# 面试题

## Vue相关

### vue双向数据绑定

Vue2

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

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

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

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

1、在自身实例化时往属性订阅器(dep)里面添加自己  
2、自身必须有一个update()方法  
3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

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

### vue虚拟dom，diff算法

**想要理解虚拟dom首先要知道什么是虚拟dom？**

虚拟dom可以简单的用一句话概括，就是用普通的js对象来描述DOM结构，因为不是真实DOM，所以称之为虚拟DOM。

虚拟dom是相对于浏览器所渲染出来的真实dom而言的，在react，vue等技术出现之前，我们要改变页面展示的内容,只能通过遍历查询dom树的方式,找到需要修改的dom,然后修改样式行为或者结构，来达到更新视图的目的。

**为什么要用虚拟DOM来描述真实的DOM呢？**

创建真实DOM成本比较高，如果用 js对象来描述一个dom节点，成本比较低，另外我们在频繁操作dom是一种比较大的开销。所以建议用虚拟dom来描述真实dom。

**Diff算法**

diff算法就是进行虚拟节点对比，并返回一个patch对象，用来存储两个节点不同的地方，最后用patch记录的消息去局部更新Dom。

**思考**

你觉得虚拟DOM的性能一定高于常规DOM吗？答案是不一定，虚拟DOM和Diff算法的出现是为了解决由命令式编程转变为函数式编程、数据驱动后所带来的性能问题的。换句话说，直接操作DOM的性能并不会低于虚拟DOM和Diff算法，甚至还会优于。

你觉得使用了虚拟DOM就真的不操作dom元素了吗？其实不是的，只是减少用户操作dom，虚拟DOM在渲染的时候其实还是会操作dom的

### 组件通讯

**1.父传子：**

在父组件的子组件标签上绑定一个属性，挂载要传输的变量。在子组件中通过props来接受数据，props可以是数组也可以是对象，接受的数据可以直接使用 props:["属性名"] props:{属性名:数据类型}

**2.子传父：**

在父组件的子组件标签上通过绑定自定义事件，接受子组件传递过来的事件。子组件通过$emit触发父组件上的自定义事件，发送参数

**3.兄弟组件传值：**

通过main.js初始化一个全局的$bus,在发送事件的一方通过$bus.$emit(“事件名”，传递的参数信息)发送，在接收事件的一方通过$bus.$on("事件名",参数)接收传递的事件

### Vuex

**1.vuex** ：是一个专为vue.js开发的状态管理器，采用集中式存储的所有组件状态，通过vuex我们可以解决组件之间数据共享的问题，后期也方便我们管理以及维护

有五个属性分别是： state、getters、mutations、actions、module

state属性： 存放状态，例如你要存放的数据

getters： 类似于共享属性，可以通过this.$store.getters来获取存放在state里面的数据

mutations： 唯一能改变state的状态就是通过提交mutations来改变，this.$store.commit()

actions: 异步的mutations，可以通过dispatch来分发从而改变state

**2.基本使用**：我通过是在根目录下新建一个store文件夹，里面创建一个index.js文件，最后在main.js中引入，并挂载到实例上，之后那个组件中需要用到vuex就调用就行

**3.高级用法-数据持久化**

vuex里面存放的数据，页面一经刷新会丢失：

解决办法： 存放在localStorage或者sessionStorage里面，进入页面时判断是否丢失，丢失再去localStorage或者sessionStorage里面取；

在vuex中可以通过安装vuex-persistedstate 插件，进行持久化的配置就行

**4.高级用法-辅助函数（语法糖）**

1. 有那几个辅助函数(4大金刚)  
mapState,mapActions,mapMutations,mapGetters  
2. 辅助函数可以把vuex中的数据和方法映射到vue组件中。达到简化操作的目的

3. 如何使用：

Import { mapActions, mapGetters, mapMutations, mapState } from 'vuex'

### 自定义指令，自定义过滤器

**vue中的自定义指令：**

vue中除了核心功能内置的指令外，也允许注册自定义指令。有的情况下，对普通DOM元素进行底层操作，这时候就会用到自定义指令。自定义指令又分为全局的自定义指令和局部自定义指令。

**全局自定义指令**是通过Vue.directive('第一个参数是指令的名称',{第二个参数是一个对象，这个对象上有钩子函数})

Vue.directive('focus', {

// el：指令所绑定的元素，可以用来直接操作 DOM。

//binding：一个对象，包含以下 property：

inserted: function (el) { // inserted 表示被绑定元素插入父节点时调用

el.focus();

}

});

**局部自定义指令：**  
是定义在组件内部的，只能在当前组件中使用

directives: {

// 指令名称

dir1: {

inserted(el) {

// 指令中第一个参数是当前使用指令的DOM

console.log(el);

console.log(arguments);

// 对DOM进行操作

el.style.width = '200px';

el.style.height = '200px';

el.style.background = '#000';

}

},

color: { // 为元素设置指定的字体颜色

bind(el, binding) {

el.style.color = binding.value;

}

}

}

**钩子函数：**  
一个指令定义对象可以提供如下几个钩子函数 (均为可选)：

**inserted**：被绑定元素插入父节点时调用 (仅保证父节点存在，但不一定已被插入文档中)。

**bind**：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。

**update**：所在组件的 VNode 更新时调用，但是可能发生在其子 VNode 更新之前。指令的值可能发生了改变，也可能没有。但是你可以通过比较更新前后的值来忽略不必要的模板更新 (详细的钩子函数参数见下)。

**componentUpdated**：指令所在组件的 VNode 及其子 VNode 全部更新后调用。

**unbind**：只调用一次，指令与元素解绑时调用。

项目中：拖拽

**vue中自定义过滤器**

* 过滤器是对即将显示的数据做进一步的筛选处理，然后显示，过滤器并没有改变原来的数据，只是在原数据的基础上产生新的数据
* 过滤器分为全局过滤器和局部过滤器

**全局过滤器**  
全局过滤器是通过Vue.filter()来定义的，定义好后，它在所有组件中都可以使用。

// global-filter是过滤器名称

// 函数第一个参数是需要过滤的数据.

// 函数第二个参数是给过滤器传递的值.

Vue.filter('global-filter',(val,...args)=>{

console.log(`需要过滤的数据是:${val}`)

return val + ' 过滤器追加的数据'

})

**局部过滤器**

局部过滤器,定义在组件内部 filters 属性上.它只能在此组件内部使用.

过滤器的使用方式是，在双花括号或v-bind中通过一个管道符来拼接,

项目中使用过滤器：时间，价钱

### vue-router(路由原理？路由守卫？传参)

**路由原理**

1. 路由就是用来解析URL以及调用对应的控制器，并返回从视图对象中提取好的网页代码给web服务器,最终返回给客户端。

hash模式：在浏览器中符号的“#”，以及#后面的字符称之为hash，用window.location.hash读取；

特点：

hash虽然在URL中，但不被包括在HTTP请求中；用来指导浏览器动作，对服务端安全无用，

hash不会重加载页面。

hash 模式下，仅 hash 符号之前的内容会被包含在请求中，如 http://www.xxx.com，因此对于后端来说，即使没有做到对路由的全覆盖，也不会返回 404 错误。

history模式：history采用HTML5的新特性；且提供了两个新方法：pushState（），

replaceState（）可以对浏览器历史记录栈进行修改，以及popState事件的监听到状态变更。

特点:

history 模式下，前端的 URL 必须和实际向后端发起请求的 URL 一致，如 地址后加上/items/id。后端如果缺少对 /items/id 的路由处理，将返回 404 错误。

**路由传参：**

三种：

分别是query，params，动态路由传参

接收：

通过query方式传递过来的参数一般是通过this.$route.query接收

通过params方式传递过来的参数一般是通过this.$route.params接收

通过动态路由传参方式传递过来的参数一般是通过this.$route.params接收

query使用path和name传参跳转都可以，而params只能使用name传参跳转。

传参跳转页面时，query不需要再路由上配参数就能在新的页面获取到参数，params也可以不用配，但是params不在路由配参数的话，当用户刷新当前页面的时候，参数就会消失。

也就是说使用params不在路由配参数跳转，只有第一次进入页面参数有效，刷新页面参数就会消失。

**路由守卫：**

2.路由守卫使用的方式有几种？ 全局的 单个路由独享的 组件级的

3.vue-router全局有三个守卫：

router.beforeEach 全局前置守卫 进入路由之前

router.beforeResolve 全局解析守卫(2.5.0+) 在beforeRouteEnter调用之后调用 router.afterEach 全局后置钩子 进入路由之后

组件内的守卫：

beforeRouteEnter

beforeRouteUpdata(2.2新增)

beforeRouteLeave

3. 路由守卫钩子函数里面的三个参数分别是什么？

**to,from,next 这三个参数：**

to和from是将要进入和将要离开的路由对象,路由对象指的是平时通过this.$route获取到的路由对象。

next:Function 这个参数是个函数，且必须调用，否则不能进入路由(页面空白)。

next() 进入该路由。

next(false): 取消进入路由，url地址重置为from路由地址(也就是将要离开的路由地址)。 next 跳转新路由，当前的导航被中断，重新开始一个新的导航。

### 生命周期(那几个？每一个生命周期的特点,可以做什么)

生命周期让我们在控制整个vue时更容易形成更好的逻辑，可以分为三个阶段，挂载阶段，更新阶段，销毁阶段

分别有：

创建前：beforeCreate() 只有一些实例本身的事件和生命周期函数

创建后：Created() 是最早使用data和methods中数据的钩子函数

挂载前：beforeMount() 指令已经解析完毕，内存中已经生成dom树

挂载后：Mounted() dom渲染完毕页面和内存的数据已经同步

更新前：beforeUptate() 当data的数据发生改变会执行这个钩子，内存中的数据是新的，页面是旧的

更新后：Updated() 内存和页面都是新的

销毁前：beforeDestroy() 即将销毁data和methods中的数据此时还是可以使用的，可以做一些释放内存的操作

销毁后：Destroyed() 已经销毁完毕

Vue3.0中的生命周期做了一些改动：

beforeCreate -> setup() 开始创建组件之前，创建的是data和method

created -> setup()

beforeMount -> onBeforeMount 组件挂载到节点上之前执行的函数。

mounted -> onMounted 组件挂载完成后执行的函数

beforeUpdate -> onBeforeUpdate 组件更新之前执行的函数。

updated -> onUpdated 组件更新完成之后执行的函数。

beforeDestroy -> onBeforeUnmount 组件挂载到节点上之前执行的函数。

destroyed -> onUnmounted 组件卸载之前执行的函数。

### 自定义组件

在vue中开发，都是用的组件化的思想开发的，一般在都会在搭建项目的时候，都会先建立组件的模板，把架子搭起来。也就是在组件中定义好<template>视图层，<script>逻辑层，<style>css样式层。

