前端面试题：

1，一些开放性题目

1）自我介绍：除了基本个人信息以外，面试官更想听的是你与众不同的地方和你的优势。

2）项目介绍

3）如何看待前端开发?

4）平时是如何学习前端开发的?

5）未来三到五年的规划是怎样的?

2，什么是函数柯里化?

柯里化，是函数式编程的一个重要概念。它既能减少代码冗余，也能增加可读性。另外，附带着还能用来装逼。

先给出柯里化的定义：在数学和计算机科学中，柯里化是一种将使用多个参数的一个函数转换成一系列使用一个参数的函数的技术。

柯里化的定义，理解起来有点费劲。为了更好地理解，先看下面这个例子：

```
function sum (a, b, c) {
    console.log(a + b + c);
}
sum(1, 2, 3); // 6
```

毫无疑问，sum 是个简单的累加函数，接受3个参数，输出累加的结果。

假设有这样的需求，sum的前2个参数保持不变，最后一个参数可以随意。那么就会想到，在函数内，是否可以把前2个参数的相加过程，给抽离出来，因为参数都是相同的，没必要每次都做运算。

如果先不管函数内的具体实现，调用的写法可以是这样：`sum(1, 2)(3);`或这样`sum(1, 2)(10);`。就是，先把前2个参数的运算结果拿到后，再与第3个参数相加。

这其实就是函数柯里化的简单应用。

3，创建对象有几种方法?

1、字面量对象 // 默认这个对象的原型链指向object

var o1 = {name: '01'};

2、通过new Object声明一个对象

var o11 = new Object({name: '011'});

3、使用显式的构造函数创建对象

o2的构造函数是M

o2这个普通函数，是M这个构造函数的实例

4、object.create()

原型、构造函数、实例、原型链

1、Object.prototype属性是整个原型链的顶端

2、原型链通过prototype原型和**proto**属性来查找的.

3、所有的引用类型（数组、对象、函数），都具有对象特性，即可自由扩展属性（除了“null”以外)。

4、所有的引用类型（数组、对象、函数)，都有一个**proto**属性,属性值是一个普通的对象（null除外)。

5、所有的函数，都有prototype属性，属性值也是一个普通的对象。

6、所有的引用类型（数组、对象、函数），**proto**属性指向它的构造函数prototype属性值

7、实例本身的属性和方法如果没有找到，就会去找原型对象的属性和方法。如果在某一级找到了，就会停止查找，并返回结果

4，怎样通过ES5及ES6声明一个类?

1、传统的构造函数，声明一个类

```
function Animal() {
    this.name = 'name';
}
```

2、es6中的class声明

```
class Animal2{
    constructor() {
        this.name = name;
    }
}
```

5，call、apply的共同点与区别?

1、改变了函数运行上下文

2、call()和apply()主要是能扩充函数赖以运行作用域。两者的作用方式相同，它们的区别在于接收参数的方式不同，对于call()而言，第一个参数this与apply()相同，其他的参数必须直接传给函数，要一个一个的列出来，而对于apply()来说，apply()可以接收一个数组或arguments对象。所以如何选择二者，在于哪种给函数传参数的方式最简单。

6，用javascript实现对象的继承,继承的几种方式,这几种方式的优缺点?

方法1：借助构造函数实现继承（部分继承）

```
/**
 * 借助构造函数实现继承
 */
function Parent1() {
    this.name = 'parent';
}
Parent1.prototype.say = function() {}; // 不会被继承
function Child1() {
    // 继承：子类的构造函数里执行父级构造函数
    // 也可以用apply
    // parent的属性都会挂载到child实例上去
    // 借助构造函数实现继承的缺点：①如果parent1除了构造函数里的内容，还有自己原型链上的东西，自己原型链上的东西不会被child1继承
    // 任何一个函数都有prototype属性,但当它是构造函数的时候，才能起到作用(构造函数是有自己的原型链的) Parent1.call(this);
    this.type = 'child1';
}
console.log(new Child1);
```

(1)如果父类的属性都在构造函数内，就会被子类继承。

(2)如果父类的原型对象上有方法，子类不会被继承。

方法2：借助原型链实现继承

```
/**
 * 借助原型链实现继承
 */
function Parent2() {
    this.name = 'name';
    this.play = [1, 2, 3]
}
function Child2() {
    this.type = 'child2';
}
Child2.prototype = new Parent2(); // prototype使这个构造函数的实例能访问到原型对象上
console.log(new Child2().__proto__);
console.log(new Child2().__proto__ === Child2.prototype); // true

var s1 = new Child2(); // 实例
var s2 = new Child2();
console.log(s1.play, s2.play);
s1.play.push(4);

console.log(s1.__proto__ === s2.__proto__); // true // 父类的原型对象
```

(1)原型链的基本原理：构造函数的实例能访问到它的原型对象上

(2)缺点：原型链中的原型对象，是共用的

方法3：组合方式

```
/**
```

```
 * 组合方式
 */
function Parent3() {
    this.name = 'name';
    this.play = [1, 2, 3];
}
function Child3() {
    Parent3.call(this);
    this.type = 'child3';
}
Child3.prototype = new Parent3();
var s3 = new Child3();
var s4 = new Child3();
s3.play.push(4);
console.log(s3.play, s4.play);
// 父类的构造函数执行了2次
// 构造函数体会自动执行,子类继承父类的构造函数体的属性和方法
```

①组合方式优化1：

```
/**
 * 组合继承的优化方式1：父类只执行了一次
 */
function Parent4() {
    this.name = 'name';
    this.play = [1, 2, 3];
}
function Child4() {
    Parent4.call(this);
    this.type = 'child4';
}
Child4.prototype = Parent4.prototype; // 继承父类的原型对象
var s5 = new Child4();
var s6 = new Child4();
console.log(s5 instanceof Child4, s5 instanceof Parent4); // true
console.log(s5.constructor); // Parent4 //prototype里有个constructor属性，子类和 父类的原型对象就是同一个对象， s5的constructor就是父类的constructor
```

②组合方式优化2（最优解决方案）：

```
/**
 * 组合继承优化2
 */
function Parent5() {
    this.name = 'name';
    this.play = [1, 2, 3];
}
function Child5() {
    Parent5.call(this);
    this.type = 'child5';
}
Child5.prototype = Object.create(Parent5.prototype); // Object.create创建的对象 就是参数 Child5.prototype.constructor = Child5; var s7 = new Child5();
console.log(s7 instanceof Child5, s7 instanceof Parent5);
console.log(s7.constructor); // 构造函数指向Child5
```

优缺点：

原型链继承的缺点

1、字面量重写原型

一是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

2、借用构造函数（类式继承）

借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

3、组合式继承

组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

7，说说你对作用域链的理解?

作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到window对象即被终止，作用域链向下访问变量是不被允许的。

8，谈一谈this在各种情况的指向问题?

1、作为构造函数执行

```
function Foo(name) {
    this.name = name;
}
var f = new Foo('zhangsan');
```

2、作为对象属性执行

```
var obj = {
    name: 'A',
    printName: function() {
        console.log(this.name);
    }
}
obj.printName();
```

3、作为普通函数执行

```
function fn() {
    console.log(this);
}
fn();
```

4、call apply bind

```
function fn1(name, age) {
    alert(name);
    console.log(this);
}
fn1.call({x: 100}, 'zhangsan', 20);
fn1.apply({x:100}, ['zhangsan', 20])
var fn2 = function (name, age) {
    // 必须是函数表达式，不能是函数声明，即不能是function fn2(name, age) {}
    alert(name); console.log(this);
}.bind({y:200});
fn2('zhangsan', 20);
```

9，闭包的特征有哪些?

1、函数嵌套函数

2.、函数内部可以引用外部的参数和变量

3、参数和变量不会被垃圾回收机制回收

10，闭包应用场景有哪些?

1、作为返回值

```
function fn() {
    var max = 10;
    return function (x) {
        if (x > max) {
            console.log(x);
        }
    }
}
var f1 = fn();
f1(15);
```

2、作为参数传递

```
var max = 10;
function fn(x) {
    if (x > max) {
        console.log(x);
    }
}(function(f) { var max = 100; f(15); })(fn);
```

11，实际开发中闭包的应用?

