets is returned. Global variables are normally referenced by their one identifier alone. This would seem to exclude global variables from the possibility of being referenced using a string that held their identifier name or an expression that built, or returned, their name. However, javascript global variables (and global function names for that matter) are properties of a global object. Any identifier that holds a reference to the global object can be used to the left of the square brackets to form a property accessor that refers to a global variable.

In a web browser the global object is the window (or frame) in which the script is running. Each window (or frame) object contains a number of properties, at least two of which are references to the window (global object) itself. These properties are 'window' and 'self'. These property names can be used as the identifier to the left of the square brackets when referring to global variables. So given a global variable defined as:-

```
var anyName = 0;
```

- that global variable can be referenced as:-

  window["anyName"]

As with any other use of the square bracket notation, the string within the brackets can be held in a variable or constructed/returned by an expression.

Code that is executing in the global context, the code within global functions (except Object constructors invoked with the new keyword) and inline code outside of any functions, could also use the this keyword to refer to the global object. The this keyword refers to an object depending on the execution context. For code executing in the global context this is the global object (on a web browser, the window object). As a result, the above variable could be referred to as this["anyName"], but only in code that is executing in the global context.

However, using the this keyword is very likely to be confusing, especially in scripts that include custom javascript objects where the methods (and constructors) of those objects would be using this to refer to their own object instances.

Some javascript implementations do not have a property of the global object that refers to the global object. Rather than trying to use the this keyword to access global variables it is possible to create your own global variable that refers to the global object.

```
var myGlobal = this;
```

- executed as inline code at the start of a script will assign a reference to the global object (this in that context). From then on all global variables can be referenced with square bracket notation as:-

  myGlobal["anyName"];
- and expect `myGlobal` to refer to the global object from any execution context.

answered Oct 27 '09 at 9:15

TigerTiger
7,648 13 50 70