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

forEach去重

let arr1 = [3, 0, 2, 3, 2, 1];

let arr2 = [];

arr1.forEach(function (self, index, arr) {

      arr.indexOf(self) === index ? arr2.push(self) : null;

});

console.log(arr2);// [3, 0, 2, 1]

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;display:table;}

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://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

1. 怎么理解盒模型？（这是整理过的用这个）

Box-sizing:content-box/border-box

标准模型：最终元素的大小为=设置的width+padding+border ,width=content（（默认是content-box）

如:content的width是100px，padding:20px;border:10px,最后元素大小=100+60是160px

IE 盒子模型：**border-box**  ：最后元素大小=width， padding和border在width里边缩

最后元素大小=width = content+padding+border,content<=width

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**

**比less多配置了node-sassss**

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>

张：

Keep-alive 几个钩子函数

深拷贝和浅拷贝

ForEach map filter区别

如何判断 是不是数组

如何让原生访问你定义的方法app

红任务微任务

函数可以提升吗

张2

**1、Vuex属性**

state => 基本数据

getters => 从基本数据派生的数据

mutations => 提交更改数据的方法，同步！

actions => 像一个装饰器，包裹mutations，使之可以异步。

modules => 模块化Vuex

**2、Const：**用于声明常量，声明后不可修改，没有声明提前，不能重复声明同一变量，声明完马上赋值。

**3、在jquery中想要找到所有元素的同辈元素**siblings()

**4、http请求出现“请求的对应资源禁止被访问”的问题**，请问返回状态码是（403）400 语法错误，401 请求要求身份认证，403拒绝执行此请求，404 找不到资

5、向上取整 Math.ceil(25.5) 26

四舍五入 Math.round(25.4） 25

向下取整 Math.floor(25.9) 25

**6、在vue中，第一次加载时会触发哪几个钩子**：beforeCreate, created, beforeMount, mounted 。

附：vue生命周期：

beforeCreate、created 创建 初始化数据事件、beforeMount、mounted 载入 DOM渲染完成、beforeUpdate、updated 更新、beforeDestroy、destroyed 销毁

**7、vue-router有哪些组件：**

<router-link :to='' class='active-class'> //路由声明式跳转 ，active-class是标签被点击时的样式

<router-view> //渲染路由的容器

<keep-alive> //缓存组件

**8、关于vue的正确说法：**

A)V-show通过修改元素不display css属性让其显示或隐藏 正

B)Vue.js双向绑定是采用数据劫持结合发布-订阅的方式，通过Object，defineProperty()来劫持各个属性的setter,getter,在数据变动时发布消息给订阅者，触发相应的监听回调。 正

C)<style scoped> 让css只在当前组件起作用 局部 正

D)v-bind 作用是动态地绑定一个或多个特性，或一个组件prop到表达式 （改成正确语法不是v-model）

**v-model** 一个组件上的v-model默认会利用名为value的prop和名为input的事件 正

**9、关于vuex说法正确的**：

A)getters可以对state进行计算操作，它就是Store的计算属性 正

B)Vuex的mutation特性可以直接变更state里的数据（状态），是同步操作 正

C)state里面存放的数据是非响应式的，Vue组件从store中读取数据。若是store中的数据发生改变，依赖这个数据的组件不会发生更新 错

D)提交mutation是更改状态的唯一方法，并且这个过程是异步的（错，在dispatch之前可以this.$store,dispatch()方法更改，strict：true严格下也能改。mutation同步）

**10、关于webpack说法：**

#### A)使用Happypack 实现多线程加速编译 正（这是提高webpack的构建速度方式，要注意的第一点是，它对file-loader和url-loader支持不好，所以这两个loader就不需要换成happypack了，其他loader可以类似地换一下）

B)使用html-webpack-plugin为html文件中引入的外部资源，可以生成创建html入口文件 正

#### C)通过externals配置来提取常用库 正（这是提高webpack的构建速度方式）

D)Delete-webpack-plugin：打包器添理源目录文件，在webpack打包器清理dist目录 错（改说的是clean-webpack-plugin）

**11、v-on可以监听多个方法**  （正）

如：<input type="text" :value="name" @input="onInput" @focus="onFocus" @blur="onBlur" />

