JavaScript部分

1. 原型与闭包

闭包是指有权访问另外一个函数作用域中的变量的函数

参考：

深入理解JavaScript原型与闭包

让你分分钟理解 JavaScript 闭包

全面理解Javascript闭包和闭包的几种写法及用途

1. 判断一个变量是不是对象非常简单。值类型(undefined,null, number, string, boolean)的类型判断用typeof，引用类型(函数、数组、对象)的类型判断用instanceof，一切引用类型都是对象，对象是属性的集合。

```
typeof null     //object
null instanceof Object   //false
```

2. 对象都是通过函数创建的,函数是对象

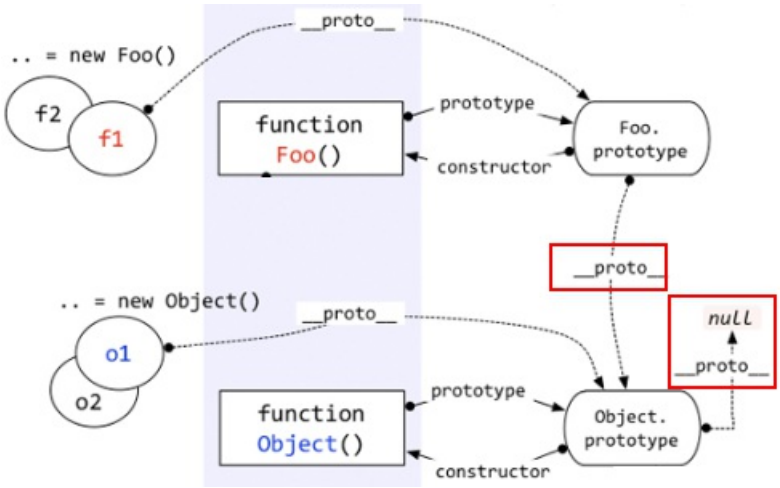
```
//  var obj = { a: 10, b: 20 };
var obj = new Object();
obj.a = 10;
obj.b = 20;
console.log(typeof (Object));  // function
```

3. 每个函数都有一个属性叫做prototype。这个prototype的属性值是一个对象（属性的集合，再次强调！），默认的只有一个叫做constructor的属性，指向这个函数本身。

```
function Fn() { }
   Fn.prototype.name = '王福朋';
   Fn.prototype.getYear = function () {
   return 1988;
};
var fn = new Fn();
console.log(fn.name);
console.log(fn.getYear());
```

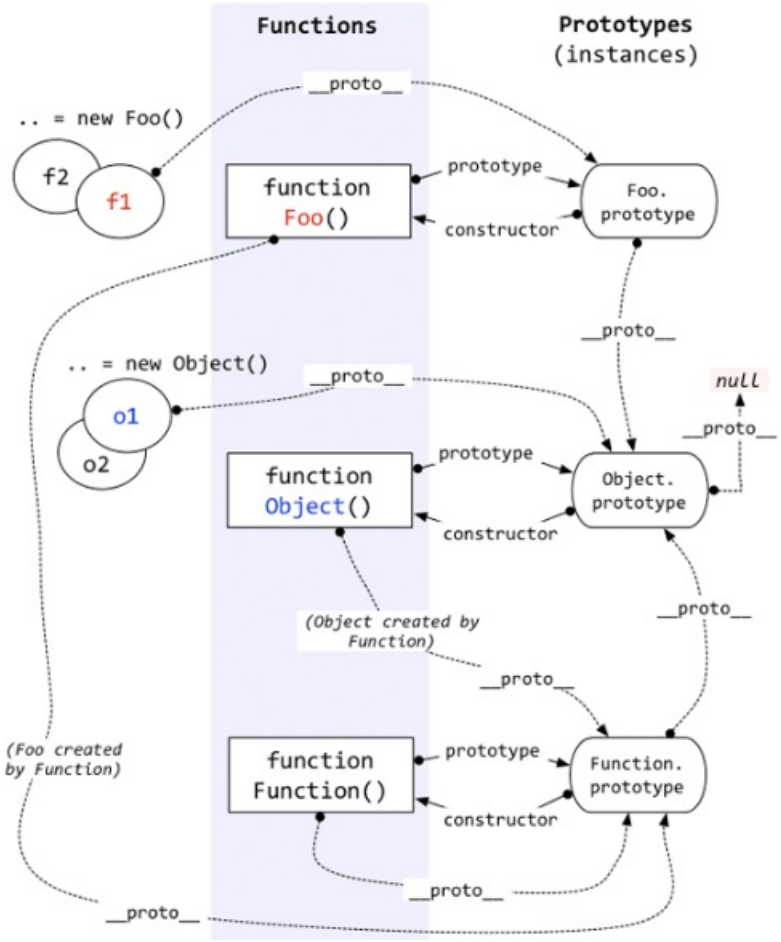
Fn是一个函数，fn对象是从Fn函数new 出来的，这样fn对象就可以调用Fn.prototype中的属性。因为每个对象都有一个隐藏的属性——“**proto**”，这个属性引用了创建这个对象的函数的prototype。即：fn.**proto** === Fn.prototype

