|  |  |
| --- | --- |
| 前端面试 | 2019 |

# 前端高频面试题【必须记下】

## 联合编辑人:郝金利

目录

[JS相关：](#_Toc1446)

[（1） 闭包 5](#_Toc29471)

[（2） 说一下JS中的原型链的理解？ 5](#_Toc16955)

[（3）说一下JS继承（含ES6的）--或者人家这样问有两个类A和B,B怎么继承A？ 6](#_Toc21886)

[（3） 说一下JS原生事件如何绑定](#_Toc1117)

[（4） 说一下JS原生常用dom操作方法？](#_Toc15992)

**[（6） (了解)JS设计模式有哪些(单例模式观察者模式等) 8](#_Toc31218)**

[（7） 说一下你对JS面像对象的理解](#_Toc4179)

[（8） 说一下JS数组常用方法（至少6个）](#_Toc8250)

[（9） 说一下JS数组内置遍历方法有哪些和区别](#_Toc12054)

[（10） 说一下JS作用域和作用域链 10](#_Toc24509)

[（11） 说一下从输入URL到页面加载完中间发生了什么？ 11](#_Toc11077)

[（12） 说一下JS事件代理（也称事件委托）是什么，及实现原理？ 11](#_Toc4261)

[（13） 说一下JS数据类型有哪些？](#_Toc10061)

[（14） 说一下call,apply,bind区别](#_Toc29421)

[git相关： 13](#_Toc30069)

[跨域的解决方案目前有三种主流解决方案 14](#_Toc16655)

[(6) 说一下你对同步和异步的理解](#_Toc6748)

[(9) 说一下你对http状态码的了解多少？](#_Toc30996)

[(10) 你上家公司项目是前后端分离的吗](#_Toc64)

[(11) 深拷贝，浅拷贝](#_Toc5939)

[Vue相关：](#_Toc18851)

[（1） 说一下vue最大特点是什么或者说vue核心是什么](#_Toc5735)

[（2） 说一下vue常用基本指令有哪些](#_Toc26129)

[Vue 组件中 data 为什么必须是函数 19](#_Toc28316)

[（3） 说一下v-if和v-show的区别 20](#_Toc25104)

[（4） 说一下vue自定义指令如何实现的和适用场景？](#_Toc17767)

[（6） 说一下vue生命周期钩子函数有哪些，分别什么时候触发](#_Toc29562)

[（7） 说一下vue组件通讯（即传值）有哪几种形式，分别是如何实现的 23](#_Toc28008)

[（8） 说一下vue封装组件中的slot作用 24](#_Toc3547)

[（9） 说一下vue转场动画如何实现的](#_Toc29121)

[（10） 说一下你对单向数据流的理解](#_Toc26244)

[（11） 说一下vue双向数据绑定的原理 26](#_Toc5707)

[（12） 说一下vue路由或前端路由实现原理 26](#_Toc20083)

[（14） 说一下vue路由懒加载解决什么问题的? 28](#_Toc14962)

[（15） 说一下如何解决vue首屏加载慢或白屏？ 28](#_Toc14403)

[（16） 说一下vue开发环境和线上环境如何切换](#_Toc15319)

[（17） 说一下你们项目中vue如何跨域的](#_Toc12984)

[（18） 说一下vue中methods,computed,watch的区别： 30](#_Toc7982)

[（19） vue如何动态添加引用数据类型属性，实现数据响应？ 31](#_Toc9389)

[（20） vue中的http请求是如何管理的](#_Toc14586)

[（22） 说一下vue和jquey的区别](#_Toc15722)

[（23） 说一下vue如何实现局部样式的或者说如何实现组件之间样式不冲突的和实现原理是什么？](#_Toc7427)

[（24） 说一下vue第三方ui样式库如何实现样式穿透的（ui库和less/sass穿透问题） >>> /deep/](#_Toc25772)

[（25） vue目录结构（面试时可能会这样问说一下vue工程目录结构） 34](#_Toc4808)

[（26） vue脚手架是你们公司搭建的，还是用的vue的脚本架？webpack了解多少？](#_Toc4065)

[（27） 说一下你对vuex的理解： 35](#_Toc32314)

[（28） vuex如何实现数据持久化（即刷新后数据还保留）？ 35](#_Toc20081)

[（29） 说一下nextTick的作用和使用场景 36](#_Toc26490)

[v-for 与 v-if 的优先级](#_Toc11519)

[vue中 keep-alive 组件的作用 40](#_Toc24958)

[（1） css3新增特性有哪些 41](#_Toc20492)

[（2） 如何实现一个div垂直居中（至少3种方法） 41](#_Toc20151)

[（30） clearfix是解决什么问题的（另一种问法div塌陷问题如何解决的,或者说一下BFC）：](#_Toc26724)

[（31） 说一下你对盒模型的理解(包括IE和w3c标准盒模型) 42](#_Toc15424)

[（32） css3动画：](#_Toc30777)

[（33） rem和em的区别：](#_Toc3859)

[（3） 手机端如何做适配的：](#_Toc21984)

[（34） vw和vh了解吗](#_Toc23641)

[（35） cookie,localStorage和sessionStorage的区别](#_Toc381)

[IE8如何支持语义化标签](#_Toc19670)

[谷歌浏览器如何显示12px以下的字号](#_Toc19758)

## JS相关：

### 闭包

答：闭包说的通俗一点就是打通了一条在函数外部访问函数内部作用域的通道。正常情况下函数外部是访问不到函数内部作用域变量的，

表象判断是不是闭包:函数嵌套函数,内部函数被return 内部函数调用外层函数的局部变量

优点：可以隔离作用域，不造成全局污染

缺点：由于闭包长期驻留内存，则长期这样会导致内存泄露

如何解决内存泄露：将暴露全外部的闭包变量置为null

适用场景：封装组件，for循环和定时器结合使用,for循环和dom事件结合.可以在性能优化的过程中,节流防抖函数的使用,导航栏获取下标的使用

### 说一下JS中的原型链的理解？

答：原型链是理解JS面向对象很重要的一点，这里主要涉及到两个点，一是\_ \_proto\_ *，二是*prototype,举个例子吧，这样还好说点，例如：我用function创建一个Person类，然后用new Person创建一个对象的实例假如叫p1吧，在Person类的原型 prototype添加一个方法，例如：play方法,那对象实例p1如何查找到play这个方法呢，有一个查找过程，具体流程是这样的：

首先在p1对象实例上查找是否有有play方法，如果有则调用执行，如果没有则用p1.\_\_proto\_\_(\_proto\_是一个指向的作用,指向上一层的原型)往创建p1的类的原型上查找，也就是说往Person.prototype上查找，如果在Person.prototype找到play方法则执行，否则继续往上查找，则用Person.prototye.\_\_proto\_\_继续往上查找，找到Object.prototype，如果Object.prototype有play方法则执行之，否则用Object.prototype.\_\_proto\_\_继续再往上查找，但Object.prototpye.\_\_proto\_\_上一级是null,也就是原型链的顶级，结束原型链的查找，这是我对原型链的理解

（3）说一下JS继承（含ES6的）--或者人家这样问有两个类A和B,B怎么继承A？

答：JS继承实现方式也很多，主要分ES5和ES6继承的实现

先说一下ES5是如何实现继承的

ES5实现继承主要是基于prototype来实现的，具体有三种方法

一是**原型链继承**：即 B.prototype=new A()

二是借**用构造函数继承(call或者apply的方式继承)**

function B(name,age) {

A.call(ths,name,age)

}

三**是组合继承**

组合继承是结合第一种和第二种方式

再说一下ES6是如何实现继承的

ES6继承是目前比较新，并且主流的继承方式，用class定义类，用extends继承类，用super()表示父类,【下面代码部分只是熟悉，不用说课】

例如：创建A类

class A ｛

constructor() {

//构造器代码，new时自动执行

}

方法1( ) { //A类的方法 }

方法2( ) { //A类的方法 }

｝

创建B类并继承A类

class B extends A {

constructor() {

super() //表示父类

}

}

实例化B类： var b1=new B( )

b1.方法1( )

### 说一下JS原生事件如何绑定

答：JS原生绑定事件主要为三种：

一是html事件处理程序

二是DOM0级事件处理程序

三是DOM2级事件处理程序

其中：html事件现在早已不用了，就是在html各种标签上直接添加事件，类似于css的行内样式，缺点是不好维护，因为散落在标签中,也就是耦合度太高

例如：<button onclick=”事件处理函数”>点我</button>

第二类是DOM0级事件，目前在PC端用的还是比较多的绑定事件方式，兼容性也好，主要是先获取dom元素，然后直接给dom元素添加事件

例如：var btn=document.getElementById(‘id元素’)

btn.onclick=function() {

//要处理的事件逻辑

}

DOM0事件如何移除呢？很简单：btn.onclick=null;置为空就行

优点：兼容性好

缺点：只支持冒泡，不支持捕获

第三类是DOM2级事件，移动端用的比较多，也有很多优点，提供了专门的绑定和移除方法

例如： var btn=document.getElementById(‘id元素’)

//绑定事件

btn.addEventListener(‘click’,绑定的事件处理函数名,false)

//移除事件

btn.removeEventListener(‘click’,要移除的事件处理函数名，false)

优点：支持给个元素绑定多个相同事件，支持冒泡和捕获事件机制

### 说一下JS原生常用dom操作方法？

答：js原生dom操作方法有？

查找：getElementByid,

getElementsByTagName,

querySelector,

querySelectorAll

插入：appendChild,insertBefore

删除：removeChild

克隆：cloneNode

设置和获取属性：setAttribute(“属性名”,”值”),getAttibute(“属性名”)

1. **说**一下ES6新增特性?

答：ES6新增特性常用的主要有：let/const,箭头函数，模板字符串，解构赋值，模块的导入(import)和导出(export default/export),Promise,还有一些数组字符串的新方法,其实有很多，我平时常用的就这些

### (了解)JS设计模式有哪些(单例模式观察者模式等)

答：JS设计模式有很多，但我知道的有单例模式，观察者模式

单例模式：就是保证一个类只有一个实例，实现的方法一般是先判断实例存在与否，如果存在直接返回，如果不存在就创建了再返回，这就确保了一个类只有一个实例对象。在JavaScript里，单例作为一个命名空间提供者，从全局命名空间里提供一个唯一的访问点来访问该对象。

观察者模式: 观察者的使用场合就是：当一个对象的改变需要同时改变其它对象，并且它不知道具体有多少对象需要改变的时候，就应该考虑使用观察者模式。

总的来说，观察者模式所做的工作就是在解耦，让耦合的双方都依赖于抽象，而不是依赖于具体。从而使得各自的变化都不会影响到另一边的变化

### 说一下你对JS面试对象的理解

答：JS面向对象主要基于function来实现的，通过function来模拟类，通过prototype来实现类方法的共享，跟其他语言有着本质的不同，自从有了ES6后，把面向对象类的实现更像后端语言的实现了，通过class来定义类，通过extends来继承父类，其实ES6类的实现本质上是一个语法糖，不过对于开发简单了好多。

### 说一下JS数组常用方法（至少6个）

答：在开发中，数组使用频率很频繁，JS数组常用方法有:push,pop,unshift,shift,splice,join,concat,forEach,filter,map,sort,some,every好多，不过都是平时开发中很常用的方法,大家可以补充一点儿es6的

### 说一下JS数组内置遍历方法有哪些和区别

答：JS数组内置遍历（遍历就是循环的意思）方法主要有：

**forEach:**这个方法是为了取代for循环遍历数组的，返回值为undefined例如：

let arrInfo=[4,6,6,8,5,7,87]

arrInfo.forEach((item,index,arr)=>{

//遍历逻辑

})

其中：

item代码遍历的每一项,

index:代表遍历的每项的索引，

arr代表数组本身

**filter:**是一个过滤遍历的方法，如果返回条件为true，则返回满足条件为true的新数组

let arrInfo=[4,16,6,8,45,7,87]

let resultArr=arrInfo.filter((item,index,arr)=>{

//例如返回数组每项值大于9的数组

return item>9

})

**map:**这个map方法主要对数组的复杂逻辑处理时用的多，特别是react中遍历数据，也经常用到，写法和forEach类似

**some:**这个some方法用于只要数组中至少存在一个满足条件的结果，返回值就为true,否则返回fasel, 写法和forEach类似

**every:**这个every方法用于数组中每一项都得满足条件时，才返回true，否则返回false, 写法和forEach类似

### 说一下JS作用域和作用域链

答：JS作用域也就是JS识别变量的范围，作用域链也就是JS查找变量的顺序

先说作用域，JS作用域主要包括全局作用域、局部作用域和ES6的块级作用域

全局作用域：也就是定义在window下的变量范围，在任何地方都可以访问，

局部作用域：是只在函数内部定义的变量范围

块级作用域：简单来说用let和const在任意的代码块中定义的变量都认为是块级作用域中的变量，例如在for循环中用let定义的变量，在if语句中用let定义的变量等等

注：尽量不要使用全局变量，因为容易导致全局的污染，命名冲突，对bug查找不利。

2️而所谓的作用域链就是由最内部的作用域往最外部,查找变量的过程.形成的链条就是作用域链

### 说一下从输入URL到页面加载完中间发生了什么？

答：大致过程是这样的：

1. DNS解析
2. TCP连接
3. 发送HTTP请求
4. 服务器处理请求并返回需要的数据
5. 浏览器解析渲染页面
6. 连接结束

输入了一个域名,域名要通过DNS解析找到这个域名对应的服务器地址(ip),通过TCP请求链接服务,通过WEB服务器(apache)返回数据,浏览器根据返回数据构建DOM树,通过css渲染引擎及js解析引擎将页面渲染出来,关闭tcp连接

### 说一下JS事件代理（也称事件委托）是什么，及实现原理？

答：JS事件代理就是通过给父级元素（例如：ul）绑定事件，不给子级元素(例如：li)绑定事件，然后当点击子级元素时，通过事件冒泡机制在其绑定的父元素上触发事件处理函数，主要目的是为了提升性能，因为我不用给每个子级元素绑定事件，只给父级元素绑定一次就好了,在原生js里面是通过event对象的targe属性实现

var ul = document.querySelector("ul");

ul.onclick = function(e){//e指event,事件对象

var target = e.target || e.srcElement; //target获取触发事件的目标(li)

if(target.nodeName.toLowerCase() == 'li'){//目标(li)节点名转小写字母，不转的话是大写字母

alert(target.innerHTML)

}

}

jq方式实现相对而言简单 $(“ul”).on(“click”,“li”,function(){//事件逻辑}) 其中第二个参数指的是触发事件的具体目标，**特别是给动态添加的元素绑定事件，这个特别起作用**

### 说一下JS数据类型有哪些？

答：JS数据类型有：

基本数据类型：number,string,Boolean,null,undefined,symbol（ES6新增）

复合类型：Object,function

### 说一下call,apply,bind区别

答：call,apply,bind主要作用都是改变this指向的，但使用上略有区别，说一下区别:

call和apply的主要区别是在传递参数上不同，**call后面传递的参数是以逗号的形式分开的，apply传递的参数是数组形式 [Apply是以A开头的,所以应该是跟Array(数组)形式的参数]**

bind返回的是一个函数形式，如果要执行，则后面要再加一个小括号 例如：bind(obj,参数1,参数2,)(),bind只能以逗号分隔形式，不能是数组形式

### git相关：

1. 你们公司项目是如何管理的？

答：主要通过git来进行项目版本控制的

1. 说几个git常用命令？

答：我工作中常用的有git add ,git status,git commit –m,git push,git pull等

1. 说一下多人操作同一个文件，如果出现冲突该如何解决？

答：当遇到多人协作修改同一个文件时出现冲突，我先将远程文件先git pull下来，手动修改冲突代码后，再git add ,git commit,git push再上传到远程仓库。如果pull也pull不下来提示冲突的话，可以先通过git stash暂存下来，然后再pull拉取，然后git stash pop，取出原来写的，手动修改，然后提交

**Ajax相关：**

1. 说一下你是如何与后端进行数据交互的

答：我和后端通过ajax来进行数据交互的，通过统一制定的接口文档，来实现前后端高效开发，如果接口文档不能详细说明，或者接口文档上的参数请求不出数据，我会主动和后端工程师沟通，直到完成跟接口相关的业务开发。当然这其中为了验证一些接口问题，会用到一些辅助工具，比方说，runapi这种在线测试工具

1. 如果后端数据接口没有准备好，你是如何工作的

答：如果后端接口还没有准备好，我会和后端工程师沟通，通过制定接口返回数据的格式，然后前端通过一些mock数据的工具（上家公司使用的是easymock,贼简单）来批量生成假数据，可以让前端和后端同时开发，而无需等待后端数据接口写好再开发，这样提升项目整体的开发效率

1. 说一下你对ajax的同源策略的理解

答：ajax同源策略是因为安全的考虑，ajax不允许访问不同域名下的资源即所谓同源策略的意思。

1. 说一下什么情况下会产生跨域及产生跨域的解决方案和实现原理？

答：产生跨域的情况有：不同协议，不同域名，不同端口以及域名和ip地址的访问都会产生跨域。

跨域的解决方案目前有三种主流解决方案：

跨域是浏览器做出的限制,和后端没关系

1. 是jsonp

jsonp实现原理：主要是利用动态创建script标签请求后端接口地址，然后传递callback参数，后端接收callback，后端经过数据处理，返回callback函数调用的形式，callback中的参数就是json

1. 是代理（前端代理和后端代理）

前端代理我在vue中主要是通过vue脚手架中的config中的index文件来配置的，其中有个proxyTable来配置跨域的

1. 是CORS

CORS全称叫跨域资源共享，主要是后台工程师设置后端代码来达到前端跨域请求的

注：现在主流框架都是用代理和CORS跨域实现的

1. 说一下原生ajax的交互过程（即流程）

答：交互流程：

先创建XHR对象即XMLHttpRequest()

然后open准备发送，open中有三个参数一是提交方式get和post,二是接口地址，三是同步和异步

第三步是用send发送

第四步再发送的过程中通过onreadystatechange来监听接收的回调函数，可以通过判断readyState＝＝4和status=＝200来判断是否成功返回，然后通过responseText接收成功返回的数据

### 说一下你对同步和异步的理解

答：同步即sync，形象的说就是代码一行行执行，前面代码和请求没有执行完，后面的代码和请求就不会被执行，

缺点：容易导致代码阻塞

优点：程序员容易理解（因为代码从上往下一行行执行，强调顺序）

异步：即async,形象的说就是代码可以在当前程序没有执行完，也可以执行后面的代码

缺点：程序员不易理解（因为不是按顺序执行的）

优点：可以解决代码阻塞问题，提升代码执行效率和性能

异步解决方案主要有三个：

1. 回调函数
2. promise（重点掌握）
3. generator(了解)
4. async和await（重点掌握）
5. **ajax缓存如何解决？**

答：通过在文件名后面添加随机数（也称为文件指纹）来实现，主要原理是浏览器对访问过的文件，首先会检测第二次请求的文件url在浏览器是否缓存过，如果缓存过就使用，否则如果是一个新的文件url，则从服务器重新请求

1. 说一下javaScript原生,jQuery,vue,react,小程序ajax与后台交互主要用的什么技术

答：javaScript原生Ajax：用的是XMLHttpRequest对象

jQuery中的Ajax: $.ajax(),$.getJSON(),$.get(),$.post()等

vue中的Ajax:vue-resource（vue1.x中用）,axios(主流)

微信小程序Ajax:用的是小程序内置的wx.request()写法和jquery的$.ajax()类似，参数url,success，data,method,fail等

### 说一下你对http状态码的了解多少？

答：

1xx（临时响应）

表示临时响应并需要请求者继续执行操作的状态代码

2xx （成功）

表示成功处理了请求的状态码。

常见的2开头的状态码有：200 – 服务器成功返回网页

3xx （重定向）

表示要完成请求，需要进一步操作。 通常，这些状态代码用来重定向

常见的3字开头的状态码有：

**301   （永久移动）  请求的网页已永久移动到新位置。 服务器返回此响应时，会自动将请求者转到新位置。   
302   （临时移动）  服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。**304   （未修改） 自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。

4xx（请求错误） 这些状态代码表示请求可能出错，妨碍了服务器的处理。

常见的4字开头的状态有：404 – 请求的网页不存在

5xx（服务器错误）

这些状态代码表示服务器在尝试处理请求时发生内部错误。 这些错误可能是服务器本身的错误，而不是请求出错。

常见的以5开头的状态码有：  
500   （服务器内部错误）  服务器遇到错误，无法完成请求。    
503   （服务不可用） 服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。

### 你上家公司项目是前后端分离的吗

答：对，是前后端分离的项目，但开始不是，开始前后端代码混在一起写的，后来重构成前后端分离的项目，方便前端和后端自己维护的代码

注：所谓前后端分离就是前后端职责明晰：后端只提供数据接口地址返回json数据，前端用ajax请求json数据，页面的所有业务逻辑都是前端来写，让前端控制力更强，并且减轻后端逻辑处理压力。

### 深拷贝，浅拷贝

用es6的Object.assign({},{})进行对象合并，如果是数组可以用es6的Array.from，或是es6的扩展运算符...arr，如果使用es5需要用循环来做浅拷贝，如果是深拷贝需要用递归的形式来实现.当然也可以使用JSON.parse(JSON.stringify(对象))的方式实现深拷贝