**12、在vuex中**，Action提交的是mutation，而不是直接变更状态，Action可以包含任意异步操作 （正）**13、jquery的load()方法**中data参数是必须的（错）

$(*selector*).load(*url,data,function(response,status,xhr*)) ，url必须，其他可选

**14、**<router-link/>设置replace属性的话，当点击时，导航后不会留下history记录（正）

设置 replace 属性的话，当点击时，会调用 router.replace() 而不是 router.push()，于是导航后不会留下 history 记录。：<router-link :to="{ path: '/abc'}" replace></router-link>

# 15、webpack的babel-loader是将ES5 转成ES6(错，改是配合babel 将es6转成es5)

**16、let和var区别**

(1)var 可重复声明，不可限制修改，函数级

(2)let是块级作用域没有声明提前、先定义在使用，块级作用域,不能重复定义变量，可以赋值

(3)const 常量 定义完变量，必须有值，不能后赋值，不能修改，块级作用域, 也没有声明提前

**17、ES6新特性及它们作用**

A)类 继承 ：(class,constructor,extends,super)让JavaScript的面向对象编程变得更加简单和易于理解

B)模块化:主要由 export 和 import 组成,每一个模块都有自己单独的作用域，模块之间的相互调用关系是通过 export 来规定模块对外暴露的接口，通过import来引用其它模块提供的接口

C)箭头函数=>function简写,箭头函数与包围它的代码共享同一个this,能帮你很好的解决this的指向问题.借助=>，就不需要诸如var self = this;或var that = this这种模式了

D)模板字符串:使得字符串的拼接更加的简洁、直观。`${变量}`完成字符串的拼接,

E)解构赋值:javaScript的一种表达式，可以方便的从数组或者对象中快速提取值赋给定义的变量。

F)延展操作符(展开操作符):...可以在函数调用/数组构造时, 将数组表达式或者string在语法层面展开；

函数调用： myFunction(...iterableObj);

数组构造或字符串：[...iterableObj, '4', ...'hello', 6];

构造对象时,进行克隆或者属性拷贝：let objClone = { ...obj };

1. 对象属性简写:在ES6中允许我们在设置一个对象的属性的时候不指定属性名。

不用es6:const student = { name:name, age:age,city:city};

用es6:const student = {name, age,city};

1. Promise:Promise 是异步编程的一种解决方案，比传统的解决方案callback更加的优雅,ES6 将其写进了语言标准，统一了用法，原生提供了Promise对象。

不使用ES6，嵌套两个setTimeout回调函数：hello 1s后在hi，很慢

setTimeout(function () {

     console.log('Hello'); // 1秒后输出"Hello"

     setTimeout(function () {

        console.log('Hi');  // 2秒后输出"Hi" }, 1000);

     }, 1000)

})

使用ES6,先hello在hi，很快

用两个then来进行异步编程串行化，避免了回调地狱：

var waitSecond = new Promise(function (resolve, reject) {

      setTimeout(resolve, 1000);

    });

    waitSecond.then(function () {

      console.log("Hello"); // 1秒后输出"Hello"

      return waitSecond;

    }).then(function () {

      console.log("Hi"); // 2秒后输出"Hi"

    });

1. Let与Const

在之前JS是没有块级作用域的，const与let填补了这方便的空白，const与let都是块级作用域。替代闭包，

**18、vue-router有哪几种导航钩子（导航守卫）？**

守卫作用：过来新的地址要先经过守卫同意了才能进去 也就是next()

分类：全局和局部

路由钩子函数有三种：

  1：全局钩子： beforeEach、afterEach

  2：单个路由里面的钩子：beforeEnter、beforeLeave

  3：局部组件路由：beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave

**19、vue中事件修饰符主要有哪些，作用？**

(1).stop—阻止冒泡 @click.stop

(2).prevent—阻止默认事件:

按键盘这个行为要干掉它，@keydown.prevent

如表单提交按钮submit或在页面上点右键，自动出现下拉菜单

(3).self—只接收自身的事件，相当于阻止事件冒泡（冒泡上来的不要）嵌套点击中父级加.self，

阻止冒泡二选一像.self放div或.stop放button上

<div @click.self=‘divFn’>

    <button @click.stop=”btnFn”></button>

</div>

总结： .self和.stop

一个事情有多种实现方法，结果一样，一是在.self外边加，一个在.stop里边加

.once: 事件只触发一次；

.passive 告诉浏览器你不想阻止事件的默认行为

.capture：捕获冒泡

加,outer inner middle ,都不加inner middle outer，加的从外向里顺序输出

<div @click.capture="outer">

      <div @click="middle">

        　　<button @click.capture="inner">点击</button>

       </div>

</div>

**20、get与post 区别，何时用post**

1）get是从服务器上获取数据，post是向服务器传送数据。

2）get是将参数数据加到URL中，用户可以看到。post是将内容放置在http请求信息体内传送，用户看不到这个过程。

3）对于get方法，服务器端是用Request.QueryString获取变量的值，对于post方法，服务器端用Request.Form获取提交的数据。

4）get传送的数据量较小，不能大于2kb。post传送的数据量较大，一般被默认为不受限制。但理论上，IIS4中最大量为80kb，IIS5中为100kb。“一个汉字字符是2个字节。1kb=1024字节， 2k就1024个字

