VIETNAM AUSTRALIA Vocational College

Map, Set, Parameters, Literals, String/Array Utils

Mentor: Nguyễn Bá Minh Đạo

daynghevietuc.com

Nội dung:

1. Map và các phương thức thường gặp
2. Set và các phương thức thường gặp
3. Default Parameter Values, Function Rest Parameter
4. Template Literals, tiện ích String/Array/Number

Map và các phương thức thường gặp

❑ **Map Object**:

➢ Một **đối tượng Map** giữ các cặp **key**-**value** nơi mà **các key có thể là bất kỳ loại dữ liệu nào**.

➢ Một **đối tượng Map** **ghi nhớ thứ tự chèn ban đầu** của **các key**.

➢ **Đối tượng Map** **có** một **thuộc tính đại diện** cho **kích thước** của **Map**.

Map và các phương thức thường gặp

❑ Map Methods:

Phương thức	Mô tả
new Map()	Khởi tạo một đối tượng Map
set()	Thiết lập giá trị cho một key trong một Map
get()	Lấy giá trị của một key trong một Map
clear()	Xóa tất cả phần tử trong một Map
delete()	Xóa một phần tử của Map thông qua một key
has()	Trả về true nếu một key tồn tại trong một Map

Map và các phương thức thường gặp

❑ Map Methods:

Phương thức	Mô tả
forEach()	Gọi lại cho từng cặp key/value trong một Map
entries()	Trả về một đối tượng vòng lặp (iterator) với các cặp [key, value] trong một Map
keys()	Trả về một đối tượng vòng lặp (iterator) với các key trong một Map
values()	Trả về một đối tượng vòng lặp (iterator) với các value trong một Map

Thuộc tính	Mô tả
size	Trả về số lượng phần tử trong một Map

Map và các phương thức thường gặp

- Có thể **tạo một đối tượng Map bằng 2 cách** sau:
  - Cách 1: Truyền một Array vào lệnh new Map()

```
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);

console.log(fruits);
```

```
Map(3) {'apples' => 500, 'bananas' => 300, 'oranges' => 200}
  [[Entries]]
    0: {"apples" => 500}
    1: {"bananas" => 300}
    2: {"oranges" => 200}
    size: 3
  [[Prototype]]: Map
```

Map và các phương thức thường gặp

- Có thể **tạo một đối tượng Map bằng 2 cách** sau:
  - Cách 2: Tạo một Map và dùng lệnh Map.set()

```
// Create a Map
const fruits = new Map();

// Set Map Values
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);

console.log(fruits);
```

```
Map(3) {'apples' => 500, 'bananas' => 300, 'oranges' => 200}
  [[Entries]]
    0: {"apples" => 500}
    1: {"bananas" => 300}
    2: {"oranges" => 200}
    size: 3
  [[Prototype]]: Map
```

Map và các phương thức thường gặp

- Các phương thức thường gặp của **đối tượng Map**:
  - Map.set() cũng được dùng để thay đổi giá trị hiện tại của Map

```
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);

// Set Map Values
fruits.set("apples", 200);

console.log(fruits);
```

```
Map(3) {'apples' => 200, 'bananas' => 300, 'oranges' => 200}
  [[Entries]]
    0: {"apples" => 200}
    1: {"bananas" => 300}
    2: {"oranges" => 200}
    size: 3
  [[Prototype]]: Map
```

Map và các phương thức thường gặp

- Các phương thức thường gặp của **đối tượng Map**:
  - Map.get() dùng để lấy giá trị hiện tại của Map
  - Map.size() dùng để trả về số lượng phần tử của Map

```
> // Create a Map
  const fruits = new Map([
    ["apples", 500],
    ["bananas", 300],
    ["oranges", 200]
  ]);

  // Get Map Value
  console.log(fruits.get("apples"));

  500
```

```
> // Create a Map
  const fruits = new Map([
    ["apples", 500],
    ["bananas", 300],
    ["oranges", 200]
  ]);

  // Get Map Value
  console.log(fruits.size);

  3
```

Map và các phương thức thường gặp

- Các phương thức thường gặp của **đối tượng Map**:
  - Map.delete() dùng để xóa phần tử hiện tại của Map
  - Map.clear() dùng để xóa tất cả các phần tử của Map

```
> // Create a Map
  const fruits = new Map([
    ["apples", 500],
    ["bananas", 300],
    ["oranges", 200]
  ]);

  // Delete Map Element
  fruits.delete("apples");

  console.log(fruits.size);
  2
```

```
> // Create a Map
  const fruits = new Map([
    ["apples", 500],
    ["bananas", 300],
    ["oranges", 200]
  ]);

  // Delete the Map
  fruits.clear();

  console.log(fruits.size);
  0
```

