# 一、JavaScript

1. 原始值和引用值类型及区别

数据转换时为true / false的情况：

|  |  |  |
| --- | --- | --- |
| 数据类型 | 转换为true | 转换为false |
| boolean | true | false |
| String | 任何非空字符串 | 空字符串 |
| Number | 任何非零数字值 | 0和NaN |
| Object | 任何对象 | null |
| Undefined | n/a | undefined |
| Array | []为true… | 无 |

2. 判断数据类型typeof、instanceof、Object.prototype.toString.call()、constructor

## 3. 类数组与数组的区别与转换：

类数组：

1. 拥有length属性，其它属性（索引）为非负整数（对象中的索引会被当做字符串来处理）;
2. 不具有数组所具有的方法；
3. 类数组是一个普通对象，而真实的数组是Array类型。

常见的类数组有: 函数的参数arugments, DOM对象列表(比如通过 document.querySelectorAll 之类的方法得到的列表),jQuery 对象 (如 $("div")).

转换方法：

//第一种方法(推荐)

Array.prototype.slice.call(arrayLike, start);或者写成：[].slice.call()

//第二种方法：展开运算符

[...arrayLike];

//第三种方法: Array.from方法用于将两类对象转为真正的数组：类似数组的对象（array-like object）和可遍历（iterable）的对象。

Array.from(arrayLike);

区分数组与类数组的方法：

1.instanceof 2.constructor 3.toString（） 4.ES 提供的方法 isArray（）

使用typeOf都会返回object所以不能使用typeOf来区分。

## 4. 数组的常见API

判断数组的方法有：

* xx instanceof Array
* Array.isArray()
* Object.prototype.toString.call() === '[object Array]'

instanceOf判断数组时在多全局对象（多个frame跨框架或多个window）中会返回false，并不能正确判断是否为数组，所以判断是否为数组最好用后两种方法。

of()类似于new Array()，但后者如果传入一个参数，则是设置数组长度。

Array.of(1,2,3,4); *// [ 1, 2, 3, 4 ]*

indexOf 和 includes

indexOf() 返回索引，不存在就返回 -1。inclues()返回布尔值。

NaN 不能通过indexOf()判断，它是通过“===”比较的。

arr = [1, '2', null, NaN];

arr.indexOf(NaN); *// -1*

arr.includes(NaN); *// true*

filter、find 和 findIndex

filter(callback(currentValue,index))：筛选. 返回函数会返回 true 的项组成的数组。

find() 返回值，不存在就返回 undefined。 findIndex() 返回第一个匹配到的索引，不存在就返回 -1。

let arr = [1, '2', null, NaN];

arr.filter (item => typeof item === 'number'); // [1, NaN]

arr.find (item => typeof item === 'number'); // 1

arr.findIndex (item => typeof item === 'number'); // 0

some 和 every: 都返回布尔值

some()对数组中的每一项运行给定函数，如果该函数对任一项返回true，则返回 true；

every()对数组中的每一项运行给定函数，如果函数对每一项都返回true，则返回 true。

不对空数组进行检测，不改变原数组

map 和 forEach：

map() 对数组中的每个元素进行处理，返回每次函数调用的结果组成的新数组。不改变原数组，

let res = [0,1,2,3,4].map((item, idx)=> item \* idx); *// [ 0, 1, 4, 9, 16 ]*

forEach() 相当于 for 循环，返回 undefined，不改变原数组

map有返回值，forEach没有返回值；要forEach能返回值出来，在它内部定义一个变量存储遍历的值就可以了。

归并方法Reduce(): reduce() 从前往后迭代，reduceRight() 则从后往前

reduce((accumulator, currentValue, currentIndex, array)=>{}, initValue)

* 第一个参数是迭代器函数，函数的作用是对数组中从左到右的每一个元素进行处理。第二个可选参数是累加器的初始值。没有时，累加器第一次的值为currentValue。
* accumulator 累加器，即函数上一次调用的返回值。第一次的时候为 initialValue || arr[0]
* currentValue 数组中函数正在处理的的值，第一次的时候是 initialValue || arr[1]
* currentIndex 数组中函数正在处理的的索引
* array :函数调用的数组

var numbers = [65, 44, 12, 4];

function getSum(total, num) { return total + num; }

function myFunction(item) { document.getElementById("demo").innerHTML = numbers.reduce(getSum); }

// 结果为125

## 5. bind、call、apply的区别

这三者都是用来改变this的指向（执行上下文）；call()、apply()可以看做是某个对象的方法，通过调用方法的形式来间接的调用函数。bind()就是将某个函数绑定到某个对象上。

call()：

第一个参数是this要指向的对象，后面传入的是参数列表，参数可以是任意类型，当第一个参数为null、undefined的时候，默认指向window；

apply()：

第一个参数是this要指向的对象，第二个参数是数组；

apply可以将一个数组转换成一个参数列表（[param1,param2,param3]）转换成(param1,param2,param3)，

  var arr = ['1','2','3','4'];        Math.max.apply(0,arr);

bind()：

和call很相似，第一个参数是this的指向，从第二个参数开始是接收的参数列表。区别在于bind方法返回值是函数以及bind接收的参数列表的使用，

bind是返回对应的函数，便于稍后调用，而不是直接执行函数；apply，call则是立即调用。

   在 ES6 的箭头函数下, call 和 apply 将失效, 对于箭头函数来说:

箭头函数体内的 this 对象, 就是定义时所在的对象, 而不是使用时所在的对象;所以不需要类似于var \_this = this这种丑陋的写法；

箭头函数不可以当作构造函数，也就是说不可以使用 new 命令, 否则会抛出一个错误

箭头函数不可以使用 arguments 对象,，该对象在函数体内不存在. 如果要用, 可以用 Rest 参数代替，

不可以使用 yield 命令, 因此箭头函数不能用作 Generator 函数

## 6. new的原理

原型（prototype）:一个简单的对象，用于实现对象的属性继承。可以简单理解成对象的父亲。,在火狐和谷歌中，每个js对象中都包含一个\_\_proto\_\_(非标准)的属性指向它构造函数的原型（该对象原型），可obj.\_\_proto\_\_进行访问。

构造函数：可以通过new来新建一个对象的函数。

实例：通过构造函数 和 new 创建出来的对象，就是实例。实例通过\_\_proto\_\_指向原型，通过constructor指向构造函数 。

new创建对象，就是调用构造函数来实例化，在调用构造函数时会执行以下操作：

1. 创建一个新对象
2. 将构造函数的作用域赋给新对象(因此 this 就指向了这个新对象)
3. 执行构造函数中的代码(为这个新对象添加属性)
4. 返回新对象

代码实现：

function news(func) {

var target = {};

target.\_\_proto\_\_ = func.prototype;

// var newObj = Object.create(fun.prototype)

//前两步或可用这句代码代替,意为创建一个新对象newObj，并让 newObj.\_\_proto\_\_ 指向 fun

var res = func.call(target);

if (typeof (res) == "object" || typeof (res) == "function") {

return res;

}

return target;

}

//生成新对象

//实例的\_\_proto\_\_指向原型，构造函数的prototype也指向原型（链接到原型）

//把函数的this绑定在了新生成的对象中

//如果传入的函数(构造函数)有自己的返回值，则返回该值

//如果如果传入的函数(构造函数)没有自己的返回值，则返回新对象

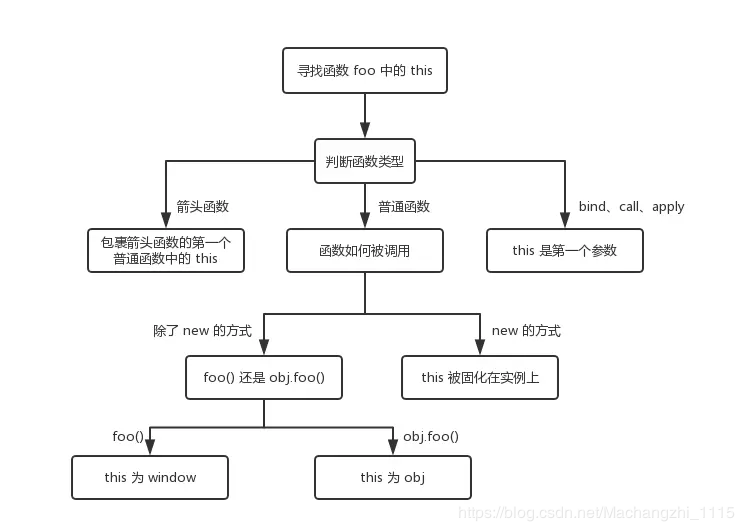
## 7. 如何正确判断this？

this的指向在函数定义的时候是确定不了的，只有***函数执行的时候才能确定***this指向谁

全局环境下：

无论是否在严格模式下，在全局执行环境中（在任何函数体外部）this 都指向全局对象。

在函数运行环境内：

1. 简单调用（此调用未设置this）  
   非严格模式下，this 的值默认指向全局对象。  
   在严格模式下，this将保持他进入执行环境时的值。如果 this 没有被执行环境（execution context）定义，那它将保持为 undefined。
2. Bind、call、apply方法  
   ES5 引入了 Function.prototype.bind。调用f.bind(someObject)会创建一个与f具有相同函数体和作用域的函数，在这个新函数中，this将永久地被绑定到了bind的第一个参数，无论此函数是如何被调用的。当第一个参数为null、undefined的时候，默认指向window；
3. 箭头函数  
   箭头函数没有自己的this，它继承外层上下文绑定的this。在箭头函数中，this与封闭词法环境的this保持一致。在全局代码中，它将被设置为全局对象。
4. 作为对象的方法  
   当函数作为对象里的方法被调用时，它们的 this 是调用该函数的对象。（此概念适用于原型链中的this和getter 与 setter 中的 this）
5. 作为构造函数  
   当一个函数用作构造函数时（使用new关键字），它的this被绑定到正在构造的新对象。（手动返回其他对象时，则绑定到返回的这个对象上）
6. 作为一个DOM事件处理函数  
   当函数被用作事件处理函数时，它的this指向触发事件的元素（一些浏览器在使用非addEventListener的函数动态添加监听函数时不遵守这个约定）。
7. 作为一个内联事件处理函数  
   当被内联on-event 处理函数调用时，它的this指向监听器所在的DOM元素。

## 8. 闭包及其作用

闭包：有权访问其他函数内部变量的函数，在一个函数内部创建一个函数，就形成了闭包，闭包的参数和变量不会被垃圾回收机制回收。闭包可以避免全局变量的污染。

闭包是一个环境，闭包是一个拥有许多变量和绑定了这些变量的环境的表达式（通常是一个函数），因而这些变量也是该表达式的一部分。

优点: 1:变量长期驻扎在内存中；（延续局部变量的寿命。）

2:避免全局变量的污染；

3:私有成员的存在 ；

特性：1:函数套函数；

2:内部函数可以直接使用外部函数的局部变量或参数；

3:变量或参数不会被垃圾回收机制回收 GC；

缺点: 会造成内存泄漏

闭包的创建方式：

1、Module模式闭包

var testModule = (function () {

var count = 0;

function privateMethod() { }

return {//返回对象

AddCount: function () {

count++;

}

};

})();

2、直接给全局变量window添加成员创建闭包

(function () {

var count=0;

//直接给全局变量window添加成员

window.GetCount=function(){

return count;

};

})();

应用场景一：原生的setTimeout传递的第一个函数不能带参数

应用场景二：闭包可将一个变量长期驻扎在内存当中,可用于循环中存索引值

场景二具体应用：绑定事件on-event异步触发，当事件触发的时候for循环早就结束了。

应用场景三：封装变量：闭包可以帮助把一些不需要暴露在全局的变量封装成私有变量。

## 9. 原型和原型链

构造函数： 构造函数模式的目的就是为了创建一个自定义类，并且创建这个类的实例。构造函数模式中拥有了类和实例的概念，并且实例和实例之间是相互独立的，即实例识别。

构造函数就是一个普通的函数，创建方式和普通函数没有区别，不同的是构造函数习惯上首字母大写。另外就是调用方式的不同，普通函数是直接调用，而构造函数需要使用new关键字来调用。

原型：每当定义一个函数数据类型(普通函数、类)时候，都会天生自带一个prototype属性，这个属性指向函数的原型对象，并且这个属性是一个对象数据类型的值。每个原型都有一个 constructor 属性指向关联的构造函数，实例原型指向构造函数

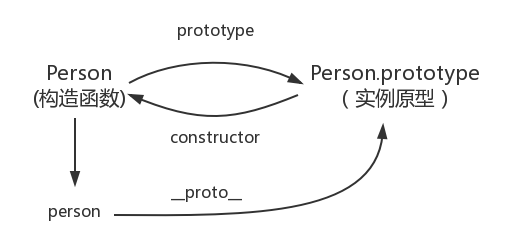
原型对象就相当于一个公共的区域，所有同一个类的实例都可以访问到这个原型对象，我们可以将对象中共有的内容，统一设置到原型对象中。

每一个对象数据类型(普通的对象、实例、)也天生自带一个属性\_\_proto\_\_，(null无原型对象没有此属性)，属性值是当前实例所属类的原型(prototype)。原型对象中有一个属性constructor, 它指向构造函数对象。

原型链：在JavaScript中万物都是对象，对象和对象之间也有关系，并不是孤立存在的。对象之间的继承关系，在JavaScript中是通过prototype对象指向父类对象，直到指向Object对象为止，这样就形成了一个原型指向的链条，专业术语称之为原型链。

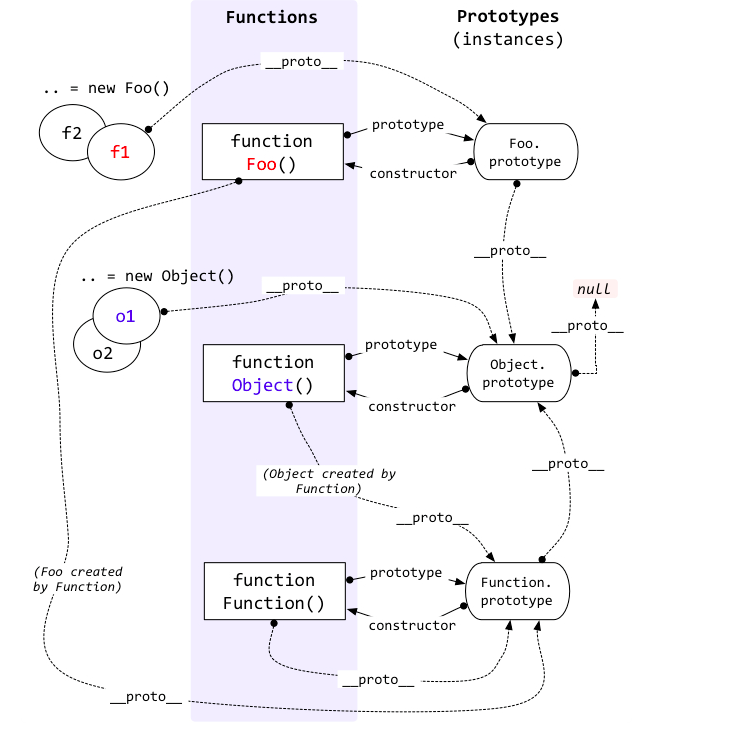
举例说明:person → Person → Object ，普通人继承人类，人类继承对象类

当我们访问对象的一个属性或方法时，它会先在对象自身中寻找，如果有则直接使用，如果没有则会去原型对象中寻找，如果找到则直接使用。如果没有则去原型的原型中寻找,直到找到Object对象的原型，Object对象的原型没有原型，如果在Object原型中依然没有找到，则返回null。以上通过\_\_proto\_\_属性来连接对象直到null的一条链即为我们所谓的**原型链。**

 我们可以使用对象的hasOwnProperty()来检查对象自身中是否含有该属性；使用in检查对象中是否含有某个属性时，如果对象中没有但是原型中有，也会返回true

Object.getPrototypeOf(object)方法返回参数对象的原型对象。

Object.keys(object)方法返回对象上可枚举的实例属性。



## 10. prototype与\_\_proto\_\_的关系与区别

prototype称为显示原型；\_\_proto\_\_可称为隐式原型

\_\_proto\_\_和constructor属性是对象独有的，对象没有prototype属性，对象的\_\_proto\_\_属性指向其构造函数的prototype。

prototype属性只有函数才有，但是由于js中函数也是一种对象，所以函数也拥有\_\_proto\_\_和constructor属性。

实例.\_\_proto\_\_===原型

原型.constructor === 构造函数

构造函数.prototype === 原型

实例.constructor === 构造函数

## 11. 继承的实现方式及比较

**1、**原型链继承

重点：让新实例的原型等于父类的实例。

特点：1、实例可继承的属性有：实例的构造函数的属性，父类构造函数属性，父类原型的属性。（新实例不会继承父类实例的属性！）

2、非常纯粹的继承关系，实例是子类的实例，也是父类的实例; 父类新增原型方法/原型属性，子类都能访问到; 简单，易于实现

缺点：1、创建子类实例时，无法向父类构造函数传参。

　2、继承单一,无法实现多继承。

　3、所有新实例都会共享父类实例的属性。（原型上的属性是共享的，一个实例修改了原型属性，另一个实例的原型属性也会被修改！）

4、要想为子类新增属性和方法，必须要在new Animal()这样的语句之后执行，不能放到构造器中

原型继承方法一: Son.prototype = Father.prototype;

原型继承方法二: Son.prototype = new Father();

function Cat(){ }

Cat.prototype = new Animal();

Cat.prototype.name = 'cat' ;

var cat = new Cat();

console.log (cat instanceof Animal); //true

console.log (cat instanceof Cat); //true

2、借用构造函数继承

核心:用 .call() 和 .apply() 将父类构造函数引入子类函数（在子类函数中做了父类函数的自执行（复制）（没用到原型）

特点： 1、解决了1中，子类实例共享父类引用属性的问题,解决了原型继承123的缺点.

2、创建子类实例时，可以向父类传递参数

3、可以实现多继承（call多个父类对象）

缺点： 1、实例并不是父类的实例，只是子类的实例

2、只能继承父类的实例属性和方法，不能继承原型属性/方法

3、无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

function Cat (name) {

Animal.call(this);

this.name = name || 'Tom';

}

var cat = new Cat();

console.log (cat instanceof Animal); // false

console.log (cat instanceof Cat); // true

3、实例继承

核心：为父类实例添加新特性，作为子类实例返回

特点： 1、不限制调用方式，不管是new 子类()还是子类(),返回的对象具有相同的效果

缺点： 1、实例是父类的实例，不是子类的实例

2、不支持多继承

function Cat(name){

var instance = new Animal();

instance.name = name || 'Tom';

return instance;

}

var cat = new Cat();

console.log (cat instanceof Animal); // true

console.log (cat instanceof Cat); // false

4、拷贝继承

特点： 1、支持多继承

缺点： 1、效率较低，内存占用高（因为要拷贝父类的属性）

2、无法获取父类不可枚举的方法（不可枚举方法，不能使用for in 访问到）

function Cat(name){

var animal = new Animal();

for (var p in animal ) {

Cat.prototype[p] = animal[p];

}

Cat.prototype.name = name || 'Tom';

}

var cat = new Cat();

console.log (cat instanceof Animal); // false

console.log (cat instanceof Cat); // true

5、组合继承（组合原型链继承和借用构造函数继承）

核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用

特点： 1、弥补了方式2的缺陷，可以继承实例属性/方法，也可以继承原型属性/方法

2、既是子类的实例，也是父类的实例

3、不存在引用属性共享问题,即每个新实例引入的构造函数属性是私有的。

4、可传参; 5、函数可复用

缺点： 1、调用了两次父类构造函数，生成了两份实例（子类实例将子类原型上的那份屏蔽了）（耗内存），子类的构造函数会代替原型上的那个父类构造函数。

function Cat(name){

Animal.call (this) ; this.name = name || 'Tom';

}

Cat.prototype = new Animal();

var cat = new Cat();

console.log (cat instanceof Animal); // true

console.log (cat instanceof Cat); // true

6、寄生组合继承

核心：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性，避免的组合继承的缺点

特点： 堪称完美：可以多重继承 解决两次调用 解决实例共享引用类型的问题 原型链保持不变

缺点： 实现较为复杂

function Cat (name) {

Animal.call (this) ;

this.name = name || 'Tom';

}

( function() { // 创建一个没有实例方法的类

var Super = function () { };

Super.prototype = Animal.prototype;

//将实例作为子类的原型

Cat.prototype = new Super () ;

} ) ();

var cat = new Cat();

console.log ( cat instanceof Animal ); // true

console.log ( cat instanceof Cat ); // true

12. 深拷贝与浅拷贝：

深拷贝：

方法一：循环递归去复制所有层级属性。遍历对象、数组直到里边都是基本数据类型，然后再去复制，就是深度拷贝。思路如下：

1、声明一个变量 来储存我们拷贝出来的内容;

2、判断 obj 是否是引用类型数据，如果不是，则直接赋值即可

3、根据判断的不同类型，再给之前的变量赋予不同的类型： [ ] : { }

4、循环obj 中的每一项，如果里面还有复杂数据类型，则利用递归再次调用copy函数

5、最后将这个变量 return 出来即可.