4. 每个函数function都有一个prototype,每个对象都有一个**proto**,称为隐式原型。特例：Object.prototype是对象，它的**proto**指向的是null



5. A instancee of B,沿着A的**proto**这条线来找，同时沿着B的prototype这条线来找，如果两条线能找到同一个引用，即同一个对象，那么就返回true。如果找到终点还未重合，则返回false。

```
console.log(Object instanceof Function) //true
console.log(Function instanceof Object) //true
console.log(Function instanceof Function) //true
```



6. 变量、函数表达式——变量声明，默认赋值为undefined；this——赋值；函数声明——赋值；

```
console.log(f1)    //function f1(){}
function f1(){}    //函数声明
console.log(f2)    //undefined
var f2=function(){}    //函数表达式
```

7. this

1情况1：构造函数

```
function Foo(){
   this.name="haha";
   this.year=1998;
   console.log(this);
}
var f1=new Foo();  // Foo&amp;nbsp;{name: "haha", year: 1998}
Foo();     //Window&amp;nbsp;{postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window,&amp;nbsp;…}
```

情况2：函数作为对象的一个属性

```
var obj={
   x:10,
   fn:function(){console.log(this);console.log(this.x);}
}
obj.fn()     // {x: 10, fn: ƒ}   10
var fn1=obj.fn;
fn1();      //Window&amp;nbsp;{postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window,&amp;nbsp;…}   undefined
```

情况3：函数用call或者apply调用

```
var obj={
   x:10
}
var fn=function(){console.log(this);console.log(this.x);}
fn.call(obj)      //Object {x:10}  10
```

情况4：全局 & 调用普通函数

```
var obj={
   x:10,
   fn:function(){
      function f(){
         console.log(this);
         console.log(this.x);
      }
      fn();       //Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}   undefined
   }
}
```

8. javascript没有块级作用域，javascript除了全局作用域之外，只有函数可以创建的作用域。作用域最大的用处就是隔离变量，不同作用域下同名变量不会有冲突。

9. 从自由变量到作用域链。要到创建这个函数的那个作用域中取值——是“创建”，而不是“调用”

```
var a=10;
function fn(){var b=20; function bar(){console.log(a+b);} return bar;}
var x=fn();
var b=200;
var x();    //30
```

10. 闭包的两种应用————函数作为返回值，函数作为参数传递。

2. 引起JavaScript内存泄漏的操作有哪些

虽然JavaScript 会自动垃圾收集，但是如果我们的代码写法不当，会让变量一直处于“进入环境”的状态，无法被回收。

1. 全局变量引起的内存泄漏

2. 闭包引起的内存泄漏

3. dom清空或删除时，事件未清除导致的内存泄漏

4. 子元素存在引用引起的内存泄漏

5. 被遗忘的计时器

参考：

JavaScript深入之4类常见内存泄漏及如何避免

【译】JavaScript 内存泄漏问题

JavaScript 常见的内存泄漏原因

3. 如何实现ajax请求

(1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.

```
var xmlHttpRequest;  //定义一个变量,用于存放XMLHttpRequest对象
function createXMLHttpRequest()    //创建XMLHttpRequest对象的方法
{
   if(window.ActiveXObject){   //判断是否是IE浏览器
      xmlHttpRequest = new ActiveXObject("Microsoft.XMLHTTP");  //创建IE浏览器中的XMLHttpRequest对象
   }
   else if(window.XMLHttpRequest)    //判断是否是Netscape等其他支持XMLHttpRequest组件的浏览器
   {
      xmlHttpRequest = new XMLHttpRequest();  //创建其他浏览器上的XMLHttpRequest对象
   }
}
```

(2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.

```
XMLHttpRequest.open(method,URL,flag,name,password)
```

(3)设置响应HTTP请求状态变化的函数.

