## Map & Set

**Using Object as a Map**

In ES5 and below the only data structure we had to map keys to values was an Object, like so:

let obj = {key: "value", a: 1}

console.log(obj.a); // 1

console.log(obj['key']); // "value"

However it does have a few pitfalls.

### Inherited Objects

Looping over objects with for-in also iterated over the inherited properties as well as the objects own properties, like so:

let base = {a:1,b:2};

let obj = Object.create(base);

obj[c] = 3;

for (prop in obj) console.log(prop)

// a

// b

// c



Object.create

creates a new objects who’s

*prototype*

points to the passed in

base

object. If it can’t find a requested property on

obj

, javascript then tries to search

the

base

object for the same property.

Perhaps this is the behaviour you want? Or perhaps you only want to print out the keys that belong to the current object?

With ES5 JavaScript to ensure you were looking at a property of the current object we need to use the hasOwnProperty function, like so:

let base = {a:1,b:2};

let obj = Object.create(base);

obj[c] = 3;

for (prop in obj) {

if (obj.hasOwnProperty(prop)) {

console.log(prop)

}

}

// c

### Overriding Functions

If we are using Object as a dictionary then we could theoretically store a key of hasOwnProperty which then leads to the code in the example above failing, like so:

let obj = {hasOwnProperty: 1};

obj.hasOwnProperty("test");

// TypeError: Property 'hasOwnProperty' is not a function

### proto property

proto holds a special meaning with respect to an objects prototype chain so we can’t use it as the name of a key.

let base = {\_\_proto\_\_:1,b:2};

for (prop in obj) console.log(prop)

// b

**Map**

Map is a new data structure introduced in ES6 which lets you map keys to values without the drawbacks of using Objects.

### Creating, getting and setting

We create a map using the new keyword, like so

let map = new Map();

We can then add entries by using the set method, like so:

let map = new Map();

map.set("A",1);

map.set("B",2);

map.set("C",3);

The set method is also chainable, like so:

let map = new Map()

.set("A",1)

.set("B",2)

.set("C",3);

Or we could initialise the Map with a an array of key-value pairs, like so:

let map = new Map([

[ "A", 1 ],

[ "B", 2 ],

[ "C", 3 ]

])

;

We can extract a value by using the get method:

map.get("A");

// 1

We can check to see if a key is present using the has method:

map.has("A");

// true

We can delete entries using the delete method:

map.delete("A");

// true

We can check for the size (number of entries) using the size property:

map.size

// 2

We can empty an entire Map by using the clear method:

map.clear()

map.size

// 0

### Looping over a Map

We use the for-of looping operator to loop over entries in a Map.

There are a couple of different method we can employ, we’ll go over each one using the below map as the example:

let map = new Map([

[ "APPLE", 1 ],

[ "ORANGE", 2 ],

[ "MANGO", 3 ]

])

;

**Using keys()**

The keys method returns the keys in the map as an array which we can loop over using for-of like so:

for (let key of map.keys()) {

console.log(key);

}

// APPLE

// ORANGE

// MANGO

**Using values()**

The values method returns the values in the map as an array which we can loop over using for-of like so:

for (let value of map.values()) {

console.log(value);

}

// 1:

// 2

// 3

**Using entries()**

The entries method returns the [key,value] pairs in the map as an array which we can loop over using for-of like so:

for (let entry of map.entries()) {

console.log(entry[0], entry[1]);

}

// "APPLE" 1

// "ORANGE" 2

// "MANGO" 3

Using *destructuring* we can access the keys and values directly, like so:

for (let [key, value] of map.entries()) {

console.log(key, value);

}

// "APPLE" 1

// "ORANGE" 2

// "MANGO" 3

Looping over key-value pairs via entries is so common that this is the default for a Map.

Therefore we don’t even need to call entries() on a map instance, like so:

for (let [key, value] of map) {

console.log(key, value);

}

// "APPLE" 1

// "ORANGE" 2

// "MANGO" 3



A distinction between Object and Map is that Maps record the

*order in which*

*elements are inserted*

. It then replays that order when looping over keys, values or

entries.

**Set**

Sets are a bit like maps but they only store keys not key-value pairs.

They are common in other programming languages but are a new addition to JavaScript in ES6.

### Creating, getting and setting

We create a Set using the new keyword, like so

let set = new Set();

We can then add entries by using the add method, like so:

let set = new Set();

set.add('APPLE');

set.add('ORANGE');

set.add('MANGO');

The

add

method is

*chainable*

, like so:

let set = new Set()

.add('APPLE')

.add('ORANGE')

.add('MANGO');

Or we can initialise the Set with an array, like so:

let set = new Set(['APPLE', 'ORANGE', 'MANGO']);

We can check to see if a value is in a set like so:

set.has('APPLE')

// true

We can delete a value from the set:

set.delete('APPLE')

We can count the number of entries in the set like so:

set.size

// 2

We can empty the entire set with the clear method:

set.clear();

set.size

// 0

Sets can only store *unique* values, so adding a value a second time has no effect:

let set = new Set();

set.add('Moo');

set.size

// 1

set.add('Moo');

set.size

// 1

### Looping over a Set

We can use the for-of loop to loop over items in our set, like so:

let set = new Set(['APPLE', 'ORANGE', 'MANGO']);

for (let entry of set) {

console.log(entry);

}

// APPLE

// ORANGE

// MANGO



Similar to Maps, Sets also record the

*order in which elements are inserted*

, it then

replays that order when looping.

**Summary**

Map and Set are great additions to JavaScript in ES6.

We no longer have to deal with Map and Sets *poor cousin* the Object and it’s many drawbacks.

**Listing**

<http://plnkr.co/edit/IXnaorsOWB1XuaePtyf4?p=preview>

*script.js*

'use strict';

// Map

let map = new Map();

map.set("A", 1);

map.set("B", 2);

map.set("C", 3);

let map2 = new Map()

.set("A", 1)

.set("B", 2)

.set("C", 3);

let map3 = new Map([

["A", 1],

["B", 2],

["C", 3]

])

;

for (let [key, value] of map) {

console.log(key, value);

}

console.log(map.get("A"));

console.log(map.has("A"));

console.log(map.size);

map.delete("A");

console.log(map.size);

map.clear();

console.log(map.size);

// Set

let set = new Set();

set.add('APPLE');

set.add('ORANGE');

set.add('MANGO');

let set2 = new Set()

.add('APPLE')

.add('ORANGE')

.add('MANGO');

let set3 = new Set(['APPLE', 'ORANGE', 'MANGO']);

console.log(set.has('APPLE'));

set.delete('APPLE');

console.log(set.size);

set.clear();

console.log(set.size);

let set4 = new Set();

set3.add('Moo');

console.log(set3.size);

// 1

set4.add('Moo');

console.log(set4.size);

// 1

for (let entry of set2) {

console.log(entry);

}