JavaScript100道面试题（https://github.com/minsion）

面试题 1. 简述JavaScript中map和foreach的区别?

推荐指数： ★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

JavaScript中map和foreach的共同点：
1.都是循环遍历数组中的每一项。
2.forEach()和map()里面每一次执行匿名函数都支持3个参数：数组中的当前项item,当前项的索引index,原始数组input。
3.匿名函数中的this都是指Window。
4.只能遍历数组。

JavaScript中map和foreach的不同点：
1.forEach()：没有返回值，即返回值为undefined
理论上这个方法是没有返回值的，仅仅是遍历数组中的每一项，不对原来数组进行修改；但是可以自己通过数组的索引来修改原来的数组，或当数组项为对象时修改对象中的值；
2.map()：有返回值，可以return 出来。
区别：map的回调函数中支持return返回值；return的是啥，相当于把数组中的这一项变为啥（并不影响原来的数组，只是相当于把原数组克隆一份，把克隆的这一份的数组中的对应项改变了）；
1.forEach()返回值是undefined，不可以链式调用。
2.map()返回一个新数组，原数组不会改变。

面试题 2. 解释下JavaScript中this是如何工作的?

推荐指数： ★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

实际上this 是在运行时进行绑定的，并不是在编写时绑定，它的上下文取决于函数调 用时的各种条件。this 的绑定和函数声明的位置没有任何关系，只取决于函数的调用方式。 总结: 函数被调用时发生 this 绑定，this 指向什么完全取决于函数在哪里被调用。 一、this 的绑定规则 this 一共有 4 中绑定规则，接下来一一介绍每种规则的解释和规则直接的优先级 默认绑定（严格/非严格模式） 隐式绑定 显式绑定 new 绑定 1.1 默认绑定（严格/非严格模式） 独立函数调用： 独立函数调用时 this 使用默认绑定规则，默认绑定规则下 this 指向 window（全局对象）。 严格模式下： this 无法使用默认绑定，this 会绑定到 undefined。

一、问题的由来
学懂 JavaScript 语言，一个标志就是理解下面两种写法，可能有不一样的结果。

```
var obj = {
foo: function () {}
};
var foo = obj.foo;
// 写法一
obj.foo()
// 写法二
foo()
```

上面代码中，虽然obj.foo和foo指向同一个函数，但是执行结果可能不一样。请看下面的例子。

```
var obj = {
```

```
foo: function () { console.log(this.bar) },
bar: 1
};
var foo = obj.foo;
var bar = 2;
obj.foo() // 1
foo() // 2
```

这种差异的原因，就在于函数体内部使用了this关键字。很多教科书会告诉你，this指的是函数运行时所在的环境。对于obj.foo()来说，foo运行在obj环境，所以this指向obj；对于foo()来说，foo运行在全局环境，所以this指向全局环境。所以，两者的运行结果不一样。

这种解释没错，但是教科书往往不告诉你，为什么会这样？也就是说，函数的运行环境到底是怎么决定的？举例来说，为什么obj.foo()就是在obj环境执行，而一旦var foo = obj.foo，foo()就变成在全局环境执行？

本文就来解释 JavaScript 这样处理的原理。理解了这一点，你就会彻底理解this的作用。

二、内存的数据结构

JavaScript 语言之所以有this的设计，跟内存里面的数据结构有关系。

```
var obj = { foo: 5 };
```

上面的代码将一个对象赋值给变量obj。JavaScript 引擎会先在内存里面，生成一个对象{ foo: 5 }，然后把这个对象的内存地址赋值给变量obj。

也就是说，变量obj是一个地址（reference）。后面如果要读取obj.foo，引擎先从obj拿到内存地址，然后再从该地址读出原始的对象，返回它的foo属性。

原始的对象以字典结构保存，每一个属性名都对应一个属性描述对象。举例来说，上面例子的foo属性，实际上是以下面的形式保存的。

```
{
foo: {
[[value]]: 5
[[writable]]: true
[[enumerable]]: true
[[configurable]]: true
}
}
```

注意，foo属性的值保存在属性描述对象的value属性里面。

三、函数

这样的结构是很清晰的，问题在于属性的值可能是一个函数。

```
var obj = { foo: function () {} };
```

这时，引擎会将函数单独保存在内存中，然后再将函数的地址赋值给foo属性的value属性。

```
{
foo: {
[[value]]: 函数的地址
...
}
}
```

由于函数是一个单独的值，所以它可以在不同的环境（上下文）执行。

```
var f = function () {};
var obj = { f: f };
// 单独执行
f()
// obj 环境执行
obj.f()
```

四、环境变量
JavaScript 允许在函数体内部，引用当前环境的其他变量。

```
var f = function () {
console.log(x);
};
```

上面代码中，函数体里面使用了变量x。该变量由运行环境提供。
现在问题就来了，由于函数可以在不同的运行环境执行，所以需要有一种机制，能够在函数体内部获得当前的运行环境（context）。所以，this就出现了，它的设计目的就是在函数体内部，指代函数当前的运行环境。

```
var f = function () {
console.log(this.x);
}
```

上面代码中，函数体里面的this.x就是指当前运行环境的x。

```
var f = function () {
console.log(this.x);
}
var x = 1;
var obj = {
f: f,
x: 2,
};
// 单独执行
f() // 1
// obj 环境执行
obj.f() // 2
```

上面代码中，函数f在全局环境执行，this.x指向全局环境的x。
在obj环境执行，this.x指向obj.x。
回到本文开头提出的问题，obj.foo()是通过obj找到foo，所以就是在obj环境执行。一旦var foo = obj.foo，变量foo就直接指向函数本身，所以foo()就变成在全局环境执行。

面试题 3. 简述异步线程，轮询机制，宏任务微任务?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

同步任务： 指的是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。
异步任务： 指的是不进入主线程，某个异步任务可以执行了，该任务才会进入主线程执行。

1. 同步和异步任务在不同的执行"场所"，同步的进入主线程，异步的进入Event Table执行并注册函数。
2. 当指定的异步事情完成时，Event Table会将这个函数移入Event Queue。
3. 主线程内的任务执行完毕为空，会去Event Queue读取对应的函数，推入主线程执行。
4. js引擎的monitoring process进程会持续不断的检查主线程执行栈是否为空，一旦为空，就会去Event
Queue那里检查是否有等待被调用的函数。上述过程会不断重复，也就是常说的Event Loop(事件循环也可以叫事件轮询)。

宏任务（macrotask ）和微任务（microtask ）
macrotask 和 microtask 表示异步任务的两种分类。
在挂起任务时，JS 引擎会将所有任务按照类别分到这两个队列中，首先在 macrotask 的队列（这个队列也被叫做 task queue）中取出第一个任务，执行完毕后取出 microtask 队列中的所有任务顺序

执行；之后再取 macrotask 任务，周而复始，直至两个队列的任务都取完。
JavaScript 执行机制：
主线程任务——>微任务——>宏任务
如果宏任务里还有微任就继续执行宏任务里的微任务，如果宏任务中的微任务中还有宏任务就在依次进行

主线程任务——>微任务——>宏任务——>宏任务里的微任务——>宏任务里的微任务中的宏任务——>知道任务全部完成

面试题 4. JavaScript阻止事件冒泡的方法?

推荐指数：★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

比如现在有一个子盒子和一个父盒子，子盒子和父盒子二者都有点击事件，但是此时，当我们点击子盒子时，只想让子盒子显示点击事件。这里我们就要用到阻止事件冒泡的方法来隔断父盒子的事件显示。
我们应该怎样阻断父盒子的点击事件呢?
可以直接在子盒子内部的点击事件里面添加stopPropagation()方法
如下所示：

```
son.addEventListener('click',function(e){
alert('son');
e.stopPropagation();
},false)
```

但是需要注意的是：这个方法也有兼容性问题，在低版本浏览器中（IE 6-8 ）通常是利用事件对象cancelBubble属性来操作的。即直接在相应的点击事件里面添加：

```
e.cancelBubble = true;
```

如果我们想要解决这种兼容性问题，就可以采用下述方法：

```
if(e && e.stopPropagation){
e.stopPropagation();
}else{
window.event.cancelBubble = true;
}
```

面试题 5. JavaScript阻止默认事件?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

1. event.preventDefault(); -- 阻止元素的默认事件。
注：a元素的点击跳转的默认事件 ，
button，radio等表单元素的默认事件 ，
div 元素没有默认事件
例：

```
<a href="http://www.baidu.com" target="_black">百度</a>


var samp = document.getElementByTagName("a");
samp.addEventListener("click",function(e){e.preventDefault()},false);
```

解释：点击链接的时候正常情况下会发生跳转，但是现在我们阻止了它的默认事件，即跳转事件，这时就不会跳转到百度了。

2. event.stopPropagation(); -- 阻止元素冒泡事件
注：嵌套元素一般都存在冒泡事件，会带来某些影响
例：

```
<div id="c1" onclick="alert(1)"><br />
    <div id="c2" onlick="alert(2)"><br />
        <input type="button" id="c3" value="点击" onclick="alert(3)"><br />
    </div><br />
</div>
```

这里点击button的时候，浏览器会先后弹出3,2,1，本来只想让绑定在button上的事件发生,却无意中触发了它的两个父级上的事件，这里我们只是做了一个简单测试，试想如果在项目开发中，某个按钮和他的父级同时绑定了很重要的事件，那么结果会惨不忍睹。这时的处理方法就是阻止冒泡事件。
给input注册click事件，同时阻止它的冒泡事件
document.getElementById('c3').addEventListener('click',function(e)
{e.stopPropagation()},false);

面试题 6. Javascript 怎样判断array 和 object ?

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

var obj = {"k1":"v1"};
var arr = [1,2];
1：通过isArray方法
使用方法： Array.isArray(obj); //obj是检测的对象

2：通过instanceof运算符来判断
instanceof运算符左边是子对象（待测对象），右边是父构造函数（这里是Array），
具体代码：
console.log("对象的结果："+(obj instanceof Array));
console.log("数组的结果："+(arr instanceof Array));

3:使用isPrototypeOf()函数
原理：检测一个对象是否是Array的原型（或处于原型链中，不但可检测直接父对象，还可检测整个原型链上的所有父对象）
使用方法: parent.isPrototypeOf(child)来检测parent是否为child的原型;isPrototypeOf()函数实现的功能和instancof运算符非常类似；
具体代码：
Array.prototype.isPrototypeOf(arr) //true表示是数组，false不是数组

4：利用构造函数constructor
具体代码：
console.log(obj.constructor == Array) //false
console.log(arr.constructor == Array) //true

5：使用typeof(对象)+类型名结合判断：
具体代码：

```
        function isArrayFour(arr) {
```

```
        if(typeof arr === "object") {
            if(arr.concat) {
                return 'This is Array'
            } else {
                return 'This not Array'
            }
        }
    }
    console.log(typeof(obj)) //"object"
    console.log(typeof(arr)) //"array"
    console.log(isArrayFour(obj)) //This not Array
    console.log(isArrayFour(arr)) //This is Array
```

面试题 7. 简述 Javascript 盒子模型?