Map và các phương thức thường gặp

- Các phương thức thường gặp của **đối tượng Map**:
  - Map.has() dùng để trả về **true** nếu tồn tại key trong Map
  - instanceof dùng để trả về **true** nếu đối tượng hiện tại là Map

```
> // Create a Map
  const fruits = new Map([
    ["apples", 500],
    ["bananas", 300],
    ["oranges", 200]
  ]);

  console.log(fruits.has("apples"));

  fruits.delete("apples");
  console.log(fruits.has("apples"));
```

```
true
false
```

```
> // Create a Map
  const fruits = new Map([
    ["apples", 500],
    ["bananas", 300],
    ["oranges", 200]
  ]);

  console.log(fruits instanceof Map);
```

```
true
```

Map và các phương thức thường gặp

❑ **Sự khác nhau** giữa **Object** và **Map** trong JavaScript:

	Object	Map
Iterable (Khả năng lặp lại)	Không thể lặp lại trực tiếp	Có thể lặp lại trực tiếp
Size (Kích thước)	Không có thuộc tính kích thước	Có thuộc tính kích thước
Key Types (Các kiểu của Key)	Các phím phải là String (hoặc Symbol)	Các khóa có thể là bất kỳ kiểu dữ liệu nào
Key Order (Thứ tự của Key)	Chìa khóa không được đặt hàng tốt	Các phím được sắp xếp theo thứ tự chèn
Defaults (Mặc định)	Có khóa mặt định	Không có khóa mặc định

Map và các phương thức thường gặp

- Các phương thức thường gặp của **đối tượng Map**:
  - Map.forEach() dùng để **gọi lại** cho **mỗi cặp key/value** trong một Map
  - Map.keys() dùng để trả về **đối tượng vòng lặp các key** trong một Map

```
> // Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);

let text = "";
fruits.forEach (function(value, key) {
  text += key + ' = ' + value + "\n"
})

console.log(text);
```

```
apples = 500
bananas = 300
oranges = 200
```

```
> // Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);

let veggies = "";
for (const x of fruits.keys()) {
  veggies += x + "\n";
}

console.log(veggies);
```

```
apples
bananas
oranges
```

Map và các phương thức thường gặp

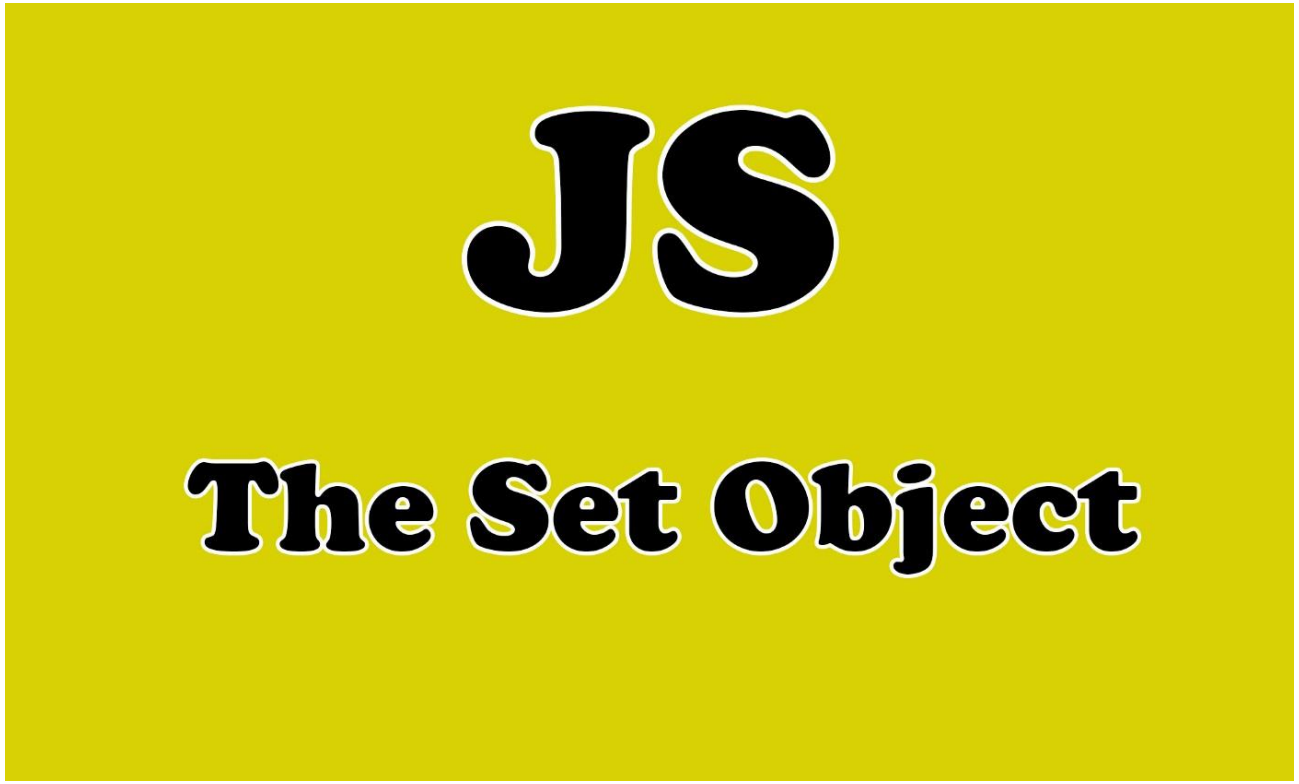
- Phương thức **đối tượng** như là **key** trong **Map**:
  - Dùng objects như là keys là kỹ thuật quan trọng trong tính năng Map

```
> // Create Objects
const apples = {name: 'Apples'};
const bananas = {name: 'Bananas'};
const oranges = {name: 'Oranges'};

// Create a Map
const fruits = new Map();

// Add the Objects to the Map
fruits.set(apples, 500);
fruits.set(bananas, 300);
fruits.set(oranges, 200);

// Get value of apples object key -> right
console.log(fruits.get(apples));

// Get value of "apples" key -> wrong
console.log(fruits.get("apples"));
```

```
500                                                    VM1642:15
```

