forEach, filter, map, reduce,...

Mở đầu

Bài viết này mình nhắm tới những người mới bắt đầu với Javascript hoặc những người mới bắt đầu làm việc với Javascript . Nhằm giúp refactor lại code nếu bạn gặp 1 số trường hợp dưới đây. Và thông qua đó giúp các bạn hiêu hơn về higher-order functions như map, reduce, and filter .....

Tại sao lại sử dụng map, filter, reduce ....

- Nhìn code sẽ dễ đọc hơn rất nhiều
- Dễ hiểu hơn
- Dễ dàng debug
- Tránh thay đổi mảng ban đầu, do đó, giảm thiểu những rủi ro có thể xảy ra .
- Không cần quản lí vòng lặp
- Hãy nhớ rằng nó luôn được trả về trong callback

1, Lặp qua tất cả các phần tử và nhận về 1 mảng đã sửa đổi

Đôi khi chúng ta có một mảng các object muốn sửa đổi / thêm các thuộc tính của từng đối tượng , cụ thể giả sử chúng ta có thể có một mảng các string mà ta sẽ biến tất cả chúng thành chữ thường . Trên thực tế có thể có vô số tình huống ở đó Map sẽ giúp chúng ta xử lý 1 cách dễ dàng .

Using for

```
var names = ["Jack", "Jecci", "Ram", "Tom"];
var upperCaseNames = [];
for(let i=0, totalNames = names.length; i< totalNames ; i= i +1) {
    upperCaseNames[i] = names[i].toUpperCase();
}
```

Using map

```
var names = ["Jack", "Jecci", "Ram", "Tom"];
var upperCaseNames = names.map(name => name.toUpperCase());
```

> Chú ý: Nếu bạn sử dụng map .Thì bạn sẽ không dùng được break, continue, return trong khi lặp . nếu muốn sử dụng chúng hãy dùng every hoặc some

2, Lặp qua tất cả các phần tử và thực hiện một hành động

Using for

```
function print(name) {
   console.log(name);
}
var names = ["Jack", "Jecci", "Ram", "Tom"];
for(let i=0, totalNames = names.length; i< totalNames ; i= i +1) {
    print(names[i])
}
```

Using forEach

```
var names = ["Jack", "Jecci", "Ram", "Tom"];
names.forEach(name=> print(name));
```

3, Lọc các phần tử của mảng

Tôi tin rằng trong quá trình bạn code bạn đã gặp tình huống phải lọc 1 số item ra khỏi mảng ban đầu .

Ví dụ: bài toán lọc các phần tử chăn trong mảng dưới đây

để làm được điều đó chúng ta phải xử lí khá rắc rối , nhưng với filter thì bạn có thể sử dụng nó 1 cách dễ dàng.

Using for

```
function isOdd(n) {
   return n %2;
}
var numbers = [1,2,3,4,5];
var odd = [];
for(let i=0, total = numbers.length; i< total ; i= i +1) {
   let number = numbers[i];
   if( isOdd(number) ) {
      odd.push(number);
   }
}
```

Using filter

```
var numbers = [1,2,3,4,5, 6, 7]
var odd = numbers.filter(n => n%2); // single line
```

Bạn thấy đấy ngắn hơn raart nhiều phải không .

4, Tính toán các giá trị thành một giá trị duy nhất

Ví dụ, với một dãy số bạn có thể dễ dàng tính được tổng của tất cả các giá trị.

Using for

```
var numbers = [1,2,3,4,5]
var result = 0;
for(let i=0, total = numbers.length; i< total ; i= i +1) {
   result = result + numbers[i];
}
```

Using reduce

```
var numbers = [1,2,3,4,5,6,7];
function sum(accumulator, currentValue){
   return accumulator + currentValue;
}
var initialVal = 0;
var result = numbers.reduce(sum, initialVal);
```

5, Kiểm tra xem một mảng có chứa giá trị nào đó không

Đôi khi bạn muốn kiểm tra 1 phần tử nào đó có tồn tại trong 1 array cho trước hay không. hãy nghĩ đến `some`

Using for

```
var names = ["ram", "raj", "rahul"];
for(let i=0, totalNames = names.length; i< totalNames ; i= i +1) {
   if(names[i] === "rahul") {
     console.log("%c found rahul", "color:red");
     return;
   }
}
```

Using some

```
var names = ["ram", "raj", "rahul"];
let isRahulPresent = names.some(name => name==="rahul");
if(isRahulPresent) {
  console.log("%c found rahul", "color:red");
}
```

6, kiểm tra xem mọi phần tử trong một mảng có đáp ứng điều kiện không

Using for

```
var num = [1,2,3,4,5, 0];
for(let i=0, total = numbers.length; i< total ; i= i +1) {
    if(num <= 0) {
      console.log("0 present in array");
    }
}
```

Using some

```
var num = [1,2,3,4,5, 0];
var isZeroFree = num.every(e => e > 0);
if(!isZeroFree) {
    console.log("0 present in array");
}
```