实现 javascript 中深拷贝和浅拷贝的方法

前言

Js 数据类型

1.javascript 变量包含两种不同数据类型的值：基本类型和引用类型。

1. 基本类型

基本类型值指的是简单的数据段，包括 es6 里面新增的一共是有 6 种，具体如下：number、string、boolean、null、undefined、symbol。

基本类型的数据是存放在栈内存中的

2. 引用类型

对象 Object 、数组 Array 、函数 Function

引用类型的数据是存放在堆内存中的

引用类型（object）是存放在堆内存中的，变量实际上是一个存放在栈内存的指针，这个指针指向堆内存中的地址。每个空间大小不一样，要根据情况开进行特定的分配，例如。

```
var person1 = {name:'jozo'};
var person2 = {name:'xiaom'};
var person3 = {name:'xiaoq'};
```

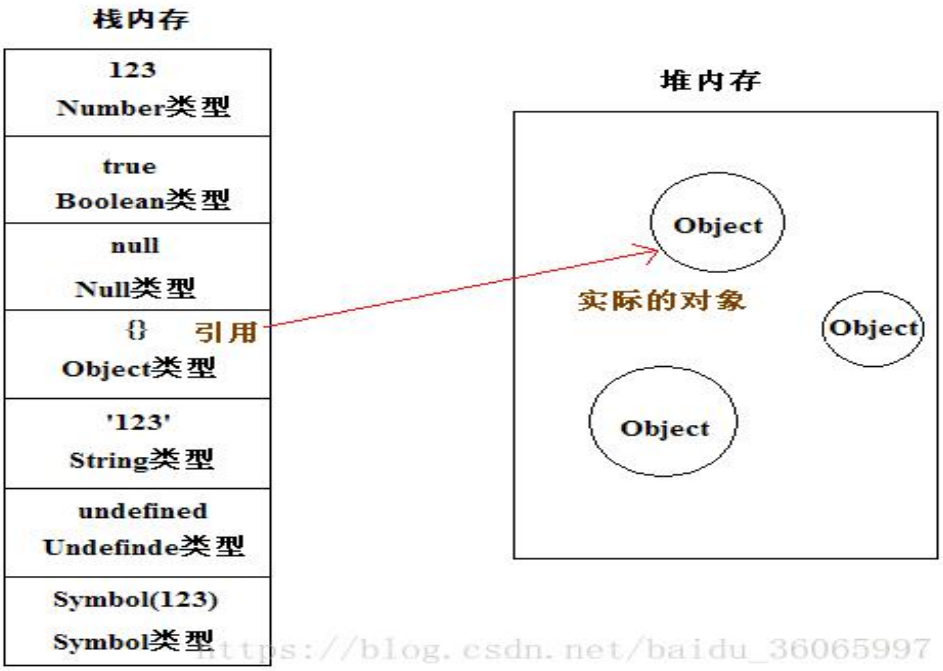
引用类型的值是可变的：
person1['name'] = 'muwoo'
console.log(person1) // {name: 'muwoo'}

3. 基本类型 和 引用数据类型 的相关区别

javascript 的变量的存储方式：栈内存（stack）和堆内存（heap）

栈：自动分配内存空间，系统自动释放，里面存放的是基本类型的值和引用类型的地址

堆：动态分配的内存，大小不定，也不会自动释放。里面存放引用类型的值。



基本数据类型是按值访问的，因为可以操作保存在变量中的实际的值。
引用类型的值是保存在内存中的对象。JavaScript 不允许直接访问内存中的位置，也就是说不能直接操作对象的内存空间。 在操作对象时， 实际上是在操作对象的引用而不是实际的对象。

传值与传址

基本类型与引用类型最大的区别实际就是 传值与传址 的区别

值传递：基本类型采用的是值传递。
地址传递：引用类型则是地址传递，将存放在栈内存中的地址赋值给接收的变量。

在我们进行赋值操作的时候，**基本数据**类型的赋值（=）是在内存中新开辟一段栈内存，然后再将值赋值到新的栈中。例如:

```
var num1 = 5;
var num2 = num1;
num1 ++ ;
console.log(num1 ); // 5
console.log(num2 ); // 6
```

