12-11

简单归纳

深拷贝与浅拷贝的概念只存在于 Object、Array、Function、RegExp、Date 等

对象的深拷贝与浅拷贝的区别如下：

浅拷贝：仅仅复制对象的引用，而不是对象本身；

深拷贝：把复制的对象所引用的全部对象都复制一遍。

浅拷贝是拷贝了对象的引用，当原对象发生变化的时候，拷贝对象也跟着变化；

深拷贝是另外申请了一块内存，内容和原对象一样，更改原对象，拷贝对象不会发生变化

但是面试官可能这么说：浅拷贝是拷贝一层，深层次的对象级别的就拷贝引用；深拷贝是拷贝多层，每一级别的数据都会拷贝出来；

浅拷贝方法

1. 直接将值传递
2. Object.assign 方法

浅拷贝的时候如果数据是基本数据类型，那么就直接赋值那种，会拷贝其本身，如果除了基本数据类型之外还有一层对象，那么对于浅拷贝而言就只能拷贝其引用，对象的改变会反应到拷贝对象上；

深拷贝方法

深拷贝就会拷贝多层，即使是嵌套了对象，也会都拷贝出来。

Js 自带的深拷贝方法

1. Array slice()、concat、Array.from()、... 操作符：只能实现一维数组的深拷贝
2. Object Object.assign()：只能实现一维对象的深拷贝
3. JSON.parse(JSON.stringify(obj))：可实现多维对象的深拷贝，但会忽略 undefined、任意的函数、symbol 值

进行 JSON.stringify()序列化的过程中，undefined、任意的函数以及 symbol 值，在序列化过程中会被忽略（出现在非数组对象的属性值中时）或者被转换成 null（出现在数组中时）。

实现深拷贝常用递归复制。

直接使用var newObj = Object.create(oldObj)，可以达到深拷贝的效果。

Object.create() es6 创建对象的另一种方式，可以理解为继承一个对象，添加的属性是在原型下。

浅拷贝

就是把父对象的属性，全部拷贝给子对象。

```
var Chinese = {
nation:'中国'
};
Var Doctor={
career:'医生'
};
function extendCopy(p) {
var c = {};
for (var i in p) {
c[i] = p[i];
}
c.uber = p; return c;
}
```

//使用的时候，这样写：

Doctor = extendCopy(Chinese);

Doctor.career ='医生' ;

alert(Doctor.nation); // 中国

但是，这样的拷贝有一个问题。那就是，如果父对象的属性等于数组或另一个对象，那么实际上，子对象获得的只是一个内存地址，而不是真正拷贝，因此存在父对象被篡改的可能。

//现在给 Chinese 添加一个"出生地"属性，它的值是一个数组。

Chinese.birthPlaces=['北京','上海','香港'];

//通过 extendCopy()函数，Doctor 继承了 Chinese。

Doctor= extendCopy(Chinese);

//然后，我们为 Doctor 的"出生地"添加一个城市：

Doctor.birthPlaces.push('厦门') ;

//看一下输入结果

alert(Doctor.birthPlaces); //北京，上海，香港，厦门 alert(Chinese.birthPlaces); //北京，上海，香港，厦门

//结果是两个的出生地都被改了。

//所以，extendCopy()只是拷贝了基本类型的数据，我们把这种拷贝叫做"浅拷贝〃。

深拷贝

就是能够实现真正意义上的数组和对象的拷贝。只要递归调用"浅拷贝"就行了。

```
var Chinese = {
  nation: "中国",
};
var Doctor = {
  career: "医生",
};
function deepCopy(p, c) {
  var c = c || {};
  for (var i in p) {
    if (typeof p[i] === "object") {
      c[i] = p[i].constructor === Array ? [] : {};
      deepCopy(p[i], c[i]);
    } else {
      c[i] = p[i];
    }
  }
  return c;
}
```

//看一下使用方法：

Doctor = deepCopy(Chinese);

//现在，给父对象加一个属性，值为数组。然后，在子对象上修改这个属性：

Chinese.birthPlaces=['北京','上海','香港'];

Doctor.birthPlaces.push('厦门') ;

alert(Doctor.birthPlaces); //北京，上海，香港，厦门

alert(Chinese.birthPlaces); //北京，上海，香港

JavaScript 中的对象一般是可变的（Mutable),因为使用了引用赋值，新的对象简单的引用了原始对象，改变新的对象将影响到原始对象。如'foo={a: 1}; bar=foo; bar.a=2'你会发现此时 'foo.a'也被改成了 '2'。

虽然这样做可以节约内存，但当应用复杂后，这就造成了非常大的隐患，Mutable 带来的优点变得得不偿失。

为了解决这个问题，一般的做法是使用 shallowCopy (浅拷贝）或 deepCopy (深拷贝）来避免被修改，但这样做造成了 CPU 和内存的浪费。

解释

基本数据类型：名值存储在栈内存中；

引用数据类型：名存在栈内存中，值存在于堆内存中，但是栈内存会提供一个引用的地址指向堆内存中的值。

目前基本数据类型有：Boolean、Null、Undefined、Number、String、Symbol，引用数据类型有：Object、Array、Function、RegExp、Date 等

深拷贝与浅拷贝的概念只存在于引用数据类型