Associative Arrays in JavaScript

You may be finding conflicting information about associative arrays in JavaScript. Well, the answer is quite simple… and then things start to get a little odd

If you are frustrated because you have been getting different answers on this subject, I"ve got good news and bad news. The good news is, the answer is simple: associative arrays are not supported in JavaScript. Arrays in JavaScript are index-based. Plain and simple, end of conversation. But the bad new is, it's not quite the end of the conversation. The reason for this is that the following code actually works just fine:

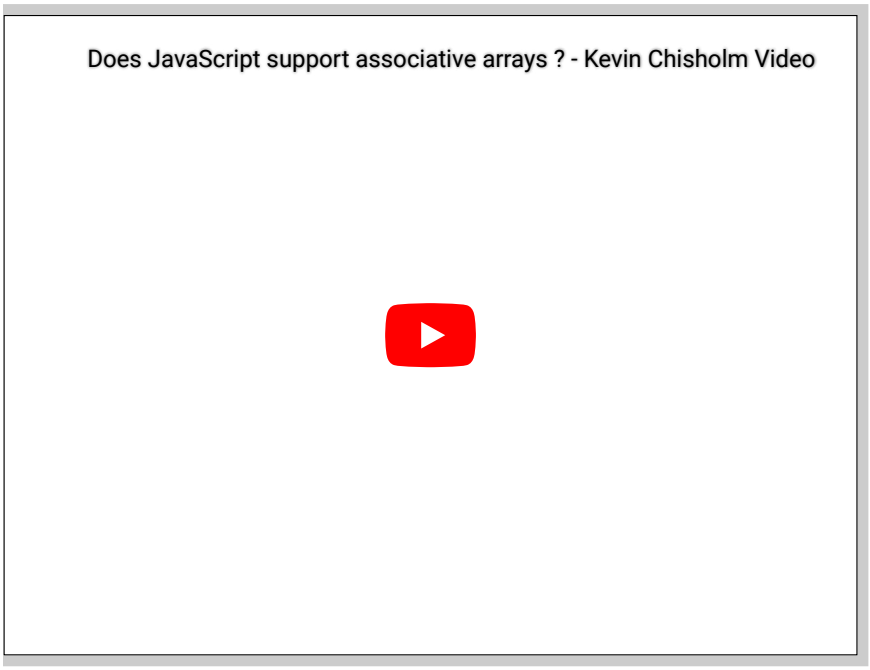
```
var arr = ["mon","tues","wed"];

arr["drink"] = "beer";
arr["music"] = "jazz";

console.log(arr[0]); //mon
console.log(arr["drink"]); //beer
```

[insert shrugged shoulders here] *"…ok Kevin, so what's the problem ?'*

The problem is: you do not have an array with five elements. You have an array with three elements, and two properties.

Huh?

Yup, this is the part where JavaScript array objects behave in an unexpected way. But hang in there, it's actually kind of cool once you understand what is happening.



Arrays are Objects

Arrays in JavaScript are numerically indexed: each array element's "key" is its numeric index. So, in a way, each element is anonymous. This is because when you use methods of the Array object such as array.shift() or array.unshift(), each element's index changes. So, after using array.shift(), array element # 2 becomes array element # 1, and so on. (array.pop() and array.push() may change the length of the array, but they don't change the existing array element's index numbers because you are dealing with the end of the array.)

All this is to say that in a JavaScript array, each element can only be identified by an index, which will always be a number, and you always have to assume that this number can change, which means that the whole "key/value" idea is out the window (i.e. no associative arrays in JavaScript. Sorry.).

OK smarty-pants, if you can't have associative arrays in JavaScript, why does this work: arr["drink"] = "beer" ?

Glad you asked. This is because in JavaScript, arrays inherit from Object(). Whether you use an array literal or instantiate the array constructor, you are creating an object, plain and simple. Consider the following:

Example # 1

```
var arr = new Array(3);
var arr = [];
var arr = ["mon","tues","wed"];
```

In Example # 1, we create an array in three different ways. The first line creates a new array with three elements, but they are all undefined. The second line creates a new array, but it is empty, with no elements (this is an array literal). The third line creates an array literal, but we provide values for the elements as we define the array. In all three cases, you are creating an instance of the Array() constructor, which inherits from Object(). So, these are ALL objects.

Example # 2:

```
var arr = [];

console.log("the length of arr is: " + arr.length) // 0
console.dir(arr) //there are no child objects

arr[0] = "mon";
arr[1] = "tues";
arr[2] = "wed";

arr["drink"] = "beer";
arr["music"] = "jazz";

console.log("the length of arr is: " + arr.length); // 3
console.dir(arr);
```

```
15
16 0 -> "mon"
17 1 -> "tues"
18 2 -> "wed"
19 drink -> "beer"
20 music -> "jazz"
```

In Example # 2, we create an array literal, but it is empty. (var arr = []; works just fine, but it is an empty array.) When we check the length property and try to inspect the object with console.dir(arr), we can clearly see that it is empty. Then we add elements to the array, and add named properties (e.g. arr["drink"] = "beer"). But when checking the array's length property, and inspecting the object, we can see that the actual "array" part of the array still has only three elements (i.e. "music" and "drink" are NOT elements in the array), and that "music" and "drink" are properties of the array object.