## Vue相关：

### 说一下vue最大特点是什么或者说vue核心是什么

答：vue最大特点我感觉就是“组件化“和”数据驱动“

组件化就是可以将页面和页面中可复用的元素都看做成组件，写页面的过程，就是写组件，然后页面是由这些组件“拼接“起来的组件树

数据驱动就是让我们只关注数据层，只要数据变化，页面（即视图层）会自动更新，至于如何操作dom，完全交由vue去完成，咱们只关注数据，数据变了，页面自动同步变化了，很方便

### 说一下vue常用基本指令有哪些

v-if：根据表达式的值的真假条件渲染元素。在切换时元素及它的数据绑定 / 组件被销毁并重建。

v-show：根据表达式之真假值，切换元素的 display CSS 属性。

v-for：循环指令，基于一个数组或者对象渲染一个列表，vue 2.0以上必须需配合 key值 使用。

v-bind：动态地绑定一个或多个特性，或一个组件 prop 到表达式。

v-on：用于监听指定元素的DOM事件，比如点击事件。绑定事件监听器。

v-model：实现表单输入和应用状态之间的双向绑定

v-pre：跳过这个元素和它的子元素的编译过程。可以用来显示原始 Mustache 标签。跳过大量没有指令的节点会加快编译。

v-once：只渲染元素和组件一次。随后的重新渲染，元素/组件及其所有的子节点将被视为静态内容并跳过。这可以用于优化更新性能。

1.4、Vue常用的修饰符

v-on 指令常用修饰符：

.stop - 调用 event.stopPropagation()，禁止事件冒泡。

.prevent - 调用 event.preventDefault()，阻止事件默认行为。

.capture - 添加事件侦听器时使用 capture 模式。

.self - 只当事件是从侦听器绑定的元素本身触发时才触发回调。

.{keyCode | keyAlias} - 只当事件是从特定键触发时才触发回调。

.native - 监听组件根元素的原生事件。

.once - 只触发一次回调。

.left - (2.2.0) 只当点击鼠标左键时触发。

.right - (2.2.0) 只当点击鼠标右键时触发。

.middle - (2.2.0) 只当点击鼠标中键时触发。

.passive - (2.3.0) 以 { passive: true } 模式添加侦听器

注意： 如果是在自己封装的组件或者是使用一些第三方的UI库时，会发现并不起效果，这时就需要用`·.native修饰符了，如：

//使用示例：

<el-input

v-model="inputName"

placeholder="搜索你的文件"

@keyup.enter.native="searchFile(params)"

>

</el-input>

v-bind 指令常用修饰符：

.prop - 被用于绑定 DOM 属性 (property)。(差别在哪里？)

.camel - (2.1.0+) 将 kebab-case 特性名转换为 camelCase. (从 2.1.0 开始支持)

.sync (2.3.0+) 语法糖，会扩展成一个更新父组件绑定值的 v-on 侦听器。

v-model 指令常用修饰符：

.lazy - 取代 input 监听 change 事件

.number - 输入字符串转为数字

.trim - 输入首尾空格过滤

### Vue 组件中 data 为什么必须是函数

问题一：Vue 组件中 data 为什么必须是函数？

简单回答

//为什么data函数里面要return一个对象

<script>

export default {

data() {

return { // 返回一个唯一的对象，不要和其他组件共用一个对象进行返回

menu: MENU.data,

poi: POILIST.data

}

}

}

</script>

因为一个组件是可以共享的，但他们**的data是私有的，所以每个组件都要return一个新的data对象，返回一个唯一的对象，不要和其他组件共用一个对象**。

### 说一下v-if和v-show的区别

答：v-if和v-show都可以显示和隐藏一个元素，但有本质区别

v-if是惰性的，只是值为false就不会加载对应元素，为true才动态加载对应元素

v-show:是无论为true和为false都会加载对应html代码，但为false时用display:none隐藏不在页面显示，但为true时页面上用display:block显示其效果

适用场景：切换频繁的场合用v-show,切换不频繁的场合用v-if

### 说一下vue自定义指令如何实现的和适用场景？

答：哦，这个问题是这样的，vue除有了v-for,v-if等自带vue指令外,但不能满足所有的开发需求，有时需要自定义指令，自定义指令创建有全局自定义指令和局部自定义指令

全局自定义指令：Vue.directive(‘指令名’,{ inserted(el) { } })

局部自定义指令：directives:{ }

1. **说一下vue过滤器做什么的(vue1.x和vue2.x这块的区别)**

答：**vue过滤器主要用于对渲染出来的数据进行格式化处理**。例如：后台返回的数据性别用0和1表示，但渲染到页面上不能是0和1我得转换为“男“和”女”，这时就会用到过滤器，还有商品价格读取出来的是普通数值，例如：230035,但我要在前面加个货币符号和千分分隔等，例如变成：￥230，035,都得需要vue过滤器

如何创建过滤器呢，跟创建自定义指令类似，也有全局和局部过滤器的形式

**全局过滤器**：Vue.filter(‘过滤器名’,function(参数1,参数2,…) {

//………..

return 要返回的数据格式

})

**局部过滤器**：在组件内部添加filters属性来定义过滤器

fitlers:{

过滤器名(参数1,参数2,,…参数n) {

//………..

return 要返回的数据格式

}

}

### 说一下vue生命周期钩子函数有哪些，分别什么时候触发

答：vue生命周期即为一个组件从出生到死亡的一个完整周期，主要包括以下**4个阶段：创建，挂载，更新，销毁**

创建前：beforeCreate, 创建后:created

挂载前：beforeMount, 挂载后：mounted

更新前：beforeUpdate, 更新后：updated

销毁前：beforeDestroy, 销毁后：destroyed

我平时用的比较多的钩了是created和mounted，created用于获取后台数据，mounted用于dom挂载完后做一些dom操作,以及初始化插件等.beforeDestroy用户清除定时器以及解绑事件等,

另外还新增了使用内置组件 keep-alive 来缓存实例，而不是频繁创建和销毁(开销大)  
actived 实例激活  
deactived 实例失效

以下为详解版,大家理解就ok:

生命周期钩子函数（11个）Function（类型），标注蓝色的那个是属于类型的意思。

beforeCreate Function 在实例初始化之后，数据观测 (data observer) 和 event/watcher 事件配置之前被调用。

created Function 在实例创建完成后被立即调用。在这一步，实例已完成以下的配置：数据观测 (data observer)， 属性和方法的运算，watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。

beforeMount Function 在挂载开始之前被调用：相关的 render 函数首次被调用。

mounted Function el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

beforeUpdate Function 数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。该钩子在服务器端渲染期间不被调用，因为只有初次渲染会在服务端进行。

updated Function 由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

activated Function keep-alive 组件激活时调用。该钩子在服务器端渲染期间不被调用。

deactivated Function keep-alive 组件停用时调用。该钩子在服务器端渲染期间不被调用。

beforeDestroy Function 实例销毁之前调用。在这一步，实例仍然完全可用。该钩子在服务器端渲染期间不被调用。

destroyed Function Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。

errorCaptured（2.5.0+ 新增） (err: Error, vm: Component, info: string) => ?boolean 当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止该错误继续向上传播。

### 说一下vue组件通讯（即传值）有哪几种形式，分别是如何实现的

答：vue组件通讯大致有三种：父传子，子传父，还有兄弟之间通讯

**第一种：父传子：主要通过props来实现的**

具体实现：父组件通过import引入子组件，并注册，在子组件标签上添加要传递的属性，子组件通过props接收，接收有两种形式一是通过数组形式[‘要接收的属性’ ]，二是通过对象形式{ }来接收，对象形式可以设置要传递的数据类型和默认值，而数组只是简单的接收

**第二种：子传父：主要通过$emit来实现**

具体实现：子组件通过通过绑定事件触发函数，在其中设置this.$emit(‘要派发的自定义事件’，要传递的值)，$emit中有两个参数一是要派发的自定义事件，第二个参数是要传递的值

然后父组件中,在这个子组件身上@派发的自定义事件,绑定事件触发的methods中的方法接受的默认值,就是传递过来的参数

**第三种：兄弟之间传值有两种方法：**

方法一：通过event bus实现

具体实现:创建一个空的vue并暴露出去，这个作为公共的bus,即当作两个组件的桥梁，在两个兄弟组件中分别引入刚才创建的bus，在组件A中通过bus.$emit（’自定义事件名’，要发送的值）发送数据，在组件B中通过bus.$on（‘自定义事件名‘,function(v) { //v即为要接收的值 }）接收数据

方法二：通过vuex实现

具体实现：vuex是一个状态管理工具，主要解决大中型复杂项目的数据共享问题，主要包括state,actions,mutations,getters和modules 5个要素，主要流程：组件通过dispatch到 actions，actions是异步操作，再actions中通过commit到mutations，mutations再通过逻辑操作改变state，从而同步到组件，更新其数据状态

### 说一下vue封装组件中的slot作用

答：vue封装组件涉及三个东西:

一是事件(v-on,$emit)，

二是传参通过props

三是slot：slot作用主要是可以实现内容分发，组件标签内嵌套内容，可通过<slot></slot>来定义占位的内容

分为具名的slot和匿名的slot

在编写可复用组件的时候，时刻考虑组件是否可复用是有好处的。一次性组件跟其他组件紧密耦合没关系，但是可复用组件一定要定义一个清晰的公开接口。

Vue.js组件 API 来自 三部分：prop、事件、slot：

**prop 允许外部环境传递数据给组件**，在vue-cli工程中也可以使用vuex等传递数据。

**事件允许组件触发外部环境的 action(就是行为,也就是方法的意思)**

**slot 允许外部环境将内容插入到组件的视图结构内。**

### 说一下vue转场动画如何实现的

答：vue转场动画主要通过vue中的提供的transition组件实现的，例如；

<transition name=”名称”>

<router-view></router-view>

</transition>

其中name为转场的名称，自己定义，可通过定义进入和离开两种转场动画，格式为：

.名称-enter ｛ ｝ //将要进入动画

.名称-enter-active { } //定义进入的过程动画

.名称-leave { } //将要离开的动画

.名称-leave-active { } //定义离开过程中的动画

### 说一下你对单向数据流的理解

答：**单向数据流**主要是vue 组件间传递数据是单向的，即数据总是由父组件传递给子组件，子组件在其内部维护自己的数据，但它无权修改父组件传递给它的数据，当开发者尝试这样做的时候，vue 将会报错。这样做是为了组件间更好的维护。

在开发中可能有多个子组件依赖于父组件的某个数据，假如子组件可以修改父组件数据的话，一个子组件变化会引发所有依赖这个数据的子组件发生变化，所以 vue 不推荐子组件修改父组件的数据

### 说一下vue双向数据绑定的原理

答：核心主要利用ES5中的Object.defineProperty实现的，然后利用里面的getter和setter来实现双向数据绑定的，大致就这些，其实要实现起来比这个要复杂一些，不过我大致了解过。

### 说一下vue路由或前端路由实现原理

答：前端路由实现原理主要通过以下两种技术实现的

第一种：利用H5的history API实现

主要通过history.pushState 和 history.replaceState来实现，不同之处在于，pushState会增加一条新的历史记录，而replaceState则会替换当前的历史记录[发布项目时,需要配置下apache]

第二种：利用url的hash实现

我们经常在 url 中看到 #，这个 # 有两种情况，一个是我们所谓的锚点，路由里的 # 不叫锚点，我们称之为 hash，我们说的就是hash,主要利用监听哈希值的变化来触发事件 —— hashchange 事件来做页面局部更新

总结：hash 方案兼容性好，而H5的history主要针对高级浏览器。

以下为具体的API的区别:

this.$router.push(location, onComplete?, onAbort?) 这个方法会向 history 栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，则回到之前的 URL。并且点击 <router-link :to="...">等同于调用 router.push(...)。

this.$router.replace(location, onComplete?, onAbort?) 这个方法不会向 history 添加新记录，而是跟它的方法名一样 —— 替换掉当前的 history 记录，所以，当用 户点击浏览器后退按钮时，并不会回到之前的 URL。

this.$router.go(n) 这个方法的参数是一个整数，意思是在 history 记录中向前或者后退多少步，类似 window.history.go(n)。

以上也可能会形成一个新的面试题:**replace和push的区别**

**可以说,以前在一个项目里面配置了一个二级路由,里面有tab切换部分(详情,评价,说明),因为返回上一页的时候,不能在这几个选项卡之间来回切换.所以我使用了this.$router.replace方法,不计入history记录中,所以不会出现,反复切换的bug**

1. **说一下vue路由钩子（或称vue路由守卫）的理解**

什么场景下用到

答：vue路由钩子是在路由跳转过程中拦截当前路由和要跳转的路由的信息，有三种路由钩子：

第一种：全局路由钩子 beforeEach(to,from,next) { }

第二种：路由独享的钩子

beforeEnter(to,from,next) {

}

第三种：组件内的钩子

beforeRouteEnter(to,from,next) {

//…

}

beforeRouteUpdate(to,from,next) {

//…

}

beforeRouteLeave(to,from,next) {

//…

}

适用场景：动态设置页面标题，判断用户登录权限等:代码示例:

//全局路由导航守卫

vueRouter.beforeEach(function (to, from, next) {

const nextRoute = [ 'detail'];

const auth = sessionStorage.getItem("username");

let FROMPATH = from.path;

//跳转至上述3个页面

if (nextRoute.indexOf(to.name) >= 0) {

//上述数组中的路径，是相当于有权限的页面，访问数组列表中的页面就应该是在登陆状态下

if (!auth) {

let params = Object.assign({frompath:FROMPATH},from.query);

next({path: '/newlogin',query:params});

}

}

//已登录的情况再去登录页，跳转至首页

if (to.name === 'newlogin') {

if (auth) {

// vueRouter.push({name: 'index'});

next({path: '/'});

}

}

next();

});

### 说一下vue路由懒加载解决什么问题的?

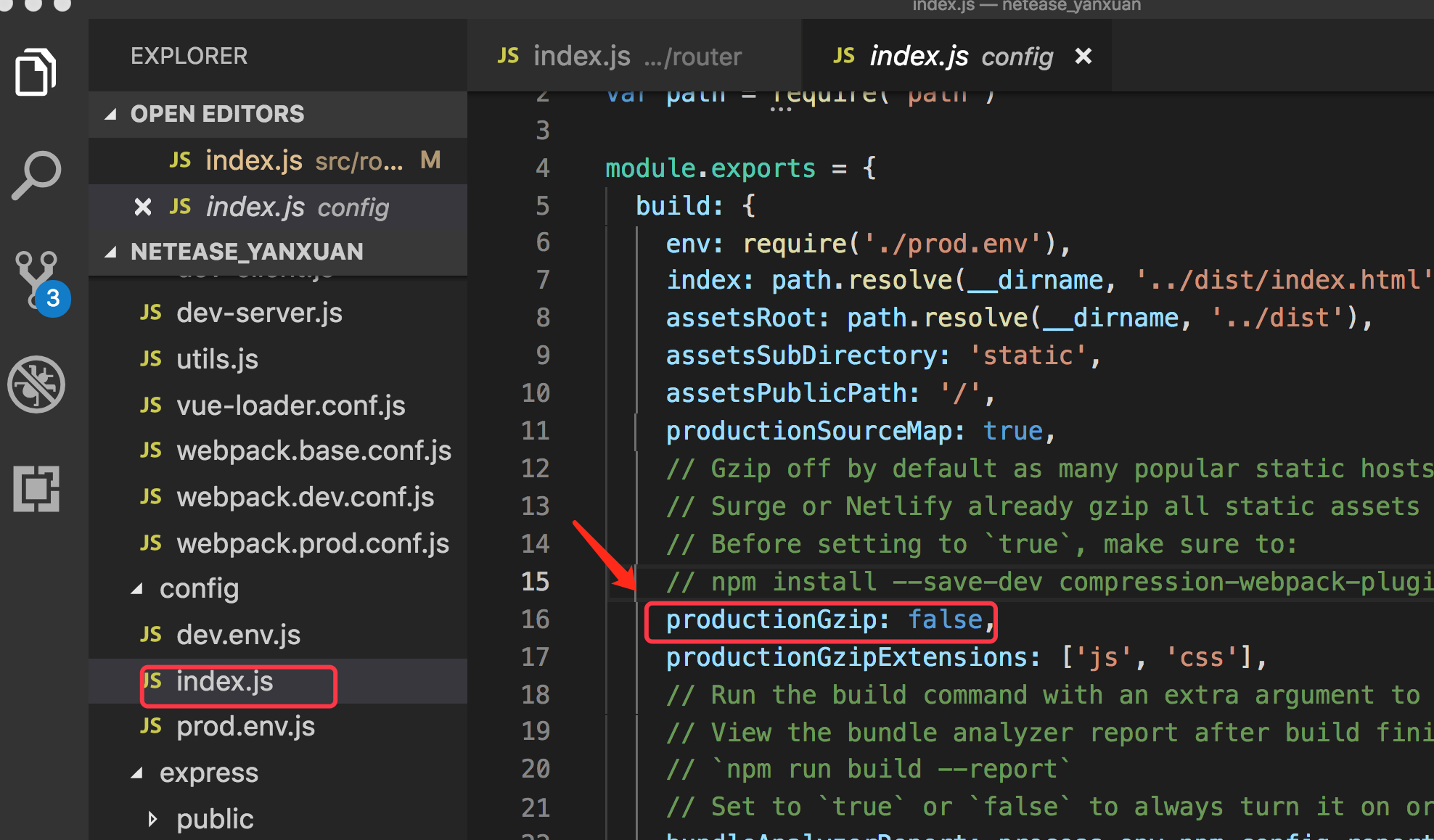
答：vue路由懒加载主要解决打包后文件过大的问题，事件触发才加载对应组件中的js

### 说一下如何解决vue首屏加载慢或白屏？

答： 1.路由懒加载



2.开启Gzip压缩



3. 使用webpack的externals属性把不需要打包的库文件分离出去，减少打包后文件的大小



4. 使用vue的服务端渲染(SSR)

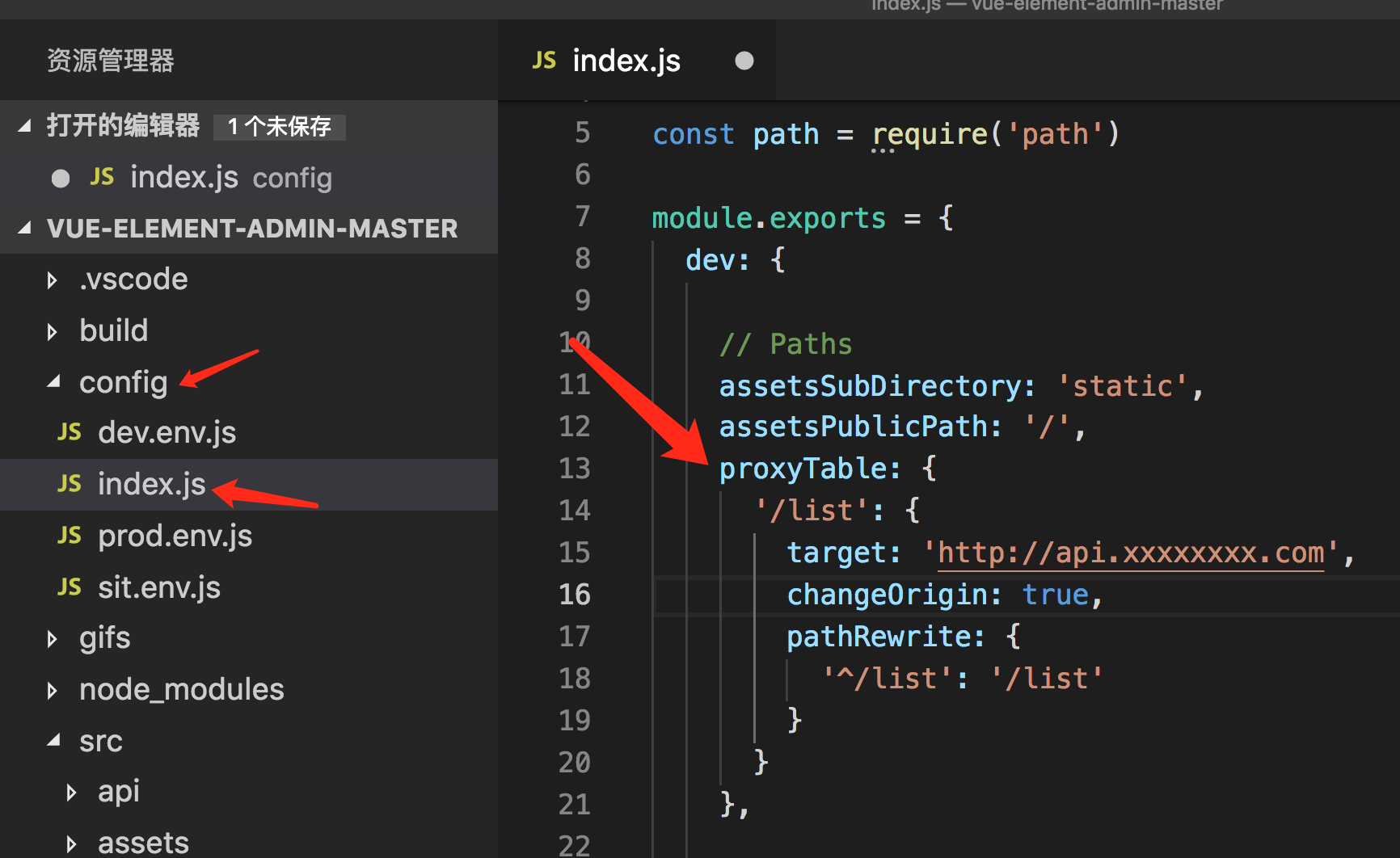
### 说一下vue开发环境和线上环境如何切换

答：主要通过检测process.env.NODE\_ENV===”production”和process.env.NODE\_ENV===”development”环境，来设置线上和线下环境地址，从而实现线上和线下环境地址的切换

