Map

Esta traducción está incompleta. Por favor, ayuda a traducir este artículo del inglés.

This is a new technology, part of the ECMAScript 2015 (ES6) standard.
This technology's specification has been finalized, but check the compatibility table for usage and implementation status in various browsers.

Resumen

El objeto `Map` es un sencillo mapa clave/valor. Cualquier valor (tanto objetos como valores primitivos) pueden ser usados como clave o valor.

Sintaxis

```
new Map([iterable])
```

Parámetros

`iterable`
: Iterable es un array o cualquier otro objeto iterable cuyos elementos son pares clave-valor (arrays de 2 elementos). Cada par clave-valor será agregado al nuevo Map.

Descripción

Un objeto Map puede iterar sobre sus elementos en orden de inserción. Un bucle `for..of` devolverá un array de `[clave, valor]` en cada iteración.

Cabe notar que un Map es un mapa de un objeto, especialmente un diccionario de diccionarios, will only map to the object's insertion order -- el cual es aleatoreo y no ordenado.

Igualdad de claves

La igualdad de claves esta basada en el algoritmo "same-value": `NaN` es considerado lo mismo que `NaN` (sin importar que `NaN !== NaN`) y todos los otros operadores son considerados iguales de acuerdo a la semantica del operador ===. En las primeras versiones de ECMAScript 6

`-0` y `+0` eran considarados distintos (even though `-0 === +0`), esto fue cambiado en posteriores versiones y ha sido implementado en Gecko 29 (Firefox 29 / Thunderbird 29 / SeaMonkey 2.26) (bug 952870) and a recent nightly Chrome.

Objetos y mapas comparados

Los `Objetos` son similares a los `Mapas` en cuanto a que ambos le permiten establecer claves a valores, recuperar dichos valores, eliminar claves, y detectar si existe algo almacenado en una clave determinada. Por esto, los `Objetos` han sido usados históricamente como `Mapas` historically; no obstante, hay diferencias importantes entre `Objetos` y `Mapas` que hacen mejor usar un `Mapa`.

- Un `Objeto` tiene un prototipo, de modo que hay claves por defecto en el mapa. No obstante, esto puede ser sorteado usando `map = Object.create(null)`.
- Las claves de un `Objeto` son `Strings`, mientras que pueden ser valores de cualquier tipo para un `Mapa`.
- Puede obtenerse fácilmente el tamaño de un `Mapa` mientras que debe mantenerse manualmente el trazado del tamaño de un `Objeto`.

Esto no significa que que deban usarse siempre Mapas en todas partes, los Objetos siguen siendo usados en muchos casos. Las instancias de Map son sólo útiles para colecciones, y debería considerar adaptar su código previo donde haya utilizado objetos para ello. Objetos serŕían usados como registros, con campos y métodos. Si no está seguro de cual de ellos usar, hágase las siguientes preguntas:

- ¿Son usualmente las claves desconocidas hasta el tiempo de ejecución? ¿Necesita localizarlas dinámicamente?
- ¿TIenen todos los valores el mismo tipo, y pueden usarse de forma intercambiable?
- ¿Necesita claves que no sean strings?
- ¿Serán añadidos o eliminados a menudo los pares clave-valor?
- ¿Tiene una cantidad arbitraria (fácilmente modificable) de pares clave-valor?
- ¿Será iterada la colección?

Todos los anteriors son indicios de que usted necesita un Mapa para una colección. Si por el contrario, tiene un número fijo de claves, opera sobre cada una de ellas individualmente y distingue entre ellas para su uso, entonces usted requiere un Objeto.

Propiedades

`Map.length`
El valor de la propiedad de longitud es 0

`Map.prototype`
Representa el prototipo para el constructorde Map. Permite la adición de propiedades a todos los objetos Map.

Instancias de Map

Todas las instancias de Map heredan de `Map.prototype`.

Propiedades

`Map.prototype.constructor`
Returns the function that created an instance's prototype. This is the `Map` function by default.

`Map.prototype.size`
Returns the number of key/value pairs in the `Map` object.

Métodos

`Map.prototype.clear()`
Removes all key/value pairs from the `Map` object.

`Map.prototype.delete(key)`
Removes any value associated to the `key` and returns the value that `Map.prototype.has(key)` would have previously returned. `Map.prototype.has(key)` will return `false` afterwards.

`Map.prototype.entries()`
Returns a new `Iterator` object that contains **an array of `[key, value]`** for each element in the `Map` object in insertion order.

