

深入理解JS对象的深度克隆及多种方式实现



jCodeLife



0.098 2020.08.08 21:33:00 字数 2,285 阅读 55

编辑文章

关于对象的深度克隆网上都会讲到，但看过很多资料、文章觉得讲得不太全、零零散散、漏这漏那，特写此文一来汇总；二来更透彻的掌握深度克隆。

我们知道

JavaScript的数据类型分两大类：

1. 原始值类型，包括：number、string、boolean、null、undefined
2. 引用值类型，包括：对象（object）、函数（function）、数组（array）

当然ES6引入了一种新的数据类型Symbol，表示独一无二的值

在JS对象中可包含所有数据类型中的任意一些类型，当我们拷贝对象时，必须要考虑到所有的数据类型的拷贝。所以我们的目标就是：实现JS中所有数据类型的深度拷贝！包括function，包括symbol

真正的内容开始咯



首先，通过typeof判断是原始值、object、function还是symbol；

```
1 function deepClone(origin, target) {
2   //origin: 要被拷贝的对象
3   var target = target || {};
4   for (var prop in origin) {
5     if (origin.hasOwnProperty(prop)) {
6       if (typeof (origin[prop]) === "object") {
7
8       } else if (typeof (origin[prop]) === "function") {
9
10      } else if (typeof (origin[prop]) === "symbol") {
11
12      } else {
13        //除了object、function、symbol，剩下都是直接赋值的原始值
14        target[prop] = origin[prop];
15      }
16    }
17  }
18 }
```

备注：

- for in循环用来遍历对象属性的，但是会遍历到原型上的
- origin.hasOwnProperty()用于过滤，看看属性到底是不是自己的（除去原型的）
- typeof后除了object、function、symbol，剩下都是直接赋值的原始值，包括number、string、boolean

接着，通过Object.prototype.toString.call()判断object类型是数组、对象还是null

```
1 function deepClone(origin, target) {
2   //origin: 要被拷贝的对象
```

推荐阅读

Vue干货小技巧——学会再也不做加班狗

阅读 1,144

vue优化页面

阅读 701

面试了几个前端，给爷整哭了！

阅读 19,305

HTML5抽奖转盘-CSS3超简单版本

阅读 55

vue 3.0 的初次体验

阅读 90

```
8      //数组
9      } else if (Object.prototype.toString.call(origin[prop]) == "[object Object]")
10         //普通对象
11     } else {
12         //null
13     }
14
15     } else if (typeof (origin[prop]) === "function") { //函数
16
17     } else if (typeof (origin[prop]) === "symbol") {
18
19     } else {
20         //除了object、function、symbol，剩下都是直接赋值的原始值
21         target[prop] = origin[prop];
22     }
23 }
24 }
25 }
```

然后，根据是数组还是对象建立相应的数组或对象；但是因为数组和对象一样，可以存放所以类型的变量，所以这两种数据类型得用到递归，调用本身函数deepClone()

```
1 function deepClone(origin, target) {
2     //origin:要被拷贝的对象
3     var target = target || {};
4     for (var prop in origin) {
5         if (origin.hasOwnProperty(prop)) {
6             if (typeof (origin[prop]) === "object") { //对象
7                 if (Object.prototype.toString.call(origin[prop]) == "[object Array]") {
8                     //数组
9                     target[prop] = [];
10                    deepClone(origin[prop],target[prop]);
11                } else if (Object.prototype.toString.call(origin[prop]) == "[object Object]") {
12                    //普通对象
13                    target[prop] = {};
14                    deepClone(origin[prop],target[prop]);
15                } else {
16                    //null
17                    origin[prop] = null;
18                }
19
20                } else if (typeof (origin[prop]) === "function") { //函数
21
22                } else if (typeof (origin[prop]) === "symbol") {
23
24                } else {
25                    //除了object、function、symbol，剩下都是直接赋值的原始值
26                    target[prop] = origin[prop];
27                }
28            }
29        }
30        return target;
31    }
```

这里有些人会有个疑问，就是当是数组时，我递归调用deepClone(origin[prop],target[prop]);传递的数组能使用for in遍历？答案是肯定的！for in也可以遍历数组，origin[prop]相当于是origin[0]、origin[1]、origin[2]、origin[3]....