执行结果:

也就是说，基本类型的复制就是在栈内存中开辟出了一个新的存储区域用来存储新的变量，这个变量有它自己的值，只不过和前面的值一样，所以如果其中一个的值改变，不会影响到另一个。

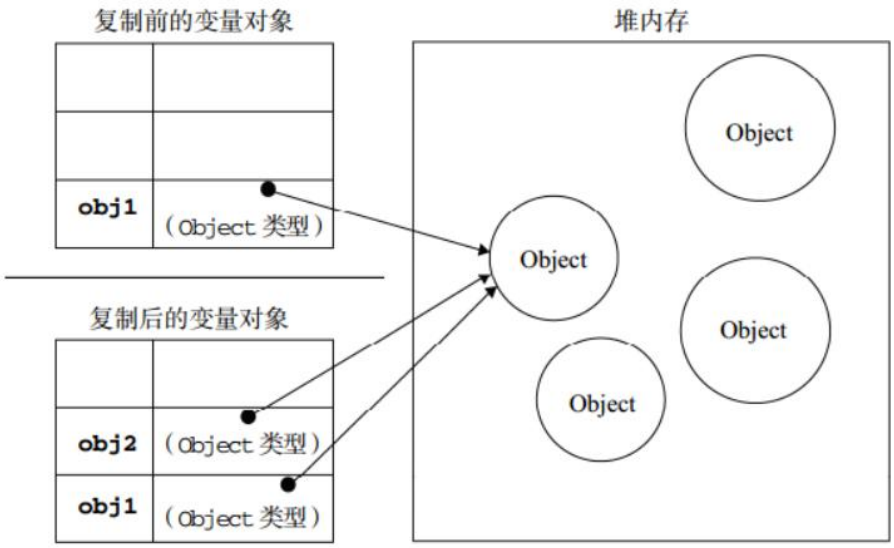
所以说，基本类型的赋值的两个变量是两个独立相互不影响的变量。

引用类型：

```
var obj1= new Object();
var obj2= obj1;
```

obj2 复制了 obj1 的值，但是结果却不是像基本类型复制一样的

执行结果：

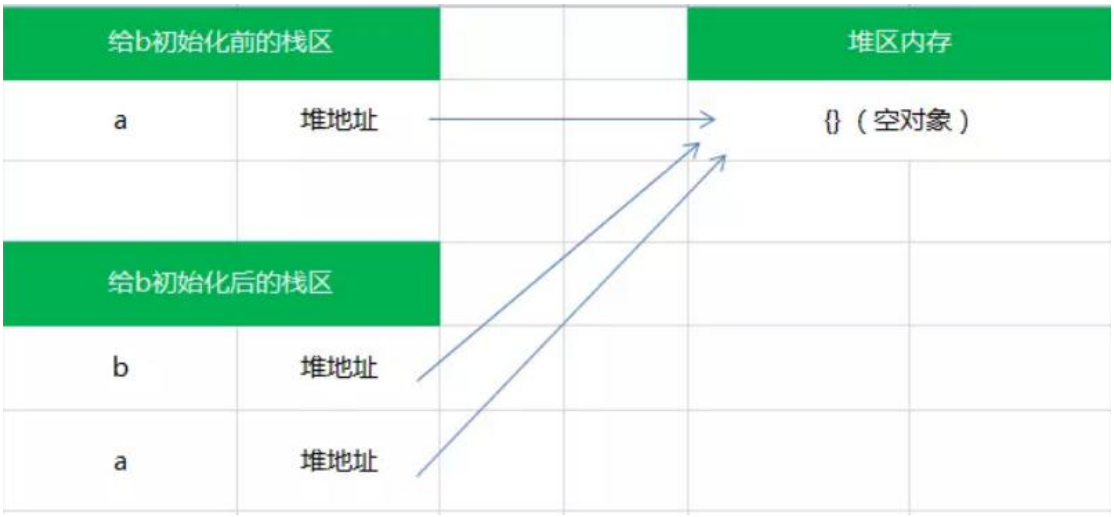


定义的对象其实是在栈内存中存储了一个指针，这个指针指向堆内存中该对象的存储地址。复制给另一个对象的过程其实是把该对象的地址复制给了另一个对象变量，两个指针都指向同一个对象，所以若其中一个修改了，则另一个也会改变。

```
obj1.name = 'laney';
alert(obj1.name);   //'laney'
```

