How to do associative array/hashing in JavaScript

Asked 10 years, 2 months ago Active 1 year, 2 months ago Viewed 709k times

543

118

I need to store some statistics using JavaScript in a way like I'd do it in C#:

```
Dictionary<string, int> statistics;

statistics["Foo"] = 10;
statistics["Goo"] = statistics["Goo"] + 1;
statistics.Add("Zoo", 1);
```

Is there an `Hashtable` or something like `Dictionary<TKey, TValue>` in JavaScript?
How could I store values in such a way?

javascript dictionary hashtable

edited Jul 25 '18 at 18:43
Davide Cannizzo
1,018 1 10 20

asked Jul 30 '09 at 17:52
George2
19.8k 100 298 439

1 js is loosely typed, so there's no way to just declare a string or int, you can just declare a var and assign it a string or int. :D – Gordon Gustafson Jul 30 '09 at 18:07

You might want to check out xDict. jsfiddle.net/very/MuVwd It is a dictionary String=>anything written in Javascript. – Robert Feb 10 '12 at 16:51

This article has an excellent explanation of how associative arrays are implemented under-the-hood in Javascript jayconrod.com/posts/52/a-tour-of-v8-object-representation – Shuklaswag Sep 1 '18 at 3:59

11 Answers

533

Use JavaScript objects as associative arrays.

> Associative Array: In simple words associative arrays use Strings instead of Integer numbers as index.

Create an object with

```
var dictionary = {};
```

> Javascript allows you to add properties to objects by using the following syntax:

```
Object.yourProperty = value;
```

An alternate syntax for the same is:

```
Object["yourProperty"] = value;
```

If you can also create key to value object maps with the following syntax

```
var point = { x:3, y:2 };

point["x"] // returns 3
point.y // returns 2
```

> You can iterate through an associative array using the for..in loop construct as follows

```
for(var key in Object.keys(dict)){
  var value = dict[key];
  /* use key/value for intended purpose */
}
```

edited Jul 16 '18 at 6:30

datashaman
3,515 1 17 23

answered Jul 30 '09 at 18:01

Alek Davis
9,023 2 33 42

35 Note that the author's approach of initializing an "associative array" with `new Array()` is frowned up. The article eventually mentions its drawbacks and suggests `new Object()` or `{}` as preferred alternatives, but that's near the end and I fear most readers won't get that far. – Daniel Lubarov May 16 '15 at 5:55

21 Fail. JavaScript doesn't support object references as keys, while something like Flash/AS3 Dictionary does. In JavaScript, `var obj1 = {}; var obj2 = {}; var table= {}; table[obj1] = "A"; table[obj2] = "B"; alert(table[obj1]); //displays B` , because it can't differentiate between keys

obj1 and obj2; they're both converted to string and just become something like "Object". Total fail, and makes type-safe serialization with references and cyclic-references intact difficult or non-performant in JavaScript. It's easy in Flash/AS3. – Triynko Jul 15 '15 at 16:52

Well, the only way in JS we can validate by checking the equality or defining an **equals** method by something like this: `Point.prototype.equals = function(obj) { return (obj instanceof Point) && (obj.x === this.x) && (obj.y === this.y); };` – Nadeem Dec 30 '16 at 20:08

1 @Leo console.log({A:'B',C:'D'}[foo]) should give you A B. – ychaouche Jan 20 at 14:07

2 @Leo The example seems wrong. `for... in` for a dictionary will loop over its keys, so `Object.keys` seems malplaced there. `Object.keys` returns an array of the keys of the dictionary, and `for... in` for an array loops over *its* "keys", which for an array are its indices, not its values. – JHH Jan 23 at 23:01

426

```
var associativeArray = {};
associativeArray["one"] = "First";
associativeArray["two"] = "Second";
associativeArray["three"] = "Third";
```

If you are coming from an object-oriented language you should check this article.

edited Sep 23 '15 at 12:39

answered Jul 30 '09 at 18:25

Dani Cricco

6,616 8 26 34

37 You could also do this in fewer lines: var associativeArray = {"one" : "First", "two" : "second", "three" : "Third"}; Then associativeArray["one"] returns "First" and assocativeArray["four"] returns null. – Tony Wickham Mar 24 '15 at 1:17

