Ways to create a Set in JavaScript?

Asked 13 years ago Modified 4 years, 8 months ago Viewed 141k times

98

In Eloquent JavaScript, Chapter 4, a set of values is created by creating an object and storing the values as property names, assigning arbitrary values (e.g. true) as property values. To check if the value is already contained in the set, the `in` operator is used:

```
var set = {};

if (!'Tom' in set) {
  set.Tom = true;
}
```

Is this idiomatic JavaScript? Wouldn't be using an array even better?

```
var set = [];

if (!'Tom' in set) {
  set.push = 'Tom';
}
```

javascript set

Share

Improve this question

Follow

edited Oct 8, 2014 at 13:42

Davi Lima

771 ● 1 ● 6 ● 20

asked Dec 2, 2011 at 22:35

helpermethod

62k ● 71 ● 196 ● 280

1 How would you expect an array for which `'Tom' in set` is true to look like? It looks a lot like you have wrong assumptions about *something*, and I'm trying to find out *about what*. – user395760 Dec 2, 2011 at 22:38

10 FYI, you need parens: `if(!('Tom' in set))`. Currently it means `false in set` since `!'Tom' === false`. – pimvdb Dec 2, 2011 at 22:41

ES6 has sets, see John's answer below – Ben Taliadoros Mar 7, 2016 at 16:14

9 Answers

Sorted by: Highest score (default)

Sets are now available in ES2015 (aka ES6, i.e. ECMAScript 6). ES6 has been the current standard for JavaScript since June 2015.

126

> ECMAScript 6 has the data structure Set which works for arbitrary values, is fast and handles NaN correctly. -Axel Rauschmayer, Exploring ES6

First two examples from Axel Rauschmayer's book Exploring ES6:

> Managing single elements:

```
> let set = new Set();
> set.add('red')

> set.has('red')
true
> set.delete('red')
true
> set.has('red')
false
```

> Determining the size of a Set and clearing it:

```
> let set = new Set();
> set.add('red')
> set.add('green')

> set.size
2
> set.clear();
> set.size
0
```

I would check out Exploring ES6 if you want to learn more about Sets in JavaScript. The book is free to read online, but if you would like to support the author Dr. Axel Rauschmayer you can purchase the book for around $30.

If you want to use Sets and ES6 now you can use Babel, the ES6 to ES5 transpiler, and its polyfills.

Edit: As of June 6th, 2017 most of the major browsers have full Set support in their latest versions (except IE 11). This means you may not need babel if you don't care to support older browsers. If you want to see compatibility in different browsers including your current browser check Kangax's ES6 compatibility table.

EDIT:

Just clarification on initialization. Sets can take any synchronous iterable in their constructor. This means they can take not just arrays but also strings, and iterators.

Take for example the following array and string initialization of a set:

```
const set1 = new Set(['a','a','b','b','c','c']);
console.log(...set1);
console.log(set1.size);
const set2 = new Set("aabbcc");
console.log(...set2);
console.log(set2.size);
```

Run code snippet Expand snippet

Both outputs of the array and string are the same. Note that `...set1` is the spread syntax. It appears that each element of the iterable is added one by one to the set, so since both the array and string have the same elements and since the elements are in the same order the set is created the same. Another thing to note about sets is when iterating over them the iteration order follows the order that the elements were inserted into the set. Here's an example of iterating over a set:

```
const set1 = new Set(['a','a','b','b','c','c']);
for(const element of set1) {
  console.log(element);
}
```

Run code snippet Expand snippet

Since you can use any iterable to initialize a set you could even use a iterator from a generator function. Here is two such examples of iterator initializations that produce the same output:

```
// a simple generator example
function* getLetters1 () {
  yield 'a';
  yield 'a';
  yield 'b';
  yield 'b';
  yield 'c';
  yield 'c';
}

// a somewhat more commonplace generator example
// with the same output as getLetters1.
function* getLetters2 (letters, repeatTimes) {
  for(const letter of letters) {
    for(let i = 0; i < repeatTimes; ++i) {
      yield letter;
```

```
    }
  }
}

console.log("------ getLetters1 ------");
console.log(...getLetters1());
const set3 = new Set(getLetters1());
console.log(...set3);
console.log(set3.size);

console.log("------ getLetters2 ------");
console.log(...getLetters2('abc', 2));
const set4 = new Set(getLetters2('abc', 2));
console.log(...set4);
console.log(set4.size);
```

Run code snippet Expand snippet

These examples' generator functions could just be written to not repeat, but if the generator function is more complicated and as long as the following doesn't impact performance too negatively you could use the Set method to help get only values from a generator that don't repeat.