推荐指数: ★★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

盒子模型定义 #
所有HTML元素可以看作盒子，在CSS中，"box model"这一术语是用来设计和布局时使用。
CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：外边距（margin）、边框（border）、内边距（padding）、实际内容（content）四个属性。

区别 #
标准盒模型:
盒子实际内容（content）的width/height=我们设置的width/height;
盒子总宽度/高度=width/height+padding+border+margin

IE盒子模型(怪异盒模型)
浏览器的 width 属性不是内容的宽度，而是内容、内边距和边框的宽度的总和；
盒子的（content）宽度+内边距padding+边框border宽度 = 我们设置的width(height也是如此)，
盒子总宽度/高度 = width/height + margin = 内容区宽度/高度 + padding + border + margin。
CSS3可以指定盒子模型种类 #
box-sizing: border-box; // 应用怪异盒模型 box-sizing: content-box;// 默认选选项，应用标准盒模型

面试题 8. Javascript 对象的key能是数字吗?

推荐指数: ★★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

可以是数字,object对应的key没有限制,只是如果是数字,取值的时候就不能用英文句号(.),只能用[]的方式取值。
var obj={1:3};// 这里 1 就是一个数组的 key
alert (obj [1]+2);// 取值的时候需要用 [], 而不能用 obj.1 的方式

面试题 9. Javascipt中async await 和promise和generator有什么区别

推荐指数: ★★★★ 试题难度: 高难 试题类型: 原理题

试题回答参考思路：

1：相同点：
这几种都可以解决异步操作的地狱回调问题
async await 和promise对象的区别： #
async和promise都是异步方法

2：区别：
async生成的结果是promise对象，async是promise的终结版。
await只能在async中使用，await是阻塞的意思，就是暂停，你一起调用2个接口，第一个执行完，不输出结果，要等最第二个接口执行完，才返回这两个的结果。是属于将异步操作转化为同步操作的做法
async await和generator： #
async是Generator的语法糖 ，也就是说，async是对generator的实现，而generator代码更复杂。
generator 除了能解决回调地域问题，还可以作为迭代器使用。generator 语法一般用于redux-saga
redux-saga 一般在umi或dva框架中(一个集成了redux、redux-saga、react-router-redux、react-router的框架)使用

面试题 10. JavaScript中手写promise ？

推荐指数： ★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

```
var promiseObj = new Promise(function (resolve, reject) {
let offer = false;
setTimeout(() => {
if (offer) {
resolve({
msg: '上班去',
company: 'xxxx公司'
})
} else {
reject({
msg: '继续面试'
})
}
},1000);
});
// 获取promiseObj保存的数据
promiseObj.then(function(res){
console.log('成功的信息:',res);
},function(err){
console.log('失败的信息:',err);
})
```

以下是一个简单的promise实现，想更深一步研究，请看末尾的链接 #

面试题 11. JavaScript中promise.all作用?

推荐指数： ★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

Promise.all(iterable) 方法返回一个 Promise 实例，此实例在 iterable 参数内所有的 promise 都“完成（resolved）”或 参数中不包含 promise 时回调完成（resolve）；如果参数中 promise 有一个失败（rejected），此实例回调失败（reject）， 失败的原因是第一个失败 promise 的结果。
它通常在启动多个异步任务并发运行并为其结果创建承诺之后使用，以便人们可以等待所有任务完成。
通俗的说法：解决promise的同步调用问题，可以将多个异步操作转为同步操作，缺点：只要有一个promise返回错误，那么后面的promise都不会再执行

面试题 12. 解释什么是工厂模式，有什么优缺点?

推荐指数： ★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

定义 #
工厂模式是一种设计模式，目的是为了创建对象，它通常在类或类的静态方法中实现，
有以下目的：当创建相似对象是执行重复操作
当编译时，不知道具体类型的情况下，为工厂客户提供一个创建对象的接口
优缺点 #
【优点】
写好了工厂方法后，只要原料足够，我们可以无限次的创建新的对象
【缺点】
没有使用 new ，而是直接调用方法进行创建
每个对象都有自己的函数(像下面的例子中，每创建一个对象，就会创建两个函数，若创建10个对象,就需要创建20个函数，而这些函数都一模一样的，完全没必要)，浪费资源

```
//使用工厂方式创建一个对象
function createCat(name, sex) {
//1.原料
var obj = new Object();
//2.加工 -- 对属性进行赋值
obj.name = name;
obj.sex = sex;
//3.方法
obj.showName = function () {
console.log("小猫名称：" + this.name);
};
obj.showSex = function () {
console.log("小猫性别：" + this.sex);
};
//4.加工结束,--输出返回
return obj;
}
//工厂模式的使用
var cat1 = createCat("小花", "母的");
//调用工厂里的方法
cat1.showName();
cat1.showSex();
var cat2 = createCat("罗小黑", "公的");
//调用工厂里的方法
```

```
cat2.showName();
cat2.showSex();
```

面试题 13. Javascript 图片/文件夹上传到后台是什么类型?

推荐指数: ★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

图片上传后台有三种格式:
file格式（创建formData来完成file上传）
base64格式
Blob流格式

面试题 14. Javascript 浅拷贝/深度拷贝的区别?

推荐指数: ★★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

深拷贝和浅拷贝是指在赋值一个对象时，拷贝的深度不同。

在进行深拷贝时，会拷贝所有的属性，并且如果这些属性是对象，也会对这些对象进行深拷贝，直到最底层的基本数据类型为止。这意味着，对于深拷贝后的对象，即使原对象的属性值发生了变化，深拷贝后的对象的属性值也不会受到影响。
相反，浅拷贝只会拷贝对象的第一层属性，如果这些属性是对象，则不会对这些对象进行拷贝，而是直接复制对象的引用。这意味着，对于浅拷贝后的对象，如果原对象的属性值发生了变化，浅拷贝后的对象的属性值也会跟着发生变化。
举个例子，假设我们有一个类 A，其中包含两个属性 x 和 y，x 是一个数字，y 是一个列表。我们将 A 的一个实例 a 赋值给另一个实例 b，如果我们使用深拷贝，那么 b 将会是一个完全独立的对象，即使我们对 a 的 y 属性进行修改，b 的 y 属性也不会受

面试题 15. Javascript 闭包是什么,闭包形成的原因和闭包的用途 ?

推荐指数: ★★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

闭包是指在一个函数内部定义的函数，并且该函数可以访问外部函数的变量。这个定义在函数内部的函数就被称为闭包。

```
function F1(){
var a = 29;
return function(){
console.log(a);
}
}
var a = 2;
var f1 = F1();
f1();
```

造成闭包的原因?
通常，函数的作用域及其所有变量都会在函数执行结束后被销毁。但是，在创建了一个闭包以后，这个函数的作用域就会一直保存到闭包不存在为止。

在javascript中，如果一个对象不再被引用，那么这个对象就会被垃圾回收机制回收；
如果两个对象互相引用，而不再被第3者所引用，那么这两个互相引用的对象也会被回收。
闭包只能取得包含函数中任何变量的最后一个值
闭包的使用场景?
最大用处有两个，一个是可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中。
1、可以读出函数内部的变量，正常情况下，外部函数是无法读取到内部函数的变量的，所以可以说：闭包就是将函数内部和函数外部连接起来的一座桥梁，即函数作为参数传递。
2、由于对象内的变量一直被引用，所以这对象不会被垃圾回收机制回收。可以始终保持在内存中。
使用闭包的注意点
1、由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在IE中可能导致内存泄露。解决方法是，在退出函数之前，将不使用的局部变量全部删除。
2、闭包会在父函数外部，改变父函数内部变量的值。所以，如果你把父函数当作对象（object）使用，把闭包当作它的公用方法（Public Method），把内部变量当作它的私有属性（private value），这时一定要小心，不要随便改变父函数内部变量的值。

面试题 16. Javascript 跨域的解决方案有哪些?

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

定义 #
知其然知其所以然，在说跨域方法之前，我们先了解下什么叫跨域，浏览器有同源策略，只有当“协议”、“域名”、“端口号”都相同时，才能称之为是同源，其中有一个不同，即是跨域。
那么同源策略的作用是什么呢？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。
那么我们又为什么需要跨域呢？一是前端和服务器分开部署，接口请求需要跨域，二是我们可能会加载其它网站的页面作为 iframe 内嵌。
跨域的几种方式 #
jsonp
jsonp缺点：只能是get方式不能是post方式
jsonp原理：script标签的src属性可以跨域
proxy代理，下面这些这些都是属性使用proxy代理进行跨域的例子
vue(实际是webpack)的devServer的proxy设置(只能是打包前使用，打包后需要后台配置跨域)
nginx反向代理
node使用中间件设置跨域
后台头部设置 Access-Control-Allow-Origin ，cors
其他方式： a. document.domain + iframe（只有在主域相同的时候才能使用该方法）b. location.hash + iframe c. window.name + iframe d. postMessage （HTML5 中的 XMLHttpRequest Level 2中的API） e. web socket web sockets是一种浏览器的API，它的目标是在一个单独的持久连接上提供全双工、双向通信

面试题 17. Http协议详解 Http请求方式有 Http响应状态码 ?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

http请求由三部分组成，分别是：请求行、消息报头、请求正文 请求方法（所有方法全为大写）有多种，各个方法的解释如下： GET 请求获取Request-URI所标识的资源 POST 在Request-URI所标识的资源后附加新的数据 HEAD 请求获取由Request-URI所标识的资源的响应消息报头 PUT 请求服务

器存储一个资源，并用Request-URI作为其标识 DELETE 请求服务器删除Request-URI所标识的资源 TRACE 请求服务器回送收到的请求信息，主要用于测试或诊断 CONNECT 保留将来使用 OPTIONS 请求查询服务器的性能，或者查询与资源相关的选项和需求

在接收和解释请求消息后，服务器返回一个HTTP响应消息。 #
HTTP响应也是由三个部分组成，分别是：状态行、消息报头、响应正文 3xx：重定向--要完成请求必须进行更进一步的操作 4xx：客户端错误--请求有语法错误或请求无法实现 5xx：服务器端错误--服务器未能实现合法的请求

301 永久移动，请求的资源被永久的移动到新url，返回信息会包括新的url。浏览器会自动定向到新url 302 临时移动，资源只是临时被移动，客户端赢继续使用原有url 304 未修改，所请求的资源未修改，服务器返回此状态码是=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

400 Bad Request //客户端请求有语法错误，不能被服务器所理解，解决方法：修改请求的参数及语法 401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用， 解决方式：即没有启用任何认证方式，只需要在IIS Manager里面启用需要的认证方式即可。 即被Authorization Rule阻挡，则需要分析这个authorization rule是否是否必须来决定对他的更改。
403 Forbidden //服务器收到请求，但是拒绝提供服务
1. 你的IP被列入黑名单。
2、你在一定时间内过多地访问此网站(一般是用采集程序)，被防火墙拒绝访问了。
3、网站域名解析到了空间，但空间未绑定此域名。
4、你的网页脚本文件在当前目录下没有执行权限。
6、以http方式访问需要ssl连接的网址。
7、浏览器不支持SSL 128时访问SSL 128的连接。
8、在身份验证的过程中输入了错误的密码。
404 Not Found //请求资源不存在，eg：输入了错误的URL 解决办法：输入正确的url地址 405 请求方式不对 ,比如原本需要post方式请求的，你写了get方式请求

面试题 18. 简述页面从发送http请求到渲染页面的全过程?