Arrays are Objects with their own special "array-ish" properties

What is happening, is that the Array() constructor returns an instance object that has some special members that other objects such as Function() and Date() do not. So when your code says: arr["drink"] = "beer" you are simply adding a new property to your object, which happens to be an array, and that property has a name of "drink", and you have assigned the value of "beer" to it.

You could have easily assigned a number, an object, an anonymous function, or one of JavaScript's other data types. This property "drink" has no connection to the elements in the array. It does not increase the value of the array's "length" property, and it cannot be accessed via the array-*ish* methods such as pop() or shift().

Let's see what happens when we take advantage of this object's "array-ness."

Example # 3:

```
1  arr.push({
2      name: "bart simpson",
3      phone: "555-1212"
4  });
5
6  arr.push(function(){
7   console.log('i am an element of the array "arr", that
8              happens to be an anonymous function');
9  });
10
11 arr.testMe = function(){
12  console.log('i am "testMe", a property of the array "arr",
13             that happens to be an anonymous function');
14 };
15
16 console.log("The length of arr is: " + arr.length);
17 console.dir(arr);
18 The output for example # 3 would be:
19
20 The length of arr is: 5
21 0 -> "mon"
22 1 -> "tues"
23 2 -> "wed"
24 3 -> Object { name = "bart simpson", phone = "555-1212" }
25 4 -> function()
26 drink -> "beer"
27 music -> "jazz"
28 testMe -> function()
```

OK, so things are gettin' pretty weird, right? Yep, but it's all cool stuff, and at the end of the day, it's no big deal. It just illustrates the way objects work in JavaScript. Let's run it down:

- First, we use the JavaScrpt Array() object's push() method to dynamically add an element to the array. It just so happens that this new element is an object literal, with two properties. Its index becomes 3.
- Next, we use the same push() method to dynamically add another element to the array. This new element is an anonymous function. Its index becomes 4.
- Next, we create a new property for our array called "testMe". This new property happens to be an anonymous function. It has NO index, because it is NOT an element in the array, just a new property that we have added.
- Next, we use the console to check the array's length property, which is now "5", and we inspect it.
- Dont' forget it is an array, but it is also sill an object; Array() inherits from Object(). When inspecting the object, we see that our two uses of push() did, in fact, add two new elements to the array; one is an object, the other is an anonymous function. We also have "testMe", wich is a new property of arr.

Ok, so what happens if we attempt to access the two functions that we added? Oh, they will run just fine, but remember: one is an element in the array, the other is a property of the arr object. So we access them differently:

Example # 4:

```
arr[4]();
arr.testMe();
```

The output for Example # 4 would be:

```
i am an element or the array "arr",
     that happens to be an anonymous function

i am "testMe", a property of the array "arr",
     that happens to be an anonymous function
```

In each case, we are simply executing a function. It's just that in the first case, that function is an element in the "arr" array. So, we have to access it using its index, which happens to be "4". This can get tricky fast, and care should be taken in doing this kind of thing, but just to illustrate a point: array elements in JavaScript can be of any data type. In the second case, we access the function by its name "testMe", because it is a PROPERTY of the array, not an element. Much easier, and there are no issues, because "testMe" will always be "testMe", so it's easy to access.

Summary

This is all pretty overboard. I mean, don't we usually want this: var arr = ["mon","tues","wed"] ? Well, yes. Most of the time we do. But the point of this post is two-fold:

1. JavaScript does NOT support associative arrays. Period.
2. Arrays are objects, so properties can be added any time. Those properties can be any data type.

In JavaScript, arrays are best used as arrays, i.e., numerically indexed lists. The great thing is that those elements in the array can be of any data type. Index # 0 can be a string, # 1 can be an object, # 2 can be an anonymous function, and # 3 can be another array.

But once you start doing things like this: arr["drink"] = "beer", you are swimming in somewhat dangerous waters. Unless you really know what you are doing, you will get odd behavior because arr["drink"] is NOT a numerically indexed "member" of the array (it is not an array "element"), and does NOT have the relation to arr[0] and arr[1] that you may think it does.

As soon as you start doing things like: arr["drink"] = "beer", it is time to consider putting this key-value pair in an object literal. You don't have to, but it's a better way to manage your data, and the approach leverages JavaScript's strengths, instead of wrestling with the somewhat odd nature of it's underlying architecture.

P.S.: If you really wanna see some odd JavaScript array behavior, try this:

```
var arr = [];

var arr = ["mon","tues","wed"];
arr["drink"] = "beer";
arr["music"] = "jazz";

arr.push({
    name: "bart simpson",
    phone: "555-1212"
});

arr.push(function(){
 console.log("anonymous function as element");
});

arr.testMe = function(){
 console.log('anonymous function as property');
};

for (var prop in arr){
console.log("Prop: " + arr[prop])
}
```

The strange output of this one is for another discussion : –)

Helpful links for associative arrays in JavaScript

- http://www.quirksmode.org/js/associative.html
- http://blog.xkoder.com/2008/07/10/javascript-associative-arrays-demystified/
- http://andrewdupont.net/2006/05/18/javascript-associative-arrays-considered-harmful/

One Comment

JavaScript Interview Questions: Arrays | Kevin Chisholm - Blog

[...] Hint: https://blog.kevinchisholm.com/javascript/associative-arrays-in-javascript/ [...]

November 18, 2017 at 6:41 am

Comments are closed.