方法二：jquery.extend()：$.extend( [deep ], target, object1 [, objectN ] )

方法三：JSON.stringify()和JSON.parse()：

原理： 用JSON.stringify将对象转成JSON字符串，再用JSON.parse()把字符串解析成对象，一去一来，新的对象产生了，而且对象会开辟新的栈，实现深拷贝。

缺陷：虽然可以实现数组或对象深拷贝,但不能处理函数。因为JSON.stringify()方法是将一个JavaScript值(对象或者数组)转换为一个JSON字符串,不接受函数。

首层浅拷贝：对目标对象的第一层进行深拷贝，后面的是浅拷贝，可以称作“首层浅拷贝”。

slice() concat() …展开运算符 Object.assign()

## 13. 防抖和节流

在进行窗口的resize、scroll，输入框内容校验等操作时，（滚动事件做复杂计算频繁调用回调函数很可能会造成页面的卡顿，这时候希望把多次计算合并成一次，只操作一个精确点）如果事件处理函数调用的频率无限制，会加重浏览器的负担，导致用户体验非常糟糕。此时可以采用防抖和节流的方式来减少调用频率，同时又不影响实际效果。

函数防抖（debounce）：当持续触发事件时，一定时间段内没有再触发事件，事件处理函数才会执行一次，如果设定的时间到来之前，又一次触发了事件，就重新开始延时。

function debounce(fn, wait) {

var timeout = null; //定义一个定时器

return function() {

if(timeout !== null) {

clearTimeout(timeout); //清除这个定时器 }

timeout = setTimeout(fn, wait);

}

}

// 处理函数

function handle() { console.log(Math.random()); }

// 滚动事件添加监听机制

window.addEventListener ('scroll', debounce(handle, 1000));

函数节流（throttle）：当持续触发事件时，保证在一定时间内只调用一次事件处理函数，意思就是说，假设一个用户一直触发这个函数，且每次触发小于既定值，函数节流会每隔这个时间调用一次。以下代码为时间戳 + 定时器：

var throttle = function(func, delay) {

var timer = null;

var startTime = Date.now(); //设置开始时间

return function() {

var curTime = Date.now();

var remaining = delay - (curTime - startTime); //剩余时间

var context = this;

var args = arguments;

clearTimeout(timer);

if (remaining <= 0) { // 第一次触发立即执行

func.apply(context, args);

startTime = Date.now();

} else { //取消当前计数器并计算新的remaining

timer = setTimeout(func, remaining); } }

}

function handle() { console.log(Math.random()); }

window.addEventListener('scroll', throttle(handle, 1000));

总结:

函数防抖：将几次操作合并为一次操作进行。原理是维护一个计时器，规定在delay时间后触发函数，但是在delay时间内再次触发的话，就会取消之前的计时器而重新设置。这样一来，只有最后一次操作能被触发。

函数节流：使得一定时间内只触发一次函数。原理是通过判断是否到达一定时间来触发函数。

区别： 函数节流不管事件触发有多频繁，都会保证在规定时间内一定会执行一次真正的事件处理函数，而函数防抖只是在最后一次事件后才触发一次函数。 比如在页面的无限加载场景下，我们需要用户在滚动页面时，每隔一段时间发一次 Ajax 请求，而不是在用户停下滚动页面操作时才去请求数据。这样的场景，就适合用节流技术来实现。

14. 作用域和作用域链、执行期上下文：

作用域[[scope]]:是变量和函数的可访问范围，控制着变量和函数的可见性与生命周期，即作用域决定了代码区块中变量和其他资源的可见性。在JavaScript中变量的作用域有全局作用域和局部作用域。JavaScript采用词法作用域(lexical scoping)，也就是静态作用域。

作用域在函数定义时就已经确定了，而不是在函数调用时确定。作用域有上下级的关系，上下级关系的确定就看函数是在哪个作用域下创建的。

词法作用域：词法作用域是指在词法分析阶段就确定了，不会改变。变量的作用域是在定义时决定而不是执行时决定，也就是说词法作用域取决于源码，通过静态分析就能确定，因此词法作用域也叫做静态作用域。

动态作用域：动态作用域是在运行时根据程序的流程信息来动态确定的，而不是在写代码时进行静态确定的。 动态作用域并不关心函数和作用域是如何声明以及在何处声明的，只关心它们在何处调用。

作用域链：JavaScript 中每个函数都都表示为一个函数对象（函数实例），函数对象有一个仅供 JavaScript 引擎使用的[[scope]] 属性。通过语法分析和预解析，将[[scope]] 属性指向函数定义时作用域中的所有对象集合。这个集合被称为函数的作用域链（scope chain），包含函数定义时作用域中所有可访问的数据。[[scope]]中所存储的执行期上下文对象的集合，这个集合呈链式链接，我们把这种链式链接叫做作用域链。

执行期上下文（ Execution Context）:当函数执行时，会创建一个称为执行期上下文的内部对象,执行期上下文定义了一个函数正在执行时的环境，执行期上下文会根据不同的运行环境变化。而且函数每执行一次，都会创建单独的 执行期上下文，所以函数每次执行时对应的执行上下文都是独一无二的，所以多次调用一个函数会导致创建多个执行上下文，当函数执行完毕，执行期上下文被销毁。（闭包除外）

无论有多少个函数上下文，但是全局上下文只有一个。执行期上下文有创建和代码执行的两个阶段。  
15. DOM常见的操作方式:

DOM 是 Document Object Model（文档对象模型）的缩写。 DOM是中立于平台和语言的接口，它允许程序和脚本动态地访问和更新文档的内容、结构和样式。 在 HTML DOM中，所有事物都是节点。DOM 是被视为节点树的 HTML。

1.查找节点:

* document.getElementById('id属性值'); 返回拥有指定id的第一个对象的引用
* document/element.getElementsByClassName('class属性值'); 返回拥有指定class的对象集合
* document/element.getElementsByTagName('标签名'): 返回拥有指定标签名的对象集合
* document.getElementsByName('name属性值'); 返回拥有指定名称的对象结合
* document/element.querySelector('CSS选择器'); 仅返回第一个匹配的元素
* document/element.querySelectorAll('CSS选择器'); 返回所有匹配的元素
* document.documentElement; 获取页面中的HTML标签
* document.body; 获取页面中的BODY标签
* document.all['']; 获取页面中的所有元素节点的对象集合型

2．创建节点

* document.createElement('元素名'); 创建新的元素节点
* document.createAttribute('属性名'); 创建新的属性节点
* document.createTextNode('文本内容'); 创建新的文本节点
* document.createComment('注释节点'); 创建新的注释节点
* document.createDocumentFragment( ); 创建文档片段节点

3．删除节点

* parentNode.removeChild( existingChild );删除已有的子节点，返回值为删除节点
* element.removeAttribute('属性名');删除具有指定属性名称的属性，无返回值
* element.removeAttributeNode( attrNode );删除指定属性，返回值为删除的属性
* 修改节点

4．修改节点

* parentNode.replaceChild( newChild, existingChild );用新节点替换父节点中已有的子节点
* element.setAttributeNode( attributeName );若原元素已有该节点，此操作能达到修改该属性值的目的
* element.setAttribute( attributeName, attributeValue );若原元素已有该节点，此操作能达到修改该属性值的目的

5．插入节点

* parent.appendChild( element/txt/comment/fragment );向父节点的最后一个子节点后追加新节点
* parent.insertBefore( newChild, existingChild );向父节点的某个特定子节点之前插入新节点
* element.setAttributeNode( attributeName );给元素增加属性节点
* element.setAttribute( attributeName, attributeValue );给元素增加指定属性，并设定属性值

6.设置样式 : ele.style.styleName = styleValue;设置ele元素的CSS样式

DOM事件：允许 JavaScript 对 HTML 事件作出反应

onclick 事件——当用户点击时

onload 事件——用户进入 onunload 事件——用户离开

onmouseover事件——鼠标移入 onmouseout事件——鼠标移出

onmousedown事件——鼠标按下 onmouseup 事件——鼠标抬起

## 16. Array.sort()方法与实现机制

当Array.sort()不传入参数的时候，sort()方法会调用默认的排序的方式，即先调用每个数组项的toString()转型方法，然后按照字符串Unicode编码顺序来对字符串进行排序，

var arr = [1, 2, 3, 15, 22, 33];

arr.sort(); //1, 15, 2, 22, 3, 33

Array.sort(func): 传入一个函数作为参数，来指明排序方式

function func (a,b){ return a-b; // 升序 return b-a; // 降序 }

当func函数return的结果>0时，a与b会交换位置，当<=0时，不会交换位置。

实现根据数组对象中某个属性值进行排序：可以用函数作为返回值，可以嵌套一层函数用来接收对象属性名，表示按照哪个属性进行排序：

compare函数中的property只是起到了标识的作用，实际上arr中的数据是传递给了内部的function，然后由内部的function参数根据property取得需要比较的值进行比较。

var arr = [ {name:' guo ',age:0}, {name:' yu ',age:18}

{name:'liu' , age:8} ];

function compare(property){

return function( a , b ) {

var value1 = a[property]; var value2 = b[property];

return value1 - value2; } }

console.log( arr.sort ( compare( 'age' ) ) ); // liu, yu, guo的顺序

## 17. Ajax的请求过程

1. 创建XMLHttpRequest对象,也就是创建一个异步调用对象.
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.
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.

1、创建XMLHttpRequest对象：

在IE浏览器中创建XMLHttpRequest对象的方式如下所示:

var xmlHttpRequest = new ActiveXObject("Microsoft.XMLHTTP");

在Netscape浏览器中创建XMLHttpRequest对象的方式如下所示:

    var xmlHttpRequest = new XMLHttpRequest();

2、创建HTTP请求

创建HTTP请求可以使用XMLHttpRequest对象的open()方法,其语法代码如下所示:

       XMLHttpRequest.open(method,URL, async,name,password)

method：该参数用于指定HTTP的请求方法，一共有get、post、head、put、delete五种方法，常用的方法为get和post。

URL：该参数用于指定HTTP请求的URL地址，可以是绝对URL，也可以是相对URL。

async：该参数为可选参数，参数值为布尔型。该参数用于指定是否使用异步方式。true表示异步方式、false表示同步方式，默认为true。

3、设置响应HTTP请求状态变化的函数:

从创建XMLHttpRequest对象开始，到发送数据、接收数据、XMLHttpRequest对象一共会经历以下5中状态。readyState属性：

* 0 （未初始化状态）表示XMLHttpRequest对象被创建，但尚未使用xhr.open()方法
* 1 （初始化状态）表示open()方法被调用，建立了连接
* 2 （发送数据状态）表示send()方法被调用，请求已接收并取得了响应的状态值和响应头。
* 3 （接收数据状态）表示响应体正在下载中
* 4 （完成状态）表示下载体已经完成，可以直接使用responseText.

 XMLHttpRequest对象可以响应readystatechange事件，该事件在XMLHttpRequest对象状态改变时（也就是readyState属性值改变时）激发。因此，可以通过该事件调用一个函数，并在该函数中判断XMLHttpRequest对象的readyState属性值。如果readyState属性值为4则使用responseText属性或responseXml属性来获取数据。

//判断XMLHttpRequest对象的readyState属性值是否为4，为4表示异步调用完成.

if ( xmlHttpRequest.readyState == 4 ) { }

4、设置获取服务器返回数据的语句

如果XMLHttpRequest对象的readyState属性值等于4，则异步调用过程完毕，就能通过XMLHttpRequest对象的responseText属性或responseXml属性来获取数据.

       但是，异步调用过程完毕，并不代表异步调用成功了，如果要判断异步调用是否成功，还要判断XMLHttpRequest对象的status属性值，只有该属性值为200，才表示异步调用成功，因此，要获取服务器返回数据的语句，还必须要先判断XMLHttpRequest对象的status属性值是否等于200

//设置获取数据的语句 state=200说明服务器成功响应返回的结果是正确的

if( xmlHttpRequest.status == 200 || xmlHttpRequest.status == 0 ) { }

5、发送HTTP请求

发送HTTP请求可以使用XMLHttpRequest对象的send()方法，其语法代码如下所示：

       XMLHttpRequest.send(data)

data是个可选参数，如果是get请求的数据写在url中，那这里即可以使用null来替代。如果是post请求，data参数的格式与在URL中传递参数的格式类似：name=myName&value=myValue; post请求还需设置请求头

6. 使用JavaScript和DOM实现局部刷新.

Ajax请求代码：

Var xmlHttpRequest = new XMLHttpRequest();

xmlHttpRequest.open('GET','url','true');

xmlHttpRequest.send();

xmlHttp.onreadystatechange = function(){

if(xmlHttpRequest.readyState == 4) {

if(xmlHttpRequest.status == 200) { …… }

}

## 18. JS的垃圾回收机制

JS的垃圾回收机制是为了以防内存泄漏，内存泄漏的含义就是当已经不需要某块内存时这块内存还存在着，垃圾回收机制就是间歇的不定期的寻找到不再使用的变量，并释放掉它们所指向的内存。垃圾收集器会按照固定的时间间隔周期性的执行。

垃圾回收有两种方法：标记清除、引用计数。

标记清除：（更常见）工作原理：是当变量进入环境时，将这个变量标记为“进入环境”。当变量离开环境时，则将其标记为“离开环境”。标记“离开环境”的就回收内存。

工作流程：

1.   垃圾回收器，在运行的时候会给存储在内存中的所有变量都加上标记。

2.   去掉环境中的变量以及被环境中的变量引用的变量的标记。

3.   再被加上标记的会被视为准备删除的变量。因为环境中的变量已经无法访问到这些变量了

4.   垃圾回收器完成内存清除工作，销毁那些带标记的值并回收他们所占用的内存空间。

引用计数：工作原理：跟踪记录每个值被引用的次数。

工作流程：

1.    跟踪记录每个值被引用的次数，当声明了一个变量并将一个引用类型的值赋值给这个变量，这个引用类型值的引用次数就是1。

2.    同一个值又被赋值给另一个变量，这个引用类型值的引用次数加1.

3.    当包含这个引用类型值的变量又被赋值成另一个值了，那么这个引用类型值的引用次数减1.

4.    当引用次数变成0时，说明没办法访问这个值了。就可以将其所占的内存空间给回收

5.    当垃圾收集器下一次运行时，它就会释放引用次数是0的值所占的内存。

缺点：循环引用的时候就会释放不掉内存。循环引用就是对象A中包含另一个指向对象B的指针，B中也包含一个指向A的引用。因为IE中的BOM、DOM的实现使用了COM，而COM对象使用的垃圾收集机制是引用计数策略。所以会存在循环引用的问题。

解决方案：手工断开js对象和DOM之间的链接。赋值为null。

obj.property = null ; ele.property = null

减少垃圾回收：

new关键字就意味着一次内存分配，例如 new Foo()。最好的处理方法是：在初始化的时候新建对象，然后在后续过程中尽量多的重用这些创建好的对象。

1、对象object优化：

为了最大限度的实现对象的重用，应该像避使用new语句一样避免使用{}来新建对象。

有一种方式能够保证对象（确保对象prototype上没有属性）的重复利用，那就是遍历此对象的所有属性，并逐个删除，最终将对象清理为一个空对象。可以使用cr.wipe(obj)方法清理对象，再为obj添加新的属性，就可以达到重复利用对象的目的。

2、数组array优化：

　　将[]赋值给一个数组对象，是清空数组的捷径（例如： arr = [];），但是，这种方式又创建了一个新的空对象，并且将原来的数组对象变成了一小片内存垃圾！实际上，将数组长度赋值为0（arr.length = 0）也能达到清空数组的目的，并且同时能实现数组重用，减少内存垃圾的产生。

3、方法function优化：

只要是动态创建方法的地方，就有可能产生内存垃圾。例如：将方法作为返回值，就是一个动态创建方法的实例。为了解决这个问题，可以将作为返回值的方法保存起来

内存泄漏：虽然有垃圾回收机制但是我们编写代码操作不当还是会造成内存泄漏。

1.    意外的全局变量引起的内存泄漏。

原因：全局变量，不会被回收。 解决：使用严格模式避免。

2.    闭包引起的内存泄漏 原因：闭包可以维持函数内局部变量，使其得不到释放。

解决：将事件处理函数定义在外部，解除闭包,或者在定义事件处理函数的外部函数中，删除对dom的引用。

3.    没有清理的DOM元素引用

原因：虽然别的地方删除了，但是对象中还存在对dom的引用

解决：手动删除。

4.    被遗忘的定时器或者回调

原因：定时器中有dom的引用，即使dom删除了，但是定时器还在，所以内存中还是有这个dom。

解决：手动删除定时器和dom。

5.    子元素存在引用引起的内存泄漏

原因：div中的ul li  得到这个div，会间接引用某个得到的li，那么此时因为div间接引用li，即使li被清空，也还是在内存中，并且只要li不被删除，他的父元素都不会被删除。

解决：手动删除清空。

19. JS中的String、Array和Math方法

## 20. addEventListener和onClick()的区别:

onclick():

优点： - 1. 简洁 - 2. 处理事件的this关键字指向当前元素

缺点：

- 不能对事件捕获或事件冒泡进行控制，只能使用事件冒泡，无法切换成事件捕获

- 一次只能对一个元素绑定一个事件处理程序，当使用window.onload属性时，会覆盖采用相同方法所绑定的事件代码.

addEventListener(‘type’,func,false):

优点：   
- 可以支持事件处理的捕获阶段，也可以支持时间处理的冒泡阶段，最后一个参数设置为false(默认值，表示事件冒泡)或者true(表示事件捕获)来切换   
- 事件处理 this与onclick一样   
- 事件处理函数中，event对象总是作为第一个可用参数   
- 你可以为某个元素绑定多个事件而不会覆盖之前绑定的处理程序 （按照顺序执行）   
缺点：   
- IE8以下不支持 // 需要使用attachEvent() / detachEvent ( 解除事件绑定 )

addEventListener对任何DOM都是有效的，而onclick仅限于HTML

解除事件绑定:

对于onclick，直接 element.onclick = null;

对于addEventListener则需要:

Element.removeEventListener ( sEvent , fnHandler , false ) ;

如果用匿名函数的方式注册的事件，不能使用removeListener注销，因为没用对应事件的引用。所以注册事件如果需要取消，最好使用一个引用。即var fun = function(){…}

正因如此，对于一个对象多次绑定同样的func，也不会重复执行，只会执行一次。对于匿名函数，就算内容一样，也会依次执行，因为并不能算是相同事件处理器。

## 21. new和Object.create的区别:

new是js中的操作符，可以创建一个用户定义的对象类型的实例或具有构造函数的内置对象的实例。new Object()方法的实质是使用引用类型Object的构造函数创建了一个新的实例。

语法： new constructor ( [aguments] )

Object.create()是Object的内置方法，可以创建一个新对象，使用现有的对象来提供新创建的对象\_\_proto\_\_

语法：Object.create ( proto , [ propertiesObject ] )

该方法有两个参数，第一个 proto 是一个对象，作为新建对象的原型；第二个参数是一个为新对象定义额外属性的对象，该对象的属性名称是新创建的对象的属性名称。如果 propertiesObject 参数不是 null 或者一个对象，则会抛错。使用该方法，创建对象会执行以下步骤：

1. 方法内部定义一个新的空对象obj
2. 将obj.\_\_proto\_\_的对象指向传入的参数（函数/对象）的proto
3. 将传入的对象属性复制到obj并且返回obj

内部实现原理：

Object.create = function (o) {

var F = function () {};

F.prototype = o;

return new F();

};

return new F()后F被销毁，Object.create创建对象是创建一个拥有指定原型和若干个指定属性的对象，也就是说可以任意指定原型，甚至是null（Object.create(null)无原型对象）。所以在上述中像o.prototype添加属性（原型链上属性）不被继承.

区别：

|  |  |  |
| --- | --- | --- |
| 比较 | new | Object.create |
| 构造函数 | 保留原构造函数属性 | 丢失原构造函数属性 |
| 原型链 | 原构造函数prototype属性 | 原构造函数/（对象）本身 |
| 作用对象 | function | function和object |

22. BOM的location对象：

Location 对象包含有关当前 URL 的信息。

Location 对象是 Window 对象的一个部分，可通过 window.location 属性来访问。

location常用的属性如下：

hash 哈希值。设置或返回从井号 (#) 开始到 URL结束的值。可以用它来记录页面的状态。若页面存在某个链接的name的值与hash值是一样的，页面载入时，浏览器会滚动到该链接的位置。 需要注意的是，如果URL中不带#,则 hash 值为空字符串，否则，hash值为#以及到URL结束的值。

Hostname: 设置或返回当前 URL 的主机名。

Host : 设置或返回主机名和当前 URL 的端口号。

href : 设置或返回完整的 URL。

当改变location中的除了hash之外的属性的值，均会导致页面跳转。

常用的方法：

reload([bForceGet]) 刷新当前页面。bForceGet为可选参数， 默认为 false，从浏览器的缓存里取当前页。true，从服务端取最新的页面, 相当于客户端点击 F5("刷新") 。

replace(URL) 当前页面跳转到指定的URL。并且用该URL替换当前的历史纪录，这样浏览历史记录中就只有一个页面，后退按钮永远处于失效状态，用户也就无法进行后退了。

assign(URL) 当前页面跳转到指定的URL。和使用location.href=URL是一样的。会产生历史记录。

## 23. 浏览器从输入URL到页面渲染的整个流程（涉及到计算机网络数据传输过程、浏览器解析渲染过程）

1. 用户输入网址
2. 浏览器通过DNS获取网站的IP地址
3. 浏览器尝试与服务器建立连接
4. 服务器发送永久重定向
5. 浏览器跟踪重定向地址
6. 服务器处理请求
7. 服务器发送HTML响应
8. 浏览器开始显示HTML
9. 浏览器发送请求以获取嵌套在html中的元素，页面开始渲染
10. 浏览器查找域名对应的 IP 地址（域名解析的过程，先进行缓存的查看）

DNS 解析顺序：

浏览器DNS缓存->本地系统DNS缓存->本地计算机HOSTS文件->ISP DNS缓存->DNS递归搜索解析。

1. 首先会在浏览器的缓存中查找，是否缓存了URL，如果有，就直接向该URL对应的服务器发送请求；如果没有则进行下一步;
2. 在本地的hosts文件中是否保存了该URL和其对应的IP地址，如果保存了，就直接向该URL对应的服务器发送请求；如果没有则进行下一步；
3. 向本地DNS服务器发送DNS请求，本地DNS服务器会首先查询它的缓存记录，如果有就将该域名对应的IP地址返回给用户，如果没有则进行下一步；
4. 首先向根域名服务器发送DNS查询请求，根域名服务器返回给可能保存了该域名的一级域名服务器地址；本地主机再根据返回的地址，向一级域名服务器发送DNS查询请求；...一直迭代，直到找到对应的域名存放的服务器，向其发送DNS查询请求，该域名服务器返回该域名对应的IP地址；

二、建立TCP/IP连接

三次握手

三、浏览器向web服务器发送http请求

客户端请求静态资源和动态资源：

1. 静态资源：如果客户端请求的是静态资源，则web服务器根据URL地址到服务器的对应路径下查找文件，然后给客户端返回一个HTTP响应，包括状态行、响应头和响应正文。
2. 动态资源：如果客户端请求的是动态资源，则web服务器会调用CGI/VM执行程序完成相应的操作，如查询数据库，然后返回查询结果数据集，并将运行的结果--HTML文件返回给web服务器。Web服务器再将HTML文件返回给用户。