Set và các phương thức thường gặp

- **Set Object**:
  - Một đối tượng Set **là một tập hợp (collection) các giá trị duy nhất**.
  - Một đối tượng Set có thể **giữ bất kỳ giá trị của bất kỳ kiểu dữ liệu nào**.



Set và các phương thức thường gặp

❑ Set Methods:

Phương thức	Mô tả
new Set()	Khởi tạo một đối tượng Set
add()	Thêm một phần tử mới trong một Set
delete()	Xóa một phần tử của Set
has()	Trả về true nếu một giá trị tồn tại trong Set
clear()	Xóa tất cả phần tử trong Set

Set và các phương thức thường gặp

❑ Set Methods:

Phương thức	Mô tả
forEach()	Gọi lại cho từng cặp key/value trong một Set
values()	Trả về một đối tượng vòng lặp (iterator) với các value trong một Set
keys()	Trả về một đối tượng vòng lặp (iterator) với các key trong một Set
entries()	Trả về một đối tượng vòng lặp (iterator) với các cặp [key, value] trong một Set

Thuộc tính	Mô tả
size	Trả về số lượng phần tử trong một Set

Set và các phương thức thường gặp

- Có thể **tạo một đối tượng Set bằng 3 cách** sau:
  - Cách 1: Truyền một Array vào hàm khởi tạo của new Set()
  - Cách 2: Tạo một Set và thêm các giá trị chuỗi vào Set đó.
  - Cách 3: Tạo một Set và thêm các biến vào Set đó.

```
// Create a Set
const letters = new Set(["a","b","c"]);

// Display set.size
console.log(letters.size);
```

```
3
```

* **Lưu ý**: khi dùng phương thức **add nhiều giá trị giống nhau**, **Set chỉ lưu phần tử đầu tiên**

```
// Create a Set
const letters = new Set();

// Add Values to the Set
letters.add("a");
letters.add("b");
letters.add("c");

// Display set.size
console.log(letters.size);
```

```
3
```

```
// Create a Set
const letters = new Set();

// Create Variables
const a2 = "a";
const b2 = "b";
const c2 = "c";

// Add the Variables to the Set
letters.add(a2);
letters.add(b2);
letters.add(c2);

// Display set.size
console.log(letters.size);
```

```
3
```

Set và các phương thức thường gặp

- Có thể **tạo một đối tượng Set bằng 3 cách** sau:
  - Cách 1: Truyền một Array vào hàm khởi tạo của new Set()
  - Cách 2: Tạo một Set và thêm các giá trị chuỗi vào Set đó.
  - Cách 3: Tạo một Set và thêm các biến vào Set đó.

```
// Create a Set
const letters = new Set(["a","b","c"]);

// Display set.size
console.log(letters.size);
```

```
3
```