在vue中使用组件封装的方式可以使我们的开发效率提高，能够把页面抽象成相对独立的模块。

我一般在创建项目的时候，都会通过创建一个views目录和一个commen目录和一个feature目录，views目录中放页面级的组件，commen中放公共组件（如：head（公共头组件），foot（公共底部组件）等），feature目录内放功能组件（如：swiper(轮播功能组件)，tabbar(切换功能组件)、list(上拉加载更多功能组件)）。组件封装是通过定义一个组件，然后定义好props里面的数据，实现其他组件需要的逻辑化代码后，也就是封装好了，然后直接调用就可以。调用是通过在需要封装组件的组件中，通过import导入，component注册好名称，最后挂载到父组件中的template即可。

组件封装的方式解决了我们传统项目，开发效率低，难以维护，复用性低等问题。

### 常见的指令，修饰符

**常用指令**

在vue中提供了一些对于页面 + 数据的更为方便的输出,这些操作就叫做指令，指令中封装了一些DOM行为, 结合属性作为一个暗号, 暗号有对应的值,根据不同的值，框架会进行相关DOM操作的绑定  
vue中的指令有很多，我们平时做项目常用的有：

v-if：是动态的向DOM树中添加或者删除元素；

v-else是搭配v-if使用的，它必须紧跟在v-if或者v-else-if后面，否则不起作用

v-show：是通过标签的CSS样式display的值是不是none，控制显示隐藏

区别：  
1、当条件为真的时候 没有区别 当条件为假的时候 v-if通过创建或删除DOM节点来实现元素的显示隐藏，v-show通过css中的display属性来控制  
2、v-if更适合数据的筛选和初始渲染 v-show更适合元素的切换

v-for： v-for是根据遍历数据来进行渲染，要配合key使用。要注意的是当v-for和v-if同处于一个节点时，v-for的优先级比v-if更高。这意味着v-if将运行在每个v-for循环中

v-on：用来绑定一个事件或者方法，简写方式是@click=""

v-bind： v-bind用来动态的绑定一个或者多个属性。没有参数时，可以绑定到一个包含键值的对象。常用于动态绑定class和style。以及href等。简写的方式是“:属性名=""”一个冒号

v-model 只能适用于在表单元素上，可以实现数据双向绑定 ，

数据双向绑定实现的原理：

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

**主要分为四步**：

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

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

第三步： Watcher订阅者是Observer和Compile之间通信的桥梁。

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

**修饰符：**

在Vue中，事件修饰符处理了许多DOM事件的细节，让我们不再需要花大量的时间去处理这些烦恼的事情，而能有更多的精力专注于程序的逻辑处理。在Vue中事件修饰符常用的主要有：

* .stop :阻止事件冒泡：由内而外，通俗的将就是阻止事件将向上级DOM元素传递
* .capture :事件捕获：由外而内，在捕获阶段，事件从window开始，之后是document对象，一直到触发事件的元素。
* .self :当事件作用在元素本身时才会触发
* .once :只触发一次
* .prevent: 阻止默认事件
* ·passive:告诉浏览器你不想阻止事件的默认行为
* ·trim:自动过滤用户输入的首尾空格

语法：@事件名.修饰符=“方法名”

### Vue2和vue3的区别

1. 双向数据绑定原理发生了改变，使用proxy替换Object.defineProerty,使用Proxy的优势：

* 可直接监听数组类型的数据变化
* 监听的目标为对象本身，不需要像Object.defineProperty一样遍历每个属性，有一定的性能提升
* 可直接实现对象属性的新增/删除

1. 默认使用懒加载

在2.x版本里。不管数据多大，都会在一开始就为其创建观察者，在数据很大时，就会造成性能的问题。在3.x中，只会对渲染出来的数据创建观察者，而且3.x的观察者更高效。

1. 3.0新加入了TypeScript以及PWA支持
2. 重构Virtual DOM

* 模板编译时的优化，将一些静态节点编译成常量
* Slot优化，将slot编译为lazy函数，将slot的渲染的决定权交给子组织

1. 生命周期有了一定的区别

Vue2--------------vue3

beforeCreate -> setup() 开始创建组件之前，创建的是data和method

created -> setup()

beforeMount -> onBeforeMount 组件挂载到节点上之前执行的函数。

mounted -> onMounted 组件挂载完成后执行的函数

beforeUpdate -> onBeforeUpdate 组件更新之前执行的函数。

updated -> onUpdated 组件更新完成之后执行的函数。

beforeDestroy -> onBeforeUnmount 组件挂载到节点上之前执行的函数。

destroyed -> onUnmounted 组件卸载之前执行的函数。

activated -> onActivated 组件卸载完成后执行的函数

deactivated -> onDeactivated

### Keep-alive

keep-alive 是 Vue 的内置组件，当它包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们。keep-alive 是一个抽象组件：它自身不会渲染成一个 DOM 元素，也不会出现在父组件链中。

在组件切换过程中 把切换出去的组件保留在内存中，防止重复渲染DOM，减少加载时间及性能消耗，提高用户体验性

被包含在 keep-alive 中创建的组件，会多出两个生命周期的钩子: activated（组件激活时使用） 与 deactivated（组价离开时调用）

如果需要缓存整个项目，直接在app.vue中用keep-alive包裹router-view即可。要缓存部分页面，需要在路由地址配置中，在meta属性中添加一个状态，在app.vue中判断一下包裹的router-view即可

例如有一个商品页面和一个详情页面，这样在两个页面切换的时候就可以用到keep-alive,在切换到详情的时候，把状态保留在内存中，而不是销毁,从而提高一个性能的优化

### 多环境变量

首先是通过在根目录下创建.env.\*（配置文件）文件，development 本地开发环境配置、staging 测试环境配置、production 正式环境配置。因为我在创建的文件中并没有定义很多变量，只定义了基础的env，所以需要在src目录下创建一个config文件夹，创建对应的环境变量文件，用来管理不同的环境。在config中创建对应的文件是为了后期修改起来方便，不需要重启项目，符合开发习惯。之后就是根据需要的环境，在封装的axios中通过解构赋值的方式导入，放在baseURL中就可以使用。

### 对axios封装(url统一管理、axios请求拦截、响应拦截、函数封装)

首先要安装axios，一般我会在项目的src目录中，新建一个network文件夹,作为我们的网络请求模块，然后在里面新建一个http.js和一个api.js文件和一个reques.js。http.js文件用来封装我们的axios，api.js用来统一管理我们的接口url,

在request.js中添加请求拦截和响应拦截。在请求拦截中，会给请求头添加token字段，还有loading动画的开启。在响应拦截中，可以做一些loading动画的关闭，还有可以根据后端返回的状态码，做一些检验token是否有效或者过期的操作。接着就是做一些axios进行的api接口的封装，这里我用到了async，await封装请求接口函数，这样可以将异步操作同步化操作，代码更加友好，避免回调地域的出现。

### Element-ui和vant-ui按需引入

首先安装按需引入的插件，在babel.config.js中添加按需引入的配置，创建一个plugin文件夹，定义一个js文件用来存放按需引入的代码，之后在建好的js文件中首先导入vue，再导入需要的vant-ui插件，通过vue.use()全局注入。

**修改样式可以用样式穿透 /deep/**

### Sass配置

安装node-sass sass-loader npm install node-sass sass-loader --save

使用lang=”scss”

### Rem、vm/vh设置

1. 通过安装cnpm install lib-flexible postcss-pxtorem --save-dev
2. Main.js 中导入插件
3. **根目录创建.postcssrc.js配置**

### Webpack配置(配置跨域、路径别名、打包分析、cdn映入、去掉console.log、单独打包第三方模块、ie兼容、eslint规范、图片压缩)

### Slot插槽

插槽就是父组件往子组件中插入一些内容。

有三种方式，默认插槽，具名插槽，作用域插槽

1. 默认插槽就是把父组件中的数据，显示在子组件中，子组件通过一个slot插槽标签显示父组件中的数据
2. 具名插槽是在父组件中通过slot属性，给插槽命名，在子组件中通过slot标签，根据定义好的名字填充到对应的位置。
3. 作用域插槽是带数据的插槽，子组件提供给父组件的参数，父组件根据子组件传过来的插槽数据来进行不同的展现和填充内容。在标签中通过slot-scope来接受数据。

### 为什么v-for使用key

key是给每一个vnode的唯一id，也是diff的一种优化策略，可以根据key，更准确， 更快的找到对应的vnode节点

### 为什么data是一个函数

1.根实例对象data可以是对象也可以是函数（根实例是单例），不会产生数据污染情况

如果不是一个函数，每个组件实例的data都是同一个引用数据，当该组件作为公共组件共享使用，一个地方的data更改，所有的data一起改变，如果data是一个函数，每个实例的data都在闭包中，就不会各自影响了。

## Css相关

### 一、什么是盒模型

1.在我们HTML页面中，每一个元素都可以被看作一个盒子，而这个盒子由：内容区（content）、填充区（padding）、边框区（border）、外边界区（margin）四部分组成。

2.盒子模型有标准盒子模型和怪异盒子模型

标准模式下: 一个块的总宽度(页面中占的宽度)= width + margin(左右) + padding(左右) + border(左右)

怪异模式下: 一个块的总宽度= width + margin(左右)（即width已经包含了padding和border值）（IE浏览器）

3. 标准和怪异模型的转换

box-sizing:content-box; 将采用标准模式的盒子模型标准

box-sizing:border-box; 将采用怪异模式的盒子模型标准

box-sizing:inherit; 规定应从父元素继承 box-sizing 属性的值。

### 二、BFC

BFC就是“块级格式化上下文”的意思，简单来说，BFC 就是一种属性，这种属性会影响着元素的定位以及与其兄弟元素之间的相互作用。BFC是一块用来独立的布局环境，保护其中内部元素不受外部影响，也不影响外部。