(4)发送HTTP请求.

(5)获取异步调用返回的数据.

(6)使用JavaScript和DOM实现局部刷新.

参考：

实现AJAX的基本步骤

w 3school AJAX教程

5. 简要介绍ES6

1. 新的变量声明方式 let/const，其中最重要的两个特性就是提供了块级作用域与不再具备变量提升。同时不能重复声明。

2. 解构赋值

```
let [a, [[b], c]] = [1, [[2], 3]];
let [a = 1, b] = []; // a = 1, b = undefined
let [a, ...b] = [1, 2, 3];     //a=1,b=[2,3]
//解构默认值
let [a = 3, b = a] = [];         // a = 3, b = 3
let [a = 3, b = a] = [1];        // a = 1, b = 1
let [a = 3, b = a] = [1, 2];      // a = 1, b = 2
let {a = 10, b = 5} = {a: 3};    // a = 3; b = 5;
```

3. ES6 引入了一种新的原始数据类型 Symbol，表示独一无二的值，最大的用法是用来定义对象的唯一属性名。Symbol 函数栈不能用 new 命令，因为 Symbol 是原始数据类型（null,undefined,Boolean,String,Number,Symbol），不是对象。

4. ES6对字符串、 数组、正则、对象、函数等拓展了一些方法

5. 为解决异步回调问题，引入了promise和 generator

   Promise 异步操作有三种状态：pending（进行中）、fulfilled（已成功）和 rejected（已失败）。除了异步操作的结果，任何其他操作都无法改变这个状态。

   Promise 对象只有：从 pending 变为 fulfilled 和从 pending 变为 rejected 的状态改变。只要处于 fulfilled 和 rejected ，状态就不会再变了即 resolved（已定型）。

   then 方法接收两个函数作为参数，第一个参数是 Promise 执行成功时的回调，第二个参数是 Promise 执行失败时的回调，两个函数只会有一个被调用。

   ES6 新引入了 Generator 函数，可以通过 yield 关键字，把函数的执行流挂起，为改变执行流程提供了可能，从而为异步编程提供解决方案。

6. 实现Class和模块，通过Class可以更好的面向对象编程，使用模块加载方便模块化编程，当然考虑到浏览器兼容性，我们在实际开发中需要使用babel进行编译。

   ES6 的模块自动开启严格模式，不管你有没有在模块头部加上 use strict;。

   模块中可以导入和导出各种类型的变量，如函数，对象，字符串，数字，布尔值，类等。

   每个模块都有自己的上下文，每一个模块内声明的变量都是局部变量，不会污染全局作用域。

   每一个模块只加载一次（是单例的）， 若再去加载同目录下同文件，直接从内存中读取。

参考：

ES6教程

6. 对JS模块化的理解

在ES6出现之前，js没有标准的模块化概念，这也就造成了js多人写作开发容易造成全局污染的情况，以前我们可能会采用立即执行函数、对象等方式来尽量减少变量这种情况，后面社区为了解决这个问题陆续提出了**AMD规范**和**CMD规范**，这里不同于Node.js的 CommonJS的原因在于服务端所有的模块都是存在于硬盘中的，加载和读取几乎是不需要时间的，而浏览器端因为加载速度取决于网速，因此需要采用异步加载，*AMD规范中使用define来定义一个模块，使用require方法来加载一个模块，现在ES6也推出了标准的模块加载方案，通过export和import来导出和导入模块*。

7. 介绍事件代理以及好处

事件委托就是利用事件冒泡，只指定一个事件处理程序，就可以管理某一类型的所有事件。好处是可以减少事件绑定，同时动态的DOM结构仍然可以监听。事件代理发生在冒泡阶段。