推荐指数： ★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

域名解析 --> 2.发起TCP的3次握手 --> 3.建立TCP连接后发起http请求 --> 4.服务器响应http请求，浏览器得到html代码 --> 5.浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 6.浏览器对页面进行渲染呈现给用户

面试题 19. JavaScript什么是长连接 ?

推荐指数： ★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

在HTTP/1.0中默认使用短连接。也就是说，客户端和服务器每进行一次HTTP操作，就建立一次连接，任务结束就中断连接。 当客户端浏览器访问的某个HTML或其他类型的Web页中包含有其他的Web资源（如JavaScript文件、图像文件、CSS文件等）， 每遇到这样一个Web资源，浏览器就会重新建立一个HTTP会话。 而从HTTP/1.1起，默认使用长连接，用以保持连接特性。使用长连接的HTTP协议，会在响应头加入这行代码： Connection:keep-alive 在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，客户端再次访问这个服务器时 ，会继续使用这一条已经建立的连接。Keep-Alive不会永久保持连接，它有一个保持时间，可

以在不同的服务器软件（如Apache）中设定这个 时间。实现长连接需要客户端和服务端都支持长连接。

面试题 20. Window. write和document.innerhtml区别?

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

主要区别：document.write是直接将内容写入页面的内容流，会导致页面全部重绘，innerHTML将内容写入某个DOM节点，不会导致页面全部重绘

面试题 21. display:none和visibility:hidden的区别是 ?

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

display:none是彻底消失，不在文档流中占位，浏览器也不会解析该元素；visibility:hidden是视觉上消失了，可以理解为透明度为0的效果，在文档流中占位，浏览器会解析该元素；
使用visibility:hidden比display:none性能上要好，display:none切换显示时visibility，页面产生回流（当页面中的一部分元素需要改变规模尺寸、布局、显示隐藏等，页面重新构建，此时就是回流。所有页面第一次加载时需要产生一次回流），而visibility切换是否显示时则不会引起回流。 所以我使用visibility:hidden，在页面渲染时第二个tab页中的轮播图就可以获取宽度做自适应了。

面试题 22. 简述Doctype的作用 ?

推荐指数： ★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

定义： #
DOCTYPE的作用 DOCTYPE是document type（文档类型）的缩写
作用: #
document.compatMode(文档的解析类型)： BackCompat：怪异模式，浏览器使用自己的怪异模式解析渲染页面。 CSS1Compat：标准模式，浏览器使用W3C的标准解析渲染页面。
Doctype声明了文档的解析类型(document.compatMode)，避免浏览器的怪异模式。这个属性会被浏览器识别并使用，但是如果你的页面没有DOCTYPE的声明，那么compatMode默认就是BackCompat，浏览器按照自己的方式解析渲染页面，那么，在不同的浏览器就会显示不同的样式。如果你的页面添加了那么，那么就等同于开启了标准模式那么浏览器就得老老实实的按照W3C的标准解析渲染页面，这样一来，你的页面在所有的浏览器里显示的就都是一个样子了。这就是Doctype的作用。

面试题 23. JavaScript中常用的数组方法?

推荐指数： ★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

join()： #
join(separator): 将数组的元素组起一个字符串，以separator为分隔符，省略的话则用默认用逗号为分隔符，该方法只接收一个参数：即分隔符。

push()和pop()：push(): #
可以接收任意数量的参数，把它们逐个添加到数组末尾，并返回修改后数组的长度。 pop()：数组末尾移除最后一项，减少数组的 length 值，然后返回移除的项。
shift() 和 unshift()：shift()： #
删除原数组第一项，并返回删除元素的值；如果数组为空则返回undefined 。 unshift:将参数添加到原数组开头，并返回数组的长度
sort()： #
按升序排列数组项——即最小的值位于最前面，最大的值排在最后面
reverse()： #
反转数组项的顺序。
concat()： #
将参数添加到原数组中。这个方法会先创建当前数组一个副本，然后将接收到的参数添加到这个副本的末尾，最后返回新构建的数组。在没有给 concat()方法传递参数的情况下，它只是复制当前数组并返回副本。
slice()： #
返回从原数组中指定开始下标到结束下标之间的项组成的新数组。slice()方法可以接受一或两个参数，即要返回项的起始和结束位置。在只有一个参数的情况下， slice()方法返回从该参数指定位置开始到当前数组末尾的所有项。如果有两个参数，该方法返回起始和结束位置之间的项——但不包括结束位置的项。
splice()：splice()： #
很强大的数组方法，它有很多种用法，可以实现删除、插入和替换。 删除：可以删除任意数量的项，只需指定 2 个参数：要删除的第一项的位置和要删除的项数。例如， splice(0,2)会删除数组中的前两项。 插入：可以向指定位置插入任意数量的项，只需提供 3 个参数：起始位置、 0（要删除的项数）和要插入的项。例如，splice(2,0,4,6)会从当前数组的位置 2 开始插入4和6。 替换：可以向指定位置插入任意数量的项，且同时删除任意数量的项，只需指定 3 个参数：起始位置、要删除的项数和要插入的任意数量的项。插入的项数不必与删除的项数相等。例如，splice (2,1,4,6)会删除当前数组位置 2 的项，然后再从位置 2 开始插入4和6。 splice()方法始终都会返回一个数组，该数组中包含从原始数组中删除的项，如果没有删除任何项，则返回一个空数组。
indexOf()和 lastIndexOf() （ES5新增）： #
indexOf()：接收两个参数：要查找的项和（可选的）表示查找起点位置的索引。其中， 从数组的开头（位置 0）开始向后查找。 lastIndexOf：接收两个参数：要查找的项和（可选的）表示查找起点位置的索引。其中， 从数组的末尾开始向前查找。 这两个方法都返回要查找的项在数组中的位置，或者在没找到的情况下返回 1。在比较第一个参数与数组中的每一项时，会使用全等操作符。
forEach() （ES5新增）forEach()： #
对数组进行遍历循环，对数组中的每一项运行给定函数。这个方法没有返回值。参数都是function类型，默认有传参，参数分别为：遍历的数组内容；第对应的数组索引，数组本身。
map() （ES5新增）map()： #
指“映射”，对数组中的每一项运行给定函数，返回每次函数调用的结果组成的数组。
filter() （ES5新增）filter()： #
“过滤”功能，数组中的每一项运行给定函数，返回满足过滤条件组成的数组。
every() （ES5新增）every()： #
判断数组中每一项都是否满足条件，只有所有项都满足条件，才会返回true。
some() （ES5新增）some()： #
判断数组中是否存在满足条件的项，只要有一项满足条件，就会返回true。
reduce()和 reduceRight() （ES5新增） #
这两个方法都会实现迭代数组的所有项，然后构建一个最终返回的值。reduce()方法从数组的第一项开始，逐个遍历到最后。而 reduceRight()则从数组的最后一项开始，向前遍历到第一项。 这两个方法都接收两个参数：一个在每一项上调用的函数和（可选的）作为归并基础的初始值。 传给 reduce()和reduceRight()的函数接收 4 个参数：前一个值、当前值、项的索引和数组对象。这个函数返回的任何值都会作为第一个参数自动传给下一项。第一次迭代发生在数组的第二项上，因此第一个参数是数组的第一项，第二个参数就是数组的第二项。

面试题 24. JavaScript字符串方法？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

charAt() 返回指定索引位置的字符
charCodeAt() 返回指定索引位置字符的 Unicode 值
concat() 连接两个或多个字符串，返回连接后的字符串
fromCharCode() 将 Unicode 转换为字符串
indexOf() 返回字符串中检索指定字符第一次出现的位置
lastIndexOf() 返回字符串中检索指定字符最后一次出现的位置
localeCompare() 用本地特定的顺序来比较两个字符串
match() 找到一个或多个正则表达式的匹配
replace() 替换与正则表达式匹配的子串
search() 检索与正则表达式相匹配的值
slice() 提取字符串的片断，并在新的字符串中返回被提取的部分
split() 把字符串分割为子字符串数组
substr() 从起始索引号提取字符串中指定数目的字符
substring() 提取字符串中两个指定的索引号之间的字符
toLocaleLowerCase() 根据主机的语言环境把字符串转换为小写，只有几种语言（如土耳其语）具有地方特有的大小写映射
toLocaleUpperCase() 根据主机的语言环境把字符串转换为大写，只有几种语言（如土耳其语）具有地方特有的大小写映射
toLowerCase() 把字符串转换为小写
toString() 返回字符串对象值
toUpperCase() 把字符串转换为大写
trim() 移除字符串首尾空白
valueOf() 返回某个字符串对象的原始值

面试题 25. 手写防抖、节流，防抖和节流的区别？

推荐指数： ★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

在前端开发的过程中，我们经常会需要绑定一些持续触发的事件，如滚动、输入等等，但有些时候我们并不希望在事件持续触发的过程中那么频繁地去执行函数,防抖和节流是比较好的解决方案。
(1)所谓防抖，就是指触发事件后在 n 秒内函数只能执行一次，如果在 n 秒内又触发了事件，则会重新计算函数执行时间。使用场景:注册的时候检查用户名是否已经被注册.
(2)所谓节流，就是指连续触发事件但是在 n 秒中只执行一次函数。节流会稀释函数的执行频率。使用场景:监听滚动条是否到了顶部或底部.

防抖和节流的区别 #
函数防抖和节流区别在于，当事件持续被触发，如果触发时间间隔短于规定的等待时间（n秒），那么函数防抖的情况下，函数将一直推迟执行，造成不会被执行的效果；函数节流的情况下，函数将每个 n 秒执行一次。

防抖——触发高频事件后 n 秒后函数只会执行一次，如果 n 秒内高频事件再
次被触发，则重新计算时间；
function debounce(fn) {

```
let timeout = null
// 创建一个标记用来存放定时器的返回值
return function() {
clearTimeout(timeout)
// 每当用户输入的时候把前一个 setTimeout clear 掉
timeout = setTimeout(() => {
// 然后又创建一个新的 setTimeout，这样就能保证输入字符后的
interval 间隔内如果还有字符输入的话，就不会执行 fn 函数
fn.apply(this, arguments)
}, 500)
}}function sayHi() {
console.log('防抖成功')}var inp =
document.getElementById('inp')inp.addEventListener('input',
debounce(sayHi)) // 防抖
节流——高频事件触发，但在 n 秒内只会执行一次，所以节流会稀释函数的执
行频率。
function throttle(fn) {
let canRun = true // 通过闭包保存一个标记
return function() {
if (!canRun) return
// 在函数开头判断标记是否为 true，不为 true 则 return
canRun = false // 立即设置为 false
setTimeout(() => {
// 将外部传入的函数的执行放在 setTimeout 中
fn.apply(this, arguments)
// 最后在 setTimeout 执行完毕后再把标记设置为 true(关键) 表
示可以执行下一次循环了。当定时器没有执行的时候标记永远是 false，在开头
被 return 掉
canRun = true
}, 500)
}}function sayHi(e) {
console.log(e.target.innerWidth,
e.target.innerHeight)}window.addEventListener('resize',
throttle(sayHi))
```

面试题 26. Javascript的typeof返回哪些数据类型?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

在javascript中，typeof操作符可返回的数据类型有：“undefined”、“object”、“boolean”、“number”、“string”、“symbol”、“function”等。

面试题 27. Javascript例举3种强制类型转换和2种隐式类型转换?