闭包实际应用中主要用于封装变量，收敛权限

```
function isFirstLoad() {
    var _list = []; // 有_的变量说明是私有变量，函数内部使用的
    return function(id) {
```

```
        if (_list.indexOf(id) >=0) {
            // 也可用includes
            return false;
        } else {
            _list.push(id);
            return true;
        }
    }
}
// 使用
var firstLoad = isFirstLoad();
console.log(firstLoad(10)); // true
console.log(firstLoad(10)); // false
console.log(firstLoad(20)); // true
// 你在isFirstLoad函数外面，根本不可能修改掉_list的值
```

12，如何理解js的单线程?

只有一个线程，同一时间只能做一件事情。

13，js为什么是单线程的?

避免dom渲染的冲突

1、浏览器需要渲染dom

2、js可以修改dom结构

3、js执行的时候，浏览器dom渲染会暂停

4、两段js也不能同时执行（都修改dom就冲突了）

5、webworder支持多线程，但是不能访问dom

14，同步和异步的区别是什么？分别举一个同步和异步的例子?

1、同步会阻塞代码执行，而异步不会。

2、alert是同步，setTimeout是异步。

同步：指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程将会一直等待下去，直到收到返回信息才继续执行下去；

异步：指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。

15，什么是任务队列?

任务队列（task queue）主要分两种：

1、宏任务（macrotask）：在新标准中叫task

（1)主要包括：script(整体代码)，setTimeout，setInterval，setImmediate，I/O，ui rendering

2、微任务（microtask）：在新标准中叫jobs

（1）主要包括：process.nextTick， Promise，MutationObserver（html5新特性）

扩展：

1、同步任务：在主线程上，排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务;

2、异步任务：不进入主线程，而进入“任务队列”（task queue）的任务，只有“任务队列”通知主线程，某个异步任务可以执行了，该任务才会进入主线程执行。

16，栈和队列的区别?

1、栈的插入和删除操作都是在一端进行的，而队列的操作却是在两端进行的。

2、队列先进先出，栈先进后出。

3、栈只允许在表尾一端进行插入和删除，而队列只允许在表尾一端进行插入，在表头一端进行删除

17，栈和堆的区别?

1、栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。

堆区（heap） — 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收。

2、堆（数据结构）：堆可以被看成是一棵树，如：堆排序;

栈（数据结构）：一种先进后出的数据结构。

event loop

18，判断数据类型的方法有哪四种?(列出四种即可)

在 ECMAScript 规范中，共定义了 7 种数据类型，分为 基本类型 和 引用类型 两大类，如下所示：

> **基本类型**：String、Number、Boolean、Symbol、Undefined、Null
>
> **引用类型**：Object

基本类型也称为简单类型，由于其占据空间固定，是简单的数据段，为了便于提升变量查询速度，将其存储在栈中，即按值访问。

引用类型也称为复杂类型，由于其值的大小会改变，所以不能将其存放在栈中，否则会降低变量查询速度，因此，其值存储在堆(heap)中，而存储在变量处的值，是一个指针，指向存储对象的内存处，即按址访问。引用类型除 Object 外，还包括 Function 、Array、RegExp、Date 等等。

鉴于 ECMAScript 是松散类型的，因此需要有一种手段来检测给定变量的数据类型。对于这个问题，JavaScript 也提供了多种方法，但遗憾的是，不同的方法得到的结果参差不齐。

下面介绍常用的4种方法，并对各个方法存在的问题进行简单的分析。

1、typeof

typeof 是一个操作符，其右侧跟一个一元表达式，并返回这个表达式的数据类型。返回的结果用该类型的字符串(全小写字母)形式表示，包括以下 7 种：number、boolean、symbol、string、object、undefined、function 等。

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

有些时候，typeof 操作符会返回一些令人迷惑但技术上却正确的值：

- 对于基本类型，除 null 以外，均可以返回正确的结果。
- 对于引用类型，除 function 以外，一律返回 object 类型。
- 对于 null ，返回 object 类型。
- 对于 function 返回 function 类型。

其中，null 有属于自己的数据类型 Null ， 引用类型中的 数组、日期、正则 也都有属于自己的具体类型，而 typeof 对于这些类型的处理，只返回了处于其原型链最顶端的 Object 类型，没有错，但不是我们想要的结果。

2、instanceof

instanceof 是用来判断 A 是否为 B 的实例，表达式为：A instanceof B，如果 A 是 B 的实例，则返回 true,否则返回 false。 在这里需要特别注意的是：**instanceof 检测的是原型**，我们用一段伪代码来模拟其内部执行过程：

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

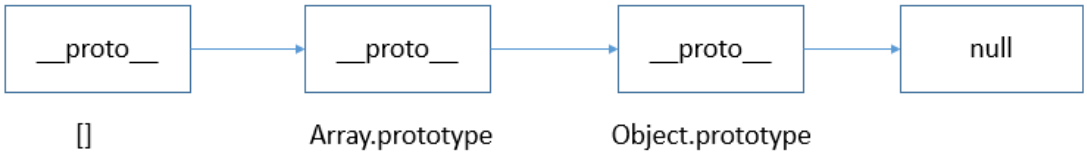
从上述过程可以看出，当 A 的 **proto** 指向 B 的 prototype 时，就认为 A 就是 B 的实例，我们再来看几个例子：

```
[] instanceof Array; // true
{} instanceof Object;// true
new Date() instanceof Date;// true

function Person(){};
new Person() instanceof Person;

[] instanceof Object; // true
new Date() instanceof Object;// true
new Person instanceof Object;// true
```

我们发现，虽然 instanceof 能够判断出 [ ] 是Array的实例，但它认为 [ ] 也是Object的实例，为什么呢?

我们来分析一下 [ ]、Array、Object 三者之间的关系:

从 instanceof 能够判断出 [ ].**proto** 指向 Array.prototype，而 Array.prototype.**proto** 又指向了 Object.prototype，最终 Object.prototype.**proto** 指向了null，标志着原型链的结束。因此，[]、Array、Object 就在内部形成了一条原型链:



从原型链可以看出，[] 的 **proto** 直接指向Array.prototype，间接指向 Object.prototype，所以按照 instanceof 的判断规则，[] 就是Object的实例。依次类推，类似的 new Date()、new Person() 也会形成一条对应的原型链 。因此，**instanceof 只能用来判断两个对象是否属于实例关系**，而不能判断一个对象实例具体属于哪种类型。**

instanceof 操作符的问题在于，它假定只有一个全局执行环境。如果网页中包含多个框架，那实际上就存在两个以上不同的全局执行环境，从而存在两个以上不同版本的构造函数。如果你从一个框架向另一个框架传入一个数组，那么传入的数组与在第二个框架中原生创建的数组分别具有各自不同的构造函数。

```
var iframe = document.createElement('iframe');
document.body.appendChild(iframe);
xArray = window.frames[0].Array;
var arr = new xArray(1,2,3); // [1,2,3]
arr instanceof Array; // false
```

针对数组的这个问题，ES5 提供了 Array.isArray() 方法 。该方法用以确认某个对象本身是否为 Array 类型，而不区分该对象在哪个环境中创建。

```
if (Array.isArray(value)){
   //对数组执行某些操作
}
```

Array.isArray() 本质上检测的是对象的 [[Class]] 值，[[Class]] 是对象的一个内部属性，里面包含了对象的类型信息，其格式为 [object Xxx] ，Xxx 就是对应的具体类型 。对于数组而言，[[Class]] 的值就是 [object Array] 。

3、constructor

当一个函数 F被定义时，JS引擎会为F添加 prototype 原型，然后再在 prototype上添加一个 constructor 属性，并让其指向 F 的引用。如下所示:

```
> function F(){}
< undefined
> F.prototype
< ▼Object {}
    ▶constructor: function F()
    ▶__proto__: Object
>
```

