深入理解JS对象的深度克隆及多种方式实现

jCodeLife
2020.08.08 21:33:00 字数 1,634 阅读 0

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

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

```
function deepClone(origin, target) {
    //origin:要被拷贝的对象
    var target = target || {};
    for (var prop in origin) {
        if (origin.hasOwnPoperty(prop){
            if (typeof (origin[prop]) === "object") {

            } else if (typeof (origin[prop]) === "function") {

            } else if (typeof (origin[prop]) === "symbol") {

            } else {
                //除了object、function、symbol，剩下都是直接赋值的原始值
                target[prop] = origin[prop];
            }
        }
    }
}
```

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.hasOwnPoperty()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

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

jCodeLife
总资产4 (约0.40元)

深入call,apply,bind到手动封装
阅读 40

区分数组还是对象的四种方法
阅读 1

命名空间
阅读 4

推荐阅读

一眼看透本质的五种思维方式
阅读 6,014

枕上留书（八十三）
阅读 3,032

27岁女白领，公开夫妻私生活引热议：纵欲上瘾，正在榨干年轻人
阅读 35,042

理直气壮的说离婚
阅读 3,123

三生三世枕上书续写27
阅读 2,520

```
                } else if (Object.prototype.toString.call(origin[prop]) == "[object Object]") {
                    //普通对象
                } else {
                    //null
                }

            } else if (typeof (origin[prop]) === "function") {//函数

            } else if (typeof (origin[prop]) === "symbol") {

            } else {
                //除了object、function、symbol，剩下都是直接赋值的原始值
                target[prop] = origin[prop];
            }
        }
    }
}
```

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()

```
function deepClone(origin, target) {
    //origin:要被拷贝的对象
    var target = target || {};
    for (var prop in origin) {
        if (origin.hasOwnPoperty(prop)) {
            if (typeof (origin[prop]) === "object") {//对象
                if (Object.prototype.toString.call(origin[prop]) == "[object Array]") {
                    //数组
                    target[prop] = [];
                    deepClone(origin[prop],target[prop]);
                } else if (Object.prototype.toString.call(origin[prop]) == "[object Object]") {
                    //普通对象
                    target[prop] = {};
                    deepClone(origin[prop],target[prop]);
                } else {
                    //null
                    origin[prop] = null;
                }

            } else if (typeof (origin[prop]) === "function") {//函数

            } else if (typeof (origin[prop]) === "symbol") {

            } else {
                //除了object、function、symbol，剩下都是直接赋值的原始值
                target[prop] = origin[prop];
            }
        }
    }
  return target;
}
```

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()

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

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：

```
var copyFn = function (fn) {
    var result = eval("0," + fn);
    for (var i in fn) {
        result[i] = fn[i]
    }
    return result
}
```

最后，实现symbol类型的拷贝：

先来了解symbol吧

Symbol是ES6 引入了一种新的数据类型，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。

```
let s = Symbol();
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。

```
let s1 = Symbol('foo');
let s2 = Symbol('bar');

console.log(s1);// Symbol(foo)
console.log(s2);// Symbol(bar)

console.log(s1.toString()); // "Symbol(foo)"
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。

```
const obj = {
  toString() {
    return 'abc';
  }
};
const sym = Symbol(obj);
console.log(sym); // Symbol(abc)
```

注意，Symbol函数的参数只是表示对当前 Symbol 值的描述，因此相同参数的Symbol函数的返回值是不相等的。

```
var a = Symbol("a");
var b = Symbol("a");
```

那我们如何拷贝 Symol 类型呢？
我们先来学习两个方法，一会会用到：

- 方法1：Object.getOwnPropertySymbols(obj)
  Object.getOwnPropertySymbols(obj)，用于返回在给定对象自身上找到的所有 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。

```
var obj = {};
var a = Symbol("a");
var b = Symbol.for("b");

obj[a] = "localSymbol";
obj[b] = "globalSymbol";

var objectSymbols = Object.getOwnPropertySymbols(obj);

console.log(objectSymbols.length); // 2
console.log(objectSymbols)         // [Symbol(a), Symbol(b)]
console.log(objectSymbols[0])      // Symbol(a)
```

- 方法2：Reflect.ownKeys(...)
  返回一个由目标对象自身的属性键组成的数组。它的返回值等同于 Object.getOwnPropertyNames(target).concat(Object.getOwnPropertySymbols(target))

```
Reflect.ownKeys({ z: 3, y: 2, x: 1 }); // [ "z", "y", "x" ]
Reflect.ownKeys([]); // ["length"]

var sym = Symbol.for("comet");
var sym2 = Symbol.for("meteor");
var obj = {
    [sym]: 0,
    "str": 0,
    "773": 0,
    "0": 0,
    [sym2]: 0,
    "-1": 0,
    "8": 0,
    "second str": 0
};
Reflect.ownKeys(obj);
// [ "0", "8", "773", "str", "-1", "second str", Symbol(comet), Symbol(meteor) ]
// 注意顺序
```

下面利用这两个方法来实现symbol的拷贝
方法一
symbol剩余部分待完善...

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...

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

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

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

```
var bob = {
    name: "Bob",
    age: 32
};

var bill = $.extend(true, {}, bob);
bill.name = "Bill";

console.log(bob.name);//Bob
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：兔子juan
js对象的深度克隆的三种方法（深拷贝）—作者：web全栈入门
js深度克隆的几种方法—作者：javascript
克隆function—作者：weixin_34194551
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

jCodeLife 书山有路勤为径，学海无涯苦作舟
总资产4 (约0.40元) 共写了8.3W字 获得95个赞 共19个粉丝

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 渡一领跑计划笔记