参考：

js中的事件委托或是事件代理详解

8. 使用new操作符实例化一个对象的具体步骤

1.构造一个新的对象

2.将构造函数的作用域赋给新对象（也就是说this指向了新的对象）

3.执行构造函数中的代码

4.返回新对象

9. js如何判断网页中图片加载成功或者失败

使用onload事件运行加载成功，使用onerror事件判断失败

10. 递归和迭代

```
// 迭代，利用变量的原值推算出变量的一个新值，迭代就是A不停的调用B.
function iteration(x){
   var sum=1;
   for(x; x>=1; x--){
       sum = sum*x;
   }
}
// 递归：程序调用自身的编程技巧称为递归。
function recursion(x){
   if(x>1){
       return x*recursion(x-1);
   }else{
       return 1;
   }
}
```

1） 递归中一定有迭代,但是迭代中不一定有递归,大部分可以相互转换。

2） 能用迭代的不用递归,递归调用函数,浪费空间,并且递归太深容易造成堆栈的溢出./*相对*/

参考：

深究递归和迭代的区别、联系、优缺点及实例对比

「递归」和「迭代」有哪些区别?

11. 策略模式

意图：定义一系列的算法,把它们一个个封装起来, 并且使它们可相互替换。

主要解决：在有多种算法相似的情况下，使用 if…else 所带来的复杂和难以维护。

优点： 1、算法可以自由切换。 2、避免使用多重条件判断。 3、扩展性良好。

缺点： 1、策略类会增多。 2、所有策略类都需要对外暴露。

使用场景： 1、如果在一个系统里面有许多类，它们之间的区别仅在于它们的行为，那么使用策略模式可以动态地让一个对象在许多行为中选择一种行为。 2、一个系统需要动态地在几种算法中选择一种。 3、如果一个对象有很多的行为，如果不用恰当的模式，这些行为就只好使用多重的条件选择语句来实现。

参考：

策略模式

12. 浏览器事件循环机制

JavaScript引擎是单线程，也就是说每次只能执行一项任务，其他任务都得按照顺序排队等待被执行，只有当前的任务执行完成之后才会往下执行下一个任务。

Javascript 有一个 main thread 主线程和 call-stack 调用栈(执行栈)，所有的任务都会被放到调用栈等待主线程执行。

1.JS 调用栈是一种后进先出的数据结构。当函数被调用时，会被添加到栈中的顶部，执行完成之后就从栈顶部移出该函数，直到栈内被清空。

2.JavaScript 单线程中的任务分为同步任务和异步任务。同步任务会在调用栈中按照顺序排队等待主线程执行，异步任务则会在异步有了结果后将注册的回调函数添加到任务队列(消息队列)中等待主线程空闲的时候，也就是栈内被清空的时候，被读取到栈中等待主线程执行。任务队列是先进先出的数据结构。

3.调用栈中的同步任务都执行完毕，栈内被清空了，就代表主线程空闲了，这个时候就会去任务队列中按照顺序读取一个任务放入到栈中执行。每次栈内被清空，都会去读取任务队列有没有任务，有就读取执行，一直循环读取-执行的操作，就形成了事件循环。

4.定时器会开启一条定时器触发线程来触发计时，定时器会在等待了指定的时间后将事件放入到任务队列中等待读取到主线程执行。定时器指定的延时毫秒数其实并不准确，因为定时器只是在到了指定的时间时将事件放入到任务队列中，必须要等到同步的任务和现有的任务队列中的事件全部执行完成之后，才会去读取定时器的事件到主线程执行，中间可能会存在耗时比较久的任务，那么就不可能保证在指定的时间执行。

```
console.log(1);
setTimeout(function() {
    console.log(2);
})
var promise = new Promise(function(resolve, reject) {
    console.log(3);
    resolve();
})
promise.then(function() {
    console.log(4);
})
console.log(5);           //1,3,5,4,2
```

参考：

js浏览器事件循环机制

13. 原生js操作Dom的方法有哪些？

获取节点的方法getElementById、getElementsByClassName、getElementsByTagName、 getElementsByName、querySelector、querySelectorAll

对元素属性进行操作的 getAttribute、 setAttribute、removeAttribute方法

对节点进行增删改的appendChild、insertBefore、replaceChild、removeChild、 createElement等

14. typeof操作符返回值有哪些，对undefined、null、NaN使用这个操作符分别返回什么

typeof的返回值有undefined、boolean、string、number、object、function、symbol。对undefined 使用返回undefined、null使用返回object，NaN使用返回number。

