ES6

Một số mẹo vặt "hay ho" của ES6 có thể bạn chưa biết - Phần 1

Để nói về ES6 thì có rất nhiều điều để nói, có thể nói từ ngày này qua ngày khác, tháng này qua tháng khác...chắc cũng chưa hết . ES6 đã ra đời cách đây 5 năm (vì ra mắt vào năm 2015 nên còn có tên là ES2015), nó thực sự là 1 cuộc cách mạng đúng nghĩa của Javascript, 1 sự đột phá mà sau này hầu hết các JS Framework đều lấy ES6 làm nền tảng, đặt nền móng cho những phiên bản ES7, ES8... tiếp theo. ES6 được bổ sung hàng loạt những tính năng mới, nhiều cú pháp mới quan trọng cho việc viết các ứng dụng phức tạp, bao gồm classes và modules, vân vân và mây mây...

Ngoài những tính năng khá phổ biến mà lâu nay khi tìm hiểu về ES6 mà chúng ta đã biết như `Arrow functions, Classes, Let & Const, Template strings` ... Thì bên cạnh đó còn khá nhiều tips hoặc những tính năng khá "hay ho" khác mà có thể các bạn chưa biết, hoặc đã biết rồi nhưng lại chưa sử dụng tới (sự thật mà nói thì bản thân mình cũng chưa thể nắm hết những tính năng của nó, nói cho oai thế thôi chứ chắc là "không" chứ không phải là "chưa" :v).

1. Định dạng "tiền tệ"

Để định dạng tiền tệ của các quốc gia trên thế giới, ES6 cung cấp cho chúng ta phương thức `Intl.NumberFormat`. Cùng xem qua ví dụ dưới đây:

```
const money = 100000;

new Intl.NumberFormat('en-US', {
  style: 'currency',
  currency: 'USD',
}).format(money); // '$ 100,000'

new Intl.NumberFormat('jp-JP', {
  style: 'currency',
  currency: 'JPY',
}).format(money); // 'JP¥ 100,000'
```

Như ví dụ trên, phương thức `Intl.NumberFormat` sẽ có cú pháp `new Intl.NumberFormat([locales[, options]])`, bao gồm:

- locales: gồm `language code + country code`, 2 thứ này các bạn có thể xem ở đây và ở đây.
- options: có rất nhiều options, nhưng ở đây chỉ để cập đến 2 options đang sử dụng: `styles` và `currency`. Tham số `style` chúng ta có 3 loại là `decimal, currency, percent`, về tham số `currency` các bạn có thể tham khảo ở đây.

2. Remove Duplicates Array

Giả sử chúng ta có 1 array chứa danh sách các phần tử, trong đó có nhiều phần tử bị trùng lặp value với nhau, vậy làm thể nào để chỉ giữ lại 1 phần tử và xóa đi các phần từ bị trùng lặp với nó.

Đương nhiên là bạn sẽ có rất nhiều cách, ES5 cũng có thể làm được...nhưng nó khá lằng nhằng, khó hiểu. Hoặc, bạn có thể dùng

```
lodash
```

chẳng hạn (cách này thì dễ nhất rồi, khỏi lo bug

). Tuy nhiên, có 1 cách cực kỳ đơn giản với ES6 mà lâu nay các bạn có thể đã bỏ qua, hãy cùng xem ví dụ dưới đây:

```
const arrray = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4];

const removeDuplicate = [...new Set(arrray)];

console.log(removeDuplicate);
// [1,2,3,4]
```

Thật đơn giản vô cùng phải không nào!