5）get安全性非常低，post安全性较高。但是执行效率却比post方法好。

仅用于POST请求（1.无法使用缓存文件；2.向服务器发送大量数据；3.发送包含未知字符的用户输入时，post比get更稳定也更可靠）

建议使用get方法的情况：1、get方式的安全性较Post方式要差些，包含机密信息的话，建议用Post数据提交方式2、在做数据查询时，建议用Get方式；而在做数据添加、修改或删除时，建议用Post方式；

--------------------------------------------------------------------------------------------------

1、\_proto\_与prototype区别关系

Blue老师：class(类)上的叫prototype，实例上的叫\_\_proto\_\_，

看图

prototype：是构造函数(function Foo())和原型对象(Foo.prototype)之间的一个属性

【每个函数都有属性prototype，该属性所存储的就是原型对象保存共享属性和方法,来实现扩展和继承】

\_\_proto\_\_：是链式的，实例对象(f1)指向原型对象(Foo.prototype)；但原型对象(Foo.prototype)本身又是另一个原型(Object.prototype)的实例对象,(Object.prototype)在往上指\_\_proto\_\_就是null。这个链就是原型链。这里\_proto\_就是原型链的重要部分

\_\_proto\_\_是私有属性在javascript上没法通过这个属性直接访问某个属性指向的对象，

【对象必然有\_proto\_属性new出来的，但不一定有prototype】

例:

var f1= new Foo(); foo.name

如果foo本身没有name属性会通过私有\_proto\_向上找【看图】->Foo.prototype(原型)->Object.prototype到头了还没有返回undefined

有两个特例

Function.\_\_proto\_\_\_ === Function.prototype 返回true,浏览器让它自己构造了自己.

Object.prototype.\_\_proto\_\_ === null 返回true,万物需要有终点,浏览器让它指向了null.

每个实例对象的\_\_proto\_\_指向其构造函数的prototype,prototype中储存着公用方法和属性.

也就是说：f1是new出来的实例有\_\_proto\_\_，Foo是构造函数也称class(类)有prototype

function Foo() {

    this.name = 'yjui';

}

var f1 = new Foo();

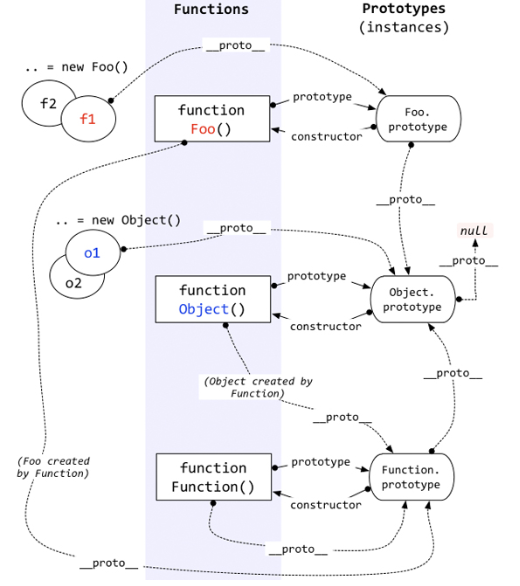
console.log(Foo.prototype) //{constructor: ƒ}

console.log(f1.\_\_proto\_\_) //{constructor: ƒ}

console.log(f1.\_\_proto\_\_ === Foo.prototype) //true

console.log(Foo.prototype.\_\_proto\_\_ === Object.prototype) //true

console.log(Object.prototype.\_\_proto\_\_ === null) //true



1. vuex相关与localtorage区别

Vuex实现了一个单向数据流actions->mutations->state

Vue Components：是vue组件，只能读State不是写，写改写唯一办法是通过Dispatch间接调用this.$store.dispatch(调用哪个action，值)一般在methods中调，使用{{$store.state}}

Actions：异步，mutations不能直接调用，想调用mutations要先调用action通过commit调用mutations完成复杂操作[功能]，比如登录，找服务器请求数据这种操作都是异步操作通过actions