15. 实现一个类型判断函数，需要鉴别出基本类型、function、null、NaN、数组、对象？

只需要鉴别这些类型那么使用typeof即可，要鉴别null先判断双等判断是否为null，之后使用typeof判断，如果是obejct的话，再用Array.isArray判断是否为数组，如果是数字再使用isNaN判断是否为NaN,（需要注意的是NaN并不是JavaScript数据类型，而是一种特殊值）如下：

```
function type(ele) {
  if(ele===null) {
    return null;
  } else if(typeof ele === 'object') {
    if(Array.isArray(ele)) {
      return 'array';
    } else {
      return typeof ele;
    }
  } else if(typeof ele === 'number') {
    if(isNaN(ele)) {
      return NaN;
    } else {
      return typeof ele;
    }
  } else{
    return typeof ele;
  }
}
```

16. javascript做类型判断的方法有哪些？

1. typeof

```
typeof '';       // string 有效
typeof 1;        // number 有效
typeof Symbol(); // symbol 有效
typeof true;     //boolean 有效
typeof undefined; //undefined 有效
typeof null;     //object 无效
typeof [] ;      //object 无效
typeof new Function();      // function 有效
typeof new Date();          //object 无效
typeof new RegExp();        //object 无效
```

2. instanceof ,instanceof 检测的是原型

```
instanceof (A,B) = {
    var L = A.__proto__;
    var R = B.prototype;
    if(L === R) {
        // A的内部属性 __proto__ 指向 B 的原型对象
        return true;
    }
    return false;
}
```

3. constructor

当一个函数 F被定义时，JS引擎会为F添加 prototype 原型，然后再在 prototype上添加一个 constructor 属性，并让其指向 F 的引用。当执行 var f = new F() 时，F被当成了构造函数，f 是F的实例对象，此时 F 原型上的 constructor 传递到了 f 上，因此 f.constructor == F

```
' '.constructor== String
```

4. tostring

toString() 是 Object 的原型方法，调用该方法，默认返回当前对象的 [[Class]] 。这是一个内部属性，其格式为 [object Xxx] ，其中 Xxx 就是对象的类型。

```
Object.prototype.toString.call('') ;   // [object String]
Object.prototype.toString.call(1) ;    // [object Number]
Object.prototype.toString.call(true) ; // [object Boolean]
Object.prototype.toString.call(Symbol()); //[object Symbol]
Object.prototype.toString.call(undefined) ; // [object Undefined]
Object.prototype.toString.call(null) ; // [object Null]
Object.prototype.toString.call(new Function()) ; // [object Function]
Object.prototype.toString.call(new Date()) ; // [object Date]
Object.prototype.toString.call([]) ; // [object Array]
Object.prototype.toString.call(new RegExp()) ; // [object RegExp]
Object.prototype.toString.call(new Error()) ; // [object Error]
Object.prototype.toString.call(document) ; // [object HTMLDocument]
Object.prototype.toString.call(window) ; //[object global] window 是全局对象 global 的引用
```

参考：

判断js数据类型的四种方法

17.JavaScript严格模式下有哪些不同？

- 不允许不使用var关键字去创建全局变量，抛出ReferenceError
- 不允许对变量使用delete操作符，抛ReferenceError
- 不可对对象的只读属性赋值，不可对对象的不可配置属性使用delete操作符，不可为不可拓展的对象添加属性，均抛TypeError
- 对象属性名必须唯一
- 函数中不可有重名参数
- 在函数内部对修改参数不会反映到arguments中
- 淘汰arguments.callee和arguments.caller
- 不可在if内部声明函数
- 抛弃w ith语句

参考：

1.javascript高级程序设计

18.setTimeout和setInterval的区别，包含内存方面的分析？

setTimeout表示间隔一段时间之后执行一次调用，而setInterval则是每间隔一段时间循环调用，直至clearInterval结束。

内存方面，setTimeout只需要进入一次队列，不会造成内存溢出，setInterval因为不计算代码执行时间，有可能同时执行多次代码，
导致内存溢出。

参考：

setTimeout和setInterval从入门到精通

19. 同源策略是什么？

同源策略是指只有具有相同源的页面才能够共享数据，比如cookie，同源是指页面具有相同的协议、域名、端口号，有一项不同就不是同源。 有同源策略能够保证w eb网页的安全性。

