# Map

Bài gốc: Tất cả những gì bạn cần biết về Map và Set trong Javascript

Cho đến bây giờ thì chúng ta đã được biết về các cấu trúc dữ liệu phức tạp dưới đây:

- Object được sử dụng để lưu trữ các value (giá trị) mà định danh bằng key (khóa)
- Array được sử dụng để lưu trữ các giá trị mà được sắp xếp theo thứ tự.

Nhưng liệu rằng chừng đó có đủ cho chúng ta sử dụng khi code trong thực tế. Đó là lý do tại sao Map và Set ra đời để giúp các lập trình viên có thêm nhiều sự lựa chọn hơn khi code.

## Map là gì

`Map` là tập hợp các giá trị được định danh bằng key như `Object`. Nhưng điểm khác biệt chính là `Map` cho phép key thuộc bất cứ kiểu dữ liệu nào.

Các phương thức và thuộc tính của `Map` là

- `new Map()` – tạo mới một map
- `map.set(key, value)` – lưu trữ value bằng key
- `map.get(key)` – return một value bởi một key, `undefined` nếu `key` không tồn tại trong map
- `map.has(key)` – return `true` nếu `key` tồn tại, `false` nếu ngược lại
- `map.delete(key)` – xóa value bằng key
- `map.clear()` – xóa mọi thứ từ map
- `map.size` – return số lượng phần tử hiện tại

Ví dụ:

```
let map = new Map()
map.set('1', 'str1') // string key
map.set(1, 'num1') // numeric key
map.set(true, 'bool1') // boolean key
// Anh em nhớ object thường không? nó sẽ tự động chuyển đổi key sang string
// Map thì giữ nguyên kiểu dữ liệu của key, đó là điểm khác nhau chính của 2 tụi
nó:
alert(map.get(1)) // 'num1'
alert(map.get('1')) // 'str1'
alert(map.size) // 3
```

Như chúng ta có thể thấy, không như object, key không được chuyển sang string. Key có thể thuộc bất cứ kiểu dữ liệu nào.

`map[key]` không phải là cách hay để sử dụng Map

Mặc dù `map[key]` cũng hoạt động. Chúng ta có thể set `map[key] = 2`, điều này giống như "đối xử" với `map` như là một Javascript object thông thường,vì thế bạn sẽ bị một số giới hạn (ví dụ chỉ string/symbol key...)

Vậy nên chúng ta sử dụng các phương thức mà `map` cung cấp như `set`, `get`,...

**Map cũng có thể sử dụng object như là một key.**

Ví dụ:

```
let john = { name: 'John' }
// Đếm số lượng truy cập của mỗi user
let visitsCountMap = new Map()
// john đóng vai trò là key của map
visitsCountMap.set(john, 123)
alert(visitsCountMap.get(john)) // 123
```

Sử dụng object như là key là một trong những tính năng đáng chú ý và quan trọng nhất của `Map`. Điều tương tự không thể xảy ra với `Object`. String là key của `Object` thì ok, nhưng chúng ta không thể xử dụng `Object` như là một key trong `Object`.

Cùng thử xem nào:

```
let john = { name: 'John' }
let ben = { name: 'Ben' }
let visitsCountObj = {} // thử sử dụng một object
visitsCountObj[ben] = 234 // thử sử dụng object như một key
visitsCountObj[john] = 123 // sử dụng john object như một key, ben object sẽ bị
thay thế
// Đây là những gì được ghi!
alert(visitsCountObj['[object Object]']) // 123
```

Vì `visitsCountObj` là một object, nó sẽ chuyển đổi tất cả các Object key, như `john` và ben ở trên thành một string giống nhau là `"[object Object]"` . Đây không phải là những gì chúng ta mong đợi.

**Cách `Map` so sánh các key để biết key đó đã tồn tại hay chưa**

Để kiểm tra các key có bằng nhau hay không, `Map` sử dụng thuật toán SameValueZero . Nó giống như việc so sánh bằng nghiêm ngặt `===`, nhưng điểm khác biệt là `NaN` được coi là bằng với `NaN`.

Thuật toán này không thể bị thay đổi hoặc tinh chỉnh.

**Chaining – chuỗi**

Mọi khi gọi `map.set`, nó return lại chính map đó, vì thế chúng ta có thể nối chuỗi để gọi như thế này:

```
map.set('1', 'str1').set(1, 'num1').set(true, 'bool1')
```

## Lặp qua Map

Để lặp qua một map, có 3 phương thức:

- `map.keys()` – return một iterable cho các key,
- `map.values()` – return một iterable cho các value,
- `map.entries()` – return một iterable cho các entry `[key, value]`, nó được sử dụng mặc định trong `for...of`

Lưu ý đây là một **iterable** chứ không phải **array** nha, ai chưa biết về iter**able thì có thể đọc lại bài viết của mình Bạn đã biết về Iterator, Iterable và Generator trong Javascript chưa?