### 说一下你们项目中vue如何跨域的

答：跨域前端和后端都可以实现，如果只针对vue,vue本身可以通过代理的方式可以实现，具体实现：

在config中的index.js中配置proxy来实现：



### 说一下vue中methods,computed,watch的区别：

答：methods中都是封装好的函数，无论是否有变化只要触发就会执行

computed:是vue独有的特性计算属性，可以对data中的依赖项再重新计算，得到一个新值，应用到视图中，和methods本质区别是computed是可缓存的，也就是说computed中的依赖项没有变化，则computed中的值就不会重新计算，而methods中的函数是没有缓存的。Watch是监听data和计算属性中的新旧变化。

vue用什么绑定事件，用什么绑定属性

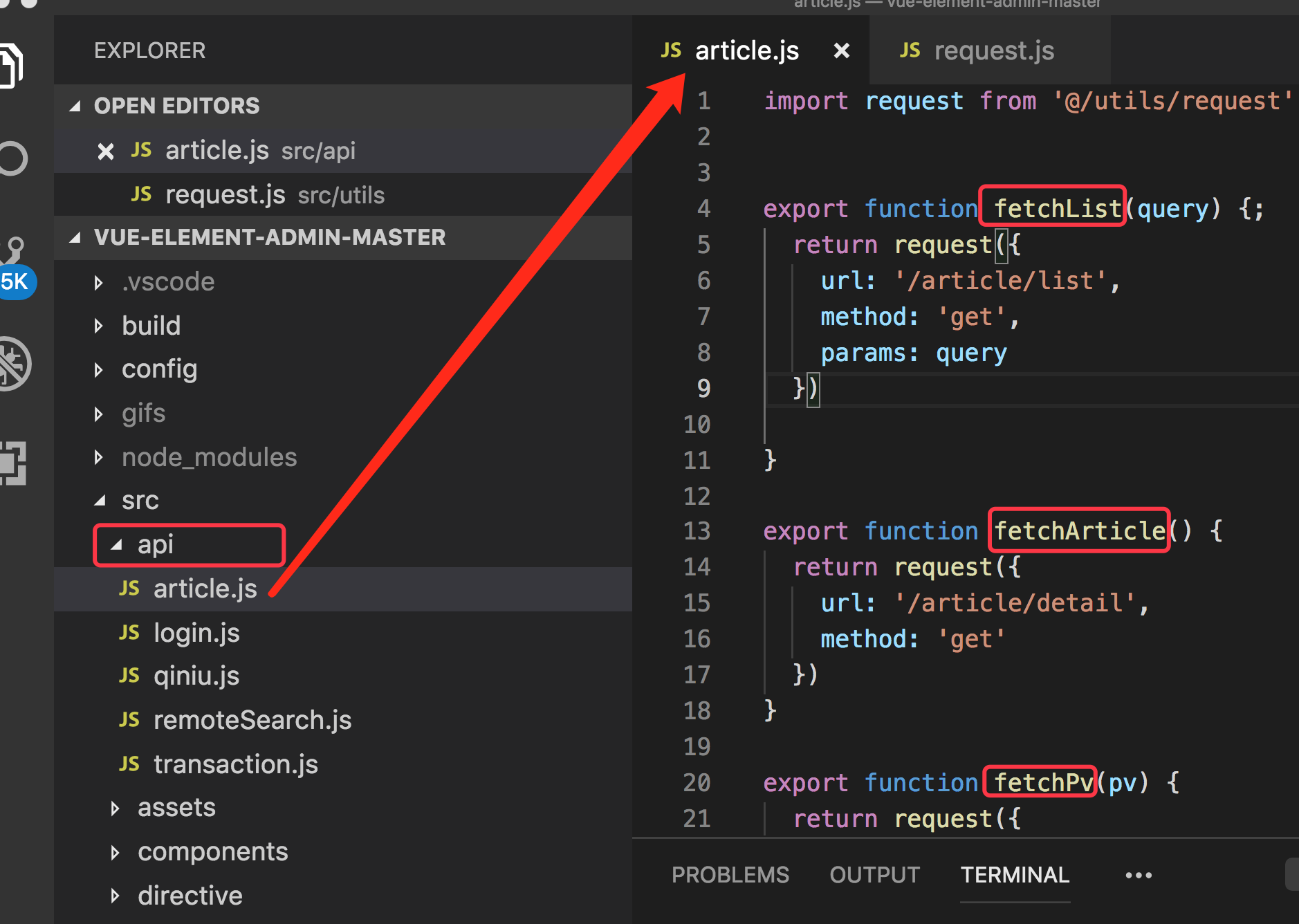
答：用v-on绑定事件，简称：@,用v-bind来绑定属性，简称：:属性

### vue如何动态添加属性，实现数据响应？

答：vue主要通过用this.$set(对象，‘属性‘，值)实现动态添加属性，以实现数据的响应注意是添加,我记忆中如果是修改引用类型属性的值,是可以自动渲染的.

### vue中的http请求是如何管理的

答：vue中的http请求如果散落在vue各种组件中，不便于后期维护与管理，所以项目中通常将业务需求统一存放在一个目录下管理，例如src下的API文件夹，这里面放入组件中用到的所有封装好的http请求并导出，再其他用到的组件中导入调用。如下面封装的HTTP请求



1. 说一下你对axios拦截器的理解：

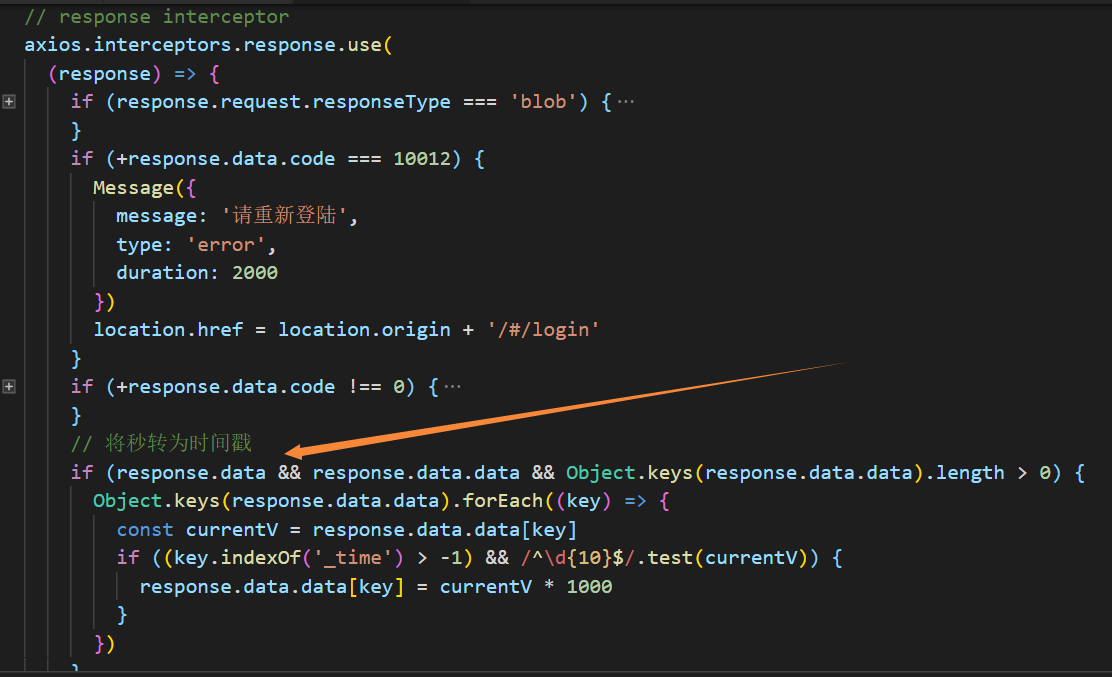
答：axios拦截器可以让我们在项目中对后端http请求和响应自动拦截处理，减少请求和响应的代码量，提升开发效率同时也方便项目后期维护

例如：





或者对公共的数据做操作



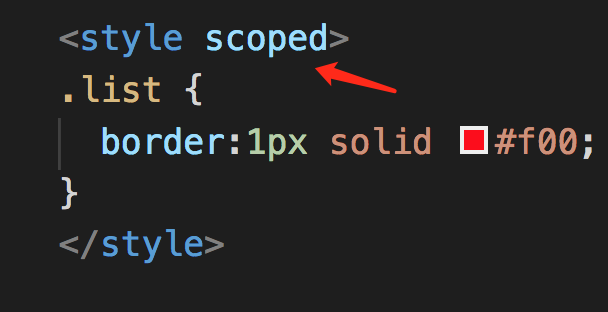
### 说一下vue和jquey的区别

答：jquery主要是玩dom操作的“神器“，强大的选择器,封装了好多好用的dom操作方法和如何获取ajax方法 例如：$.ajax（）非常好用

vue:主要用于数据驱动和组件化，很少操作dom，当然vue可能通过ref来选择一个dom或组件

### 说一下vue如何实现局部样式的或者说如何实现组件之间样式不冲突的和实现原理是什么？

答：css没有局部样式的概念，vue脚手架通过实现了，即在style标签上添加scoped



scoped的实现原理：vue通过postcss给每个dom元素添加一个以data-开头的随机自定义属性实现的

### 说一下vue第三方ui样式库如何实现样式穿透的（ui库和less/sass穿透问题） >>> /deep/



1. vue目录结构（面试时可能会这样问说一下vue工程目录结构）：这个了解下就可以

答：统一的目录结构可以方便团队协作和职责明晰，也方便项目后期维护和管理，具体vue项目目录结构包括：

build:项目构建目录

config:项目配置，包括代理配置，线上和线下环境配置

node\_modules:node包目录，npm install安装的包都在这个目录

src:平时开发时的目录

static:存入一些静态资源资源目录，我们可以把一些图片，字体，json数据放在这里。

.eslintrc.js：Eslint代码检查配置文件

.babelrc:ES6配置

.gitignore:忽略提交到远程仓库的配置

### vue脚手架是你们公司搭建的，还是用的vue的脚本架？webpack了解多少？

答：我们公司用的vue官方的脚手架（vue-cli）,vue-cli版本有3.0和2.9.x版本

webpack是一个前端模块化打包构建工具，vue脚手架本身就用的webpack来构建的，webpack本身需要的入口文件通过entry来指定，出口通过output来指定，默认只支持js文件，其他文件类型需要通过对应的loader来转换，例如：less需要less,less-loader,sass需要sass-loader,css需要style-loader,css-loader来实现。当然本身还有一些内置的插件来对文件进行压缩合并等操作

### 说一下你对vuex的理解：

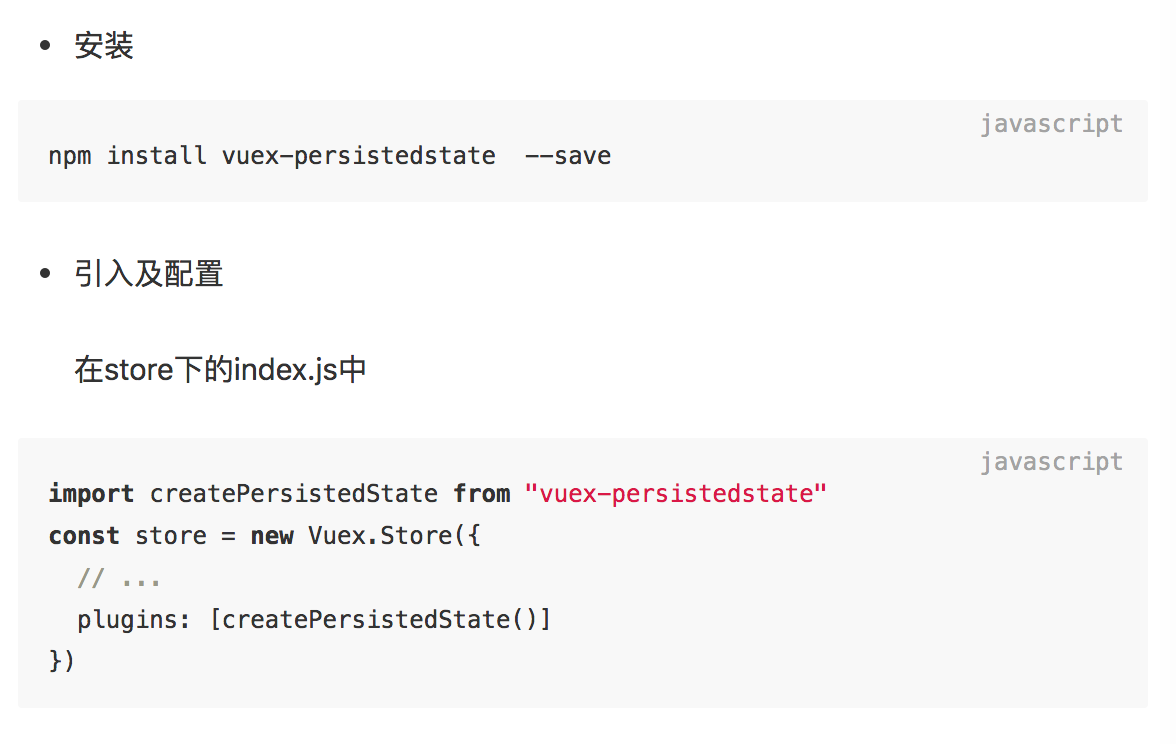
答：vuex是一个状态管理工具，主要解决大中型复杂项目的数据共享问题，主要包括state,actions,mutations,getters和modules 5个要素，主要流程：组件通过dispatch到 actions，actions是异步操作，再actions中通过commit到mutations，mutations再通过逻辑操作改变state，从而同步到组件，更新其数据状态,而getters相当于组件的计算属性对,组件中获取到的数据做提前处理的.再说到辅助函数的作用.

### vuex如何实现数据持久化（即刷新后数据还保留）？

答：因为vuex中的state是存储在内存中的，一刷新就没了，例如登录状态，解决方案有：

第一种：利用H5的本地存储(localStorage,sessionStorage)

第二种：利用第三方封装好的插件，例如：vuex-persistedstate



1. 说一下nextTick的作用和使用场景？

答：vue中的nextTick主要用于处理数据动态变化后，DOM还未及时更新的问题，用nextTick就可以获取数据更新后最新DOM的变化

适用场景：

第一种：有时需要根据数据动态的为页面某些dom元素添加事件，这就要求在dom元素渲染完毕时去设置，但是created与mounted函数执行时一般dom并没有渲染完毕，所以就会出现获取不到，添加不了事件的问题，这回就要用到nextTick处理

第二种：在使用某个第三方插件时 ，希望在vue生成的某些dom动态发生变化时重新应用该插件，也会用到该方法，这时候就需要在 $nextTick 的回调函数中执行重新应用插件的方法，例如:应用滚动插件better-scroll时





第三种：数据改变后获取焦点

**再来看以下解析:**

何为$nextTick？

简单回答：

因为Vue的异步更新队列，$nextTick是用来知道什么时候DOM更新完成的。

详细解读：

我们先来看这样一个场景：有一个div，默认用 v-if 将它隐藏，点击一个按钮后，改变 v-if 的值，让它显示出来，同时拿到这个div的文本内容。如果v-if的值是 false，直接去获取div内容是获取不到的，因为此时div还没有被创建出来，那么应该在点击按钮后，改变v-if的值为 true，div才会被创建，此时再去获取，示例代码如下：

<div id="app">

<div id="div" v-if="showDiv">这是一段文本</div>

<button @click="getText">获取div内容</button>

</div>

<script>

var app = new Vue({

el : "#app",

data:{

showDiv : false

},

methods:{

getText:function(){

this.showDiv = true;

var text = document.getElementById('div').innnerHTML;

console.log(text);

}

}

})

</script>

这段代码并不难理解，但是运行后在控制台会抛出一个错误：Cannot read property 'innnerHTML of null，意思就是获取不到div元素。这里就涉及Vue一个重要的概念：异步更新队列。

异步更新队列

**Vue在观察到数据变化时并不是直接更新DOM，而是开启一个队列，并缓冲在同一个事件循环中发生的所以数据改变。在缓冲时会去除重复数据，从而避免不必要的计算和DOM操作。然后，在下一个事件循环tick中，Vue刷新队列并执行实际（已去重的）工作**。所以如果你用一个for循环来动态改变数据100次，其实它只会应用最后一次改变，如果没有这种机制，DOM就要重绘100次，这固然是一个很大的开销。

Vue会根据当前浏览器环境优先使用原生的Promise.then和MutationObserver，如果都不支持，就会采用setTimeout代替。  
知道了Vue异步更新DOM的原理，上面示例的报错也就不难理解了。事实上，在执行this.showDiv = true时，div仍然还是没有被创建出来，直到下一个vue事件循环时，才开始创建。**$nextTick就是用来知道什么时候DOM更新完成的**，所以上面的示例代码需要修改为：

<div id="app">

<div id="div" v-if="showDiv">这是一段文本</div>

<button @click="getText">获取div内容</button>

</div>

<script>

var app = new Vue({

el : "#app",

data:{

showDiv : false

},

methods:{

getText:function(){

this.showDiv = true;

this.$nextTick(function(){

var text = document.getElementById('div').innnerHTML;

console.log(text);

});

}

}

})

</script>

这时再点击事件，控制台就打印出div的内容“这是一段文本“了。  
理论上，我们应该不用去主动操作DOM，因为Vue的核心思想就是数据驱动DOM，但在很多业务里，我们避免不了会使用一些第三方库，比如 popper.js、swiper等，这些基于原生javascript的库都有创建和更新及销毁的完整生命周期，与Vue配合使用时，就要利用好$nextTick。

v-for 与 v-if 的优先级

当它们处于同一节点，**v-for的优先级比v-if更高**，这意味着 v-if将分别重复运行于每个 v-for循环中。当你想为仅有的一些项渲染节点时，这种优先级的机制会十分有用，如下：

<li v-for="todo in todos" v-if="!todo.isComplete">

{{ todo }}

</li>

上面的代码只传递了未完成的 todos。

而如果你的目的是有条件地跳过循环的执行，那么可以将 v-if 置于外层元素 (或 <template>)上

### vue中 keep-alive 组件的作用

**keep-alive：主要用于保留组件状态或避免重新渲染。**

比如： 有一个列表页面和一个 详情页面，那么用户就会经常执行打开详情=>返回列表=>打开详情这样的话 列表 和 详情 都是一个频率很高的页面，那么就可以对列表组件使用<keep-alive></keep-alive>进行缓存，这样用户每次返回列表的时候，都能从缓存中快速渲染，而不是重新渲染。

1、属性：

include:字符串或正则表达式。只有匹配的组件会被缓存。

exclude：字符串或正则表达式。任何匹配的组件都不会被缓存。

2、用法：

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。和 <transition>相似，<keep-alive>是一个抽象组件：它自身不会渲染一DOM 元素，也不会出现在父组件链中。

当组件在<keep-alive> 内被切换，在 2.2.0 及其更高版本中，**activated 和 deactivated生命周期 将会在 树内的所有嵌套组件中触发**。

**CSS相关：**

### css3新增特性有哪些

答：css3比css2多了好多针对移动端的特性，比如：圆角：border-radius,盒阴影：box-shadow,还有动画：transition(过渡),transform（实现位移，倾斜，旋转，绽放）,animation（关键帧动画）等

### 如何实现一个div垂直居中（至少3种方法）

答：其实实现水平垂直剧中方法有很多：

第一种：定位：

第一种思路：通过给div设置绝对定位，并且left,right,top,bottom设置为0,margin:auto即可以水平垂直居中

第二种思路：通过给div设置绝对定位，left为50%,top为50%,再给div设置距左是自身的一半即：margin-left:自身宽度/2,margin-top:自身高度/2。

第三种思路：通过给div设置绝对定位，left为50%,top为50%，再给div设置跨左和跟上是自身的一半：transform:translate3d(-50%,-50%,0)

第四种：flex布局:

思路：有两个div，父级div和子级div，给父级div设置display:flex,并且设置父级div的水平居中justify-content:center，并且给父级div设置垂直居中align-items:center即可

### clearfix是解决什么问题的（另一种问法div塌陷问题如何解决的,或者说一下BFC）：

答：解决的方法有很多，主要目的是让父级元素有高度

方法一：给父级元素设置绝对定位：position:absolute

方法二：给父级元素设置overflow:hidden;

方法三：通过伪对象来实现

.clearfix:after {

 content: " ";

            display: block;