`Map.prototype.forEach(callbackFn[, thisArg])`
Calls callbackFn once for each key-value pair present in the `Map` object, in insertion order. If a thisArg parameter is provided to forEach, it will be used as the this value for each callback.

`Map.prototype.get(key)`
Returns the value associated to the `key`, or `undefined` if there is none.

`Map.prototype.has(key)`
Returns a boolean asserting whether a value has been associated to the `key` in the `Map` object or not.

`Map.prototype.keys()`
Returns a new `Iterator` object that contains the **keys** for each element in the `Map` object in insertion order.

`Map.prototype.set(key, value)`
Sets the value for the `key` in the `Map` object. Returns the `Map` object.

`Map.prototype.values()`
Returns a new `Iterator` object that contains the **values** for each element in the `Map` object in insertion order.

`Map.prototype[@@iterator]()`
Returns a new `Iterator` object that contains **an array of `[key, value]`** for each element in the `Map` object in insertion order.

Ejemplos

Ejemplo: Usando el objeto Map

```
var myMap = new Map();

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

// setting the values
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

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

myMap.get("a string");   // "value associated with 'a string'"
                         // because keyString === 'a string'
myMap.get({});           // undefined, because keyObj !== {}
myMap.get(function() {}) // undefined, because keyFunc !== function () {}
```

Ejemplo: Usando `NaN` como claves de Map

`NaN` también puede ser usado como una clave. Aún cuando cada clave `NaN` no es igual a sí misma (`NaN !== NaN` es verdadera), el siguiente ejemplo funciona, porque las claves NaNs `NaN`s no son distinguibles unas de otras:

```
var myMap = new Map();
myMap.set(NaN, "not a number");

myMap.get(NaN); // "not a number"

var otherNaN = Number("foo");
myMap.get(otherNaN); // "not a number"
```

Ejemplo: Iterando `Map` con `for..of`

Los Map pueden ser iterados usando un bucle `for..of`:

```
var myMap = new Map();
myMap.set(0, "zero");
myMap.set(1, "one");
for (var [key, value] of myMap) {
  alert(key + " = " + value);
}
// Mostrará 2 alertas; primero con "0 = zero" y segundo con "1 = one"

for (var key of myMap.keys()) {
  alert(key);
}
// Will show 2 alerts; first with "0" and second with "1"

for (var value of myMap.values()) {
  alert(value);
}
// Will show 2 alerts; first with "zero" and second with "one"

for (var [key, value] of myMap.entries()) {
  alert(key + " = " + value);
}
// Will show 2 alerts; first with "0 = zero" and second with "1 = one"

myMap.forEach(function(value, key, myMap) {
```

```
myMap.forEach(function(value, key, myMap) {
  alert(key + " = " + value);
})
// Will show 2 alerts; first with "0 = zero" and second with "1 = one"
```

Ejemplo: Relación con los objetos `Array`

```
var kvArray = [["key1", "value1"], ["key2", "value2"]];

// Use the regular Map constructor to transform a 2D key-value Array into
var myMap = new Map(kvArray);

myMap.get("key1"); // returns "value1"

// Use the spread operator to transform a map into a 2D key-value Array.
alert(uneval([...myMap])); // Will show you exactly the same Array as kvAr

// Or use the spread operator on the keys or values iterator to get
// an array of only the keys or values
alert(uneval([...myMap.keys()])); // Will show ["key1", "key2"]
```

Especificaciones

Especificación	Estado	Comentario
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Map' in that specification.	ST Standard	Definición inicial.

Compatibilidad con Navegadores

Desktop Mobile

Característica	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Soporte Básico	31 [1] 38	13 (13)	11	No support	7.1
Constructor argument: `new Map(iterable)`	38	13 (13)	No support	No support	No support
iterable	38	17 (17)	No support	No support	7.1
`Map.clear()`	31 [1] 38	19 (19)	11	No support	7.1
`Map.keys()`, `Map.values()`, `Map.entries()`	37 [1] 38	20 (20)	No support	No support	7.1
`Map.forEach()`	36 [1] 38	25 (25)	11	No support	7.1
Key equality for -0 and 0	34 [1] 38	29 (29)	No support	No support	No support

[1] La característica está disponible detrás de una configuración. en `chrome://flags`, activa la entrada “Habilitar JavaScript experimental”.

Ver también

- Map and Set bug at Mozilla
- ECMAScript Harmony proposal
- `Set`
- `WeakMap`
- `WeakSet`