四、浏览器渲染：

浏览器拿到HTML文件后，根据渲染规则进行渲染：

1. 解析HTML，生成DOM树
2. 解析CSS，生成CSSOM树
3. 将DOM树和CSS规则树合并，生成渲染树（Render Tree）
4. 布局：结合呈现树，把dom节点的大小、位置计算出来。将计算后的大小和位置信息实装到DOM节点中，但是这时候并没有完成所有的工作。这里会涉及到回流和重绘。
5. 绘制：布局完成后，将渲染树显示到浏览器上，比如背景颜色，字体颜色等。

五、四次挥手：

## 24. 跨域、同源策略及跨域实现方式和原理

同源策略：如果两个页面的协议，端口和域名都相同，则两个页面具有相同的**源**。

缓存策略：

同源策略又分为以下两种：

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。

同源策略是浏览器的一个安全功能，不同源的客户端脚本在没有明确授权的情况下，不能读写对方资源。所以xyz.com下的js脚本采用ajax读取abc.com里面的文件数据是会被拒绝的。

同源策略限制了从同一个源加载的文档或脚本如何与来自另一个源的资源进行交互。这是一个用于隔离潜在恶意文件的重要安全机制。

不受同源策略限制的：

1. 页面中的链接，重定向以及表单提交是不会受到同源策略限制的。

2. 跨域资源的引入是可以的。但是js不能读写加载的内容。如嵌入到页面中的<script src="..."></script>，<img>，<link>，<iframe>等。

跨域实现方式：

1. jsonp请求；jsonp的原理是利用<script>标签的天然跨域特性，可以不受限制地从其他域中加载资源，类似的标签还有<img>.优点

优点：使用简便，没有兼容性问题，目前最流行的一种跨域方法。

缺点：1.只支持 GET 请求。

2.由于是从其它域中加载代码执行，因此如果其他域不安全，很可能会在响应中夹带一些恶意代码。

3.要确定 JSONP 请求是否失败并不容易。虽然 HTML5 给 script 标签新增了一个 onerror 事件处理程序，但是存在兼容性问题。

1. document.domain；这种方式用在主域名相同子域名不同的跨域访问中。非常适用于 iframe 跨域的情况。

因为浏览器是通过document.domain属性来检查两个页面是否同源，因此只要通过设置相同的document.domain，两个页面就可以共享Cookie（此方案仅限主域相同，子域不同的跨域应用场景）。 document.domain = 'test.com';

1. window.name；window的name属性有个特征：在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的，每个页面对window.name都有读写的权限，window.name是*持久存在*一个窗口载入过的所有页面中的，并不会因新页面的载入而进行重置。
2. window.postMessage；这是html5中实现跨域访问的一种新方式，可以使用它来向其它的window对象发送消息，无论这个window对象是属于同源或不同源。非常适合解决 DOM 跨域。 （还可以解决多标签通信）
3. CORS；CORS背后的基本思想，就是使用自定义的HTTP头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功还是应该失败。
4. Web Sockets；web sockets原理：在JS创建了web socket之后，会有一个HTTP请求发送到浏览器以发起连接。取得服务器响应后，建立的连接会使用HTTP升级从HTTP协议交换为web sockt协议。
5. location.hash 跨域：是子框架具有修改父框架 src 的 hash 值，通过这个属性进行传递数据，且更改 hash 值，页面不会刷新。但是传递的数据的字节数是有限的。
6. 图像 Ping 跨域：由于<img>标签不受浏览器同源策略的影响，允许跨域引用资源。因此可以通过 img 标签的 src 属性进行跨域，这也就是图像 Ping 跨域的基本原理。

优点: 用于实现跟踪用户点击页面或动态广告曝光次数有较大的优势。

缺点:1.只支持 GET 请求。

2.只能浏览器与服务器的单向通信，因为浏览器不能访问服务器的响应文本。

## 25. 浏览器的回流（Reflow）和重绘（Repaints）

浏览器下载完页面中的所有组件——HTML标记、JavaScript、CSS、图片之后会解析生成两个内部数据结构——DOM树和渲染树（在构建渲染树之前，和DOM树同期会构建Style Tree。DOM树与Style Tree合并为渲染树）.

DOM树表示页面结构，渲染树表示DOM节点如何显示。

一旦DOM和渲染树构建完成，浏览器就开始显示（绘制）页面元素。  
 当DOM的变化影响了元素的几何属性（比如宽、高或位置），导致浏览器需要重新计算元素的几何属性，其他元素的几何属性和位置也会因此受到影响。浏览器会使渲染树中受到影响的部分失效，并重新构造渲染树。这个过程称为重排reflow，重排负责元素的几何属性更新。完成重排后，浏览器会重新绘制受影响的部分到屏幕，该过程称为重绘repaint。重绘负责元素的样式更新。

注意⚠：重排一定会导致重绘，重绘不一定会发生重排。

table及其内部元素除外，它可能需要多次计算才能确定好其在渲染树中节点的属性，通常要花3倍于同等元素的时间。这也是为什么要避免使用table做布局的一个原因。

重排触发机制：

1、添加或者删除可见的DOM元素  
 2、元素位置改变  
 3、元素尺寸改变  
 4、元素内容改变（例如：一个文本被另一个不同尺寸的图片替代）  
 5、页面渲染初始化（这个无法避免）  
 6、浏览器窗口尺寸大小发生改变

触发属性：

div盒子模型：Width height padding margin display border-width border min-height

定位属性和float：top bottom left right position float clear

节点内部文字结构：text-align overflow-y font-weight overflow font-family

line-height vertival-align white-space font-size

重绘：元素的属性或者样式发生变化。

触发属性：color border-style border-radius visibility text-decoration  
background background-image background-position background-repeat  
background-size outline-color outline outline-style outline-width

box-shadow

透明度（opacity）不会触发重绘：实际上透明度的改变后，GPU在绘画时只是简单的降低之前已经画好的纹理的alpha值来达到效果，并不需要整体的重绘。不过这个前提是这个被修改opacity本身必须是一个图层，如果图层下还有其他节点，GPU也会将他们透明化。

div.style.width = '200px'; div.style.background = 'red';

div.style.height = '300px'; //久远浏览器，分别设置宽高时，会触发两次重排

大多数浏览器通过队列化修改并批量执行来优化重排过程。即以上代码只会构成一次重排。

但是还是有一些特殊的元素几何属性会造成这种优化失效。

比如：offsetTop, offsetLeft,scrollTop, scrollLeft, clientTop, clientLeft, 请求了getComputedStyle()或者 IE的 currentStyle。以上都需要实时回馈给用户的几何属性或者是布局属性，不能再依靠浏览器的优化，因此浏览器不得不立即执行渲染队列中的“待处理变化”，并随之触发重排返回正确的值。

避免重绘和重排：

将需要多次重排的元素，position属性设为absolute或fixed，这样此元素就脱离了文档流，它的变化不会影响到其他元素。例如有动画效果的元素就最好设置为绝对定位。

改变样式时：

1、使用内联样式的cssText方法：el.style.cssText = 'border-left: 1px; border-right: 2px; padding: 5px';

2、切换类名：el.className = 'active';

批量修改DOM：（以下过程引发两次重排：1和3）（fragment元素的应用）

1、让该元素脱离文档流 2、对其进行多重改变 3、将元素带回文档中

(方法：改变display、fragment文档碎片(最推荐)、克隆节点)

文档碎片的使用：var fragment = document.createDocumentFragment();

appendNode(fragment, data); ul.appendChild(fragment);

缓存布局信息：

为了尽量减少对布局信息的查询次数，查询时，将其赋值给局部变量，使用局部变量参与计算。可以避免重复取值访问DOM

强迫浏览器创建图层：（使用3d硬件，不应过多）

在Blink和WebKit的浏览器中，一当一个节点被设定了透明度的相关过渡效果或动画时，浏览器会将其作为一个单独的图层，但很多开发者使用translateZ(0)或者translate3d(0,0,0)去使浏览器创建图层。这种方式可以消除在动画开始之前的图层创建时间，使得动画尽快开始（创建图层和绘制图层还是比较慢的），而且不会随着抗锯齿而导出突变。不过这种方法需要节制，否则会因为创建过多的图层导致崩溃

Chrome中的抗锯齿：

Chrome中，非根图层以及透明图层使用grayscale antialiasing而不是subpixel antialiasing，如果抗锯齿方法变化，这个效果将会非常显著。如果你打算预处理一个节点而不打算等到动画开始，可以通过这种强迫浏览器创建图层的方式进行

修改节点位置时：

使用left和top会触发重布局，修改时的代价相当大。取而代之的更好方法是使用translate，这个不会触发重布局

26. JavaScript中的arguments

是一个对应于传递给函数的参数的类数组对象。可以使用arguments对象在函数中引用函数的参数。此对象包含传递给函数的每个参数的条目，第一个条目的索引从0开始。可以根据arguments的length属性获取传入的参数的数量.（除了length属性和索引元素之外没有任何Array属性）

ES6中的箭头函数没有自己的 arguments 对象

typeof arguments 返回object // arguments\_ proto \_ = object

callee属性（此属性在严格模式中被禁止了）：表示的是当前函数的一个引用，即这个属性里面存储的是调用这个函数的代码

可以利用arguments.callee实现递归

function factorial(num) {

if(num<=1) { return 1; }

else { return num \* arguments.callee(num-1); }

}

## 27. EventLoop事件循环

js是单线程和eventloop的循环机制。执行栈可以理解为是用来存储函数调用的栈，遵循先进后出的原则。

JavaScript 在执行时，同步任务会排好队，在主线程上按照顺序执行，前面的执行完了再执行后面的，排队的地方叫执行栈（execution context stack）。JavaScript 对异步任务不会停下来等待，而是将其挂起，继续执行执行栈中的同步任务，当异步任务有返回结果时，异步任务会加入与执行栈不一样的队列，即任务队列（task queue），所以任务队列中存放的是异步任务执行完成后的结果，通常是回调函数。

当执行栈的同步任务已经执行完成，此时主线程闲下来，它便会去查看任务队列是否有任务，如果有，主线程会将最先进入任务队列的任务加入到执行栈中执行，执行栈中的任务执行完了之后，主线程便又去任务队列中查看是否有任务可执行。主线程去任务队列读取任务到执行栈中去执行，这个过程是循环往复的，这便是 Event Loop事件循环。

Event Loop 6 个阶段：

1.timers 2.I/O callbacks 3.idle, prepare 4.poll

5.check 6.close callbacks

node中的事件循环的顺序：

外部输入数据-->轮询阶段(poll)-->检查阶段(check)-->关闭事件回调阶段(close callback)-->定时器检测阶段(timer)-->I/O事件回调阶段(I/O callbacks)-->闲置阶段(idle, prepare)-->轮询阶段...

## 28. 宏任务与微任务:

宿主环境提供的叫宏任务，由语言标准提供的叫微任务。

宿主环境：就是能使javascript完美运行的环境，只要能完美运行javascript的载体就是javascript的宿主环境。目前我们常见的两种宿主环境有浏览器和node。

语言标准：虽然JavaScript是一种编程语言，但其实JavaScript由ECMA制定标准，称之为ECMAScript,所以由语言标准提供的就是微任务，比如ES6提供的promise。

同步任务：在主线程上排队的任务，只有前一个任务执行完毕，才能执行后一个任务。

异步任务：不进入主线程，某个异步任务可以执行了，该任务才会进入主线程执行。

在执行异步任务时：异步任务又分为宏任务和微任务，微任务优先级高于宏任务。

运行机制：

1.在执行栈中执行第一个宏任务。

2. 执行过程中遇到微任务，将微任务添加到微任务队列中。

3. 当前宏任务执行完毕，立即执行微任务队列中的任务。

4. 当前微任务队列中的任务执行完毕，检查渲染，GUI线程接管渲染。

5. 渲染完毕后，js线程接管，开启下一次事件循环，执行下一次宏任务（事件队列中取）。

 微任务 (microtask) ：

* process.nextTick（优先级高于promise.then）
* promise（new Promise在实例化的过程中所执行的代码都是同步进行的， then中注册的回调才是异步执行的。）
* MutationOberver
* Object.observe

process.nextTick方法可以在当前“执行栈”的尾部——下一次Event Loop(主线程读取“任务队列”)之前触发回调函数。也就是说，它指定的任务总是发生在所有异步任务之前。

 宏任务 (macrotask) ：

* 整体javascript代码
* setTimeout(setTimeout优先级比setImmediate高)
* setInterval
* setImmediate
* I/O
* UI渲染

setImmediate方法是在当前“任务队列”的尾部添加事件，也即是说，它指定的任务总是在下一次Event Loop时执行。

requestAnimationFrame也属于异步执行的方法，但该方法既不属于宏任务，也不属于微任务。其是GUI渲染之前执行，但在微任务之后， requestAnimationFrame不一定会在当前帧必须执行，而是由浏览器根据当前的策略自行决定在哪一帧执行。

## 29. BOM属性对象方法：

1. windwo对象

alert(str): 用于向用户展示一些用户不可控的警告信息

confirm(str)：用于向用户展示一段信息并确认结果

prompt(str，str): 用于向用户展示一段信息并收集用户输入结果

print(): 显示打印对话框（等同与点击浏览器菜单栏打印选项）

find(): 显示查找对话框（等同与点击浏览器菜单栏查找选项）

1. location对象

hash：保存当前url中的哈希值（url中#号后面的任意个字符），url中不包含哈希值则返回空串

host：保存当前url中的域名和端口号

hostName：保存当前url中的域名

pathName：保存当前url中路径或者文件名

port：保存当前url中的端口号

protocol：保存当前url使用的协议

search：保存url中的查询字符串

assign(url):导航到参数url位置并生成一条历史记录（等同与 location.href=url，window.localtion=url)

replace(url):导航到参数rul位置但不生成历史记录

reload(boolean)：重新加载当前文档，传参true表示强制从服务器加载所有资源

1. navigator

userAgent：用户代理字符串，通常用来判断当前使用的是什么浏览器

plugins：保存安装所有插件信息的数组，用来判断当前用户是否安装了页面所必须的插件，IE无此属性（通过ActiveXobject对象来判断）

## 30. 函数柯里化及其通用封装

柯里化：是把接受多个参数的函数变换成接受一个单一参数（最初函数的第一个参数）的函数，并且返回接收余下的参数而且返回结果的新函数的技术。

优点： 1) 参数复用 2) 提前确认 3) 延迟运行

普通函数：function add(x, y) { return x + y } add(1,2);

柯里化之后：

function add (x) {

return function(y) {

return x + y;

}

}

add(2)(3);

通用封装：

function add(...args) {

return args.reduce((sums, i) => sums + i);

}

function carrying(fn, ...args1) {

return function (...args2) {

return fn.call(this, ...args1, ...args2);

}

}

carrying(add,1)(2);

实现bind()函数：

bind函数实际就是个现成的柯里化函数，它接收形参并返回一个新的函数，实现机制就是柯里化。

Function.prototype.bind = Function.prototype.bind ||

function (context, ...args1) {

var \_this = this

return function(...args2) {

return \_this.call(context, ...args1, ...args2)

}

}

## 31. JS的map()和reduce()方法

高阶函数：就是一个函数可以接收另一个函数作为参数（map , reduce , filter , sort）

js中的map是映射的意思，即原数组被映射成新的数组，而这个数组是由原数组中的每个元素调用一个特定的方法的返回值组成的新数组。

注意⚠： map() 不会对空数组进行检测。map() 不会改变原始数组。

比如：首先定义一个方法fun，然后map（fun），那么执行结果就是所有输入的值执行完F这个方法后产生的新数组。

var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]; // pow()自定义一个幂函数函数体省略

arr.map(pow); // [1, 4, 9, 16, 25, 36, 49, 64, 81]

js中的reduce（）方法：把函数执行后的结果再进行累加执行，这个函数必须接收两个参数，第一个参数是一个callback，用于针对各数组项的操作；第二个参数则是传入的初始值，这个初始值用于单个数组项的操作。reduce()把结果继续和序列的下一个元素做累积计算。最终计算出一个值。（详情见4）

注意⚠： reduce() 对于空数组是不会执行回调函数的。

var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]; // add表示累加函数（自定义）

arr.reduce(add,0); // 从0开始加到数组最后一个元素，输出结果：45

32. “==”和“===”的区别:

== 相等，=== 严格等于。

==， 两边值类型不同的时候，要先进行类型转换，再比较值。

===，不做类型转换，类型不同的一定不等。

特例：null == undefined //true(都表示未赋值) null === undefined //false

注意⚠：Js中 ‘/’ 运算符表示除法而不是取商！ ‘%’ 表示取余

## 33. setTimeout用作倒计时为何会产生误差？

因为 JavaScript 是一个单线程序的解释器，一定时间内只能执行一段代码。为了控制要执行的代码，就有一个 JavaScript 任务队列。这些任务会按照将它们添加到队列的顺序执行。

setTimeout() 的第二个参数告诉 JavaScript 再过多长时间把当前任务添加到队列中。如果队列是空的，那么添加的代码会立即执行；如果队列不是空的，那么它就要等前面的代码执行完了以后再执行

定时器是属于宏任务(macrotask) 。如果当前执行栈所花费的时间大于定时器时间，那么定时器的回调在 宏任务(macrotask) 里，来不及去调用，所有这个时间会有误差。

## 34.setInterval和setTimeout的性能:

setTimeout有最小时间间隔限制，HTML5标准为4ms，小于4ms按照4ms处理，但是每个浏览器实现的最小间隔都不同.setInterval最小时间间隔为10ms。

setInterval()方法会出现程序并不是按照我们设定的精确时间而调用的问题！

setTimeout递归执行的代码必须是上一次执行完了并间隔一定时间才再次执行。

比如说: setTimeout延迟时间为1秒执行, 要执行的代码需要2秒来执行,那这段代码上一次与下一次的执行时间为3秒. 而不是我们想象的每1秒执行一次.

setInterval是排队执行的:

比如说: setInterval每次执行时间为1秒,而执行的代码需要2秒执行, 那它还是每次去执行这段代码, 上次还没执行完的代码会排队, 上一次执行完下一次的就立即执行, 这样实际执行的间隔时间为2秒.

原理：当函数开始执行时在栈中创建出来一个栈帧，这个栈帧的执行是需要时间的，在这个函数运行的时候定时器是没有能力终止他的运行的，因此当函数的运行时间大于间隔时间时，就会等待函数执行完再执行下一次，导致误差。

setTimeout是先延迟后执行，setInterval是先执行，后使用时间间隔；无限次执行。

# 二、ES6 :

1. let、const和var的概念与区别

块作用域由 { } 包括，if语句和 for语句里面的{ }也属于块作用域。

Var、let、const关键字的区别：

* 1. var定义的变量，没有块的概念，可以跨块访问, 不能跨函数访问。
  2. let定义的变量，只能在块作用域里访问，不能跨块访问，也不能跨函数访问。 Let声明的变量只能在let命令所在的代码块内有效。
  3. const声明一个只读的常量（引用了一个值），使用时必须初始化(即必须赋值)，且不能修改常量的值，只能在块作用域里访问，若是数组或对象（引用类型），可修改其中属性，但不能对数组对象重新赋值。

预编译四部曲：

① 创建AO对象 activeobject 也就是函数的执行器上下文

② 创建变量和形参的声明，赋值undefined(初始化)

③ 给形参赋值

④ 函数声明（不包括函数表达式）

变量提升：

var存在创建、初始化提升，赋值不提升；

let只会提升创建过程，不会自动初始化赋值undefined，且不能重复声明已存在的变量。

Const如果用来定义一个基本数据类型的变量，那么值不能改变

如果用来定义一个对象类型，则对象中的属性以及方法等内容可以改变，因为const只是保存的引用对象变量的地址。

2. 变量提升与暂时性死区: let、const都会产生死区。

变量提升： 即变量可以在声明之前使用，值为undefined。

暂时性死区： 当前作用域顶部到该变量声明位置中间的部分，都是该变量的死区，在死区中，禁止访问该变量。由此给出结论，let、const声明的变量存在变量提升， 但是由于死区我们无法在声明前访问这个变量，访问会报错。

3. 变量的结构赋值

解构：ES6允许按照一定模式，从数组和对象中提取值，对变量进行赋值，这被称为解构。

数组的解构赋值：

本质上，这种写法属于模式匹配，只要等号两边的模式相同，左边的变量就会被赋予对应的值。如果解构不成功，变量的值就等于undefined。另一种情况是不完全解构，即等号左边的模式，只匹配一部分的等号右边的数组，这种情况下，解构依然可以成功。

解构不成功的时候，才会有默认值，没有赋值，没有定义。

对象的解构赋值：变量和属性必须同名，才可以解构赋值，如果解构失败，值是 undefined

默认值跟数组解构的默认值是一样的，解构成功就直接赋值，不走默认值，解构不成功，就是没有值，或者值是 undefined 走默认值；

函数参数等号左边是形参，右边是默认值，函数调用不传参，走函数形参的默认值；函数调用传参，默认值就不走了，走实参

1. 箭头函数及其this问题（匿名函数和箭头函数）

## 5. Symbol概念及其作用

Symbol()函数接受一个可选参数，可以添加一段文本描述即将创建的Symbol，这段描述不可用于属性访问，但是建议每次创建Symbol时都添加一段描述，便于阅读代码和调试Symbol程序。

Symbol的描述被存储在内部的[[Description]]属性中,只有调用Symbol的toString()方法时才可以读取这个属性.在执行console.log的时候隐式的调用了toString()方法。

Symbol()函数会返回symbol类型的值，该类型具有静态属性和静态方法。它的静态属性会暴露几个内建的成员对象；

它的静态方法会暴露全局的symbol注册，且类似于内建对象类，但作为构造函数来说它并不完整，因为它不支持语法："new Symbol()"。

var sym = Symbol("foo");

每个从Symbol()返回的symbol实例都是唯一的。一个symbol值能作为对象属性的标识符；这是该数据类型仅有的目的。

Symbol()每次都会创建一个新的 symbol类型

var s1 = Symbol('foo'); var s2 = Symbol('foo')

console.log(s1===s2); // false

全局共享symbol体系：

let gs1 = Symbol.for('global\_symbol\_1') //注册一个全局Symbol

let gs2 = Symbol.for('global\_symbol\_1') //获取全局Symbol

console.log(gs1 === gs2); // true

这样一个Symbol在所有window环境下都保持唯一了。

Symbols 在 for...in 迭代中不可枚举。另外，Object.getOwnPropertyNames() （不考虑属性的可枚举性一律返回）和Object.keys()（返回所有可枚举属性名） 都不会返回 symbol 对象的属性。但是能使用 Object.getOwnPropertySymbols() （返回包含所有Symbol自有属性的数组）得到它们，还有反射API 也可以得到全部的属性包括symbol，。利用该特性，可以把一些不需要对外操作和访问的属性使用Symbol来定义。更好的设计我们的数据对象，让“对内操作”和“对外选择性输出”变得更加优雅。

应用场景1：使用Symbol来作为对象属性名(key)；

应用场景2：使用Symbol来替代常量；

应用场景3：使用Symbol定义类的私有属性/方法。

## 6. Set和Map数据结构

Set: 是一组key的集合，不存储value, 而且key不重复，可自动排重

方法： add(value)：添加某个值，返回Set结构本身。

delete(value)：删除某个值，返回一个布尔值，表示删除是否成功。

has(value)：返回一个布尔值，表示该值是否为Set的成员。

clear()：清除所有成员，没有返回值。

Map: 键/值对的集合。具有极快的查找速度。

属性： 构造函数： 指定创建映射的函数。

Prototype原型： 为映射返回对原型的引用。

size： 返回映射中的元素数。

方法： clear： 从映射中移除所有元素。

delete： 从映射中移除指定的元素。

forEach： 对映射中的每个元素执行指定操作。

get： 返回映射中的指定元素。

has： 如果映射包含指定元素，则返回 true。

set： 添加一个新建元素到映射。

toString： 返回映射的字符串表示形式。

valueOf： 返回指定对象的原始值。

forEach是iterable内置的方法，它接收一个函数，每次迭代就自动回调该函数。

array.forEach(function (element, index, array) {

// element: 指向当前元素的值 // index: 指向当前索引

// array: 指向Array对象本身

});

set.forEach(function (element, sameElement, set) {

// element和sameElement都是元素本身

})

map.forEach(function ( value, key, map) { } );

## 7. Proxy

Proxy 对象用于定义基本操作的自定义行为（如属性查找，赋值，枚举，函数调用等）。

handler：包含陷阱（traps）的占位符对象。

traps：提供属性访问的方法。这类似于操作系统中捕获器的概念。

target：代理虚拟化的对象。它通常用作代理的存储后端。根据目标验证关于对象不可扩展性或不可配置属性的不变量（保持不变的语义）。

语法：let p = new Proxy(target, handler);

target：用Proxy包装的被代理对象（可以是任何类型的对象，包括原生数组，函数，甚至另一个代理）。

handler：一个对象，其属性是当执行一个操作时定义代理的行为的函数。

当外界每次对 p 进行操作时，就会执行 handler 对象上的一些方法。handler 能代理的一些常用的方法如下：

get：读取 set：修改 has：判断对象是否有该属性 construct：构造函数...