接下来，深度克隆function

有两种方法克隆function

- 通过eval()
- 通过new Function()

```
1 var a = function(){alert(1)}
2 var b = eval("0,"+a);//方法一
3 var c = new Function("return "+a()); //方法二
```

推荐阅读

Vue干货小技巧——学会再也不做加班狗

阅读 1,144

vue优化页面

阅读 701

面试了几个前端，给爷整哭了！

阅读 19,305

HTML5抽奖转盘-CSS3超简单版本

阅读 55

vue 3.0 的初次体验

阅读 90

但是如果函数上面附有许多静态属性，我们可以封装一个专门的函数来实现函数的深度拷贝：

```
1 var copyFn = function (fn) {  
2   var result = eval("0," + fn);  
3   for (var i in fn) {  
4     result[i] = fn[i]  
5   }  
6   return result  
7 }
```



最后，实现symbol类型的拷贝：

温馨提示：不考虑拷贝symbol类型的可以跳过这部分，这部分也比较长

先来了解一下symbol吧

Symbol是ES6 引入了一种新的数据类型，用于表示独一无二的值

在ES6出来之前，对象属性名都是字符串，这容易造成属性名的冲突。比如，你使用了一个他人提供的对象，但又想为这个对象添加新的方法（mixin 模式），新方法的名字就有可能与现有方法产生冲突。如果有一种机制，保证每个属性的名字都是独一无二的就好了，这样就从根本上防止属性名的冲突。这就是 ES6 引入Symbol的原因。

Symbol 值通过Symbol函数生成。

```
1 let s = Symbol();  
2 typeof s; // "symbol"
```

这就是说，对象的属性名现在可以有两种类型，一种是原来就有的字符串，另一种就是新增的Symbol 类型。凡是属性名属于 Symbol 类型，就都是独一无二的，可以保证不会与其他属性名产生冲突。

注意，Symbol函数前不能使用new命令，否则会报错。这是因为生成的 Symbol 是一个原始类型的值，不是对象。也就是说，由于 Symbol 值不是对象，所以不能添加属性。基本上，它是一种类似于字符串的数据类型。

Symbol函数可以接受一个字符串作为参数，表示对 Symbol 实例的描述，主要是为了在控制台显示，或者转为字符串时，比较容易区分。

```
1 let s1 = Symbol('foo');  
2 let s2 = Symbol('bar');  
3  
4 console.log(s1); // Symbol(foo)  
5 console.log(s2); // Symbol(bar)  
6  
7 console.log(s1.toString()); // "Symbol(foo)"  
8 console.log(s2.toString()); // "Symbol(bar)"
```

如果 Symbol 的参数是一个对象，就会调用该对象的toString方法（没有就找原型上的toString方法），将其转为字符串，然后才生成一个 Symbol 值。

```
1 const obj = {  
2   toString() {  
3     return 'abc';  
4   }  
5 };  
6 const sym = Symbol(obj);  
7 console.log(sym); // Symbol(abc)
```

推荐阅读

Vue干货小技巧——学会再也不做加班狗

阅读 1,144

vue优化页面

阅读 701

面试了几个前端，给爷整哭了！

阅读 19,305

HTML5抽奖转盘-CSS3超简单版本

阅读 55

vue 3.0 的初次体验

阅读 90

```
1 var a = Symbol("a");
2 var b = Symbol("a");
3 console.log(a == b); // false
4
5 b = a;
6 console.log(a == b); // true
```

那我们如何拷贝 Symbol 类型呢？

有文章提到有个方法可以实现：

- 方法：Object.getOwnPropertySymbols(obj)