推荐指数：★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

强制：String()，Boolean() Number() parseInt() parseFloat()
隐式 + - ==

面试题 28. Javascript数组方法pop() push() unshift() shift() 简单描述？

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

Push()尾部添加
pop()尾部删除
Unshift()头部添加
shift()头部删除

面试题 29. Javascipt的call和apply的区别？

推荐指数：★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

一、定义
apply：应用某一对象的一个方法，用另一个对象替换当前对象。

call：调用一个对象的一个方法，以另一个对象替换当前对象。

二、共同点：
都“可以用来代替另一个对象调用一个方法，将一个函数的对象上下文从初始的上下文改变为由 thisObj 指定的新对象。”

三、不同点：
apply：最多只能有两个参数——新this对象和一个数组 argArray。如果给该方法传递多个参数，则把参数都写进这个数组里面，当然，即使只有一个参数，也要写进数组里面。如果 argArray 不是一个有效的数组或者不是 arguments 对象，那么将导致一个 TypeError。如果没有提供 argArray 和 thisObj 任何一个参数，那么 Global 对象将被用作 thisObj，并且无法被传递任何参数。

call：则是直接的参数列表，主要用在js对象各方法互相调用的时候，使当前this实例指针保持一致,或在特殊情况下需要改变this指针。如果没有提供 thisObj 参数，那么 Global 对象被用作 thisObj。

简单地说，apply和call功能一样，只是传入的参数列表形式不同。

如 func.call(func1,var1,var2,var3) 对应的apply写法为：func.apply(func1,[var1,var2,var3])

也就是说：call调用的为单个，apply调用的参数为数组

```
function sum(a,b){
console.log(this === window);//true
console.log(a + b);
}
sum(1,2);
sum.call(null,1,2);
sum.apply(null,[1,2]);
```

面试题 30. JavaScript 闭包是什么，有什么特性，对页面有什么影响?简要介绍你理解的闭包?

推荐指数： ★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

闭包就是能够读取其他函数内部变量的函数

一、变量的作用域
要理解闭包，首先必须理解Javascript特殊的变量作用域。
变量的作用域无非就是两种：全局变量和局部变量。
Javascript语言的特殊之处，就在于函数内部可以直接读取全局变量。

Js代码

```
var n=999;
function f1(){
    alert(n);
}
f1(); // 999
```

另一方面，在函数外部自然无法读取函数内的局部变量。
Js代码

```
function f1(){
    var n=999;
}
alert(n); // error
```

这里有一个地方需要注意，函数内部声明变量的时候，一定要使用var命令。如果不用的话，你实际上声明了一个全局变量！
Js代码

```
function f1(){
    n=999;
}
f1();
alert(n); // 999
```

二、如何从外部读取局部变量?
出于种种原因，我们有时候需要得到函数内的局部变量。但是，前面已经说过了，正常情况下，这是办不到的，只有通过变通方法才能实现。
那就是在函数的内部，再定义一个函数。
Js代码

```
function f1(){
    n=999;
    function f2(){
        alert(n); // 999
    }
}
```

在上面的代码中，函数f2就被包括在函数f1内部，这时f1内部的所有局部变量，对f2都是可见的。但是反过来就不行，f2内部的局部变量，对f1 就是不可见的。这就是Javascript语言特有的“链式作用域”结构（chain scope），
子对象会一级一级地向上寻找所有父对象的变量。所以，父对象的所有变量，对子对象都是可见的，反之则不成立。
既然f2可以读取f1中的局部变量，那么只要把f2作为返回值，我们不就可以在f1外部读取它的内部变量

了吗！

Js代码

```
function f1(){
    n=999;
    function f2(){
        alert(n);
    }
    return f2;
}
var result=f1();
result(); // 999
```

三、闭包的概念

上一节代码中的f2函数，就是闭包。

各种专业文献上的“闭包”（closure）定义非常抽象，很难看懂。我的理解是，闭包就是能够读取其他函数内部变量的函数。

由于在Javascript语言中，只有函数内部的子函数才能读取局部变量，因此可以把闭包简单理解成“定义在一个函数内部的函数”。

所以，在本质上，闭包就是将函数内部和函数外部连接起来的一座桥梁。

---b

四、闭包的用途

闭包可以用在许多地方。它的最大用处有两个，一个是前面提到的可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中。

怎么来理解这句话呢？请看下面的代码。

Js代码

```
function f1(){
    var n=999;
    nAdd=function(){n+=1}
    function f2(){
        alert(n);
    }
    return f2;
}
var result=f1();
result(); // 999
nAdd();
result(); // 1000
```

在这段代码中，result实际上就是闭包f2函数。它一共运行了两次，第一次的值是999，第二次的值是1000。这证明了，函数f1中的局部变量n一直保存在内存中，并没有在f1调用后被自动清除。

为什么会这样呢？原因就在于f1是f2的父函数，而f2被赋给了一个全局变量，这导致f2始终在内存中，而f2的存在依赖于f1，因此f1也始终在内存中，不会在调用结束后，被垃圾回收机制（garbage collection）回收。

这段代码中另一个值得注意的地方，就是“nAdd=function(){n+=1}”这一行，首先在nAdd前面没有使用var关键字，因此 nAdd是一个全局变量，而不是局部变量。其次，nAdd的值是一个匿名函数（anonymous function），而这个

匿名函数本身也是一个闭包，所以nAdd相当于是一个setter，可以在函数外部对函数内部的局部变量进行操作。

五、使用闭包的注意点

1）由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在IE中可能导致内存泄露。解决方法是，在退出函数之前，将不使用的局部变量全部删除。

2）闭包会在父函数外部，改变父函数内部变量的值。所以，如果你把父函数当作对象（object）使用，把闭包当作它的公用方法（Public Method），把内部变量当作它的私有属性（private value），这时一定要小心，不要随便

改变父函数内部变量的值。

面试题 31. JavaScript 添加 删除 替换 插入到某个接点的方法?

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

obj.appendChidl()

obj.innersetBefore

obj.replaceChild

obj.removeChild

面试题 32. 阐述Javascript的同源策略?

推荐指数：★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

JavaScript的同源策略是什么?

如果两个页面拥有相同的协议（protocol）,端口（如果指定），和主机，那么这两个页面就是属于同一个源

览器有一个很重要的概念——同源策略(Same-OriginPolicy)。所谓同源是指，域名，协议，端口相同。不同源的客户端脚本(javascript、ActionScript)在没明确授权的情况下，不能读写对方的资源。

简单的来说，浏览器允许包含在页面A的脚本访问第二个页面B的数据资源，这一切是建立在A和B页面是同源的基础上。

如果Web世界没有同源策略，当你登录FreeBuf账号并打开另一个站点时，这个站点上的JavaScript可以跨域读取你的FreeBuf账号数据，这样整个Web世界就无隐私可言了。

同源：协议、域名、端口全部相同才是同源，考虑到安全性，不同源之间不能够进行数据通信

解决不同源通信问题：

①通过iframe，虽然iframe属性中sandbox将安全性提升到了最高，但是我们可以通过他的不同的属性值开放不同的安全方面的限制。通过allow-same-origin以及allow-scripts开放出来对不同源之间数据的传递以及cookie或localstorage的信息共享

②在Ajax使用方面，不同源意味着发送请求出现Access-Control-Allow-Origin访问拒绝的提示，解决办法如下：

如果服务器方是我们自己的，那么可以通过跨域资源共享（CORS）来进行设置请求头Access-ControlAllow-Origin:域名或者*来进行开发这部分权限，达到不同源的数据共享

如果服务器是别人的，我们可以通过proxy来实现桥接，首先我们和自己的服务是同源的，那么我们携带请求的URL及参数，同源服务器进行代理请求，处理数据，然后返回给我们自己，实现不同源之间的数据共享script标签的src属性特性是不同源照样可以下载进行页面加载，我们可以通过动态添加script方式，携带一个回调函数名，并在script中提前把这个回调函数写好，将请求发送到不同源的服务端，服务端返回这个没有经过执行的回调函数，中间包含我们需要的数据，当返回客户端的时候，我

们提前写好的回调函数直接调用，即可获取到不同源之间返回的数据，实现资源共享
③通过window.name来进行跨域，在一个窗口的生命周期内，窗口载入的所有页面共享一个window.name
④通过H5新增的新特性postMessage()来进行跨域窗口处理数据

面试题 33. JavaScript 如何阻止事件冒泡和默认事件?

推荐指数: ★★★★ 试题难度: 高难 试题类型: 原理题

试题回答参考思路:

```
function stopBubble(e)
{
if (e && e.stopPropagation)
e.stopPropagation()
else
window.event.cancelBubble=true
}
return false
```

面试题 34. Javascript 阐述This对象的理解?

推荐指数: ★★★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

this是js的一个关键字，随着函数使用场合不同，this的值会发生变化。
但是有一个总原则，那就是this指的是调用函数的那个对象。
this一般情况下：是全局对象Global。 作为方法调用，那么this就是指这个对象

面试题 35. 清除浮动有哪些方式？比较好的方式是哪一种?

推荐指数: ★★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

（1）父级div定义height。
（2）结尾处加空div标签clear:both。
（3）父级div定义伪类:after和zoom。
（4）父级div定义overflow:hidden。
（5）父级div定义overflow:auto。
（6）父级div也浮动，需要定义宽度。
（7）父级div定义display:table。
（8）结尾处加br标签clear:both。

面试题 36. DOM怎样添加、移除、移动、复制、创建和查找节点

推荐指数: ★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

```
// 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
// 添加、移除、替换、插入
appendChild()
removeChild()
replaceChild()
insertBefore() //在已有的子节点前插入一个新的子节点
// 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
```

面试题 37. Javascript null和undefined的区别?

推荐指数：★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

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。

undefined：

（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：

（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。

面试题 38. Javascript new操作符具体作用?

推荐指数：★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

（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 。

面试题 39. 简述一下src与href的区别?