使用 Proxy 的好处是：对象只需关注于核心逻辑，一些非核心的逻辑（如：读取或设置对象的某些属性前记录日志；设置对象的某些属性值前，需要验证；某些属性的访问控制等）可以让 Proxy 来做。从而达到关注点分离，降级对象复杂度的目的。

以下代码：当对象中不存在属性名时，缺省返回数为37。使用了 get。

let handler = {

get: function(target, name){

return name in target ? target[name] : 37;

}

};

let p = new Proxy({}, handler);

p.a = 1;

p.b = undefined;

console.log(p.a, p.b); // 1, undefined

console.log('c' in p, p.c); // false, 37

## 8. Reflect对象

Reflect 是一个内置的对象，它提供拦截 JavaScript 操作的方法。这些方法与处理器对象的方法相同。Reflect不是一个函数对象，因此它是不可构造的。

与大多数全局对象不同，Reflect不是一个构造函数。不能将其与一个new运算符一起使用，或者将Reflect对象作为一个函数来调用。Reflect的所有属性和方法都是静态的（就像Math对象）。

Reflect.get(target, name, receiver)：此方法查找并返回target的name属性，如果没有，则返回undefined。如果第一个参数不是对象，则Reflect.get则会报错。

如果name属性是读取函数(getter)，则读取函数里的this绑定Reflect.get的receiver参数。

Reflect.set(target, name, value, receiver)：此方法设置target对象的name属性等于value。

如果name参数是一个赋值函数，则赋值函数的this绑定Reflect.set的receiver参数。

var myObject = {

foo: 4,

set bar(value) {

return this.foo = value;

},

};

var myReceiverObject = {

foo: 0,

};

Reflect.set(myObject, 'bar', 1, myReceiverObject);

myObject.foo // 4

myReceiverObject.foo // 1

Reflect.has(obj, name):Reflect.has对应 name in obj 里面的in操作，如果第一个参数不是对象，Reflect.has和in都会报错。

var myObject = { foo: 1, };

'foo' in myObject // 旧写法 true

Reflect.has(myObject, 'foo') // 新写法 true

Reflect.deleteProperty(obj, name):此方法等同于delete obj[name]，用于删除对象属性。该方法返回一个布尔值。如果删除成功或删除的属性不存在，则返回true，如果删除失败，删除的属性依然还在，则返回false。

Reflect.construct(target, args)：此方法等同于new target(...args)，这提供了一种不使用new，来调用构造函数的方法。

Reflect.getPrototypeOf(obj)：此方法用于读取对象的\_\_proto\_\_属性，对应Object.getPrototypeOf(obj)方法。它们的区别是，如果参数不是对象，Object.getPrototypeOf会将参数转化为对象，而Reflect.getPrototypeOf会报错。

Reflect.setPrototypeOf(obj, newProto)：此方法是设置对象的\_\_proto\_\_属性，返回第一个参数对象，对应Object.setPrototypeOf(obj, newProto)。如果第一个参数不是对象，Object.setPrototypeOf会返回第一个参数对象，而Reflect.setPrototypeOf会报错。如果第一个参数是undefined或null，则两个都会报错。

Reflect.apply(func, thisArgs, args)：对一个函数进行调用操作，同时可以传入一个数组作为调用参数。和 Function.prototype.apply() 功能类似。

Reflect.defineProperty(target, propertyKey, attributes)：此方法等同于Object.defineProperty，用来为对象定义属性。未来后者会被逐渐废除。如果第一个参数不是对象，就会抛出错误信息。

Reflect.getOwnPropertyDescriptor(target, propertyKey) ：该方法等同于Object.getOwnPropertyDescriptor，用于得到指定属性的描述对象，将来会代替后者。如果第一个参数不是对象Object.getOwnPropertyDescriptor不报错，返回undefined，而Reflect.getOwnPropertyDescriptor会报错，表示参数非法。

Reflect.isExtensible (target) ：该方法等同于Object.isExtensible，返回一个布尔值，表示当前对象是否可扩展。如果参数不是对象，Object.isExtensible会返回false，因为本来就是不可扩展的，而Reflect.isExtensible则报错。

Reflect.preventExtensions(target)：等同于Object.preventExtensions，用于让一个对象变为不可扩展，返回一个布尔值，表示是否操作成功。如果参数不是对象，Object.preventExtensions在ES5下会报错，ES6下会返回传入的值，Reflect.preventExtensions则会报错。

Reflect.ownKeys (target)：Reflect.ownKeys方法用于返回对象的所有属性，等同于Object.getOwnPropertyNames与Object.getOwnPropertySymbols之和。

作用：

1. 将object对象一些内部的方法，放到Reflect对象上。也就是说，从Reflect对象上可以拿到语言内部的方法。说明现阶段这些方法存在于object和Reflect对象上，未来只会存在于Reflect对象上。
2. object操作对象时出现报错返回false，而reflect会返回false。
3. 让操作对象的编程变为函数式编程
4. 保持和proxy对象的方法一一对应：Reflect对象的方法与Proxy对象的方法一一对应，只要是Proxy对象的方法，就能在Reflect对象上找到对应的方法。

## 9. Promise（手撕Promise A+规范、Promise.all、Promise相关API和方法）

10. Iterator和for...of（Iterator遍历器的实现）  
11. 循环语法比较及使用场景（for、forEach、for...in、for...of）

12. Generator及其异步方面的应用

## 13. async函数

async函数对 Generator 函数的改进，体现在以下四点：

（1）内置执行器 （2）更好的语义 （3）更广的适用性 （4）返回值是 Promise

Async函数是一个异步操作函数，本质上，Async函数是Generator函数的语法糖。async函数就是将 Generator 函数的星号（\*）替换成async，将yield替换成await，仅此而已。

async function test(){

const a = await fn1();

const b = await fn2(a);

}

test();

async函数不需要像 Generator 函数，需要调用next方法才能执行，其调用方式与普通函数调用方式一致。

async表示函数里有异步操作，await表示紧跟在后面的表达式需要等待结果

async函数的返回值是 Promise.

async 注意点：

（1）await 命令后面的 Promise对象结果可能是 rejected，因此需要对此进行捕获，以免错误丢失。

（2）多个 await 的操作，如果互相无逻辑关系，可将多个 await 并发处理：Promise.all

（3）await 只能写在 async 函数中

（4）async 函数可以保留运行堆栈

## 14. 几种异步方式的比较（回调、setTimeout、Promise、Generator、async）

一、回调函数：

function f1(callback){

　　setTimeout(function () {

　　　　// f1的任务代码

　　 callback();

}, 1000);

}

回调函数的优点是简单、容易理解和部署，缺点是不利于代码的阅读和维护，各个部分之间高度耦合（Coupling），流程会很混乱，而且每个任务只能指定一个回调函数。

二、事件监听：

采用事件驱动模式。任务的执行不取决于代码的顺序，而取决于某个事件是否发生。

f1.on('done', f2);

function f1(){

setTimeout(function () {

　　　　// f1的任务代码

　　 f1.trigger('done');

}, 1000);

}

优点：容易理解，可以绑定多个事件，每个事件可以指定多个回调函数，而且可以’去耦合’，有利于实现模块化。缺点是整个程序都要变成事件驱动型，运行流程会变得很不清晰。

三、发布/订阅模式（观察者模式）：

假设存在一个"信号中心"，某个任务执行完成，就向信号中心"发布"一个信号，其他任务可以向信号中心"订阅"这个信号，从而知道什么时候自己可以开始执行。

jQuery.subscribe("done", f2); //f2向"信号中心"jQuery订阅"done"信号

function f1(){

setTimeout(function () {

// f1的任务代码 // f1执行完成后，发布"done"信号，从而引发f2的执行

jQuery.publish("done");

}, 1000);

}

jQuery.unsubscribe("done", f2); // f2完成执行后也可以取消订阅

这种方法的性质与"事件监听"类似，但是明显优于后者。因为我们可以通过查看"消息中心"，了解存在多少信号、每个信号有多少订阅者，从而监控程序的运行。

四、promise：

每一个异步任务返回一个Promise对象，该对象有一个then方法，允许指定回调函数。

f1().then(f2).then(f3);

优点：回调函数变成了链式写法，程序的流程可以看得很清楚，而且有一整套的配套方法，可以实现许多强大的功能；（前三种没有的好处：）如果一个任务已经完成，再添加回调函数，该回调函数会立即执行。所以不用担心是否错过了某个事件或信号。

五、generator（生成器）函数：

Generator函数，在Javascript中可以创建迭代对象，还有另一个功能：即实现异步。

Generator函数是ES6提供的一种异步编程解决方案，其行为类似于状态机，封装了多个内部状态。

* Generator 函数除了状态机，还是一个遍历器对象生成函数。
* 可暂停函数, yield可暂停，next方法可启动，每次返回的是yield后的表达式结果。
* yield表达式本身没有返回值，或者说总是返回undefined。next方法可以带一个参数，该参数就会被当作上一个yield表达式的返回值。

function \* show(){

yield read('1.txt');

yield read('2.txt');

}

六、async / await函数：

实现原理就是就是将 Generator 函数和自动执行器，包装在一个函数里。

async/await本质上还是基于Generator函数，可以说是Generator函数的语法糖，async就相当于之前写的run函数(执行Generator函数的函数),而await就相当于yield，只不过await表达式后面只能跟着Promise对象，如果不是Promise对象的话，会通过Promise.resolve方法使之变成Promise对象。async修饰function,其返回一个Promise对象。await必须放在async修饰的函数里面，就相当于yield只能放在Generator生成器函数里一样。

async function readByAsync(){

try{

let a1 = await read('1.txt');

let a2 = await read('2.txt');

}catch(e){ //TODO handle the exception }

}

特点：

async/await是基于Promise实现的，它不能用于普通的回调函数。

async/await与Promise一样，是非阻塞的。

async/await使得异步代码看起来像同步代码，这正是它的魔力所在。

Promise 对象、Generator 函数、async 函数比较：

（1）Promise 的 API（then、catch等等）太多，使得实际逻辑展示不清晰

（2）Generator 函数的执行必须依赖执行器

（3）async 函数实现简介，语句清晰

## 15. class基本语法及继承:

类和模块的内部，默认就是严格模式，所以不需要使用use strict指定运行模式。

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

toString() {

return '(' + this.x + ', ' + this.y + ')';

}

}

var point = new Point（）

Point.prototype.constructor === Point // true

类的数据类型就是函数，类本身就指向构造函数。类的所有方法都定义在类的prototype属性上面。在类的实例上面调用方法，其实就是调用原型上的方法。实例的属性除非显式定义在其本身（即定义在this对象上），否则都是定义在原型上（即定义在class上）。

类的内部所有定义的方法，都是不可枚举的。

类的方法内部如果含有this，它默认指向类的实例。

注意⚠：

1、类不存在变量提升

2、类必须使用new调用，否则会报错。而构造函数可以用new或者不使用new。

加new执行构造函数时：函数内部的this指向当前实例对象

不加new执行构造函数时：非严格模式下函数内部的this指向window对象，严格模式为undefined。返回值是构造函数的执行结果；

constructor方法：是类的默认方法，通过new命令生成对象实例时，自动调用该方法。一个类必须有constructor方法，如果没有显式定义，一个空的constructor方法会被默认添加。

super关键字：必须使用继承才会用到super关键字

①super作为函数调用时，代表父类的构造函数。

②super作为对象时，在普通方法中，指向父类的原型对象；在静态方法中，指向父类。

如果子类调用constructor，那么子类必须在constructor方法中调用super方法，否则新建实例时会报错。这是因为子类没有自己的this对象，而是继承父类的this对象，然后对其进行加工。如果不调用super方法，子类就得不到this对象。

Class A{

constructor(name) {

this.name = name;

}

}

Class B extends A{

constructor(){

super();

}

}

16. 模块加载方案比较（CommonJS和ES6的Module）  
17. ES6模块加载与CommonJS加载的原理

callee和caller是arguments对象的两个内部属性。

callee返回正在执行的函数本身的引用。callee是arguments的一个属性，这个属性是一个指针，指向拥有这个arguments对象的函数。可利用该属性完成递归。可以降低耦合性： return num\*factorial(num-1);

caller；这个属性中保存着调用当前函数的函数引用，如果是全局作用于中调用当前的函数就返回null。

arguments.callee在严格模式下会导致错误；不能为caller赋值，否则报错；

## 18.项目难点：图片上传

<input type="file" id="imgUp" onchange="preImg(this.id,'imgPre');" multiple="multiple" accept="image/\*" />

accept:表示控件允许上传的文件格式 multiple表示可以上传多个文件

获得选中图片的url有两种方式：

1、FileReader：

用来把文件读入内存，并且读取文件中的数据。FileReader接口提供了一个异步API，使用该API可以在浏览器主线程中异步访问文件系统，读取文件中的数据。

var reads = new FileReader(); reads.readAsDataURL( element.files[0] );

reads.onload = function (e) { // onload表示成功读取

document.getElementById('img').src = this.result; //图片预览显示

};

2、createObjectURL：

URL.createObjectURL() 静态方法会创建一个 DOMString，其中包含一个表示参数中给出的对象的URL。这个 URL 的生命周期和创建它的窗口中的 document 绑定。这个新的URL 对象表示指定的 File 对象或 Blob 对象。

url = window.URL.createObjectURL(element.files.item(0));

区别：

1. 通过FileReader.readAsDataURL(file)可以获取一段data:base64的字符串
2. FileReader.readAsDataURL是异步执行（过一段时间）
3. FileReader.readAsDataURL则返回包含很多字符的base64，并会比blob url消耗更多内存，但是在不用的时候会自动从内存中清除（通过垃圾回收机制）
4. 通过URL.createObjectURL(blob)可以获取当前文件的一个内存URL
5. createObjectURL是同步执行（立即的）
6. createObjectURL返回一段带hash的url，并且一直存储在内存中，直到document触发了unload事件（例如：document close）或者执行revokeObjectURL来释放。

使用createObjectURL可以节省性能并更快速，只不过需要在不使用的情况下手动释放内存。如果不太在意设备性能问题，并想获取图片的base64，则推荐使用FileReader.readAsDataURL;

将图片url传给后台数据使用FormData；调用它的 append() 方法来添加字段

var formdata = new FormData();

formdata.append("username", "Groucho");

// 通过append()方法在数据末尾追加key为username值为Groucho的数据

formdata.get("username "); // 获取key为username的第一个值

formdata.getAll("username "); //获取key为username的所有值，返回值为数组

formdata.set("name","laoli"); // 通过set(key, value)来设置或修改数据

formdata.has("username") // 通过has(key)来判断是否存在对应的key值

formdata.delete("name"); //通过delete(key)可以删除数据

//构造一个包含Form表单数据的FormData对象，需要在创建FormData对象时指定表单元素。

var formdata=new FormData(document.getElementById("advForm"));

FormData的作用：1、异步上传文件

2、将form表单元素的name与value进行组合，实现表单数据的序列化，从而减少表单元素的拼接，提高工作效率。

问题：

1、解决input file上传图片无法上传相同的图片 如果是相同图片onChange事件只会触发一次

办法：onChange事件代码最后清除元素的value

document.querySelector('#uploader').value = null 或者 this.value = null;

2、上传多张图片时，需要拼接file数组：

方式1：

$.each(getImgFiles(), function(i, file){

formData.append('files', file);

});

方式2：

$.each(getImgFiles(), function(i, file){

formData.append('files[]', file);

});

3、如果使用jquery封装的ajax提交表单数据则需要注意以下两点：

processData : false, // 告诉jQuery不要去处理发送的数据

contentType : false, // 告诉jQuery不要去设置Content-Type请求头

# 三、HTML/CSS

1. CSS权重及其引入方式：

引入方式：

内联样式 在标签中写<style>标签

内部样式css 在head标签写style标签

外部css文件 使用link标签

导入样式 @import url("外部样式表的名称")

权重：

!important :无穷大 行间样式：1000 id:100

class / 属性 / 伪类:10 标签 / 伪元素: 1 通配符：0

！important>内联样式>id选择器>类选择器>标签选择器>通配符>继承

## 2. <a></a>标签全部作用

(1) 页面跳转

<a href = "路径" target="属性值">这个路径在这指的是需要跳转的页面的网址</a>

(2) 定位

设定锚点（#）：在要定位到的位置找一个标签给他加id属性，属性值就是锚点名称

(3) 下载资源或其他：a 标签跳转到的文件 浏览器打不开，然后会选择提供下载。

target属性：指定在何处显示链接的资源。target属性的四个关键字具有特殊的意义

\_self ：加载当前页面，如果没有指定属性，则此值为默认；

\_top： 加载响应进入顶层页面，如果没有父级框架或页面，效果与 \_self相同；

\_parent：在父级页面打开地址；

\_blank：在新窗口打开；

download属性：指示浏览器下载URL而不是导航到URL。

<a href="http://qq.com" download>QQ</a>在点击此链接后并不会打开 http://qq.com，而是将此网页保存到本地。

href属性：是<a>标签必须要有的属性，它为<a>标签定义一个超文本链接，但是href属性的值有多种选择：

1. 最常见的值是一个网页的URL，如 <a href="http://qq.com">QQ</a>；
2. 无协议<a>标签，如 <a href="//qq.com">QQ</a>，此时所用协议与当前页面相同；
3. <a>标签href使用锚点#，此时点击<a>标签不发送请求，如 <a href="#aaa">aaa<a/>，此时点击a链接页面锚点变为aaa，使用锚点可以实现当前页面的跳转；
4. <a>标签href内容为空， <a href="">QQ</a>，点击a链接会刷新当前页面；
5. 类似 <a href="?name=xxx">a链接</a> 的<a>标签，再点击后会发送一个GET请求，请求的值就是 name=xxx
6. 使用路径作为href的值， <a href="main.html">main</a>，点击后会跳转到main.html；
7. 伪协议： <a href="javascript: alert(1);"></a> 会执行href中的JavaScript代码，<a href="javascript:;"></a>将不会执行任何操作；

3. 用CSS画三角形:

利用边框border；

Width = 0； height = 0；border：100px solid red；此时由于宽高为0，border为100会显示4个三角形组合的正方形。

## 4. 未知宽高元素水平垂直居中（方案及比较）

1. Flex弹性盒子布局

父级添加如下属性：

display: flex; justify-content: center; align-items: center;

优点：代码量小，简单快捷方便。

缺点：兼容不好

1. transform + 父相子绝：

top: 50%; left: 50%; transform: translate(-50%, -50%);

优点：高大上,可以在webkit内核的浏览器中使用  
缺点：不支持IE9以下不支持transform属性

1. display的table-cell属性：

display: table-cell; vertical-align: middle; text-align: center;

优点：父元素（parent）可以动态的改变高度（table元素的特性）

缺点：IE8以下不支持，表格渲染影响性能

5. 元素种类的划分:

内联(行级)标签：标签在一行内，宽度与高度由内容决定，只有在内容超过HTML的宽度时，才会换行。内联元素一般不可以包含块级元素。

常见：<a><img><input><span><em><br><textarea><select>

块状标签：标签独占一行，可指定宽、高。

常见：<div><h1>~<h6><hr><ul><ol><li><p><table><form><dl>

内联块状元素（display:inline-block）就是同时具备内联元素、块状元素的特点。

特点：和其他元素都在一行上；元素的高度、宽度、行高以及顶和底边距都可设置。

缺陷：会产生间隙，（原因：换行符、制表符等产生的）可设置父级font-size：0消除间隙。

**非替换元素:**

如果元素的内容包含在文档中，则称之为非替换元素。比如一个段落的文本都在该元素本身之内，这个段落就是一个非替换元素。

**替换元素:**

作为其他内容占位符的一个元素称为替换元素，根据标签和属性的值来显示内容的元素。比如img元素，它只是指向一个图像文件，这个文件插入到文档流中。大多数表单元素（input，根据type属性来显示内容）也是替换元素。

## 6. 盒子模型及其理解

由里向外是content , padding , border , margin.