If you want to know more about sets without reading Dr. Rauschmayer's chapter of his book you can check out the MDN docs on Set. MDN also has more examples of iterating over a set such as using `forEach` and using the `.keys`, `.values`, and `.entries` methods. MDN also has examples such as set union, set intersection, set difference, symmetric set difference, and set superset checking. Hopefully most of those operations will become available in JavaScript without needing to build your own functions supporting them. In fact, there is this TC39 proposal for new Set methods which should hopefully add the following methods to Set in JavaScript at some future point in time if the proposal reaches stage 4:

> - Set.prototype.intersection(iterable) - method creates new Set instance by set intersection operation.
> - Set.prototype.union(iterable) - method creates new Set instance by set union operation.
> - Set.prototype.difference(iterable) - method creates new Set without elements present in iterable.
> - Set.prototype.symmetricDifference(iterable) - returns Set of elements found only in either this or in iterable.
> - Set.prototype.isSubsetOf(iterable)
> - Set.prototype.isDisjointFrom(iterable)

- Set.prototype.isSupersetOf(iterable)

Share Improve this answer Follow

edited Oct 1, 2019 at 19:59

answered Jun 17, 2015 at 21:34

John
7,304 ● 2 ● 40 ● 58

Also Set's themselves are iterable, so you can initialize sets with other sets to make a copy or do something like new Set([...setA, ...setB]) for a union operation. – John Oct 1, 2019 at 19:52

1 @LanceKind I added some additional information including showing that you can initialize sets with any iterable not just arrays. – John Oct 1, 2019 at 20:01

sets have a max size – Dave Ankin May 22 at 12:55

@DaveAnkin I didn't know this, but apparently it's implementation specific and not layed out that way in the actual spec stackoverflow.com/a/72809580/2066736 – John May 23 at 17:39

32

I use dict objects as sets. This works with strings and numbers, but I suppose would cause problems if you wanted to have a set of objects using custom equality and comparison operators:

Creating a set:

```
var example_set =
{
    'a':true,
    'b':true,
    'c':true
}
```

Testing for inclusion in a set

```
if( example_set['a'] ){
    alert('"a" is in set');
}
```

Adding an element to a set

```
example_set['d'] = true;
```

Removing an element from a set

`delete example_set['a'];`

Share Improve this answer Follow

answered Feb 26, 2012 at 23:48

Chris Dutrow
50.3k ● 67 ● 195 ● 262

1 The code I was working on was using an older version of the engine which did not have support for `Set`. This helped. – Abhijith Madhav Dec 29, 2016 at 14:01

If you used TypeScript, what type would you use for `example_set`? – PlsWork Feb 23, 2022 at 20:53

@PlsWork `Record<string, true>` is the best way to go I believe – Jiri Kapoun Dec 21, 2022 at 9:43

16

Sets do not allow duplicate entries and don't typically guarantee predefined ordering. Arrays do both of these, thus violating what it means to be a set (unless you do additional checks).

Share Improve this answer Follow

answered Dec 2, 2011 at 22:38

Jon Newmuis
26.4k ● 3 ● 47 ● 57

2 Array doesn't filter duplicate entries... for example arr.push({id:1, name: "Jake"}) like as NSSet in Objective-C :) – iTux Dec 14, 2015 at 16:43

If you take it mot-a-mot, no. But you can just use the id as the array (map) key. `arr[id] = {"name": "Jake"};` – Buffalo May 4, 2017 at 6:39

11

The first way is idiomatic JavaScript.

Any time you want to store a key/value pair, you must use a JavaScript object. As for arrays, there are several problems:

1. The index is a numerical value.
2. No easy way to check to see if a value is in an array without looping through.
3. A set doesn't allow duplicates. An array does.