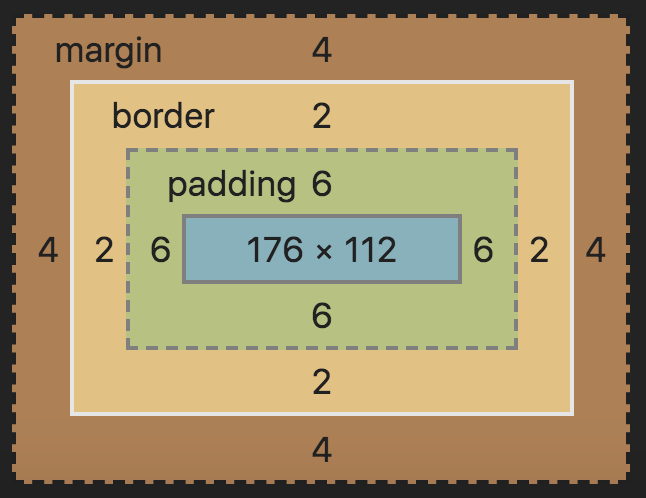
            clear: both;

             height: 0;

}

### 说一下你对盒模型的理解(包括IE和w3c标准盒模型)

答：哦，盒模型其实就是浏览器把一个个标签都看一个形象中的盒子，那每个盒子（即标签）都会有内容(width,height)，边框(border)，以及内容和边框中间的缝隙（即内间距padding），还有盒子与盒子之间的外间距（即margin）,用图表示为：



当然盒模型包括两种：IE盒模型和w3c标准盒模型

IE盒模型总宽度即就是width宽度=border+padding+内容宽度

标准盒模型总宽度＝border+padding+width

那如何在IE盒模型宽度和标准盒模型总宽度之间切换呢，可以通过box-sizing:border-box或设置成content-box来切换

其中：box-sizing：border-box //IE盒模型

box-sizing：content-box //w3c盒模型

### css3动画：

答：哦，您问的css3动画啊，css3动画大致上包括两种：

第一种：过渡动画：主要通过transition来实现，通过设置过渡属性，运动时间，延迟时间和运动速度实现。

第二种：关键帧动画：主要通过animation配合@keyframes实现

transition动画和animation动画的主要区别有两点：

第一点transition动画需要事件来触发，animation不需要

第二点:transition只要开始结束两种状态，而animation可以实现多种状态，并且animation是可以做循环次数甚至是无限运动

### rem和em的区别：

答：rem和em都是相对单位，主要参考的标签不同：

rem是相对于根字号，即相对于<html>标签的font-size实现的，浏览器默认字号是font-size:16px

em:是相对于父元素标签的字号，和百分比%类似，%也是相对于父级的，只不过是%相对于父级宽度的，而em相对于父级字号的

### 手机端如何做适配的：

答：前端做适配没有最好的方法，只有适合的方法，目前前端主要做适配的方法有：百分比，em,rem,媒体查询(即media query),flex布局（即弹性盒），vw,vh等

目前我在项目中用的多的是rem，flex布局，有时会用到媒体查询，在做pc响应式布局时用

主要是用了一个手淘的js库[flexible.js](http://caibaojian.com/t/flexible-js" \t "http://caibaojian.com/_blank" \o "View all posts in flexible.js),在页面变化时,检测页面宽度,除以10份,动态的赋值给font-size.属性.;而页面的布局我是通过rem来进行布局的,所以就可以适配所有的移动端设备了

### vw和vh了解吗

答：vw和vh是最近2年才慢慢火起来的css布局单位，现在已经被一些公司在使用，

vw和vh分别相对于屏幕宽度和屏幕高度的，1vw相当于屏幕宽度的1%,100vw相当于满屏宽度100%,

vh和vh类似，只不过是相对于屏幕高度的，1vh相当于屏幕高度的1%,100vh相当于满屏高度的100%,

你平时怎么写css（或者说佻用过css预处理器吗，用过Less和SASS吗？）

答：一般咱们说我平时写css用less，可以定义变量，嵌套，混入等功能，sass语法和less类似都是css预处理器，方便开发人员写快速高效写css

H5相关：

### cookie,localStorage和sessionStorage的区别

答：cookie可以设置失效时间，但没有自己的存取取的方法，需要时封装，每次请求时跟随请求发送,而localStorage和sessionStorage可以有自己存取的方法例如：setItem(),getItem(),removeItem(),clear() 如：localStorage.setItem(‘属性’，值) 

1. autio,video

答：H5提供的音频和视频组件，可以实现视频播放，我以前用它做过音乐播放器，不过好久没用了，里面有play,pause还有获取当前时间和总时间的方法，得看一下audio,video的API，用时我也能很快上手的

1. 基它

**兼容性相关**:

1. pc端：
   1. 一般是IE兼容的问题，例如：利用css hack来解决 \* ，\_,等
   2. JS有特性匹配来解决某个JS方法是否存在
2. 手机端：主要遇到过移动端300ms延迟和点透问题，可以通过fastclick插件解决

性能优化相关：

答：性能优化是一个很大的一个话题：优化的方面有很多，1是网络加载方面的优化：例如：减少http请求（合并文件，css雪碧图，图片懒加载），减少dom操作（dom缓存），代码封装，第三方库用cdn加载，gzip压缩等

### IE8如何支持语义化标签

一个库 html5shiv.js,直接引入就ok,大家可以百度下这个的文件,原理就是把语义化标签在低版本浏览器转化成了块级元素,让浏览器可以解析

### 谷歌浏览器如何显示12px以下的字号

中文版的chrome有个12px字体限制的问题，就是当字体小于12px时候都以12px来显示，这个问题在中文网站中并不突出，因为中文字体为了显示清晰一般都定义为大于或等于12px，但如果是一些英文网站那就不好说了，这时12px的限制就会破坏页面的美感，甚至因为文字变大而导致页面变形。

以前有个属性#chrome10px{ -webkit-text-size-adjust:none; font-size:10px; },但是新版谷歌已经不起作用了.我们可以通过css3的缩放来实现这个问题,比方说我要展示10px的文字,我可以通过设置字体20px,然后scale(0.5);

**常用算法相关**：排序，去重，地址栏解析成对象，深拷贝和浅拷贝。。。

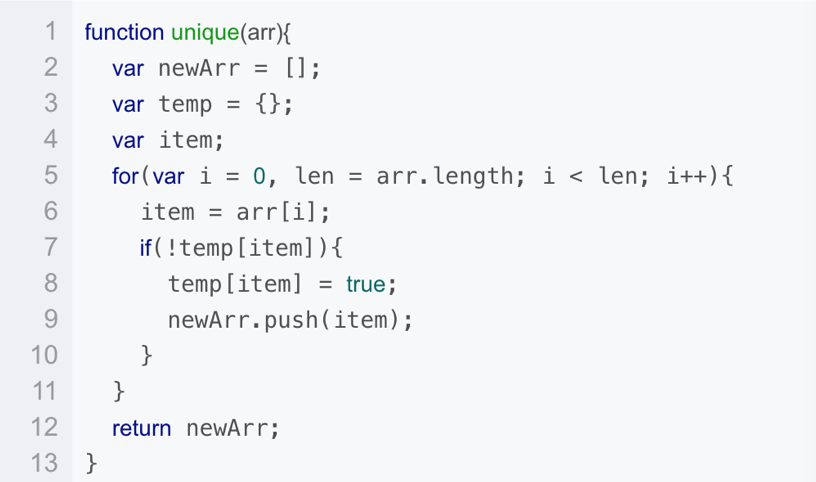
一、数组排序：



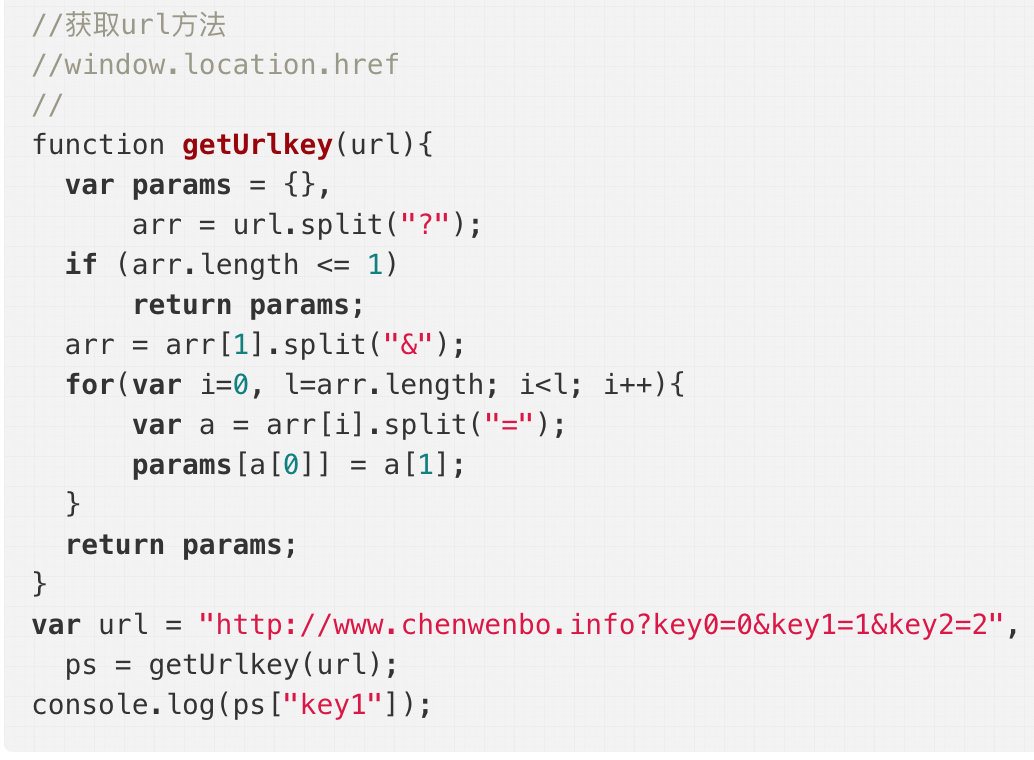
二、数组去重：

答：第一种：利用ES6的set来实现 例如：[...**new** Set(arr)]

第二种：借用临时对象的方式



三、地址栏解析成对象：

答：

四、对象深拷贝，浅拷贝的解理：

答：对象浅拷贝可以理解为改变一个对象属性值，另一个对象属性也会发生改变，即互相影响，

对象深拷贝即就是说改变一个对象属性，另一个对象属性值不会发生改变，可以通过多种方法来实现对象深拷贝，

第一种方法：通过JSON.stringify和JSON.parse来实现

var obj={name:’1610A’}

var obj2=JSON.parse(JSON.stringify(obj))

第二种方法：通过递归来实现



同学们可以参照以下代码,看下函数兼容性处理,以及封装

*var* EventUtil={

[addHandler](https://www.cnblogs.com/hykun/p/EventUtil.html" \l "addHandler):*function*(*element*,*type*,*handler*){ //添加事件

if(element.addEventListener){

element.addEventListener(type,handler,false); //使用DOM2级方法添加事件

}else if(element.attachEvent){ //使用IE方法添加事件

element.attachEvent("on"+type,handler);

}else{

element["on"+type]=handler; //使用DOM0级方法添加事件

}

},

[removeHandler](https://www.cnblogs.com/hykun/p/EventUtil.html" \l "removeHandler):*function*(*element*,*type*,*handler*){ //取消事件

if(element.removeEventListener){

element.removeEventListener(type,handler,false);

}else if(element.detachEvent){

element.detachEvent("on"+type,handler);

}else{

element["on"+type]=null;

}

},

[getEvent](https://www.cnblogs.com/hykun/p/EventUtil.html" \l "getEvent):*function*(*event*){ //使用这个方法跨浏览器取得event对象

return *event*?*event*:*window*.*event*;

},

[getTarget](https://www.cnblogs.com/hykun/p/EventUtil.html" \l "getTarget):*function*(*event*){ //返回事件的实际目标

return *event*.target||*event*.srcElement;

},

[preventDefault](https://www.cnblogs.com/hykun/p/EventUtil.html" \l "preventDefault):*function*(*event*){ //阻止事件的默认行为

if(*event*.preventDefault){

*event*.preventDefault();

}else{

*event*.returnValue=false;

}

},

[stopPropagation](https://www.cnblogs.com/hykun/p/EventUtil.html" \l "stopPropagation):*function*(*event*){ //立即停止事件在DOM中的传播

//避免触发注册在document.body上面的事件处理程序

if(*event*.stopPropagation){

*event*.stopPropagation();

}else{

*event*.cancelBubble=true;

}

}

}

**自我成长相关：**

平时怎么学习新技术的？

答：上官网看文档，上github上找相关开源项目练手，上技术社区博客园，csdn,51cto,掘金，简书参加技术问题讨论，上知乎，通过专业书籍（例如：高程三，javascript权威指南即犀牛书）系统学习，加入相关技术群参考讨论，

未来的职业规划是什么？

答：2年内先做好技术，小有所成后，其他机会也就慢慢来了

Websocket知识点:

它是一种做及时性聊天的技术,在用的过程中,首先初始化一个websocket实例,当我要发送数据的时候触发onopen方法里面的send方法,携带发送的内容,接收服务器的数据是onmessage方法,关闭本次长链接用onclose方法

这个东西其实和ajax比较像,有具体固定的格式步骤.只不是这种是长链接,ajax技术是短连接,局部刷新

**Q: CSS 有哪些样式可以给子元素继承!**

* 可继承的:font-size,font-weight,line-height,color,cursor等
* 不可继承的一般是会改变盒子模型的:display,margin、border、padding、height等

更加全面的可以到引擎找

**Q: 行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？**

* 行内: input,span,a,img以及display:inline的元素
* 块级: p,div,header,footer,aside,article,ul以及display:block这些
* void: br,hr

**Q: CSS3实现一个扇形**

* 思路跟画实体三角形一个道理,只不过多了一个圆角属性

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>扇形</title>

<style>

.sector {

width: 0;

height: 0;

border-width: 50px;

border-style: solid;

border-color: #f00 transparent transparent;

border-radius: 50px;

}

</style>

</head>

<body>

<div class="sector"></div>

</body>

</html>

**Q: box-sizing常用的属性有哪些? 分别有啥作用?**

box-sizing有两个值:content-box(W3C标准盒模型),border-box(怪异模型),

这个css 主要是改变盒子模型大小的计算形式

可能有人会问padding-box,这个之前只有 Firefox 标准实现了,目前50+的版本已经废除;

用一个栗子来距离,一个div的宽高分别100px,border为5px,padding为5px

<style>

.test {

box-sizing: content-box;

border: 5px solid #f00;

padding:5px;

width: 100px;

height: 100px;

}

</style>

<div class="test"></div>

<!--

content-box的计算公式会把宽高的定义指向 content,border和 padding 另外计算,

也就是说 content + padding + border = 120px(盒子实际大小)

而border-box的计算公式是总的大小涵盖这三者, content 会缩小,来让给另外两者

content(80px) + padding(5\*2px) + border(5\*2px) = 100px

-->

**Q: 清除浮动的方式有哪些?比较好的是哪一种?**

常用的一般为三种.clearfix, clear:both,overflow:hidden;

比较好是 .clearfix,伪元素万金油版本...后两者有局限性..等会再扯

.clearfix:after {

visibility: hidden;

display: block;

font-size: 0;

content: " ";

clear: both;

height: 0;

}

<!--

为毛没有 zoom ,\_height 这些...IE6,7这类需要 csshack 不再我们考虑之内了

.clearfix 还有另外一种写法...

-->

.clearfix:before, .clearfix:after {

content:"";

display:table;

}

.clearfix:after{

clear:both;

overflow:hidden;

}

.clearfix{

zoom:1;

}

<!--

用display:table 是为了避免外边距margin重叠导致的margin塌陷,

内部元素默认会成为 table-cell 单元格的形式

-->

clear:both:若是用在同一个容器内相邻元素上,那是贼好的...有时候在容器外就有些问题了, 比如相邻容器的包裹层元素塌陷

overflow:hidden:这种若是用在同个容器内,可以形成 BFC避免浮动造成的元素塌陷

**Q: CSS 中transition和animate有何区别? animate 如何停留在最后一帧!**

这种问题见仁见智,我的回答大体是这样的..待我捋捋.

transition一般用来做过渡的, 没时间轴的概念, 通过事件触发(一次),没中间状态(只有开始和结束)

而animate则是做动效,有时间轴的概念(帧可控),可以重复触发和有中间状态;

过渡的开销比动效小,前者一般用于交互居多,后者用于活动页居多;

至于如何让animate停留在最后一帧也好办,就它自身参数的一个值就可以了

animation-fill-mode: forwards;

<!--backwards则停留在首帧,both是轮流-->

让我们来举个栗子....自己新建一个 html 跑一下....

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Box-sizing</title>

<style>

.test {

box-sizing: border-box;

border: 5px solid #f00;

padding: 5px;

width: 100px;

height: 100px;

position:absolute;

/\*

简写的姿势排序

@keyframes name : 动画名

duration 持续时间

timing-function 动画频率

delay 延迟多久开始

iteration-count 循环次数

direction 动画方式,往返还是正向

fill-mode 一般用来处理停留在某一帧

play-state running 开始,paused 暂停 ....

更多的参数去查文档吧..我就不一一列举了

\*/

animation: moveChangeColor ease-in 2.5s 1 forwards running;

}

@keyframes moveChangeColor {

from {

top:0%;

left:5%;

background-color:#f00

}

to{

top:0%;

left:50%;

background-color:#ced;

}

}

</style>

</head>

<body>

<div class="test"></div>

</body>

</html>

**Q: 块级元素水平垂直居中的方法**

我们要考虑两种情况,定宽高和不定宽高的;

方案 N 多种,我记得我很早写过这类的笔记

传送门:[网页元素居中攻略记](https://link.juejin.im?target=http%3A%2F%2Fblog.csdn.net%2Fcolumn%2Fdetails%2Fcenter-layout.html" \t "_blank)

**Q: 说说样式权重的优先级;**

!important > 行内样式 > id > class > tag

样式权重可以叠加, 比如 id>class

**Q: 对HTML语义化的理解**

简言之:就是不滥用标签(比如 DIV)/随意嵌套(比如 span>div) ,

类的命名要合理, 利于浏览器解析乃至引擎收录,也利于团队协作和维护

**Q: JS有几种数据类型,其中基本数据类型有哪些!**

**七种数据类型**

* Boolean
* Null
* Undefined
* Number
* String
* Symbol (ECMAScript 6 新定义)
* Object

(ES6之前)其中5种为基本类型:string,number,boolean,null,undefined,

ES6出来的Symbol也是原始数据类型 ，表示独一无二的值

Object 为引用类型(范围挺大),也包括数组、函数,

**Q: null和undefined的差异**

大体说一下,想要知其所以然请引擎搜索

相同点:

* 在 if判断语句中,值都默认为 false
* 大体上两者都是代表**无**,具体看差异

差异:

* null转为数字类型值为0,而undefined转为数字类型为 NaN(Not a Number)
* undefined是代表调用一个值而该值却没有赋值,这时候默认则为undefined
* null是一个很特殊的对象,最为常见的一个用法就是作为参数传入(说明该参数不是对象)
* 设置为null的变量或者对象会被内存收集器回收

**Q: JS 的DOM 操作(Node节点获取及增删查改);**

* 获取(太多了,有document.getElementById/ClassName/Name/TagName 等,或者 querySelector)

// example

// get Node

var element = document.querySelector('#test');

// 追加

element.appendChild(Node);

// 删除

element.removeChild(Node);

// 查找

element.nextSibling // 获取元素之后的兄弟节点 , 会拿到注释文本,空白符这些

element.nextElementSibling // 等同, 获取标签(不会拿到注释文本这些)

element.previousSibling // 和上面同理,往前找兄弟节点

element.previousElementSibling

// 改动,比如 属性这些

element.setAttribute(name, value); // 增加属性

element.removeAttribute(attrName); //删除属性

// 来一个简易的练习题,随便一个网页追加插入一块DOM(非覆盖:不能 innerHTML);

/\*

<div id="test">

<span>Hello, World</span>

</div>

\*/

// 以上面的例子为例

var test = document.createElement('div'); // 创建一个块级元素

test.setAttribute("id","test"); // 设置其id 属性

var span = document.createElement('span'); // 创建一个 span

span.innerText = "Hello,world"; // 插入 span 的文本内容

test.appendChild(span); // 组合节点

element.appendChild(test); //追加到某个节点区域

**Q: 给一个 DOM添加捕获和冒泡的两种写法的事件点击,谁先执行?**

分情况分析:

* 有拿到节点的,优先捕获,没有才往上冒泡寻找
* 若是通过node.addEventListener('event',callback,bubble or capture); 谁先调用谁先执行

**stackoverflow 有相关的探讨**:

* [Event listeners registered for capturing phase not triggered before bubbling - why?](https://link.juejin.im?target=https%3A%2F%2Fstackoverflow.com%2Fquestions%2F11841330%2Fevent-listeners-registered-for-capturing-phase-not-triggered-before-bubbling-w" \t "_blank)

**Q: 谈谈你对ajax 的理解,以及用原生 JS 实现有哪些要点需要注意;**

ajax全称是异步 javascript 和 XML,用来和服务端进行数据交互的,让无刷新替换页面数据成了可能;

至于有哪些要要点,来一个简短的ajax请求

var xhr = new XMLHttpRequest(); // 声明一个请求对象

xhr.onreadystatechange = function(){

if(xhr.readyState === 4){ // readyState 4 代表已向服务器发送请求

if(xhr.status === OK){ // // status 200 代表服务器返回成功

console.log(xhr.responseText); // 这是返回的文本

} else{

console.log("Error: "+ xhr.status); // 连接失败的时候抛出错误

}

}

}

xhr.open('GET', 'xxxx');

// 如何设置请求头? xhr.setRequestHeader(header, value);

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

xhr.send(null); // get方法 send null(亦或者不传,则直接是传递 header) ,post 的 send 则是传递值

更为详细的可以阅读此处;

* [<<ajax 概念 by 阮一峰>>](https://link.juejin.im?target=http%3A%2F%2Fjavascript.ruanyifeng.com%2Fbom%2Fajax.html%23toc19" \t "_blank)
* [<<XMLHttpRequest2 用法指南>>](https://link.juejin.im?target=https%3A%2F%2Fwww.html5rocks.com%2Fzh%2Ftutorials%2Ffile%2Fxhr2%2F" \t "_blank)

**Q: JS 实现一个闭包函数,每次调用都自增1;**

这里主要考察了闭包,函数表达式以及 IIFE(立即执行表达式)

var add = (function() {

// 声明一变量,由于下面 return所以变量只会声明一次

var count = 0;

return function() {

return console.log(count++);

};

})();

add(); // 0

add(); // 1

add(); // 2

**Q: ['1','2','3'].map(parseInt) 输出什么,为什么?**

['1','2','3'].map(parseInt); // [1,NaN,NaN]

// 刨析

// map有三个参数:数组元素，元素索引，原数组本身

// parseInt有两个参数,元素本身以及进制

// 理清了这两个就好办了...