Ví dụ:

```
let recipeMap = new Map([
  ['cucumber', 500],
  ['tomatoes', 350],
  ['onion', 50]
])
// Lặp qua các key (vegetables)
for (let vegetable of recipeMap.keys()) {
  alert(vegetable) // cucumber, tomatoes, onion
}
// Lặp qua các value (amounts)
for (let amount of recipeMap.values()) {
  alert(amount) // 500, 350, 50
}
// Lặp qua các [key, value] entries
for (let entry of recipeMap) {
  // Tương tự như recipeMap.entries()
  alert(entry) // cucumber,500 (and so on)
}
```

Thứ tự các key khi chèn vào được bảo toàn.

`Map` bảo đảm cho các key khi chèn đúng thứ tự ban đầu, không như Object không phải lúc nào cũng theo thứ tự:

Ví dụ:

```
let recipeMap = new Map([
  ['onion', 50],
  ['cucumber', 500],
  ['tomatoes', 350]
])
const obj = { foo: 'foo', 1: '1', bar: 'bar' }
console.log(Object.keys(obj)) // ["1", "foo", "bar"]
console.log(recipeMap.keys()) // MapIterator {"onion", "cucumber", "tomatoes"}
```

Bên cạnh đó, `Map` cũng có phương thức `forEach` được xây dựng sẵn, tương tự như bới Array:

```
recipeMap.forEach((value, key, map) => {
  alert(`${key}: ${value}`) // cucumber: 500 etc
})
```

## `Object.entries`: Tạo Map từ Object

Khi một `Map` được tạo, chúng ta có thể truyền một array (hoặc một itarable) với cặp key/value cho việc khởi tạo, như thế này:

```
// mảng các cặp [key, value]
let map = new Map([
  ['1', 'str1'],
  [1, 'num1'],
  [true, 'bool1']
])
alert(map.get('1')) // str1
```

Nếu chúng ta có một object thông thường, chúng ta có thể tạo một Map từ nó, sau đó chúng ta có thể sử dụng phương thức có sẵn là Object.entries(obj) để return về một mảng các cặp key/value đúng định dạng.

Vậy nên chúng ta có thể tạo một map từ một object như thế này:

```
let obj = {
  name: 'John',
  age: 30
}
let map = new Map(Object.entries(obj))
alert(map.get('name')) // John
```

Ở đây, `Object.entries` return về mảng các cặp key/value: `[ ["name","John"], ["age", 30] ]` . Đó là những gì chúng ta cần cho `Map`.

## `Object.fromEntries`: Tạo Object từ Map

Chúng ta vừa thấy cách tạo `Map` từ một object thông thường với `Object.entries(obj)`

Phương thức `Object.fromEntries` làm điều ngược lại: đưa vào một mảng của các cặp `[key, value]`, nó tạo một object từ chúng

```
let prices = Object.fromEntries([
  ['banana', 1],
  ['orange', 2],
  ['meat', 4]
])
```

```
// bây giờ prices = { banana: 1, orange: 2, meat: 4 }
alert(prices.orange) // 2
```

Chúng ta có thể sử dụng `Object.fromEntries` để get một object từ `Map`.

Ví dụ trong trường hợp chúng ta lưu trữ data trong `Map`, nhưng chúng ta muốn truyền data đó vào một thư viện hay một hàm nào đó yêu cầu là một object thông thường.

Đây là cách chúng ta làm:

```
let map = new Map()
map.set('banana', 1)
map.set('orange', 2)
map.set('meat', 4)
let obj = Object.fromEntries(map.entries()) // Tạo một object thường (*)
// xong!
// obj = { banana: 1, orange: 2, meat: 4 }
alert(obj.orange) // 2
```

`map.entries()` return một iterable của các cặp key/value, chính xác đúng định dạng cho `Object.fromEntries`.

Chúng ta cũng có thể làm cho dòng `(*)` ngắn hơn

```
let obj = Object.fromEntries(map) // không cần .entries()
```

Cho kết quả giống nhau, bởi vì `Object.fromEntries` mong đợi tham số là iterable object. Không cần thiết là một mảng. Và việc lặp qua `map` sẽ return cặp key/value giống như `map.entries()`