当执行 var f = new F() 时，F 被当成了构造函数，f 是F的实例对象，此时 F 原型上的 constructor 传递到了 f 上，因此 f.constructor == F

```
> var f = new F()
< undefined
> f.constructor == F
< true
>
```

可以看出，F 利用原型对象上的 constructor 引用了自身，当 F 作为构造函数来创建对象时，原型上的 constructor 就被遗传到了新创建的对象上， 从原型链角度讲，构造函数 F 就是新对象的类型。这样做的意义是，让新对象在诞生以后，就具有可追溯的数据类型。

同样，JavaScript 中的内置对象在内部构建时也是这样做的：

```
> ''.constructor == String
< true
> new Number(1).constructor == Number
< true
> true.constructor == Boolean
< true
> new Function().constructor == Function
< true
> new Date().constructor == Date
< true
> new Error().constructor == Error
< true
> [].constructor == Array
< true
> document.constructor == HTMLDocument
< true
> window.constructor == Window
< true
>
```

细节问题：

> \1. null 和 undefined 是无效的对象，因此是不会有 constructor 存在的，这两种类型的数据需要通过其他方式来判断。
>
> \2. 函数的 constructor 是不稳定的，这个主要体现在自定义对象上，当开发者重写 prototype 后，原有的 constructor 引用会丢失，constructor 会默认为 Object

```
> function F(){}
< undefined
> F.prototype = {a:'xxxx'}
< Object {a: "xxxx"}
> var f = new F()
< undefined
> f.constructor == F
< false
> f.constructor
< function Object() { [native code] }
>
```

为什么变成了 Object?

因为 prototype 被重新赋值的是一个 { }，{ } 是 new Object() 的字面量，因此 new Object() 会将 Object 原型上的 constructor 传递给 { }，也就是 Object 本身。

因此，为了规范开发，在重写对象原型时一般都需要重新给 constructor 赋值，以保证对象实例的类型不被篡改。

4、toString

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

对于 Object 对象，直接调用 toString() 就能返回 [object Object] 。而对于其他对象，则需要通过 call / apply 来调用才能返回正确的类型信息。

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

19，js变量按照存储方式区分为哪些类型，并描述其特点?

1、存储在栈中：值类型。

2、存储在堆中：引用类型

引用类型的"数据"存储在堆中，

引用类型"指向堆中的数据的指针"存储在栈中。

20，js中有哪些内置函数/ 数据封装类对象?

内置函数：

Number、String、Boolean

Array、Object、Function

Date、RegExp、Error

21，js变量按照存储方式区分为哪些类型，并描述其特点?

1、值类型和引用类型。

2、值类型存储的是值， 赋值之后原变量的值不改变 。

3、 引用类型存储的是地址， 赋值之后是把原变量的引用地址赋值给新变量， 新变量改变原来的会跟着改变。

22，数组怎么去重有哪些？（方法）

1、使用数组方法indexOf来判断

```
function sele(arr){
    var temp = [];
    for( var i = 0 ; i < arr.length ; i++ ){
        if( temp.indexOf( arr[ i ] ) == -1 ){
            temp.push( arr[ i ] );
        }
    }
    return temp;
}
var arr = ['aa', 'bb', 'cc', '', 1, 0, '1', 1, 'bb', null, undefined, null];
console.log(sele(arr));
```

2个缺点，一是效率问题，因为加上indexOf相当于是2重循环，二是indexOf的兼容性问题:IE8-不兼容。

2、使用数组方法indexOf第二种方法 IE8-不兼容

```
function sele( arr ) {
    var temp = [];
    for( var i = 0 ; i < arr.length ; i++ ){
        if( arr.indexOf( arr[ i ] ) == i ){
            temp.push( arr[ i ] );
        }
    }
    return temp;
}
```

比方法（1）效率还要差

3、循环

```
function sele( arr ) {
    var temp = [];
    for( var i = 0 ; i < arr.length ; i++ ){
        for( var j = i + 1 ; j < arr.length ; j++ ){
            if( arr[ i ] === arr[ j ] ){
                j = ++i;
            }
        }
        temp.push( arr[ i ] );
    }return temp;
}
```

4、

```
function unique3(array) {
    var result = [];
    var hash = {};
    for(var i=0; i<array.length; i++) {
        var key = (typeof array[i]) + array[i];
        if(!hash[key]) {
            result.push(array[i]);
            hash[key] = true;
        }
    }
    return result;
}
var arr = ['aa', 'bb', 'cc', '', 1, 0, '1', 1, 'bb', null, undefined, null];
console.log(unique3(arr));
```

以上方法中之所以给key添加了类型前缀，是因为要区分'1'和1。

5、使用es6中includes方法

```
function sele( arr ) {
    var temp = [];
    arr.forEach(( v ) => {
        temp.includes( v ) || temp.push( v );
    })
    return temp;
}
```

6、es6的set

```
function unique(array){
    return Array.from(new Set(array));
}
// 或者写成（建议写法）
const unique = arr => [...new Set(arr)]
// 也可以是
const unique = arr => {return [...new Set(arr)]}
var arr = ['aa', , '', 1, 0, '1', 1, , null, undefined, null];
console.log(unique(arr));
```

23，一句话数组去重?

new Set()：Set本身是一个构造函数，用来生成Set数据结构

```
const unique = arr => [...new Set(arr)]
var arr = ['aa', , '', 1, 0, '1', 1, , null, undefined, null];
console.log(unique(arr));
```

24，哪些操作会造成内存泄漏?

1、垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收。

2、setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。

3、闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

25，js内存泄漏的解决方式

1、global variables：对未声明的变量的引用在全局对象内创建一个新变量。在浏览器中，全局对象就是 window。

```
function foo(arg) {
    bar = 'some text';
    // 等同于window.bar = 'some text';
}
```

（1）解决：

①创建意外的全局变量

```
function foo() {
    this.var1 = 'potential accident'
}
```

②可以在 JavaScript 文件开头添加 “use strict”，使用严格模式。这样在严格模式下解析 JavaScript 可以防止意外的全局变量。

③在使用完之后，对其赋值为 null 或者重新分配。

2、被忘记的 Timers 或者 callbacks

（1）解决：

3、闭包：一个可以访问外部（封闭）函数变量的内部函数。

（1）解决：

4、DOM 引用

(1)解决：

26，dom是哪种基本的数据结构?

树

27，dom结构操作/ 怎样添加、移除、移动、复制、创建和查找节点/ dom操作的

常用api?

1、创建新节点

```
createDocumentFragment() //创建一个DOM片段
createElement() //创建一个具体的元素
createTextNode() //创建一个文本节点
```

2、添加、移除、替换、插入

```
appendChild()
removeChild()
replaceChild()
insertBefore() //并没有insertAfter()
```

3、查找

```
getElementsByTagName() //通过标签名称
getElementsByName() //通过元素的Name属性的值(IE容错能力较强， 会得到一个数组，其中包括id等于name值的)
getElementById() //通过元素Id，唯一性
```

- dom事件

28，通用事件绑定/ 编写一个通用的事件监听函数?

```
function bindEvent(elem, type, selector, fn) {
    if (fn == null) {
        fn = selector; selector = null;
    }
    elem.addEventListner(type, function(e) {
        var target;
        if (selector) {
            target = e.target;
            if (target.matches(selector)) {
                fn.call(target, e);
            }
        } else {
            fn(e);
        }
    })
}// 使用代理
var div1 = document.getElementById('div1');
bindEvent(div1, 'click', 'a', function(e) {
    console.log(this.innerHTML);
});
```

```
// 不使用代理
var a = document.getElementById('a1');
bindEvent(div1, 'click', function(e) {
    console.log(a.innerHTML);
})
```

1、代理的好处

（1）代码简洁

（2）减少浏览器内存占用

2、事件冒泡

事件冒泡的应用：代理

29，bom常用属性有哪些?