盒模型的两种标准：

标准模型（content-box）：盒模型的宽高width只是内容（content）的宽高，

IE模型（border-box）：总宽高width=内容(content)+填充(padding)+边框(border)

box-sizing(切换盒子模型): content-box (默认)、border-box 、paading-box。

## 7. position定位方式及其区别（文档流）/ CSS定位属性：

static：默认，无定位，元素出现在正常的流中。

relative：生成相对定位的元素，相对于原来正常位置进行定位，仍在文档流中。

absolute：生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。

fixed：生成绝对定位的元素，相对于浏览器窗口进行定位。

inherit：继承，规定应该从父元素继承 position 属性的值。

sticky：粘性定位，是relative和fixed的混合。元素在跨越特定阈值前为相对定位，之后为固定定位。（兼容不好）

sticky使用条件：  
1、父元素不能overflow:hidden或者overflow:auto属性。  
2、必须指定top、bottom、left、right4个值之一，否则只会处于相对定位  
3、父元素的高度不能低于sticky元素的高度  
4、sticky元素仅在其父元素内生效

8. margin塌陷及合并问题:

合并：两个兄弟结构的元素在垂直方向上的margin是合并的(bottom和top组合)

解决方法：（都改变了HTML结构，不推荐，最好通过数学计算来解决）

1)给弟弟元素加上一层父级元素并设置overflow:hidden;

2)给两个都加一层父级再加BFC

塌陷：父子嵌套元素在垂直方向的margin,父子元素是结合在一起的,他们两个的margin会取其中最大的值。

解决方法：触发BFC(块级格式上下文),改变父级的渲染规则.父级设置如下属性：

(1) position : absolute / fixed

(2) display : inline-block;

(3) float : left / right

(4) overflow : hidden

## 9. 浮动模型及清除浮动的方法：

float会使元素脱离文档流，按照指定方向发生移动，遇到父级边界或者相邻的浮动元素停了下来。float可设置文字环绕或使元素宽度由内容填充。

缺陷：会造成父级高度坍塌（如果父元素只包含浮动元素，因为浮动元素是不包含在正常流的，浮动孩子将会脱离页面的常规流，因此父元素相当于不包含任何元素，高度为0。）

解决办法：

1. 父级设置高度（非常不推荐）
2. 父级设置overflow: hidden / auto ;
3. 添加一个空标签设置clear:both; 但是改变了DOM结构。
4. 给父级样式添加伪元素::after或者::before （推荐此方法）

.clear::after { content: ''; display: block; clear: both; }

10. display及相关属性:

该属性不支持继承。

|  |  |
| --- | --- |
| 值 | 描述 |
| none | 此元素不会被显示。 |
| block | 此元素将显示为块级元素，此元素前后会带有换行符。 |
| inline | 默认。此元素会被显示为内联元素，元素前后没有换行符。 |
| inline-block | 行内块元素。（CSS2.1 新增的值） |
| list-item | 此元素会作为列表显示。 |
| table | 此元素会作为块级表格来显示（类似 <table>），表格前后带有换行符。 |
| inline-table | 此元素会作为内联表格来显示（类似 <table>），表格前后没有换行符。 |
| table-row | 此元素会作为一个表格行显示（类似 <tr>）。 |
| table-cell | 此元素会作为一个表格单元格显示（类似 <td> 和 <th>）（触发BFC） |
| inherit | 规定应该从父元素继承 display 属性的值。 |

## 11. IFC与BFC:

Box:

一个页面是由很多个 Box 组成的，元素的类型和 display 属性决定了这个 Box 的类型。不同类型的 Box，会参与不同的 Formatting Context。

Block level的box会参与形成BFC，比如display值为block , list-item, table的元素.

Inline level的box会参与形成IFC，比如display值为inline，inline-table，inline-block的元素。

FC:

FC的含义就是Fomatting Context。它是CSS2.1规范中的一个概念。它是页面中的一块渲染区域，并且有一套渲染规则，它决定了其子元素将如何定位，以及和其他元素的关系和相互作用。BFC和IFC都是常见的FC。

BFC（Block Formatting Context）：“块级格式化上下文”。其布局规则如下：

1.内部的盒子会在垂直方向上，一个一个地放置；

2.盒子垂直方向的距离由margin决定，属于同一个BFC的两个相邻Box的上下margin会发生重叠；

3.每个元素的margin box的左边，与包含块border box的盒子的左边相接触，即使存在浮动也是如此；

4.BFC的区域不会与float重叠；

5.BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素，外面的元素也不会影响到容器里的子元素

6.计算BFC的高度时，浮动元素也参与计算。（清除浮动可利用此条）

如何让元素变成BFC？

1. 根元素html（注意，不是body）
2. position为absolute或fixed
3. display为inline-block，table-cell，table-caption
4. overflow不为visible
5. float属性不为none

BFC的作用：通常会用来解释以下2个问题：

1. 防止垂直margin重叠合并

2. 清除浮动父级元素产生的高度坍塌

3. 布局：自适应两栏布局

IFC（行内格式化上下文）：

IFC的line box（线框）高度由其包含行内元素中最高的实际高度计算而来（不受到竖直方向的 padding/margin 影响)。

IFC也是一种布局规则，inline元素和inline-block符合IFC的布局规则

IFC的生成条件是元素满足inline-level box。

IFC的特性：

1.一个line box的宽度由包含它的元素的宽度和包含它的元素里面有没有float元素来决定，而高度由内部元素中实际高度最高的元素而计算出来。

2. line box的高度是足够高来包含他内部的容器们的，也可能比它包含的容器们都高（比如在基线对齐的时候），当他包含的内部容器的高度小于line box的高度的时候，内部容器的垂直位置由自己的vertical这个属性来确定，当内部的容器盒子太多了一个line box装不下来，他们折行之后会变成两个或者多个line box， line box们相互之间垂直方向不能分离，不能重叠。

3．line box的左边缘挨着包含它的元素的左边缘，并且右边缘挨着包含它的元素的右边缘，浮动元素会在包含他们的元素的边缘和line box的边缘之间，所以虽然在同一个IFC下的line box们通常拥有相同的宽度（就是包含他们的容器的宽度），但是也会因为浮动元素的捣乱，导致line box们的可用宽度产生变化不一样了。在同一个IFC下的line box们的高度也会不一样。

4．如果一个line box 里的内联元素们的宽度总和小于这个line box的宽度，那么他们在这个line box里的水平方向的排布方式由 text-align这个属性来决定。

5．在一个line box中，当他包含的内部容器的高度小于line box的高度的时候，内部容器的垂直位置由自己的vertical这个属性来确定。

6．当内联元素的宽度超过了line box的宽度，那么它会折行分裂成了几个line box，如果这个元素里面的内容不可以折行，例如只有一个字，或者white-space设置了nowrap/pre。那么内联元素会溢出line box。

7．line box 的生存条件是在IFC中并且包含inline-level元素，如果line box里没有文本，空白，换行符，内联元素，也没有其他的存在IFC环境中的元素，(如inline-block，inline- table，images等)，将会被视为零高度，也将会被视为没有意义。

8．IFC中是不可能有块级元素的，当插入块级元素时（如p中插入div）会产生两个匿名块与div分隔开，即产生两个IFC，每个IFC对外表现为块级元素，与div垂直排列。

IFC的应用：

水平居中：当一个块要在环境中水平居中时，设置其为inline-block则会在外层产生IFC，通过text-align则可以使其水平居中。

垂直居中：创建一个IFC，用其中一个元素撑开父元素的高度，然后设置其vertical-align:middle，其他行内元素则可以在此父元素下垂直居中。

GFC(GridLayout Formatting Contexts)："网格布局格式化上下文"，当为一个元素设置display值为grid的时候，此元素将会获得一个独立的渲染区域。

FFC(Flex Formatting Contexts)："自适应格式化上下文"，display值为flex或者inline-flex的元素将会生成自适应容器（flex container）

## 12. 圣杯布局和双飞翼布局的实现:

圣杯布局：（利用relative定位+float+left+right属性+负margin）

<div class="content">自适应区域</div>

<div class="left">左侧固定</div>

<div class="right">右侧固定</div>

实现原理：首先给外层container的div设置左右padding已留出左右div的位置，三个div设置float；由于content的宽度为100%，即占据了第一行的所有空间，所以left和right被挤到了第二排；给left盒子设置margin-left：-100%，left这个块会向左移动100%，这里的100%是父级宽度的100%,就正好到了父级的左边，同理，right盒子设置margin-left：-200px；向左移动了200px，到达父级的右侧。最后使用position：relative 定位到相应位置，添加左右left / right 负的本身宽度，偏移到浏览器窗口的最左边和最右边。

双飞翼布局：（无需设置relative定位，通过给中间div新增额外div设置margin实现）

<div class="middle">

<div class="content">自适应区域</div>

</div>

<div class="left">左侧固定</div>

<div class="right">右侧固定</div>

双飞翼布局与圣杯布局的区别是使用一个块将中间元素包裹，设置中间父元素width100%，内容元素设置margin-left / right。

圣杯布局和双飞翼布局基本上是一致的，其中，中间栏放到文档流前面，保证先行渲染。然后，都是三栏全部float:left浮动，区别在于解决中间栏div的内容不被遮挡上，

圣杯布局是中间栏在添加相对定位，并配合left和right属性，效果上表现为三栏是单独分开的（如果可以看到空隙的话），

而双飞翼布局是在中间栏的div中嵌套一个div，内容写在嵌套的div里，然后对嵌套的div设置margin-left和margin-right，效果上表现为左右两栏在中间栏的上面，中间栏还是100%宽度，只不过中间栏的内容通过margin的值显示在中间。

## 13. Flex布局

设置在父级容器的属性：

flex-direction：决定主轴的方向（即项目的排列方向）。

flex-wrap：定义，如果一条轴线排不下，如何换行。

flex-flow：是flex-direction属性和flex-wrap属性的简写形式，默认值为row nowrap。

justify-content：定义了项目在主轴上的对齐方式。

align-items：定义了项目在交叉轴上的对齐方式。

align-content：定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

设置在项目上的属性：

order：定义项目的排列顺序。数值越小，排列越靠前，默认为0。

flex-grow：定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大。

flex-shrink：定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。

flex-basis：定义了在分配多余空间之前，项目占据的主轴空间。默认值为auto，即项目的本来大小。

Flex：是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto。后两个属性可选。

align-self：允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch。

## 14. px、em、rem的区别

一、px是固定的像素，一旦设置了就无法因为适应页面大小而改变。

二、em和rem相对于px更具有灵活性，他们是相对长度单位，更适用于响应式布局。

三、em是相对于其父元素来设置字体大小的，这样进行任何元素设置，都要知道父元素的大小。而rem是相对于根元素<html>，这表示只需要在根元素确定一个字体大小的值。

em和%都是相对于父级元素的单位，会随父级元素的属性（font-size或其它属性）变化而变化。子元素字体大小的em是相对于父元素字体大小；元素的width / height / padding / margin用em的话是相对于该元素的font-size。

rem作用于非根元素时，相对于根元素字体大小；rem作用于根元素字体大小时，相对于其出初始字体大小。

默认：1em = 1rem = 100% = 16 px

总结：

1. rem 和 em 单位是由浏览器基于你的设计中的字体大小计算得到的像素值。
2. em 单位基于使用他们的元素的字体大小。
3. rem 单位基于 html 元素的字体大小。
4. em 单位可能受任何继承的父元素字体大小影响
5. rem 单位可以从浏览器字体设置中继承字体大小。
6. 使用 em 单位应根据组件的字体大小而不是根元素的字体大小。
7. 在不需要使用em单位，并且需要根据浏览器的字体大小设置缩放的情况下使用rem。
8. 使用rem单位，除非你确定你需要 em 单位，包括对字体大小。
9. 媒体查询中使用 rem 单位
10. 不要在多列布局中使用 em 或 rem -改用 %。
11. 不要使用 em 或 rem，如果缩放会不可避免地导致要打破布局元素。

注意⚠：样式的reset中需先设置html字体的初始化大小为50px，这是为了防止js被禁用或者加载不到或者执行错误。而兼容样式的reset中需先设置body字体的初始化大小为16px，这是为了让body内的字体大小不继承父级html元素的50px，而用系统默认的16px。

## 15.实现适配的原理：

适配方法：

1)通过媒体查询的方式

2)以天猫首页为代表的 flex 弹性布局

3)以淘宝首页为代表的 rem+viewport缩放

4)rem 方式

rem适配原理：改变了一个元素在不同设备上占据的css像素的个数。

实现手段：动态获取当前视口宽度width，除以一个固定的数n，得到rem的值。表达式为rem = width / n。

优点:可以使用完美视口。

缺点: 1.计算设计图上的px值到rem的转换比例麻烦。

2.字体大小并不能使用rem，字体的大小和宽度，并不成线性关系，所以字体大小不能使用rem；由于设置了根元素字体的大小，会影响所有没有设置字体大小的元素。

注意⚠：移动端必须加上meta标签:

<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />

// 动态为根元素设置字体大小 // 获取屏幕宽度

var width = document.documentElement.clientWidth;  
// 设置根元素字体大小。此时1rem为宽的100等分;

document.documentElement.style.fontSize = width / 100 + 'px';

viewport适配原理: viewport适配方案中，每一个元素在不同设备上占据的css像素的个数是一样的。但是css像素和物理像素的比例是不一样的，所以是等比的。根据rem将页面放大dpr倍, 然后viewport设置为1/dpr.

如iphone6 plus的dpr为3, 则页面整体放大3倍, 1px(css单位)在plus下默认为3px(物理像素) ,然后viewport设置为1/3, 这样页面整体缩回原始大小. 从而实现高清。

优点:不用计算元素在设计图上的比例 缺点:不能使用完美视口。

基本概念：1px = (dpr)的平方\*dp

物理像素：硬件上屏幕上横向和纵向有多少个像素点。

逻辑像素dp：程序认为屏幕上横向和纵向有多少个像素点。

设备像素比：设备像素比（dpr） = 物理像素 / 设备独立像素

## 16. 媒体查询

主要是通过查询设备的宽度来执行不同的 css 代码，最终达到界面的配置。

@media mediatype and | not | only (media feature) {

CSS-Code;

}

@media screen and (max-width: 600px) { /\*当屏幕尺寸小于600px时，应用下面的CSS样式\*/

/\* css代码\*/

}

//屏幕大于400px的时候就引用\*\*\*.css这个外部的css样式文件

<link type="text/css" href="\*\*\*.css" media="screen and (min-width: 400px)">

Mediatype媒体类型：

|  |  |
| --- | --- |
| 值 | 描述 |
| all (常用) | 用于所有设备 |
| screen (常用) | 用于电脑屏幕，平板电脑，智能手机等。 |
| speech | 应用于屏幕阅读器等发声设备 |
| print | 用于打印机和打印预览 |

优点：

* 可以做到设备像素比的判断，方法简单，成本低，特别是对移动和PC维护同一套代码的时候。目前像Bootstrap等框架使用这种方式布局。
* 图片便于修改，只需修改css文件
* 调整屏幕宽度的时候不用刷新页面即可响应式展示

缺点：

* 代码量比较大，维护不方便
* 为了兼顾大屏幕或高清设备，会造成其他设备资源浪费，特别是加载图片资源
* 为了兼顾移动端和PC端各自响应式的展示效果，难免会损失各自特有的交互方式

注意⚠：需要初始化设置:< meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">

name="viewport"：标记显示设备为视口；

width = device-width：宽度等于当前设备的宽度；

initial-scale：初始的缩放比例（默认设置为1.0）；

minimum-scale：允许用户缩放到的最小比例（默认设置为1.0）；

maximum-scale：允许用户缩放到的最大比例（默认设置为1.0）；

user-scalable：用户是否可以手动缩放（默认设置为no，不希望用户放大缩小页面）。

## 17. Less预处理语言

因为CSS需要书写大量看似没有逻辑的代码，不方便维护及扩展，不利于复用，因此衍生了预处理语言。它们让 CSS变成一门可以使用变量、循环、继承、自定义方法等多种特性的标记语言，简化了 CSS 的编写，并且降低了CSS的维护成本，逻辑性得以增强。

使用方式：

CDN下载引入：

<link rel="stylesheet/less" href="style.less">

<script src="less.min.js"></script>

npm下载包： npm install -g less

预处理语言：Sass、Less 、Stylus

Sass 与 Stylus 相比于 Less 功能更为丰富，但Less更易入门学习。Less 没有去掉任何 CSS 的功能，而是在现有的语法上，增添了许多额外的功能特性。

变量使用原则 ：多次频繁出现的值、需要修改的值设为变量。

1.首先需要声明变量使用 @变量名：变量值；

@color: #999; @width: 50%; @bgColor: skyblue;//不要添加引号

2.使用变量 属性值设为@变量名：

。class { color: @color; width: @width; }

less中的变量类型：

1. 数字类 1 10px ②字符串：无引号字符串 red 和有引号字符串"gdak"

③颜色类 red #000000 rgb() ④值列表类型，用逗号或空格分隔

## 18. vh与vw，vmin与vmax

vw、vh、vmin、vmax 是一种视窗单位，也是相对单位。它相对的不是父节点或者页面的根节点。而是由视窗（Viewport）大小来决定的，视窗是你的浏览器实际显示内容的区域，其单位 1，代表1%。（可说为适配而生）

vw：视窗宽度的百分比（1vw 代表视窗的宽度为 1%） vh：视窗高度的百分比

vmin：当前 vw 和 vh 中较小的一个值。 vmax：当前 vw 和 vh 中较大的一个值。

vw、vh 与 % 百分比的区别：

% 是相对于父元素的大小设定的比率，vw、vh 是视窗大小决定的。

vw、vh 优势在于能够直接获取高度，而用 % 在没有设置 body 高度的情况下，是无法正确获得可视区域的高度的。

vmin、vmax 用处：

做移动页面开发时，如果使用 vw、wh 设置字体大小（比如 5vw），在竖屏和横屏状态下显示的字体大小是不一样的。

由于 vmin 和 vmax 是当前较小的 vw 和 vh 和当前较大的 vw 和 vh。这里就可以用到 vmin 和 vmax。使得文字大小在横竖屏下保持一致。

## 19. H5的语义化作用及语义化标签:

作用：

* 为了在没有CSS的情况下，页面也能呈现出很好地内容结构、代码结构.
* 有利于[SEO](http://baike.baidu.com/view/1047.htm)优化：和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息： [爬虫](http://baike.baidu.com/view/998403.htm)依赖于标签来确定上下文和各个关键字的权重；
* 方便其他设备解析（如屏幕阅读器、盲人阅读器、移动设备）以意义的方式来渲染网页；
* 更具可读性和可维护性，遵循W3C标准的团队都遵循这个标准，可以减少差异化。

常见标签：

1.header：用在页面的头部或者板块的头部，

2.footer：定义页脚，不仅是整个页面的页脚，也可用在某一章节。会包含创作者的姓名、文档的创作日期以及/或者联系信息。

3.nav：导航, 页面的导航链接区域。

4.hgroup：用于对网页或区段的标题元素（h1-h6）进行组合。

5.section：页面上的板块用于划分页面上不同区域，或者划分文章里的不同的章节.

6.article：显示一个独立的文章内容。artilce可以嵌套，则内层的artilce对外层的article标签有隶属的关系。

7.aside：用来装载非正文类的内容，比如侧边栏，广告

8.figure：用于对元素进行组合，一半用于图片或视频

9.time用来表现时间或日期

10.datalist：选项列表，与input元素配合使用可制作输入值的下拉列表，与select不同的是datalist即可下拉选择，也可以输入值，还有模糊匹配的功能。

11.details：用于描述文档或文档某个部分的细节，summary是元素的标题

属性：open：（定义 details 是否可见。）

12.address：定义文章，或页面作者的详细联系信息，样式是斜体

13.keygen：给表单添加一个公钥，只要用于后端加密

14.progress：定义进度条。

15. mark ：定义带有记号的文本。高亮显示文本。

16. figure标签规定独立的流内容（图像、图表、照片、代码等）.