2 Sorry @JuusoOhtonen, I wrote the post 6 years ago (it is incredible how fast time goes by). I have updated the link. Please check it and don't hesitate to ask if you have any doubts – Dani Cricco Sep 23 '15 at 12:42

Just perfect, runs even in IE, great thanks – partizanos Nov 25 '15 at 10:46

130

Unless you have a specific reason not to, just use a normal object. Object properties in Javascript can be referenced using hashtable-style syntax:

```
var hashtable = {};
hashtable.foo = "bar";
hashtable['bar'] = "foo";
```

Both `foo` and `bar` elements can now then be referenced as:

```
hashtable['foo'];
hashtable['bar'];
// or
hashtable.foo;
hashtable.bar;
```

Of course this does mean your keys have to be strings. If they're not strings they are converted internally to strings, so it may still work, YMMV.

edited Jul 20 '17 at 19:57

rogerdpack
38.4k 21 149 275

answered Sep 24 '08 at 23:23

roryf
21.7k 15 75 100

1 Keys as integers caused me no problem. stackoverflow.com/questions/2380019/… – Jonas Elfström Mar 5 '10 at 12:54

9 Jonas: bear in mind that your integers are converted to strings when the property is being set: `var hash = {}; hash[1] = "foo"; alert(hash["1"]);` alerts "foo". – Tim Down Jul 8 '10 at 22:09

17 What if one of your keys is "**proto**" or "**parent**"? – PleaseStand Jan 3 '11 at 2:21

5 Note that ***Objects cannot be used as keys*** in JavaScript. Well, they can, but they're converted to their String representations, so any Object will end up as the exact same key. See @TimDown's jshashtable suggestion below. – ericsoco Jul 5 '13 at 20:22

21 This example is confusing because you're using foo and bar as both key and value in two instances. Much clearer to show that `var dict = {}; dict.key1 = "val1"; dict["key2"] = "val2";` dict's key1 element can be referenced equivalently by both `dict["key1"]` and `dict.key1` . – Jim Mar 27 '14 at 20:05

128

All modern browsers support a javascript Map object. There are a couple of reasons that make using a Map better than Object:

- An Object has a prototype, so there are default keys in the map.
- The keys of an Object are Strings, where they can be any value for a Map.
- You can get the size of a Map easily while you have to keep track of size for an Object.

Example:

```
var myMap = new Map();

var keyObj = {},
    keyFunc = function () {},
    keyString = "a string";

myMap.set(keyString, "value associated with 'a string'");
myMap.set(keyObj, "value associated with keyObj");
myMap.set(keyFunc, "value associated with keyFunc");

myMap.size; // 3

myMap.get(keyString);    // "value associated with 'a string'"
myMap.get(keyObj);       // "value associated with keyObj"
myMap.get(keyFunc);      // "value associated with keyFunc"
```

If you want keys that are not referenced from other objects to be garbage collected, consider using a WeakMap instead of a Map.

answered May 6 '15 at 21:42

Vitalii Fedorenko
82.8k 21 136 110

5 Hopefully in a few years this will be the most voted for answer. – Cameron Lee May 13 '15 at 21:45

1 @CameronLee surely it will – Loïc Faure-Lacroix Jun 7 '15 at 10:30

1 This `Map` is barely useful when your key is an object but should be compared by value, not reference. – Siyuan Ren Dec 11 '15 at 15:07

6 More than a year after this answer was written, it's still NOT true that "all modern browsers support Map." Only on the desktop can you count on at least basic Map support. Not on mobile devices. E.g., Android browser has no Map support at all. Even on the Desktop, some implementations are incomplete. For instance, IE11 still doesn't support enumerating via "for...of...", so if you want IE compatibility you have to use the disgusting .forEach kludge. Also, JSON.stringify() doesn't work for Map in any browser I've tried. Also initializers don't work in IE or Safari. – Dave Burton Aug 22 '16 at 7:07

1 This is pretty safe, now for **Desktop**: `|Basic support| Chrome: v38; Edge: v12; Firefox: v13; IE: v11; Opera: v25; Safari: v7.1;` Still not for **Mobile**: `|Basic support| Chrome: v38; Android: none; Firefox: v13; IE Mobile: none; Opera Mobile: none; Safari Mobile: v8;` – kayleeFrye_onDeck Apr 21 '17 at 1:07