1、navigator

```
var ua = navigator.userAgent;
ua.indexOf('chrome');
```

2、screen

```
screen.width
screen.height
```

3、location

```
location.href
location.protocol // http: https
location.host // learn/191
location.pathname
location.search
location.hash
```

4、history

```
history.back();
history.forward();
```

30，如何解决跨域问题

JSONP：

原理是：动态插入script标签，通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。

由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求。

优点是兼容性好，简单易用，支持浏览器与服务器双向通信。缺点是只支持GET请求。

JSONP：json+padding（内填充），顾名思义，就是把JSON填充到一个盒子里

```
functioncreateJs(sUrl){
    var oScript =document.createElement('script');
    oScript.type = 'text/javascript';
    oScript.src= sUrl;
    document.getElementsByTagName('head')[0].appendChild(oScript);
}
createJs('jsonp.js');
box({
    'name': 'test'
});
functionbox(json){
    alert(json.name);
}
```

CORS：

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

通过修改document.domain来跨子域

将子域和主域的document.domain设为同一个主域.前提条件：这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，否则无法利用document.domain进行跨域

主域相同的使用document.domain

使用window.name来进行跨域

window对象有个name属性，该属性有个特征：即在一个窗口(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

使用HTML5中新引进的window.postMessage方法来跨域传送数据

还有flash、在服务器上设置代理页面等跨域方式。个人认为window.name的方法既不复杂，也能兼容到几乎所有浏览器，这真是极好的一种跨域方法。

31，你觉得jQuery或zepto源码有哪些写的好的地方

(答案仅供参考)

Query源码封装在一个匿名函数的自执行环境中，有助于防止变量的全局污染，然后通过传入window对象参数，可以使window对象作为局部变量使用，好处是当jquery中访问window对象的时候，就不用将作用域链退回到顶层作用域了，从而可以更快的访问window对象。同样，传入undefined参数，可以缩短查找undefined时的作用域链。

```
  (function( window, undefined ) {
     //用一个函数域包起来，就是所谓的沙箱
     //在这里边var定义的变量，属于这个函数域内的局部变量，避免污染全局
     //把当前沙箱需要的外部变量通过函数参数引入进来
     //只要保证参数对内提供的接口的一致性，你还可以随意替换传进来的这个参数
    window.jQuery = window.$ = jQuery;
  })( window );
```

jquery将一些原型属性和方法封装在了jquery.prototype中，为了缩短名称，又赋值给了jquery.fn，这是很形象的写法。

有一些数组或对象的方法经常能使用到，jQuery将其保存为局部变量以提高访问速度。

jquery实现的链式调用可以节约代码，所返回的都是同一个对象，可以提高代码效率。

32，谈谈浮动和清除浮动

浮动的框可以向左或向右移动，直到他的外边缘碰到包含框或另一个浮动框的边框为止。由于浮动框不在文档的普通流中，所以文档的普通流的块框表现得就像浮动框不存在一样。浮动的块框会漂浮在文档普通流的块框上。

33，谈谈你在项目中用过哪些设计模式

工厂模式：

主要好处就是可以消除对象间的耦合，通过使用工程方法而不是new关键字。将所有实例化的代码集中在一个位置防止代码重复。

工厂模式解决了重复实例化的问题，但还有一个问题,那就是识别问题，因为根本无法搞清楚他们到底是哪个对象的实例。

```
function createObject(name,age,profession){//集中实例化的函数var obj = newObject();
  obj.name =name;
  obj.age = age;
  obj.profession= profession;
  obj.move =function () {
    returnthis.name + ' at ' + this.age + ' engaged in ' + this.profession;
  };
  return obj;
}
var test1 = createObject('trigkit4',22,'programmer');//第一个实例var test2
=createObject('mike',25,'engineer');//第二个实例
```

构造函数模式

使用构造函数的方法，即解决了重复实例化的问题，又解决了对象识别的问题，该模式与工厂模式的不同之处在于：

1）构造函数方法没有显示的创建对象 (new Object());

2）直接将属性和方法赋值给 this 对象;

3）没有 renturn 语句。

34，谈谈性能优化问题

代码层面：避免使用css表达式，避免使用高级选择器，通配选择器。

缓存利用：缓存Ajax，使用CDN，使用外部js和css文件以便缓存，添加Expires头，服务端配置Etag，减少DNS查找等

请求数量：合并样式和脚本，使用css图片精灵，初始首屏之外的图片资源按需加载，静态资源延迟加载。

请求带宽：压缩文件，开启GZIP，

代码层面的优化

用hash-table来优化查找

少用全局变量

用innerHTML代替DOM操作，减少DOM操作次数，优化javascript性能

用setTimeout来避免页面失去响应

缓存DOM节点查找的结果

避免使用CSS Expression

避免全局查询

避免使用with(with会创建自己的作用域，会增加作用域链长度)

多个变量声明合并

避免图片和iFrame等的空Src。空Src会重新加载当前页面，影响速度和效率

尽量避免写在HTML标签中写Style属性

移动端性能优化

尽量使用css3动画，开启硬件加速。

适当使用touch事件代替click事件。

避免使用css3渐变阴影效果。

可以用transform: translateZ(0)来开启硬件加速。

不滥用Float。Float在渲染时计算量比较大，尽量减少使用

不滥用Web字体。Web字体需要下载，解析，重绘当前页面，尽量减少使用。

合理使用requestAnimationFrame动画代替setTimeout

CSS中的属性（CSS3 transitions、CSS3 3D transforms、Opacity、Canvas、WebGL、Video）会触发GPU渲染，请合理使用。过渡使用会引发手机过耗电增加

PC端的在移动端同样适用

35，说说你对闭包的理解

使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念

闭包有三个特性：

1）函数嵌套函数

2）函数内部可以引用外部的参数和变量

3）参数和变量不会被垃圾回收机制回收

36，请你谈谈Cookie的弊端

cookie虽然在持久保存客户端数据提供了方便，分担了服务器存储的负担，但还是有很多局限性的。

第一：每个特定的域名下最多生成20个cookie

1）IE6或更低版本最多20个cookie

2）IE7和之后的版本最后可以有50个cookie。

3）Firefox最多50个cookie

4）chrome和Safari没有做硬性限制

IE和Opera 会清理近期最少使用的cookie，Firefox会随机清理cookie。

cookie的最大大约为4096字节，为了兼容性，一般不能超过4095字节。

IE 提供了一种存储可以持久化用户数据，叫做userdata，从IE5.0就开始支持。每个数据最多128K，每个域名下最多1M。这个持久化数据放在缓存中，如果缓存没有清理，那么会一直存在。

优点：极高的扩展性和可用性

1）通过良好的编程，控制保存在cookie中的session对象的大小。

2）通过加密和安全传输技术（SSL），减少cookie被破解的可能性。

3）只在cookie中存放不敏感数据，即使被盗也不会有重大损失。

4）控制cookie的生命期，使之不会永远有效。偷盗者很可能拿到一个过期的cookie。

缺点：

1）`Cookie`数量和长度的限制。每个domain最多只能有20条cookie，每个cookie长度不能超过4KB，否则会被截掉.

2）安全性问题。如果cookie被人拦截了，那人就可以取得所有的session信息。即使加密也与事无补，因为拦截者并不需要知道cookie的意义，他只要原样转发cookie就可以达到目的了。

3）有些状态不可能保存在客户端。例如，为了防止重复提交表单，我们需要在服务器端保存一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

37，浏览器本地存储

在较高版本的浏览器中，js提供了sessionStorage和globalStorage。在HTML5中提供了localStorage来取代globalStorage。

html5中的Web Storage包括了两种存储方式：sessionStorage和localStorage。

sessionStorage用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此sessionStorage不是一种持久化的本地存储，仅仅是会话级别的存储。

而localStorage用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

38，web storage和cookie的区别

Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的。Cookie的大小是受限的，并且每次你请求一个新的页面的时候Cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。

除此之外，Web Storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要前端开发者自己封装setCookie，getCookie。

但是cookie也是不可以或缺的：cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地"存储"数据而生

浏览器的支持除了IE 7 及以下不支持外，其他标准浏览器都完全支持(ie及FF需在web服务器里运行)，值得一提的是IE总是办好事，例如IE7、IE6中的userData其实就是javascript本地存储的解决方案。通过简单的代码封装可以统一到所有的浏览器都支持web storage。

localStorage和sessionStorage都具有相同的操作方法，例如setItem、getItem和removeItem等

39，cookie和session的区别：

1）cookie数据存放在客户的浏览器上，session数据放在服务器上。

2）cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗

考虑到安全应当使用session。

3）session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能

考虑到减轻服务器性能方面，应当使用COOKIE。

4）单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5）所以个人建议：

将登陆信息等重要信息存放为SESSION

其他信息如果需要保留，可以放在COOKIE中

40，display:none和visibility:hidden的区别?

display:none 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。

visibility:hidden 隐藏对应的元素，但是在文档布局中仍保留原来的空间。

41，CSS中link和@import的区别是?