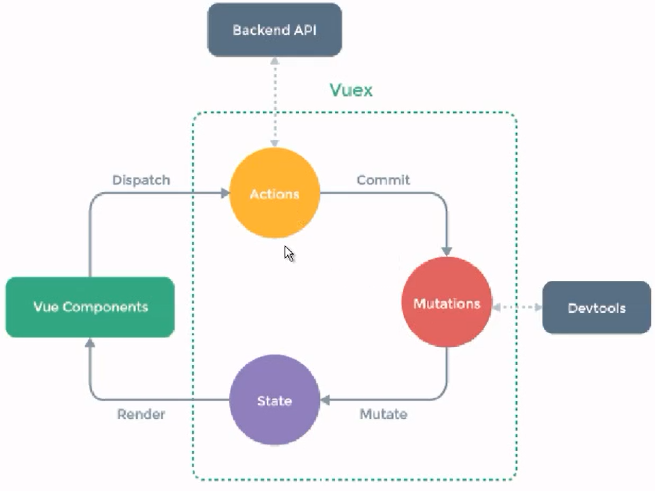
Mutations：同步，mutations专门修改state,

state：全局存储状态数据 ，【初始化的】

最后调render()函数进行服务器端渲染返回VNode给组件。dispatch和commit 与render是方法

简要介绍各模块在流程中的功能：

dispatch一个动作到action ，action做异步处理，action调用之后mutation改变state,state改变完之后组件view内存也会发生变化



vuex存储在内存，localstorage（本地存储）

应用场景:vuex用于组件之间的传值,localstorage,sessionstorage则主要用于不同页面之间的传值。

永久性：vuex当刷新页面(F5刷新,属于清除内存了)就没了，sessionstorage页面关闭后就清除掉了，localstorage不会。

vuex全局变量，刷页面就全变量就不存在了。用户信息建议存储到localStorage里面存储一份localStorage.setItem(key,value)//存储数据

3、foreach和for区别

4、数组循环几种区别

写法：

arr.forEach/filter/map/every/some/forEach/(function(currentValue,index,arr),thisValue)

forEach():数组每个元素都执行一次回调函数即：迭代(循环) 遍历，不支持break跳出循环用try/catch/every/some代替,返undefined

foreach遍历数组的话，使用break不能中断循环，使用return也不能返回到外层函数

filter() ：筛选数组，创建一个新的数组，返回符合条件所有元素的新数组。遇到return true不会终止迭代

map()：返回原数组处理后的新数组。不改变原数组

every()：检查每个元素是否都符合条件,回值true/false,都满足才true

some():查找数组中是否有满足条件的元素,有一个满足就返true,遇return会终止遍历

**forin与forof与Object.keys()区别**

forin与forof相同点写法：

for(let I in/of arr){}

Object.keys(obj)

For in遍历数组或者对象的属性，包各原型方法method和name属性

for of是ES6新增的循环方法

遍历对象时：forin取属性名，for of无法循环遍历对象报错

遍历数组时：for in遍历的是数组的索引（即键名），而for of遍历的是数组元素值。

自定义属性：for in可以，for of输出不来

for in遍历数组包括原型方法属性，for of遍历只是数组内元素，不包括数组原型属性

for in更适合遍历对象取键名，for of适合遍历数组取元素

Object.keys()更适合遍历对象，返回属性名组成的数组，属性名直接放数组里不用push了而forin需要push才能把键名放数组里

6、var和let区别

7、闭包：

闭包就是能够读取其他函数内部变量的函数

正常下：函数内部可以直接读取全局变量，但是在函数外部无法读取函数内部的局部变量

8、vue优势：

易于开发：上手快，简单易学，提供页面数据渲染模板引擎如v-if,v-for,v-show等，事件绑定如@click等，代码清晰明了，逻辑简单，单页面，组件复用，前后分离

高性能：Vue提出虚拟DOM概念，以及数据驱动DOM思想，减少页面渲染成本，大幅度提高性能。

生态完善：许多开发者提供了对vue支持插件如ElementUI，完善vue的生态，故使用vue进行开发遇到难题会很少。

开源社区活跃：这提供了vue未来更加强大的可能性。

$option

过滤如果不用filter怎么用js实现过滤 提到$option

9、还用了哪些框架

10、Event Loop&宏任务微任务

1、什么是Event Loop(事件循环)？

Event Loop指的是js的执行机制，js是单线程的,任务分为同步任务和异步任务，

同步任务会放执行栈中也就是主线程。异步任务会提交给异步进程来处理，异步进程处理内容如ajax(回调函数也有触发)、DOM事件、setTimeout/setInterval。它们被触发后就放到任务队列里。