推荐指数：★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

href 是指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，用于超链接。

src是指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求src资源时会将其指向的资源下载并应用到文档内，例如js脚本，img图片和frame等元素。当浏览器解析到该元素时，

会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将js脚本放在底部而不是头部。

面试题 40. Javascript中callee和caller的作用?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

caller是返回一个对函数的引用，该函数调用了当前函数；
callee是返回正在被执行的function函数，也就是所指定的function对象的正文。

面试题 41. Javascript垃圾回收方法?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

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。

面试题 42. JavaScript 继承方式及其优缺点 ?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

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

```
/**
①组合方式优化1:
②组合方式优化2（最优解决方案）：
* 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。

面试题 43. javascript对象的几种创建方式?

推荐指数： ★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

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

面试题 44. Javascript继承的6种方法?

推荐指数： ★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

1，原型链继承
2，借用构造函数继承
3，组合继承(原型+借用构造)
4，原型式继承
5，寄生式继承
6，寄生组合式继承

面试题 45. JavaScript原型，原型链 ？有什么特点?

推荐指数： ★★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

* 原型对象也是普通的对象，是对象一个自带隐式的 __proto__ 属性，原型也有可能有自己的原型，如果一个原型对象的原型不为null的话，我们就称之为原型链。
* 原型链是由一些用来继承和共享属性的对象组成的（有限的）对象链

所有普通的 [[Prototype]] 链最终都会指向内置的 Object.prototype，其包含了 JavaScript 中许多通用的功能为什么能创建“类”，借助一种特殊的属性：所有的函数默认都会拥有一个名为 prototype 的共有且不可枚举的属性，它会指向另外一个对象，这个对象通常被称为函数的原型

```
function Person(name) {
this.name = name;
```

```
}
Person.prototype.constructor = Perso
```

在发生 new 构造函数调用时，会将创建的新对象的 [[Prototype]] 链接到 Person.prototype 指向的对象，这个机制就被称为原型链继承
方法定义在原型上，属性定义在构造函数上首先要说一下 JS 原型和实例的关系：每个构造函数（constructor）都有一个原型对象
（prototype），这个原型对象包含一个指向此构造函数的指针属性，通过 new 进行构造函数调用生成的实例，此实例包含一个指向原型对象的指针，也就是通过 [[Prototype]] 链接到了这个原型对象
然后说一下 JS 中属性的查找：当我们试图引用实例对象的某个属性时，是按照这样的方式去查找的，首先查找实例对象上是否有这个属性，如果没有找到，就去构造这个实例对象的构造函数的prototype所指向的对象上去查找，如果还找不到，就从这个 prototype 对象所指向的构造函数的prototype 原型对象上去查找

什么是原型链：这样逐级查找形似一个链条，且通过 [[Prototype]] 属性链接，所以被称为原型链
什么是原型链继承，类比类的继承：当有两个构造函数 A 和 B，将一个构造函数 A 的原型对象的，通过其 [[Prototype]] 属性链接到另外一个 B 构造函数的原型对象时，这个过程被称之为原型继承。

面试题 46. JavaScript的数据对象有那些属性值?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

writable：这个属性的值是否可以改。
    configurable：这个属性的配置是否可以删除，修改。
    enumerable：这个属性是否能在for...in循环中遍历出来或在Object.keys中列举出来。
    value：属性值。
* 当我们需要一个属性的时，Javascript引擎会先看当前对象中是否有这个属性， 如果没有的话，就会查找他的Prototype对象是否有这个属性。

```
function clone(proto) {
function Dummy() {
}
Dummy.prototype = proto;
Dummy.prototype.constructor = Dummy;
return new Dummy();
//等价于Object.create(Person);
}
function object(old) {
function F() {
}
;
F.prototype = old;
return new F();
}
var newObj = object(oldObject);
```

面试题 47. Javascript 描述以下变量的区别：null，undefined或undeclared?

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题 ▸

试题回答参考思路：

JavaScript的最初版本是这样区分的：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。

但是，上面这样的区分，在实践中很快就被证明不可行。目前，null和undefined基本是同义的，只有一些细微的差别。

null表示"没有对象"，即该处不应该有值。典型用法是：

用来初始化一个变量，这个变量可能被赋值为一个对象。
用来和一个已经初始化的变量比较，这个变量可以是也可以不是一个对象。
当函数的参数期望是对象时，被用作参数传入。
当函数的返回值期望是对象时，被用作返回值传出。
作为对象原型链的终点。
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：

变量被声明了，但没有赋值时，就等于undefined。
调用函数时，应该提供的参数没有提供，该参数等于undefined。
对象没有赋值的属性，该属性的值为undefined。
函数没有返回值时，默认返回undefined。
该如何检测它们?

null：表示无值；undefined：表示一个未声明的变量，或已声明但没有赋值的变量，或一个并不存在的对象属性。

==运算符将两者看作相等。如果要区分两者，要使用===或typeof运算符。

以下是不正确的用法：

```
1 var exp = undefined;
2
3 if (exp == undefined) {
4 alert("undefined");
5 }
```

exp为null时，也会得到与undefined相同的结果，虽然null和undefined不一样。注意：要同时判断undefined和null时可使用本法。

typeof 返回的是字符串，有六种可能："number"、"string"、"boolean"、"object"、"function"、"undefined"。

以下是正确的用法：

```
1 var exp = undefined;
2
3 if(typeof(exp) == undefined) {
4 alert("undefined");
5 }
```

JS中如何判断null?

以下是不正确的用法：

```
1 var exp = null;
2
3 if(exp == null) {
4 alert("is null");
5 }
```

exp为undefined时，也会得到与null相同的结果，虽然null和undefined不一样。注意：要同时判断null和undefined时可使用本法。

```
1 var exp=null;
2
3 if(!exp) {
4 alert("is null");
5 }
```

如果exp为undefined或者数字零，也会得到与null相同的结果，虽然null和二者不一样。注意：要同时判断null、undefined和数字零时可使用本法。

```
1 var exp = null;
2
3 if(typeof(exp) == "null") {
4 alert("is null");
5 }
```

为了向下兼容，exp为null时，typeof总返回object。这种方式也不太好。

以下是正确的用法：

```
1 var exp = null;
2
3 if(!exp&&typeof(exp) != "undefined" && exp != 0) {
4 alert("is null");
5 }
```

面试题 48. 请举出一个匿名函数的典型用例?

推荐指数：★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

匿名函数：就是没有函数名的函数。
匿名函数的基本形式为(function() {
...
}
)();
前面的括号包含函数体，后面的括号就是给匿名函数传递参数并立即执行， 匿名函数的作用是用于闭包和避免全局变量的污染以及函数名的冲突
函数的定义，大致可分为三种方式：
第一种：函数申明，这也是最常规的一种

```
function double(x) {
return 2 * x;
}
```

第二种：这种方法使用了Function构造函数，把参数列表和函数体都作为字符串，很不方便，不建议使用。

```
var double = new Function('x', 'return 2 * x;');
```

第三种：

```
var double = function(x) {
return 2* x;
}
```

面试题 49. 请指出JavaScript宿主对象和原生对象的区别?

推荐指数：★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

原生对象

ECMA-262 把本地对象（native object）定义为“独立于宿主环境的 ECMAScript 实现提供的对象”。

“本地对象”包含哪些内容：Object、Function、Array、String、Boolean、Number、Date、RegExp、Error、EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URIError。

由此可以看出，简单来说，本地对象就是 ECMA-262 定义的类（引用类型）。

内置对象

ECMA-262 把内置对象（built-in object）定义为“由 ECMAScript 实现提供的、独立于宿主环境的所有对象，在 ECMAScript 程序开始执行时出现”。这意味着开发者不必明确实例化内置对象，它已被实例化了。

同样是“独立于宿主环境”。根据定义我们似乎很难分清“内置对象”与“本地对象”的区别。而ECMA-262 只定义了两个内置对象，即 Global 和 Math （它们也是本地对象，根据定义，每个内置对象都是本地对象）。如此就可以理解了。内置对象是本地对象的一种。

宿主对象

何为“宿主对象”？主要在这个“宿主”的概念上，ECMAScript中的“宿主”当然就是我们网页的运行环境，即“操作系统”和“浏览器”。

所有非本地对象都是宿主对象（host object），即由 ECMAScript 实现的宿主环境提供的对象。所有的BOM和DOM都是宿主对象。因为其对于不同的“宿主”环境所展示的内容不同。其实说白了就是，ECMAScript官方未定义的对象都属于宿主对象，因为其未定义的对象大多数是自己通过ECMAScript程序创建的对象。

面试题 50. Javascript 请解释变量声明提升?

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

在JS里定义的变量，存在于作用域链里，而在函数执行时会先把变量的声明进行提升，仅仅是把声明进行了提升，而其值的定义还在原来位置。示例如下：

```
var test = function() {
console.log(name); // 输出：undefined
var name = "jeri";
console.log(name); // 输出：jeri
}

test();
```

上述代码与下述代码等价。

```
var test = function() {
var name;
console.log(name); // 输出：undefined
name = "jeri";
console.log(name); // 输出：jeri
}

test();
```

由以上代码可知，在函数执行时，把变量的声明提升到了函数顶部，而其值定义依然在原来位置

面试题 51. 简述attribute和property的区别？

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

1. 定义

Property：属性，所有的HTML元素都由HTMLElement类型表示，HTMLElement类型直接继承自Element并添加了一些属性，添加的这些属性分别对应于每个HTML元素都有下面的这5个标准特性：id,title,lang,dir,className。DOM节点是一个对象，因此，他可以和其他的JavaScript对象一样添加自定义的属性以及方法。property的值可以是任何的数据类型，对大小写敏感，自定义的property不会出现在html代码中，只存在js中。

Attribute：特性，区别于property，attribute只能是字符串，大小写不敏感，出现在innerHTML中，通过类数组attributes可以罗列所有的attribute。

2. 相同之处

标准的 DOM properties 与 attributes 是同步的。公认的（非自定义的）特性会被以属性的形式添加到DOM对象中。如，id，align，style等，这时候操作property或者使用操作特性的DOM方法如getAttribute()都可以操作属性。不过传递给getAttribute()的特性名与实际的特性名相同。因此对于class的特性值获取的时候要传入“class”。

3. 不同之处

1).对于有些标准的特性的操作，getAttribute与点号(.)获取的值存在差异性。如href，src，value，

style，onclick等事件处理程序。
2).href：getAttribute获取的是href的实际值，而点号获取的是完整的url，存在浏览器差异。

面试题 52. 请指出document.onload和document.ready两个事件的区别?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

页面加载完成有两种事件
一是ready，表示文档结构已经加载完成（不包含图片等非文字媒体文件）
二是onload，指示页面包含图片等文件在内的所有元素都加载完成

面试题 53. Javascript ==和===有什么不同?

推荐指数：★★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

首先，== equality 等同，=== identity 恒等。 ==， 两边值类型不同的时候，要先进行类型转换，再比较。 ===，不做类型转换，类型不同的一定不等。

先说 ===，这个比较简单。下面的规则用来判断两个值是否===相等：

如果类型不同，就[不相等]
如果两个都是数值，并且是同一个值，那么[相等]；(！例外)的是，如果其中至少一个是NaN，那么[不相等]。（判断一个值是否是NaN，只能用isNaN()来判断）
如果两个都是字符串，每个位置的字符都一样，那么[相等]；否则[不相等]。
如果两个值都是true，或者都是false，那么[相等]。
如果两个值都引用同一个对象或函数，那么[相等]；否则[不相等]。
如果两个值都是null，或者都是undefined，那么[相等]。
再说 ==，根据以下规则：

如果两个值类型相同，进行 === 比较。
如果两个值类型不同，他们可能相等。根据下面规则进行类型转换再比较：
如果一个是null、一个是undefined，那么[相等]。
如果一个是字符串，一个是数值，把字符串转换成数值再进行比较。
如果任一值是 true，把它转换成 1 再比较；如果任一值是 false，把它转换成 0 再比较。
如果一个是对象，另一个是数值或字符串，把对象转换成基础类型的值再比较。对象转换成基础类型，利用它的toString或者valueOf方法。js核心内置类，会尝试valueOf先于toString；例外的是Date，Date利用的是toString转换。非js核心的对象，令说（比较麻烦，我也不大懂）
任何其他组合，都[不相等]。

面试题 54. 如何从浏览器的URL中获取查询字符串参数?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