(1) link属于HTML标签，而@import是CSS提供的;

(2) 页面被加载的时，link会同时被加载，而@import被引用的CSS会等到引用它的CSS文件被加载完再加载;

(3) import只在IE5以上才能识别，而link是HTML标签，无兼容问题;

(4) link方式的样式的权重 高于@import的权重.

42，position:absolute和float属性的异同

- 共同点：对内联元素设置float和absolute属性，可以让元素脱离文档流，并且可以设置其宽高。
- 不同点：float仍会占据位置，absolute会覆盖文档流中的其他元素。

43，介绍一下box-sizing属性?

- box-sizing属性主要用来控制元素的盒模型的解析模式。默认值是content-box。
- content-box：让元素维持W3C的标准盒模型。元素的宽度/高度由border + padding + content的宽度/高度决定，设置width/height属性指的是content部分的宽/高
- border-box：让元素维持IE传统盒模型（IE6以下版本和IE6~7的怪异模式）。设置width/height属性指的是border + padding + content

标准浏览器下，按照W3C规范对盒模型解析，一旦修改了元素的边框或内距，就会影响元素的盒子尺寸，就不得不重新计算元素的盒子尺寸，从而影响整个页面的布局。

44，选择符有哪些？哪些属性可以继承?

1）id选择器（ # myid）

2）类选择器（.myclassname）

3）标签选择器（div, h1, p）

4）相邻选择器（h1 + p）

5）子选择器（ul > li）

6）后代选择器（lia）

7）通配符选择器（ * ）

8）属性选择器（a[rel = "external"]）

9）伪类选择器（a: hover, li:nth-child）

45，优先级算法如何计算?

优先级为:

!important > id > class > tag

important 比 内联优先级高,但内联比 id 要高

46，CSS3新增伪类有那些?

CSS3新增伪类举例：

p:first-of-type选择属于其父元素的首个 `<p>` 元素的每个 `<p>` 元素。

p:last-of-type 选择属于其父元素的最后 `<p>` 元素的每个 `<p>` 元素。

p:only-of-type 选择属于其父元素唯一的 `<p>` 元素的每个 `<p>` 元素。

p:only-child 选择属于其父元素的唯一子元素的每个 `<p>` 元素。

p:nth-child(2) 选择属于其父元素的第二个子元素的每个 `<p>` 元素。

:enabled :disabled控制表单控件的禁用状态。

:checked 单选框或复选框被选中。

47，CSS3有哪些新特性?

CSS3实现圆角（border-radius），阴影（box-shadow），

对文字加特效（text-shadow、），线性渐变（gradient），旋转（transform）

transform:rotate(9deg) scale(0.85,0.90)translate(0px,-30px) skew(-9deg,0deg);//旋转,缩放,定位,倾斜

增加了更多的CSS选择器 多背景 rgba

在CSS3中唯一引入的伪元素是::selection.

媒体查询，多栏布局

border-image

CSS3中新增了一种盒模型计算方式：box-sizing。盒模型默认的值是content-box, 新增的值是padding-box和border-box，几种盒模型计算元素宽高的区别如下：

content-box（默认）

布局所占宽度Width：

Width = width + padding-left + padding-right + border-left + border-right

布局所占高度Height:

Height = height + padding-top + padding-bottom + border-top + border-bottom

padding-box

布局所占宽度Width：

Width = width(包含padding-left + padding-right) + border-top + border-bottom

布局所占高度Height:

Height = height(包含padding-top + padding-bottom) + border-top + border-bottom

border-box

布局所占宽度Width：

Width = width(包含padding-left + padding-right + border-left + border-right)

布局所占高度Height:

Height = height(包含padding-top + padding-bottom + border-top + border-bottom)

48，对BFC规范的理解?

BFC，块级格式化上下文，一个创建了新的BFC的盒子是独立布局的，盒子里面的子元素的样式不会影响到外面的元素。在同一个BFC中的两个毗邻的块级盒在垂直方向（和布局方向有关系）的margin会发生折叠。

（W3C CSS 2.1 规范中的一个概念，它决定了元素如何对其内容进行布局，以及与其他元素的关系和相互作用。

49，说说你对语义化的理解?

1）去掉或者丢失样式的时候能够让页面呈现出清晰的结构

2）有利于SEO：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：爬虫依赖于标签来确定上下文和各个关键字的权重；

3）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；

4）便于团队开发和维护，语义化更具可读性，是下一步吧网页的重要动向，遵循W3C标准的团队都遵循这个标准，可以减少差异化。

50，栈和队列的区别?

栈的插入和删除操作都是在一端进行的，而队列的操作却是在两端进行的。

队列先进先出，栈先进后出。

栈只允许在表尾一端进行插入和删除，而队列只允许在表尾一端进行插入，在表头一端进行删除

51，栈和堆的区别?

栈区（stack）： 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。

堆区（heap） ： 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收。

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

52，快速 排序的思想并实现一个快排?

“快速排序”的思想很简单，整个排序过程只需要三步：

（1）在数据集之中，找一个基准点

（2）建立两个数组，分别存储左边和右边的数组

（3）利用递归进行下次比较

```
<scripttype="text/javascript">
    functionquickSort(arr){
      if(arr.length<=1){
        return arr;//如果数组只有一个数，就直接返回；
      }
```

```
    var num = Math.floor(arr.length/2);//找到中间数的索引值，如果是浮点数，则向下取整
    var numValue =arr.splice(num,1);//找到中间数的值
    var left = [];
    var right = [];
    for(var i=0;i<arr.length;i++){
      if(arr[i]<numValue){
        left.push(arr[i]);//基准点的左边的数传到左边数组
      }
      else{
       right.push(arr[i]);//基准点的右边的数传到右边数组
      }
    }
    returnquickSort(left).concat([numValue],quickSort(right));//递归不断重复比较
  }

  alert(quickSort([32,45,37,16,2,87]));//弹出“2,16,32,37,45,87”
</script>
```

53，常见兼容性问题?

png24位的图片在iE6浏览器上出现背景，解决方案是做成PNG8.也可以引用一段脚本处理.