**1.BFC触发的几种方式：**

1. 浮动元素，float 除 none 以外的值；
2. 绝对定位元素，position（absolute，fixed）；
3. display 为以下其中之一的值 inline-block，table-cell，table-caption；
4. overflow 除了 visible 以外的值（hidden，auto）

**2.BEC的应用**

1. 可以用来自适应布局

利用BFC的这个原理可以实现两栏布局，左边定宽，右边自适应。不会相互影响，哪怕高度不相等。

给左边盒子加浮动，右边盒子加overflow:hidden;变成BFC，就可以消除外部左边盒子因浮动对他的影响

1. 可以清除浮动

一个父元素中的子元素，设置浮动时，父元素没有设置高度，这时子元素脱离文档流，父元素感知不到子元素的高度，造成父元素的塌陷。  
这时候给父元素添加overflow:hidden / auto,变成BFC就可以解决这种问题。

1. 解决垂直边距重叠

**有两种边距重叠的情况：**

1.父子关系的边距重叠

父子关系，如果子元素设置了外边距，在没有把父元素变成BFC的情况下，父元素也会产生外边距。

解决办法是给父元素添加一个 overflow:hidden，这样父元素就变为BFC，不会随子元素产生外边距

2.同级兄弟关系的重叠

同级元素在垂直方向上外边距会出现重叠现象，最后外边距的大小取两者绝对值大的那个

可通过添加一个空元素或伪类元素，设置overflow:hidden;解决

### 三、清除浮动

**1.为什么要清除浮动？**

清除浮动主要是为了解决，父元素因为子级元素浮动引起的内部高度为0的问题

1.比如,我给父盒子设置一个boder,内部放两个盒子一个big、一个small，未给big和small设置浮动，则他们会默认撑开父盒子

2.如果给这两个盒子都加上了float属性的时候，底部的盒子就会顶上来，因为父盒子没设置高度，变成一条线，big和small已经浮动了

小结：当父元素不给高度的时候，内部元素不浮动的时候会撑开，而浮动时父元素会变成一条线，所以这个时候就需要解决浮动

**2.清除浮动的四种方式**

1. 额外标签法（给最后一个浮动的标签后，新加一个标签，给其设置clear:both;,）（但这种方式是不推荐使用的，因为添加无意义的标签，语义化差）
2. 父元素添加overfiow属性（过触发BFC的方式，实现清除浮动）
3. 使用after伪元素清除浮动

优点：符合闭合浮动思想，结构语义化正确,不容易出现其他为题  
缺点：IE6-7不支持伪元素：after，使用zoom:1触发

1. 使用before和after双伪元素清除浮动

除了这四种还有一种更为简单的方式，给父元素定义好height高度，就解决了父级元素无法获取到高度的问题。

### 四、元素居中的方式

方法一：给父元素设置成弹性盒子，子元素横向居中，纵向居中

方法二：父相子绝后，子部分向上移动本身宽度和高度的一半，也可以用transfrom:translate(-50%,-50%)（最常用方法）

方法三：父相子绝，子元素所有定位为0，margin设置auto自适应

### 五、两/三栏布局（圣杯双飞翼）、

1. **两栏布局,左边定宽，右边自适应**

左边左浮动，右边加oveflow:hidden;变成[BFC](https://blog.csdn.net/weixin_43638968/article/details/109164600)清除左侧浮动元素的影响

1. **三栏布局，圣杯布局、双飞翼布局**

原来我做的主要是后台管理和一些移动端的项目，这两种布局呢是我原来查阅资料时看到的，所以就做了个简单的理解。

圣杯布局和双飞翼布局是前端工程师需要日常掌握的重要布局方式。两者的功能相同，都是为了实现一个两侧宽度固定，中间宽度自适应的三栏布局。（中间先加载渲染）

1. 首先要给两侧设置padding预留出相应的空间
2. 随后分别为三列设置宽度与浮动，同时对footer设置清除浮动
3. 根据浮动的特性，由于center的宽度为100%，即占据了第一行的所有空间，所以left和right被“挤”到了第二行。
4. 接下来的工作是将left放置到之前预留出的位置上，这里使用负外边距
5. 这里使用position: relative和right: 200px将left的位置在原有位置基础上左移200px，以完成left的放置
6. 接下来放置right，只需添加一条声明即可
7. 至此，布局效果完成。不过还需要考虑最后一步，那就是页面的最小宽度：要想保证该布局效果正常显示，由于两侧都具有固定的宽度，所以需要给定页面一个最小的宽度，但这并不只是简单的200+150=350px。回想之前left使用了position: relative，所以就意味着在center开始的区域，还存在着一个left的宽度。所以页面的最小宽度应该设置为200+150+200=550px

### 六、flex布局

Flex布局是css3中新增的一种布局方式，他大多用于移动端中，在PC端中用的比较少。

在PC端中呢会存在一些兼容性的问题，因为flex是css3中新增的一些布局方式，旧版本的浏览可能不支持，所以有时候需要添加浏览器兼容的前缀来解决。

Flex 布局，可以简便、完整、响应式地实现各种页面布局，任何一个容器都可以指定为 Flex 布局，行内元素也可以使用 Flex 布局。

注意，设为 Flex 布局以后，子元素的float、clear和vertical-align属性将失效。

我在项目中常用到的有九宫格布局，列表布局等，都会经常用到。

Flex的属性：

* flex-direction
* flex-wrap
* flex-flow
* justify-content
* align-items
* align-content

### 七、常见的块级、行级、空元素

在CSS中规范规定，每个元素都有display属性，确定该元素的类型，每个元素都有默认的display值，比如div默认display属性值为“block”，成为“块级”元素；span默认display属性值为“inline”，是“行内”元素。

我们在平常的项目中经常使用到的有

* 行内元素有：**span a b i img input select strong**
* 块级元素有：**div p h1-h6 ul table form ul ol li dl dt dd…**
* 空元素(没有内容)： <br> <hr> <img> <input> <link> <meta>

### 八、Css Hack

CSS hack是通过在[CSS样式](http://www.w3cschool.cn/codecamp/style-the-html-body-element.html)中加入一些特殊的符号，让不同的浏览器识别不同的符号（什么样的浏览器识别什么样的符号是有标准的，CSS hack就是让你记住这个标准），以达到应用不同的[CSS](http://www.w3cschool.cn/css)样式的目的。

一般来说是针对不同的浏览器写不同的CSS,就是 CSS Hack。  
CSS Hack常见的有三种形式：属性Hack、选择符Hack、条件注释Hack， Hack主要针对IE浏览器

1. **条件Hack**

条件注释只有在IE浏览器下才能执行，这个代码在非IE浏览下被当做注释视而不见。可以通过IE条件注释载入不同的CSS、JS、HTML和服务器代码等。

1. **选择符 Hack**

比如IE6能识别 \*html .class{}，IE7能识别\*+html .class{}

1. **属性 Hack**  
   比如IE6能识别下划线“\_”和星号“\*”，IE7能识别星号“\*”，但不能识别下划线”\_ ”，而firefox两个都不能认识。

写CSS hack需要遵循以下三条原则：

* 有效： 能够通过 Web 标准的验证
* 只针对太古老的/不再开发的/已被抛弃的浏览器， 而不是目前的主流浏览器
* 代码要丑陋。让人记住这是一个不得已而为之的 Hack, 时刻记住要想办法去掉它。现在很多hacks已经抛弃了最初的原则，而滥用hack会导致浏览器更新之后产生更多的兼容性问题。因此，并不推荐使用CSS hack来解决兼容性问题。

### 九、src与href的区别

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。

经常用到的主要是css外部样式的链接，a标签超链接的使用

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；

经常在script引入时，还有img图片的请求时用到

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。

Frame:是一个现在废弃的标签，可以使一个网站显示在当前的页面上

### 十、link与@import区别

页面中使用CSS的方式主要有3种：行内添加定义style属性值，页面头部内嵌调用和外面链接调用，其中外面引用有两种：Link引入和@import导入，两者都是外部引用CSS的方式，但是存在一定的区别：

1. link是 标签，除了加载CSS外，还可以定义RSS等其他事务；@import属于CSS范畴，只能加载CSS。
2. link引用CSS时，在页面载入时同时加载；@import需要页面网页完全载入以后加载。
3. link是XHTML标签，无兼容问题；@import是在CSS2.1提出的，低版本的浏览器不支持。
4. link支持使用Javascript控制DOM去改变样式；而@import不支持。

### 十一、css3/H5新特新

* 1. **css3新特性**

css3中新增的一些特性大概可以分为：选择器，背景和边框属性，文本效果，2D/3D转换

1. 选择器

常规选择器

:last-child /\* 选择元素最后一个孩子 \*/

:first-child /\* 选择元素第一个孩子 \*/

:nth-child(1) /\* 按照第几个孩子给它设置样式 \*/

:nth-child(even) /\* 按照偶数 \*/

:nth-child(odd) /\* 按照奇数 \*/

:disabled /\* 选择每个禁用的dom元素 \*/

:checked /\* 选择每个被选中的dom元素 \*/

:not(selector) /\* 选择非 selector 元素的每个元素 \*/

::selection /\* 选择被用户选取的元素部分 \*/

2.背景和边框

背景:

规定背景图片的尺寸:background-size

规定背景图片的定位区域:background-origin（content-box、padding-box、margin-box）

边框:

border-radius: 圆角

border-shadow/text-shadow: 阴影

border-image: 边框图片

3.文本效果

|  |  |
| --- | --- |
| text-shadow | 向文本添加阴影 |
| text-justify | 规定当 text-align 设置为 “justify” 时所使用的对齐方法 |
| text-emphasis | 向元素的文本应用重点标记以及重点标记的前景色 |
| text-outline | 规定文本的轮廓 |
| text-overflow | 规定当文本溢出包含元素时发生的事情 |
| text-wrap | 规定文本的换行规则 |
| word-break | 规定非中日韩文本的换行规则 |
| word-wrap | 允许对长的不可分割的单词进行分割并换行到下一行 |
| text-decoration | 文本修饰符：overline、line-through、underline (上划线、中划线、下划线) |

4.2D/3D转换

变形transform：

有rotate旋转、scale缩放、translate位移、skew倾斜

过渡transition：

transition是一个复合属性，可以同时定义transition-property、transition-duration、transition-timing-function、transition-delay子属性值