// ['1','2','3'].map(parseInt); 等于如下

['1','2','3'].map(function(item,index,array){

return parseInt(item,index); // 是不是一目了然

});

// parseInt("1",0); => 1

// parseInt("2",1); => NaN

// parseInt("3",2); => NaN

**Q: 对数组 ['2018-03-05', '2013-06-12','2019-03-12','2018-03-05','2014-02-22'] 去重且排序**

我这里用的是结合 ES6的,代码量很短

//很好理解, Set 具有值唯一性(但不是所有值,等会我抛出我的另外一篇文章)

// 结合...解构,可以把可迭代(比如 arguments/nodelist 等)的转为数组

// sort 里面传入 两个值比较,返回-1和1是因为1代表这个数大排后(相对),-1代表小(相对),0为相等

let arr = [...new Set(['2018-03-05', '2013-06-12','2019-03-12','2018-03-05','2014-02-22'])].sort(function(a,b){

return a<b ? -1:1; // 这里返回的是升序的,降序改下返回值就好了.所以是相对

})

// ["2013-06-12", "2014-02-22", "2018-03-05", "2019-03-12"]

对于数组去重的,有兴趣的可以看下我这篇水文:

* [JS数组去重!!!一篇不怎么靠谱的"深度"水文](https://link.juejin.im?target=https%3A%2F%2Fjuejin.im%2Fpost%2F5913e038570c350069235fbe" \t "_blank)

**Q: 对数组[1,2,3,4,5,'6',7,'8','a','b','z']进行乱序**

// 我们依旧可以用上面的 sort 的原理实现乱序

let tempArr = [1,2,3,4,5,'6',7,'8','a','b','z'].sort(function(){

return Math.random() > 0.5 ? -1 : 1;

})

// 因为里面有随机数,所以答案没有标准答案,我这边跑了一次是输出这个

//["6", "z", 3, "b", 5, 2, 7, "8", "a", 1, 4]

**上面和这道题逗涉及到数组顺序的问题,想了解下为什么 a-b,a>b这类可以更改排序**

可以看看知乎对于这块的探讨: [传送门:javascript排序return a-b?](https://link.juejin.im?target=https%3A%2F%2Fwww.zhihu.com%2Fquestion%2F22781233" \t "_blank)

**Q: 求[1, 10, 11, -1,'-5',12, 13, 14, 15, 2, 3, 4, 7, 8, 9]内最大值与最小值之差**

// 来一个很粗糙的版本,只当传入是数组且可以隐性转为数字的

function MaxMinPlus(arr) {

// 返回最大值与最小值之差

return Array.isArray(arr) ? Math.max.apply(Math, arr) - Math.min.apply(Math, arr) : console.log('传入的不是数组亦或者未能解决的错误')

}

// 结果是 20

// 若是要完善的话,要考虑传入的是非数组,

//传入字符串的时候要判断,然后切割为数组..

// 都要考虑进去代码量不短

**Q: 请给Array实现一个方法,去重后返回重复的字符(新数组)**

var testArr = [1,6,8,3,7,9,2,7,2,4,4,3,3,1,5,3];

Array.prototype.extraChar = function(){

var cacheExtraChar = []; // 缓存重复出现的字符

var that = this; // 缓存 this;

this.map(function(item,index){

// 怎么理解这段代码呢?

// 就是向前往后查找一遍和从后往前查找一遍,不等就是没有重复

// 为什么还要判断一遍缓存,是过滤缓存数组内多次写入

(that.indexOf(item) !== that.lastIndexOf(item)) && cacheExtraChar.indexOf(item) === -1 ? cacheExtraChar.push(item) : -1;

});

return cacheExtraChar;

}

testArr.extraChar(); // [1, 3, 7, 2, 4]

// 若是还需要排序就再排序下

[1,6,8,3,7,9,2,7,2,4,4,3,3,1,5,3]

.extraChar()

.sort(function(a,b){return a-b}) // [1, 2, 3, 4, 7]

**Q: 一个数组中 par中存放了多个人员的信息,每个人员的信息由 name 和 age 构成({name:'张三',age:15}).请用 JS 实现年龄从小到大的排序;**

var par = [{age:5,name:'张三'},{age:3,name:'李四'},{age:15,name:'王五'},{age:1,name:'随便'}]

var parSort = par.sort(function(a,b){

return a.age - b.age;

})

**Q: 判断一个回文字符串和同字母异序字符串**

* 回文字符串就是正序倒序都是一样的;
* 同字母异序字符串则是字符串都一样,但是位置可能不一定一样,比如abcefd和dceabf=>return true

后者的思路就是用排序把异序扭正..

**普通版**

// 回文判断 , 比如用 abcba

var isPalindromes = function(params){

params = params.toString().toLowerCase()

return params === params.split('').reverse().join('');

}

// 同字母异序判定,比如`abcefd`和`dceabf`

var isAnagram = function(str1, str2) {

str1 = str1.toString().toLowerCase();

str2 = str2.toString().toLowerCase();

return str1.split('').sort().join('') === str2.split('').sort().join('')

}

**进阶版:多一些特殊字符**

若是我们要去除所有非字母数字的字符,则需要用到正则

// 进阶版: isPalindromes('abc\_ &b #@a')

var isPalindromes = function(params){

// 传入参数先转为字符串且全部转为小写,最后去除多余字符比较

params = params.toString().toLowerCase().replace(/[\W\_\s]/g,'');

console.log(params)

return params === params.split('').reverse().join('');

}

// 进阶版同字母异序: isAnagram('ab \*&cef#d','!d@ce^abf')

var isAnagram = function(str1, str2) {

str1 = str1.toString().toLowerCase().replace(/[\W\_\s]/g,'');

str2 = str2.toString().toLowerCase().replace(/[\W\_\s]/g,'');

return str1.split('').sort().join('') === str2.split('').sort().join('')

}

**Q: JS 实现String.trim()方法;**

// 原生是有 trim()方法的.我们要模拟一个;

String.prototype.emuTrim = function(){

// 这条正则很好理解,就是把头部尾部多余的空格字符去除

return this.replace(/(^\s\*)|(\s\*$)/g,'');

}

' fsaf fsdaf f safl lllll '.emuTrim(); //"fsaf fsdaf f safl lllll"

**Q: JS 实现函数运行一秒后打印输出0-9;给定如下代码**

for(var i=0;i<10;i++){

// TODO

}

* 解法

// 这道题涉及到作用域

for(var i=0;i<10;i++){

setTimeout((function(i){

return function(){

console.log(i);

}

})(i),1000);

}

若是用到 ES6...那简直不能再简便了

for(let i=0;i<10;i++){

setTimeout(function(){

console.log(i);

},1000);

}

**Q: 实现对一个数组或者对象的浅拷贝和"深度"拷贝**

浅拷贝就是把属于源对象的值都复制一遍到新的对象,不会开辟两者独立的内存区域;

深度拷贝则是完完全全两个独立的内存区域,互不干扰

* 浅拷贝

// 这个 ES5的

function shallowClone(sourceObj) {

// 先判断传入的是否为对象类型

if (!sourceObj || typeof sourceObj !== 'object') {

console.log('您传入的不是对象!!')

}

// 判断传入的 Obj是类型,然后给予对应的赋值

var targetObj = sourceObj.constructor === Array ? [] : {};

// 遍历所有 key

for (var keys in sourceObj) {

// 判断所有属于自身原型链上的 key,而非继承(上游 )那些

if (sourceObj.hasOwnProperty(keys)) {

// 一一复制过来

targetObj[keys] = sourceObj[keys];

}

}

return targetObj;

}

// ES6 可以用 Object.assign(targeObj, source1,source2,source3) 来实现对象浅拷贝

* 深度拷贝

// 就是把需要赋值的类型转为基本类型(字符串这些)而非引用类型来实现

// JOSN对象中的stringify可以把一个js对象序列化为一个JSON字符串，parse可以把JSON字符串反序列化为一个js对象

var deepClone = function(sourceObj) {

if (!sourceObj || typeof sourceObj !== 'object') {

console.log('您传入的不是对象!!');

return;

}

// 转->解析->返回一步到位

return window.JSON

? JSON.parse(JSON.stringify(sourceObj))

: console.log('您的浏览器不支持 JSON API');

};

* [深拷贝的考虑点实际上要复杂的多,详情看看知乎怎么说](https://link.juejin.im?target=https%3A%2F%2Fzhuanlan.zhihu.com%2Fp%2F33489557" \t "_blank)

**Q: this对象的理解**

简言之:谁调用指向谁,运行时的上下文确定,而非定义的时候就确定;

强行绑定 this的话,可以用 call,apply,bind,箭头函数....来修改this的指向

这类的文章太多,自行搜索吧....

Q: 看到你说到 bind,能用 JS简单的模拟个么?

Function.prototype.emulateBind = function (context) {

var self = this;

return function () {

return self.apply(context);

}

}

这个实现很粗糙...更为详细全面,考虑周全的(比如参数的处理什么的)...自行谷歌.

**Q:JS 的作用域是什么?有什么特别之处么?**

作用域就是有它自身的上下文区域(比如函数内),内部会有变量声明提升,函数声明提升这些;

函数声明提升优于变量声明提升..

作用域有全局作用域和块级作用域(局部,比如用 let 或者单纯花括号的);

作用域会影响this的指向

坐等补充,我回答的时候,面试大佬只是 嗯..恩...恩...也不知道具体如何

**Q: 怎么解决跨域问题,有哪些方法...**

我一般用这三种,cors,nginx反向代理,jsonp

* jsonp : 单纯的 get 一些数据,局限性很大...就是利用script标签的src属性来实现跨域。
* nginx 反向代理: 主要就是用了nginx.conf内的proxy\_pass http://xxx.xxx.xxx,会把所有请求代理到那个域名,有利也有弊吧..
* cors的话,可控性较强,需要前后端都设置,兼容性 IE10+ ,比如
  + Access-Control-Allow-Origin: http://foo.example // 子域乃至整个域名或所有域名是否允许访问
  + Access-Control-Allow-Methods: POST, GET, OPTIONS // 允许那些行为方法
  + Access-Control-Allow-Headers: X-PINGOTHER, Content-Type // 允许的头部字段
  + Access-Control-Max-Age: 86400 // 有效期

Q: 对于想携带一些鉴权信息跨域如何走起?比如cookie!

需要配置下 header Access-Control-Allow-Credentials:true ,具体用法看下面的nginxdemo

当然cros的配置不仅仅这些,还有其他一些,具体引擎吧....

若是我们要用 nginx或者 express 配置cors应该怎么搞起? 来个简易版本的

* nginx

location / {

# 检查域名后缀

add\_header Access-Control-Allow-Origin xx.xx.com;

add\_header Access-Control-Allow-Methods GET,POST,OPTIONS;

add\_header Access-Control-Allow-Credentials true;

add\_header Access-Control-Allow-Headers DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type;

add\_header Access-Control-Max-Age 86400;

}

* express, 当然这货也有一些别人封装好的 cors中间件,操作性更强...

let express = require('express');

let app = express();

//设置所有请求的头部

app.all('\*', (req, res, next) => {

res.header("Access-Control-Allow-Origin", "xx.xx.com");

res.header("Access-Control-Allow-Headers", "DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type");

res.header("Access-Control-Allow-Credentials","true")

res.header("Access-Control-Allow-Methods","PUT,POST,GET,DELETE,OPTIONS");

next();

});

有些还会跟你死磕,,除了这些还有其他姿势么...我说了一个HTML5的postMessage....

..因为真心没用过,只是以前查阅的时候了解了下..只能大体点下

这货用于iframe 传递消息居多, 大体有这么两步步

* window打开一个实例,传递一个消息到一个x域名
* x 域名下监听message事件,获取传递的消息

这货的兼容性没那么好,而且没考虑周全下容易遭受 CSRF 攻击

**Q: 对于XSS 和 CSRF 如何防范**

这里就不说概念性的东西了

* XSS的防范
  + 我能想到的就是转义<>这些造成代码直接运行的的标签..轮询或者正则替换
    - 而面试官说这种的效率最低下,我回来仔细找了找相关资料好像没有更优方案...有的留言...
  + 若是有用到 cookie,设置为http-only,避免客户端的篡改
* CSRF的防范一般这几种
  + 验证码,用户体验虽然不好,,但是很多场合下可以防范大多数攻击
  + 验证 HTTP Referer 字段,判断请求来源
  + token加密解密,这种是目前很常用的手段了...

任何防范都有代价的,比如验证码造成的体验不好,token滥用造成的性能问题,轮询替换造成的响应时间等

**Q: 描述下cookie,sessionStorage,localSotrage的差异..**

* cookie : 大小4KB 左右,跟随请求(请求头),会占用带宽资源,但是若是用来判断用户是否在线这些挺方便
* sessionStorage和localStorage大同小异,大小看浏览器支持,一般为5MB,数据只保留在本地,不参与服务端交互.
  + sessionStorage的生存周期只限于会话中,关闭了储存的数据就没了.
  + localStorage则保留在本地,没有人为清除会一直保留

**Q: javascript的原型链你怎么理解?**

原型链算是 JS 内一种独有的机制,

所有对象都有一个内置[[proto]]指向创建它的原型对象(prototype)

原型链的基本用来实现继承用的

**Q: javascript里面的继承怎么实现，如何避免原型链上面的对象共享**

我在写的时候,用了两种,一个是 ES5和 ES6的方案

* ES5:寄生组合式继承:通过借用构造函数来继承属性和原型链来实现子继承父。

function ParentClass(name) {

this.name = name;

}

ParentClass.prototype.sayHello = function () {

console.log("I'm parent!" + this.name);

}

function SubClass(name, age) {

//若是要多个参数可以用apply 结合 ...解构

ParentClass.call(this, name);

this.age = age;

}

SubClass.prototype = Object.create(ParentClass.prototype);

SubClass.prototype.constructor = SubClass;

SubClass.prototype.sayChildHello = function (name) {

console.log("I'm child " + this.name)

}

let testA = new SubClass('CRPER')

// Object.create()的polyfill

/\*

function pureObject(o){

//定义了一个临时构造函数

function F() {}

//将这个临时构造函数的原型指向了传入进来的对象。

F.prototype = obj;

//返回这个构造函数的一个实例。该实例拥有obj的所有属性和方法。

//因为该实例的原型是obj对象。

return new F();

}

\*/

* ES6: 其实就是ES5的语法糖,不过可读性很强..

class ParentClass {

constructor(name) {

this.name = name;

}

sayHello() {

console.log("I'm parent!" + this.name);

}

}

class SubClass extends ParentClass {

constructor(name) {

super(name);

}

sayChildHello() {

console.log("I'm child " + this.name)

}

// 重新声明父类同名方法会覆写,ES5的话就是直接操作自己的原型链上

sayHello(){

console.log("override parent method !,I'm sayHello Method")

}

}

let testA = new SubClass('CRPER')

**Q: ES6+你熟悉么,用过哪些特性?**

* 箭头函数
* 类及引入导出和继承( class/import/export/extends)
* 字符串模板
* Promise
* let,const
* async/await
* 默认参数/参数或变量解构装饰器
* Array.inclueds/String.padStart|String.padEnd/Object.assign

**Q: let 和 const 有啥差异?**

* let 会产生块级作用域,不会造成变量提升,无法重新声明(但可以重新赋值);
* const
  + 是常量,若是基本数据类型,具有不变性(无法重新赋值改动)
  + 引用值可以调整内部值(可能设计的时候没有考虑周全!

**Q: async和await的用途?**

* 让 promise 的异步变成同步运行成了可能,await 可以等到 promise 执行完毕

**Q: 箭头函数的this指向谁?**

肯定很多小伙伴会说指向局部方法内!!答案是错误的...

箭头函数所改变的并非把 this 局部化，而是完全不把 this 绑定到里面去;

就是 this 是取自外部的上下级作用域(但是又不是常规 function的语法糖)..

因为箭头函数里并不支持 var self = this 或者 .bind(this) 这样的写法。

**Q: 问的时候你用过静态方法,静态属性,私有变量么?**

静态方法是ES6之后才有这么个玩意,有这么些特点

* 方法不能给 this引用,可以给类直接引用
* 静态不可以给实例调用,比如 let a = new ParentClass => a.sayHello() 会抛出异常
* 父类静态方法,子类非static方法没法覆盖父类
* 静态方法可以给子类继承
* 静态属性可以继承也可以被修改

看下面的代码..

class ParentClass {

constructor(name) {

this.name = name;

}

static sayHello() {

console.log("I'm parent!" + this.name);

}

static testFunc(){

console.log('emm...Parent test static Func')

}

}

class SubClass extends ParentClass {

constructor(name) {

super(name);

}

sayChildHello() {

console.log("I'm child " + this.name)

}

static sayHello() {

console.log("override parent method !,I'm sayHello Method")

}

static testFunc2() {

console.log(super.testFunc() + 'fsdafasdf');

}

}

ParentClass.sayHello(); // success print

let a = new ParentClass('test');

a.sayHello() // throw error

SubClass.sayHello(); // 同名 static 可以继承且覆盖

SubClass.testFunc2(); // 可以继承

let testA = new SubClass('CRPER');

**私有变量这个我没答出来,只是说了下没有private这个关键字和基本用下划线的人为区分**

所以回来只是找了下相关的资料,发现有一个比较好的模拟方案,就是WeakMap;

WeakMap可以避免内存泄露,当没有被值引用的时候会自动给内存寄存器回收了.

const \_ = new WeakMap(); // 实例化,value 必须为对象,有 delete,get,has,set四个方法,看名字都知道了

class TestWeakMap {

constructor(id, barcode) {

\_.set(this, { id,barcode });

}

testFunc() {

let { id,barcode } = \_.get(this); // 获取对应的值

return { id,barcode };

}

}

当然你也可以用Symbol来实现一个私有变量,这也是一个好法子

**Q: 谈谈你对 Promise 的理解? 和 ajax 有关系么?**

Promise和ajax没有半毛钱直接关系.promise只是为了解决"回调地狱"而诞生的;

平时结合 ajax是为了更好的梳理和控制流程...这里我们简单梳理下..

Promise有三种状态,Pending/resolve()/reject();

一些需要注意的小点,如下

* 在 Pending 转为另外两种之一的状态时候,状态不可在改变..
* Promise的 then为异步.而(new Promise())构造函数内为同步
* Promise的catch不能捕获任意情况的错误(比如 then 里面的setTimout内手动抛出一个Error)
* Promise的then返回Promise.reject()会中断链式调用
* Promise的 resolve若是传入值而非函数,会发生值穿透的现象
* Promise的catch还是then,return的都是一个新的 Promise(在 Promise 没有被中断的情况下)

Promise 还有一些自带的方法,比如race,all,前者有任一一个解析完毕就返回,后者所有解析完毕返回...

实现一个延时的 promise 函数, 可以用async和await

const delay = (time)=> new Promise((resolve,reject)=>{

setTimeout(resolve,time)

})

// test

let testRun = async function(){

console.log(1);

await delay(2000);

console.log('我两秒后才触发',3)

}

// 1 => Promise = > 3

这段代码的运行结果是什么?

var test = new Promise((resolve,reject)=>{

resolve();

});

test

.then(data => {

// promise start

console.log('promise first then : ', data);

return Promise.resolve(1); // p1

})

.then(data => {

// promise p1

console.log('get parent(p1) resolve data : ', data);

return Promise.reject(new Error('哎呀,中断了,你能奈我何!')); // p2

})

.then(data => {

// promise p2

console.log('result of p2: ', data);

return Promise.resolve(3); // p3

})

.catch(err => {

console.log('err: ', err);

return false;

});

// promise first then : undefined

// get parent(p1) resolve data : 1

// err: Error: 哎呀,中断了,你能奈我何!