49

Since every object in JS behaves like - and is generally implemented as - a hashtable, i just go with that...

```
var hashSweetHashTable = {};
```

answered Sep 24 '08 at 23:18

Shog9 ♦

134k 32 212 228

26 Downvoted because it doesn't show how to actually access values in the "hashtable". – IQAndreas Mar 14 '15 at 7:16

I'm 9 years late (I didn't know much of anything about programming, let alone this site back then), but... What if you are trying to store points on a map, and need to see if something is already at a point on a map? In that case, you would be best using HashTable for this, looking up by coordinates (an *object*, not a *string*). – Mike Warren Sep 29 '17 at 13:52

@MikeWarren `if (hashSweetHashTable.foo)` should enter the if block if `foo` is set. – Koray Tugay Aug 30 at 1:01

20

so in C# the code looks like:

```
Dictionary<string,int> dictionary = new Dictionary<string,int>();
dictionary.add("sample1", 1);
dictionary.add("sample2", 2);
```

or

```
var dictionary = new Dictionary<string, int> {
    {"sample1", 1},
    {"sample2", 2}
};
```

in JavaScript

```
var dictionary = {
    "sample1": 1,
    "sample2": 2
}
```

C# dictionary object contains useful methods like `dictionary.ContainsKey()` in JavaScript we could use the `hasOwnProperty` like

```
if (dictionary.hasOwnProperty("sample1"))
    console.log("sample1 key found and its value is"+ dictionary["sample1"]);
```

edited Nov 14 '16 at 22:41

aloisdg
11.3k 2 51 65

answered Nov 5 '15 at 0:23

Raj
301 4 11

1 Upvote for me not having to write an answer about `hasOwnProperty` – brichins Dec 5 '18 at 22:58

18

If you require your keys to be be any object rather than just strings then you could use my jshashtable.

answered Jun 1 '10 at 11:32

Tim Down
263k 60 394 479

3 How many hours did I spend stumbling around the fact that Objects can't really be used as keys for JS-style-Object-as-associative-arrays before I found this? Thank you, Tim. – ericsoco Jul 5 '13 at 20:19

1 Flash/AS3 Dictionary, along with most other languages, support object references as keys. JavaScript still hasn't implemented it yet, but I think it's in a future spec as a some kind of Map class. Again with the polyfills in the meantime; so much for standards. Oh, wait... finally in 2015, Map appears to have arrived: stackoverflow.com/a/30088129/88409, and is supported by "modern" browsers, lol: kangax.github.io/compat-table/es6/#Map (and not really widely supported). Only a decade behind AS3. – Triynko Jul 15 '15 at 17:07

Tim, perhaps you should update jshashtable to use Map() where available. – Dave Burton Aug 22 '16 at 7:14

1 @DaveBurton: Good plan. I shall do so as soon as I have some time. – Tim Down Aug 22 '16 at 10:11

6

I created this to achieve some problem, such as object key mapping, ability of enumeration (with `forEach()` method) and clearing.

```
function Hashtable() {
    this._map = new Map();
    this._indexes = new Map();
    this._keys = [];
    this._values = [];
    this.put = function(key, value) {
        var newKey = !this.containsKey(key);
        this._map.set(key, value);
        if (newKey) {
            this._indexes.set(key, this.length);
            this._keys.push(key);
```

```
            this._values.push(value);
        }
    };
    this.remove = function(key) {
        if (!this.containsKey(key))
            return;
        this._map.delete(key);
        var index = this._indexes.get(key);
        this._indexes.delete(key);
        this._keys.splice(index, 1);
        this._values.splice(index, 1);
    };
    this.indexOfKey = function(key) {
        return this._indexes.get(key);
    };
    this.indexOfValue = function(value) {
        return this._values.indexOf(value) != -1;
    };
    this.get = function(key) {
        return this._map.get(key);
    };
    this.entryAt = function(index) {
        var item = {};
        Object.defineProperty(item, "key", {
            value: this.keys[index],
            writable: false
        });
        Object.defineProperty(item, "value", {
            value: this.values[index],
            writable: false
        });
        return item;
    };
    this.clear = function() {
        var length = this.length;
        for (var i = 0; i < length; i++) {
            var key = this.keys[i];
            this._map.delete(key);
            this._indexes.delete(key);
        }
        this._keys.splice(0, length);
    };
    this.containsKey = function(key) {
        return this._map.has(key);
    };
    this.containsValue = function(value) {
        return this._values.indexOf(value) != -1;
    };
    this.forEach = function(iterator) {
```

```
        for (var i = 0; i < this.length; i++)
            iterator(this.keys[i], this.values[i], i);
    };
    Object.defineProperty(this, "length", {
        get: function() {
            return this._keys.length;
        }
    });
    Object.defineProperty(this, "keys", {
        get: function() {
            return this._keys;
        }
    });
    Object.defineProperty(this, "values", {
        get: function() {
            return this._values;
        }
    });
    Object.defineProperty(this, "entries", {
        get: function() {
            var entries = new Array(this.length);
            for (var i = 0; i < entries.length; i++)
                entries[i] = this.entryAt(i);
            return entries;
        }
    });
}
```