Object.getOwnPropertySymbols(obj)，用于返回在给定对象自身找到的所有 Symbol 类型的属性的数组。

注意：因为所有的对象在初始化的时候不会包含任何的 Symbol，除非你在对象上赋值了 Symbol 否则Object.getOwnPropertySymbols()只会返回一个空的数组。

```
1 var obj = {};
2 var a = Symbol("a");
3 var b = Symbol.for("b");
4
5 obj[a] = "localSymbol";
6 obj[b] = "globalSymbol";
7
8 var objectSymbols = Object.getOwnPropertySymbols(obj);
9
10 console.log(objectSymbols.length); // 2
11 console.log(objectSymbols) // [Symbol(a), Symbol(b)]
12 console.log(objectSymbols[0]) // Symbol(a)
```

但是我有点不解，这方法不就是找到对象身上的 Symbol 类型的属性吗，怎么就能实现symbol类型的拷贝呢？单纯查找区分的话，我在typeof的时候不是也能找到吗？不解！在写个代码测试 try try

```
1 var obj = {
2   c: Symbol("cc"),
3   fn: function(){
4     // console.log(typeof this.c)
5   }
6 };
7 for (var temp in obj){
8   console.log(temp, typeof(obj[temp]))
9 }
```

打印：'c symbol'和'fn function'

证实for in确实可以遍历出对象中定义symbol类型的属性。

那我们猜测一下，通过obj.xx=Symbol()方式后面添加的属性，能不能通过forin拿出来

```
1 var obj = {
2   c: Symbol("cc"),
3   fn: function(){
4     // console.log(typeof this.c)
5   }
6 };
7 var a = Symbol("aa");
8 var b = Symbol.for("bb");
9 obj[a] = "localSymbol";
10 obj[b] = "globalSymbol";
11 for (var temp in obj){
12   console.log(temp, typeof(obj[temp])) // 'c symbol' 和 'fn function'
13 }
14
15 console.log(obj[a], obj[b]) // localSymbol globalSymbol
```

推荐阅读

Vue干货小技巧——学会再也不做加班狗

阅读 1,144

vue优化页面

阅读 701

面试了几个前端，给爷整哭了！

阅读 19,305

HTML5抽奖转盘-CSS3超简单版本

阅读 55

vue 3.0 的初次体验

阅读 90

那怎么办？

再回过头来试试getOwnPropertySymbols代替forin找出对象的symbol属性

```
1 var obj = {
2   c: Symbol("cc"),
3   fn: function(){}
4 };
5 var a = Symbol("aa");
6 var b = Symbol.for("bb");
7 obj[a] = "localSymbol";
8 obj[b] = "globalSymbol";
9
10 var objectSymbols = Object.getOwnPropertySymbols(obj);
11 console.log(objectSymbols.length); // 2
12 console.log(objectSymbols)        // [Symbol(aa), Symbol(bb)]
13 console.log(typeof(objectSymbols[0])) // symbol
```

发现通过getOwnPropertySymbols可以找出后面添加的symbol值

小结:

- 通过typeof找出symbol类型是直接在里面定义的，但是个人感觉这种值没多大意义，只是用于作一个对象的标识。
- 通过getOwnPropertySymbols找出对象声明之后通过对象[prop]=Symbol("xxx")的形式添加的属性。

所以分两个部分来处理symbol:

1. 对象声明时的symbol值:

特点: for in遍历的到，并可以直接用typeof判断为symbol

2. 对象声明之后的symbol值:

特点: 通过for in遍历不出来，而是通过getOwnPropertySymbols获取这类symbol值。

到这完成了一大半，剩下就是怎么赋值拷贝了。

我的观点是，反正拷贝的结果就是看上去和功能都是一模一样，但是彼此独立不关联。

symbol的特性就是独一无二，所以天生不关联，我们只要考虑生成的时候传递的一样就行，即Symbol(context)中的context值。

上面我们讲过toString方法，我们深入试试传入不同类型时，返回的值有什么区别:

```
1 //Symbol传入不同类型的值
2 var _string = 'aa';
3 var _number = 1;
4 var _boolean = true;
5 var _undefined = undefined;
6 var _null = null;
7 var _array = [1,2];
8 var _fn = function(){a:1};
9 var _obj = {name:"alice"};
10 var _symbol = Symbol("s");
11 console.log(Symbol(_string).toString()); //Symbol(aa)
12 console.log(Symbol(_number).toString()); //Symbol(1)
13 console.log(Symbol(_boolean).toString()); //Symbol(true)
14 console.log(Symbol(_undefined).toString()); //Symbol()
15 console.log(Symbol(_null).toString()); //Symbol(null)
16 console.log(Symbol(_fn).toString()); //Symbol(function(){a:1})
17 console.log(Symbol(_obj).toString()); //Symbol([object Object])
18 console.log(Symbol(_symbol).toString()); //test.html:431 Uncaught TypeError: Cannot convert a
```

我们通过Symbol传入不同类型的值，发现

Symbol传入的值必须是字符串，如果不是，会调用String()发生隐式类型转换，将其转成字符串。特殊的

Symbol(undefined).toString()返回是Symbol()

推荐阅读

Vue干货小技巧——学会再也不做加班狗

阅读 1,144

vue优化页面

阅读 701

面试了几个前端，给爷整哭了！

阅读 19,305

HTML5抽奖转盘-CSS3超简单版本

阅读 55

vue 3.0 的初次体验

阅读 90

```
3   var tempArr = str.split("");
4   var arr = tempArr[1].split("")[0];
5   return Symbol(arr);
6 }
```

我们先来解决对象声明时的symbol:

```
1 function deepClone(origin, target) {
2   //origin:要被拷贝的对象
3   var target = target || {};
4   for (var prop in origin) {
5     if (origin.hasOwnProperty(prop)) {
6       if (typeof (origin[prop]) === "object") { //对象
7         if (Object.prototype.toString.call(origin[prop]) == "[object Array]") {
8           //数组
9           target[prop] = [];
10          deepClone(origin[prop], target[prop]);
11        } else if (Object.prototype.toString.call(origin[prop]) == "[object Object]") {
12          //普通对象
13          target[prop] = {};
14          deepClone(origin[prop], target[prop]);
15        } else {
16          //null
17          console.log(11111111111)
18          target[prop] = null;
19        }
20      }
21      } else if (typeof (origin[prop]) === "function") { //函数
22        var _copyFn = function (fn) {
23          var result = new Function("return " + fn)();
24          for (var i in fn) {
25            result[i] = fn[i]
26          }
27          return result
28        }
29        target[prop] = _copyFn(origin[prop]);
30      } else if (typeof (origin[prop]) === "symbol") { //里面的symbol
31        target[prop] = _copySymbol(origin[prop]);
32        console.log(target[prop])
33      } else {
34        //除了object、function、symbol，剩下都是直接赋值的原始值number,string,boolean
35        target[prop] = origin[prop];
36      }
37    }
38    //但有个特殊的是，string中可能有symbol，
39    //那我可以通过getOwnPropertySymbols先找出来
40    // var _tempArr = Object.getOwnPropertySymbols(origin);
41    // console.log(_tempArr)
42    // for (var i = 0;i<_tempArr.length;i++){
43    //   _tempArr[i] ==
44    // }
45    // if (typeof (origin[prop] === "string" )) {
46
47    // } else {
48    // }
49
50  }
51 }
52 }
53 function _copySymbol(val) {
54   var str = val.toString();
55   var tempArr = str.split("(");
56   var arr = tempArr[1].split("")[0];
57   return Symbol(arr);
58 }
59 return target;
60 }
61
62 var student = {
63   name: "alice",
64   age: 12,
65   isOldPerson: false,
66   sex: undefined,
67   money: null,
68   grader: [{
```