动画animation

动画的使用，首先通过@（-webkit-)keyframes 定义动画名称及动画的行为，再通过animation属性设置动画特征相关值进行调用

这些新属性大多在新版浏览器得到了支持，有些需要添加浏览器前缀（-webkit-、-moz-、-ms-、-o-）

* 1. **html5新特性**

1. 语义化标签：文档头部:header、文档尾部:footer、文档中的节点:section、导航:nav、侧边栏:aside、独立内容区域:article  
2. 增强型表单：HTML5 拥有多个新的表单 Input 输入类型。这些新特性提供了更好的输入控制和验证  
3. 新增表单属性：（placehoder、required、min 和 max）、音频视频：（audio、video、canvas ）画布、地理定位、拖拽、 本地存储：localStorage 没有时间限制的数据存储；  
sessionStorage， session 的数据存储，当用户关闭浏览器窗口后，数据会被删除  
10. 新事件:

|  |  |
| --- | --- |
| onresize | 当调整窗口大小时触发 |
| ondrag | 当拖动元素时触发 |
| onscroll | 当滚动元素滚动元素的滚动条时触发 |
| onmousewheel | 当转动鼠标滚轮时触发 |
| onerror | 当错误发生时触发 |
| onplay | 当媒介数据将要开始播放时触发 |
| onpause | 当媒介数据暂停时触发 |

### 十二、浏览器兼容性问题

在不同的浏览器中，浏览器的内核都是不相同的，所以各个浏览器对网页的解析存在一定的差异。  
浏览器内核主要分为两种，一是渲染引擎，另一个是js引擎，所以浏览器的兼容性问题一般是指:css兼容、js兼容

**Css兼容性问题：**

**1.不同浏览器的标签默认的margin和padding不同**

解决办法：可以通过设置全局样式来解决这个问题，这样所有的样式都会重置，初始值都会相同了。

**2.css3新属性，加浏览器前缀兼容早期浏览**

-moz- / 火狐浏览器 /  
-webkit- / Safari, 谷歌浏览器等使用Webkit引擎的浏览器 /  
-o- / Opera浏览器(早期) /  
-ms- / IE /

**3.块属性标签float后，又有横行的margin的情况下，IE浏览器margin加倍的问题**

设置为float的div在IE下设置的margin会加倍。这是一个IE6都存在的bug。解决方案是在这个div里面加上display:inlin.

**4.设置较小高度标签（小于一般10px），在IE6，IE7，遨游中高度超出自己设置高度**

可以通过给高出的标签设置overflow:hidden

或设置行高line-height小于你设置的高度

**5.行内属性标签，设置display:black后采用float布局，又有横行的margin的情况下，IE6间距bug**

IE6里的间距比超过设置的间距，可以通过在display:block;后面加入display:inline;display:table;

**6.IE浏览器div最小高度和宽度的问题**

加宽高，并且宽高加auto

**7.超链接访问过hover样式就不出现的问题**

被点击访问过的超链接样式不在具有hover和active了,很多人应该都遇到过这个问题,解决技巧是改变CSS属性的排列顺序: L-V-H-A

a:link{}>a:visited{}>a:hover{}>a:active{}

**8.图片默认有间距**

几个img标签放在一起的时候，有些浏览器会有默认的间距，通配符清除间距也不起作用。  
可以通过使用float属性为img布局(所有图片左浮)

9. **css hack解决浏览器兼容性**

不同浏览器，识别不同的样式，css hack本身就是处理浏览器兼容的

**二、js兼容**

1. **事件绑定**

IE: dom.attachEvent();  
标准浏览器： dom.addEventListener(‘click',function(event){},false);

**var x = document.getElementById("myBtn");**

**if (x.addEventListener) { //所有主流浏览器，ie9+**

**x.addEventListener("click", myFunction);**

**} else if (x.attachEvent) { // IE 8 及更早 IE 版本**

**x.attachEvent("onclick", myFunction);**

**}**

**2.event事件对象问题**

**document.onclick=function(ev){//谷歌火狐的写法，IE9以上支持，往下不支持；**

**var e=ev;**

**console.log(e);**

**}**

**document.onclick=function(){//谷歌和IE支持，火狐不支持；**

**var e=event;**

**console.log(e);**

**}**

**document.onclick=function(ev){//兼容写法；**

**var e=ev||window.event;**

**var mouseX=e.clientX;//鼠标X轴的坐标**

**var mouseY=e.clientY;//鼠标Y轴的坐标**

**}**

**3.** **event.srcElement(事件源对象)问题**

IE： event对象有srcElement属性，但是没有target属性；  
Firefox: event对象有target属性，但是没有srcElement属性。

解决办法：srcObj = event.srcElement?event.srcElement:event.target;

**4.获取元素的非行间样式值**

IE: dom.currentStyle[‘width’] 获取元素高度  
标准浏览器：window.getComputedStyle(obj, null)['width'];

**// 获取元素属性值的兼容写法**

**function getStyle(obj,attr){**

**if(obj.currentStyle){**

**//兼容IE**

**obj.currentStyle[attr];**

**return obj.currentStyle[attr];**

**}else{**

**//非IE，**

**return window.getComputedStyle(obj, null)[attr];**

**}**

**}**

**5.阻止事件冒泡传播**

**//js阻止事件传播，这里使用click事件为例**

**document.onclick=function(e){**

**var e=e||window.event;**

**if (e.stopPropagation) {**

**e.stopPropagation();//W3C标准**

**}else{**

**e.cancelBubble=true;//IE.... true:**  不被上层元素的事件控制

**}**

**}**

**6.阻止事件默认行为**

**//js阻止默认事件 一般阻止a链接href，form表单submit提交**

**document.onclick=function(e){**

**var e=e||window.event;**

**if (e.preventDefault) {**

**e.preventDefault();//W3C标准**

**}else{**

**e.returnValue='false';//IE.. false:** 不会进行判断就直接执行下去

**}**

**}**

**7.** **ajax兼容问题**

IE： ActiveXObject  
其他： xmlHttpReuest

在IE6以前不是用XMLHttpRequest创建的，所以我们要兼容ie6以前的浏览器要判断他有没有XMLHttpRequest()

跨浏览器兼容解决方案:

**<script>**

**window.onload = function(){**

**var oBtn = document.getElementById('btn');**

**oBtn.onclick = function(){**

**//1.创建ajax对象**

**//只支持非IE6浏览器**

**var oAjax = null;**

**if(window.XMLHttpRequest){**

**oAjax = new XMLHttpRequest();**

**//alert(new XMLHttpRequest());**

**}else{**

**//只支持IE6浏览器**

**oAjax = new ActiveXObject("Microsoft.XMLHTTP");**

**}**

**//2.连接服务器,这里加个时间参数,每次访问地址都不一样,浏览器就不用浏览器里的缓冲了,但**

**// 但服务器那端是不解析这个时间的**

**oAjax.open("get","a.txt?t=" + new Date().getTime(),true);**

**//3.发送**

**oAjax.send(null);**

**//4.接受信息**

**oAjax.onreadystatechange = function(){**

**//浏览器与服务器之间的交互,进行到哪一步了,当等于4的时候,代表读取完成了**

**if(oAjax.readyState==4){**

**//状态码,只有等于200,代表接受完成,并且成功了**

**if(oAjax.status==200){**

**alert("成功" + oAjax.responseText);**

**}else{**

**alert("失败");**

**}**

**}**

**};**

**};**

**};**

**</script>**

## ****Js相关****

JavaScript分三个部分:

    1. ECMAScript标准---基本语法

    2. DOM--->Document Object Model 文档对象模型,操作页面元素的

    3. BOM--->Browser Object Model 浏览器对象模型,操作浏览器的

    浏览器中有个顶级对象:window----皇上

    页面中顶级对象:document-----总管太监

    页面中所有的内容都是属于浏览器的,页面中的内容也都是window的，因为页面中的所有内容都是window的,window是可以省略的.

    变量是window的

### 原型和原型链

1. prototype

每个函数都有一个prototype属性，被称为显示原型

2.\_ \_proto\_ \_

每个实例对象都会有\_ \_proto\_ \_属性,其被称为隐式原型

每一个实例对象的隐式原型\_ \_proto\_ \_属性指向自身构造函数的显式原型prototype

3. constructor

每个prototype原型都有一个constructor属性，指向它关联的构造函数。

4. 原型链

获取对象属性时，如果对象本身没有这个属性，那就会去他的原型\_ \_proto\_ \_上去找，如果还查不到，就去找原型的原型，一直找到最顶层(Object.prototype)为止。Object.prototype对象也有\_ \_proto\_ \_属性值为null。

### 作用域

**1.作用域：**

作用域就是一个变量可以使用的范围，主要分为全局作用域和函数作用域

全局作用域就是Js中最外层的作用域

函数作用域是js通过函数创建的一个独立作用域，函数可以嵌套，所以作用域也可以嵌套

Es6中新增了块级作用域（由大括号包裹，比如：if(){},for(){}等）

**2.自由变量：**

当前作用域外的变量都是自由变量，一个变量在当前作用域没有定义，但是被使用了，就会向上级作用域，一层一层依次查找，直至找到为止，找到这个变量后就会停止，不会继续查找这个变量，如果全局作用域都没有找到这个变量就会报错。这个自由变量查找的过程就是作用域链。

**3.变量提升:**

每个var声明的变量，function声明的函数存在变量提升。let const不存在变量提升

在js中声明之前未定义，会在js的最上方会形成一个预解析池，用来存储声明了但没有先定义的变量名

### 闭包

**闭包：**

简单的理解就是函数中套了一个函数，内层函数可以访问外层函数中的变量

有时候需要用到函数内的局部变量，在正常情况下是不能读取到的，这个时候就需要用到闭包。

闭包可以封装对象的私有属性和方法，就是可以简单的理解成闭包就是一个私有作用域，可以定义属性和方法，vue中的data就是一种闭包的形式。

闭包作为回调函数，可以实现函数的复用

优点：闭包因为长期驻扎在内存中。可以重复使用变量，不会造成变量污染

缺点：闭包会使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，可能会导致内存泄露。解决方法是在退出函数之前，将不使用的变量全部删除。

