面试1：

问题一：

笔试题：

1. **水平垂直居中：**

width: 200px;height: 200px;left: 50%;top: 50%;position: absolute;margin: -100px 0 0 -100px;

1. **解释下css sprites ，如何使用**

CSS Sprites 是一种网页图片处理的一种方式，就是把网页中一些背景图片整合到一张图

片文件中，用CSS”background-image、background-repeat、background-position的组合进行背景定位。

优点：（1）减少网页的 http 请求，提高性能（2）减少图片的字节：多张图片合并成1张；

(3)减少了命名困扰：集合图起名就行，不用对每一个小元素进行命名提高制作效率；

(4)更换风格方便：只需要在一张或少张图片上修改图片的颜色或样式，整个网页

的风格就可以改变，维护起来更加方便。

缺点：(1)图片合成比较麻烦；(2)背景设置时，需要得到每一个背景单元的精确位置，；

1. 维护合成图片时，最好只是往下加图片，而不要更改已有图片。
2. **rem和em区别？**

Em：EM是相对于父元素来设计字体大小的1 ÷ 父元素的font-size × 需要转换的像素值 = em值

body选择器中声明Font-size=62.5%；将你的原来的px数值除以10，然后换上em作为单位；*12px=1.2em,*

Rem：相对于根元素<html>默认16px，需要转换值÷16=rem值

# 4、[一行代码实现数组去重（ES6）](https://www.cnblogs.com/raocheng/articles/6549556.html)

let array = Array.from(new Set([1, 1, 1, 2, '3', 3, 2, 4]));

console.log(array);//[1, 2, "3", 3, 4]

new Set(array)返回类数组结象 {1, 2, 3, 4, 5}有size

Array.from可以把类似数组的对象转换为数组[1, 2, 3, 4, 5]有length

1. 右边宽度固定，左边自适应，编写CSS代码

（1）

<div class='wraper'>

    <div class="left">自适应区</div>

    <div class="right">固定宽度区</div>

</div>

.wraper {display: table;}

.left { display: table-cell;border: 1px solid #00f;width: 100%;}

.right {width: 200px;border: 1px solid #f00;}

或：

.wraper { display: flex;}

.left {border: 1px solid #00f;flex: 1;}

.right { width: 200px;border: 1px solid #f00;}

**5、js原型继承的几种方式**

在 JavaScript 中，继承都是源于原型，有多种实现方式有call的都是在子类里写的

1、原型链继承：重点圈起来：将父类实例赋值给子类原型对象

function Parent() {

    this.name = "a";

}

function A() {

    this.age = "20";

}

//将父类实例赋值给子类原型对象A继承了Parent,通过原型，形成链条

A.prototype = new Parent();

// 将constructor指向本身，保证原型链不断。

A.prototype.constructor = A;

var a = new A();

console.log(a.name)//结果：a

#### **2、构造继承：**重点圈起来：执行父构造，将This指向本身，拉取父私有属性

// 父级

function Parent(name) {

    this.name = name;

}

// 创建子类、添加子类属性。

function A(name) {

    Parent.call(this, name) // 执行父构造，将This指向本身，拉取父私有属性；

}

A.prototype.getName = function () {

    console.log('我叫' + this.name)

}

var a = new A('李四');

console.log(a.name)  //李四

a.getName()          //我叫李四

#### **3、组合继承**

重点圈起来：组合继承（原型链继承+构造函数继承）

用Parent.call()也用

Parent函数中如this.age=12,结果age=12,父类没有取自己的，自己没有undefined

// 父

function Parent(age) {

    this.name = ['Lee', 'Jack', 'Hello']

    this.age = age;

}

Parent.prototype.run = function () {

    return this.name + this.age;

};