```
function parseQueryString ( name ){
name = name.replace(/[[]/,"\[");
var regexS = "[\?&]"+name+"=([^&#]*)";
```

```
var regex = new RegExp( regexS );
var results = regex.exec( window.location.href );

if(results == null) {
return "";
} else {
return results[1];
}
}
```

面试题 55. Javascript 什么是三元表达式?“三元”表示什么意思?

推荐指数: ★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

三元表达式:?:。三元--三个操作对象。

在表达式boolean-exp ? value0 : value1 中，如果“布尔表达式”的结果为true，就计算“value0”，而且这个计算结果也就是操作符最终产生的值。如果“布尔表达式”的结果为false，就计算“value1”，同样，它的结果也就成为了操作符最终产生的值

面试题 56. JavaScript里函数参数arguments是数组吗?

推荐指数: ★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

在函数代码中，使用特殊对象 arguments，开发者无需明确指出参数名，通过使用下标就可以访问相应的参数。

arguments虽然有一些数组的性质，但其并非真正的数组，只是一个类数组对象。其并没有数组的很多方法，不能像真正的数组那样调用.jion(),.concat(),.pop()等方法。

面试题 57. Javascript 什么是use strict?使用它的好处和坏处分别是什么?

推荐指数: ★★★★ 试题难度: 高难 试题类型: 原理题

试题回答参考思路:

在代码中出现表达式-"use strict"; 意味着代码按照严格模式解析，这种模式使得Javascript在更严格的条件下运行。

好处:

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。
坏处:

同样的代码，在"严格模式"中，可能会有不一样的运行结果；一些在"正常模式"下可以运行的语句，在"严格模式"下将不能运行。

面试题 58. 阐述JavaScript事件委托是什么？

推荐指数： ★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

1.什么是事件委托?
事件委托也称之为事件代理（Event Delegation）。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在子元素的响应事件委托给父元素，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。

举个通俗的例子：比如一个宿舍的同学同时快递到了，一种方法就是他们一个个去领取，还有一种方法就是把这件事情委托给宿舍长，让一个人出去拿好所有快递，然后再根据收件人一 一分发给每个宿舍同学；
在这里，取快递就是一个事件，每个同学指的是需要响应事件的 DOM 元素，而出去统一领取快递的宿舍长就是代理的元素，所以真正绑定事件的是这个元素，按照收件人分发快递的过程就是在事件执行中，需要判断当前响应的事件应该匹配到被代理元素中的哪一个或者哪几个。

一个事件触发后，会在子元素和父元素之间传播（propagation）。这种传播分成三个阶段。
（1）捕获阶段：从window对象传导到目标节点（上层传到底层）称为“捕获阶段”（capture phase），捕获阶段不会响应任何事件；
（2）目标阶段：在目标节点上触发，称为“目标阶段”
（3）冒泡阶段：从目标节点传导回window对象（从底层传回上层），称为“冒泡阶段”（bubbling phase）。事件代理即是利用事件冒泡的机制把里层所需要响应的事件绑定到外层。

事件委托的优点：
【1】可以大量节省内存占用，减少事件注册，比如在ul上代理所有li的click事件就非常棒

- item 1
- item 2
- item 3

  ……
- item n

// …… 代表中间还有未知数个 li
如上面代码所示，如果给每个li列表项都绑定一个函数，那对内存的消耗是非常大的，因此较好的解决办法就是将li元素的点击事件绑定到它的父元素ul身上，执行事件的时候再去匹配判断目标元素。

【2】可以实现当新增子对象时无需再次对其绑定（动态绑定事件）

假设上述的例子中列表项li就几个，我们给每个列表项都绑定了事件；
在很多时候，我们需要通过 AJAX 或者用户操作动态的增加或者删除列表项li元素，那么在每一次改变的时候都需要重新给新增的元素绑定事件，给即将删去的元素解绑事件；

如果用了事件委托就没有这种麻烦了，因为事件是绑定在父层的，和目标元素的增减是没有关系的，执行到目标元素是在真正响应执行事件函数的过程中去匹配的；所以使用事件在动态绑定事件的情况下是可以减少很多重复工作的。

使用事件委托注意事项：使用“事件委托”时，并不是说把事件委托给的元素越靠近顶层就越好。事件冒泡的过程也需要耗时，越靠近顶层，事件的”事件传播链”越长，也就越耗时。如果DOM嵌套结构很深，事件冒泡通过大量祖先元素会导致性能损失。

2.怎么阻止默认动作?

有一些html元素默认的行为，比如说a标签，点击后有跳转动作；form表单中的submit类型的input有一个默认提交跳转事件；reset类型的input有重置表单行为。

如果你想阻止这些浏览器默认行为，JavaScript为你提供了方法。

如下代码：

```
var $a = document.getElementsByTagName("a")[0];
$a.onclick = function(e){
alert("跳转动作被我阻止了")
e.preventDefault();
//return false;//也可以
}
```

默认事件没有了。

既然return false 和 e.preventDefault()都是一样的效果，那它们有区别吗？当然有。
仅仅是在HTML事件属性 和 DOM0级事件处理方法中，才能通过返回 return false 的形式组织事件宿主的默认行为。

3.怎么阻止冒泡事件

```
function stopBubble(e){
if(e&&e.stopPropagation){//非IE
e.stopPropagation();
}
else{//IE
window.event.cancelBubble=true;
}
}
```

面试题 59. 解释JavaScript eval 作用?

推荐指数：★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

它的功能是把对应的字符串解析成JS代码并运行；
应该避免使用eval，不安全，非常耗性能（2次，一次解析成js语句，一次执行）。
由JSON字符串转换为JSON对象的时候可以用eval，var obj =eval('('+ str +')');

面试题 60. 简述在Javascript中什么是伪数组？如何将伪数组转化为标准数组?

推荐指数：★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

伪数组（类数组）：无法直接调用数组方法或期望length属性有什么特殊的行为，但仍可以对真正数组遍历方法来遍历它们。典型的是函数的argument参数，还有像调用getElementsByTagName,document.childNodes之类的,它们都返回NodeList对象都属于伪数组。
可以使用Array.prototype.slice.call(fakeArray)将数组转化为真正的Array对象。