### This的指向

在js中this不是固定不变的，它会随着执行环境的改变而改变。要注意的是this取什么值，是在执行时确认的，定义时无法确认。

this的调用大概分为五种场景：

1.浏览器里，在全局范围内的this 指向window对象；

2.在函数中，this永远指向最后调用他的那个对象；

3.构造函数中，this指向new出来的那个新的对象；

4.Call、apply、bind中的this被强绑定在指定的那个对象上；

5.箭头函数中this比较特殊,箭头函数this为父作用域的this，不是调用时的this.要知道前四种方式,都是调用时确定,也就是动态的,而箭头函数的this指向是静态的,声明的时候就确定了下来；

**call**:参数是单个使用的，  
**apply**:参数是一个集合时使用，  
**bind**:使用bind会改变this，不会改变数据，需要在调用的地方加一个括号

### 垃圾回收和内存机制

**垃圾回收：**

浏览器的js具有自动垃圾回收机制，也就是说，执行环境会负责管理代码在执行中使用的内存，垃圾收集器会定期的找出那些不在继续使用的变量，然后释放内存。但是这个过程不是实时的，因为其开销比较大并且GC时停止响应其他操作，所以垃圾回收器会按照固定的时间间隔周期性的执行。

js中最常用的垃圾回收方式就是标记清除。当变量进入环境时，例如，在函数中声明一个变量，就将这个变量标记为“进入环境”。从逻辑上讲，永远不能释放进入环境的变量所占用的内存，因为只要执行流进入相应的环境，就可能会用到它们。而当变量离开环境时，则将其标记为“离开环境”。

引用计数的含义是跟踪记录每个值被引用的次数。当声明了一个变量并将一个引用类型值赋给该变量时，则这个值的引用次数就是1。如果同一个值又被赋给另一个变量，则该值的引用次数加 1。相反，如果包含对这个值引用的变量又取得了另外一个值，则这个值的引用次数减 1。当这个值的引用次数变成 0 时，则说明没有办法再访问这个值了，因而就可以将其占用的内存空间回收回来。这样，当垃圾回收器下次再运行时，它就会释放那些引用次数为 0 的值所占用的内存。

**内存泄漏：**

1. **循环引用**

一个很简单的例子：一个DOM对象被一个Javascript对象引用，与此同时又引用同一个或其它的Javascript对象，这个DOM对象可能会引发内存泄露。这个DOM对象的引用将不会在脚本停止的时候被垃圾回收器回收。要想破坏循环引用，引用DOM元素的对象或DOM对象的引用需要被赋值为null。

1. **闭包**

在闭包中引入闭包外部的变量时，当闭包结束时此对象无法被垃圾回收（GC）。

1. **DOM泄露**

当原有的DOM被移除时，子结点引用没有被移除则无法回收

1. **Times计时器泄露**

### 宏任务和微任务

js中的一个机制，就是遇到宏任务，先执行宏任务，将宏任务放入eventqueue，然后在执行微任务。

宏任务和微任务都包含一些事件

宏任务：setTimeout,setInterval,Ajax,DOM事件

微任务：Promise async/await

要想明白这个机制 就要理解js单线程。因为JS是单线程语言，只能同时做一件事儿。js任务需要排队顺序执行，如果一个任务时间过长，后边的任务也会等着。假如，我们在请求一个网址时，图片加载很慢，网页总不能一直卡不出来，这个时候就可以用异步来解决了

异步就是由单线程这个背景而来的，解决了单线程等待的这个问题，异步的特点不会阻塞后面代码的执行。也就是请求处理过程中，你不能闲着，会产生异步的请求，回头再处理，然后继续执行下面的请求

异步和单线程是相辅相成的，js是一门单线程脚本语言，所以需要异步来辅助

### 七、Js基础

#### 一、数据类型

基本数据类型（值类型）： Number、String、Boolean、Undefined、Null、Symbol（es6新增独一无二的值） 和 BigInt（es10新增）；

引用数据类型: Object。包含Object、Array、 function、Date、RegExp。

备注： 基本数据类型，又称值类型。

1.堆栈内存：

引用类型存储把值存储在堆内存中，堆内存是从下往上存储。生成唯一内存地址。然后在栈内存中把地址赋值给变量。栈内存是从上往下存储的。之所以如此划分内存主要考虑到特别大的对象进行值传递时的效率问题

2.Typeof运算符：

typeof 能有效检测基本类型，检测引用类型都返回object,其中null属于特殊的引用类型返回object,function属于特殊引用类型类型不用于存储数据，typeof检测返回function.

3.==和===

===三等表示全等，判断左右两边对象或值是否类型相同且值相等。

==二等表示值相等。判断操作符两边对象或值是否相等类型可以不同，类型不同时，使用Number()转换成Number类型在进行判断。例外规则，null==undefined，null/undefined进行运算时不进行隐式类型转换。通常把值转为Boolean值，进行条件判断。Boolean(null)===Boolean(undefined)>false===false 结果为true

4.if语句和逻辑运算

所有基本类型中Boolean值是false的只有6个，分别是 : 0 NaN ' ' null undefined false  
引用类型Boolean值全是true.

if条件是单个值时，如果是truly值，条件成立， 如果是falsely值，条件不成立

#### 二、手写深拷贝

深拷贝和浅拷贝的区别

1.浅拷贝： 将原对象的引用直接赋给新对象，新对象只是原对象的一个引用，而不复制对象本身，新旧对象还是共享同一块内存

2.深拷贝： 创建一个新的对象和数组，将原对象的各项属性的“值”（数组的所有元素）拷贝过来，是“值”而不是“引用”，新对象跟原对象不共享内存，修改新对象不会改到原对象

为什么要使用深拷贝？

我们希望在改变新的数组（对象）的时候，不改变原数组（对象）

arr=[100, [{a : 'hello'}, {b : "world"}], { c: "123456789" }];

    //判断修改的是不是'object'或者null,如果不是 object 或者 null 那么直接返回

    function deepClone(obj = {}) {

        if (typeof obj !== 'object' || obj == null) {

            return obj;

        }

        let result;

        //判断上面下来的obj是不是数组 用instanceof来检测 因为是数组应用类型

        obj instanceof Array?result=[]:result={}

        for (var item in obj) {

            //查找一个对象是否有某个属性

            if (obj.hasOwnProperty(item)) {

                // 递归调用

                result[item] = deepClone(obj[item])

            }

        }

        return result;

    }

    let arr1=deepClone(arr);

    arr1[1][0].a='vue'

    arr1[0]=99

    console.log('原数组',arr)

    console.log('新数组',arr1)

#### 三、类型判断

数据类型判断大概有四种**typeof、instanceof、constructor、Object.prototype.toString.call()**

**1.Type:**

基本数据类型中：Number，String，Boolean，undefined 以及引用数据类型中Function ,可以使用typeof检测数据类型,分别返回对应的数据类型小写字符。

另：用typeof检测构造函数创建的Number，String，Boolean都返回object

基本数据类型中：null 。引用数据类型中的：Array，Object，Date，RegExp。不可以用typeof检测。都会返回小写的object

**2 . instanceof**  
除了使用typeof来判断，还可以使用instanceof。instanceof运算符需要指定一个构造函数，或者说指定一个特定的类型，它用来判断这个构造函数的原型是否在给定对象的原型链上。

**3.constructor**

constructor是prototype对象上的属性，指向构造函数。根据实例对象寻找属性的顺序，若实例对象上没有实例属性或方法时，就去原型链上寻找，因此，实例对象也是能使用constructor属性的

**4 . 使用Object.prototype.toString.call()检测对象类型**

可以通过toString() 来获取每个对象的类型。为了每个对象都能通过 Object.prototype.toString() 来检测，需要以 Function.prototype.call() 或者 Function.prototype.apply() 的形式来调用，传递要检查的对象作为第一个参数，称为thisArg。

#### 四、数组方法

1.push() 可以添加一个或多个参数到数组的尾部，添加之后原来的数组会发生改变，返回的是添加后的数组的长度

2.pop() 从数组尾部删除一个元素，原数组会发生改变，返回数组中被删除的元素

3.unshift() 可以添加一个或多个参数到数组的头部，添加后原来的数组会发生改变，返回的是添加后的数组的长度

4.shift() 从数组头部删除一个元素，原数组会发生改变，返回数组中被删除的元素

5.slice() 从数组中截取，如果不传参，会返回原数组。如果只传入一个参数，会从头部开始删除，直到数组结束，原数组不会改变；传入两个参数，第一个是开始截取的索引，第二个是结束截取的索引，原数组不会改变。最多可以接受两个参数。

6.splice()截取类 没有参数，返回空数组，原数组不变；一个参数，从该参数表示的索引位开始截取，直至数组结束，返回截取的 数组，原数组改变；两个参数，第一个参数表示开始截取的索引位，第二个参数表示截取的长度，返回截取的 数组，原数组改变；三个或者更多参数，第三个及以后的参数表示要从截取位插入的值。

7.reverse() 数组翻转

8.sort() 数组排序

9.join() 数组拼接

10.isArray() 判断是否是数组

11.toString() 数组转字符串

#### 五、数组去重

1. 使用ES6中的set是最简单的去重方法

2. 利用Map数据结构去重

创建一个空Map数据结构，遍历需要去重的数组，把数组的每一个元素作为key存到Map中。由于Map中不会出现相同的key值，所以最终得到的就是去重后的结果。

3递归去重

4.forEach+indexOf

定义一个空数组，通过forEach循环，indexOf判断值是否是-1，如果是push到新的数组中

#### 六、数组排序

1.冒泡排序法

将数组中的相邻两个元素进行比较，将比较大（较小）的数通过两两比较移动到数组末尾（开始），执行一遍内层循环，确定一个最大（最小）的数，外层循环从数组末尾（开始）遍历到开始（末尾）

2. 插入排序法（插队排序）

将要排序的数组分成两部分，每次从后面的部分取出索引最小的元素插入到前一部分的适当位置

·从第一个元素开始，该元素可以认为已经被排序；

·取出下一个元素，在已经排序的元素序列中从后向前扫描；

·如果该元素（已排序）大于新元素，将该元素移到下一位置；

·重复步骤3，直到找到已排序的元素小于或者等于新元素的位置；

·将新元素插入到该位置后；

·重复步骤2~5。

3.快速排序法

实现思路是，将一个数组的排序问题看成是两个小数组的排序问题，以一个数为基准(中间的数)，比基准小的放到左边，比基准大的放到右边，而每个小的数组又可以继续看成更小的两个数组，一直递归下去，直到数组长度大小最大为2。