引用类型的赋值是传址。只是改变指针的指向，例如，也就是说引用类型的赋值是对象保存在栈中的地址的赋值，这样的话两个变量就指向同一个对象，因此两者之间操作互相有影响。例如：

```
var a={};
var b=a;
```



浅拷贝只复制指向某个对象的指针，而不复制对象本身，新旧对象还是共享同一块内存。但深拷贝会另外创造一个一模一样的对象，新对象跟原对象不共享内存，修改新对象不会改到原对象。

深拷贝和浅拷贝的区分

对象的拷贝又分为浅拷贝和深拷贝：

浅拷贝和深拷贝只针对 object 和 Array 这样的复杂的对象

所谓的浅拷贝，只会将对象的各个属性进行依次复制，并不会进行递归复制，也就是说只会复制目标对象的第一层属性，这种拷贝就叫做浅拷贝

而深拷贝，要求要复制一个复杂的对象，这就可以利用递归的思想来实现，省性能又不会发生引用，它不仅将原对象的各个属性逐个复制出去，而且将原对象各个属性所包含的对象也依次采用深复制的方法递归复制到新对象上。

一句话总结，浅拷贝对于第一层属性值为引用类型数据的时候，不做处理，直接复制引用地址，深拷贝会开辟新的栈，生成新的数据。

浅拷贝

浅拷贝对于目标对象第一层为基本数据类型的数据，就是直接赋值，即「传值」；而对于目标对象第一层为引用数据类型的数据，就是直接赋存于栈内存中的堆内存地址,即「传址」,并没有开辟新的栈，也就是复制的结果是两个对象指向同一个地址，修改其中一个对象的属性，则另一个对象的属性也会改变。

一句话总结，浅拷贝对于第一层属性值为引用类型数据的时候，不做处理，直接复制引用地址，深拷贝会开辟新的栈，生成新的数据。

对象的浅拷贝(只拷贝一层)

先来看一段代码的执行:

```
var obj = {a: 1, b: {c: 2}}
var obj1 = obj
var obj2 = shallowCopy(obj);
function shallowCopy(src) {
var dst = {};
        for (var prop in src) {
                if (src.hasOwnProperty(prop)) {
     dst[prop] = src[prop];
}
}
 return dst;
 }
var obj3 = Object.assign({}, obj)
var {...obj4}= obj;  //扩展运算符
obj.a = 2
obj.b.c = 3
console.log(obj) // {a: 2, b: {c: 3}}
console.log(obj1) // {a: 2, b: {c: 3}}
console.log(obj2) // {a: 1, b: {c: 3}}
console.log(obj3) // {a: 1, b: {c: 3}}
```

- obj1： 只是将指针改变，其引用的仍然是同一个对象
- obj2： 重新创建了新对象，但是，如果原对象 obj 中存在另一个对象，则不会对对象做另一次拷贝，而是只复制其变量对象的地址。这是因为浅拷贝只复制一层对象的属性，并不包括对象里面的为引用类型的数据，这就是是进行浅拷贝了一层
- obj3: Object.assign 合并对象

Object.assign 拷贝的属性是有限制的，只拷贝源对象的自身属性（不拷贝继承属性），也不拷贝不可枚举的属性（enumerable: false）

法一：for 循环

```
var obj  = {
   name:"zs",
   age:"18",
   gender:"男",
   hobby:["吃饭","唱歌","爬山"]
   }

var newobj = {};
for(var key in obj){
   newobj[key] = obj[key];
   }
console.log(newobj);
newobj.hobby.push("旅游");
console.log(obj)
```

法二：使用 Object.assign()方法

```
var obj  = {
   name:"zs",
   age:"18",
   gender:"男",
   hobby:["吃饭","唱歌","爬山"]}
var newobj = Object.assign({},obj)
console.log(newobj);
newobj.hobby.push("旅游")
console.log(obj)
```