// 这里在 then 返回 Promise.reject()的时候已经中断了链式调用.直接给 catch捕获到

别急,假如你不管有没有捕获到错误,最后再执行一个回调函数如何实现?

这里说的就是类似try..catch..finally,给Promise实现一个 finally;

// finally比较好加，按照现在社区的讨论，finally的特点如下：

// url : https://www.v2ex.com/t/205715

//1. 不接收任何参数，原来的value或者Error在finally里是收不到的

//2. 处理后不影响原Promise的状态，该reject还是reject，该resolve还是resolve

//3. 不影响Promise向后传递的传，resolve状态还是传递原来的value，reject状态还是传递原来的Error

Promise.prototype.finally = function (callback) {

let P = this.constructor; // 这里拿到的是 Promise 的构造函数

//不管前面的 Promise 是fulfilled还是rejected，都会执行回调函数callback。

return this.then(

value => P.resolve(callback()).then(() => value),

reason => P.resolve(callback()).then(() => { throw reason })

);

};

// 用法很简单,就是可以传入一个回调函数..

// https://developers.google.com/web/updates/2017/10/promise-finally

// 这个 url 中说了 node 及 chrome 的哪些版本已经实现了 finally 及用法

// ES 2018已经把 finally 追加到 promise 的原型链中..

* [<<Promise 必知必会（十道题)>>](https://link.juejin.im?target=https%3A%2F%2Fjuejin.im%2Fpost%2F5a04066351882517c416715d" \t "_blank): 有助于你更加深刻的了解 promise 的运行情况
* [关于 Promise 的 9 个提示](https://link.juejin.im?target=https%3A%2F%2Fjuejin.im%2Fpost%2F5a9600526fb9a06333155141" \t "_blank)
* 更多的Promise 详情可以参考[<<JavaScript Promise迷你书（中文版）>>](https://link.juejin.im?target=http%3A%2F%2Fliubin.org%2Fpromises-book%2F" \t "_blank);

**Q: 谈谈你对 TCP 的理解;**

Q: TCP 是在哪个OSI 的哪个层!通讯过程是全双工还是半双工(单工)?

A: 传输层,全双工

Q: TCP的通讯的过程是怎么样的!

A: 整个过程是三次握手,四次挥手..

Q: 你说的没错,说说整个过程如何?

A: 举个栗子,我把 TCP 比做两个人用对讲机沟通(大白话)..三次握手就是.A1(吼叫方,客户端)想要呼叫 A2(控制室的某某,服务端)..

A1对着对讲机说"over over ,听到请回答"(第一次,请求应答) ...

A2收到回应"收到收到,你说"(第二次,确认应答)

A1开始巴拉巴拉个不停而 A2没拒绝(第三次,通讯建立)

而四次挥手则是两者确认互相倾述完毕的过程..

A1说:"控制室,报告完毕了"(第一次挥手)

A2说:"知道了...那么你废话说完就好好听我指挥....巴拉巴拉.."(第二次挥手)

A1此时等待控制室说完毕,而控制室等回应(第三次挥手)

等到 A1回馈控制室确认都知道完毕了..(第四次挥手)...

**以上都是瞎掰,可能有些地方描述不当,笑笑就好了**

TCP没有百分百建立成功的,会造成链接失败的情况有很多..

比如长时间没应答(A1吼了半天没有反应或者 A2应答了而 A1不再鸟它)..亦或者丢包(对讲机也没了);

TCP 协议相关的文章网上很多,若是要更加全面的了解该协议请自行引擎..

我建议阅读<<**TCP-IP详解卷1~卷3**>>,这个是网络圣经...很厚...我只看了一丢丢..

**Q: TCP 你了解了,那么 OSI 七层协议和五层网络架构应该知道吧?**

对于这类的问题我也只能大体点了下,毕竟不是专攻网络这块的...

OSI 七层涵盖:物理层,数据链路层,网络层,传输层,会话层,表示层,应用层;

五层模型就是"会话,表示,应用层"同为一层;

Q: DNS 的大体的执行流程了解么,属于哪个层级?工作在哪个层级?

DNS 属于应用层协议, 至于TCP/UDP哪一层上面跑,看情况 , 大体的执行流程是这样的; **DNS 默认端口是53,走 UDP**

1. 优先读取浏览器缓存
2. 其次系统的缓存
3. 都没有的情况下,找本地hosts文件(比如你写了映射关系优先寻找)
4. 再没有的情况找最近的域名解析服务器
5. 再没有则扩大访问,最终找到根服务器,还是没有就失败了..

DNS 的解析的几个记录类型需要了解:

* A: 域名直接到 IP
* CNAME: 可以多个域名映射到一个主机,类似在 Github Page就用 CNAME 指向
* MX: 邮件交换记录,用的不多,一般搭建邮件服务器才会用到
* NS: 解析服务记录,可以设置权重,指定谁解析
* TTL: 就是生存时间(也叫缓存时间),一般的域名解析商都有默认值,也可以人为设置
* TXT: 一般指某个主机名或域名的说明

回来我找下相关的资料,有兴趣的可以深入了解下,传送门如下:

* [梳理Linux下OSI七层网络与TCP/IP五层网络架构](https://link.juejin.im?target=http%3A%2F%2Fwww.cnblogs.com%2Fkevingrace%2Fp%2F5909719.html" \t "_blank)
* [TCP/IP（六）应用层（DNS和HTTP协议）](https://link.juejin.im?target=https%3A%2F%2Fwww.html5rocks.com%2Fzh%2Ftutorials%2Ffile%2Fxhr2%2F" \t "_blank)
* [DNS域名解析解剖](https://link.juejin.im?target=https%3A%2F%2Fzhuanlan.zhihu.com%2Fp%2F28305778" \t "_blank)

**Q: HTTP 和 HTTPS 有何差异? 听说过 SPDY 么?**

我只是粗浅的回答了下...

HTTP相对于 HTTPS来说,速度较快且开销较小(没有 SSL/TSL) 对接,默认是80端口;

HTTP容易遭受域名劫持,而HTTPS相对来说就较为安全但开销较大(数据以加密的形式传递),默认端口为443..

HTTP是明文跑在 TCP 上.而HTTPS跑在SSL/TLS应用层之下,TCP上的

Q: 那么 HTTPS中的TLS/SSL是如何保护数据的...

一般有两种形式,非对称加密,生成公钥和私钥,私钥丢服务器,公钥每次请求去比对验证;

更严谨的采用 CA(Certificate Authority),给密钥签名....

Q: 你说到对称加密和非对称加密,能说说整个流程如何运转的么(HTTPS)

* 对称加密:
  + 双方都有同样的密钥,每次通讯都要生成一个唯一密钥,速度很快
  + 安全性较低且密钥增长的数量极快
* 非对称加密(一般用 RSA)
  + 安全性很高,对资源消耗很大(CPU),目前主流的加密算法(基本用于交换密钥或签名,而非所有通讯内容)
* CA(数字签名):
  + 这个是为了防止中间人给偷换了造成数据被窃取而诞生的
  + 用一些权威机构颁布的算法来签名,权威机构做中间人,通讯过程都会跟机构核对一遍

懂得真心不多,回来找了下相关资料,有兴趣可以点击看看;

* [深入揭秘HTTPS安全问题&连接建立全过程](https://link.juejin.im?target=https%3A%2F%2Fzhuanlan.zhihu.com%2Fp%2F22142170" \t "_blank)
* [深入理解 https 通信加密过程:口语化](https://link.juejin.im?target=https%3A%2F%2Fklionsec.github.io%2F2017%2F07%2F31%2Fhttps-learn%2F" \t "_blank) " : 看了上面那篇文章来看下面,会清晰很多

Q: SPDY 听说过么.什么来的?

谷歌推行一种协议(HTTP 之下SSL之上[TCP]),可以算是HTTP2的前身,有这么些优点

* 压缩数据(HEADER)
* 多路复用
* 优先级(可以给请求设置优先级)

而这些优点基本 HTTP2也继承下来了..

Q: 你对 HTTP 的状态吗了解多少...

这里列举一丢丢常见的..

* 1XX: 一般用来判断协议更换或者确认服务端收到请求这些
  + 100: 服务端收到部分请求,若是没有拒绝的情况下可以继续传递后续内容
  + 101: 客户端请求变换协议,服务端收到确认
* 2xx: 请求成功,是否创建链接,请求是否接受,是否有内容这些
  + 200: (成功)服务器已成功处理了请求。
  + 201: (已创建)请求成功并且服务器创建了新的资源。
  + 202: (已接受)服务器已接受请求，但尚未处理。
  + 204: (无内容)服务器成功处理了请求，但没有返回任何内容。
* 3XX: 一般用来判断重定向和缓存
  + 301: 所有请求已经转移到新的 url(永久重定向),会被缓存
  + 302: 临时重定向,不会被缓存
  + 304: 本地资源暂未改动,优先使用本地的(根据If-Modified-Since or If-Match去比对服务器的资源,缓存)
* 4XX: 一般用来确认授权信息,请求是否出错,页面是否丢失
  + 400: 请求出错
  + 401: 未授权,不能读取某些资源
  + 403: 阻止访问,一般也是权限问题
  + 404: 页面丢失,资源没找到
  + 408: 请求超时
  + 415: 媒介类型不被支持，服务器不会接受请求。
* 5XX: 基本都是服务端的错误
  + 500: 服务端错误
  + 502: 网关错误
  + 504: 网关超时

Q: HTTP的请求报文是怎么样的,能大体的说下么?

HTTP 的请求报文 = 请求行 + 请求头 + 请求体;

* 请求行: 这个好理解就是访问的方法+ 协议+ 访问的 URL 构成
* 请求头: 这个也好理解,比如 accept,content-type,user-agent这类值键对,服务端可以直接读取的
* 请求体: 比如 POST 提交的一个表单,我们编码后放在上面需要传递的

想深入了解的具体引擎搜索

Q: 请求报文知道,那你说说cookie是如何跟随请求的?

Cookie 就是保存在 HTTP 协议的请求或者应答头部(Cookie 是由服务端生成),这样一路漂泊...

Q: Cookie 隔离是什么,如何做;

cookie 隔离就是降低 header 的数据包含,以达到加快访问速度的目的

方案: 静态资源丢 CDN或者非主域来加载

**Q: 浏览器缓存和服务端的缓存控制你了解多少,说说看?**

* Last-Modified:
  + 第一次请求资源从服务器拉取的会自动带上该属性
  + 第二次请求会跟服务端比对If-Modified-Since的时间,没变动则使用本地的(状态304)
  + 结合Expires(过期时间:缓存的载止时间),跟随请求一起发出..资源没过期拿本地,否则重新请求
* Cache-control 是 HTTP1.1的东西,判断资源过期结合max-age来替代Expires[http 1.0]
* Etag:
  + 第一次请求url 时候会给服务器上标记(一串字符串)
  + 第二次请求时候会比对服务端的If-None-Match,没有改动依旧拿缓存(304)

**Q: 几个短而让我印象深刻的题**

if(!("a" in window)){

var a = 10;

}

console.log(a); // undefined

// !("a" i n window) , 返回 true

// 留言小伙伴的刨析,非常感谢,还是涉及变量提升的问题

/\*

var a;

if(!("a" in window)){

a = 10;

}

\*/

// 变种题

(function(){

var x = c = b = {a:1}

})()

console.log(x.a); // error , x is not defined

console.log(c,b) // {a: 1} {a: 1}

var count = 0;

console.log(typeof count === "number"); // true , 这个不用解释了

console.log(!!typeof count === "number"); // false

// 这里涉及到就是优先级和布尔值的问题

// typeof count 就是字符串"number"

// !!是转为布尔值(三目运算符的变种),非空字符串布尔值为 true

// 最后才=== 比较 , true === "number" , return false

(function(){

var a = b = 3;

})()

console.log(typeof a === "undefined"); // false

console.log(typeof b === "undefined"); // false

// 这里涉及的就是立即执行和闭包的问题,还有变量提升,运算符执行方向(=号自左向右)

// 那个函数可以拆成这样

(function()

var a; /\* 局部变量,外部没法访问\*/

b = 3; /\* 全局变量,so . window.b === 3 , 外部可以访问到\*/

a = b;

})()

// 若是改成这样,这道题应该是对的

console.log(typeof b === "number" && b ===3

); // true

function foo(something){

this.a = something;

}

var obj1 = {

foo:foo

};

var obj2 = {};

obj1.foo(2)

console.log(obj1.a) // 2 ,此时的 this 上下文还在 obj1内,若是 obj1.foo 先保存当做引用再执行传参,则上下文为 window

obj1.foo.call(obj2,3); // 用 call 强行改变上下文为 obj2内

console.log(obj2.a); // 3

var bar = new obj1.foo(4); // 这里产生了一个实例

console.log(obj1.a); // 2

console.log(bar.a); // 4; new的绑定比隐式和显式绑定优先级更高

function fn(){

alert(a);

var a = 200;

alert(a);

}

fn(); // undefined / 200 ; 涉及变量提升

alert(a); // undefined

var a;

alert(a); // undefined

var a = 300;

alert(a); // 300

var obj1= {

name:'obj1',

fn:function(){

console.log(this.name);

}

};

var obj2 = {name:'obj2'};

var obj3 = {name:'obj3'};

// 这道题主要涉及的是 this 指向的问题..

obj1.fn(); // obj1

var newFn = obj1.fn;

newFn(); // undefined, this 指向 window

newFn.call(obj2);// obj2, this 指向 obj2

obj3.fn = newFn;

/\*

ƒ (){

console.log(this.name);

}

\*/

obj3.fn(); // 这里指向的是 obj3 .所以输出 obj3

// 这道题来作为笔试题很绕,因为要回答的答案很多(脑海构思)..反正我是遇到了..

// 这道题主要考核的是对原型链继承这块的理解

function Parent(){

this.a = 1;

this.b = [1,2,this.a];

this.c = {demo:5};

this.show = function(){

console.log(this.a + '' + this.c.demo + ':' + this.b)

}

}

function Child(){

this.a = 2;

this.change = function(){

this.b.push(this.a);

this.a = this.b.length;

this.c.demo = this.a++;

}

}

Child.prototype = new Parent();

var parent = new Parent();

var child1 = new Child();

var child2 = new Child();

child1.a = 11;

child2.a = 12;

// 这前面几个还算简单,继续看下去

parent.show(); // 15:1,2,1

// 因为 Child 自身没有 show 的方法,所以往原型链的上游找;

// 找到父类的,this 因为没更改,所以输出结果如下

child1.show(); // 115:1,2,1

child2.show(); // 125:1,2,1

child1.change(); // 改变一些数据,没有输出

child2.change(); // +1

parent.show(); // 15:1,2,1

child1.show(); // 55:1,2,1,11,12

child2.show(); // 65:1,2,1,11,12

// 这道题也很绕,函数递归调用的

function test(a,b){

console.log(b);

return {

test:function(c){

return test(c,a);

}

};

// 这道题的理解,拆成这样就好理解了

/\*function test(a,b){

console.log("a:"+a,"b:"+b);

return {

test:function(c){

console.log("a:"+a,"b:"+b,"c"+c);

return test(c,a);

}

}

}\*/

var a = test(100); // undefined, 这个是不言而喻的;

a.test(200); // 100;

a.test(300); // 100;

var b = test(101).test(201).test(301); // undefined/101/201

var c = test(102).test(202); // undefined / 102

c.test(302); // 202

**Q:有字符串 var test='abc345efgabcab'; 请根据提示实现对应要求**

* 去掉字符串中的 a,b,c 字符 ,形成结果'345efg';

test.replace(/[abc]/g,''); // "345efg"

* 将字符串的数字用括号括起来, 形成结果: abc[3][4][5]efg....'

test.replace(/\d/g,'[$&]'); // "abc[3][4][5]efgabcab"

// 若是有分组则按照$1, $2, $3的形式进行引用，而 $& 则表示的是整个正则表达式匹配的内容。

* 将字符串中的每个数字的值分别乘以2,输出:'abc6810....'

var temp = test.split('').map(function(item){

return /^\d$/.test(item) ? item \* 2 : item;

}).join('');

// "abc6810efgabcab"

**Q: 使用不少于三种方式替换文本"dream"改成"package",提供字符串"I have a dream";**

* 正则替换

// 这是最简单的代码量了..

var str = "I have a dream";

str.replace(/dream/g,"package");

// 不用正则也可以直接字符串替换

str.replace("dream","package")

* 数组遍历更改

// 很直白的大脑回路

var str = "I have a dream";

str.split(" ").map(function(item){

return item === "dream" ? item = "package":item;

}).join(" ");

* 数组查询切割法

var str = "I have a dream";

var tempArr = str.split(" "); // ["I", "have", "a", "dream"]

var removeIndex = tempArr.indexOf('dream'); // 3

tempArr.splice(removeIndex,1,"package");

var transStr = tempArr.join(" "); // "I have a package";

**这类东东弄成数组还是挺好弄的**

这个是留言区小伙伴提供的方法..大同小异,如下;

// 源代码

// 字符串也有数组的 slice 以及 concat 的方法..思路和数组差不多

var str = 'I haved a dream';

str.indexOf('dream') !== -1 ? str.slice(0,str.indexOf('dream')).concat('package'):str;

**Q: 还有一道题目是涉及事件循环,执行优先权的..**

就是 macrotask和microtask 相关的, 具体记不起来了..那时候给了答案虽然对了.

要说出所以然,给秀了一脸..回来找了下相关的资料;

* [JavaScript 运行机制详解：再谈Event Loop](https://link.juejin.im?target=http%3A%2F%2Fwww.ruanyifeng.com%2Fblog%2F2014%2F10%2Fevent-loop.html" \t "_blank)
* [深入理解事件循环和异步流程控制](https://link.juejin.im?target=https%3A%2F%2Fjuejin.im%2Fpost%2F5a2e21486fb9a0450407d370" \t "_blank)
* [所有你需要知道的关于完全理解 Node.js 事件循环及其度量](https://link.juejin.im?target=https%3A%2F%2Fjuejin.im%2Fpost%2F5984816a518825265674c8f6" \t "_blank)

**Q: 你对基础算法这块掌握的如何....**

来,这纸给你,写个快排试试...

// 快排的大体思路是这样的,

// 找个中位值,从原数组切割出来,

// 剩下的作为两个数组,每次都去比较;

// 直到递归的结果出来, 平均复杂度O(nlog n)

function quickSort(arr) {

//如果数组长度<=1,则直接返回

if (arr.length <= 1) {

return arr;

}

// 中间位(基准)取长度的一半向下取整

var pivotIndex = Math.floor(arr.length / 2);

//把中间位从原数组切割出来, splice 会改变原数组!!!!

var pivot = arr.splice(pivotIndex, 1)[0];

//定义两个空数组来存放比对后的值

var left = [];

var right = [];

//比基准小的放在left，比基准大的放在right

for (var i = 0 , j = arr.length; i < j; i++) {

if (arr[i] <= pivot) {

left.push(arr[i]);

} else {

right.push(arr[i]);

}

}

//递归下去 arr = [ left , pivot , right]

// 怎么个递归法,就是比对后的数组还是会重复之前的取基准再切开比较..直到最后没有可以切了

return quickSort(left).concat([pivot], quickSort(right));

}

Q: 写一个二分法查找

// 二分法跟快排的思路差不多,对半比较

// 这个只用于排序好数组内的查询,高低位都知道的情况下

function binSearch(target, arr, start, end) {

var start = start || 0; // 允许从什么位置开始,下标

var end = end || arr.length - 1; // 什么位置结束,下标

start >= end ? -1 : ''; // 没有找到,直接返回-1

var mid = Math.floor((start + end) / 2); // 中位下标

if (target == arr[mid]) {

return mid; // 找到直接返回下标

} else if (target > arr[mid]) {

//目标值若是大于中位值,则下标往前走一位

return binSearch(target, arr, start, mid - 1);

} else {

//若是目标值小于中位值,则下标往后退一位

return binSearch(target, arr, mid + 1, end);

}

}

// binSearch(5,[1,2,3,4,5,6,7,8]) => 4

// 无序的数组则需要先排序好数组,否则会堆栈溢出(死循环)

这类的文章很多,有兴趣的可以阅读下面的一些文章

传送门:

* [<<十大经典排序算法总结（JavaScript描述>>](https://link.juejin.im?target=https%3A%2F%2Fjuejin.im%2Fpost%2F57dcd394a22b9d00610c5ec8" \t "_blank)
* [JavaScript数据结构和算法](https://link.juejin.im?target=https%3A%2F%2Fgithub.com%2Fzoro-web%2Fblog%2Fissues%2F4" \t "_blank)
* [javascript 常见排序算法](https://link.juejin.im?target=http%3A%2F%2Fwww.cnblogs.com%2Fxianyulaodi%2Fp%2F6001122.html" \t "_blank)

**Q: 设计模式你了解多少?**

* [Javascript常用的设计模式详解](https://link.juejin.im?target=http%3A%2F%2Fwww.cnblogs.com%2Ftugenhua0707%2Fp%2F5198407.html" \t "_blank)
* [js-design-pattern](https://link.juejin.im?target=https%3A%2F%2Fgithub.com%2FToNiQian%2Fjs-design-pattern" \t "_blank)

**Q: 思维拓展题: 你有两个玻璃球,有个100米的高楼,求玻璃球在哪个楼层扔下会碎(用的次数最少);**

问题的要点: 玻璃球碎(有限个数) ,确定楼层数 , 最少次数 => 就是求最优的公式

在这道题上给秀的一脸,我的第一次的思路

先折半,就变成[1-50][51-100], 那就是 1+50 = 51次 ...

