彻底搞清 JavaScript forEach & map

> 链接：https://segmentfault.com/a/1190000021171048

背景

JavaScript中,数组的遍历我们肯定都不陌生，最常见的两个便是forEach 和 map。

(当然还有别的譬如for, for in, for of, reduce, filter, every, some, ...)

之所以几天要写这个， 是因为前几天写代码的时候犯了一个低级且愚蠢的错误, 最后搞出了个小bug。

最后找到原因， 生气， 甚至还有点想笑， 今天就写一下这两个方法。

正文

我扑街的代码是这样的， 要给一个数组中的对象加一个属性， 我随手就写了如下代码：

```
// Add input_quantity into every list item
const dataSoruceAdapter = data => data.forEach((item) => {
  item.input_quantity = item.quantity
})

// ...
const transformedDataSource = dataSoruceAdapter(defaultDataSource);
const [orderList, setOrderList] = useState(transformedDataSource);
```

后面发现， 数组是空的... 囧

感觉自己真蠢， 改一下：

```
const dataSoruceAdapter = data => data.map((item) => {
  item.input_quantity = item.quantity
  return item
})
```

搞定。

我们仔细看一下forEach 和 map 这两个方法:

对比和结论

forEach: 针对每一个元素执行提供的函数。

map: 创建一个新的数组，其中每一个元素由调用数组中的每一个元素执行提供的函数得来。

直接说结论吧：

forEach方法不会返回执行结果，而是undefined。

也就是说，forEach会修改原来的数组，而map方法会得到一个新的数组并返回。

下面我们看下具体的例子。

forEach

forEach 方法按升序为数组中含有效值的每一项执行一次callback 函数，那些已删除或者未初始化的项将被跳过（例如在稀疏数组上）。

> forEach 接收两个参数: arr.forEach(callback[, thisArg]);

callback 函数会被依次传入三个参数：

1. 数组当前项的值
2. 数组当前项的索引
3. 数组对象本身

比如：

```
const arr = ['1', '2', '3'];
// callback function takes 3 parameters
// the current value of an array as the first parameter
// the position of the current value in an array as the second parameter
// the original source array as the third parameter
const cb = (str, i, origin) => {
  console.log(`${i}: ${Number(str)} / ${origin}`);
};
arr.forEach(cb);
// 0: 1 / 1,2,3
// 1: 2 / 1,2,3
// 2: 3 / 1,2,3
```

如果 thisArg 参数有值，则每次 callback 函数被调用的时候，this 都会指向 thisArg 参数上的这个对象。

如果省略了 thisArg 参数, 或者赋值为 null 或 undefined，则 this 指向全局对象。

举个勉强的例子，从每个数组中的元素值中更新一个对象的属性：

```
function Counter() {
  this.sum = 0;
  this.count = 0;
}

Counter.prototype.add = function(array) {
  array.forEach(function(entry) {
    this.sum += entry;
    this.count++;
  }, this);
  // console.log(this) -> Counter
};
```

```
var obj = new Counter();
obj.add([1, 3, 5, 7]);

obj.count;
// 4
obj.sum;
// 16
```

如果使用 `箭头函数` 来传入函数参数，thisArg 参数会被忽略，因为箭头函数在词法上绑定了 this 值。

一般情况下， 我们只会用到callback的前两个参数。

map

map 做的事情和 for循环 一样，不同的是， map 会创建一个新数组。

所以， 如果你不打算使用返回的新数组， 却依旧使用map的话， 这是违背map的设计初衷的。

map 函数接收两个参数：

1. callback callback 也有三个参数：
    1. currentValue
    2. index
    3. array
2. thisArg(可选)

这一点和forEach 是类似的。

具体的例子：

```
const arr = ['1', '2', '3'];
// callback function takes 3 parameters
// the current value of an array as the first parameter
// the position of the current value in an array as the second parameter
// the original source array as the third parameter
const cb = (str, i, origin) => {
  console.log(`${i}: ${Number(str)} / ${origin}`);
};
arr.map(cb);
// 0: 1 / 1,2,3
// 1: 2 / 1,2,3
// 2: 3 / 1,2,3
```

callback 方法用例：

```
arr.map((str) => { console.log(Number(str)); })
```

map 之后的结果是一个新数组， 和原数组是 `不相等` 的:

```
const arr = [1];
const new_arr = arr.map(d => d);
arr === new_arr; // false
```

你也可以指定第二个参数 `thisArg` 的值：

```
const obj = { name: 'Jane' };

[1].map(function() {
  console.dir(this); // { name: 'Jane' }
}, obj);
```

但是如果你使用的是箭头函数， 此时的 this将会是 window:

```
[1].map(() => {
  console.dir(this);   // window
}, obj);
```

箭头函数和普通函数的工作机制是不同的，不是本范围，你可以看这篇文章了解具体细节：

> https://www.geeksforgeeks.org...