Documentation of class `Hashtable`

Methods:

- `get(key)`
  Returns the value associated to the specified key.
  Parameters:
  `key` : The key from which retrieve the value.

- `put(key, value)`
  Associates the specified value to the specified key.
  Parameters:
  `key` : The key to which associate the value.
  `value` : The value to associate to the key.

- `remove(key)`
  Removes the specified key with its value.
  Parameters:
  `key` : The key to remove.

- `clear()`
  Clears all the hashtable, removing both the keys and values.

- `indexOfKey(key)`
  Returns the index of the specified key, based on the adding order.
  Parameters:
  `key` : The key of which get the index.

- `indexOfValue(value)`
  Returns the index of the specified value, based on the adding order.
  Parameters:
  `value` : The value of which get the index.
  Notes:
  This information is retrieved by `indexOf()` method of an array, so it compares object just with `toString()` method.

- `entryAt(index)`
  Returns an object with two properties: key and value, representing the entry at the specified index.
  Parameters:
  `index` : The index of the entry to get.

- `containsKey(key)`
  Returns whether the hashtable contains the specified key.
  Parameters:
  `key` : The key to check.

- `containsValue(value)`
  Returns whether the hashtable contains the specified value.
  Parameters:
  `value` : The value to check.

- `forEach(iterator)`
  Iterates all entries in the specified `iterator` .
  Parameters:
  `value` : A method with 3 parameters: `key` , `value` and `index` , where `index` represents the index of the entry.

Properties:

- `length` (*Read-only*)
  Gets the count of the entries in the hashtable.
- `keys` (*Read-only*)
  Gets an array of all keys in the hashtable.
- `values` (*Read-only*)
  Gets an array of all values in the hashtable.
- `entries` (*Read-only*)
  Gets an array of all entries in the hashtable. They're represented in the same form of the method `entryAt()`.

edited Oct 14 '17 at 23:23

answered Oct 13 '17 at 22:16

Davide Cannizzo
1,018 1 10 20

5

```
function HashTable() {
    this.length = 0;
    this.items = new Array();
    for (var i = 0; i < arguments.length; i += 2) {
        if (typeof (arguments[i + 1]) != 'undefined') {
            this.items[arguments[i]] = arguments[i + 1];
            this.length++;
        }
    }

    this.removeItem = function (in_key) {
        var tmp_previous;
        if (typeof (this.items[in_key]) != 'undefined') {
            this.length--;
            var tmp_previous = this.items[in_key];
            delete this.items[in_key];
        }

        return tmp_previous;
    }

    this.getItem = function (in_key) {
        return this.items[in_key];
    }
```

```
    this.setItem = function (in_key, in_value) {
        var tmp_previous;
        if (typeof (in_value) != 'undefined') {
            if (typeof (this.items[in_key]) == 'undefined') {
                this.length++;
            } else {
                tmp_previous = this.items[in_key];
            }

            this.items[in_key] = in_value;
        }

        return tmp_previous;
    }

    this.hasItem = function (in_key) {
        return typeof (this.items[in_key]) != 'undefined';
    }

    this.clear = function () {
        for (var i in this.items) {
            delete this.items[i];
        }

        this.length = 0;
    }
}
```

answered Nov 4 '11 at 15:51

Birey

1,550 14 20

1 For people who are down voting this, can you please comment why? This answer was posted in 2011 and not in current date. – Birey Jun 15 '15 at 15:38