17. figcaption ：定义 figure 元素的标题，用于对figure的内容 进行说明。

…

## 20. Web Worker和Web Socket

web socket提供更高效的传输协议，web worker提供多线程提高web应用计算效率。

Websocket：

WebSocket 是HTML5提供的一种在单个 TCP 连接上进行全双工通讯的协议。WebSocket 使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在 WebSocket API 中，浏览器和服务器只需要完成***一次握手***，两者之间就直接可以创建***持久性***的连接，自由进行数据传送和接收，并进行双向数据传输。不需要使用轮询的方式。

Ajax轮询：轮询是在特定的的时间间隔（如每1秒），由浏览器对服务器发出HTTP请求，然后由服务器返回最新的数据给客户端的浏览器。这种传统的模式带来很明显的缺点，即浏览器需要不断的向服务器发出请求，然而HTTP请求可能包含较长的头部，其中真正有效的数据可能只是很小的一部分，显然这样会浪费很多的带宽等资源。WebSocket能更好的节省服务器资源和带宽，并且能够更实时地进行通讯。

WebSocket 客户端基于事件的编程模型与 node 类似：

创建 WebSocket 对象：var Socket = new WebSocket(url, [protocol] );

第一个参数 url, 指定连接的 URL，它的url前缀是ws://（非加密） 或者wss://（SSL加密传输）。第二个参数 protocol 是可选的，指定了可接受的子协议。

通过 send() 方法来向服务器发送数据（send()方法只能接受字符串，json对象要先序列化成json字符串），close() 方法来关闭连接；

通过 onmessage 事件来接收服务器返回的数据，webSocket.onclose()连接关闭时会触发，.onopen()连接建立成功时触发。onerror()通信发生错误时触发。

数据传输过程: websocket的数据传输形式是：frame。比如会将一条消息分为几个frame，按照先后顺序传输出去。这样做会有几个好处：

（1）大数据的传输可以分片传输，不用考虑到数据大小导致的长度标志位不足够的情况。

（2）和http的[chunk](http://blog.csdn.net/xifeijian/article/details/42921827)一样，可以边生成数据边传递消息，即提高传输效率。

Websocket 使用和 HTTP 相同的 TCP 端口，可以绕过大多数防火墙的限制。默认情况下，Websocket 协议使用 80 端口；运行在 TLS 之上时，默认使用 443 端口。

与socket的区别：

Socket 其实并不是一个协议，是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。当两台主机通信时，让 Socket 去组织数据，以符合指定的协议。TCP 连接则更依靠于底层的 IP 协议，IP 协议的连接则依赖于链路层等更低层次。

而WebSocket 则是一个典型的应用层协议。

总的来说：Socket 是传输控制层协议，WebSocket 是应用层协议。

Web worker：

web worker 是运行在后台的 JavaScript，独立于其他脚本，不会影响页面的性能。

当在 HTML 页面中执行脚本时，页面的状态是不可响应的，直到脚本已完成。

除了DOM操作之外，理论上任何JS脚本任务都可放入worker中执行；语法上的限制，则是不能跨域访问JS。worker常用于需要消耗大量时间和CPU资源的复杂计算，以换来前台用户操作的友好型；换句话说，从用户体验上看，提高了服务性能。

使用方法：

worker的主线程和子线程间通过postMessage()来发送消息，通过向 web worker 添加一个 "onmessage" 事件监听器来获取接受到的消息;

创建 web worker 对象后，它会继续监听消息（即使在外部脚本完成之后）直到其被终止为止。如需终止 web worker，并释放浏览器/计算机资源，使用 terminate() 方法即可。

主线程调用worker.terminate()结束线程；Worker线程通过调用this.close()结束自身线程

// 创建一个Worker对象并向它传递将在新线程中执行的脚本的URL

var worker =new Worker("worker.js");

worker.postMessage("hello world"); //向worker发送数据

worker.onmessage =function(evt){ //接收worker传过来的数据函数

……

}

worker . terminate () ;

## 21. CSS3及相关动画

transition(过渡)：

包含四个属性值：

执行变换的属性：transition-property，

变换延续的时间：transition-duration，

在延续时间段，变换的速率变化：transition-timing-function，

变换延迟时间：transition-delay

transition需要事件触发，所以没法在网页加载时自动发生。

transition是一次性的，不能重复发生，除非一再触发。

// 规定width属性延迟2s后在5s内以ease-in的过渡效果开始过渡。

transition:width 5s ease-in 2s;

transform(变形)：

包括旋转rotate、扭曲skew、缩放scale和移动translate以及矩阵变形matrix。

transform: translateZ(0); 来开启硬件加速；过多开启会影响性能。

transform:rotate(9deg); //旋转9度

transform:translate(100px,-20px); //在x轴向右移动100px，y轴向上移动20px

animation：

@keyframes 规则用于创建动画。在 @keyframes 中规定某项 CSS 样式，就能创建由当前样式逐渐改为新样式的动画效果。

当在 @keyframes 中创建动画时，需要捆绑到某个选择器，否则不会产生动画效果。

通过规定至少以下两项 CSS3 动画属性，即可将动画绑定到选择器：

规定动画的名称； 规定动画的时长;

必须定义动画的名称和时长。如果忽略时长，则动画不会允许，因为默认值是 0。

用百分比来规定变化发生的时间，或用关键词 "from" 和 "to"，等同于 0% 和 100%。0% 是动画的开始，100% 是动画的完成。

为了得到最佳的浏览器支持，应该始终定义 0% 和 100% 选择器。animation 属性是一个简写属性，用于设置六个动画属性:

animation: name duration timing-function delay iteration-count direction;

|  |  |
| --- | --- |
| 属性 | 描述 |
| animation-name | 规定 @keyframes 动画的名称。 |
| animation-duration | 规定动画完成一个周期所花费的秒或毫秒。默认是 0。 |
| animation-timing-function | 规定动画的速度曲线。默认是 "ease"。 |
| animation-delay | 规定动画何时开始。默认是 0。 |
| animation-iteration-count | 规定动画被播放的次数。默认是 1。 |
| animation-direction | 规定动画是否在下一周期逆向地播放。默认是 "normal"。 |

div{

animation：myanimation 5s;

}

@keyframes myanimation{

0% {background-color:red;}

100% { background-color:green;}

}

CSS3动画与JS动画的区别：

JS动画：兼容更好

缺点：JavaScript在浏览器的主线程中运行，而其中还有很多其他需要运行的JavaScript、样式计算、布局、绘制等对其干扰。这也就导致了线程可能出现阻塞，从而造成丢帧的情况。需要引入各种库之类的；

优点：JavaScript的动画与CSS预先定义好的动画不同，可以在其动画过程中对其进行控制：开始、暂停、回放、中止、取消都是可以做到的。而且一些动画效果，比如视差滚动效果，曲线运动，冲击闪烁等，只有JavaScript能够完成。

实现的是帧动画，可以很细致的控制每一个效果；

支持复用，可以实现一个函数 重复调用函数切换不同的参数就可以；

CSS动画：兼容性不是很好

缺点：缺乏强大的控制能力。而且很难以有意义的方式结合到一起，使得动画变得复杂且易于出问题。

优点：浏览器可以对动画进行优化。它必要时可以创建图层，然后在主线程之外运行。

实现的是补间动画，对于细节的帧数把控不是很细致；

不支持复用，代码量大；

帧动画：使用定时器，每隔一段时间，更改当前的元素的状态。  
补间动画：过渡（加过渡只要状态发生改变产生动画）动画(多个节点来控制动画)性能会更好。

## 22. 如何实现响应式布局

媒体查询：最简单的方式，但是无法满足很多需求

借助原生Javascript：成本高，不推荐使用

第三方开源框架Bootstrap：

主要是利用栅格式布局的原理，会自动将屏幕分为12列。

* col-xs-\* 超小屏幕，手机 (宽度<768px)
* col-sm-\* 小屏幕，平板 (宽度≥768px)
* col-md-\* 中等屏幕，桌面显示器 (宽度≥992px)
* col-lg-\* 超大屏幕，(宽度>1200px)

## 23.响应式布局和自适应布局的区别：

响应式布局是实现不同屏幕分辨率的终端上浏览网页的不同展示方式。通过响应式设计能使网站在手机和平板电脑上有更好的浏览阅读体验。换句话说就是一个网站能够兼容多个终端，而不是为了每一个终端做一个特定的版本。

自适应布局是指能忘了使网页自适应的显示在不同大小终端设备上的新网页设计方式及技术，它需要开发多套界面来适应不同的终端。

自适应布局通过检测视口分辨率，来判断当前访问的设备是：pc端、平板、手机，从而请求服务层，返回不同的页面；

响应式布局通过检测视口分辨率，针对不同客户端在客户端做代码处理，来展现不同的布局和内容。

自适应布局需要开发多套界面，而响应式布局只需要开发一套界面就可以了。

自适应对页面做的屏幕适配是在一定范围：比如pc端一般要大于1024像素，手机端要小于768像素。而响应式布局是一套页面全部适应。

自适应布局如果屏幕太小会发生内容过于拥挤。而响应式布局正是为了解决这个问题而衍生出的概念，它可以自动识别屏幕宽度并做出相应调整的网页设计。

响应式布局是利用媒体查询，和基于内容的断点，调用不同的css样式来让网页适应不同分辨率的屏幕；响应式web设计则是网页的布局针对屏幕大小的尺寸而进行响应

自适应布局：自适应web设计是网页内容根据设备的不同而进行“适应”。

24.link和import的区别:

区别1：link是XHTML标签，除了加载CSS外，还可以定义RSS等其他事务；@import属于CSS范畴，只能加载CSS。

区别2：link引用CSS时，在页面载入时同时加载；@import需要页面网页完全载入以后加载。

区别3：link是XHTML标签，无兼容问题；@import是在CSS2.1提出的，低版本的浏览器不支持。

区别4：link支持使用[Javascript](https://baike.sogou.com/v2484.htm?ch=ww.wap.chain.baike)控制DOM去改变样式；而@import不支持。

## 25. SEO的概念及实现

SEO：搜索引擎优化。是一种利用搜索引擎的规则提高网站在有关搜索引擎内的自然排名的方式。SEO是帮助搜索爬虫和算法获得最大化访问。

网站优化可以分为两块：站外优化和站内优化，站内优化最重要。

- 合理的title、description、keywords：搜索对这三项的权重逐个减小，title值强调重点即可，重要关键词出现不要超过2次，而且要靠前，不同页面title要有所不同；description把页面内容高度概括，长度合适，不可过分堆砌关键词，不同页面description有所不同；keywords列举出重要关键词即可  
- 语义化的HTML代码，符合W3C规范：语义化代码让搜索引擎容易理解网页  
- 重要内容HTML代码放在最前：搜索引擎抓取HTML顺序是从上到下，有的搜索引擎对抓取长度有限制，保证重要内容一定会被抓取.

- 重要内容不要用js输出：爬虫不会执行js获取内容.

- 少用iframe：搜索引擎不会抓取iframe中的内容.

- 非装饰性图片必须加alt.

- 提高网站速度：网站速度是搜索引擎排序的一个重要指标

SEO优化：

标题与关键词：设置有吸引力切合实际的标题，标题中要包含所做的关键词。  
网站结构目录： 最好不要超过三级，每级有“面包屑导航”，使网站成树状结构分布

页面元素： 给图片标注"Alt"可以让搜索引擎更友好的收录

网站内容：每天有规律的更新网站的内容，最好是原创，会使搜索引擎更加喜欢

友情链接： 对方一定要是正规网站，每天有专业的团队或者个人维护更新

内链的布置：使网站形成类似蜘蛛网的结构，不会出现单独连接的页面或链接，提高搜索引擎的抓取率

外链建设：最好选择权重高的网站来发布，这样可以增加网站的流量。

流量分析： 通过统计工具(百度统计，CNZZ)分析流量来源，指导下一步的SEO

## 26. HTML5的新特性：

1. 语义标签 (见19)
2. 增强表单类型

email 应该包含 e-mail 地址的输入域。提交表单时，会自动验证 email 域的值。

url 用于应该包含 URL 地址的输入域。在提交表单时，会自动验证 url 域的值。

number 应该包含数值的输入域.

range 应该包含一定范围内数字值的输入域。显示为滑动条。

search 用于搜索域，比如站点搜索或 Google 搜索。显示为常规的文本域。

Date pickers (date, month, week, time, datetime, datetime-local)具体描述为下：

|  |  |
| --- | --- |
| date | 选取日、月、年 |
| month | 选取月、年 |
| week | 选取周和年 |
| time | 选取时间（小时和分钟） |
| datetime | 选取时间、日、月、年（UTC 时间） |
| datetime-local | 选取时间、日、月、年（本地时间） |

1. video (视频)、audio (音频)、embed (嵌入内容)
2. Canvas（标签只是图形容器，必须使用脚本来绘制图形。）
3. SVG：可伸缩的矢量图形
4. Geolocation（地理定位）用于定位用户的位置。

鉴于该特性可能侵犯用户的隐私，除非用户同意，否则用户位置信息是不可用的。

使用getCurrentPosition(success,error)方法来获取用户的位置。以实现“LBS服务”；

watchPosition(func) - 返回用户的当前位置，并继续返回用户移动时的更新位置

clearWatch() - 停止 watchPosition() 方法,清除定位监视。

属性：coords.latitude：十进制数的纬度 coords.longitude：十进制数的经度

coords.accuracy：位置精度 coords.altitude：海拔，海平面以上以米计.

1. 拖放API：

拖放是一种常见的特性，即抓取对象以后拖到另一个位置。在 HTML5 中，拖放是标准的一部分，任何元素都能够拖放。

拖放的过程分为源对象和目标对象。源对象是指你即将拖动元素，而目标对象则是指拖动之后要放置的目标位置。

|  |  |  |
| --- | --- | --- |
| Event | On Event Handler | 描述 |
| drag | ondrag | 当拖动元素或选中的文本时触发。 |
| dragend | ondragend | 当拖拽操作结束时触发 (比如松开鼠标按键或敲“Esc”键). (见结束拖拽) |
| dragenter | ondragenter | 当拖动元素或选中的文本到一个可释放目标时触发（见 指定释放目标）。 |
| dragexit | ondragexit | 当元素变得不再是拖动操作的选中目标时触发。 |
| dragleave | ondragleave | 当拖动元素或选中的文本离开一个可释放目标时触发。 |
| dragover | ondragover | 当元素或选中的文本被拖到一个可释放目标上时触发（每100毫秒触发一次）。 |
| dragstart | ondragstart | 当用户开始拖动一个元素或选中的文本时触发（见开始拖动操作）。 |
| drop | ondrop | 当元素或选中的文本在可释放目标上被释放时触发（见执行释放）。 |

让一个元素被拖动需要添加 draggable 属性，再加上全局事件处理函数ondragstart

<p draggable="true" ondragstart=" handler(event) ">This is draggable.</p>

1. Web Worker (见20)
2. WebSocket (见20)
3. Web storage (localStorage和sessionStorage)

27. Less和Sass使用：

28、继承属性与不可继承属性：

可以继承的属性很少，只有颜色，文字，字体间距行高对齐方式，和列表的样式可以继承。

所有元素可继承：visibility和cursor。

内联元素可继承：[letter-spacing](https://www.baidu.com/s?wd=letter-spacing&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1m4PHwhPjf1PWczujDk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLn1bknjmsn1cdn104Pjc4rf)、 [word-spacing](https://www.baidu.com/s?wd=word-spacing&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1m4PHwhPjf1PWczujDk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLn1bknjmsn1cdn104Pjc4rf)、 [white-space](https://www.baidu.com/s?wd=white-space&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1m4PHwhPjf1PWczujDk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLn1bknjmsn1cdn104Pjc4rf)、 [line-height](https://www.baidu.com/s?wd=line-height&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1m4PHwhPjf1PWczujDk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLn1bknjmsn1cdn104Pjc4rf)、 color、font、 [font-family](https://www.baidu.com/s?wd=font-family&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1m4PHwhPjf1PWczujDk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLn1bknjmsn1cdn104Pjc4rf)、font-size、font-style、font-variant、[font-weight](https://www.baidu.com/s?wd=font-weight&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1m4PHwhPjf1PWczujDk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLn1bknjmsn1cdn104Pjc4rf)、[text-decoration](https://www.baidu.com/s?wd=text-decoration&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1m4PHwhPjf1PWczujDk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLn1bknjmsn1cdn104Pjc4rf)、text-transform、direction。

终端块状元素可继承：text-indent和text-align。

列表元素可继承：list-style、list-style-type、list-style-position、list-style-image。

其余都为不可继承属性.

29.HTML与XHTML的区别：

* XHTML 元素必须被正确地嵌套。
* XHTML 元素必须被关闭。（非空标签必须使用结束标签。）
* XHTML标签名必须用小写字母。（XHTML标签名和属性对大小写敏感。）
* XHTML 文档必须拥有根元素。

## 30.浏览器兼容：

1、不同浏览器的标签默认的外补丁( margin )和内补丁(padding)不同

解决方案： css 里增加通配符 \* { margin: 0; padding: 0; }

2、IE6双边距问题；在 IE6中设置了float , 同时又设置margin , 就会出现边距问题

解决方案：设置display:inline;

3、当标签的高度设置小于10px，在IE6、IE7中会超出自己设置的高度

解决方案：超出高度的标签设置overflow:hidden,或者设置line-height的值小于你的设置高度

4、图片默认有间距

解决方案：使用float 为img 布局

5、IE9一下浏览器不能使用opacity

解决方案：

opacity: 0.5; filter: alpha(opacity = 50);

filter: progid:DXImageTransform.Microsoft.Alpha(style = 0, opacity = 50);

6、边距重叠问题；当相邻两个元素都设置了margin 边距时，margin 将取最大值，舍弃最小值；

解决方案：为了不让边重叠，可以给子元素增加一个父级元素，并设置父级元素为overflow:hidden；

7、cursor:hand 显示手型在safari 上不支持

解决方案：统一使用 cursor:pointer

8、两个块级元素，父元素设置了overflow:auto；子元素设置了position:relative ;且高度大于父元素，在IE6、IE7会被隐藏而不是溢出；

解决方案：父级元素设置position:relative

9、hack：是通过在CSS样式中加入一些特殊的符号，让不同的浏览器识别不同的符号，以达到应用不同的CSS样式的目的。使用hack可以把浏览器分为3类：IE6 ；IE7和遨游；[其他](http://xiazai.zol.com.cn/others%20mobilesoft_soft_index/others%20mobilesoft_page_1.html)（IE8 [Chrome](http://xiazai.zol.com.cn/detail/33/327560.shtml) ff [Safari](http://xiazai.zol.com.cn/detail/37/366660.shtml) [opera](http://xiazai.zol.com.cn/detail/34/339345.shtml)等）

IE6认识的hack 是下划线\_ 和星号 \*

IE7 和遨游认识的hack是星号 \*

因为优先级相同且冲突的属性设置后一个会覆盖掉前一个，所以书写的次序是很重要的。

# 四、Vue.js

1. V-if和V-show的区别

v-if：动态为元素添加删除标签元素来切换显示隐藏，有较高的切换渲染，是真正的条件渲染，可结合v-else-if和v-else等指令一起使用。

v-show：切换元素的display属性来切换元素的显示与隐藏。有较高的初始渲染。

## vue生命周期以及钩子函数

总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后： beforeCreated阶段，vue实例的挂载元素$el和data都为undefined，还未初始化。在created阶段，vue实例的数据对象data初始化了，$el还没有。

载入前/后：在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。

更新前/后：当data变化时，会触发beforeUpdate和updated方法。

销毁前/后：在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

## vue如何实现数据的双向绑定

VueJS使用 ES5提供的 Object.defineProperty方法，监控对数据的操作，从而可以自动触发数据同步。并且，由于是在不同的数据上触发同步，可以精确的将变更发送给绑定的视图，而不是对所有的数据都执行一次检测。

vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

具体步骤：

第一步：需要observe的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter和getter；这样给这个对象的某个值赋值，就会触发setter，那么就能监听到数据变化。

第二步：compile解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图。

第三步：Watcher订阅者是Observer和Compile之间通信的桥梁，主要做的事情是:

1、在自身实例化时往属性订阅器(dep)里面添加自己

2、自身必须有一个update()方法

3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

第四步：MVVM作为数据绑定的入口，整合Observer、Compile和Watcher三者，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

Object.defineProperty(obj, prop, descriptor): 直接在一个对象上定义一个新属性，或者修改一个对象的现有属性， 并返回这个对象。

obj：要在其上定义属性的对象。

prop：要定义或修改的属性的名称。

descriptor：将被定义或修改的属性描述符。

属性描述符：数据描述符和存取描述符。描述符必须是这两种形式之一；不能同时是两者。

公共属性：configurable：当且仅当该属性的 configurable 为 true 时，该属性描述符才能够被改变，同时该属性也能从对应的对象上被删除。默认为 false。

enumerable：当且仅当该属性的enumerable为true时，该属性才能够出现在对象的枚举属性中。默认为 false。

数据描述符是一个具有值的属性，该值可能是可写的，也可能不是可写的。

value：该属性对应的值。可以是任何有效的 JavaScript 值（数值，对象，函数等）。默认为 undefined。

writable：当且仅当该属性的writable为true时，value才能被赋值运算符改变。默认为 false。

存取描述符是由getter-setter函数对描述的属性。

get：一个给属性提供 getter 的方法，如果没有 getter 则为 undefined。当访问该属性时，该方法会被执行，方法执行时没有参数传入，但是会传入this对象（由于继承关系，这里的this并不一定是定义该属性的对象）。默认为 undefined。

set：一个给属性提供 setter 的方法，如果没有 setter 则为 undefined。当属性值修改时，触发执行该方法。该方法将接受唯一参数，即该属性新的参数值。默认为 undefined。

Object.defineProperty(obj, "b", {

get : function(){

return bValue;

},

set : function(newValue){

bValue = newValue;

},

enumerable : true,

configurable : true

});

## vue父子组件之间传值

父组件向子组件传值：$ref； props：注重数据传递

子组件向父组件传值：$emit()

1. 非父子组件之间的传值：
   1. 、bus总线机制

const bus = new Vue() // bus也是一个vue实例对象

②、使用vuex状态管理器

## Vue-loader：

Vue-loader是基于webpack解析.vue文件的一个加载器loader，解析和转换 .vue 文件，提取出其中的逻辑代码 script、样式代码 style、以及 HTML 模版 template，再分别把它们交给对应的 Loader 去处理。最后将他们组装成一个commonjs模块；module.exports出一个vue.js组件对象。

vue-loader 的作用：

1. 允许为 Vue 组件的每个部分使用其它的 webpack loader，例如在 <style> 的部分使用 Sass 和在 <template> 的部分使用 Pug；
2. 允许在一个 .vue 文件中使用自定义块，并对其运用自定义的 loader 链；
3. 使用 webpack loader 将 <style> 和 <template> 中引用的资源当作模块依赖来处理；
4. 为每个组件模拟出 scoped CSS；
5. 在开发过程中使用热重载来保持状态。

## Vue中的slot插槽使用

slot：父组件需要在子组件内放一些DOM，那么这些DOM是显示、不显示、在哪个地方显示、如何显示，就是slot分发负责的活。插槽显不显示、怎样显示是由父组件来控制的，而插槽在哪里显示就由子组件来进行控制。