#### 数组扁平化

数组扁平化就是将一个多维数组转换为一个一维数组

实现基本方式

1、对数组的每一项进行遍历。

2、判断该项是否是数组。

3、如果该项不是数组则将其直接放进新数组。

4、是数组则回到1，继续迭代。

5、当数组遍历完成，返回这个新数组。

#### 八、字符串

### 高频面试题

#### for···in和for···of的区别

①从遍历数组角度来说，for···in遍历出来的是key(即下标)，for···of遍历出来的是value(即数组的值)；

②从遍历字符串的角度来说，同数组一样。  
③从遍历对象的角度来说，for···in会遍历出来的为对象的key，但for···of会直接报错。

④如果要使用for…of遍历普通对象，需要配合Object.keys()一起使用。

#### 二、null和undefined区别

在 if 语句中 null 和 undefined 都会转为false两者用相等运算符比较也是相等

**1.null表示没有对象，可能将来要赋值一个对象，即该处不应该有值**

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

1. **undefined表示缺少值，即此处应该有值，但没有定义**

1）定义了形参，没有传实参，显示undefined

2）对象属性名不存在时，显示undefined

3）函数没有写返回值，即没有写return，拿到的是undefined

4）写了return，但没有赋值，拿到的是undefined

#### 三、this, call, apply, bind

1.浏览器里，在全局范围内的this 指向window对象；

2.在函数中，this永远指向最后调用他的那个对象；

3.构造函数中，this指向new出来的那个新的对象；

4.Call、apply、bind中的this被强绑定在指定的那个对象上；

5.箭头函数中this比较特殊,箭头函数this为父作用域的this，不是调用时的this.要知道前四种方式,都是调用时确定,也就是动态的,而箭头函数的this指向是静态的,声明的时候就确定了下来；

6.apply、call、bind都是js给函数内置的一些API，调用他们可以为函数指定this的执行,同时也可以传参。

#### 四、继承

1.原型链继承

父类的实例作为子类的原型

function Woman(){

}

Woman.prototype= new People();

Woman.prototype.name = 'haixia';

let womanObj = new Woman();

优点：简单易于实现，父类的新增实例与属性子类都能访问

缺点：

1）.可以在子类中增加实例属性，如果要新增加原型属性和方法需要在new 父类构造函数的后面

2）无法实现多继承

3）创建子类实例时，不能向父类构造函数中传参数

2.借用构造函数继承（伪造对象、经典继承）

复制父类的实例属性给子类

function Woman(name){

//继承了People

People.call(this); //People.call(this，'tom);

this.name = name || 'kitty'

}

let womanObj = new Woman();

优点：

1）解决了子类构造函数向父类构造函数中传递参数

2）可以实现多继承（call或者apply多个父类）

缺点：

1）方法都在构造函数中定义，无法复用

2）不能继承原型属性/方法，只能继承父类的实例属性和方法

3.实例继承（原型式继承）

function Wonman(name){

let instance = new People();

instance.name = name || 'wangxiaoxia';

return instance;

}

let wonmanObj = new Wonman();

优点：

1）不限制调用方式

2）简单，易实现

缺点：

不能多次继承

4.组合式继承

调用父类构造函数，继承父类的属性，通过将父类实例作为子类原型，实现函数复用

function People(name,age){

this.name = name || 'wangxiao'

this.age = age || 27

}

People.prototype.eat = function(){

return this.name + this.age + 'eat sleep'

}

function Woman(name,age){

People.call(this,name,age)

}

Woman.prototype = new People();

Woman.prototype.constructor = Woman;

let wonmanObj = new Woman(ren,27);

wonmanObj.eat();

缺点：

由于调用了两次父类，所以产生了两份实例

优点：

1）函数可以复用

2)不存在引用属性问题

3)可以继承属性和方法，并且可以继承原型的属性和方法

ES5继承和ES6继承的区别：

es5继承首先是在子类中创建自己的this指向，最后将方法添加到this中

Child.prototype=new Parent() || Parent.apply(this) || Parent.call(this)

es6继承是使用关键字先创建父类的实例对象this，最后在子类class中修改this

#### 五、设计模式

一、单例模式

1. 定义

保证一个类仅有一个实例，并提供一个访问它的全局访问点

2. 核心

确保只有一个实例，并提供全局访问

3. 实现

在vue脚手架项目开发中，我们需要对axios进行请求拦截，响应拦截，多次调用封装好的axios实例也仅设置一次，封装后的axios就是要一个单例

二、观察者、订阅者模式

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

### 九、dom操作

DOM是网页中用来表示文档中对象的标准模型，他是由节点和对象组成的结构集合。在浏览器解析HTML标签时，会构建一个DOM树结构，把html结构化成js可以识别的树模型。  
树模型构成的层级结构，可以很容易的表明家族成员之间的关系，把复杂的关系简明地表示出来  
由此呢js也提供了一些dom的操作

一、dom元素获取

1. document.getElementById(id的值) 通过id来获取元素的对象，返回值是一个对象
2. document.getElementsByName(name) 通过name属性来获取对象的,返回值是一个数组，与getElementById()方法类似，但他是查询的name元素，而不是id属性
3. document.getElementsByTagName() 通过标签来获取元素的对象, 返回值是一个数组
4. document.getElementsByClassName() 通过class类名来获取的对象,返回值是一个数组
5. document.querySelector() css选择器，返回与该模式匹配的第一个元素，结果为一个元素；如果没找到匹配的元素，则返回null
6. document.querySelectorAll() css选择器，返回与该模式匹配的所有元素，结果为一个类数组二、dom创建

二、dom操作

1. 创建：新的标签(元素节点) = document.createElement("标签名")  
2. 删除：父节点.removeChild(子节点);  
3. 插入：insertBefore(新插入的节点，参照物节点) 往某个节点的前面插入一个新的节点  
4. 追加：appendChild(新的节点的名) 当前对象追加一个子节点

### 十、js操作BOM

 浏览器对象模型（BOM ：Browser  Object  Model）是JavaScript的组成之一，它提供了独立于内容与浏览器窗口进行交互的对象，使用浏览器对象模型可以实现与HTML的交互。它的作用是将相关的元素组织包装起来，提供给程序设计人员使用，从而降低开发人员的劳动量，提高设计Web页面的能力。

BOM是一个分层结构

浏览器中有个顶级对象:window----皇上

    页面中顶级对象:document-----总管太监

    页面中所有的内容都是属于浏览器的,页面中的内容也都是window的，因为页面中的所有内容都是window的,window是可以省略的.

    变量是window的

### 十一、事件

#### 事件绑定、事件流、自定义事件

1.DOM事件三种级别：

DOM0级事件

DOM0 级时间分两种，一是直接在标签内直接添加执行语句，二是定义执行函数。

<input type="text" id="test">

<input type="button" value="button" onclick="alert(document.getElementById('test').value)">

<script>

document.getElementById('button').onclick=function(){ alert(document.getElementById('test').value);

}

</script>

DOM2 级事件

第一个参数：事件名称

第二个参数：执行函数

第三个参数：指定冒泡还是捕获，默认是false，冒泡。

element.addEventListener('click',function(){},false)

DOM3 级事件

同DOM2级一样，只不过添加了更多的事件类型，鼠标事件、键盘事件

element.addEventListener('keyup',function(){},false)

2.DOM事件两种类型

事件类型分两种：事件捕获、事件冒泡。

事件捕获就是由外往内，从事件发生的顶点开始，逐级往下查找，一直到目标元素。

事件冒泡就是由内往外，从具体的目标节点元素触发，逐级向上传递，直到根节点。

3.DOM事件的事件流（事件传播）

事件流就是，事件传播过程。

DOM完整的事件流包括三个阶段：事件捕获阶段、目标阶段和事件冒泡阶段。

事件通过捕获到达目标元素，这个时候就是目标阶段。从目标节点元素将事件上传到根节点的过程就是第三个阶段，冒泡阶段。

4.事件捕获的具体流程

当事件发生在 DOM元素上时，该事件并不完全发生在那个元素上。在捕获阶段，事件从window开始，之后是document对象，一直到触发事件的元素。

5.事件冒泡的具体过程

当事件发生在DOM元素上时，该事件并不完全发生在那个元素上。在冒泡阶段，事件冒泡，或者事件发生在它的父代，祖父母，直到到达window为止。

6.我们也可以通过 new Event()自定义事件

var eve = new Event('test'); //通过new Event 创建事件

dom.addEventListener('test', function () { //注册事件

console.log('test dispatch');

});

setTimeout(function () {

dom.dispatchEvent(eve); //触发事件

}, 1000);

#### 二、事件委托

瀑布流：无限上拉列表中，如果给每一个图片绑定点击事件，非常繁琐且消耗内存。所以我们可以把每张图片上的点击事件委托给共同的父元素。

事件委托，又名事件代理。事件委托就是利用事件冒泡，就是把子元素的事件都绑定到父元素上。如果子元素阻止了事件冒泡，那么委托也就没法实现了

好处：提高性能，减少了事件绑定，从而减少内存占用

应用场景 在vue中事件委托

我们经常遇到vue中v-for一个列表，列表的每一项都绑定了@click处理事件。我们都知道绑定这么多监听，从性能方面来说是不太好的。那我们我们可以通过把每个item的click事件委托给父元素的形式来实现

#### 三、封装一个通用的事件绑定函数

需要点击每个a，来。弹出他们的内容

<div id="div3">

<a href="#">a1</a><br>

<a href="#">a2</a><br>

<a href="#">a3</a><br>

<a href="#">a4</a><br>

<button id='btn1'>加载更多...</button>

</div>

// 封装通用的事件绑定函数

function bindEvent(elem, type, fn) {

elem.addEventListener(type, fn)

}

//获取父元素

const fu = document.getElementById('div3')

bindEvent(fu, 'click', function (event) {

// console.log(event.target) // 获取触发的元素

let target=event.target

event.preventDefault() // 阻止默认行为

//过滤符合条件的子元素，主要是过滤掉 加载更多

if(target.nodeName.toLowerCase()==="A"){

alert(target.innerHTML;