3. Xử lý chuỗi với Spread

Toán tử `spread` (...) chắc hẳn đã khá quen thuộc với các bạn đã tìm hiểu về ES6. Để tách 1 chuỗi thành 1 array thì cũng khá đơn giản, các bạn xem ví dụ sau:

```
const sun = 'asterisk';

// ES6 Spread
const sun2 = [...sun];
// [ 'a', 's', 't', 'e', 'r', 'i', 's', 'k' ]

// hoặc ES6 Array.from
const sun3 = Array.from(sun);
// [ 'a', 's', 't', 'e', 'r', 'i', 's', 'k' ]
```

Vậy nó có gì đặc biệt! Chúng ta cùng xem qua 1 ví dụ khác để thấy được sự linh hoạt và hiệu quả khi sử dụng toán tử `spread` nhé.

```
const sun = 'asterisk';

const formatName = [
  ...sun[0].toUpperCase(),
  ...sun.slice(1),
].join('');

console.log(formatName); // Asterisk
```

Function trên sẽ format chuỗi với việc chuyển các ký tự đầu tiên thành chữ hoa thay cho chữ thường. Điều này có vẻ khá hữu ích khi xử lý các đoạn string chứa tên riêng đấy nhỉ.

4. Skip value với Destructuring

Thường để tránh các phép gán biến vô dụng cho các giá trị bạn không mong muốn, thông thường chúng ta có 2 cách sau đây:

- Sử dụng 1 space
- Thêm prefix "_"

```
const [, value2] = ['value1', 'value2'];

const [_value1, value2] = ['value1', 'value2'];
```

Hãy xét 1 ví dụ cụ thể để thấy cái sự "hay ho" của Destructuring nhé:

```
const url = 'sun-asterisk/framgia/com/jp';

const array = url.split('/'); // ["sun-asterisk", "framgia", "com", "jp"]

const [domain, , ,type] = array;

const name = `${domain}.${type}`;
// 'sun-asterisk.jp'
```

5. Khai báo Object với Dynamic Keys

Cách dễ dàng hơn để tạo các đối tượng với `Dynamic Keys`

Với phiên bản Javascript trước ES6, chúng ta phải viết kèm `key` tương ứng với `object`. Với ES6, cuối cùng chúng ta có thể tạo `Dynamic Keys` khi khai báo 1 Object!

```
let item = '';

// ES5
let object = {};
lock[item] = 'value';

// ES6
let object = {
  [item]: 'value';
}
```

Phần 2

1. Kiểm tra 1 biến có giá trị NaN

Javscript có một điều thật kỳ lạ, NaN là giá trị duy nhất KHÔNG BAO GIỜ bằng chính nó. Vậy làm thế nào để chúng ta kiểm tra tính đúng sai của nó?

```
const value = NaN;

value === NaN;
// false
```

Đừng lo, ES6 đã giới thiệu một phương pháp mới giải quyết vấn đề này là `Object.is`. Hãy xem nó hoạt động như thế nào!

```
const divide = 5 / "two"; //NaN

divide === NaN;

Object.is(divide, NaN); // true
```

2. Tính toán với kiểu dữ liệu string

Nếu giả sử bạn nhận được dữ liệu trả về bao gồm cả số và chuỗi, nhưng vì 1 lý do nào đó mà bạn...lười không muốn mất công convert string qua number rồi mới thực hiện việc tính toán, vậy thì áp dụng ngay "mẹo" dưới đây xem sao

```
const string = '100';
const number = 5;

// Nếu thực hiện tính tổng thông thường
console.log(string + number); // 1005

// Hack 1 chút với phương thức +string
console.log(+string + number); // 105
```

3. Kiểm tra sự tồn tại của subString trong String

Ở ES5, để kiểm tra sự tồn tại của string trong string, hoặc item trong array, chúng ta thường dùng cú pháp `indexOf`.

Nhưng từ phiên bản ES6, bạn có thể sử dụng phương thức `includes()` để kiểm tra xem một chuỗi có chứa một chuỗi con hay không. Nó sẽ xác định xem chuỗi đó có tồn tại trong một chuỗi khác hay không, kết quả trả về sẽ là `true` hoặc `false`.

```
const company = 'sun asterisk';

// ES5
company.indexOf('sun') !== -1; // true

// ES6
company.includes('sun'); // true
```

hoặc

```
const company = ['sun', 'asterisk', 'framgia'];

company.includes('sun'); // true
```