插槽用于将所携带的内容，插入到指定的某个位置，从而使模板分块，具有模块化的特质和更大的重用性。

$destroy()销毁组件，没有销毁slot。

单个插槽：当子组件模板只有一个没有属性的插槽时，父组件传入的整个内容片段将插入到插槽所在的 DOM 位置，并替换掉插槽标签本身。

最初在 <slot> 标签中的任何内容都被视为备用内容。备用内容在子组件的作用域内编译，并且只有在宿主元素为空，且没有要插入的内容时才显示备用内容。

具名插槽：solt元素可以用一个特殊的特性name来进一步配置如何分发内容。多个插槽可以有不同的名字。  
这样可以将父组件模板中 slot 位置，和子组件 slot 元素产生关联，便于插槽内容对应传递。

作用域插槽(scoped slots)：可以访问组件内部数据的可复用插槽(reusable slot)  
在父级中，具有特殊特性 slot-scope 的<template> 元素必须存在，表示它是作用域插槽的模板。slot-scope 的值将被用作一个临时变量名，此变量接收从子组件传递过来的 prop 对象。

## 计算属性computed和 watch 的区别:

计算属性是依赖的值改变会重新执行函数，计算属性是取返回值作为最新结果，所以里面不能异步的返回结果。不能写异步逻辑。

侦听属性是侦听的值改变会重新执行函数，将一个值重新赋值作为最新结果，所以赋值的时候可以进行一些异步操作。

所以区别来源于用法，只是需要动态值，那就用计算属性；需要知道值的改变后执行业务逻辑，才用 watch，用反或混用虽然可行，但都是不正确的用法。

computed 是一个对象时，它有哪些选项？  
computed 和 methods 有什么区别？  
computed 是否能依赖其它组件的数据？  
watch 是一个对象时，它有哪些选项？

1. 有get和set两个选项
2. methods是一个方法，它可以接受参数，而computed不能，computed是可以缓存的，methods不会。
3. computed可以依赖其他computed，甚至是其他组件的data
4. watch 配置  
   handler  
   deep 是否深度  
   immeditate 是否立即执行

应用场景:

watch：侦听一个数的变化，当该数据变化时，来处理其他与之相关数据的变化（该数据影响别的多个数据）

computed：获得一个值或结果，该结果受其他依赖的影响。（一个数据受多个数据影响）

## vue-router路由实现原理

router-link：页面中相当于a链接的标签，to跳转时url中不用加#

router-view：显示路由配置path对应的组件内容

实现路由的方式有两种：vue通过参数mode来设置，默认是hash模式。

hash模式：

使用 URL 的 hash 来模拟一个完整的 URL，于是当 URL 改变时，页面不会重新加载。

每次hash值的变化会触发hashchange这个事件，通过这个事件就可以知道 hash 值发生了哪些变化，然后便可以监听hashchange来实现更新页面部分内容的操作。

history模式：单页路由的 url 就不会多出一个#，变得更加美观。

这种模式充分利用 history.pushState API 来完成 URL 跳转而无须重新加载页面。

当使用 history 模式时，URL 就像正常的 url，例如 http://yoursite.com/user/id

不过这种模式还需要后台配置支持。因为我们的应用是单页客户端应用，如果后台没有正确的配置，当用户在浏览器直接访问 http://oursite.com/user/id 就会返回 404

所以要在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面，这个页面就是app 依赖的页面。

history.pushState()与hash比较：

1. pushState设置的新url可以是与当前url同源的任意url，而hash只可修改#后面的部分，故可设置与当前同文档的url
2. pushState设置的新url可以与当前url一模一样，这样也会把记录添加到栈中，而hash设置的新的值必须与原来不一样才会触发记录添加到栈中
3. pushState通过stateObject可以添加任意类型的数据记录中，而hash只可添加短字符串
4. pushState可额外设置title属性供后续使用。

## 动态路由匹配：

需要把某种模式匹配到的所有路由，全都映射到同个组件。使用 “动态路径参数”来实现。

一个“路径参数”使用冒号 : 标记。当匹配到一个路由时，参数值会被设置到 this.$route.params，可以在每个组件内使用。

定义动态路由：在router目录下的index.js文件中，对path属性加上…/:\*\*\*。

路由组件传参：

组件中获取动态路由参数：使用router对象的params.id：$route.params.\*\*\*

但是在组件中直接使用 this.$route.params 会降低组件复用的灵活性。

更好的办法是将变量以props的形式传递给组件。 也更加方便组件的重用和测试。

有3种在路由配置中给组件传递props的模式：布尔模式，对象模式，函数模式

## vue-router如何响应路由参数的变化:

当使用动态路由参数时，例如从 /content?id=1 到 content?id=2，此时原来的组件实例会被复用。这也意味着组件的生命周期钩子不会再被调用，此时vue应该如何响应路由参数的变化？

方法一：使用watch的$route对象

watch: { '$route' (to, from) { // 对路由变化作出响应... } }

//路径发生变化，$route会重新赋值，监控了这个属性，会执行函数内代码

方法二：引入beforeRouteUpdate守卫：beforeRouteUpdate (to, from, next){ }

## vue-router导航钩子函数

“导航”表示路由正在发生改变。

vue-router 提供的导航守卫主要用来通过跳转或取消的方式守卫导航。有多种机会植入路由导航过程中。参数或查询的改变并不会触发进入/离开的导航守卫。

导航钩子有：

全局导航钩子：前置守卫（beforeEach）、解析守卫（beforeResolve）、后置钩子（afterEach）

前置守卫：router.beforeEach ( (to, from, next) => { // ... } );

每个守卫方法接收三个参数：

to: Route:即将要进入的目标 路由对象

from: Route: 当前导航正要离开的路由

next: Function，一定要调用该方法来resolve这个钩子。执行效果依赖 next 方法的调用参数。确保要调用 next方法，否则钩子就不会被 resolved，如果不用就拦截

* next(): 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是confirmed (确认的)。
* next(false): 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 from 路由对应的地址。
* next(’/’) 或者 next({ path: ‘/’ }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: ‘home’ 之类的选项以及任何用在router-link的 to prop或 router.push中的选项。
* next(error): (2.4.0+) 如果传入 next 的参数是一个 Error 实例，则导航会被终止且该错误会被传递给 router.onError()注册过的回调。

解析守卫：在导航被确认之前，同时在所有组件内守卫和异步路由组件被解析之后，解析守卫就被调用。

全局后置钩子：和守卫不同的是，这些钩子不会接受 next 函数也不会改变导航本身。

router.afterEach((to, from) => { // ... });

路由配置的守卫：beforeEnter: (to, from, next) => { // ... }

组件内的导航钩子：beforeRouteEnter、beforeRouterUpdate、beforeRouteLeave 、afterEnter

beforeRouteEnter (to, from, next) { } ：在渲染该组件的对应路由被 confirm 前调用；不能获取组件实例 `this` ；因为当守卫执行前，组件实例还没被创建。

beforeRouteUpdate (to, from, next) { }：在当前路由改变，但是该组件被复用时调用；举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候；由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用；可以访问组件实例 `this`。

beforeRouteLeave (to, from, next) { }：导航离开该组件的对应路由时调用；可以访问组件实例 `this`。

完整的导航解析流程：

1. 导航被触发。
2. 在失活的组件里调用离开守卫。
3. 调用全局的 beforeEach 守卫。
4. 在重用的组件里调用 beforeRouteUpdate 守卫 (2.2+)。
5. 在路由配置里调用 beforeEnter。
6. 解析异步路由组件。
7. 在被激活的组件里调用 beforeRouteEnter。
8. 调用全局的 beforeResolve 守卫 (2.5+)。
9. 导航被确认。
10. 调用全局的 afterEach 钩子。
11. 触发 DOM 更新。
12. 用创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数。

## nextTick()：

在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后，立即使用这个回调函数，获取更新后的DOM。

两种写法：全局：Vue.nextTick(callback)； 局部：vm/this.$nextTick(callback)

vue中数据和dom渲染由于是异步的，所以要让dom结构随数据改变这样的操作都应该放进this.$nextTick()的回调函数中。

应用场景：

1.在Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中，因为在created()钩子函数执行的时候DOM并未进行任何渲染，此时进行DOM操作无异于徒劳，所以一定要将DOM操作的js代码放进Vue.nextTick()的回调函数中。这样就相当于是在mouted里调用这个生命钩子， mounted钩子函数执行时所有的DOM挂载和渲染都已完成，此时在该钩子函数中进行任何DOM操作都不会有问题 。

2.在数据变化后要执行的某个操作，而这个操作需要使用随数据改变而改变的DOM结构的时候，这个操作都应该放进Vue.nextTick()的回调函数中。在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。

注意⚠： mounted 不会承诺所有的子组件也都一起被挂载。如果希望等到整个视图都渲染完毕，可以用 vm.$nextTick 替换掉 mounted

## 对Vue.js的template编译的理解：

简述：先转化成AST树，再得到的render函数返回VNode（Vue的虚拟DOM节点）。

AST：抽象语法树，是源代码的抽象语法结构的树状表现形式，这里特指编程语言的源代码。

createCompiler：用以创建编译器，返回值是compile以及compileToFunctions。

Compile：是一个编译器，会将传入的template转换成对应的AST树、render函数以及staticRenderFns函数。另外compile还负责合并option（createCompiler会根据平台区分传入一个baseOptions，compile会将平台自有的option与本身传入的option进行合并得到最终的finalOptions）。

首先，通过compile编译器把template编译成AST语法树；

然后，AST会经过generate（将AST语法树转化成render funtion字符串的过程）得到render函数，render的返回值是VNode，VNode是Vue的虚拟DOM节点，里面有（标签名、子节点、文本等等）。

## Keep-alive：

keep-alive是Vue提供的一个抽象组件。它自身不会被渲染成一个DOM元素，也不会出现在父组件链中；使用keep-alive包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们，避免了组件反复创建和渲染，有效提升系统性能。总的来说，keep-alive用于保存组件的渲染状态。

当组件在keep-alive内被切换时组件的activated、deactivated这两个生命周期钩子函数会被执行。这两个钩子函数只在使用keep-alive之后才会有，否则不会存在。

activated()：被包裹组件被激活时调用，在vue对象存活的情况下，进入当前存在activated()函数的页面时，一进入页面就触发；可用于初始化页面数据等。

deactivated()：被包裹组件被移除时调用

被包裹在keep-alive中的组件的状态将会被保留，例如我们将某个列表类组件内容滑动到第100条位置，那么我们在切换到一个组件后再次切换回到该组件，该组件的位置状态依旧会保持在第100条列表处。

用法：

<keep-alive :include="whiteList" :exclude="blackList" :max="amount">

<component :is="currentComponent"></component>

或者<router-view></router-view>

</keep-alive>

include定义缓存白名单，keep-alive会缓存命中的组件；exclude定义缓存黑名单，被命中的组件将不会被缓存；max定义缓存组件上限，超出上限使用LRU的策略置换缓存数据。

## vuex相关



vuex：vue框架中状态管理模式。

应用场景：中大型单页应用中，多个组件共享数据或者是跨组件之间需要传递数据.通过状态（数据源）集中管理驱动组件的变化。

Vuex的状态和属性：

有五种状态： State、 Getter、Mutation 、Action、 Module （就是mapAction等）

vuex的流程：

当组件进行数据修改的时候需要调用dispatch来触发actions里面的方法。actions里面的每个方法中都会有一个commit方法，当方法执行的时候会通过commit来触发mutations里面的方法进行数据的修改。mutations里面的每个函数都会有一个state参数，这样就可以在mutations里面进行state的数据修改，当数据修改完毕后，会传导给页面。页面的数据也会发生改变。

特性：

Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。

不能直接改变 store 中的状态。改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。

state：就是存放数据/状态的地方，类似一个容器仓库

action：包含任意异步操作，通过提交 mutation 间接更变状态而不是直接变更状态。

Action通过 store.dispatch()方法分发action中的方法，action中的commit可以触发mutations中的方法.若是同步，则直接在组件中commit触发vuex中mutations的方法。

mutation： 必须是同步函数, 更改 store 中的状态的唯一方法是提交 mutation。

Vuex 中的 mutation 非常类似于事件：每个 mutation 都有一个字符串的 事件类型 (type) 和一个回调函数。这个回调函数就是实际进行状态更改的地方，并且它会接受 state 作为第一个参数：store.commit('increment')

Getter：可以认为是 store 的计算属性。就像计算属性一样，getter 的返回值会根据它的依赖被缓存起来，且只有当它的依赖值发生了改变才会被重新计算。

Module: 将 store 分割成模块（module）。每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块——从上至下进行同样方式的分割.

vuex和本地存储的区别:

vuex 可以实时改变其他依赖状态值的状态，本地存储不能.

刷新页面之后vuex数据消失，本地存储不会

## axios相关

axios：是一个请求后台资源的模块。返回一个promise对象。axios.get() / post()发送请求

axios.post(‘api/user’).then()：跨域添加用户操作；

axios.put(‘api/user/8′)：跨域更新操作

1. vue.cli项目中src目录每个文件夹和文件的用法

assets文件夹是放静态资源；components是放组件；router是定义路由相关的配置;view视图；app.vue是一个应用主组件；main.js是入口文件

# 五、前端性能优化

35条雅虎军规：

1. 尽量减少 HTTP 请求个数——须权衡
2. 使用 CDN（内容分发网络）
3. 为文件头指定 Expires 或 Cache-Control ，使内容具有缓存性。
4. 避免空的 src 和 href
5. 使用 gzip 压缩内容
6. 把 CSS 放到顶部
7. 把 JS 放到底部
8. 避免使用 CSS 表达式
9. 将 CSS 和 JS 放到外部文件中
10. 减少 DNS 查找次数
11. 精简 CSS 和 JS
12. 避免跳转
13. 剔除重复的 JS 和 CSS
14. 配置 ETags
15. 使 AJAX 可缓存
16. 尽早刷新输出缓冲
17. 使用 GET 来完成 AJAX 请求
18. 延迟加载
19. 预加载
20. 减少 DOM 元素个数
21. 根据域名划分页面内容
22. 尽量减少 iframe 的个数
23. 避免 404
24. 减少 Cookie 的大小
25. 使用无 cookie 的域
26. 减少 DOM 访问
27. 开发智能事件处理程序
28. 用 <link> 代替 @import
29. 避免使用滤镜
30. 优化图像
31. 优化 CSS Spirite
32. 不要在 HTML 中缩放图像——须权衡
33. favicon.ico要小而且可缓存
34. 保持单个内容小于25K
35. 打包组件成复合文本

## 七大手段：

包括减少请求数量、减小资源大小、优化网络连接、优化资源加载、减少重绘回流、使用性能更好的API和构建优化

1. 减少请求数量

【合并】：如果不进行文件合并，有如下3个隐患：

1、文件与文件之间有插入的上行请求，增加了N-1个网络延迟

2、受丢包问题影响更严重

3、经过代理服务器时可能会被断开

但是，文件合并本身也有自己的问题

1、首屏渲染问题

2、缓存失效问题

所以，对于文件合并，有如下改进建议

1、公共库合并

2、不同页面单独合并

【图片处理】：

1.雪碧图（CSS sprite精灵）：当整合图片比较大时，一次加载比较慢。随着字体图片、SVG图片的流行，该技术渐渐退出了历史舞台.

原理：将许多的小图片整合到一张大图片中，利用css中的background-image属性，background-position属性定位某个图片位置，来达到在大图片中引用某个部位的小图片的效果。

优点：

* 减少网页的http请求，提升网页加载速度。
* 合并多张小图片成大图，能减少字节总数（大图大小<=多张小图大小）.

缺点：

* 前期需要处理图片将小图合并，多些许工程量。
* 对于需要经常改变的图片维护起来麻烦。

2.Base64：将图片的内容以Base64格式内嵌到HTML中，把每三个8Bit的字节转换为四个6Bit的字节，也就是将资源原本二进制形式转成以64个字符基本单位，所组成的一串字符串。

优点：

* base64的图片会随着html或者css一起下载到浏览器,减少了HTTP请求.
* 可解决跨域问题

缺点：

* 老东西（低版本）的IE浏览器不兼容。
* 由于Base64用8位字符表示信息中的6个位，体积会比原来的图片约大33%.
* css中过多使用base64图片会使得css过大，不利于css的加载。

3.使用字体图标来代替图片

【减少重定向】：

尽量避免使用重定向，当页面发生了重定向，就会延迟整个HTML文档的传输。在HTML文档到达之前，页面中不会呈现任何东西，也没有任何组件会被下载，降低了用户体验

如果一定要使用重定向，如http重定向到https，要使用301永久重定向，而不是302临时重定向。因为，如果使用302，则每一次访问http，都会被重定向到https的页面。而永久重定向，在第一次从http重定向到https之后 ，每次访问http，会直接返回https的页面

【使用缓存】：

使用cach-control或expires这类强缓存时，缓存不过期的情况下，不向服务器发送请求。强缓存过期时，会使用last-modified或etag这类协商缓存，向服务器发送请求，如果资源没有变化，则服务器返回304响应，浏览器继续从本地缓存加载资源；如果资源更新了，则服务器将更新后的资源发送到浏览器，并返回200响应

【不使用CSS @import】：

CSS的@import会造成额外的请求

【避免使用空的src和href】

a标签设置空的href，会重定向到当前的页面地址

form设置空的method，会提交表单到当前的页面地址

1. 减少资源大小

【压缩】

1、HTML压缩：HTML代码压缩就是压缩在文本文件中有意义，但是在HTML中不显示的字符，包括空格，制表符，换行符等

2、CSS压缩：CSS压缩包括无效代码删除与CSS语义合并

3、JS压缩与混乱：JS压缩与混乱包括无效字符及注释的删除、代码语义的缩减和优化、降低代码可读性，实现代码保护

4、图片压缩：针对真实图片情况，舍弃一些相对无关紧要的色彩信息

【webp】

在安卓下可以使用webp格式的图片，它具有更优的图像数据压缩算法，能带来更小的图片体积，同等画面质量下，体积比jpg、png少了25%以上，而且同时具备了无损和有损的压缩模式、Alpha 透明以及动画的特性

【开启gzip】

HTTP协议上的GZIP编码是一种用来改进WEB应用程序性能的技术。大流量的WEB站点常常使用GZIP压缩技术来让用户感受更快的速度。这一般是指WWW服务器中安装的一个功能，当有人来访问这个服务器中的网站时，服务器中的这个功能就将网页内容压缩后传输到来访的电脑浏览器中显示出来。一般对纯文本内容可压缩到原大小的40%

1. 优化网络连接

【使用CDN】：CDN全称是Content Delivery Network，即内容分发网络

它能够实时地根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。其目的是使用户可就近取得所需内容，解决 Internet网络拥挤的状况，提高用户访问网站的响应速度

【使用DNS预解析】

当浏览器访问一个域名的时候，需要解析一次DNS，获得对应域名的ip地址。在解析过程中，按照浏览器缓存、系统缓存、路由器缓存、ISP(运营商)DNS缓存、根域名服务器、顶级域名服务器、主域名服务器的顺序，逐步读取缓存，直到拿到IP地址

DNS预解析(Prefetch)就是根据浏览器定义的规则，提前解析之后可能会用到的域名，使解析结果缓存到系统缓存中，缩短DNS解析时间，来提高网站的访问速度。

【并行连接】

由于在HTTP1.1协议下，chrome每个域名的最大并发数是6个。使用多个域名，可以增加并发数

【持久连接】

使用keep-alive或presistent来建立持久连接，持久连接降低了时延和连接建立的开销，将连接保持在已调谐状态，而且减少了打开连接的潜在数量

【管道化连接】

在HTTP2协议中，可以开启管道化连接，即单条连接的多路复用，每条连接中并发传输多个资源，这里就不需要添加域名来增加并发数了

1. 优化资源加载

【资源加载位置】

通过优化资源加载位置，更改资源加载时机，使尽可能快地展示出页面内容，尽可能快地使功能可用

1、CSS文件放在head中，先外链，后本页

2、JS文件放在body底部，先外链，后本页

3、处理页面、处理页面布局的JS文件放在head中，如babel-polyfill.js文件、flexible.js文件

4、body中间尽量不写style标签和script标签

【资源加载时机】

1、异步script标签

defer:  异步加载，在HTML解析完成后执行。defer的实际效果与将代码放在body底部类似

async: 异步加载，加载完成后立即执行

2、模块按需加载

在SPA等业务逻辑比较复杂的系统中，需要根据路由来加载当前页面需要的业务模块。按需加载，是一种很好的优化网页或应用的方式。这种方式实际上是先把代码在一些逻辑断点处分离开，然后在一些代码块中完成某些操作后，立即引用或即将引用另外一些新的代码块。这样加快了应用的初始加载速度，减轻了它的总体体积，因为某些代码块可能永远不会被加载。

3、使用资源预加载preload和资源预读取prefetch

preload让浏览器提前加载指定资源，需要执行时再执行，可以加速本页面的加载速度。

prefetch告诉浏览器加载下一页面可能会用到的资源，可以加速下一个页面的加载速度。

4、资源懒加载与资源预加载

资源延迟加载也称为懒加载，延迟加载资源或符合某些条件时才加载某些资源。

资源预加载是提前加载用户所需的资源，保证良好的用户体验。

资源懒加载和资源预加载都是一种错峰操作，在浏览器忙碌的时候不做操作，浏览器空间时，再加载资源，优化了网络性能。

1. 减少重绘回流
2. 使用性能更好的API（要注意其浏览器兼容性）

1、用对选择器

2、使用requestAnimationFrame来替代setTimeout和setInterval

　　希望在每一帧刚开始的时候对页面进行更改，目前只有使用 requestAnimationFrame 能够保证这一点。使用 setTimeout 或者 setInterval 来触发更新页面的函数，该函数可能在一帧的中间或者结束的时间点上调用，进而导致该帧后面需要进行的事情没有完成，引发丢帧。

3、使用IntersectionObserver来实现图片可视区域的懒加载

　　传统的做法中，需要使用scroll事件，并调用getBoundingClientRect方法，来实现可视区域的判断，即使使用了函数节流，也会造成页面回流。使用IntersectionObserver，则没有上述问题。

4、使用web worker

　　客户端javascript一个基本的特性是单线程：比如，浏览器无法同时运行两个事件处理程序，它也无法在一个事件处理程序运行的时候触发一个计时器。Web Worker是HTML5提供的一个javascript多线程解决方案，可以将一些大计算量的代码交由web Worker运行，从而避免阻塞用户界面，在执行复杂计算和数据处理时，这个API非常有用