浏览器默认的margin和padding不同。解决方案是加一个全局的*{margin:0;padding:0;}来统一。

IE6双边距bug:块属性标签float后，又有横行的margin情况下，在ie6显示margin比设置的大。

浮动ie产生的双倍距离（IE6双边距问题：在IE6下，如果对元素设置了浮动，同时又设置了margin-left或margin-right，margin值会加倍。）

```
#box{ float:left; width:10px; margin:0 0 0 100px;}
```

这种情况之下IE会产生20px的距离，解决方案是在float的标签样式控制中加入

`_display:inline;`将其转化为行内属性。(_这个符号只有ie6会识别)

渐进识别的方式，从总体中逐渐排除局部。

首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。

接着，再次使用“+”将IE8和IE7、IE6分离开来，这样IE8已经独立识别。

```
css
  .bb{
   background-color:#f1ee18;/*所有识别*/
  .background-color:#00deff\9; /*IE6、7、8识别*/
  +background-color:#a200ff;/*IE6、7识别*/
  _background-color:#1e0bd1;/*IE6识别*/
  }
```

怪异模式问题：漏写DTD声明，Firefox仍然会按照标准模式来解析网页，但在IE中会触发

怪异模式。为避免怪异模式给我们带来不必要的麻烦，最好养成书写DTD声明的好习惯。现在

可以使用html5推荐的写法：`<doctype html>`

上下margin重合问题

ie和ff都存在，相邻的两个div的margin-left和margin-right不会重合，但是margin-top和margin-bottom却会发生重合。

解决方法，养成良好的代码编写习惯，同时采用margin-top或者同时采用margin-bottom。

54，解释下浮动和它的工作原理？清除浮动的技巧

浮动元素脱离文档流，不占据空间。浮动元素碰到包含它的边框或者浮动元素的边框停留。

1）使用空标签清除浮动。

这种方法是在所有浮动标签后面添加一个空标签 定义cssclear:both. 弊端就是增加了无意义标签。

2）使用overflow。

给包含浮动元素的父标签添加css属性 overflow:auto; zoom:1; zoom:1用于兼容IE6。

3）使用after伪对象清除浮动。

该方法只适用于非IE浏览器。具体写法可参照以下示例。使用中需注意以下几点。一、该方法中必须为需要清除浮动元素的伪对象中设置 height:0，否则该元素会比实际高出若干像素；

55，浮动元素引起的问题和解决办法?

浮动元素引起的问题：

（1）父元素的高度无法被撑开，影响与父元素同级的元素

（2）与浮动元素同级的非浮动元素（内联元素）会跟随其后

（3）若非第一个元素浮动，则该元素之前的元素也需要浮动，否则会影响页面显示的结构

解决方法：

使用CSS中的clear:both;属性来清除元素的浮动可解决2、3问题，对于问题1，添加如下样式，给父元素添加clearfix样式：

.clearfix:after{content:".";display: block;height:0;clear: both;visibility: hidden;}

.clearfix{display: inline-block;} /* for IE/Mac */

56，清除浮动的几种方法：

1）额外标签法，`<divstyle="clear:both;"></div>`（缺点：不过这个办法会增加额外的标签使HTML结构看起来不够简洁。）

2）使用after伪类

```
#parent:after{
    content:".";
    height:0;
    visibility:hidden;
    display:block;
    clear:both;
}
```

3）浮动外部元素

4）设置overflow为hidden或者auto

57，position的值有哪些， relative和absolute分别是相对于谁进行定位的?

- absolute :生成绝对定位的元素， 相对于最近一级的 定位不是 static 的父元素来进行定位。
- fixed （老IE不支持）生成绝对定位的元素，通常相对于浏览器窗口或 frame 进行定位。
- relative 生成相对定位的元素，相对于其在普通流中的位置进行定位。
- static 默认值。没有定位，元素出现在正常的流中
- sticky 生成粘性定位的元素，容器的位置根据正常文档流计算得出

58，html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？如何区分 HTML 和 HTML5?

HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

拖拽释放(Drag and drop) API

语义化更好的内容标签（header,nav,footer,aside,article,section）

音频、视频API(audio,video)

画布(Canvas) API

地理(Geolocation) API

本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；

sessionStorage 的数据在浏览器关闭后自动删除

表单控件，calendar、date、time、email、url、search

新的技术webworker, websocket,Geolocation

移除的元素

纯表现的元素：basefont，big，center，font, s，strike，tt，u；

对可用性产生负面影响的元素：frame，frameset，noframes；

支持HTML5新标签：

IE8/IE7/IE6支持通过document.createElement方法产生的标签，

可以利用这一特性让这些浏览器支持HTML5新标签，

当然最好的方式是直接使用成熟的框架、使用最多的是html5shim框架

```
<!--[if lt IE 9]>
    <script>src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>
<![endif]-->
```

如何区分： DOCTYPE声明\新增的结构元素\功能元素

59，如何实现浏览器内多个标签页之间的通信?

调用localstorge、cookies等本地存储方式

60，什么是 FOUC（无样式内容闪烁）？你如何来避免 FOUC?

FOUC - FlashOf Unstyled Content 文档样式闪烁

`<styletype="text/css"media="all">@import"../fouc.css";</style>`

而引用CSS文件的@import就是造成这个问题的罪魁祸首。IE会先加载整个HTML文档的DOM，然后再去导入外部的CSS文件，因此，在页面DOM加载完成到CSS导入完成中间会有一段时间页面上的内容是没有样式的，这段时间的长短跟网速，电脑速度都有关系。

解决方法简单的出奇，只要在`<head>`之间加入一个`<link>`或者`<script>`元素就可以了。

61，null和undefined的区别

null是一个表示"无"的对象，转为数值时为0；undefined是一个表示"无"的原始值，转为数值时为NaN。

当声明的变量还未被初始化时，变量的默认值为undefined。

null用来表示尚未存在的对象，常用来表示函数企图返回一个不存在的对象。

undefined表示"缺少值"，就是此处应该有一个值，但是还没有定义。典型用法是：

（1）变量被声明了，但没有赋值时，就等于undefined。

(2) 调用函数时，应该提供的参数没有提供，该参数等于undefined。

（3）对象没有赋值的属性，该属性的值为undefined。

（4）函数没有返回值时，默认返回undefined。

null表示"没有对象"，即该处不应该有值。典型用法是：

（1） 作为函数的参数，表示该函数的参数不是对象。

（2） 作为对象原型链的终点。

62，new操作符具体干了什么呢?

1）创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。

2）属性和方法被加入到 this 引用的对象中。

3）新创建的对象由 this 所引用，并且最后隐式的返回 this 。

```
var obj = {};
obj.__proto__ = Base.prototype;
Base.call(obj);
```

63，js延迟加载的方式有哪些?

defer和async、动态创建DOM方式（创建script，插入到DOM中，加载完毕后callBack）、按需异步载入js

64，谈一谈Javascript垃圾回收方法

标记清除（mark and sweep）

这是JavaScript最常见的垃圾回收方式，当变量进入执行环境的时候，比如函数中声明一个变量，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”。

垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

引用计数(reference counting)

在低版本IE中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加1，如果该变量的值变成了另外一个，则这个值得引用次数减1，当这个值的引用次数变为0的时 候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为0的值占用的空间。

在IE中虽然JavaScript对象通过标记清除的方式进行垃圾回收，但BOM与DOM对象却是通过引用计数回收垃圾的，
也就是说只要涉及BOM及DOM就会出现循环引用问题。

65，哪些操作会造成内存泄漏?

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。

垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收。

setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。

闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

66，列举IE与其他浏览器不一样的特性?

- IE支持currentStyle，FIrefox使用getComputStyle
- IE 使用innerText，Firefox使用textContent
- 滤镜方面：IE:filter:alpha(opacity= num)；Firefox：-moz-opacity:num
- 事件方面：IE：attachEvent：火狐是addEventListener
- 鼠标位置：IE是event.clientX；火狐是event.pageX
- IE使用event.srcElement；Firefox使用event.target
- IE中消除list的原点仅需margin:0即可达到最终效果；FIrefox需要设置margin:0;padding:0以及list-style:none
- CSS圆角：ie7以下不支持圆角

67，WEB应用从服务器主动推送Data到客户端有那些方式?

Javascript数据推送

- Commet：基于HTTP长连接的服务器推送技术
- 基于WebSocket的推送方案
- SSE（Server-Send Event）：服务器推送数据新方式

68，对前端界面工程师这个职位是怎么样理解的？它的前景会怎么样?

前端是最贴近用户的程序员，比后端、数据库、产品经理、运营、安全都近。

1）实现界面交互

2）提升用户体验

3）有了Node.js，前端可以实现服务端的一些事情

前端是最贴近用户的程序员，前端的能力就是能让产品从 90分进化到 100 分，甚至更好，

参与项目，快速高质量完成实现效果图，精确到1px；

与团队成员，UI设计，产品经理的沟通；

做好的页面结构，页面重构和用户体验；

处理hack，兼容、写出优美的代码格式；

针对服务器的优化、拥抱最新前端技术。

69，一个页面从输入 URL到页面加载显示完成，这个过程中都发生了什么?