4. Remove Duplicates Array

Ở phần trước, mình đã giới thiệu qua 1 phương thức để loại bỏ các phần tử trùng lặp trong 1 mảng, đó là dùng cú pháp `...`. Ngoài ra, còn có thêm 1 phương thức nữa của ES6 để làm việc này.

```
const arrray = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4];

const removeDuplicate = Array.from(new Set(arrray));

console.log(removeDuplicate);
// [1,2,3,4]
```

Phần 3

1. Truyền Array vào function dưới dạng đối số

Nếu bạn muốn truyền một Array vào một function. Bạn có thể sử dụng ES6 spread để biến mảng đó thành một danh sách các đối số.

```
const array = ['Sun', 'Asterisk', 'Viet Nam'];

function company(name1, name2, name3) {
  console.log(name1); // 'Sun'
  console.log(name2); // 'Asterisk'
  console.log(name3); // 'Viet Nam'
}

// ES5
company.apply(null, array);

// ES6
company(...array);
```

Thoạt nhìn qua thì có vẻ như không có gì đặc biệt cả, chỉ là truyền 1 array vào funtion bằng cú pháp `...` của ES6 thôi, hmm! Vậy chúng ta cùng xem xét thêm 1 ví dụ nữa xem nó hiệu quả như thế nào.

```
const numbers = [5, 7, 3];

// Cách này thì work nhưng nó rất là...củ chuối
Math.max(numbers[0], numbers[1], numbers[2]);

// Cách này thì lại không work, hmmm
Math.max(numbers); // NaN

// Đừng lo, ES6 đã ở đây
Math.max(...numbers); // 7
```

2. Làm tròn số thập phân thành số nguyên

Để làm tròn chữ số thập phân thành số nguyên, có thể các bạn đều đã biết tới cách dùng hàm `parseInt`. Tuy nhiên, `parseInt` có 1 điểm trừ đó là...

```
const number = 1600000000000000000000.8;

const result = parseInt(number);

console.log(result); // 1
```

...ôi không, tại sao lại là

```
1
```

, kết quả mong đợi của chúng ta phải là

```
1.6+e21
```

. (Nếu các bạn chưa hiểu

```
e
```

biểu thị giá trị bao nhiêu trong Javascript thì các bạn có thể xem qua bài viết về Decimal number của mình ở đây nhé)

Bởi vì khi số có giá trị quá lớn và được convert qua giá trị mới chứa chuỗi (ký tự `.`, `+`, `e`) thì hàm `parseInt` sẽ chỉ lấy ký tự đầu tiên mà thôi. Vì vậy, để hàm `parseInt` hoạt động chính xác, đầu tiên chúng ta cần convert kiểu dữ liệu number qua string rồi mới thực hiện làm tròn.

```
const number = 1600000000000000000000.8;

const result = number.toString();
```

```
console.log(result); // "1.6e+21"
```

Và để giải quyết sự "rối rắm" trên, ES6 đã cung cấp cho chúng ta 1 hàm để làm việc này mà không cần phải quan tâm đến giá trị đầu vào của number là bao nhiêu.

```
const number = 1600000000000000000000.8;

const result = Math.trunc(number);

console.log(result); // "1.6e+21"
```

3. Kiểm tra kiểu dữ liệu của Array

Trong JavaScript, Array thật sự không phải là...Array, bản chất của Array là Object. Vì vậy, bạn không thể chỉ đơn giản là dùng `typeof` để kiểm tra tính đúng đắn của kiểu dữ liệu Array,

```
const array = [1, 2, 3, 4];

typeof array; // 'object'
```

Nhưng đừng lo lắng! ES6 cung cấp cho chúng ta 1 function đó là `Array.isArray()`, giúp dễ dàng hơn để kiểm tra xem một giá trị có phải là một Array hay không.

```
const array1 = [];

Array.isArray(array1); // true

const array2 = [1, 2, 3, 4];

Array.isArray(array2); // true
```

hoặc

```
// Object
Array.isArray({}); // false

// Object
Array.isArray({ name: 'Sun' }); // false

// Number
Array.isArray(1000); // false

// Boolean
Array.isArray(true); // false
```