DOM 同源策略：禁止对不同源页面 DOM 进行操作。这里主要场景是 iframe 跨域的情况，不同域名的 iframe 是限制互相访问的。

XMLHttpRequest 同源策略：禁止使用 XHR 对象向不同源的服务器地址发起 HTTP 请求。

跨域解决方法： 1. CORS（跨域资源共享）； 2. JSONP跨域； 3. 图像ping跨域； 4. 服务器代理； 5. document.domain 跨域； 6. w indow .name 跨域； 7. location.hash 跨域； 8. postMessage 跨域；

浏览器同源策略及跨域的解决方法

20. ES6之前JavaScript如何实现继承？

ES6之前的继承是通过**原型**来实现的，也就是每一个构造函数都会有一个prototype属性，然后如果我们调用一个实例的方法或者属性，首先会在自身寻找，然后在构造函数的prototype上寻找，而prototype本质上就是一个实例，因此如果prototype上还没有则会往prototype上的构造函数的prototype寻找，因此实现继承
可以让构造函数的prototype是父级的一个实例就是以实现继承。

21. 如何阻止事件冒泡和默认事件？

标准的DOM对象中可以使用事件对象的stopPropagation()方法来阻止事件冒泡，但在IE8以下中IE的事件对象通过设置事件对象的cancelBubble属性为true来阻止冒泡；默认事件的话通过事件对象的preventDefault()方法来阻止，而IE通过设置事件对象的returnValue属性为false来阻止默认事件。

22. addEventListener有哪些参数？

有三个参数，第一个是事件的类型，第二个是事件的回调函数，第三个是一个表示事件是冒泡阶段还是捕获阶段捕获的布尔值，true表示捕获，false表示冒泡。

23. 如何实现懒加载？

当访问一个页面的时候，先把img元素或是其他元素的背景图片路径替换成一张大小为1*1px图片的路径（这样就只需请求一次），只有当图片出现在浏览器的可视区域内时，才设置图片正真的路径，让图片显示出来。这就是图片懒加载

参考：

滚动加载图片（懒加载）实现原理

24. 什么是函数节流和函数去抖？

函数节流就是让一个函数无法在很短的时间间隔内连续调用，而是间隔一段时间执行，这在我们为元素绑定一些事件的时候经常会用到，比如我们
为w indow 绑定了一个resize事件，如果用户一直改变窗口大小，就会一直触发这个事件处理函数，这对性能有很大影响。

函数去抖是在我们事件结束后的一段时间内才会执行，会有一个延迟性。现在我们有一个需求，有一个输入框要求输入联想，在用户输入的过程中，需要按照一定的时间像后台发送ajax请求，获取数据。对于这样的需求，我们可以通过函数节流来实现

参考：

JS函数节流

25.请说一下实现jsonp的实现思路？

jsonp的原理是使用script标签来实现跨域，因为script标签的的src属性是不受同源策略的影响的，因此可以使用其来跨域。一个最简单的jsonp就是创建一个script标签，设置src为相应的url，在url之后添加相应的callback，格式类似于
url?callback=xxx，服务端根据我们的callback来返回相应的数据，类似于res.send(req.query.callback + '('+ data + ')')，这样就实现了一个最简单的jsonp

参考：

jsonp原理详解

26.浏览器内核有哪些？分别对应哪些浏览器？

常见的浏览器内核有Trident、Gecko、WebKit、Presto，对应的浏览器为Trident对应于IE，Gecko对应于火狐浏览器，Webkit有chrome和safari，Presto有Opera。

27.什么是深拷贝，什么是浅拷贝？

浅拷贝是指仅仅复制对象的引用，而不是复制对象本身；深拷贝则是把复制对象所引用的全部对象都复制一遍。

深拷贝方法1：递归复制所有层级的属性

```
function deepClone(obj){
    let objClone = Array.isArray(obj)?[]:{};
    if(obj && typeof obj==="object"){
        for(key in obj){
            if(obj.hasOwnProperty(key)){
                //判断ojb子元素是否为对象，如果是，递归复制
                if(obj[key]&&typeof obj[key] ==="object"){
                    objClone[key] = deepClone(obj[key]);
                }else{
                    //如果不是，简单复制
                    objClone[key] = obj[key];
                }
            }
        }
    }
    return objClone;
}
```

