JavaScript Maps And Its Methods

The JavaScript ES6 has introduced two new data structures, i.e Map and WeakMap.

Map is similar to objects in JavaScript that allows us to store elements in a key/value pair.

The elements in a Map are inserted in an insertion order. However, unlike an object, a map can contain objects, functions and other data types as key.

To create a Map, we use the new Map() constructor. For example,

JS script.js

```
// create a Map
const map1 = new Map(); // an empty map
console.log(map1); // Map {}
```

Insert Item to Map

After you create a map, you can use the set() method to insert elements to it. For example,

JS script.js

```
// create a set
let map1 = new Map();

// insert key-value pair
map1.set('info', {name: 'John', age: 22});
console.log(map1); // Map {"info" ⇒ {name: "John", age: 22}}
```

You can also use objects or functions as keys. For example,

JS script.js

```
// Map with object key
let map2 = new Map();

let obj = {};
map2.set(obj, {name: 'John', age: "22"});

console.log(map2); // Map {{} ⇒ {name: "John", age: "22"}}
```

Access & Check Map Elements

You can access Map elements using the get() method. For example,

JS script.js

```
let map1 = new Map();
map1.set('info', {name: 'John', age: "22"});

// access the elements of a Map
console.log(map1.get('info')); // {name: "John", age: "22"}
```

You can use the has() method to check if the element is in a Map. For example,

JS script.js

```
let map1 = new Map();
map1.set('info', {name: 'John', age: "22"});

// check if the element is in a Map
console.log(map1.has('info')); // true
```

Removing Elements

You can use the clear() and the delete() method to remove elements from a Map.

- The delete() method removes only the specified key/value pair from a Map object.
- The clear() method removes all key/value pairs from a Map object. For example,

JS script.js

```
let map1 = new Map();
map1.set('info', {name: 'John', age: "22"});

let map2 = new Map();
map1.set('details', {gender: 'Male'});

// removing a particular element
map1.delete('info');
console.log(map1); // Map(1)

// removing all element
map1.clear();
console.log(map1); // Map(0)
```

Iterate Through a Map

You can iterate through the Map elements using the for...of loop or forEach() method. The elements are accessed in the insertion order. For example,

JS script.js

```
let map1 = new Map();
map1.set('name', 'John');
map1.set('age', '22');

// looping through Map
for (let [key, value] of map1) {
    console.log(key + '- ' + value);
}

// name- John
// age- 22
```

You could also get the same results as the above program using the forEach() method.

Iterate Over Map Keys

You can iterate over the Map and get the key using the keys() method. For example,

JS script.js

```
let map1 = new Map();
map1.set('name', 'John');
map1.set('age', '22');

// looping through the Map
for (let key of map1.keys()) {
  console.log(key)
}

// name
// age
```

Iterate Over Map Values

You can iterate over the Map and get the values using the values() method. For example,

JS script.js

```
let map1 = new Map();
map1.set('name', 'John');
map1.set('age', '22');

// looping through the Map
for (let values of map1.values()) {
    console.log(values);
}

// John
// 22
```

Get Key/Values of Map

You can iterate over the Map and get the key/value of a Map using the entries() method. For example,

JS script.js

```
let map1 = new Map();
map1.set('name', 'John');
map1.set('age', '22');

// looping through the Map
for (let elem of map1.entries()) {
    console.log(`${elem[0]}: ${elem[1]}`);
}

// name: John
// age: 22
```