```
function log(){
var args = Array.prototype.slice.call(arguments);
//为了使用unshift数组方法，将argument转化为真正的数组
args.unshift('(app)');
console.log.apply(console, args);
};
```

面试题 61. 简述对webpack的理解?

推荐指数：★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

WebPack 是一个模块打包工具，你可以使用WebPack管理你的模块依赖，并编绎输出模块们所需的静态文件。它能够很好地管理、打包Web开发中所用到的HTML、JavaScript、CSS以及各种静态文件（图片、字体等），让开发过程更加高效。对于不同类型的资源，webpack有对应的模块加载器。webpack模块打包器会分析模块间的依赖关系，最后 生成了优化且合并后的静态资源。
webpack的两大特色：
1.code splitting（可以自动完成）
2.loader 可以处理各种类型的静态文件，并且支持串联操作
webpack 是以commonJS的形式来书写脚本滴，但对 AMD/CMD 的支持也很全面，方便旧项目进行代码迁移。
webpack具有requireJs和browserify的功能，但仍有很多自己的新特性：
1. 对 CommonJS 、 AMD 、ES6的语法做了兼容
2. 对js、css、图片等资源文件都支持打包
3. 串联式模块加载器以及插件机制，让其具有更好的灵活性和扩展性，例如提供对CoffeeScript、ES6的支持
4. 有独立的配置文件webpack.config.js
5. 可以将代码切割成不同的chunk，实现按需加载，降低了初始化时间
6. 支持 SourceUrls 和 SourceMaps，易于调试
7. 具有强大的Plugin接口，大多是内部插件，使用起来比较灵活
8.webpack 使用异步 IO 并具有多级缓存。这使得 webpack 很快且在增量编译上更加快

面试题 62. 简述你如何给一个事件处理函数命名空间，为什么要这样做?

推荐指数：★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

任何作为type参数的字符串都是合法的；如果一个字符串不是原生的JavaScript事件名，那么这个事件处理函数会绑定到一个自定义事件上。这些自定义事件绝对不会由浏览器触发，但可以通过使用.trigger()或者.triggerHandler()在其他代码中手动触发。如果type参数的字符串中包含一个点(.)字符，那么这个事件就看做是有命名空间的了。这个点字符就用来分隔事件和他的命名空间。举例来说，

如果执行.bind('click.name',handler)，那么字符串中的click是事件类型，而字符串name就是命名空间。命名空间允许我们取消绑定或者触发一些特定类型的事件，而不用触发别的事件。参考unbind()来获取更多信息。

jQuery的bind/unbind方法应该说使用很简单，而且大多数时候可能并不会用到，取而代之的是直接用click/keydown之类的事件名风格的方法来做事件绑定操作。

但假设如下情况：需要在运行时根据用户交互的结果进行不同click事件处理逻辑的绑定，因而理论上会无数次对某一个事件进行bind/unbind操作。但又希望unbind的时候只把自己绑上去的处理逻辑给释放掉而不是所有其他地方有可能的额外的同一事件绑定逻辑。这时候如果直接用.click()/.bind('click')加上.unbind('click')来进行重复绑定的话，被unbind掉的将是所有绑定在元素上的click处理逻辑，潜在会影响到该元素其他第三方的行为。

当然如果在bind的时候是显示定义了function变量的话，可以在unbind的时候提供function作为第二个参数来指定只unbind其中一个处理逻辑，但实际应用中很可能会碰到各种进行匿名函数绑定的情况。对于这种问题，jQuery的解决方案是使用事件绑定的命名空间。即在事件名称后添加.something来区分自己这部分行为逻辑范围。

比如用.bind('click.myCustomRoutine',function(){...});同样是把匿名函数绑定到click事件（你可以用自己的命名空间多次绑定不同的行为方法上去），当unbind的时候用.unbind('click.myCustomRoutine')即可释放所有绑定到.myCustomRoutine命名空间的click事件，而不会解除其他通过.bind('click')或另外的命名空间所绑定的事件行为。同时，使用命令空间还可以让你一次性unbind所有此命名空间下的自定义事件绑定，通过.unbind('.myCustomRoutine')即可。要注意的是，jQuery的命名空间并不支持多级空间。

因为在jQuery里面，如果用.unbind('click.myCustomRoutine.myCustomSubone')，解除的是命名空间分别为myCustomRoutine和myCustomSubone的两个并列命名空间下的所有click事件，而不是"myCustomRoutine下的myCustomSubone子空间"。

面试题 63. JavaScript中的split、slice、splice函数区别?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

a . string.split(separator,limit)
该方法是将string分割成片段来创建一个字符串数组，可选参数limit可以限制被分割的片段数量
separator参数可以是字符串,或者正则表达式,separator是一个空字符，会返回一个单字符的数组。
limit是返回的数量限制
//"hello world".split('',1);//["h"]
// "hello world".split('',5);//["h", "e", "l", "l", "o"]
//"hello world".split('',-1);//["h", "e", "l", "l", "o", " ", "w", "o", "r", "l", "d"]
// "hello world".split(' ',2);//["hello", "world"]
//"hello world".split(' ',3);//["hello", "world"]
b .string.slice(start,end)
解释：slice方法复制string的一部分来构造一个新的字符串
// "hello world".slice(0,5);//"hello"
//"hello world".slice(0,-1);//"hello worl"
c.数组的splice
array.splice(start,deleteCount,item...)
splice方法从array中移除一个或多个数组，并用新的item替换它们。
参数start是从数组array中移除元素的开始位置,默认从0开始，参数deleteCount是要移除的元素的个

数。
//var a=['a','b','c'];//var b=a.splice(1,1,'e','f'); //a=['a','e','f','c'],b=['b']

面试题 64. 全面阐述JavaScript ES6的理解？

推荐指数：★★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

编程语言JavaScript是ECMAScript的实现和扩展，由ECMA（一个类似W3C的标准组织）参与进行标准化。ECMAScript定义了：

语言语法 — 语法解析规则、关键字、语句、声明、运算符等。

类型 — 布尔型、数字、字符串、对象等。

原型和继承

内建对象和函数的标准库 — JSON、Math、数组方法、对象自省方法等。

ECMAScript标准不定义HTML或CSS的相关功能，也不定义类似DOM（文档对象模型）的Web API，这些都在独立的标准中进行定义。ECMAScript涵盖了各种环境中JS的使用场景，无论是浏览器环境还是类似node.js的非浏览器环境。

新标准
上周，ECMAScript语言规范第6版最终草案提请Ecma大会审查，这意味着什么呢?

这意味着在今年夏天，我们将迎来最新的JavaScript核心语言标准。

这无疑是一则重磅新闻。早在2009年，上一版ES5刚刚发布，自那时起，ES标准委员会一直在紧锣密鼓地筹备新的JS语言标准——ES6。

ES6是一次重大的版本升级，与此同时，由于ES6秉承着最大化兼容已有代码的设计理念，你过去编写的JS代码将继续正常运行。事实上，许多浏览器已经支持部分ES6特性，并将继续努力实现其余特性。这意味着，在一些已经实现部分特性的浏览器中，你的JS代码已经可以正常运行。如果到目前为止你尚未遇到任何兼容性问题，那么你很有可能将不会遇到这些问题，浏览器正飞速实现各种新特性。

版本号6
ECMAScript标准的历史版本分别是1、2、3、5。

那么为什么没有第4版？其实，在过去确实曾计划发布提出巨量新特性的第4版，但最终却因想法太过激进而惨遭废除（这一版标准中曾经有一个极其复杂的支持泛型和类型推断的内建静态类型系统）。

ES4饱受争议，当标准委员会最终停止开发ES4时，其成员同意发布一个相对谦和的ES5版本，随后继续制定一些更具实质性的新特性。这一明确的协商协议最终命名为“Harmony”，因此，ES5规范中包含这样两句话：

ECMAScript是一门充满活力的语言，并在不断进化中。

未来版本的规范中将持续进行重要的技术改进。

这一声明许下了一个未来的承诺。

兑现承诺
2009年发布的改进版本ES5，引入了Object.create()、Object.defineProperty()、getters和setters、严格模式以及JSON对象。我已经使用过所有这些新特性，并且我非常喜欢ES5做出的改进。但事实上，这些改进并没有深入影响我编写JS代码的方式，对我来说最大的革新大概就是新的数组方法：.map()、. filter()这些。

但是，ES6并非如此！经过持续几年的磨砺，它已成为JS有史以来最实质的升级，新的语言和库特性就像无主之宝，等待有识之士的发掘。新的语言特性涵盖范围甚广，小到受欢迎的语法糖，例如箭头函数（arrow functions）和简单的字符串插值（string interpolation），大到烧脑的新概念，例如代理（proxies）和生成器（generators）。

ES6将彻底改变你编写JS代码的方式！

面试题 65. JavaScript 中的hoisting是什么?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

hoisting意味着所有的声明都被移动到范围的顶部，这发生在代码运行之前。
对于函数，这意味着您可以从范围内的任何地方调用它们，甚至在它们被定义之前。

```
hello(); // Prints "Hello world! " even though the function is called "before" declaration

function hello(){
console.log("Hello world! ");
}
```

对于变量，hoisting有点不同，它在定义变量范围之前，它们为 undefined 。

例如，在定义变量之前调用它：

```
console.log(dog);
var dog = "Spot";
```

结果是：

```
undefined
```

这可能会令人惊讶，因为您可能预计它会导致错误。
如果你声明一个函数或变量，无论你在哪里声明它，它总是被移动到范围的顶部

面试题 66. 简述Javascript isNan() 函数 ?

推荐指数：★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

您可以使用 isNan() 函数来检查值的类型是否为数字且为 NaN。
（是的，NaN 是数字类型，尽管它的名称是“非数字”。这是因为它是底层的数字类型，即使它没有数值。）

例如：

```
function toPounds(kilos) {
if (isNaN(kilos)) {
return 'Not a Number! Cannot be a weight.';
```

```
}
return kilos * 2.2;
}

console.log(toPounds('this is a test'));
console.log(toPounds('100'));
```

输出：

```
Not a Number! Cannot be a weight.
220.00000000000003
```

面试题 67. JavaScript 中的负无穷大是什么?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

如果在 JavaScript 中将负数除以零，则会得到负无穷大。
例如：

```
console.log(-10/0)
```

输出：

```
-Infinity
```

面试题 68. Javascript 什么是未声明变量？未定义的变量怎么样?

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

程序中根本不存在未声明的变量。如果您的程序试图读取未声明的变量，则会引发运行时错误。
调用未声明变量的示例显然会导致错误：

```
console.log(dog);
```

输出：

```
error: Uncaught ReferenceError: dog is not defined
```

未定义的变量在程序中声明但没有值。如果程序尝试读取未定义的变量，则会返回未定义的值并且应用程序不会崩溃。
未定义变量的示例是：

```
let car;
console.log(car);
```

输出：

```
undefined
```

面试题 69. JavaScrpit隐式类型强制有什么作用？举个例子 ？

推荐指数：★★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

隐式类型强制意味着一个值在幕后从一种类型转换为另一种类型。当表达式的操作数属于不同类型时，就会发生这种情况。

例如，字符串强制转换意味着在数字上应用 + 运算符，字符串会自动将数字转换为字符串。

例如：
var x = 1;
var y = "2";
x + y // Returns "12"
但是在处理减法时，强制以另一种方式起作用。它将字符串转换为数字。

例如：
var x = 10;
var y = "10";
x - y // Returns 0

面试题 70. JavaScript 是静态类型语言还是动态类型语言？这是什么意思？

推荐指数：★★★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

JavaScript 是动态类型的。
这意味着在运行时检查对象的类型。（在静态类型语言中，类型在编译时检查。）
换句话说，JavaScript 变量与类型无关，这意味着您可以毫无问题地更改数据类型。
var num = 10;
num = "Test";
在静态类型语言（如 C++）中，不可能以这种方式将整数更改为字符串

面试题 71. 简述JavaScript 中的 NaN 是什么？

推荐指数：★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

NaN属性代表一个“不是数字”的值。这个特殊的值是因为运算不能执行而导致的，不能执行的原因可能是其中的运算对象之一非数字（例如，"abc"/4），也可能是是运算的结果非数字（例如，除数为零）。
虽然这看上去很简单，但NaN有一些令人惊讶的特点，如果你不知道它们，可能会导致令人头痛的Bug。
首先，虽然NaN意味着“不是数字”，但是它的类型是 Number。
console. log（typeof NaN ==="number"）;//true
此外，NaN和任何内容比较一甚至是它自己本身一结果都是 false 。
console. log（NaN === NaN）；// logs "false"
可以用一种半可靠的方法来判断了一个数字是否等于NaN，使用内置函数 isNaNO但即使使用 isTaNA也并非是一个完美的解决方案。
一个更好的解决办法是使用 value！= value，如果值等于NaN，只会产生true.另外， ECMAScript6中拓展了两个处理NaN的方法。一个是 Number isnaN（）函数，比全局 isNaN（函数更可靠，比较的时候不会做数据类型的转换。另一个是 Object is，它可以判断两个NaN是否相等。

面试题 72. 解释JavaScript 中的展开运算符是什么？

推荐指数：★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

扩展运算符允许将可迭代对象（数组/对象/字符串）扩展为单个参数/元素。让我们举个例子来看看这个行为，

```
function sum(a, b, c) {
return a + b + c;
}
const nums = [15, 25, 35];
console.log(sum(...nums));
```

输出：

75

面试题 73. JavaScript 中如何处理异常?

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

如果表达式抛出错误，您可以使用 try...catch 语句处理它们。

使用此结构的想法是尝试运行一个表达式，例如带有输入的函数，并捕获可能的错误。

例如：

```
function weekDay(dayNum) {
if (dayNum < 1 || dayNum > 7) {
throw 'InvalidDayNumber'
} else {
return ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'][dayNum - 1];
}
}

try { // Try to run the following
let day = weekDay(8);
console.log(day);
}
catch (e) { // catch an error if the above try failed
let day = 'unknown';
console.log(e);
}
```

面试题 74. 解释JavaScript 中的“作用域”是什么意思?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

范围定义了“代码的可见性”。
更正式地说，范围描述了变量、函数和其他对象在代码中的哪些位置可以访问。范围是在运行时在您的代码中创建的。

例如，块作用域表示花括号之间的“区域”：

```
if(true) {
```

```
let word = "Hello";
}
console.log(word); // ERROR OCCURS
```

在这里，除了在 if 语句中，不能从其他任何地方访问变量 word

面试题 75. 简述 JavaScript 中的高阶函数是什么?

推荐指数: ★★★★★ 试题难度: 中级 试题类型: 原理题

试题回答参考思路:

高阶函数对另一个函数进行操作。它要么接受一个函数作为参数，要么返回另一个函数。
例如:

```
function runThis(inputFunction) {
inputFunction();
}
runThis(function() { console.log("Hello world") });
```

输出

```
Hello
world
```

另外一个例子:

```
function giveFunction() {
return function() {
console.log("Hello world")
}
}
var action = giveFunction();
action()
```

输出:

```
Hello world
```

面试题 76. 简述JavaScript什么是bind()方法?

推荐指数: ★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

bind() 方法返回一个新函数，其 this 已设置为另一个对象。
与 apply() 和 call() 不同，bind() 不会立即执行函数。相反，它返回一个新版本的函数，其 this 被设置为另一个值。
让我们看一个例子:

```
let person = {
name: 'John',
getName: function() {
console.log(this.name);
}
}
;
window.setTimeout(person.getName, 1000);
```

这不会打印名称“John”，而是打印 undefined。要理解为什么会发生这种情况，请以等效方式重写最后一行:

let func = person.getName;
setTimeout(func, 1000);
setTimeout() 接收与人对象分开的函数，但没有人的名字。因此，当 setTimeout() 调用 person.getName 时，名称是未定义的。
要解决此问题，您需要将 getName() 方法绑定到 person 对象：
let func = person.getName.bind(person);
setTimeout(func, 1000);
输出：
John
让我们检查一下这种方法是如何工作的：
person.getName 方法绑定到 person 对象。
绑定函数 func 现在将 this 值设置为 person 对象。当您将这个新的绑定函数传递给 setTimeout() 函数时，它知道如何获取此人的姓名

面试题 77. 简述JavaScript中什么是柯里化?

推荐指数：★★★★ 试题难度：高难 试题类型：原理题

试题回答参考思路：

柯里化意味着将具有 n 个参数的函数转换为具有一个或更少参数的 n 个函数。
例如，假设您有一个函数 add() 可以对两个数字求和：
function add(a, b) {
return a + b;
}
您可以通过以下方式调用此函数：
add(2,3)
然后让我们柯里化函数：
function add(a) {
return function(b) {
return a + b;
}
}
现在您可以通过以下方式调用此柯里化函数：
add(2)(3)
柯里化不会改变函数的行为，它改变了它被调用的方式。

面试题 78. 解释什么是 JavaScript 中的承诺?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

Promise 是一个对象，它可能在未来产生一个值。

承诺总是处于可能的状态之一：已履行、已拒绝或未决。

创建承诺如下所示：

```
const promise = new Promise(function(resolve, reject) {
// implement the promise here
})
```

例如，让我们创建一个在被调用两秒后解析的承诺。

```
const promise = new Promise(resolve => {
setTimeout(() => {
resolve("Hello, world!");
}, 2000);
}, reject => {});
```

现在 promises 的关键是您可以在使用 .then() 方法解析 promise 后立即执行代码：

```
promise.then(result => console.log(result));
```

输出：

```
Hello, world!
```

Promise 可以链接在一起，以便已解决的 promise 返回一个新的 promise

面试题 79. JavaScript为什么要使用promises?

推荐指数：★★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

在 JavaScript 中，promises 对于异步操作很有用。
过去，必须使用回调来处理异步操作，即使用在操作完成后立即执行的函数。这导致了“回调地狱”，一个带有嵌套回调的金字塔形代码。
Promises 通过减少回调地狱和编写更清晰的代码，为回调提供了另一种方法

面试题 80. 解释JavaScript 中的回调函数？

推荐指数：★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

回调函数是作为参数传递给另一个函数的函数。当某些动作完成时，此函数在传递给它的函数内部执行以“回调”。
让我们看一个例子：

```
function greetName(name) {
console.log('Hello ' + name);
}
function askName(callback) {
let name = prompt('Enter your name.');
callback(name);
}
askName(greetName);
```

此代码会提示您输入姓名，当您输入姓名时，它会对该姓名说“你好”。因此，回调函数（在本例中为 greetName）仅在您输入名称后执行。

面试题 81. 简述为什么在 JavaScript 中使用回调?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

回调很有用，因为 JavaScript 是一种事件驱动的语言。换句话说，它不是等待响应，而是在监听其他事件的同时继续执行。
上面的例子展示了回调在 JavaScript 中的用处：

```
function greetName(name) {
console.log('Hello ' + name);
}
function askName(callback) {
let name = prompt('Enter your name.');
callback(name);
}
askName(greetName);
```

面试题 82. 简述什么是 JavaScript 中的严格模式?

推荐指数：★★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

严格模式允许您将程序设置为在严格的上下文中运行。这可以防止执行某些操作。此外，还会抛出更多异常。
表达“使用严格”；告诉浏览器启用严格模式。
例如：

```
"use strict";
number = 1000;
```

这会导致错误，因为严格模式会阻止您为未声明的变量赋值。

面试题 83. 解释什么是JavaScript立即调用函数?

推荐指数：★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

立即调用函数（IIFE）在定义后立即运行。
例如：

```
(function() {
// action here
}
)();
```

要了解 IIFE 的工作原理，请查看它周围的括号：
当 JavaScript 看到关键字 function 时，它假设有一个函数声明来了。
但是上面的声明是无效的，因为函数没有名字。
为了解决这个问题，使用了声明周围的第一组括号，这告诉解释器这是一个函数表达式，而不是声明。

```
(function () {
// action here;
}
)
```

然后，要调用该函数，需要在函数声明的末尾添加另一组括号，这类似于调用任何其他函数：

```
(function () {
// action here
}
)();
```

面试题 84. 解释为什么要在 JavaScript 中使用严格模式?

推荐指数: ★★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

严格模式有助于编写“安全”的 JavaScript 代码。这意味着错误的语法实践会转化为真正的错误。
例如，严格模式会阻止创建全局变量。
要声明严格模式，请添加“use strict”；在你想处于严格模式下的语句之前的语句:

```
'use strict';
const sentence = "Hello, this is very strict";
```

面试题 85. JavaScript如何删除属性及其值?

推荐指数: ★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

您可以使用 delete 关键字从对象中删除属性及其值。
让我们看一个例子:

```
var student = {name: "John", age:20};
delete student.age;
console.log(student);
```

输出:

```
{name: "John"}
```

面试题 86. 如何检查 JavaScript 中变量的类型?

推荐指数: ★★★ 试题难度: 初级 试题类型: 原理题

试题回答参考思路:

使用 typeof 运算符。

```
typeof "John Abraham" // Returns "string"
typeof 100 // Returns "number"
```

面试题 87. JavaScript语言中preventDefault() 方法有什么作用?

推荐指数: ★★★★★ 试题难度: 高难 试题类型: 原理题

试题回答参考思路:

preventDefault() 取消一个方法。名称 preventDefault 很好地描述了这种行为。它阻止事件采用默认行为。
例如，您可以在单击提交按钮时阻止表单提交:

```
document.getElementById("link").addEventListener("click", function(event){
event.preventDefault()
});
```

面试题 88. JavaScript什么是setTimeout()方法?

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

setTimeout() 方法在指定的毫秒数后调用一个函数（一次）。例如，让我们在一秒（1000 毫秒）后记录一条消息：

```
setTimeout(function() {
console.log("Good day");
}, 1000);
```

面试题 89. JavaScript什么是setInterval()方法?

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

setInterval() 方法以自定义间隔定期调用函数。
例如，让我们每秒定期记录一条消息：

```
setInterval(function() {
console.log("Good day");
}, 1000);
```

面试题 90. 简述什么是 ECMAScript?

推荐指数：★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

ECMAScript 是构成 JavaScript 基础的脚本语言。
ECMAScript 由 ECMA 国际标准组织标准化（查看 ECMA-262 和 ECMA-402 规范）。

面试题 91. 简述什么是JSON stringify?

推荐指数：★★★★ 试题难度：中级 试题类型：原理题

试题回答参考思路：

当您向服务器发送数据时，它必须是一个字符串。
要将 JavaScript 对象转换为字符串，可以使用 JSON.stringify() 方法。

```
var dataJSON = {name: "Matt", age: 51};
var dataString = JSON.stringify(dataJSON);
console.log(dataString);
```

输出：

```
'{"name":"Matt","age":51}'
```

面试题 92. JavaScript如何将 JSON 字符串转换为 JSON 对象?

推荐指数：★★★ 试题难度：初级 试题类型：原理题

试题回答参考思路：

当您从服务器接收数据时，它始终是字符串格式。要将 JSON 字符串转换为 JavaScript 对象，请使用 JSON.parse() 方法。

```
var data = '{"name":"Matt", "age":51}';
var dataJSON = JSON.parse(data);
console.log(dataJSON);
```

输出：

```
{
name:"Matt",
age:51
}
```

面试题 93. JavaScript如何为变量分配默认值?

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

使用逻辑运算符 || 在分配中提供默认值。

```
const a = b || c;
```

这使得如果 b 是假的，那么 c 将被分配给 a。（Falsy 表示 null、false、undefined、0、空字符串或 NaN。）

面试题 94. 简述JavaScript标签中 defer和 async属性的区别？

推荐指数： ★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

区别如下。

（1） defer属性规定是否延迟执行脚本，直到页面加载为止， async属性规定脚本一旦可用，就异步执行。

（2） defer并行加载 JavaScript文件，会按照页面上 script标签的顺序执行， async并行加载JavaScript文件，下载完成立即执行，不会按照页面上 script标签的顺序执行

面试题 95. 列举几种类型的DOM节点？

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

整个文档是一个文档（ Document）节点。
每个HTML标签是一个元素（ Element）节点。
每一个HTML属性是一个属性（ Attribute）节点。
包含在HTML元素中的文本是文本（Text）节点

面试题 96. 简述解释一下JavaScript unshift（）方法?

推荐指数： ★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

该方法在数组启动时起作用，与 push()不同。它将参数成员添加到数组的顶部下面给出一段示例代。

```
var name=["john"]
name. unshift（"charlie"）；
name.unshift（"joseph"，"Jane"）；
console. log（name）；
```

输出如下所示。

```
[" joseph "， Jane "，" charlie "，" john "]
```

面试题 97. 简述为什么不建议在 JavaScript中使用 innerHTML?

推荐指数：★★★★★ 试题难度： 高难 试题类型： 原理题

试题回答参考思路：

通过 innerHTML修改内容，每次都会刷新，因此很慢。在 innerHTML中没有验证的机会，因此更容易在文档中插入错误代码，使网页不稳定

面试题 98. 如何在不支持 JavaScript的旧浏览器中隐藏 JavaScript代码?

推荐指数：★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

在< script>标签之后的代码中添加“<！ --”，不带引号。
在< /script>标签之前添加“//-->”，代码中没有引号。
旧浏览器现在将 JavaScript代码视为一个长的HTML注释，而支持 JavaScript的浏览器则将"<！ -"和"//-->"作为一行注释

面试题 99. 讲解一下 JavaScript对象的几种创建方式 ?

推荐指数：★★★ 试题难度： 初级 试题类型： 原理题

试题回答参考思路：

有以下创建方式：
（1） Object构造函数式。
（2）对象字面量式。
（3）工厂模式。
（4）安全工厂模式。
（5）构造函数模式。
（6）原型模式。
（7）混合构造函数和原型模式。
（8）动态原型模式。
（9）寄生构造函数模式。
（10）稳妥构造函数模式。

面试题 100. JavaScript如何实现异步编程?

推荐指数：★★★★ 试题难度： 中级 试题类型： 原理题

试题回答参考思路：

具体方法如下：

方法1，通过回调函数。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），流程混乱，而且每个任务只能指定一个回调函数。

方法2，通过事件监听，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”（ Decoupling），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。

方法3，采用发布/订阅方式。性质与“事件监听”类似，但是明显优于后者。

方法4，通过 Promise对象实现， Promise对象是 Commonjs工作组提出的一种规范，旨在为异步编程提供统一接口。它的思想是，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