// 子

function A(age) {

    Parent.call(this, age); //对象冒充 子类A中的age指向Parent里的

}

A.prototype = new Parent(); //原型链继承

var a = new A(100);   //Parent函数中如this.age=12,结果age=12,父类没有取自己的，自己没有undefined

console.log(a.run());//结果：Lee,Jack,Hello100

#### **4、寄生组合继承**

重点圈起来: 将父类原型对象直接赋值给一个空属性的构造函数，再将空属性的构造函数实例赋值给子类原型对象,其根本是为了解决父实例继承的出现的两次构造。

**<https://blog.csdn.net/qq_42926373/article/details/83149347>**

**https://www.cnblogs.com/luoguixin/p/6195902.html**

1. **a标签的几种状态？**

a:link 普通的、未被访问的链接

a:visited 用户已访问的链接

a:hover 鼠标指针位于链接的上方

a:active 链接被点击的时刻

7、怎么理解盒模型？

标准模型：最终元素的width=content+padding+border （（默认是content-box）

content的width是100px，padding:20px;border:10px,最后元素width是160px

IE 盒子模型：**border-box**  ：元素的width=width padding和border里原本宽里边缩

1. 怎么让div的红色背景#ff0000半透明，但div里文字正常

background: rgba(255, 0, 0, 0.2)

1. let和var的区别

var是函数作用域，在函数内声明了var，整个函数内都有效,for里定义var在循环外也可以访问，有声明提前，赋值不提前，能重复定义

而let是块作用域，先定义后使用，不能重复定义声明，赋值可以，没声明提前

10、运行结果：2

alert(1 && 2)

因为js的逻辑运算符并不做转换，它是把为真的那个值本身给你，而不是转成boolean，这个跟绝大多数语言都不一样（当然，其他语言本身&&也不允许boolean以外的值参与运算）

1 && 2 => 2  
false || 'abc' => 'abc'

12、this的指向——运行结果：11

window.val = 1;

var json = {

    val: 10,

    dbl: function () {

        val \* 2;  //局部优先val是1,this.val是10

        console.log(this.val)

    }

}

json.dbl();

console.log(json.val + val)//11

13、面向对象-new时的初始值——最重要是理解js中变量在原型链中查找的顺序，

new C1().name：由于这里没有参数，默认被赋值成了undefined，所以到了if这里就进不去了，因此在C1本地属性中找不到name这个属性，只能傻逼兮兮的往外找了，又因为C1.prototype.name = "Tom"的

存在，在prototype中找到了name属性，所以最后打印出来的答案是"Tom"

new C2().name： 由于这次还是没有参数，同样默认被赋值成了undefined，于是本地属性name被赋值成了undefined。于是在查找的时候一下子就查到了name的值为undefined, 因此C2.prototype.name = "Tom"并没有什么卵用，最终答案为undefined

new C3().name：同样是没有参数，undefined作为参数进来以后情况变成了这样：this.name = undefined || "John"，然后结果很明显了本地属性name被赋值成"John"。接着是从内往外查找，一下子就锁定了本地属性name，此时的值为"John"。因此C3.prototype.name = "John"同样没有什么用

function C1(name) {

    if (name) {

        this.name = name;

    }

}

function C2(name) {

    this.name = name;

    console.log(this.name)

}

function C3(name) {

    this.name = name || 'john'//undefined||'john'->john

}

C1.prototype.name = 'Tom';

C2.prototype.name = 'Tom';

C3.prototype.name = 'Tom';

// 结果：Tomundefinedjohn

console.log((new C1().name) + (new C2().name) + (new C3().name));

总结：

如果原型链上有相同的方法。那么会优先找本地方法，找到并执行，原型链上的方法就不执行了。  
同样的，属性的查找也是这么个顺序。

function Foo() {

    this.say = function () {

        console.log('本地方法');

    }

}

Foo.prototype.say = function () {

    console.log('prototype方法');

}

new Foo().say();//结果：本地方法

4：考点 自执行函数,形参与实参,默认参数是5,所以test传值2根本就没接收,所以答案是10

var test = function (i) {

    return function () {

        console.log(i \* 2)//10

    }(2)

}

test(5)

