Save it

Like it

Garbage Collection

DhaneshMane

What is Garbage Collection ?

JavaScript Garbage Collection is a form of memory management whereby objects that are no longer referenced are automatically deleted and their resources are reclaimed.

Weak Collections

- Map and Set's references to objects are strongly held and will not allow for garbage collection.
- WeakMap and WeakSet ES6 collections are 'weak' because they allow for objects which are no longer needed to be cleared from memory.

DhaneshMane

WeakMap

- A WeakMap is a collection of key/value pairs whose **keys must be objects only**. Primitive data types as keys are not allowed
- WeakMap does not support iteration and methods keys(), values(), entries(), so there’s no way to get all keys or values from it.

Methods

- weakMap.get(key)
- weakMap.set(key, value)
- weakMap.delete(key)
- weakMap.has(key)

WeakMap Example

```
const aboutAuthor = new WeakMap( ); // Create New WeakMap
const currentAge = {}; // key must be an object
const currentCity = {}; // keys must be an object

aboutAuthor.set(currentAge, 30); // Set Key Values
aboutAuthor.set(currentCity, 'Denver'); // Key Values can
be of different data types

console.log(aboutAuthor.has(currentCity)); // Test if
WeakMap has a key

aboutAuthor.delete(currentAge); // Delete a key
```

DhaneshMane

UseCases of WeakMap

To keep an object's private data private

```
var Person = (function() {
  var privateData = new WeakMap();

  function Person(name) {
    privateData.set(this, { name: name });
  }

  Person.prototype.getName = function() {
    return privateData.get(this).name;
  };

  return Person;
}());
```

DhaneshMane

To keep track of DOM node edits, removals, and changes

```
_makeClone() {
  this._containerClone = this.container.cloneNode(true);
  this._cloneToNodes = new WeakMap();
  this._nodesToClones = new WeakMap();

  ...

  let n = this.container;
  let c = this._containerClone;

  // find the currentNode's clone
  while (n !== null) {
    if (n === this.currentNode) {
    this._currentNodeClone = c;
    }
    this._cloneToNodes.set(c, n);
    this._nodesToClones.set(n, c);

    n = iterator.nextNode();
    c = cloneIterator.nextNode();
  }
}
```

DhaneshMane

Caching

cache.js

```
let cache = new WeakMap();

// calculate and remember the result
function process(obj) {
  if (!cache.has(obj)) {
    let result = /* calculate the result for */ obj;

    cache.set(obj, result);
  }

  return cache.get(obj);
}
```

main.js

```
let obj = {/* some object */};

let result1 = process(obj);
let result2 = process(obj);

// ...later, when the object is not needed any more:
obj = null;

// Can't get cache.size, as it's a WeakMap,
// but it's 0 or soon be 0
// When obj gets garbage collected, cached data will be
removed as well
```

DhaneshMane

WeakSet

- WeakSet behaves similarly to WeakMap
- Similar to Set, but we can only add objects (not primitive types).
- *An object exists in the set as long as it can be accessed from elsewhere.
- *Like set, it supports add, has, and delete, but not size, keys(), and iterations.

Methods

- weakSet.add(key)
- weakSet.delete(key)
- weakSet.has(key)

DhaneshMane

Example : we are on a page where we are showing Messages and we are showing unread messages as notifications. When a message is deleted, it will automatically be deleted from unread messages.

```
let messages = [
    {text: "Merhaba", from: "Oğuz"},
    {text: "Naber?", from: "Sezer"},
    {text: "Dudu Dudu", from: "Tarkan"}
];
let read = new WeakSet();

read.add(messages[0]);
read.add(messages[1]);
read.add(messages[0]);
read.add(messages[0]);
//A message can be read more than once. But the array will not change

messages.shift();
//When the message is deleted, it is also deleted from the read.
```

DhaneshMane

Follow me for more

Thank you

https://www.linkedin.com/in/dhaneshmane/