}

})

### 十二、Ajax

#### 一、原生Ajax的创建过程

1.创建xhr 核心对象

var xhr=new XMLHttpRequest();

2.调用open 准备发送

参数一：请求方式

参数二: 请求地址

参数三：true异步，false 同步

xhr.open('post','http://www.baidu.com/api/search',true)

3.如果是post请求，必须设置请求头。

xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')

4.调用send 发送请求 （如果不需要参数，就写null）

xhr.send('user=tom&age=10&sex=女')

5.监听异步回调 onreadystatechange

判断readyState 为4 表示请求完成

判断status 状态码 为 200 表示接口请求成功

responeseText 为相应数据。字符串类型。

xhr.onreadystatechange=function(){

if(xhr.readyState==4){

if(xhr.status==200){

console.log(xhr.responseText);

var res=JSON.parse(xhr.responseText);

console.log(res);

if(res.code==1){

modal.modal('hide');

location.reload();

}

}

}

**备注：如果是post请求，想要传json格式数据。**

设置请求头

1.xhr.setRequestHeader('Content-Type', 'application/json')

open发送数据

2.xhr.open({\_id:xxx,user:xxxx,age:xxxx})

#### 二、Jsonp的原理

**JSONP原理：**

ajax 请求受同源策略影响，不允许进行请求，我们利用 script 标签的 src 属性不受同源策略的约束，利用这个特性jsonp需要以下步骤：

1.动态创建<script></script>(document.createElement('script'))

2.设置src 属性，(src中要包含参数callback=fn)进行跨域请求

3.将 <script></script>添加到页面中执行 （body.appendChild('script')）

4.页面要提前定义好callback。

5.后端会返回回调函数执行并包裹参数callback(data)

备注：

服务端不再返回JSON格式的数据，而是返回回调函数包裹数据（fn({name:'tom',age:18}），在src中进行了调用，这样实现了跨域。

### 十三、存储

本地存储分为cookie、localStorage、sessionStorage，

Cookie

Cookie设计初衷是用来和服务器通讯，而不是本地存储，他只是被‘借用’到本地存储。

Cookie有一些缺点：存储空间小，最大4k、http请求时需要发送到服务器，增加请求数据量、只能用document.cookie=’…’ 来修改，太过简陋

localStorage、sessionStorage

在HTML5中，新加入了一个localStorage/sessionStorage特性，这个特性主要是用来作为本地存储来使用的，解决了cookie存储空间不足的问题(cookie中每条cookie的存储空间为4k)，localStorage中一般浏览器支持的是5M大小。

优点：HTML5专门为存储而设计，最大可存5M、API简单易用setItem getItem、不会随着http请求被发送出去

localStorage、sessionStorage不同点：

localStorage数据会永久存储，除非代码或手动删除

sessionStroage 数据只存在于当前会话，浏览器关闭则清空

一般用localStorage会更多一些

注意：localStorage只支持string类型的存储。

### 十四、Es6

#### let、var、const区别

在ES5中，声明变量只有var和function两种形式。但是因为var声明的变量会有一定的缺点（内层变量可能覆盖外层变量的问题以及用来计数的循环变量泄露为全局变量，下面有介绍），ES6提出了使用let和const声明变量,弥补了ES5中var的缺点。

1.是否存在变量提升？

var声明的变量存在变量提升（将变量提升到当前作用域的顶部）。即变量可以在声明之前调用，值为undefined。

let和const不存在变量提升。即它们所声明的变量一定要在声明后使用，否则报ReferenceError错。

2.是否存在暂时性死区？

let和const存在暂时性死区。即只要块级作用域内存在let命令，它所声明的变量就“绑定”（binding）这个区域，不再受外部的影响。

在代码块内，使用let命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”（temporal dead zone，简称 TDZ）。

总之，暂时性死区的本质就是，只要一进入当前作用域，所要使用的变量就已经存在了，但是不可获取，只有等到声明变量的那一行代码出现，才可以获取和使用该变量。

3.是否允许重复声明变量？

var允许重复声明变量。

let和const在同一作用域不允许重复声明变量。

4.是否存在块级作用域？

var不存在块级作用域。

let和const存在块级作用域。

什么是块级作用域：

ES5中作用域有：全局作用域、函数作用域。没有块作用域的概念。因此也有一系列的问题。

ECMAScript 6(简称ES6)中新增了块级作用域。块作用域由{ }包括，if语句和for语句里面的{ }也属于块作用域。

5. 是否能修改声明的变量？

var和let可以。

const声明一个只读的常量。一旦声明，常量的值就不能改变。const声明的变量不得改变值，这意味着，const一旦声明变量，就必须立即初始化，不能留到以后赋值。

#### 二、es6解构赋值

解构赋值就是从目标对象或数组中提取自己想要的变量。最常用的场景是：element-ui,vant-ui按需引入，请求接口返回数据，提取想要数据。

常见的几种方式有

1.默认值

2.交换变量

3.将剩余数组赋给一个变量

当结构一个数组时，可以使用剩余模式，将数组剩余部分赋值给一个变量

4.给新的变量名赋值

可以从一个对象中提取变量并赋值给和对象属性名不同的新的变量名

#### 三、箭头函数与普通函数的区别

1、箭头函数是匿名函数，不能作为构造函数，不能使用new

箭头函数相当于匿名函数，并且简化了函数定义。箭头函数有两种格式：一种只包含一个表达式，连{ ... }和return都省略掉。还有一种可以包含多条语句，这时候就不能省略{ ... }和return。

2.箭头函数内没有arguments,可以用展开运算符…解决

arguments:是一个方法调用的集合，是一个伪数组，不是真的数组，不具有数组的操作的方法，可以用展开运算解决（...）

3.箭头函数的this，始终指向父级上下文（箭头函数的this取决于定义位置父级的上下文，跟使用位置没关系，普通函数this指向调用的那个对象）

4.箭头函数不能通过call（）、apply（）、bind（）方法直接修改它的this指向。

5.箭头函数没有原型属性

#### 四、class与class继承

传统的javascript中只有对象，没有类的概念。它是基于原型的面向对象语言。原型对象特点就是将自身的属性共享给新对象。这样的写法相对于其它传统面向对象语言来讲，独树一帜也可以说难以接受！

ES5中如果要生成一个对象实例，需要先定义一个构造函数，然后通过new操作符来完成。

构造函数生成实例的执行过程：

1.当使用了构造函数，并且new 构造函数(),后台会隐式执行new Object()创建对象;

2.将构造函数的作用域给新对象，（即new Object()创建出的对象），而函数体内的this就代表new Object()出来的对象。

3.执行构造函数的代码。

4.返回新对象（后台直接返回）;

ES6中的类

ES6引入了class（类）这个概念，通过class关键字可以定义类。该关键字的出现使得javascript在对象写法上更加清晰，更像是一种面向对象的语言。

注意项:

1.在类中声明方法的时候，千万不要给该方法加上function关键字

2.方法之间不要用逗号分隔，否则会报错

ES5中的继承 （组合继承:原型链继承 + 借用构造函数)

原型链继承：

父类的实例作为子类的原型

借用构造函数继承：

在子类内,使用call()调用父类方法,并将父类的this修改为子类的this.相当于是把父类的实例属性复制了一份放到子类的函数内.

组合继承:

既能调用父类实例属性，又能调用父类原型属性

#### 五、promise使用及实现

什么是promise

promise是异步编程的一种方案，解决了地狱回调的问题，是一种链式调动的方式

Promise 简单说就是一个容器，里面保存着某个未来才会结束的事件（通常是一个异步操作）的结果。

promise 是一个对象，从它可以获取异步操作的的最终状态（成功或失败）。

Promise是一个构造函数，对外提供统一的 API，自己身上有all、reject、resolve等方法，原型上有then、catch等方法。

名词约定

promise（首字母小写）对象指的是“Promise实例对象”

Promise 首字母大写且单数形式，表示“Promise构造函数”

Promises 首字母大写且复数形式，用于指代“Promises规范”

Promise的两个特点

Promise对象的状态不受外界影响

1）pending 初始状态

2）fulfilled 成功状态

3）rejected 失败状态

Promise 有以上三种状态，只有异步操作的结果可以决定当前是哪一种状态，其他任何操作都无法改变这个状态

Promise的状态一旦改变，就不会再变，任何时候都可以得到这个结果，状态不可以逆，只能由 pending变成fulfilled或者由pending变成rejected

使用 new 来创建一个promise对象。

Promise接受一个「函数」作为参数，该函数的两个参数分别是resolve和reject。这两个函数就是就是「回调函数」

resolve函数的作用：在异步操作成功时调用，并将异步操作的结果，作为参数传递出去；

reject函数的作用：在异步操作失败时调用，并将异步操作报出的错误，作为参数传递出去

then()方法： then 方法就是把原来的回调写法分离出来，在异步操作执行完后，用链式调用的方式执行回调函数。

catch()方法： 当执行 resolve 的回调（也就是上面 then 中的第一个参数）时，如果抛出异常了（代码出错了），那么也不会报错卡死 js，而是会进到这个 catch 方法中。

all()方法： Promise 的 all 方法提供了并行执行异步操作的能力，并且在所有异步操作执行完后才执行回调。

race()方法： race 按字面解释，就是赛跑的意思。race 的用法与 all 一样，只不过 all 是等所有异步操作都执行完毕后才执行 then 回调。而 race 的话只要有一个异步操作执行完毕，就立刻执行 then 回调。

promise简单举例

//Promise异步封装ajax

<script>

function Feach(method, ) {

let p = new Promise((resolve, reject) => {

// 手写原生ajax

let xhr = new XMLHttpRequest(); //创建ajax请求

xhr.open(method, url, true); //发起请求

xhr.send(null); //发送请求，发送的参数是空的

//异步回调函数

xhr.onreadystatechange = function () {

//readystate：请求返回五种状态

// 0(未初始化):还没有调用send()方法

// 1(载入):已调用send()方法，正在发送请求

// 2(载入完成):send()方法执行完成，已经接收全部响应内容

// 3(交互):正在解析响应内容

// 4(完成):响应内容解析完成，可以在客户端调用了

if (xhr.readyState == 4 && xhr.status == 200) {

resolve(xhr.responseTex t); //responseText:获取字符式的响应数据 responseXML:获取XML形式的响应数据

} else if (xhr.status == 400) {

reject('失败')