Phần 4

1. Generate 1 random string

Khi muốn tạo 1 chuỗi các ký tự ngẫu nhiên (vì mục đích, lý do bảo mật nào đó), thì không rõ các bạn đã và đang dùng thuật toán hay library nào, có thể là `cryptoJS` chẳng hạn. Nhưng theo mình có 1 cách đơn giản, mà mình đã áp dụng nhiều trong các project cá nhân từ trước đến nay và cảm thấy khá an toàn. Ví dụ dưới đây sẽ tạo 1 chuỗi ngẫu nhiên với 10 ký tự:

```
Math.random().toString(36).substr(2, 10);
"y99yu80obg"
Math.random().toString(36).substr(2, 10);
"oo4a5967xj"
Math.random().toString(36).substr(2, 10);
"8bjf4jvod8"
Math.random().toString(36).substr(2, 10);
"ji8yx6k29j"
```

Thật sự đơn giản không nào

2. Check empty Object

Chắc hẳn hầu hết các bạn đã từng làm JS thì 96,69% đều biết đến function `isEmpty` của lodash, để kiểm tra xem 1 Object có bị rỗng hay không, Tuy nhiên, vì lý do nào đó nếu không dùng lodash thì chúng ta có những cách xử lý nào khác không, với ES6 hoặc ES5 chẳng hạn...

Với ES5:

```
const emptyObj = {};

function isEmpty(obj) {
  for(let prop in obj) {
    if(obj.hasOwnProperty(prop)) {
      return false;
    }
  }

  return JSON.stringify(obj) === JSON.stringify({});
}

isEmpty(emptyObj) // true
```

Có vẻ khá phức tạp nhỉ!?! Vậy nếu dùng ES6 thì sao...

```
const emptyObj = {};

Object.keys(emptyObj).length === 0 && emptyObj.constructor === Object
// true

// or shorter
Object.entries(emptyObj).length === 0
// true
```

Khá đơn giản phải không nào

3. Convert Object to Array hoặc Array to Object

Trong khi làm việc, có những lúc cần convert một Object sang Array hoặc ngược lại, hãy cùng xem xét 1 vài trường hợp cụ thể áp dụng ES6 nhé.

Convert Array to Object với shallow array:

```
const names = ['Nguyen', 'Van', 'Thi', 'Mau'];
const obj = {...names};
// or
const obj = Object.assign({}, names);
console.log(obj);
// {0: "Nguyen", 1: "Van", 2: "Thi", 3: "Mau"}
```

Convert Array to Object với deep array:

```
const entries = [
  ['foo', 'bar'],
  ['baz', 42]
];

Object.fromEntries(entries)
// {foo: "bar", baz: 42}
```

Convert Object to Array:

```
const obj = {"0":"Banana","1":"Orange","2":"Apple","3":"Mango"};
const convertArray = Object.entries(obj)

console.log(convertArray)

// 0: (2) ["0", "Banana"]
// 1: (2) ["1", "Orange"]
// 2: (2) ["2", "Apple"]
// 3: (2) ["3", "Mango"]
// length: 4
// __proto__: Array(0)
```

4. Find and replace value của object trong Array

Để tìm kiếm và thay thế các giá trị mong muốn của 1 hoặc nhiều object trong Array, ta hoàn toàn có thể tận dụng function `findIndex` để giúp chúng ta làm việc này 1 cách nhanh chóng, ví dụ:

```
const item = {id: 2, name: 'abc'}
const items = [{id:1, name: 'ghi'}, {id:2, name: 'def'}, {id:3, name: 'mno'}];

const foundIndex = items.findIndex(x => x.id == item.id);
items[foundIndex] = item;
// {id: 2, name: "abc"}
```

Nếu với ES5, cũng hoàn toàn có thể được nhưng theo mình thì sẽ phức tạp hơn 1 chút với thêm 1 lệnh `if`:

```
items.forEach((element, index) => {
  if(element.id === item.id) {
    items[index] = item;
  }
});

// {id: 2, name: "abc"}
```