1. 构建优化

【打包公共代码】

使用CommonsChunkPlugin插件，将公共模块拆出来，最终合成的文件能够在最开始的时候加载一次，便存到缓存中供后续使用。这会带来速度上的提升，因为浏览器会迅速将公共的代码从缓存中取出来，而不是每次访问一个新页面时，再去加载一个更大的文件

webpack 4 将移除 CommonsChunkPlugin, 取而代之的是两个新的配置项 optimization.splitChunks 和 optimization.runtimeChunk;通过设置 optimization.splitChunks.chunks: "all" 来启动默认的代码分割配置项

【动态导入和按需加载】

webpack提供了两种技术通过模块的内联函数调用来分离代码，优先选择的方式是，使用符合 ECMAScript 提案 的 import() 语法。第二种，则是使用 webpack 特定的 require.ensure

【剔除无用代码】

tree shaking 是一个术语，通常用于描述移除 JavaScript 上下文中的未引用代码(dead-code)。它依赖于 ES2015 模块系统中的静态结构特性，例如 import 和 export。这个术语和概念实际上是兴起于 ES2015 模块打包工具 rollup

JS的tree shaking主要通过uglifyjs插件来完成，CSS的tree shaking主要通过purify CSS来实现的

【长缓存优化】

1、将hash替换为chunkhash，这样当chunk不变时，缓存依然有效

2、使用Name而不是id

每个 module.id 会基于默认的解析顺序(resolve order)进行增量。也就是说，当解析顺序发生变化，ID 也会随之改变

为什么不使用表格布局？

* 更多的标签，增加文件大小；
* 不易维护，无法适应响应式设计；
* 性能考量，默认的表格布局算法会产生大量重绘

# 六、HTTP与计算机网络

* 1. TCP/IP协议分层管理:

应用层: 决定了向用户提供应用服务时通信的活动。

TCP/IP协议族内预存了各类通用的应用服务。比如，FTP（文件传输协议）和DNS（域名系统）服务就是其中的两类。HTTP协议也处于该层。

传输层:

传输层对上层应用层，提供处于网络连接中的两台计算机之间的数据传输。

在传输层有两个性质不同的协议：TCP（传输控制协议）和UDP（用户数据报协议）。

RTP（也属于传输层）： 实时传输协议，为数据提供了具有实时特征的端对端传送服务，如在组播或单播网络服务下的交互式视频音频或模拟数据。

网络层: 用来处理在网络上流动的数据包。数据包是网络传输的最小数据单位。该层规定了通过怎样的路径到达对方计算机，并把数据包传送给对方。

与对方计算机之间通过多台计算机或网络设备进行传输时，网络层所起的作用就是在众多的选项内选择一条传输路线。

数据链路层: 用来处理网络的硬件部分。包括控制操作系统、硬件的设备驱动、NIC（网络适配器，即网卡）及光纤等物理可见部分。硬件上的范畴均在链路层的作用范围内。

## TCP和UDP的异同：

相同点： UDP协议和TCP协议都是传输层协议。

TCP（传输控制协议）：

（1）TCP是面向连接的传输层协议

（2）TCP提供可靠的服务

（3）TCP是面向字节流的

（4）TCP 提供双全工通信

（5）TCP 连接智能有两个端点，即使点对点（一对一）

（6）TCP的首部开销小，只有8个字节，比TCP20个字节要短。

UDP（用户数据报协议）：

（1）UDP是无连接

（2）UDP使用尽最大努力交付，即不保证可靠性

（3）UDP是面向报文的

（4）UDP 没有拥塞控制，所以即使网络出现拥塞，也不会使源主机的发送速率降低。

（5）UDP 支持一对一、一对多、多对一、多对多的交互通信。

（6）UDP的首部开销小，只有8个字节，比TCP20个字节要短。

UDP应用场景：

1.面向数据报方式

2.网络数据大多为短消息

3.拥有大量Client

4.对数据安全性无特殊要求

5.网络负担非常重，但对响应速度要求高

## 三次握手四次挥手机制及原因：

三次握手：

第一次：A 的TCP客户进程也是首先创建传输模块TCB，然后向B发出连接请求报文段，这时候首部的同步位SYN=1，同时选择一个初始序号seq=x 。TCP 规定 SYN段报文不带数据，单要消耗一个序号，同时客户端进入SYN-SENT状态

第二次：B 收到连接请求报文后，如果同意连接则向A发送确认，在确认报文中把SYN  ACK都置1 ，确认号是ack=x+1 ，同时也为自己选择一个初始序号seq=y ，   同样这个报文段不能携带数据，但是要消耗一个序号，这时候服务器进入SYN-RECD 状态。

 第三次：客户端接受到服务器的确认后，还要向服务器再确认一次，确认报文段的ACK=1 确认号为ack=y+1 ，序号为seq=x+1 ，这时候已经建立连接，客户端进入ESTABLISHED状态，当服务器接受到客户端的确认后也进入ESTABLISHED状态。

为什么最后客户端还要发送一次请求确认：

为了防止已经失效的连接请求报文突然又传送到了服务器，从而产生错误。

四次挥手：

第一次：A客户端向服务器发送释放连接请求，并停止发送数据，主动关闭TCP连接，A把连接释放报文段首部的中止控制位置FIN值1 ，其许好为seq=u 等于前面已经传送过来的数据的最后一个字节的序号加1，这时候A进入FIN-WAIT-1状态，等待B的确认。

第二次：B服务器收到连接释放报文后即发出确认，确认号是ack=u+1 ，自己的序号为seq=v 等于B前面已经传送过来的数据的最后一个字节的序号加1 ，这时候B服务器进入CLOSED-WAIT状态（等待关闭状态），这时候TCP处于半关闭状态，即A没有数据可发送了，

单B要发送数据A仍要接受，也就是说A到B的连接断了，B到A的没有，这个状态可能会持续一段时间。

第三次：A收到B的确认后，就进入FIN-WAIT-2状态（终止等待2状态）等待B发送连接释放报文段。若B没有数据向A发送了，则通知TCP释放，发出的连接释放报文段使FIN=1 ，现在假定B的序号为w（在半关闭状态B可能又发送了一些数据）。B还必须发送上次的确认号ack=u+1,这时候B就进入LAST-ACK （最后确认）状态，等待A的确认。

第四次：A 收到B的连接释放报文后必须对此进行确认，在确认报文中把ACK置1，确认号ack=w+1 ，而自己的序号为seq=w+1 ，然后A进入TIME-WAIT（时间等待）状态。注意：此时TCP的连接没有断掉，必须经过时间等待器设置的时间2MSL后，A才进入CLOSED状态。

为什么在TIME-WAIT 状态必须等待2MSL的时间：

首先：保证A发送的最后一个ACK报文段能够到达B服务器。这个报文段有可能会丢失，处在LAST-ACK状态的B收不到A发送的确认报文，B超时后会重传这个FIN+ACK报文段，A就会在2MSL时间内接受报文，重新发送确认报文，重启2MSL计时器，最后A和B都进入CLOSED状态。如果A 发送确认后不等待一段时间就释放连接的话，则确认报文丢失后，A就接受不到B重传的FIN+ACK报文，因而也不会在发送一次确认报文，这样B就无法进入CLOSED状态。

第二：防止已失效的连接请求报文段出现在本连接中，A在发送完最后一个ACK报文后，再经过2MSL后，就可以使本链接持续时间内所产生的所有报文段都从网络中消失。

## HTTP方法:

1. GET 请求指定的页面信息，并返回实体主体。
2. HEAD类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
3. POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
4. PUT 从客户端向服务器传送的数据取代指定的文档的内容。
5. DELETE 请求服务器删除指定的页面。
6. CONNECT HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
7. OPTIONS 允许客户端查看服务器的性能。
8. TRACE 回显服务器收到的请求，主要用于测试或诊断。

## GET和POST的区别:

1. GET参数通过URL传递，POST放在Request body(消息主体)中。
2. GET在浏览器回退时是无害的，而POST会再次提交请求。
3. GET产生的URL地址可以被收藏为书签，而POST不可以。
4. GET请求会被浏览器主动缓存，而POST不会，除非手动设置。
5. GET请求只能进行url编码，而POST支持多种编码方式。
6. GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。
7. GET请求在URL中传送的参数是有长度限制的，而POST没有。（HTTP 协议没有 Body 和 URL 的长度限制，对 URL 限制的大多是浏览器和服务器的原因。）
8. 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
9. GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。（但都不是很安全，因为HTTP是明文传输，要想安全就用HTTPS）
10. GET请求数据在 URL 中对所有人都是可见的，而post数据不会显示在 URL 中。

但是：GET和POST本质上就是TCP链接（HTTP的底层是TCP/IP），并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

区别：GET产生一个TCP数据包；POST产生两个TCP数据包。（header 和 body 分开发送是部分浏览器或框架的请求方法，不属于 post 必然行为。）

解释：对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；

而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200（返回数据）。

因为POST需要两步，时间上消耗的要多一点，看起来GET比POST更有效。因此Yahoo团队有推荐用GET替换POST来优化网站性能。但这是一个坑！跳入需谨慎。为什么？

1. GET与POST都有自己的语义，不能随便混用。

2. 据研究，在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点。

3. 并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次。

## HTTP建立持久连接的意义：

HTTP/1.0 版的主要缺点是，每个TCP连接只能发送一个请求。发送数据完毕，连接就关闭，如果还要请求其他资源，就必须再新建一个连接。

HTTP/1.1允许HTTP设备在失误处理结束之后将TCP链接保持在打开状态，以方便未来的HTTP请求可以重复使用现存的连接, 不用声明Connection: keep-alive（这是HTTP1.0中为了解决复用而引用的非标准办法）。持久连接会在不同事务之间保持打开状态，直到客户端或服务器决定将其关闭为止。在最后一个请求发送Connection: close关闭连接。

虽然1.1版允许复用TCP连接，但是同一个TCP连接里面，在一条TCP连接上，所有的数据通信是按次序进行的。服务器只有处理完一个回应，才会进行下一个回应。要是前面的回应特别慢，后面就会有许多请求排队等着。这称为“队头堵塞”。

避免队头阻塞：

HTTP/2 复用TCP连接，在一个连接里，客户端和浏览器都可以同时发送多个请求或回应，而且不用按照顺序一一对应，这样就避免了"队头堵塞"。

举例来说，在一个TCP连接里面，服务器同时收到了A请求和B请求，于是先回应A请求，结果发现处理过程非常耗时，于是就发送A请求已经处理好的部分， 接着回应B请求，完成后，再发送A请求剩下的部分。这样双向的、实时的通信，就叫做多工。

因为 HTTP/2 的数据包是不按顺序发送的，同一个连接里面连续的数据包，可能属于不同的回应。因此，必须要对数据包做标记，指出它属于哪个回应。

HTTP/2 将每个请求或回应的所有数据包，称为一个数据流（stream）。每个数据流都有一个独一无二的编号。数据包发送的时候，都必须标记数据流ID，用来区分它属于哪个数据流。另外还规定，客户端发出的数据流，ID一律为奇数，服务器发出的，ID为偶数。

数据流发送到一半的时候，客户端和服务器都可以发送信号（RST\_STREAM帧），取消这个数据流。1.1版取消数据流的唯一方法，就是关闭TCP连接。这就是说，HTTP/2 可以取消某一次请求，同时保证TCP连接还打开着，可以被其他请求使用。

客户端还可以指定数据流的优先级。优先级越高，服务器就会越早回应。

1.1 版还引入了管道机制（pipelining），即在同一个TCP连接里面，客户端可以同时发送多个请求。这样就进一步改进了HTTP协议的效率。

举例来说，客户端需要请求两个资源。以前的做法是，在同一个TCP连接里面，先发送A请求，然后等待服务器做出回应，收到后再发出B请求。管道机制则是允许浏览器同时发出A请求和B请求，但是服务器还是按照顺序，先回应A请求，完成后再回应B请求。

持久连接的优点：

降低了时延和连接建立的开销；

避免了慢启动的拥塞适应阶段，可以快速进行数据的传输。

## HTTP状态码：

1\*\* 信息，服务器收到请求，需要请求者继续执行操作

2\*\* 成功，操作被成功接收并处理

3\*\* 重定向，需要进一步的操作以完成请求

4\*\* 客户端错误，请求包含语法错误或无法完成请求

5\*\* 服务器错误，服务器在处理请求的过程中发生了错

* 100 Continue    继续。客户端应继续其请求
* 101 Switching Protocols 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议
* 200 OK  请求成功。一般用于GET与POST请求
* 201 Created 已创建。成功请求并创建了新的资源
* 202 Accepted    已接受。已经接受请求，但未处理完成
* 203 Non-Authoritative Information   非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本
* 204 No Content  无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
* 205 Reset Content   重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
* 206 Partial Content 部分内容。服务器成功处理了部分GET请求
* 300 Multiple Choices    多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
* 301 Moved Permanently永久重定向。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
* 302 Found临时重定向。表示请求的资源临时搬到了其他位置，与301类似。但资源只是临时被移动。客户端应继续使用原有URI
* 303 See Other   查看其它地址。与301类似。使用GET和POST请求查看
* 304 Not Modified：未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
* 305 Use Proxy   使用代理。所请求的资源必须通过代理访问
* 306 Unused  已经被废弃的HTTP状态码
* 307 Temporary Redirect  临时重定向。与302类似。使用GET请求重定向
* 400 Bad Request 客户端请求的语法错误，服务器无法理解
* 401 Unauthorized    请求要求用户的身份认证
* 402 Payment Required    保留，将来使用
* 403 Forbidden   服务器理解请求客户端的请求，但是拒绝执行此请求
* 404 Not Found 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面
* 405 Method Not Allowed  客户端请求中的方法被禁止
* 406 Not Acceptable  服务器无法根据客户端请求的内容特性完成请求
* 407 Proxy Authentication Required   请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权
* 408 Request Time-out    服务器等待客户端发送的请求时间过长，超时
* 409 Conflict    服务器完成客户端的PUT请求是可能返回此代码，服务器处理请求时发生了冲突
* 410 Gone 客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置
* 411 Length Required 服务器无法处理客户端发送的不带Content-Length的请求信息
* 412 Precondition Failed 客户端请求信息的先决条件错误
* 413 Request Entity Too Large    由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息
* 414 Request-URI Too Large   请求的URI过长（URI通常为网址），服务器无法处理
* 415 Unsupported Media Type  服务器无法处理请求附带的媒体格式
* 416 Requested range not satisfiable 客户端请求的范围无效
* 417 Expectation Failed  服务器无法满足Expect的请求头信息
* 500 Internal Server Error   服务器内部错误，无法完成请求
* 501 Not Implemented 服务器不支持请求的功能，无法完成请求
* 502 Bad Gateway 充当网关或代理的服务器，从远端服务器接收到了一个无效的请求
* 503 Service Unavailable 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中
* 504 Gateway Time-out    充当网关或代理的服务器，未及时从远端服务器获取请求
* 505 HTTP Version not supported  服务器不支持请求的HTTP协议的版本，无法完成处理
  1. Web服务器及其组成
  2. HTTP报文的结构
  3. HTTP报文首部
  4. HTTP通用首部字段
  5. HTTP请求首部字段、响应首部字段、实体首部字段
  6. Cookie相关首部字段

## 13. HTTPS与HTTP区别及实现方式

HTTP：是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。

HTTPS：是以安全为目标的HTTP通道，简单讲是HTTP的安全版，即HTTP下加入SSL层，HTTPS的安全基础是SSL/TLS，因此加密的详细内容就需要SSL/TLS。

SSL标准化之后的名称改为 TLS，这两种是同一种协议。SSL协议位于TCP/IP协议与各种应用层协议之间，为数据通讯提供安全支持。SSL协议可分为两层：

SSL记录协议：它建立在可靠的传输协议（如TCP）之上，为高层协议提供数据封装、压缩、加密等基本功能的支持。

SSL握手协议：它建立在SSL记录协议之上，用于在实际的数据传输开始前，通讯双方进行身份认证、协商加密算法、交换加密密钥等。

HTTPS协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。

HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。简单来说，HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。

HTTPS和HTTP的区别主要如下：

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

客户端在使用HTTPS方式与Web服务器通信时有以下几个步骤:

（1）客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

（2）Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

（3）客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。

（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

（5）Web服务器利用自己的私钥解密出会话密钥。

（6）Web服务器利用会话密钥加密与客户端之间的通信。

HTTPS的优点:

尽管HTTPS并非绝对安全，掌握根证书的机构、掌握加密算法的组织同样可以进行中间人形式的攻击，但HTTPS仍是现行架构下最安全的解决方案，主要有以下几个好处：

（1）使用HTTPS协议可认证用户和服务器，确保数据发送到正确的客户机和服务器；

（2）HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。

（3）HTTPS是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。

（4）谷歌曾在2014年8月份调整搜索引擎算法，并称“比起同等HTTP网站，采用HTTPS加密的网站在搜索结果中的排名将会更高”。

HTTPS的缺点:虽然说HTTPS有很大的优势，但其相对来说，还是存在不足之处的：

（1）HTTPS协议握手阶段比较费时，会使页面的加载时间延长近50%，增加10%到20%的耗电；

（2）HTTPS连接缓存不如HTTP高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响；

（3）SSL证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用。

（4）SSL证书通常需要绑定IP，不能在同一IP上绑定多个域名，IPv4资源不可能支撑这个消耗。

（5）HTTPS协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用。最关键的，SSL证书的信用链体系并不安全，特别是在某些国家可以控制CA根证书的情况下，中间人攻击一样可行。

14. Cookie与Session  
15. 基于HTTP的功能追加协议（SPY、WebSocket、HTTP）  
16. 常见的Web攻击分类  
18. 存储机制localStorage、sessionStorage与Cookie存储技术  
19. XSS攻击及防御  
20. CSRF攻击及防御

## 21.DNS:

DNS就是将域名解析成IP地址。

* 主要用UDP，但是当请求字节过长超过512字节时用TCP协议，将其分割成多个片段传输。
* DNS协议默认端口号是53。
* 操作系统的DNS缓存：windows DNS缓存的默认值是 MaxCacheTTL，它的默认值是86400s，也就是一天。macOS 严格遵循DNS协议中的TTL。
* 游览器的DNS缓存：chrome对每个域名会默认缓存60s；IE将DNS缓存30min；Firefox默认缓存时间只有1分钟；Safari约为10S。

主机解析域名的顺序：

　　1、找缓存

　　2、找本机的hosts文件

　　3、找DNS服务器

域名解析总体可分为以下过程：

(1) 输入域名后, 先查找自己主机对应的域名服务器，域名服务器先查找自己的数据库中的数据.

(2) 如果没有， 就向上级域名服务器进行查找， 依次类推

(3) 最多回溯到根域名服务器, 肯定能找到这个域名的IP地址

(4) 域名服务器自身也会进行一些缓存， 把曾经访问过的域名和对应的IP地址缓存起来, 可以加速查找过程。

具体可描述如下：

1、当客户机提出查询请求时，首先在本地计算机的缓存中查找。如果在本地无法获得查询信息，则将查询请求发给DNS服务器。

2、首先客户机将域名查询请求发送到本地DNS服务器,当本地DNS服务器接到查询后，首先在该服务器管理的区域的记录中查找，如果找到该记录，则利用此记录进行解析；如果没有区域信息可以满足查询要求，服务器在本地的缓存中查找。

3、如果本地服务器不能在本地找到客户机查询的信息，将客户机请求发送到根域名DNS服务器。

4、根域名服务器负责解析客户机请求的根域部分，它将包含下一级域名信息的DNS服务器地址返回给客户机的DNS服务器地址。

5、客户机的DNS服务器利用根域名服务器解析的地址访问下一级DNS服务器，得到再下一级域我的DNS服务器地址。

6、按照上述递归方法逐级接近查询目标，最后在有目标域名的DNS服务器上找到相应IP地址信息。

7、客户机的本地DNS服务器将递归查询结果返回客户机。

8、客户机利用从本地DNS服务器查询得到的IP访问目标主机，就完成了一个解析过程。

递归查询：本机向本地域名服务器发出一次查询请求，就静待最终的结果。如果本地域名服务器无法解析，自己会以DNS客户机的身份向其它域名服务器查询，直到得到最终的IP地址告诉本机

迭代查询：本地域名服务器向根域名服务器查询，根域名服务器告诉它下一步到哪里去查询，然后它再去查，每次它都是以客户机的身份去各个服务器查询。

# 七、前端工程化

1. 前端工程化的流程（架构选型、业务开发、测试、打包构建、部署上线、项目监控）  
2. Webpack基本概念与配置  
3. loader与plugin原理与实现  
4. Webpack的模块热替换及实现  
5. Webpack的优化问题  
6. SPA及其优缺点  
7. SSR实现及优缺点

## 8.设计模式（工厂模式、单例模式、原型模式、代理模式、适配器模式、观察者模式等...）

## 9.模块化：commonjs、AMD、CMD

# 八、手撕的代码部分

1. Promise（A+规范）、then、all方法  
2. Iterator遍历器实现  
3. Thunk函数实现（结合Generator实现异步）  
4. async实现原理（spawn函数）  
5. class的继承  
6. 防抖和节流  
7. Ajax原生实现  
8. 深拷贝的几种方法与比较  
9. 继承的几种实现与比较  
10. 未知宽高的元素水平垂直居中  
11. 三栏布局的实现  
12. 两栏布局的实现

## 13. 数组去重

使用set有什么好处

14. 几种排序算法的实现及其复杂度比较  
15. 前序后序遍历二叉树（非递归）  
16. 二叉树深度遍历（分析时间复杂度）  
17. 跨域的实现（JSONP、CORS）

## 19.提取URL中的get参数

先提取url中？后面的字符串再以&分割，得到的是a=1这种情况，再以=分割得到[a,1]，

20. 判断是否字符串回文

# 九、数据可视化

## 1. Canvas和SVG的区别

2. 在考虑设计可视化图表时，结合Canvas和SVG特性会怎么取舍  
3. 常见的可视化组件库  
4. 可视化组件库如Echarts的设计思路  
5. 一些偏向底层的可视化组件库和前端框架结合方面需要考虑哪些问题  
6. 可视化组件如何做到数据驱动？