推荐阅读

Vue干货小技巧——学会再也不做加班狗

阅读 1,144

vue优化页面

阅读 701

面试了几个前端，给爷整哭了！

阅读 19,305

HTML5抽奖转盘-CSS3超简单版本

阅读 55

vue 3.0 的初次体验

阅读 90

```
74     },
75     key: Symbol("s1-key"),
76     book: {
77       English: true
78     }
79   }
80   var a = Symbol('a')
81   var b = Symbol.for("b")
82   student[a] = "aaa"
83   student[b] = "bbb"
84
85   var res = deepClone(student);
```

student

```
{name: "alice", age: 12, isOldPerson: false, sex: undefined, money: null, ...}
  age: 12
  book: {English: true}
  grader: (2) [{...}, 100]
  isOldPerson: false
  key: Symbol(s1-key)
  money: null
  name: "alice"
  sex: undefined
  study: f ()
  Symbol(a): "aaa"
  Symbol(b): "bbb"
  __proto__: Object
```

res

```
{name: "alice", age: 12, isOldPerson: false, sex: undefined, money: null, ...}
  age: 12
  book: {English: true}
  grader: (2) [{...}, 100]
  isOldPerson: false
  key: Symbol(s1-key)
  money: null
  name: "alice"
  sex: undefined
  study: f ()
    arguments: null
    caller: null
    length: 0
    name: ""
    prototype: {constructor: f}
    __proto__: f ()
    [[FunctionLocation]]: VM350:3
    [[Scopes]]: Scopes[1]
  __proto__: Object
```

可以看到，剩下最后一个问题了，就是就是后面添加的symbol，即在forin外头实现。

也是手写对象深度克隆的最终代码

```
1 function deepClone(origin, target) {
2   //origin: 要被拷贝的对象
3   var target = target || {};
4   for (var prop in origin) {
5     if (origin.hasOwnProperty(prop)) {
6       if (typeof (origin[prop]) === "object") { //对象
7         if (Object.prototype.toString.call(origin[prop]) == "[object Array]") {
8           //数组
9           target[prop] = [];
10          deepClone(origin[prop], target[prop]);
11        } else if (Object.prototype.toString.call(origin[prop]) == "[object Object]") {
12          //普通对象
13          target[prop] = {};
14          deepClone(origin[prop], target[prop]);
15        } else {
16          //null
17          target[prop] = null;
18        }
19      }
20    } else if (typeof (origin[prop]) === "function") { //函数
```

推荐阅读

Vue干货小技巧——学会再也不做加班狗

阅读 1,144

vue优化页面

阅读 701

面试了几个前端，给爷整哭了！

阅读 19,305

HTML5抽奖转盘-CSS3超简单版本

阅读 55

vue 3.0 的初次体验

阅读 90

```
27         }
28         target[prop] = _copyFn(origin[prop]);
29     } else if (typeof (origin[prop]) === "symbol") { // 里面的symbol
30         target[prop] = _copySymbol(origin[prop]);
31     } else {
32         // 除了 object、function、symbol，剩下都是直接赋值的原始值 number, string, boolean, und
33         target[prop] = origin[prop];
34     }
35 }
36 }
37 function _copySymbol(val) {
38     var str = val.toString();
39     var tempArr = str.split("");
40     var arr = tempArr[1].split("")[0];
41     return Symbol(arr);
42 }
43 // 通过 getOwnPropertySymbols 找出来的 symbol
44 var _symArr = Object.getOwnPropertySymbols(origin);
45 if (_symArr.length) { // 查找成功
46     _symArr.forEach(symKey => {
47         target[symKey] = origin[symKey];
48     });
49 }
50 return target;
51 }
```

