**[外婆的彭湖湾](https://www.cnblogs.com/penghuwan/)**

随笔 - 109  文章 - 0  评论 - 840

[**【javascript】详解javaScript的深拷贝**](https://www.cnblogs.com/penghuwan/p/7359026.html)

**目录**

**正文**

**前言：** 最开始意识到深拷贝的重要性是在我使用redux的时候（react + redux）， redux的机制要求在reducer中必须返回一个新的对象，而不能对原来的对象做改动，事实上，当时我当然不会主动犯这个错误，但很多时候，一不小心可能就会修改了原来的对象，例如：var newObj = obj; newObj.xxx = xxx  实际上，这个时候newObj和obj两个引用指向的是同一个对象，我修改了newObj，实际上也就等同于修改了obj，这，就是我和深浅拷贝的第一次相遇。

[回到顶部](https://www.cnblogs.com/penghuwan/p/7359026.html#_labelTop)

**浅谈深拷贝和浅拷贝**

**深拷贝和浅拷贝的区别**

**1.浅拷贝： 将原对象或原数组的引用直接赋给新对象，新数组，新对象／数组只是原对象的一个引用**

**2.深拷贝： 创建一个新的对象和数组，将原对象的各项属性的“值”（数组的所有元素）拷贝过来，是“值”而不是“引用”**

**为什么要使用深拷贝？**

**我们希望在改变新的数组（对象）的时候，不改变原数组（对象）**

**深拷贝的要求程度**

我们在使用深拷贝的时候，一定要弄清楚我们对深拷贝的要求程度：**是仅“深”拷贝第一层级的对象属性或数组元素**，**还是递归拷贝所有层级的对象属性和数组元素？**

**怎么检验深拷贝成功**

改变任意一个新对象/数组中的属性/元素,     都不改变原对象/数组

[回到顶部](https://www.cnblogs.com/penghuwan/p/7359026.html#_labelTop)

**只对第一层级做拷贝**

**深拷贝数组（只拷贝第一级数组元素）**

**1.直接遍历**

[复制代码](javascript:void(0);)

var array = [1, 2, 3, 4];

function copy (array) {

  let newArray = []

for(let item of array) {

  newArray.push(item);

  }

  return  newArray;

}

var copyArray = copy(array);

copyArray[0] = 100;

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

console.log(copyArray); // [100, 2, 3, 4]

[复制代码](javascript:void(0);)

该方法不做解释（逃...）

**2. slice()**

var array = [1, 2, 3, 4];

var copyArray = array.slice();

copyArray[0] = 100;

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

console.log(copyArray); // [100, 2, 3, 4]

slice() 方法返回一个从已有的数组中截取一部分元素片段组成的新数组**（不改变原来的数组！）**

**用法：array．slice(start,end)**　**start表示是起始元素的下标，　end表示的是终止元素的下标**

当slice()不带任何参数的时候，默认返回一个长度和原数组相同的新数组

**3. concat()**

var array = [1, 2, 3, 4];

var copyArray = array.concat();

copyArray[0] = 100;

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

console.log(copyArray); // [100, 2, 3, 4]

concat() 方法用于连接两个或多个数组**。( 该方法不会改变现有的数组，而仅仅会返回被连接数组的一个副本。)**

**用法：array.concat(array1,array2,......,arrayN)**

因为我们上面调用concat的时候没有带上参数，所以var copyArray = array.concat();实际上相当于var copyArray = array.concat([]);

也即把返回数组和一个空数组合并后返回

但是，事情当然不会这么简单，我上面的标题是 “深拷贝数组（只拷贝第一级数组元素）”，这里说的意思是对于一级数组元素是基本类型变量（如number,String,boolean）的简单数组, 上面这三种拷贝方式都能成功，**但对第一级数组元素是对象或者数组等引用类型变量的数组，上面的三种方式都将失效**，例如：

[复制代码](javascript:void(0);)

var array = [

{ number: 1 },

  { number: 2 },

{ number: 3 }

];

var copyArray = array.slice();

copyArray[0].number = 100;

console.log(array); //  [{number: 100}, { number: 2 }, { number: 3 }]

console.log(copyArray); // [{number: 100}, { number: 2 }, { number: 3 }]

[复制代码](javascript:void(0);)

**深拷贝对象**

**1.直接遍历**

[复制代码](javascript:void(0);)

var obj = {

name: '彭湖湾',

job: '学生'

}

function copy (obj) {

   let newObj = {};

for (let item in obj ){

newObj[item] = obj

}

return newObj;

}

var copyObj = copy(obj);

copyObj.name = '我才不是彭湖湾呢！ 哼 (。・`ω´・)';

console.log(obj); // {name: "彭湖湾", job: "学生"}

console.log(copyObj); // {name: "我才不是彭湖湾呢！ 哼 (。・`ω´・)", job: Object}

[复制代码](javascript:void(0);)