法三：扩展运算符，浅拷贝

```
var obj  = {
   name:"zs",
   age:"18",
   gender:"男",
   hobby:["吃饭","唱歌","爬山"]
   };
var newobj = {...obj}
console.log(newobj);
newobj.hobby.push("旅游")
console.log(obj)
```

数组的浅拷贝

对于数组，更常见的浅拷贝方法便是 `slice(0)`和 `concat()`

```
var arr = [1,2,3];
var arr01 =  arr ;
var arr02 = arr. slice(0)
var arr03 = arr. concat([])
```

法一：for 循环

法二： slice

利用数组自身的方法，slice、concat 方法在运行后会返回新的数组

```
let arr1 = [1,2,3];
let arr2 = arr1.slice(0);
```

法三： concat

```
let arr1 = [1,2,3];
let arr2 = arr1.concat();
```

法四：扩展运算符

```
let arr1 = [1,2,3];
let [...arr2] = arr1;
```

法五：Array.from

如果参数是一个真正的数组，Array.from 会返回一个一模一样的新数组

```
let arr1 = [1,2,3];
let arr2 = Array.from(arr1);
```

深拷贝的实现方式

对象的深拷贝（拷贝多层）

深拷贝的复制则是开辟新的栈，两个对象对应两个不同的地址，修改一个对象的属性，不会改变另一个对象的属性。

也就是说，深拷贝是对对象以及对象的所有子对象进行拷贝。那么如何实现这样一个深拷贝呢?

法一：JSON.parse(JSON.stringify(obj))

```
let obj1={count:1,name:'grace',age:1};

let obj2 = JSON.parse(JSON.stringify(obj1));
```

```
var obj  = {
   name:"zs",
   age:"18",
   gender:"男",
   hobby:["吃饭","唱歌","爬山"]
   }
var newobj = JSON.parse(JSON.stringify(obj));// 修改 newobj 中的 hobby
newobj.hobby.push("旅游")// 观察 obj 中的 hobby 的变化。
console.log(obj)
```

//但是这种方法也有不少坏处，譬如它会抛弃对象的 constructor。也就是深拷贝之后，不管这个对象原来的构造函数是什么，在深拷贝之后都会变成 Object。要复制的 function 会直接消失，所以这个方法只能用在单纯只有数据的对象

```
var target = {
a: 1,
b: 2,
hello: function() {
console.log("Hello, world!");
}
};
var copy = JSON.parse(JSON.stringify(target));
console.log(copy); // {a: 1, b: 2}
console.log(JSON.stringify(target)); // "{"a":1,"b":2}"
```

法二：遍历递归实现属性拷贝

既然浅拷贝只能实现非 object 第一层属性的复制，那么遇到 object 只需要通过递归实现浅拷贝其中内部的属性即可：

```
var obj  = {
        name:"zs",
        age:"18",
        gender:"男",
        friends:{
            1:'aa',
            2:'bb',
            3:'cc'
        },
        hobby:["吃饭","唱歌","爬山"]
    };
```

多次循环

```
var newObj={};
    for(var key in obj) {
```

```
        if(typeof obj[key]=='object'){
            newObj[key]=[];
            for(var key2 in obj[key]) {
                if(typeof obj[key][key2]=='object'){
                    console.log('op');
                } else {
                    newObj[key][key2] = obj[key][key2];
                }
            }

        } else {
            newObj[key] = obj[key];
        }

    }
```

法二：递归拷贝

```
function extendDeep (source) {
  var target;
  if (typeof source !== 'object') {
     target = source;
     return target

   }
  target = Array.isArray(source) ? [] : {} ;
  for (var key in source) {
      if (typeof source[key] !== 'object') {
        target[key] = source[key]
      } else {
        target[key] = extendDeep(source[key])
      }

    }

  return target
 }
```

面试题

```
<script>
var a = {"x": 1};
```

```
var b = a;
a.x = 2;
console.log(b.x);


a = {"x": 3};
console.log(b.x);
a.x = 4;
console.log(b.x);
</script>

var a = 1
var b = a
b = 2
```
请问 a 显示是几？

```
var a = {name: 'a'}
var b = a
b = {name: 'b'}
```
请问现在 a.name 是多少？