2 I didn't down-vote but... you should not use an array as an object. Not 100% sure if this was your intent. Use slice on arrays not delete to re-index; delete is ok but will set to undefined -- better to be explicit; use = undefined on an object too b/c it's faster (but more memory). In short: always use an object: `{}` not an array: `[]` or `new Array()` if you intend to have string keys otherwise the js engine has a problem -- it will either see 2 types for 1 variable which means no optimization or it will run with array and realize it has to change to object (possible re-allocation). – Graeme Wicksted Sep 4 '15 at 20:13

2 Just like with Alex Hawkins's answer, please provide some explanation why this rather complex looking code is actually useful and better than the other shorter answers given here. – Thomas Tempelmann Feb 4 '16 at 11:04

2

https://gist.github.com/alexhawkins/f6329420f40e5cafa0a4

```
var HashTable = function() {
  this._storage = [];
  this._count = 0;
  this._limit = 8;
}


HashTable.prototype.insert = function(key, value) {
  //create an index for our storage location by passing it through our hashing function
  var index = this.hashFunc(key, this._limit);
  //retrieve the bucket at this particular index in our storage, if one exists
  //[[ [k,v], [k,v], [k,v] ] , [ [k,v], [k,v] ]  [ [k,v] ] ]
  var bucket = this._storage[index]
    //does a bucket exist or do we get undefined when trying to retrieve said index?
  if (!bucket) {
    //create the bucket
    var bucket = [];
    //insert the bucket into our hashTable
    this._storage[index] = bucket;
  }

  var override = false;
  //now iterate through our bucket to see if there are any conflicting
  //key value pairs within our bucket. If there are any, override them.
  for (var i = 0; i < bucket.length; i++) {
    var tuple = bucket[i];
    if (tuple[0] === key) {
      //overide value stored at this key
      tuple[1] = value;
      override = true;
    }
  }

  if (!override) {
    //create a new tuple in our bucket
    //note that this could either be the new empty bucket we created above
    //or a bucket with other tupules with keys that are different than
    //the key of the tuple we are inserting. These tupules are in the same
    //bucket because their keys all equate to the same numeric index when
    //passing through our hash function.
    bucket.push([key, value]);
    this._count++
      //now that we've added our new key/val pair to our storage
      //let's check to see if we need to resize our storage
      if (this._count > this._limit * 0.75) {
```

```
        this.resize(this._limit * 2);
      }
    }
    return this;
  };


  HashTable.prototype.remove = function(key) {
    var index = this.hashFunc(key, this._limit);
    var bucket = this._storage[index];
    if (!bucket) {
      return null;
    }
    //iterate over the bucket
    for (var i = 0; i < bucket.length; i++) {
      var tuple = bucket[i];
      //check to see if key is inside bucket
      if (tuple[0] === key) {
        //if it is, get rid of this tuple
        bucket.splice(i, 1);
        this._count--;
        if (this._count < this._limit * 0.25) {
          this._resize(this._limit / 2);
        }
        return tuple[1];
      }
    }
  };



  HashTable.prototype.retrieve = function(key) {
    var index = this.hashFunc(key, this._limit);
    var bucket = this._storage[index];

    if (!bucket) {
      return null;
    }

    for (var i = 0; i < bucket.length; i++) {
      var tuple = bucket[i];
      if (tuple[0] === key) {
        return tuple[1];
      }
    }

    return null;
  };
```

```
HashTable.prototype.hashFunc = function(str, max) {