* **Lưu ý**: khi dùng phương thức **add nhiều giá trị giống nhau**, **Set chỉ lưu phần tử đầu tiên**

```
// Create a Set
const letters = new Set();

// Add Values to the Set
letters.add("a");
letters.add("b");
letters.add("c");

// Display set.size
console.log(letters.size);
```

```
3
```

```
// Create a Set
const letters = new Set();

// Create Variables
const a2 = "a";
const b2 = "b";
const c2 = "c";

// Add the Variables to the Set
letters.add(a2);
letters.add(b2);
letters.add(c2);

// Display set.size
console.log(letters.size);
```

```
3
```

Set và các phương thức thường gặp

- Các phương thức thường gặp của **đối tượng Set**:
  - Set.forEach() dùng để **gọi lại** cho **mỗi cặp key/value** trong một Set
  - Set.values() dùng để trả về **đối tượng vòng lặp các value** trong một Set

```
// Create a Set
const letters = new Set(["a","b","c"]);

// List all Elements
let text = "";
letters.forEach (function(value) {
  text += value + "\n";
})

console.log(text);
```

```
a
b
c
```

```
// Create a Set
const letters = new Set(["a","b","c"]);

// Display set.size
console.log(letters.values());
```

```
▶ SetIterator {'a', 'b', 'c'}
```

```
// Create a Set
const letters = new Set(["a","b","c"]);

// List all Elements
let text = "";
for (const x of letters.values()) {
  text += x + "\n";
}

console.log(text);
```

```
a
b
c
```

Set và các phương thức thường gặp

- Các phương thức thường gặp của **đối tượng Set**:
  - Set **không có** keys, keys(), entries() **làm cho** Set **tương thích** với Map
  - Set.keys() dùng để **trả về giá trị giống** như phương thức values()
  - Set.entries() **trả về các cặp** [value, value] **thay vì các cặp** [key, value]

```
// Create a Set
const letters = new Set(["a","b","c"]);

// Display set.size
console.log(letters.keys());
```

```
SetIterator {'a', 'b', 'c'}
  [[Entries]]
    0: "a"
    1: "b"
    2: "c"
  [[Prototype]]: Set Iterator
  [[IteratorHasMore]]: true
  [[IteratorIndex]]: 0
  [[IteratorKind]]: "values"
```

```
// Create a Set
const letters = new Set(["a","b","c"]);

// List all entries
const iterator = letters.entries();
let text = "";
for (const entry of iterator) {
  text += entry + "\n";
}

console.log(text);
```

```
a,a
b,b
c,c
```

Set và các phương thức thường gặp

- Các phương thức thường gặp của **đối tượng Set**:
  - typeof dùng để trả về **kiểu object** cho đối tượng Set
  - instanceof dùng để trả về **true** nếu đối tượng hiện tại là Set

```
// Create a new Set
const letters = new Set(["a","b","c"]);

// Display typeof Set
console.log(typeof letters);

// Display instanceof Set
console.log(letters instanceof Set);
```

```
object                                    VM4720:5
true                                      VM4720:8
```

Default Parameter Values, Function Rest Parameter

- **ES6 cho phép các tham số trong hàm có giá trị mặc định:**
- **ES6 cung cấp tham số rest (...) cho phép một hàm coi một số lượng vô hạn đối số là một mảng.**

```
function myFunction(x, y = 10) {
  // y is 10 if not passed or undefined
  return x + y;
}

myFunction(5);
15
```

```
function sum(...args) {
  let sum = 0;
  for (let arg of args) sum += arg;
  return sum;
}

let x = sum(4, 9, 16, 25, 29, 100, 66, 77);

console.log(x);
326
```

Template Literals, tiện ích String/Array/Number

❑ Cú pháp **Back-Ticks**:

➢ **Template Literals** (chuỗi mẫu) thường **sử dụng back-ticks (``) thay vì quotes (“”) để định nghĩa** một **chuỗi**.

```
> let text2 = `Hello world!`;
  console.log(text2);

  Hello world!
```

➢ Với **template literials**, chúng ta **có thể dùng cả nháy đơn, nháy đôi** bên **trong string**.

```
> let text3 = `He's often called "Nguyễn Văn Tèo"`;
  console.log(text3);

  He's often called "Nguyễn Văn Tèo"
```

Template Literals, tiện ích String/Array/Number

- **Biến/biểu thức thay thế** (Variable/Expression Substitution):
  - **Template Literals** cho phép **sử dụng biến** trong **chuỗi**.

```
let firstName2 = "Tèo";
let lastName2 = "Nguyễn";

let fullName2 = `Welcome ${firstName2}, ${lastName2}!`;

console.log(fullName2);
```

```
Welcome Tèo, Nguyễn!
```