Share Improve this answer Follow

answered Dec 2, 2011 at 22:39

Justin Niessner
245k ● 40 ● 414 ● 544

What would be best to assign to the property value? – helpermethod Dec 2, 2011 at 22:50

1 It is possible to remove duplicate elements from JavaScript arrays. stackoverflow.com/a/12166248/975097 – Anderson Green Dec 31, 2012 at 0:54

10

If you want to create a set from an array, simply do:

```
let arr = [1, 1, 2, 1, 3];
let mySet = new Set(arr); // Set { 1, 2, 3 }
```

This is a sugar syntax that I quite fancied when programming in Python, so glad that ES6 finally made it possible to do the same thing.

NOTE: then I realize what I said didn't directly answer your question. The reason you have this "hack" in ES5 is because lookup time in an object by keys is significantly faster (O(1)) than in an array (O(n)). In performance critical applications, you can sacrifice this bit of readability or intuition for better performance.

But hey, welcome to 2017, where you can use proper **Set** in all major modern browsers now!

Share

Improve this answer

Follow

edited Feb 20, 2018 at 20:42

answered Oct 10, 2017 at 11:47

benjaminz

3,228 ● 3 ● 38 ● 51

8

Sets in `ES6`/`ES2015`:

`ES6`/`ES2015` now has built in sets. A set is data structure which allows storage of unique values of any type, whether this are primitive values or object references. A set can be declared using the `ES6` built in set constructor in the following manner:

```
const set = new Set([1, 2, 3, 4, 5]);
```

When creating a set using the Set constructor our newly created set object inherits from the `Set.prototype`. This has all sorts of auxiliary methods and properties. This allows you to easily do the following things:

Example:

```
const set = new Set([1, 2, 3, 4, 5]);

// checkout the size of the set
console.log('size is: ' + set.size);

// has method returns a boolean, true if the item is in the set
console.log(set.has(1));

// add a number
set.add(6);

// delete a number
set.delete(1);

// iterate over each element using a callback
set.forEach((el) => {
  console.log(el);
});

// remove all the entries from the set
set.clear();
```

Run code snippet Expand snippet

Browser compatibility:

All major browser now fully support sets except IE where some features are missing. For exact reference please refer to the mdn docs.

Share Improve this answer Follow

answered Oct 24, 2018 at 12:54

Willem van der Veen

36.4k ● 18 ● 204 ● 176

@Velojet: Ok, fair enough. – kjhughes Mar 7, 2019 at 12:31

3

There are two problems with using bare javascript objects to emulate sets: first, an object can have an inherited property which would screw the "in" operator and second, you can only store scalar values in this way, making a set of objects is not possible. Therefore, a realistic implementation of Sets should provide methods `add` and `contains` instead of plain `in` and property assignments.

Share Improve this answer Follow

answered Dec 3, 2011 at 0:24

georg

215k ● 56 ● 321 ● 397

@kojiro: objects are converted to strings when used as keys: `set={};set[{x:1}]=123;alert(set[{z:99}])` – georg Dec 25, 2011 at 9:48

3

You can try Buckets, is a javascript data structure library and has everything you need to manipulate sets.

Share Improve this answer Follow

answered Feb 2, 2012 at 20:28

Daniel

331 ● 3 ● 3

do it provide an addAll(array) method? – jorrebor Nov 8, 2013 at 12:42

1

Basic creation and usage of Set object 🔷

```
let mySet = new Set()

mySet.add(2)         // Set {2}
mySet.add(7)         // Set {2, 7}
mySet.add(7)         // Set {2, 7}
mySet.add('my text') // Set {2, 7, 'my text'}
let myObj = { a: 1, b: 2 }
mySet.add(myObj)     // Set {2, 7, 'my text', {...}}
mySet.has(2)         // true
mySet.has(myObj)     // true
mySet.size           // 4
```

Iteration

```
for (let item of mySet) console.log(item)  // 2, 7, 'my text', {a:1, b:2}
mySet.forEach(value => console.log(value)) // 2, 7, 'my text', {a:1, b:2}
```

Convert to array

```
var myArr = Array.from(mySet)              // [2, 7, 'my text', {a:1, b:2}]
```

> ❕ The most distinct feature Set offers is every value in Set object must be unique. So you can not add duplicate values.

Share edited Apr 22, 2020 at 7:36 answered Apr 22, 2020 at 7:13

Improve this answer

Follow

Hasan Sefa Ozalp
7,178 ● 5 ● 36 ● 47