分为4个步骤：

（1），当发送一个URL请求时，不管这个URL是Web页面的URL还是Web页面上每个资源的URL，浏览器都会开启一个线程来处理这个请求，同时在远程DNS服务器上启动一个DNS查询。这能使浏览器获得请求对应的IP地址。

（2），浏览器与远程`Web`服务器通过`TCP`三次握手协商来建立一个`TCP/IP`连接。该握手包括一个同步报文，一个同步-应答报文和一个应答报文，这三个报文在浏览器和服务器之间传递。该握手首先由客户端尝试建立起通信，而后服务器应答并接受客户端的请求，最后由客户端发出该请求已经被接受的报文。

（3），一旦`TCP/IP`连接建立，浏览器会通过该连接向远程服务器发送`HTTP`的`GET`请求。远程服务器找到资源并使用HTTP响应返回该资源，值为200的HTTP响应状态表示一个正确的响应。

（4），此时，`Web`服务器提供资源服务，客户端开始下载资源。

请求返回后，便进入了我们关注的前端模块

简单来说，浏览器会解析`HTML`生成`DOM Tree`，其次会根据CSS生成CSS Rule Tree，而`javascript`又可以根据`DOM API`操作`DOM`

70，javascript对象的几种创建方式

1）工厂模式

2）构造函数模式

3）原型模式

4）混合构造函数和原型模式

5）动态原型模式

6）寄生构造函数模式

7）稳妥构造函数模式

71，如何解决跨域问题

JSONP：

原理是：动态插入script标签，通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。

由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求。

优点是兼容性好，简单易用，支持浏览器与服务器双向通信。缺点是只支持GET请求。

JSONP：json+padding（内填充），顾名思义，就是把JSON填充到一个盒子里

```
functioncreateJs(sUrl){
  var oScript =document.createElement('script');
  oScript.type = 'text/javascript';
  oScript.src= sUrl;
  document.getElementsByTagName('head')[0].appendChild(oScript);
}
createJs('jsonp.js');
box({
  'name': 'test'
});
functionbox(json){
  alert(json.name);
}
```

CORS：

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

通过修改document.domain来跨子域

将子域和主域的document.domain设为同一个主域.前提条件：这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，否则无法利用document.domain进行跨域

主域相同的使用document.domain

使用window.name来进行跨域

window对象有个name属性，该属性有个特征：即在一个窗口(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

使用HTML5中新引进的window.postMessage方法来跨域传送数据

还有flash、在服务器上设置代理页面等跨域方式。个人认为window.name的方法既不复杂，也能兼容到几乎所有浏览器，这真是极好的一种跨域方法。

72，创建ajax的过程

(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.

```
var xmlHttp = new XMLHttpRequest();
 xmlHttp.open('GET','demo.php','true');
 xmlHttp.send()
 xmlHttp.onreadystatechange = function(){
   if(xmlHttp.readyState === 4 & xmlHttp.status=== 200){
   }
}
```

73，异步加载和延迟加载

1）异步加载的方案： 动态插入script标签

2）通过ajax去获取js代码，然后通过eval执行

3）script标签上添加defer或者async属性

4）创建并插入iframe，让它异步执行js

5）延迟加载：有些 js 代码并不是页面初始化的时候就立刻需要的，而稍后的某些情况才需要的。

74，ie各版本和chrome可以并行下载多少个资源

IE6 两个并发，iE7升级之后的6个并发，之后版本也是6个

Firefox，chrome也是6个

75，Flash、Ajax各自的优缺点，在使用中如何取舍?

- Flash适合处理多媒体、矢量图形、访问机器；对CSS、处理文本上不足，不容易被搜索。
- Ajax对CSS、文本支持很好，支持搜索；多媒体、矢量图形、机器访问不足。
- 共同点：与服务器的无刷新传递消息、用户离线和在线状态、操作DOM

76，请解释一下 JavaScript的同源策略。

概念:同源策略是客户端脚本（尤其是Javascript）的重要的安全度量标准。它最早出自Netscape Navigator2.0，其目的是防止某个文档或脚本从多个不同源装载。

这里的同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议。

指一段脚本只能读取来自同一来源的窗口和文档的属性。

77，为什么要有同源限制?

我们举例说明：比如一个黑客程序，他利用Iframe把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过Javascript读取到你的表单中input中的内容，这样用户名，密码就轻松到手了。

缺点：

现在网站的JS 都会进行压缩，一些文件用了严格模式，而另一些没有。这时这些本来是严格模式的文件，被 merge 后，这个串就到了文件的中间，不仅没有指示严格模式，反而在压缩后浪费了字节。

78，GET和POST的区别，何时使用POST?

GET：一般用于信息获取，使用URL传递参数，对所发送信息的数量也有限制，一般在2000个字符

POST：一般用于修改服务器上的资源，对所发送的信息没有限制。

GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值，

也就是说Get是通过地址栏来传值，而Post是通过提交表单来传值。

然而，在以下情况中，请使用 POST 请求：

无法使用缓存文件（更新服务器上的文件或数据库）

向服务器发送大量数据（POST 没有数据量限制）

发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

79，事件、IE与火狐的事件机制有什么区别？ 如何阻止冒泡?

1）我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为。

2）事件处理机制：IE是事件冒泡、firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件。；

3）`ev.stopPropagation()`;注意旧ie的方法`ev.cancelBubble = true`;

80，ajax的缺点和在IE下的问题?

ajax的缺点

1）ajax不支持浏览器back按钮。

2）安全问题 AJAX暴露了与服务器交互的细节。

3）对搜索引擎的支持比较弱。

4）破坏了程序的异常机制。

5）不容易调试。

IE缓存问题

在IE浏览器下，如果请求的方法是GET，并且请求的URL不变，那么这个请求的结果就会被缓存。解决这个问题的办法可以通过实时改变请求的URL，只要URL改变，就不会被缓存，可以通过在URL末尾添加上随机的时间戳参数('t'= + newDate().getTime())

或者：

```
open('GET','demo.php?rand=+Math.random()',true);//
```

Ajax请求的页面历史记录状态问题

可以通过锚点来记录状态，location.hash。让浏览器记录Ajax请求时页面状态的变化。

还可以通过HTML5的history.pushState，来实现浏览器地址栏的无刷新改变

81，谈谈你对重构的理解

网站重构：在不改变外部行为的前提下，简化结构、添加可读性，而在网站前端保持一致的行为。也就是说是在不改变UI的情况下，对网站进行优化，
在扩展的同时保持一致的UI。

对于传统的网站来说重构通常是：

表格(table)布局改为DIV+CSS

使网站前端兼容于现代浏览器(针对于不合规范的CSS、如对IE6有效的)

对于移动平台的优化

针对于SEO进行优化

深层次的网站重构应该考虑的方面

减少代码间的耦合

让代码保持弹性

严格按规范编写代码

设计可扩展的API

代替旧有的框架、语言(如VB)

增强用户体验

通常来说对于速度的优化也包含在重构中

压缩JS、CSS、image等前端资源(通常是由服务器来解决)

程序的性能优化(如数据读写)

采用CDN来加速资源加载

对于JS DOM的优化

HTTP服务器的文件缓存

82，HTTP状态码

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

200 OK 正常返回信息

201 Created 请求成功并且服务器创建了新的资源

202 Accepted 服务器已接受请求，但尚未处理

301 Moved Permanently 请求的网页已永久移动到新位置。

302 Found 临时性重定向。

303 SeeOther 临时性重定向，且总是使用 GET 请求新的 URI。

304 Not Modified 自从上次请求后，请求的网页未修改过。

400 BadRequest 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

401 Unauthorized 请求未授权。

403Forbidden 禁止访问。

404 NotFound 找不到如何与 URI 相匹配的资源。

500 InternalServer Error 最常见的服务器端错误。

503 ServiceUnavailable 服务器端暂时无法处理请求（可能是过载或维护）。

83，说说你对Promise的理解

依照 Promise/A+ 的定义，Promise 有四种状态：

pending: 初始状态, 非fulfilled 或 rejected.