面试大佬说,你用了快排的思路就肯定不是最优的..

憋了许久,想到开平方 , 这样的话,最多只要20次

然后又说给我三个球,在1000米的高楼,判断多少次...但是根据我上面的话,

开立方, , 那最多不超过30次;

至于第一次丢球的位置如何确定, 就是开平之后的值作为一个区间.

若 N 个球和 M 米的大厦...第一次丢球的高度区间就是这个了

面试大佬说这个还可以...那就暂且告一段落

...回来用万能的搜索引擎找了下..最优方案+最少次数需要考虑的东西很多,没那么简单

传送门: [知乎有人讨论了这个问题](https://link.juejin.im?target=https%3A%2F%2Fwww.zhihu.com%2Fquestion%2F31855632" \t "_blank);

但是高数还老师了..这种帖子看的一脸懵逼....抽空再好好研究下

**Q: 你对优化这块了解多少?**

大体常见的手段了解.

比如从客户端着手的:

* 压缩代码(JS/CSS),压缩图片
* 合并一些小图片(css sprite)
* 若是打包的代码尽可能切割成多个 chunk,减少单一 chunk过大
* 静态文件采用 cdn 引入
* HTTP的缓存头使用的合理
* 减小第三方库的依赖
* 对于代码应该考虑性能来编写,比如使用requestAnimationFrame绘制动画,尽可能减少页面重绘(DOM 改变)
* 渐进升级,引入preload这些预加载资源
* 看情况用service worker来缓存资源(比如移动端打算搞 PWA)

比如从服务端着手:

* 带宽,域名解析, 多域名解析等
* 页面做服务端渲染,减小对浏览器的依赖(不用客户端解析)
* 渐进升级,比如引入 HTTP2(多路复用,头部压缩这些可以明显加快加载速度)

当然,这是这些都是很片面的点到...实际工作中去开展要复杂的多;

比如我们要多个维度去考虑的话,要去优化 DOM 的绘制时间,资源的加载时间,域名解析这些;

要全面的优化一个项目是一个大工程...

**Q: MySQL有哪些索引类型? 索引的数据结构储存方式? MySQL和 MongoDB的差异**

MySQL索引类型:

* 普通索引: 就普通的类型
* 唯一索引: 代表索引的值唯一不重复(允许有空值),相对于上面多了个UNIQUE
* 主键索引:(创建表的跟随创建,唯一索引，不允许有空值)
* 组合索引(就是将多个字段都建立到一个索引)

索引有利有弊,用的好加快查询速度,滥用索引会造成大量磁盘空间占用,维护性也会增多; 索引不会包含null的列;

索引的数据结构储存方式,我只简单了解过B-Tree

至于MySQL 和 MongoDB的差异;

前者是关系型数据库, 后者非关系型数据库(数据是以文档的方式储存,值为 key-value);

MySQL应用层面很广,有事务系统这些,链表查询这些都很方便.经常作为很多系统的主力数据库

而MongoDB作为NoSQL,虽然有些层面不如 MySQL,但是应用层面也挺广, 比如结合前端做一些用户的概要信息的维护,一些缓存信息的维护.

em....后端了解不多,也能点到即止....大学的时候学过一些..都差不多还给老师....

**Q: JS时间分段**

给定一个时间段和步长，枚举该时间段内步长的划分

例如：时间段3:00-5:00，步长为20分钟

那么返回的数组为

['3:00-3:20', '3:20-3:40'....]等

这类问题,一般都要先梳理好思路再来写;

* 给定字符串时间段,切割,转换为分钟
* 跨日及跨时问题

// 这个东东我的小伙伴也写出来了.我的是在它的解答方式上加以注释和对参数的判断做了考虑

// 他的解法方案在他的 github 上 https://github.com/lyh2668/blog/issues/1 , by lyh2668

// 方便一些小伙伴的理解,以下代码包含ES6的姿势(参数默认值,剪头函数)

let inputDateRange = (date, step = 30, separator = '-') => {

let startTime, endTime; // 开始时间和结束时间

if (Object.prototype.toString.call(date) === '[object String]') {

date = date.trim(); // 去除两边的空格

var tempDate = '';

if (separator) {

tempDate = date.split(separator);

} else {

if (date.indexOf('-') !== -1) {

tempDate = date.split('-');

} else if (date.indexOf('~')) {

tempDate = date.split('~');

} else {

console.log('您传入的也许不是一个时间段!!!');

}

}

startTime = time2min(tempDate[0]); // 传入的开始时间

endTime = time2min(tempDate[1]); //传入的结束时间

} else if (Object.prototype.toString.call(date) === '[object Array]') {

if (date.length === 2) {

startTime = time2min(date[0]); // 传入的开始时间

endTime = time2min(date[1]); //传入的结束时间

}

} else {

console.log('您传入的也许不是一个时间段!!!');

}

// 传入的 step 是否为数字,否则截图数字部分转化

// 为什么和 NaN 比较(自身不等性),若是传入的连正则都没法识别,那只能给默认值了

Object.prototype.toString.call(step) === '[object Number]'

? (step = parseInt(step, 10))

: parseInt(step.replace(/[W\s\b]/g, ''), 10) === NaN

? (step = parseInt(step.replace(/[W\s\b]/g, ''), 10))

: (step = 30);

// 若是开始时间大于结束时间则结束时间往后追加一天

startTime > endTime ? (endTime += 24 \* 60) : '';

let transformDate = []; // 储存转换后的数组,时间分段

// 开始遍历判断,用 while

while (startTime < endTime) {

// 如果开始时间+步长大于结束时间,则这个分段结束,否则结束时间是步长递增

let right = startTime + step > endTime ? endTime : startTime + step;

transformDate.push(`${min2time(startTime)}-${min2time(right)}`);

startTime += step; // 步长递增

}

return transformDate;

};

// 时间转化为分钟

let time2min = time => {

// 获取切割的

time.indexOf(':') ? (time = time.trim().split(':')) : '';

return time[0] \* 60 + parseInt(time[1]); // 返回转化的分钟

};

// 分钟转会字符串时间

let min2time = minutes => {

let hour = parseInt(minutes / 60); // 返回多少小时

let minute = minutes - hour \* 60; // 扣除小时后剩余的分钟数

hour >= 24 ? (hour = hour - 24) : ''; // 若是大于等于24小时需要扣除一天得到所剩下的小时

minute < 10 ? (minute = '0' + minute) : ''; // 小于10的都要补零

hour < 10 ? (hour = '0' + hour) : ''; // 小于10的都要补零

return `${hour}:${minute}`;

};

// test ,支持字符串传入时间段

inputDateRange('3:00-5:00','20d'); // ["03:00-03:20", "03:20-03:40", "03:40-04:00", "04:00-04:20", "04:20-04:40", "04:40-05:00"]

// 亦或者数组传入

inputDateRange(['3:00','5:00'],'45df.3d'); // ["03:00-03:45", "03:45-04:30", "04:30-05:00"]

// step 支持数字亦或者带特殊字符的数字

inputDateRange(['6:00','8:00'],'55df.3d'); // ["06:00-06:55", "06:55-07:50", "07:50-08:00"]

inputDateRange('3:00-5:00',60); // ["03:00-04:00", "04:00-05:00"]

* [JS不靠谱系列之枚举出时间段和对应的分钟数](https://link.juejin.im?target=https%3A%2F%2Fjuejin.im%2Fpost%2F598c32dcf265da3e3d12557d" \t "_blank)

**Q: Vue-Router的两种模式主要依赖什么实现的**

* hash主要依赖location.hash来改动 URL,达到不刷新跳转的效果.每次 hash 改变都会触发hashchange事件(来响应路由的变化,比如页面的更换)
* history主要利用了 HTML5的 historyAPI 来实现,用pushState和replaceState来操作浏览历史记录栈

**Q: MVVM 和 MVC 的差异? 听说过 MVP?**

这类的文章好多,三个开发模式的诞生都有前后,不是同时出现的.

传送门:

* [MVC，MVP 和 MVVM 的图示](https://link.juejin.im?target=http%3A%2F%2Fwww.ruanyifeng.com%2Fblog%2F2015%2F02%2Fmvcmvp_mvvm.html" \t "_blank)
* [浅析前端开发中的 MVC/MVP/MVVM 模式](https://link.juejin.im?target=https%3A%2F%2Fjuejin.im%2Fpost%2F593021272f301e0058273468" \t "_blank)

**Q: 求100~999的所有"水仙花"数, 就是三位数中各数字的立方和等于自身,比如153=1^3+5^3+3^3**

* 常规遍历法

function threeWaterFlower(rangeStart, rangeEnd) {

var temp = [];

rangeStart = rangeStart || 100;

rangeEnd = rangeEnd || 999;

for (var i = rangeStart; i <= rangeEnd; i++) {

var t = i.toString().split('');

Math.pow(t[0], 3) + Math.pow(t[1], 3) + Math.pow(t[2], 3) == i

? temp.push(i)

: '';

}

return temp;

}

threeWaterFlower(100,999); // [153, 370, 371, 407]

threeWaterFlower(); // [153, 370, 371, 407]

* 拓展写法,ES6版+不定花数,不折腾不舒服版本

let manyWaterFlower = (rangeStart = 100, rangeEnd = 999, flower = 3) => {

let temp = [];

for (let i = rangeStart; i <= rangeEnd; i++) {

let t = i

.toString()

.split('')

.map(item => Math.pow(item, flower))

.reduce((cur,next)=> parseInt(cur)+parseInt(next));

let transformT = parseInt(t, 10);

transformT == i ? temp.push(i) : '';

}

return temp;

}

manyWaterFlower(); // [153, 370, 371, 407]

manyWaterFlower(100,10000,4); // [1634, 8208, 9474]

manyWaterFlower(100,10000,5); // [4150, 4151]

这种是穷举遍历,若是要快一点呢(考虑的周全一点呢),以及传参范围的矫正

相信小伙伴都看得懂,我已经尽量注释了..

let manyWaterFlower = (flower = 3,rangeStart, rangeEnd ) => {

let temp = [];// 缓存所有找到的花值

// 这一段就是填充开始循环的范围,处理完毕后转为数字,推荐的开始值

let flowerRecommandStart = Number(

''.padStart(flower, '0').replace(/^(\d{1})/g, '1')

);

let flowerRecommandEnd = Number(''.padStart(flower, '9'));

// 判断是否传入开始值

if (rangeStart) {

rangeStart > flowerRecommandStart

? (rangeStart = flowerRecommandStart)

: rangeStart;

} else {

rangeStart = flowerRecommandStart;

}

// 判断是否有传入结束值

if (rangeEnd) {

rangeEnd > flowerRecommandEnd ? (rangeEnd = flowerRecommandEnd) : rangeEnd;

} else {

rangeEnd = flowerRecommandEnd;

}

// 若是初始值大于结束值

if (rangeStart > rangeEnd) {

rangeEnd = flowerRecommandEnd;

}

for (let i = rangeStart; i <= rangeEnd; i++) {

let t = i

.toString()

.split('')

.map(item => Math.pow(item, flower))

.reduce((cur, next) => parseInt(cur) + parseInt(next));

let transformT = parseInt(t, 10);

transformT == i ? temp.push(i) : '';

}

return temp;

};

console.time('manyWaterFlower');

manyWaterFlower(4)

console.timeEnd('manyWaterFlower');

// VM34013:4 manyWaterFlower: 8.112060546875ms ,这个是跑出来的时间

用上个例子的代码,从100到9999的,我们跑一下看看

console.time('manyWaterFlower');

manyWaterFlower(100,9999,4)

console.timeEnd('manyWaterFlower');

// VM3135:4 manyWaterFlower: 10.51904296875ms

// 我的 MBP 跑10花直接卡死...跑7花有点久...

console.time('7 flower')

manyWaterFlower(7);

console.timeEnd('7 flower')

// 7 flower: 6489.608154296875ms

// 8 花 CPU 的风扇狂叫....

console.time('8 flower')

manyWaterFlower(8);

console.timeEnd('8 flower')

// VM644:3 8 flower: 68010.26489257812ms

// 对了我们还没有考虑数值溢出的问题..因为正整数在 JS 的范围是有限的.

// 有兴趣的小伙伴可以自行完善

**Q: 请使用递归算法在 TODO 注释后实现通过节点 key 数组寻找 json 对象中的对应值**

比如console.log(findNode(['a1', 'b2'], data)) === data.a1.b2

// 请使用递归算法在 TODO 注释后实现通过节点 key 数组寻找 json 对象中的对应值

var data = {

a1: {

b1: 1,

b2: 2,

b3: {

b4: 5

}

},

a2: {

b1: 3,

b2: 4

}

};

function findNode(inPath, inData) {

// TODO

// 判断传入的是否是一个数组

if (Array.isArray(inPath)) {

// 当长度为1的时候寻找该 key 是否有值,有则返回,无则返回-1

if (inPath.length === 1) {

return inData[inPath[0]] ? inData[inPath[0]]: -1;

}else{

return findNode(inPath.slice(1), inData[inPath[0]]);

}

} else{

console.log('您传入的不是一个数组')

}

}

console.log(findNode(['a1', 'b2'], data)); // 2

console.log(findNode(['a1', 'b3','b4'], data)); // 5

* 来个拓展版?支持字符串或数组传入;findNode('a1.b2',data)?

var data = {

a1: {

b1: 1,

b2: 2,

b3: {

b4: 5

}

},

a2: {

b1: 3,

b2: 4

}

};

// 判断格式

function isType(params) {

let type = Object.prototype.toString.call(params);

if (type === '[object String]') {

params = params.split('.');

return params;

}

if (type === '[object Array]') {

return params;

}

}

function findNode(inPath, inData) {

inPath = isType(inPath);

// 判断传入的是否是一个数组

if (Array.isArray(inPath)) {

// 当长度为1的时候寻找该 key 是否有值,有则返回,无则返回-1

if (inPath.length === 1) {

return inData[inPath[0]] ? inData[inPath[0]]: -1;

}else{

return findNode(inPath.slice(1), inData[inPath[0]]);

}

} else {

console.log('您传入的不是一个数组');

}

}

console.log(findNode(['a1', 'b2'], data)); // 2

console.log(findNode('a1.b3.b4', data)); // 5

**Q: webpack 是什么?webpack 常见的优化手段有哪些;**

webpack 是一个资源处理工具,它的出现节省了我们的人力和时间; 可以对资源打包,解析,区分开发模式等等...

常见的优化手段:

* 分离第三方库(依赖),比如引入dll
* 引入多进程编译,比如happypack
* 提取公共的依赖模块,比如commonChunkPlugin
* 资源混淆和压缩:比如UglifyJS
* 分离样式这些,减小bundle chunk的大小,比如ExtractTextPlugin
* GZIP 压缩,在打包的时候对资源对齐压缩,只要部署的服务器能解析即可..减少请求的大小
* 还有按需加载这些,一般主流的框架都有对应的模块懒加载方式.
* 至于tree shaking目前webpack3/4已经默认集成

**Q: 从你输入一个 URL 到页面渲染的大体过程...**

大体过程是这样的,想了解很细致的可以自行引擎;

1. IP->DNS(浏览器=>系统缓存=>DNS 服务器)->域名解析完成(这一步不用太多解析吧)
2. TCP 协议走完->HTTP(S) 协议->缓存->(分析请求头)-> 回馈报文
3. 请求文档下来->DOM->CSSDOM->静态资源下载->render(绘制文档)->js 解析
4. 用户看到页面

**Q: Vue 的组件的通讯手段有哪些..**

* 父-> 子: props
* 子-> 父: on+emit
* 父<>子: on.sync(语法糖)来的
* 兄弟 : event bus | vuex

**Q: Vuex你怎么理解?**

vuex是一个状态管理容器(你也可以理解为全局变量),数据的流向是是单向数据流,

且数据并不具有持久化的特性(默认情况下刷新就重置所有状态);

里面的一些数据乃至方法,可以大致理解为 vue 的一些特性,比如

| **Vuex** | **Vue** |
| --- | --- |
| state | data |
| getter | computed |
| mutation/actions | methods |

至于单向数据流(全局单例模式)怎么理解

state只能给mutation(同步操作) 改动, action只能反馈给mutation,可以进行异步操作(比如和后端交互拉取数据), state能触发 render,action能用dispatch分发..如图

**结语**

还有一些题目是看你个人发挥的,没法写,比如

* Q: 让你来为公司的一个项目做技术选型,你会怎么做,为什么?
* Q: React,Angular,Vue的比较?
* Q: 说说你对 VNode的理解,diff的过程;
* Q: Vue的双向绑定如何实现,用了什么模式(订阅模式),大体如何实现的..
* Q: cmd/amd/commonjs的差异
* Q: 小程序以及React Native的差异..等等

**微信小程序面试题**

**1. 简单描述下微信小程序的相关文件类型**

微信小程序项目结构主要有四个文件类型

* WXML（WeiXin Markup Language）是框**架设计**的一套标签语言，结合基础组件、事件系统，可以构建出页面的结构。内部主要是微信自己定义的一套组件
* WXSS (WeiXin Style Sheets)是一套样式语言，用于描述 WXML 的组件样式
* js 逻辑处理，网络请求
* json 小程序设置，如页面注册，页面标题及tabBar

主要文件

* app.json 必须要有这个文件，如果没有这个文件，项目无法运行，因为微信框架把这个作为配置文件入口，整个小程序的全局配置。包括页面注册，网络设置，以及小程序的 window 背景色，配置导航条样式，配置默认标题
* app.js 必须要有这个文件，没有也是会报错！但是这个文件创建一下就行 什么都不需要写以后我们可以在这个文件中监听并处理小程序的生命周期函数、声明全局变量
* app.wxss 可选

**2. 简述微信小程序原理**

微信小程序采用 JavaScript、WXML、WXSS 三种技术进行开发,本质就是一个单页面应用，所有的页面渲染和事件处理，都在一个页面内进行，但又可以通过微信客户端调用原生的各种接口

微信的架构，是数据驱动的架构模式，它的 UI 和数据是分离的，所有的页面更新，都需要通过对数据的更改来实现

小程序分为两个部分 webview 和 appService 。其中 webview 主要用来展现 UI ，appService 有来处理业务逻辑、数据及接口调用。它们在两个进程中运行，通过系统层 JSBridge 实现通信，实现 UI 的渲染、事件的处理

**3. 小程序的双向绑定和vue哪里不一样**

小程序直接 this.data 的属性是不可以同步到视图的，必须调用：

this.setData({

// 这里设置

})

**4. 小程序的wxss和css有哪些不一样的地方**

WXSS 和 CSS 类似，不过在 CSS 的基础上做了一些补充和修改

* 尺寸单位 rpx

rpx 是响应式像素,可以根据屏幕宽度进行自适应。规定屏幕宽为 750rpx。如在 iPhone6 上，屏幕宽度为 375px，共有 750 个物理像素，则 750rpx = 375px = 750 物理像素

* 使用 @import 标识符来导入外联样式。@import 后跟需要导入的外联样式表的相对路径，用;表示语句结束

/\*\* index.wxss \*\*/

@import './base.wxss';

.container{

color: red;

}

**5. 小程序页面间有哪些传递数据的方法**

* 使用全局变量实现数据传递 在 app.js 文件中定义全局变量 globalData， 将需要存储的信息存放在里面

// app.js

App({

// 全局变量

globalData: {

userInfo: null

}

})

使用的时候，直接使用 getApp() 拿到存储的信息

* 使用 wx.navigateTo 与 wx.redirectTo 的时候，可以将部分数据放在 url 里面，并在新页面 onLoad 的时候初始化

//pageA.js

// Navigate

wx.navigateTo({

url: '../pageD/pageD?name=raymond&gender=male',

})

// Redirect

wx.redirectTo({

url: '../pageD/pageD?name=raymond&gender=male',

})

// pageB.js

...