试试

```
1  var student = {
2      name: "alice",
3      age: 12,
4      isOldPerson: false,
5      sex: undefined,
6      money: null,
7      grader: [{
8          English: 120,
9          math: 80
10     }, 100],
11     study: function () {
12         console.log("I am a student, I have to study every day!");
13     },
14     key: Symbol("s1-key"),
15     book: {
16         English: true
17     }
18 }
19 var a = Symbol('a')
20 var b = Symbol.for("b")
21 student[a] = "11111111111111"
22 student[b] = "2222222222222222"
23 var res = deepClone(student);
```

推荐阅读

[Vue干货小技巧——学会再也不做加班狗](#)

阅读 1,144

[vue优化页面](#)

阅读 701

[面试了几个前端，给爷整哭了！](#)

阅读 19,305

[HTML5抽奖转盘-CSS3超简单版本](#)

阅读 55

[vue 3.0 的初次体验](#)

阅读 90


```
book: {English: true}
```

测试结果

完美！

另外还有一方法，通过`Reflect.ownKeys(origin)`，用于返回一个由目标对象自身的属性键组成的数组，这也能找到后面的`symbol`类型。你可以自己试试怎么实现。

其他方式实现深度克隆

除了上面手写实现以外，还有以下几个方法：

- 通过扩展运算符...

```
1 var a = {name:"aaa"}
2 var b = {...a}
```

- 通过合并对象的方法`Object.assign`

```
1 var newObj = Object.assign([],oldObj);
```

缺点：`Object.assign`只对顶层属性做了赋值，完全没有继续做递归把下层属性做深度拷贝。简而言之，只实现第一层深度拷贝，后续层次还是浅拷贝。

- 通过`json`的方法实现

```
1 var obj = {a:1};
2 var str = JSON.stringify(obj); //序列化对象
3 var newObj = JSON.parse(str); //还原
```

缺点：

如果`obj`里面有时间对象，则`JSON.stringify`后再`JSON.parse`的结果，时间将只是字符串的形式。而不是时间对象；

如果`obj`里有`RegExp`、`Error`对象，则序列化的结果将只得到空对象；

如果`obj`里有`function`，`Symbol`类型，`undefined`，则序列化的结果会把函数或`undefined`丢失；

如果`obj`里有`NaN`、`Infinity`和`-Infinity`，则序列化的结果会变成`null`

- `jquery`中的`$.extend`

```
1 var bob = {
2   name: "Bob",
3   age: 32
4 };
5
6 var bill = $.extend(true, {}, bob);
7 bill.name = "Bill";
8
9 console.log(bob.name); //Bob
10 console.log(bill.name); //Bill
```

参考文献：

深度克隆对象（不考虑`function`）-腾讯课堂渡一教育

JS对象深度克隆的实现 - 作者：鱼子ivan

推荐阅读

Vue干货小技巧——学会再也不做加班狗

阅读 1,144

vue优化页面

阅读 701

面试了几个前端，给爷整哭了！

阅读 19,305

HTML5抽奖转盘-CSS3超简单版本

阅读 55

vue 3.0 的初次体验

阅读 90



写下你的评论...

评论 0

赞 3


...

深入理解JS对象的深度克隆及多种方式实现


Symbol (js的第七种数据类型) — 作者: 刘欢乐

如何实现一个深拷贝 (考虑循环引用对象、和symbol类型) — 作者: Dream_Lee_1997

Object.getOwnPropertySymbols--MDN web docs

 3人点赞 >




 笔记 (1)



"小礼物走一走，来简书关注我"

赞赏支持


还没有人赞赏，支持一下



jCodeLife

书山有路勤为径，学海无涯苦作舟

总资产13 (约1.21元) 共写了11.8W字 获得216个赞 共21个粉丝



写下你的评论...

全部评论 0


只看作者

关闭评论

按时间倒序

按时间正序

被以下专题收入，发现更多相似内容

 投稿管理

+ 收入我的专题

 领跑计划笔记