  var hash = 0;
  for (var i = 0; i < str.length; i++) {
    var letter = str[i];
    hash = (hash << 5) + letter.charCodeAt(0);
    hash = (hash & hash) % max;
  }
  return hash;
};


HashTable.prototype.resize = function(newLimit) {
  var oldStorage = this._storage;

  this._limit = newLimit;
  this._count = 0;
  this._storage = [];

  oldStorage.forEach(function(bucket) {
    if (!bucket) {
      return;
    }
    for (var i = 0; i < bucket.length; i++) {
      var tuple = bucket[i];
      this.insert(tuple[0], tuple[1]);
    }
  }.bind(this));
};


HashTable.prototype.retrieveAll = function() {
  console.log(this._storage);
  //console.log(this._limit);
};

/*****************************TESTS*****************************/

var hashT = new HashTable();

hashT.insert('Alex Hawkins', '510-599-1930');
//hashT.retrieve();
//[ , , , [ [ 'Alex Hawkins', '510-599-1930' ] ] ]
hashT.insert('Boo Radley', '520-589-1970');
//hashT.retrieve();
//[ , [ [ 'Boo Radley', '520-589-1970' ] ], , [ [ 'Alex Hawkins', '510-599-1930' ] ] ]
hashT.insert('Vance Carter', '120-589-1970').insert('Rick Mires', '520-589-
1970').insert('Tom Bradey', '520-589-1970').insert('Biff Tanin', '520-589-1970');
//hashT.retrieveAll();
```

```
/*
[ ,
  [ [ 'Boo Radley', '520-589-1970' ],
    [ 'Tom Bradey', '520-589-1970' ] ],
  ,
  [ [ 'Alex Hawkins', '510-599-1930' ],
    [ 'Rick Mires', '520-589-1970' ] ],
  ,
  ,
  [ [ 'Biff Tanin', '520-589-1970' ] ] ]
*/

//overide example (Phone Number Change)
//
hashT.insert('Rick Mires', '650-589-1970').insert('Tom Bradey', '818-589-
1970').insert('Biff Tanin', '987-589-1970');
//hashT.retrieveAll();

/*
[ ,
  [ [ 'Boo Radley', '520-589-1970' ],
    [ 'Tom Bradey', '818-589-1970' ] ],
  ,
  [ [ 'Alex Hawkins', '510-599-1930' ],
    [ 'Rick Mires', '650-589-1970' ] ],
  ,
  ,
  [ [ 'Biff Tanin', '987-589-1970' ] ] ]

*/

hashT.remove('Rick Mires');
hashT.remove('Tom Bradey');
//hashT.retrieveAll();

/*
[ ,
  [ [ 'Boo Radley', '520-589-1970' ] ],
  ,
  [ [ 'Alex Hawkins', '510-599-1930' ] ],
  ,
  ,
  [ [ 'Biff Tanin', '987-589-1970' ] ] ]


*/

hashT.insert('Dick Mires', '650-589-1970').insert('Lam James', '818-589-
1970').insert('Ricky Ticky Tavi', '987-589-1970');
```

```
hashT.retrieveAll();


/* NOTICE HOW HASH TABLE HAS NOW DOUBLED IN SIZE UPON REACHING 75% CAPACITY ie 6/8. It
is now size 16.
 [,
  ,
  [ [ 'Vance Carter', '120-589-1970' ] ],
  [ [ 'Alex Hawkins', '510-599-1930' ],
    [ 'Dick Mires', '650-589-1970' ],
    [ 'Lam James', '818-589-1970' ] ],
  ,
  ,
  ,
  ,
  ,
  [ [ 'Boo Radley', '520-589-1970' ],
    [ 'Ricky Ticky Tavi', '987-589-1970' ] ],
  ,
  ,
  ,
  ,
  [ [ 'Biff Tanin', '987-589-1970' ] ] ]



*/
console.log(hashT.retrieve('Lam James'));  //818-589-1970
console.log(hashT.retrieve('Dick Mires')); //650-589-1970
console.log(hashT.retrieve('Ricky Ticky Tavi')); //987-589-1970
console.log(hashT.retrieve('Alex Hawkins')); //510-599-1930
console.log(hashT.retrieve('Lebron James')); //null
```

answered Feb 10 '15 at 20:06

Alex Hawkins
556 6 10

3 Looks nice. Now, please also explain WHY this is useful and may be better suited than all the other answers here. – Thomas Tempelmann Feb 4 '16 at 11:02

You can create one using like the following:

1

```
var dictionary = { Name:"Some Programmer", Age:24, Job:"Writing Programs"  };

//Iterate Over using keys
for (var key in dictionary) {
  console.log("Key: " + key + " , " + "Value: "+ dictionary[key]);
}

//access a key using object notation:
console.log("Her Name is: " + dictionary.Name)
```

Run code snippet | Copy snippet to answer | Expand snippet

answered Jul 30 '18 at 10:09

Ali Ezzat Odeh

1,774 1 12 17