该方法不做解释（逃...）

**2.ES6的Object.assign**

[复制代码](javascript:void(0);)

var obj = {

name: '彭湖湾',

job: '学生'

}

var copyObj = Object.assign({}, obj);

copyObj.name = '我才不叫彭湖湾呢！ 哼  (。・`ω´・)';

console.log(obj);   // {name: "彭湖湾", job: "学生"}

console.log(copyObj);  // {name: "我才不叫彭湖湾呢！ 哼  (。・`ω´・)", job: "学生"}

[复制代码](javascript:void(0);)

**Object.assign：用于对象的合并，将源对象（source）的所有可枚举属性，复制到目标对象（target），并返回合并后的target**

**用法： Object.assign(target, source1, source2);**  所以 copyObj = Object.assign({}, obj);  这段代码将会把obj中的一级属性都拷贝到 ｛｝中，然后将其返回赋给copyObj

**3.ES6扩展运算符：**

[复制代码](javascript:void(0);)

var obj = {

    name: '彭湖湾',

    job: '学生'

}

var copyObj = { ...obj }

copyObj.name = '我才不叫彭湖湾呢！ 哼  (。・`ω´・)'

console.log(obj.name) //   彭湖湾

console.log(copyObj.name)  // 我才不叫彭湖湾呢！ 哼  (。・`ω´・)

[复制代码](javascript:void(0);)

**扩展运算符（...）用于取出参数对象的所有可遍历属性，拷贝到当前对象之中**

对多层嵌套对象，很遗憾，上面三种方法，都会失败：

[复制代码](javascript:void(0);)

var obj = {

   name: {

firstName: '彭',

lastName: '湖湾'

   },

   job: '学生'

}

var copyObj = Object.assign({}, obj)

copyObj.name.lastName = '湖水的小浅湾';

console.log(obj.name.lastName); // 湖水的小浅湾

console.log(copyObj.name.lastName); // 湖水的小浅湾

[复制代码](javascript:void(0);)

[回到顶部](https://www.cnblogs.com/penghuwan/p/7359026.html#_labelTop)

**拷贝所有层级**

有没有更强大一些的解决方案呢？使得我们能够

**1.不仅拷贝第一层级，还能够拷贝数组或对象所有层级的各项值**

**2. 不是单独针对数组或对象，而是能够通用于数组，对象和其他复杂的JSON形式的对象**

请看下面：

**下面这一招可谓是“一招鲜，吃遍天”**

**1.JSON.parse(JSON.stringify(XXXX))**

[复制代码](javascript:void(0);)

var array = [

    { number: 1 },

    { number: 2 },

    { number: 3 }

];

var copyArray = JSON.parse(JSON.stringify(array))

copyArray[0].number = 100;

console.log(array); //  [{number: 1}, { number: 2 }, { number: 3 }]

console.log(copyArray); // [{number: 100}, { number: 2 }, { number: 3 }]

[复制代码](javascript:void(0);)

**能用大招杀的就不要用q杀嘛！！**

**2.手动写递归**

**你说啥？ 你说上面的那种方法太无脑,  一定要自己写一段递归才有做技术的感觉？ OK成全你！**

[复制代码](javascript:void(0);)

var array = [

   { number: 1 },

   { number: 2 },

   { number: 3 }

];

function copy (obj) {

        var newobj = obj.constructor === Array ? [] : {};

        if(typeof obj !== 'object'){

            return;

        }

        for(var i in obj){

           newobj[i] = typeof obj[i] === 'object' ?

           copy(obj[i]) : obj[i];

        }

        return newobj

}

var copyArray = copy(array)

copyArray[0].number = 100;

console.log(array); //  [{number: 1}, { number: 2 }, { number: 3 }]

console.log(copyArray); // [{number: 100}, { number: 2 }, { number: 3 }]

[复制代码](javascript:void(0);)

【注意】：上文的所有的示例都忽略了一些特殊的情况： 对对象/数组中的Function，正则表达式等特殊类型的拷贝

[回到顶部](https://www.cnblogs.com/penghuwan/p/7359026.html#_labelTop)

**存在大量深拷贝需求的代码——immutable提供的解决方案**

实际上，即使我们知道了如何在各种情况下进行深拷贝，我们也仍然面临一些问题： 深拷贝实际上是很消耗性能的**。（我们可能只是希望改变新数组里的其中一个元素的时候不影响原数组，但却被迫要把整个原数组都拷贝一遍，这不是一种浪费吗？）**所以，当你的项目里有大量深拷贝需求的时候，性能就可能形成了一个制约的瓶颈了。

**immutable的作用**：

通过immutable引入的一套API，实现：

**1.在改变新的数组（对象）的时候，不改变原数组（对象）**

**2.在大量深拷贝操作中显著地减少性能消耗**

先睹为快：

const { Map } = require('immutable')

const map1 = Map({ a: 1, b: 2, c: 3 })

const map2 = map1.set('b', 50)

map1.get('b') // 2

map2.get('b') // 50