深拷贝方法2：借用JSON对象的parse和stringify

```
function deepClone(obj){
    let _obj = JSON.stringify(obj),
        objClone = JSON.parse(_obj);
    return objClone
}
```

深拷贝方法3：借用JQuery的extend方法。

$.extend([deep], target, object1 [, objectN])

deep表示是否深拷贝，为true为深拷贝，为false，则为浅拷贝

target Object类型 目标对象，其他对象的成员属性将被附加到该对象上。

object1 objectN可选。 Object类型 第一个以及第N个被合并的对象。

```
let a=[0,1,[2,3],4],
b=$.extend(true,[],a);
```

深拷贝与浅拷贝的区别，实现深拷贝的几种方法

28.原生js字符串方法有哪些？

简单分为获取类方法，获取类方法有charAt方法用来获取指定位置的字符，获取指定位置字符的unicode编码的charCodeAt方法，与之相反的fromCharCode方法，通过传入的unicode返回字符串。查找类方法有indexof()、lastIndexOf()、search()、match()方法。截取类的方法有substring、slice、substr三个方法，其他的还有replace、split、toLow erCase、toUpperCase方法。

29.原生js字符串截取方法有哪些？有什么区别？

js字符串截取方法有substring、slice、substr三个方法，substring和slice都是指定截取的首尾索引值，不同的是传递负值的时候substring会当做0来处理，而slice传入负值的规则是-1指最后一个字符，substr方法则是第一个参数是开始截取的字符串，第二个是截取的字符数量，和slice类似，传入负值也是从尾部算起的。

30.let和const的异同有哪些？

let和const都是对变量的声明，都有块级作用域的概念，不同的是const是对常量的声明，因此声明同时必须赋值，且之后不能更改，而let则可以。

31.将静态资源放在其他域名的目的是什么？

这样做的主要目的是在请求这些静态资源的时候不会发送cookie，节省了流量，需要注意的是cookie是会发送给子域名的（二级域名），所以这些静态资源是不会放在子域名下的，而是单独放在一个单独的主域名下。同时还有一个原因就是浏览器对于一个域名会有请求数的限制，这种方法可以方便做CDN。

参考：

为什么淘宝、腾讯等会把静态资源放在另外一个主域名下？

为什么很多网站的静态资源会使用独立的域名？

32.如何实现对一个DOM元素的深拷贝，包括元素的绑定事件？

```
//使用cloneNode，但是在元素上绑定的事件不会拷贝
function clone(origin) {
    return Object.assign({},origin);
}
//实现了对原始对象的克隆，但是只能克隆原始对象自身的值，不能克隆她继承的值，如果想要保持继承链，可以采用如下方法：
function clone(origin) {
    let originProto=Object.getPrototypeOf(origin);
    return Object.assign(Object.create(originProto),origin);
}
```

参考：

如何实现对一个DOM元素的深拷贝，包括元素的绑定事件？

33.简要介绍一下WebPack的底层实现原理

webpack是把项目当作一个整体，通过给定的一个主文件，webpack将从这个主文件开始找到你项目当中的所有依赖的文件，使用loaders来处理它们，最后打包成一个或多个浏览器可识别的js文件.

参考：

webpack简单原理以及实现方法

webpack中文网

34.ajax的readyState有哪几个状态，含义分别是什么？

ajax的readyState共有5个状态，分别是0-4，其中每个数字的含义分别是0代表还没调用open方法，1代表的是未调用send方法，也就是还没发送数据给服务器
2代表的是还没有接收到响应，3代表的是开始接收到了部分数据，4代表的是接收完成，数据可以在客户端使用了。

35.AMD和CMD的区别？

最明显的区别就是在模块定义时对依赖的处理不同

AMD推崇依赖前置，在定义模块的时候就要声明其依赖的模块

CMD推崇就近依赖，只有在用到某个模块的时候再去require

参考：

前端模块化，AMD与CMD的区别

36.SVG和Canvas的区别

Canvas：依赖分辨率；不支持事件处理器；弱的文本渲染能力；能够以 .png 或 .jpg 格式保存结果图像；最适合图像密集型的游戏，其中的许多对象会被频繁重绘
SVG：不依赖分辨率；支持事件处理器；最适合带有大型渲染区域的应用程序（比如谷歌地图）；复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）；不适合游戏应用

37.对于ES7和ES8了解多少

参考：

10分钟学会ES7+ES8