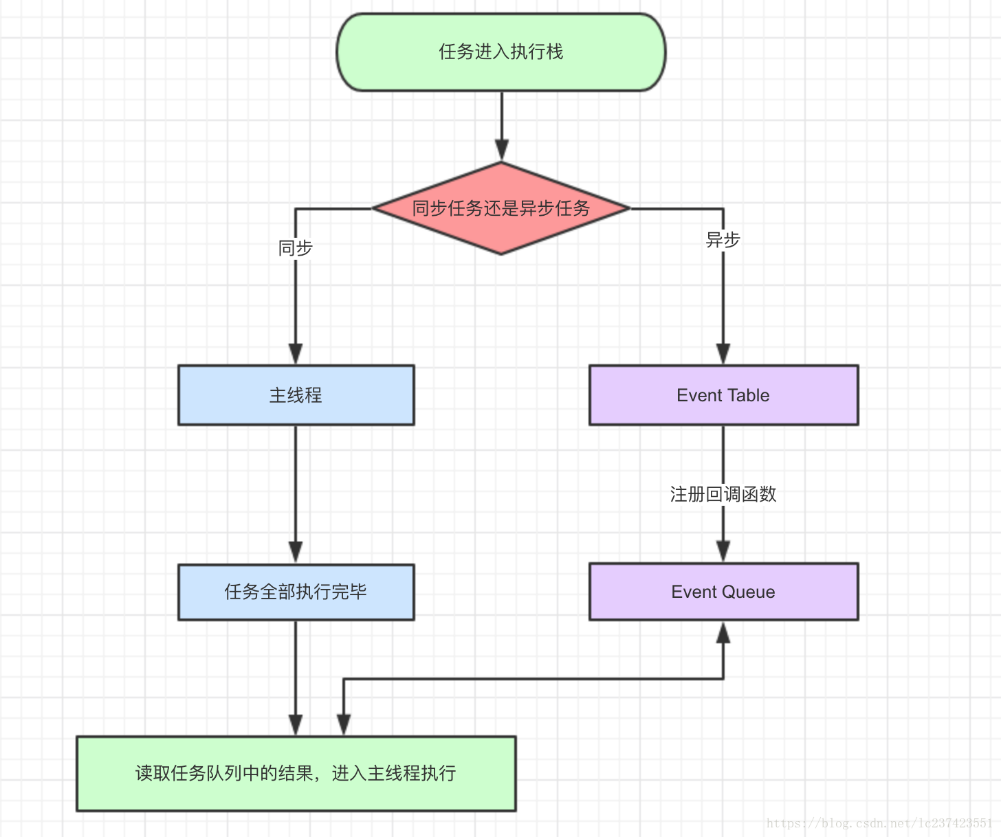
主线程的同步任务都执行完后在到任务队列里看有没有异步任务,如果有就取回来放执行栈中执行,

在回头看还有没有异步任务如有在取回在执行，这种循环的过程叫事件循环Event Loop

javascript是单线程同步非阻塞同一时间只能做一个件事，这样导致的体验差的问题。阻塞-异步-放任务队列

什么时候入任务队列中？

任务队列中放的都是异步任务。当异步函数达到触发条件时，比如定时器到时间，Ajax请求返回数据，根据异步模块类型压入到任务队列里。



**2、异步任务**

异步任务又分为宏任务和微任务、他们之间区别主要是执行顺序的不同。

微任务：Promise中的.then .catch .finally都是微任务。 但new Promise(是同步主线程的要先执行没有任何宏/微任务说法)