  - **Template Literals** cho phép **sử dụng biểu thức** trong **chuỗi**.

```
> let price = 10;
  let VAT = 0.25;
  let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;

  console.log(total);
```

```
  Total: 12.50
```

Template Literals, tiện ích String/Array/Number

- **Mẫu HTML** (HTML Templates):
  - **Template Literals** cho phép **sử dụng HTML** trong **chuỗi**.

```
> let header = "Templates Literals";
  let tags = ["template literals", "javascript", "es6"];

  let html = `<h2>${header}</h2><ul>`;

  for (const x of tags) {
    html += `<li>${x}</li>`;
  }

  html += `</ul>`;
  document.getElementById("demo").innerHTML = html;
<· '<h2>Templates Literals</h2><ul><li>template literals</li><li>javascript</li><li>es6</li></ul>'
```

  - **Templates Literals không được hỗ trợ trong Internet Explorer**.

Template Literals, tiện ích String/Array/Number

❑ Các tiện ích của String/Array/Number:

➢ String.includes() trả về true **nếu** một **chuỗi chứa một giá trị** nào đó người dùng **cần tìm**, **ngược lại** trả về false.

```
> let text = "Hello world, welcome to the universe.";
  console.log(text.includes("world"));

  true
```

➢ String.startWith() trả về true **nếu** một **chuỗi** bắt đầu **bằng một giá trị** nào đó người dùng **cần tìm**, **ngược lại** trả về false.

```
> let text = "Hello world, welcome to the universe.";
  console.log(text.startsWith("Hello"));

  true
```

Template Literals, tiện ích String/Array/Number

❑ Các tiện ích của String/Array/Number:

➢ String.endWith() trả về true **nếu** một **chuỗi** kết thúc **bằng một giá trị** nào đó người dùng **cần tìm**, **ngược lại** trả về false.

```
> let text = "John Doe";
  console.log(text.endsWith("Doe"));

  true
```

➢ Array.from() trả về đối tượng Array từ bất kỳ đối tượng khác hoặc bất kỳ đối tượng vòng lặp nào (iterable object).

```
> const myArr = Array.from("ABCDEFG");
  console.log(myArr);

  ▶ (7) ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

Template Literals, tiện ích String/Array/Number

- Các tiện ích của String/Array/Number:
  - Array.keys() trả về đối tượng Array Iterator với các keys của một mảng.

```
> const fruits = ["Banana", "Orange", "Apple", "Mango"];
  const keys = fruits.keys();

  let fruitKeys = "";
  for (let x of keys) {
    fruitKeys += x + " ";
  }

  console.log(fruitKeys);
```

```
0 1 2 3
```

Template Literals, tiện ích String/Array/Number

❑ Các tiện ích của String/Array/Number:

➢ Array.find() trả về giá trị của phần tử đầu tiên trong mảng phù hợp với giá trị cần tìm.

```
> const numbers = [4, 9, 16, 25, 29];
  let first = numbers.find(myFunction);

  console.log("First number over 18 is " + first);

  function myFunction(value, index, array) {
    return value > 18;
  }
```

```
First number over 18 is 25
```

Template Literals, tiện ích String/Array/Number

❑ Các tiện ích của String/Array/Number:

➢ Array.findIndex() trả về vị trí (index) của phần tử đầu tiên trong mảng phù hợp với giá trị cần tìm.

```
> const numbers = [4, 9, 16, 25, 29];

  console.log("First number over 18 has index " + numbers.findIndex(myFunction));

  function myFunction(value, index, array) {
    return value > 18;
  }
```

```
  First number over 18 has index 3
```

➢ Number.isInteger() trả về true nếu đối số truyền vào là kiểu số nguyên.

```
> Number.isInteger(10) + " - " + Number.isInteger(10.5);
< 'true - false'
```

Template Literals, tiện ích String/Array/Number

❑ Các phương thức mới của ES6 - isFinite(), isNaN():

➢ isFinite() trả về false nếu đối số truyền vào là Infinity không NaN. Ngược lại, trả về true.

```
> isFinite(10 / 0) + " - " + isFinite(10 / 1);
< 'false - true'
```

➢ isNaN() trả về true nếu đối số truyền vào là NaN. Ngược lại, trả về giá trị false.

```
> isNaN("Hello");      // returns true
< true
```

Tổng kết:

- ❑ Map và các phương thức thường gặp
- ❑ Set và các phương thức thường gặp
- ❑ Default Parameter Values, Function Rest Parameter
- ❑ Template Literals, tiện ích String/Array/Number