fulfilled: 成功的操作.

rejected: 失败的操作.

settled: Promise已被fulfilled或rejected，且不是pending

另外， fulfilled 与 rejected 一起合称 settled。

Promise 对象用来进行延迟(deferred) 和异步(asynchronous) 计算。

Promise 的构造函数

构造一个 Promise，最基本的用法如下：

```
var promise = new Promise(function(resolve, reject) {
  if (...) { // succeed
    resolve(result);
  } else {  // fails
    reject(Error(errMessage));
  }
});
```

Promise 实例拥有 then 方法（具有 then 方法的对象，通常被称为 thenable）。它的使用方法如下：

promise.then(onFulfilled, onRejected)

接收两个函数作为参数，一个在 fulfilled 的时候被调用，一个在 rejected 的时候被调用，接收参数就是 future，onFulfilled对应 resolve, onRejected 对应 reject。

84，说说你对前端架构师的理解

负责前端团队的管理及与其他团队的协调工作，提升团队成员能力和整体效率；
带领团队完成研发工具及平台前端部分的设计、研发和维护；
带领团队进行前端领域前沿技术研究及新技术调研，保证团队的技术领先
负责前端开发规范制定、功能模块化设计、公共组件搭建等工作，并组织培训。

实现一个函数clone，可以对JavaScript中的5种主要的数据类型（包括Number、String、Object、Array、Boolean）进行值复制

```
Object.prototype.clone = function(){
    var o = this.constructor === Array ? [] : {};
    for(var e inthis){
        o[e] = typeofthis[e] === "object" ? this[e].clone() : this[e];
    }
    return o;
}
```

85，说说严格模式的限制

严格模式主要有以下限制：

变量必须声明后再使用

函数的参数不能有同名属性，否则报错

不能使用with语句

不能对只读属性赋值，否则报错

不能使用前缀0表示八进制数，否则报错

不能删除不可删除的属性，否则报错

不能删除变量delete prop，会报错，只能删除属性delete global[prop]

eval不会在它的外层作用域引入变量

eval和arguments不能被重新赋值

arguments不会自动反映函数参数的变化

不能使用arguments.callee

不能使用arguments.caller

禁止this指向全局对象

不能使用fn.caller和fn.arguments获取函数调用的堆栈

增加了保留字（比如protected、static和interface）

设立"严格模式"的目的，主要有以下几个：

- 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;
- 消除代码运行的一些不安全之处，保证代码运行的安全；
- 提高编译器效率，增加运行速度；
- 为未来新版本的Javascript做好铺垫。

注：经过测试均不支持严格模式。

86，如何删除一个cookie

1）将时间设为当前时间往前一点。

var date = newDate();

date.setDate(date.getDate() - 1);//真正的删除

setDate()方法用于设置一个月的某一天。

2）expires的设置

document.cookie= 'user='+ encodeURIComponent('name') + ';expires = ' + newDate(0)

`<strong>`，`<em>`和`<b>`，`<i>`标签

`<strong>`标签和`<em>`标签一样，用于强调文本，但它强调的程度更强一些。

em 是 斜体强调标签，更强烈强调，表示内容的强调点。相当于html元素中的`<i>...</i>`;

`< b >` `< i >`是视觉要素，分别表示无意义的加粗，无意义的斜体。

em 和 strong 是表达要素(phraseelements)。

87，说说你对AMD和Commonjs的理解

CommonJS是服务器端模块的规范，Node.js采用了这个规范。CommonJS规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。AMD规范则是非同步加载模块，允许指定回调函数。

AMD推荐的风格通过返回一个对象做为模块对象，CommonJS的风格通过对module.exports或exports的属性赋值来达到暴露模块对象的目的。

88，document.write()的用法

document.write()方法可以用在两个方面：页面载入过程中用实时脚本创建页面内容，以及用延时脚本创建本窗口或新窗口的内容。

document.write只能重绘整个页面。innerHTML可以重绘页面的一部分

编写一个方法求一个字符串的字节长度

假设：一个英文字符占用一个字节，一个中文字符占用两个字节

```
 functionGetBytes(str){
    var len = str.length;
    var bytes = len;
    for(var i=0; i<len; i++){
      if (str.charCodeAt(i) >255) bytes++;
    }
    return bytes;
  }
alert(GetBytes("你好,as"));
```

89，git fetch和git pull的区别

git pull：相当于是从远程获取最新版本并merge到本地

git fetch：相当于是从远程获取最新版本到本地，不会自动merge

90，说说你对MVC和MVVM的理解

MVC

View 传送指令到 Controller

Controller 完成业务逻辑后，要求 Model 改变状态

Model 将新的数据发送到 View，用户得到反馈

所有通信都是单向的。

Angular它采用双向绑定（data-binding）：View的变动，自动反映在 ViewModel，反之亦然。

组成部分Model、View、ViewModel

View：UI界面

ViewModel：它是View的抽象，负责View与Model之间信息转换，将View的Command传送到Model；

Model：数据访问层

91，请解释什么是事件代理

事件代理（Event Delegation），又称之为事件委托。是 JavaScript 中常用绑定事件的常用技巧。顾名思义，“事件代理”即是把原本需要绑定的事件委托给父元素，让父元素担当事件监听的职务。事件代理的原理是DOM元素的事件冒泡。使用事件代理的好处是可以提高性能。

92，attribute和property的区别是什么?

attribute是dom元素在文档中作为html标签拥有的属性；

property就是dom元素在js中作为对象拥有的属性。

所以：

对于html的标准属性来说，attribute和property是同步的，是会自动更新的，

但是对于自定义的属性来说，他们是不同步的，

93，说说网络分层里七层模型是哪七层

- 应用层：应用层、表示层、会话层（从上往下）（HTTP、FTP、SMTP、DNS）
- 传输层（TCP和UDP）
- 网络层（IP）
- 物理和数据链路层（以太网）

每一层的作用如下：

物理层：通过媒介传输比特,确定机械及电气规范（比特Bit）

数据链路层：将比特组装成帧和点到点的传递（帧Frame）

网络层：负责数据包从源到宿的传递和网际互连（包PackeT）

传输层：提供端到端的可靠报文传递和错误恢复（段Segment）

会话层：建立、管理和终止会话（会话协议数据单元SPDU）

表示层：对数据进行翻译、加密和压缩（表示协议数据单元PPDU）

应用层：允许访问OSI环境的手段（应用协议数据单元APDU）

各种协议

ICMP协议： 因特网控制报文协议。它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。
TFTP协议： 是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。
HTTP协议： 超文本传输协议，是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用

于分布式超媒体信息系统。
DHCP协议： 动态主机配置协议，是一种让系统得以连接到网络上，并获取所需要的配置参数手段。

94，说说mongoDB和MySQL的区别

MySQL是传统的关系型数据库，MongoDB则是非关系型数据库

mongodb以BSON结构（二进制）进行存储，对海量数据存储有着很明显的优势。

对比传统关系型数据库,NoSQL有着非常显著的性能和扩展性优势，与关系型数据库相比，MongoDB的优点有：
①弱一致性（最终一致），更能保证用户的访问速度：
②文档结构的存储方式，能够更便捷的获取数据。

95，讲讲304缓存的原理

服务器首先产生ETag，服务器可在稍后使用它来判断页面是否已经被修改。本质上，客户端通过将该记号传回服务器要求服务器验证其（客户端）缓存。

304是HTTP状态码，服务器用来标识这个文件没修改，不返回内容，浏览器在接收到个状态码后，会使用浏览器已缓存的文件

客户端请求一个页面（A）。 服务器返回页面A，并在给A加上一个ETag。 客户端展现该页面，并将页面连同ETag一起缓存。 客户再次请求页面A，并将上次请求时服务器返回的ETag一起传递给服务器。 服务器检查该ETag，并判断出该页面自上次客户端请求之后还未被修改，直接返回响应304（未修改——Not Modified）和一个空的响应体。

96，什么样的前端代码是好的

高复用低耦合，这样文件小，好维护，而且好扩展。