问：

1. **介绍下MVVM？**

MVVM是Model-View-ViewModel的简写。即模型-视图-视图模型。

**模型**指的是后端传递的数据。

**视图**指的是所看到的页面。

**视图模型是**mvvm模式的核心，它是连接view和model的桥梁。不需要查找DOM直接更新View。

它有两个数据传递方向：

一是将模型转化成视图，即将后端传递的数据转化成所看到的页面。实现的方式是：数据绑定。

二是将视图转化成模型，即将所看到的页面转化成后端的数据。实现的方式是：DOM 事件监听。以上两个方向都实现的，我们称之为数据的双向绑定。

总结：在MVVM的框架下视图和模型是不能直接通信的。它们通过ViewModel来通信，ViewModel通常要实现一个observer观察者，当数据发生变化，ViewModel能够监听到数据的这种变化，然后通知到对应的视图做自动更新，而当用户操作视图，ViewModel也能监听到视图的变化，然后通知数据做改动，这实际上就实现了数据的双向绑定。并且MVVM中的View 和 ViewModel可以互相通信

### **MVVM的实现方式**

更新数据方式通常做法是 vm.set('property', value)。

Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

# **2、vue中双向数据绑定(v-model)的原理**

本质是value和v-on的结合体,就是绑定他的value,通过v-on触发实现双向绑定，依赖于Object.defineProperty(),通过这个函数可以监听到get,set事件****，****然后用他的set,get方法来通知订阅者，触发update方法，从而实现更新视图

**3、vue中使用sass的配置什么**

运行如下命令

npm install --save-dev sass-loader

npm install --save-dev node-sass

build文件夹下的webpack.base.conf.js添加如下代码：

{

test: /\.sass$/,

loaders: ['style', 'css', 'sass']

}

修改style：<style lang="scss">

1. axios 封装

ajax封装：

var methods = {

    //全站ajax请求状态处理

    ajax: function (url, data, successCallback, errorCallBack) {

        $.ajax({

            "type": "post",

            "url": AJAXURL + url,

            "async": true,

            "data": data,

            "success": function (res) {

                //返回状态处理

                if (res.code == 200) {

                    if (successCallback) {

                        successCallback(res);

                    }

                } else if (res.code == 406) {

                    //跳转至登录

                    methods.toLogin();

                } else {

                    if (errorCallBack) {

                        errorCallBack(res)

                    } else {

                        methods.prompt(res.msg);

                    }

                }

            }

        });

    },

}

axios全局封装

Vue.prototype.$axios = function (config) {

    return new Promise((resolve, reject) => {

        axios({

            method: config.method ? config.method : "post",

            url: config.url ? config.url : "",

            data: config.data ? config.data : ""

        }).then(res => {

            console.log(config.data)

            resolve(res.data);

        }).catch(error => {

            // 请求失败，

            reject(error.data);

        });

    })

}

var vm = new Vue({

    el: '#app',

    data: {},

    created() {

        this.$axios({

            url: 'https://mall.faw-vw.com/Dealer/Index/shop',

            data: {

                dealer\_id: 969

            }

        }).then(res => {

        });

    }

})

面试2：

线程和异步方法的区别 （异步应该去跟同步比较才对。）

## 单线****单线程与多线程的区别****

单线程：只有一个线程，代码顺序执行，容易出现代码阻塞（页面假死）

多线程程序：有多个线程，线程间独立运行，能有效地避免代码阻塞，并且提高程序的运行性能

promiss内部原理

async await

https是怎么进行加密的

Mvvm

个人理解性能优化

# [vue的diff算法原理](https://www.cnblogs.com/wangtong111/p/11198393.html)

diff算法的本质是**找出两个对象之间的差异**

diff算法的核心是**子节点数组对比**，思路是通过**首尾两端对比**

渲染真实DOM的开销很大，有时候我们修改了某个数据，直接渲染到真实dom上会引起整个dom树的重绘和重排。我们希望只更新我们修改的那一小块dom，而不是整个dom，diff算法就帮我们实现了这点。  
diff算法的本质就是：**找出两个对象之间的差异，目的是尽可能做到节点复用。**  
此处说到的对象，指的其实就是vue中的**virtual dom（虚拟dom树）**，即**使用js对象来表示页面中的dom结构**。

key的作用主要是 ：决定节点是否可以复用、建立key-index的索引，主要是替代遍历，提升性能。

组件化理解

### **快速排序原理**

https://www.cnblogs.com/chenhuichao/p/13528620.html