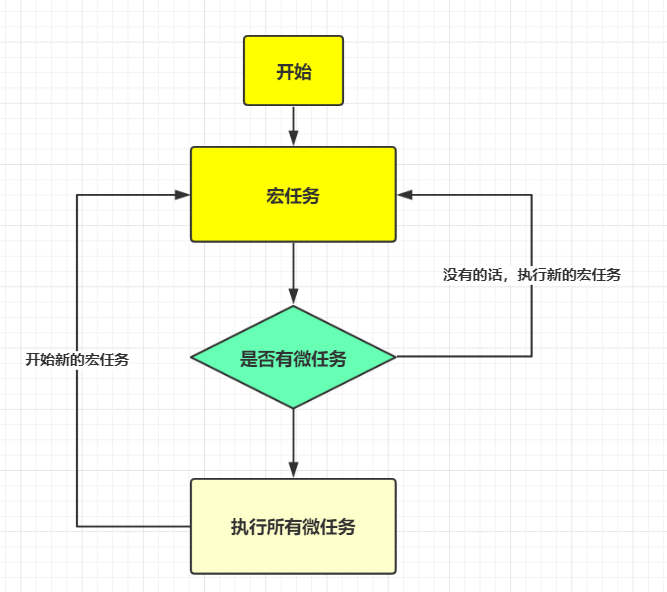
宏任务：script是一个大的宏任务，代表setTimeout、setInterval【是浏览器、node发起的，包含script(整体代码)，setImmediate(node里的东西)、I/O（读写文件）】

**在当前的微任务没有执行完成时，是不会执行下一个宏任务的，微任务先于下一个宏任务**

**执行宏任务时顺便把微任务执行完**

执行顺序：

主线程(同步)>微任务>下一个宏任务



11、什么是防抖与节流

12、一个页面从输入URL到页面加载显示完成发生了什么了？

13、\_proto\_与prototype的区别关系

Blue老师：class(类)上的叫prototype，实例上的叫\_\_proto\_\_，而且\_\_proto\_\_的名字就已经告诉你不要去用它了

prototype：是构造函数(function Foo())和原型对象(Foo.prototype)之间的一个属性

【每个函数都有属性prototype，该属性所存储的就是原型对象保存共享属性和方法,来实现扩展和继承】

\_\_proto\_\_：是链式的，实例对象(f1)指向原型对象(Foo.prototype)；但原型对象(Foo.prototype)本身又是另一个原型(Object.prototype)的实例对象,(Object.prototype)在往上指\_\_proto\_\_就是null。这个链就是原型链。这里\_proto\_就是原型链的重要部分

\_\_proto\_\_是私有属性在javascript上没法通过这个属性直接访问某个属性指向的对象，

【对象必然有\_proto\_属性new出来的，但不一定有prototype】

有两个特例

Function.\_\_proto\_\_\_ === Function.prototype 返回true,浏览器让它自己构造了自己.

Object.prototype.\_\_proto\_\_ === null 返回true,万物需要有终点,浏览器让它指向了null.

每个实例对象的\_\_proto\_\_指向其构造函数的prototype,prototype中储存着公用方法和属性.

prototype：在javascript中，函数本身就是一个包含了方法和属性的对象，每个函数中都有一个prototype属性，该属性所存储的就是原型对象。原型对象用来保存共享属性和方法。可以通过原型来实现为对象扩展属性，实现继承。

14、for in 、Object.keys、Object.getOwnPrototype区别

15、MVVM理解

M-V-VM（Model-View-ViewModel）

M（Model）模型：数据保存—存放着各种数据，有的是data里的属性，有的是从后端返回的数据

V（View）视图 ：用户界面，也就是DOM。即template或html写页面部分也就是UI效果

VM（View-Model）视图模型:连接View和Model的桥梁，当数据变化时，ViewModel够监听到数据的变化（通过Data Bindings），自动更新视图，而当用户操作视图，ViewModel也能监听到视图的变化（通过DOM Listeners），然后通知数据做改动，这就实现了数据的双向绑定。

也是vue的实例对象，充当一个UI视配器的角色。也就是说view中每一个UI元素，都应该在ViewModel中找到与之相对应的属性。把data里的数据给到html页面渲染当中实现双向数据绑定。

为什么用mvvm:

统一团队的开发标准，框架层面上清晰，代码层面干净

16、用css画三角形

.up {width: 0;height: 0;border-left: 5px solid transparent;

border-right: 5px solid transparent;border-bottom: 10px solid red;}

.left {width: 0; height: 0;border-top: 5px solid transparent;border-bottom: 5px solid transparent;border-right: 10px solid red;}

17、常用的性能优化

18、继承的几种方式与区别

19、写一段原型里的代码

20、computed与watch区别

**21、$nextTick**

Blue老师:这个是等vue完成本次渲染的时候回调用的，一般用来等待渲染完成，比如说

假设你有个arr  
arr=[]

然后它是li身上的  
<li v-for="arr">

然后你比如  
arr.push(55);  
li[0].xxx  
这时候因为你数据变了之后li不可能立即渲染完成所以就错了

但这样就对  
arr.push(55);  
this.$nexttick(()=>{  
  li[0].xxx  
});