写到这里， 就想起来一个经典的题目。

一个使用技巧案例

> ["1", "2", "3"].map(parseInt);

这道题我们都见过， 我们期望输出 [1, 2, 3], 而实际结果是 [1, NaN, NaN]。

我们简单来分析下过程, 首先看一下参数传递情况：

```
// parseInt(string, radix) -> map(parseInt(value, index))
第一次迭代(index is 0): parseInt("1", 0); // 1
第二次迭代(index is 1):  parseInt("2", 1); // NaN
第三次迭代(index is 2): parseInt("3", 2); //NaN
```

看到这， 解决办法也就呼之欲出了：

```
function returnInt(element) {
  return parseInt(element, 10);
}

['1', '2', '3'].map(returnInt); // [1, 2, 3]
```

是不是很容易理解?

回到正题。

forEach 和 map 的区别

看两行代码你就懂了：

```
[1,2,3].map(d => d + 1); // [2, 3, 4];
[1,2,3].forEach(d => d + 1); // undefined;
```

Vue作者，尤雨溪大佬，有这么一个形象的比方：

foreach 就是你按顺序一个一个跟他们做点什么，具体做什么，随便:

```
people.forEach(function (dude) {
  dude.pickUpSoap();
});
```

map 就是你手里拿一个盒子（一个新的数组），一个一个叫他们把钱包扔进去。结束的时候你获得了一个新的数组，里面是大家的钱包，钱包的顺序和人的顺序一一对应。

```
var wallets = people.map(function (dude) {
  return dude.wallet;
});
```

reduce 就是你拿着钱包，一个一个数过去看里面有多少钱啊？每检查一个，你就和前面的总和加一起来。这样结束的时候你就知道大家总共有多少钱了。var totalMoney = wallets.reduce(function (countedMoney, wallet) { return countedMoney + wallet.money; }, 0);

十分的贴切。

话题链接：https://www.zhihu.com/questio...

顺便送一段简单的原生实现, 感受下区别：

```
Array.prototype.map = function (fn) {
    var resultArray = [];
    for (var i = 0,len = this.length; i < len ; i++) {
         resultArray[i] = fn.apply(this,[this[i],i,this]);
    }
    return resultArray;
}

Array.prototype.forEach = function (fn) {
    for (var i = 0,len = this.length; i < len ; i++) {
         fn.apply(this,[this[i],i,this]);
    }
}

Array.prototype.reduce= function (fn) {
    var formerResult = this[0];
    for (var i = 1,len = this.length; i < len ; i++) {
         formerResult = fn.apply(this,[formerResult,this[i],i,this]);
    }
    return formerResult;
}
```

很简单的实现，仅仅实现功能，没做容错处理和特别严格的上下文处理。

什么时候使用 map 和 forEach

因为这两个的区别主要在于是不是返回了一个值， 所以需要生成新数组的时候， 就用map, 其他的就用forEach.

在 React 中， map 也经常被用来遍历数据生成元素:

```
const people = [
  { name: 'Josh', whatCanDo: 'painting' },
  { name: 'Lay', whatCanDo: 'security' },
  { name: 'Ralph', whatCanDo: 'cleaning' }
];

function makeWorkers(people) {
  return people.map((person) => {
    const { name, whatCanDo } = person;
    return <li key={name}>My name is {name}, I can do {whatCanDo}</li>
  });
}

<ul> {makeWorkers(people)}</ul>
```

当你不需要生成新书组的时候，用forEach:

```
const mySubjectId = ['154', '773', '245'];

function countSubjects(subjects) {
  let count = 0;

  subjects.forEach(subject => {
    if (mySubjectId.includes(subject.id)) {
      count += 1;
    }
  });

  return count;
}

const countNumber = countSubjects([
  { id: '223', teacher: 'Mark' },
  { id: '154', teacher: 'Linda' }
]);

countNumber; // 1
```

这段代码也可以简写为:

```
subjects.filter(subject => mySubjectId.includes(subject.id)).length;
```

谁更快

有些人说map 更快， 也有人说forEach 更快， 我也不确定， 所以就做了个测试， 得到如下结果：

结果1:

Testing in Chrome 78.0.3904 / Mac OS X 10.14.6		
Test		**Ops/sec**

forEach	`array.forEach(function(val, idx) {`
	`  array[idx]= val*0.8;`
	`});`
		2,876,604
		±0.45%
		fastest

map	`array= array.map(function(val) {`
	`  return val*0.8;`
	`});`
		2,080,994
		±0.38%
		28% slower

结果2:

Testing in Chrome 78.0.3904 / Mac OS X 10.14.6		
Test		**Ops/sec**

Map	`let doubled = arr.map(num => {`
	`  return num * 2;`
	`});`
		5,337
		±4.38%
		fastest

ForEach	`arr.forEach((num, index)=> {`
	`  return arr[index] = num * 2;`
	`});`
		4,534
		±2.00%
		13% slower

代码几乎都是一样的， 但是运行之后的结果恰恰相反。

其实吧， 我们不用纠结到底那个快，反正，都没有for快。

`可读性`， 才是我们要考虑的。

结论

说了一大堆， 相信大家肯定对这两个方法都有更清楚的认知了，我们再回顾下结论：

forEach 会修改原来的数组，而map方法会得到一个新的数组并返回。

所以需要生成新数组的时候， 就用map, 否则就用forEach.

以上就是全部内容，希望多大家有所帮助。