}

}

})

return p;

}

let url = "http://wthrcdn.etouch.cn/weather\_mini?city=%E5%8C%97%E4%BA%AC";

Feach('get',url).then(res => {

console.log(res)

}, err => {

console.log(err)

})

</script>

<script>

//promise异步加载图片

// 创建一个函数

function requestImg(){

// 创建一个promise对象

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

// 创建一个图像对象

var img=new Image();

// 图像在页面加载完成后执行成功的回调

img.onload=function(){

resolve(img);

}

img.src="./qzone.png"

});

return p;

}

//延时函数，用于给请求计时

function timeout(){

// 创建一个promise对象，如果请求超时则执行错误的回调

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

setTimeout(()=>{

let span=documnet.createElement('span')

span.innerText='加载失败'

resolve(span);

},5000);

});

return p;

}

// promise执行一次后如果执行成功就执行then方法，否则执行catch方法

Promise.race([requestImg(),timeout()]).then(function(results){

console.log(results);

}).catch(function(reason){

console.log(reason)

});

//上面代码requestImg函数异步请求一张图片，timeout函数是一个延时5秒的异步操作，我们将他们在一起放在race中赛跑

//如果5秒内图片请求成功那么便进入then方法，执行正常的流程。

// 如果5秒内图片还未成功返回，那么则进入catch，报“图片请求超时”的信息

</script>

#### 六、async await

**二、async/await**

Async 和 await 是一种同步的写法，但还是异步的操作，两个内容还是必须同时去写才会生效不然的话也是不会好使

1.asayc的用法，它作为一个关键字放到函数前面，这样普通函数就变为了异步函数

2.异步async函数调用，跟普通函数的使用方式一样

3.异步async函数返回一个promise对象

4.async函数配合await关键字使用(阻塞代码往下执行)是异步方法，但是阻塞式的

**优点：**

1.方便级联调用：即调用依次发生的场景；

2.同步代码编写方式： Promise使用then函数进行链式调用，一直点点点，是一种从左向右的横向写法；async/await从上到下，顺序执行，就像写同步代码一样，更符合代码编写习惯；

3.多个参数传递： Promise的then函数只能传递一个参数，虽然可以通过包装成对象来传递多个参数，但是会导致传递冗余信息，频繁的解析又重新组合参数，比较麻烦；async/await没有这个限制，可以当做普通的局部变量来处理，用let或者const定义的块级变量想怎么用就怎么用，想定义几个就定义几个，完全没有限制，也没有冗余工作；

4.同步代码和异步代码可以一起编写： 使用Promise的时候最好将同步代码和异步代码放在不同的then节点中，这样结构更加清晰；async/await整个书写习惯都是同步的，不需要纠结同步和异步的区别，当然，异步过程需要包装成一个Promise对象放在await关键字后面；

5.sync/await是对Promise的优化： async/await是基于Promise的，是进一步的一种优化，不过在写代码时，Promise本身的API出现得很少，很接近同步代码的写法；

**使用场景**：async主要来处理异步的操作，  
需求：执行第一步，将执行第一步的结果返回给第二步使用。在ajax中先拿到一个接口的返回数据，然后使用第一步返回的数据执行第二步操作的接口调用，达到异步操作。

#### 七、generator函数

#### 八、Es6中新的数据类型symbol

Symbol 值通过Symbol函数生成。这就是说，对象的属性名现在可以有两种类型，一种是原来就有的字符串，另一种就是新增的 Symbol

类型。凡是属性名属于 Symbol 类型，就都是独一无二的，可以保证不会与其他属性名产生冲突。

注意，Symbol函数前不能使用new命令，否则会报错。这是因为生成的 Symbol 是一个原始类型的值，不是对象。也就是说，由于

Symbol 值不是对象，所以不能添加属性。基本上，它是一种类似于字符串的数据类型。

### 状态码

**http状态码分类：**

100-199 提示信息 – 表示请求正在处理  
200-299 成功 – 表示请求正常处理完毕  
300-399 重定向 – 要完成请求必须进行更进一步的处理  
400-499 客户端错误 – 请求有语法错误或请求无法实现  
500-599 服务器端错误 – 服务器处理请求出错

**常见的状态码有哪些？**

200：请求成功，浏览器会把响应体内容（通常是html）显示在浏览器中；

404：(客户端问题)请求的资源没有找到

400: 语义有误，当前请求无法被服务器理解。

401: 当前请求需要用户验证

403: 服务器已经理解请求，但是拒绝执行它。

500：(服务端问题)请求资源找到了，但服务器内部发生了不可预期的错误；

301/302/303：(网站搬家了，跳转)重定向

### 当url输入到页面发生了什么

**大致过程是：**

浏览器的地址栏输入URL并按下回车，

查找当前的URL是否存在缓存，并比较缓存是否过期，

DNS解析URL对应的IP，

根据IP建立TCP连接（三次握手），

HTTP发起请求，服务器处理请求，浏览器接收HTTP响应，

渲染页面，构建DOM树，

关闭TCP连接（四次挥手）。

### 跨域

理解跨域的概念：协议、域名、端口都相同才同域，否则都是跨域

跨域就是指浏览器不允许当前页面的所在源，去请求另一个源的数据

1. 在vue 中主要是通过vue 脚手架中的vue.config.js 文件来配置的，通过在devServer中的proxy来配置跨域的前缀

2. Jsonp是解决跨域的一种解决方案，实现原理主要是利用动态创建 script 标签，设置src属性，页面要提前定义好callback。后端会返回回调函数执行，并包裹参数callback(data)

callback 中的参数就是 json

1. cors是一种跨域解决方案，它使用额外的 HTTP 头来告诉浏览器让运行在一个origin 上的Web应用被准许访问来自不同源服务器上的指定的资源。当一个资源从与该资源本身所在的服务器不同的域、协议或端口请求一个资源时，资源会发起一个跨域 HTTP 请求。

### Git

1. git init 初始化git仓库 (mac中Command+Shift+. 可以显示隐藏文件)
2. git status 查看文件状态
3. git add 文件列表 追踪文件
4. git commit -m 提交信息 向仓库中提交代码
5. git log 查看提交记录

1.分支明细

（1）主分支（master）：第一次向 git 仓库中提交更新记录时自动产生的一个分支。

（2）开发分支（develop）：作为开发的分支，基于 master 分支创建。

（3）功能分支（feature）：作为开发具体功能的分支，基于开发分支创建

2.分支命令

（1）git branch 查看分支

（2）git branch 分支名称 创建分支

（3）git checkout 分支名称 切换分支

（4）git merge 来源分支 合并分支 (备注：必须在master分支上才能合并develop分支)

（5）git branch -d 分支名称 删除分支（分支被合并后才允许删除）（-D 强制删除）

3.暂时保存更改

（1）存储临时改动：git stash

（2）恢复改动：git stash pop

**多人冲突：**

是当前修改是左箭头方向，传入的是右箭头的方向，中间用等于号分割，等号上边是当前修改，下边是传入的修改。

两人同时提交可能会出现冲突，解决办法是手动修改冲突

### 五、http和https的区别

**HTTP与HTTPS的区别**

1、HTTP是超文本传输协议，信息是明文传输，HTTPS是具有安全性的SSL加密传输协议。

2、HTTPS协议需要ca申请证书，一般免费证书少，因而需要一定费用。

3、HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样。前者是80，后者是443。

4、HTTP连接是无状态的，HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，安全性高于HTTP协议。

### 六、三次握手和四次挥手

**三次握手：**

第一次：建立连接时，客户端发送syn包到服务器，等待服务端确认

第二次：服务器收到syn包，必须确认客户的syn，同时也发送一个syn包，即syn+ACK包

第三次：客户端收到服务器的syn和ack包，向服务器发送确认包ack，发送完毕，客户端和服务端连接成功，完成三次握手

**四次挥手：**

第一次：浏览器发送完数据后，发送fin请求断开连接

第二次：服务器发送ack到客户端，确认客户端的断开请求

第三次：服务器请求断开fin的请求

第四次：客户端确认服务器的断开ack

### 七、使用基于token的登录流程

使用基于 Token 的身份验证方法，大概的流程是这样的：

1. 客户端使用用户名跟密码请求登录
2. 服务端收到请求，去验证用户名与密码
3. 验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端
4. 客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里或者 Local Storage 里
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的 Token
6. 服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据

### 八、Get和post

**Get和post的区别：**

1、Get是不安全的，因为在传输过程，数据被放在请求的URL中；Post提交的数据在HTTP包的请求包体中，对用户来说都是不可见的，相对安全。

2、Get传送的数据量较小，这主要是因为受URL长度限制；Post传送的数据量较大，一般被默认为不受限制。

4、Get执行效率却比Post方法好。Get是form提交的默认方法。

4

**Ge和post的选择：**

1.私密性的信息请求使用post（如注册、登陆）。

2.查询信息使用get。

### 九、web安全及防护

1.XSS攻击原理：

攻击者往Web页面里插入恶意 html标签或者javascript代码。

用来窃取cookie中的用户信息

解决：对一些输入的字符进行过滤，尽量采用post表单提交的方式。

2.CSRF攻击（跨站请求伪造）：

登录受信任的网站A，并在本地生成Cookie,在不登出A的情况下，携带cookie去访问危险网站B

解决：通过验证码的方式解决

3.SQL注入攻击

就是通过吧SQL命令插入到Web表单递交或输入域名，最终达到欺骗服务器执行恶意的SQL命令。

解决：表单输入时通过正则表达式将一些特殊字符进行转换

### 十、html和XML

 html被称为超文本标记语言， 是一种描述性语言，用html 可以创建能在互联网上传输的信息页，是构成网页文档的主要语言,它是由很多的标签组成

xml 即可扩展标记语言，是Internet环境中跨平台的、依赖于内容的技术，是当前处理结构化文档信息的有力工具，满足了Web内容发布与交换的需要，适合作为各种存储与共享的通用平台。

 都可以通过DOM  变成方式来访问。

 都可以通过CSS来改变外观。

 html和xml 都是标记语言，都是基于文本编辑和修改的。

xml不是要来取代html的，是对html的补充，用来与html协同工作的语言，基于上面这些优势，xml将来成为所有的数据处理和数据传输的常用工具非常可观。