Page({

onLoad: function(option){

console.log(option.name + 'is' + option.gender)

this.setData({

option: option

})

}

})

需要注意的问题：

wx.navigateTo 和 wx.redirectTo 不允许跳转到 tab 所包含的页面

onLoad 只执行一次

* 使用本地缓存 Storage 相关

**6. 小程序的生命周期函数**

* onLoad 页面加载时触发。一个页面只会调用一次，可以在 onLoad 的参数中获取打开当前页面路径中的参数
* onShow() 页面显示/切入前台时触发
* onReady() 页面初次渲染完成时触发。一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互
* onHide() 页面隐藏/切入后台时触发。 如 navigateTo 或底部 tab 切换到其他页面，小程序切入后台等
* onUnload() 页面卸载时触发。如 redirectTo 或 navigateBack 到其他页面时

详见 [生命周期回调函数](https://developers.weixin.qq.com/miniprogram/dev/framework/app-service/page.html" \l "%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F%E5%9B%9E%E8%B0%83%E5%87%BD%E6%95%B0)

**7. 怎么封装微信小程序的数据请求**

参考 [这里](https://segmentfault.com/a/1190000014789969)

网络请求小程序提供了wx.request， 仔细看一下 api，这不就是n年前的 $.ajax 吗，好古老啊。

// 官方例子

wx.request({

url: 'test.php', //仅为示例，并非真实的接口地址

data: {

x: '' ,

y: ''

},

header: {

'content-type': 'application/json' // 默认值

},

success: function(res) {

console.log(res.data)

}

})

小程序支持ES6，那么就应该支持Promise 了，很开心~， 话不多说直接上代码吧

Promise封装

**c**onst baseUrl = 'https://api.it120.cc';

const http = ({ url = '', param = {}, ...other } = {}) => {

wx.showLoading({

title: '请求中，请耐心等待..'

});

let timeStart = Date.now();

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

wx.request({

url: getUrl(url),

data: param,

header: {

'content-type': 'application/json' // 默认值 ,另一种是 "content-type": "application/x-www-form-urlencoded"

},

...other,

complete: (res) => {

wx.hideLoading();

console.log(`耗时${Date.now() - timeStart}`);

if (res.statusCode >= 200 && res.statusCode < 300) {

resolve(res.data)

} else {

reject(res)

}

}

})

})

}

const getUrl = (url) => {

if (url.indexOf('://') == -1) {

url = baseUrl + url;

}

return url

}

// get方法

const \_get = (url, param = {}) => {

return http({

url,

param

})

}

const \_post = (url, param = {}) => {

return http({

url,

param,

method: 'post'

})

}

const \_put = (url, param = {}) => {

return http({

url,

param,

method: 'put'

})

}

const \_delete = (url, param = {}) => {

return http({

url,

param,

method: 'put'

})

}

module.exports = {

baseUrl,

\_get,

\_post,

\_put,

\_delete

}

使用

const api = require('../../utils/api.js')

// 单个请求

api.get('list').then(res => {

console.log(res)

}).catch(e => {

console.log(e)

})

// 一个页面多个请求

Promise.all([

api.get('list'),

api.get(`detail/${id}`)

]).then(result => {

console.log(result)

}).catch(e => {

console.log(e)

})

**登陆问题**

做一个应用，肯定避免不了登录操作。用户的个人信息啊，相关的收藏列表等功能都需要用户登录之后才能操作。一般我们使用token做标识。

小程序并没有登录界面，使用的是 wx.login 。 wx.login 会获取到一个 code，拿着该 code 去请求我们的后台会最后返回一个token到小程序这边，保存这个值为 token 每次请求的时候带上这个值。

一般还需要把用户的信息带上比如用户微信昵称，微信头像等，这时候就需要使用 wx.getUserInfo ,这里涉及到一个用户授权的问题

带上用户信息就够了嘛？ too young too simple！我们的项目不可能只有小程序，相应的微信公众平台可能还有相应的App，我们需要把账号系统打通，让用户在我们的项目中的账户是同一个。这就需要用到微信开放平台提供的 UnionID 。

**登陆**

//app.js

App({

onLaunch: **function** () {

console.log('App onLaunch');

**var** that = **this**;

// 获取商城名称

wx.request({

url: 'https://api.it120.cc/'+ that.globalData.subDomain +'/config/get-value',

data: {

key: 'mallName'

},

success: **function**(res) {

wx.setStorageSync('mallName', res.data.data.value);

}

})

**this**.login();

**this**.getUserInfo();

},

login : **function** () {

**var** that = **this**;

**var** token = that.globalData.token;

// 如果有token

**if** (token) {

// 检查token是否有效

wx.request({

url: 'https://api.it120.cc/' + that.globalData.subDomain + '/user/check-token',

data: {

token: token

},

success: **function** (res) {

// 如果token失效了

**if** (res.data.code != 0) {

that.globalData.token = null;

that.login(); // 重新登陆

}

}

})

**return**;

}

// 【1】调用微信自带登陆

wx.login({

success: **function** (res) {

// 【2】 拿到code去访问我们的后台换取其他信息

wx.request({

url: 'https://api.it120.cc/'+ that.globalData.subDomain +'/user/wxapp/login',

data: {

code: res.code

},

success: **function**(res) {

// 如果说这个code失效的

**if** (res.data.code == 10000) {

// 去注册

that.registerUser();

**return**;

}

// 如果返回失败了

**if** (res.data.code != 0) {

// 登录错误

wx.hideLoading();

// 提示无法登陆

wx.showModal({

title: '提示',

content: '无法登录，请重试',

showCancel:false

})

**return**;

}

// 【3】 如果成功后设置token到本地

that.globalData.token = res.data.data.token;

// 保存用户信息

wx.setStorage({

key: 'token',

data: res.data.data.token

})

}

})

}

})

},

// 注册？？ [这个看需求]

registerUser: **function** () {

**var** that = **this**;

wx.login({

success: **function** (res) {

**var** code = res.code; // 微信登录接口返回的 code 参数，下面注册接口需要用到

wx.getUserInfo({

success: **function** (res) {

**var** iv = res.iv;

**var** encryptedData = res.encryptedData;

// 下面开始调用注册接口

wx.request({

url: 'https://api.it120.cc/' + that.globalData.subDomain +'/user/wxapp/register/complex',

data: {code:code,encryptedData:encryptedData,iv:iv}, // 设置请求的 参数

success: (res) =>{

wx.hideLoading();

that.login();

}

})

}

})

}

})

},

// 获取用户信息

getUserInfo:**function**() {

wx.getUserInfo({

success:(data) =>{

**this**.globalData.userInfo = data.userInfo;

wx.setStorage({

key: 'userInfo',

data: data.userInfo

})

**return** **this**.globalData.userInfo;

}

})

},

globalData:{

userInfo:null,

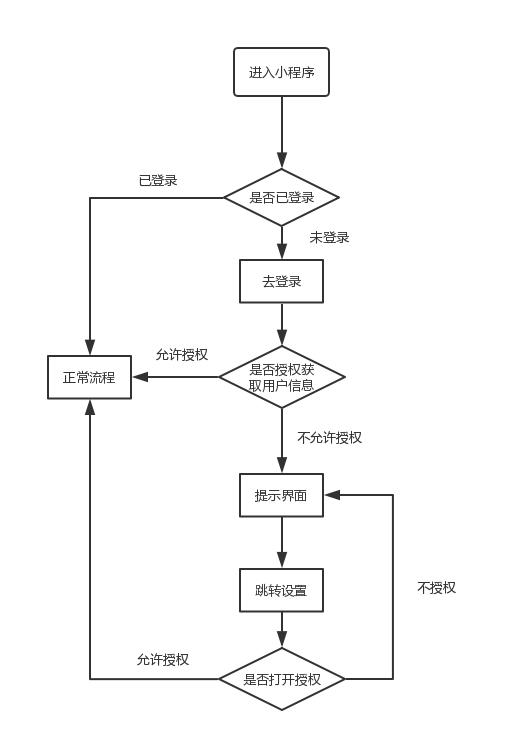
subDomain:"34vu54u7vuiuvc546d",

token: null

}

})

**授权问题**



getUserInfo: function () {

// 先调用wx.getSetting 获取用户权限设置

wx.getSetting({

success(res) {

console.log('1');

**if** (!res.authSetting['scope.userInfo']) {

wx.authorize({

scope: 'scope.userInfo',

success() {

// 用户已经同意小程序使用录音功能，后续调用 wx.getUserInfo接口不会弹窗询问

wx.getUserInfo({

success: (**data**) => {

**this**.globalData.userInfo = **data**.userInfo;

wx.setStorage({

key: 'userInfo',

**data**: **data**.userInfo

})

**return** **this**.globalData.userInfo;

}

})

}

})

} **else** {

console.log(2);

}

}

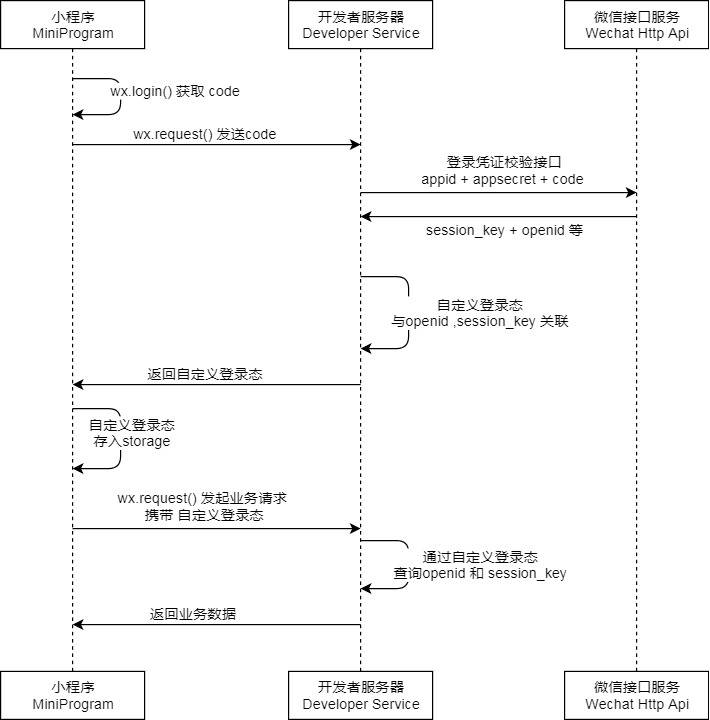
})

},

**授权2**

## 小程序登录流程

这里引用下官方的一张登录流程图，我就按照登录流程图来讲下我的理解。



#### 第一步

客户端（小程序）获取当前微信登录用户的登录凭证（code）

可通过wx.login api获得。这里有地方需要注意

1.wx.login不会弹授权弹框

2.wx.login换取的code只能使用一次,如果需要新code只能重新调用wx.login接口

wx.login({

success:(res)=>{

let code= res.code

}

})

复制代码

#### 第二步

通过上一步获得的临时登录凭证传给服务器端获取openid和session\_key.服务器端需要通过appid、appsecret、（这里的数据可以从小程序管理后台获得）code（第一步获取到的code）向微信服务端发送请求获取seeeion\_key和openid。为了安全。建议将获得的session\_key加密后再传给客户端。

#### 第三步

客户端获得加密后的登录态后把登录态存在本地以便后面进行业务请求。由于小程序中不存在cookie机制。所以可以把登录态存储在storage中。

以上就是微信官方登录流程图的一个大体过程。

但是在实际应用中可能要复杂点？我们接下来看。

## 登录态在实际应用中的维护

这里看一下微信官方的说明

通过 wx.login 接口获得的用户登录态拥有一定的时效性。

用户越久未使用小程序，用户登录态越有可能失效。

反之如果用户一直在使用小程序，则用户登录态一直保持有效。

具体时效逻辑由微信维护，对开发者透明。

开发者只需要调用 wx.checkSession 接口检测当前用户登录态是否有效。

复制代码

这说明如果用户一直在使用小程序。登录态就不会过期。反之就会过期。这里可以通过wx.checkSession api 来判断登录态是否过期。

接下来上代码。来看下在应用中的登录态维护。

目前在小程序中需要拉起微信登录授权的弹框。需要在wxml文件中调用button组件来调用：如下

<button bindgetuserinfo="getInfo" hover-class="none" open-type="getUserInfo"></button>

复制代码

这样用户点击按钮的时候会弹出授权获取用户信息的弹窗。用户点击允许我们就可以拿到数据进行登录并进行业务请求。 如果点击拒绝可以获取不需要登录可查看的数据请求，并安利用户拒绝后的结果。重新引导用户进行授权。

下面是用户非首次进入应用的一个登录态维护（首次进入通过button来授权。所以success回调是不会执行的。直接fail的回调。）

// 小程序启动判断用户是否授权，根据是否授权来请求不同的业务数据

wx.getSetting({

success: (res) => {

//用户已授权

if (res.authSetting['scope.userInfo']) {

// 判断登录态是否过期

wx.checkSession({

// 登录态未过期，直接进行业务请求

success: (res) => {

//业务请求代码。。。

},

// 登录态已过期 。重新调用wx.login进行登录换取code

fail: (res) => {

// 可以在这里进行重新登录后的回调

wx.login({

success: function(res) {

let code = res.code;

}

})

}

})

}

// 为授权

else {

// 执行未授权的业务代码

}

}

})

复制代码

附上登录态过期的回调。

/\*\*

\* 登录失败后重新登录

\*/

getToken: function(fn) {

let that = this;

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

//登录获取code

wx.login({

success: function(res) {

var code = res.code;

that.globalData.code = code;

resolve([fn, code]);

},

fail: function(res) {

reject();

}

})

});

getLogin.then(([fn, code]) => {

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

//使用该api需要在页面通过button组件触发授权弹窗

wx.getUserInfo({

success: function(res) {

//这里的iv，encryptedData等数据是用来服务器端进行解密的。

let requestData = {

"Data": {

"IV": res.iv,

"EncryptedData": res.encryptedData,

"JsCode": code,

},

}

//发送请求

wx.request({

url: that.apiList.login.getLogin,

data: requestData,

method: "POST",

success: function(res) {

//获取到自定义登录态存入storage

if (res.data && res.data.Success) {

that.globalData.token = res.data.Data.Key;

wx.setStorageSync('LoginSessionKey', res.data.Data.Key);

resolve(fn);

} else {

reject();

}

},

fail: function() {

Hq.tipMaskNoneIcon('您的网络开小差了');

}

})

}

})

});

}).then((fn) => {

that.getCountryInfo(fn);

}, function() {}))

},

//执行fn回调函数

getCountryInfo: function(fn) {

if (typeof fn == 'function') {

//登录成功后进行业务请求。

fn();

} else {

Hq.afterSend();

}

},

复制代码

以上就是我的一些理解。有语句不通，逻辑不清晰的地方,请不吝留言赐教!

**8. 哪些方法可以用来提高微信小程序的应用速度**

1、提高页面加载速度

2、用户行为预测

3、减少默认 data 的大小

4、组件化方案

**9. 微信小程序的优劣势**

优势

* 即用即走，不用安装，省流量，省安装时间，不占用桌面
* 依托微信流量，天生推广传播优势
* 开发成本比 App 低

缺点

* 用户留存，即用即走是优势，也存在一些问题
* 入口相对传统 App 要深很多
* 限制较多,页面大小不能超过2M。不能打开超过10个层级的页面

**10. 怎么解决小程序的异步请求问题**

小程序支持大部分 ES6 语法

* 在返回成功的回调里面处理逻辑
* Promise 异步

**11. 小程序关联微信公众号如何确定用户的唯一性**

如果开发者拥有多个移动应用、网站应用、和公众帐号（包括小程序），可通过 unionid 来区分用户的唯一性，因为只要是同一个微信开放平台帐号下的移动应用、网站应用和公众帐号（包括小程序），用户的 unionid 是唯一的。换句话说，同一用户，对同一个微信开放平台下的不同应用，unionid 是相同的

**12. 如何实现下拉刷新**

* 首先在全局 config 中的 window 配置 enablePullDownRefresh
* 在 Page 中定义 onPullDownRefresh 钩子函数,到达下拉刷新条件后，该钩子函数执行，发起请求方法
* 请求返回后，调用 wx.stopPullDownRefresh 停止下拉刷新

参考 [这里](https://juejin.im/post/5a781c756fb9a063606eb742)：（<https://juejin.im/post/5a781c756fb9a063606eb742>）

下拉刷新和上拉加载是业务上一个很常见的需求，在微信小程序里，提供了下拉刷新的方法 onPullDownRefresh 。而实现上拉加载相对来说就比较不方便了。

## 下拉刷新

虽然微信的官方文档有很多坑，但下拉刷新介绍的还是很全面的。在这里稍稍带过。

* 首先在全局 config 中的 window 配置 enablePullDownRefresh .
* 在 Page 中定义 onPullDownRefresh 钩子函数。到达下拉刷新条件后，该钩子函数执行，发起请求方法。
* 请求返回后，调用 wx.stopPullDownRefresh 停止下拉刷新。

### config

config = {

pages: [

'pages/index'

],

window: {

backgroundTextStyle: 'light',

navigationBarBackgroundColor: '#ccc',

navigationBarTitleText: 'WeChat',

navigationBarTextStyle: '#000',

enablePullDownRefresh: true

}

}复制代码

### page

onPullDownRefresh() {

wepy.showNavigationBarLoading()

setTimeout(()=>{

this.getData = '数据拿到了'

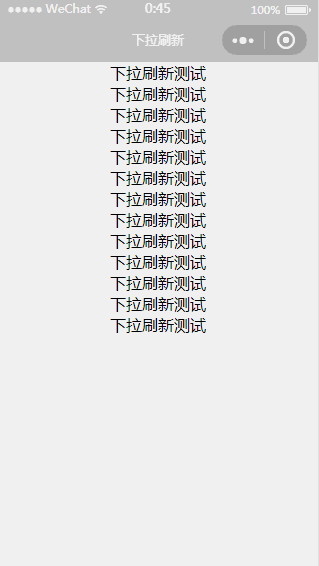
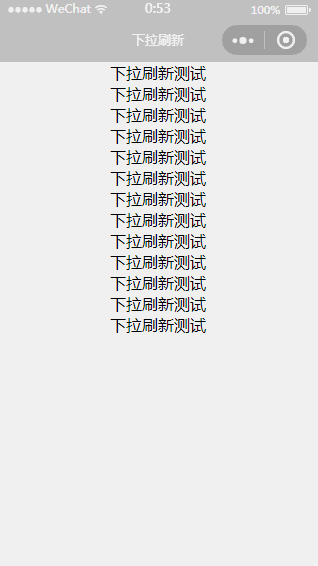
wepy.stopPullDownRefresh()

wepy.hideNavigationBarLoading()

this.$apply()

},3000)

}复制代码

效果如下：  
[](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fcheckmind%2FmyNote%2Fblob%2Fmaster%2Fwepy%2Fimages%2F1.gif)  
你会发现下拉的过程有些僵硬。这实际上是没有添加背景色的原因，加上背景色后再试试。  
[](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fcheckmind%2FmyNote%2Fblob%2Fmaster%2Fwepy%2Fimages%2F2.gif)  
现在感觉好多了吧。下拉刷新有现成的配置和方法，很容易实现，可上拉加载就不同了。

## 上拉加载

首先看一下要实现的效果，这是3g端的上拉加载。小程序要实现同样的效果。  
[](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fcheckmind%2FmyNote%2Fblob%2Fmaster%2Fwepy%2Fimages%2F4.gif)  
首先功能有

* 点击回到顶部 这个很好实现，有对应的回到顶部函数
* 滑动屏幕记录当前页数 这个也很好实现，主要是监听滚动事件，判断对应滚动条高度，去计算其与子容器的高度即可。
* 上拉加载动画

这里有两个实现的方案。一个是 page 自带的下拉触底钩子事件 onReachBottom 能做的只是下拉到底部的时候通知你触底了，一个是 scroll-view 标签自带事件。现在用两个方法分别实现一下上拉加载。

### 上拉触底事件 onReachBottom

模板

<template>

<view class="loading"></view>

<view class="container"

@touchmove="moveFn"

@touchstart="startFn"

@touchend="endFn"

style="transform:translate3d(0,{{childTop}}px,0)">

<repeat for="{{list}}"

key="index"

index="index"

item="item">

<view>{{ item }}<text>{{index}}</text></view>

</repeat>

</view>

</template>复制代码

钩子函数

data = {

getData: '',

top: 0,

lastTop: 0,

canDrag: false,

list: []

}

onReachBottom() {

this.canDrag = true

}

methods = {

moveFn(ev) {

let nowY = ev.changedTouches[0].clientY

nowY = nowY-this.lastTop

if(nowY > 0 )

this.canDrag = false

if( nowY<=0 && this.canDrag ) {

this.top = nowY

}

if( -this.top>= this.maxTop )

this.top = -this.maxTop

},

startFn(ev) {

this.lastTop = ev.changedTouches[0].clientY

},

endFn() {

if(this.top <= -this.maxTop) {

this.text = "去请求数据了"

setTimeout(()=>{

this.text = "请求回来了"

this.canDrag = false

this.list.push(...["数据","数据","数据"])

this.$apply()

this.top = 0;

return

},1000)

}

},

gotoTop() {

wepy.pageScrollTo({

scrollTop: 0

})

}

}复制代码

完成后看一下效果：

### 滚动容器实现上拉加载

scroll-view： 可滚动视图区域。  
它的具体用法不赘述，看官方文档就行了。这里提解决上述问题的方法即可。

* bindscrolltolower 类比原生全局钩子 onReachBottom  
  模板

<scroll-view scroll-y

id="content"

@scroll="scroll"

@scrolltolower="lower"

scroll-top="{{gotoTopNum}}"

lower-threshold="100"

style="transform:translate3d(0,{{childTop}}px,0)">

<view class="sty-search"

@touchmove="moveContent"

@touchstart="startContent"

@touchend="endContent">...</view>

</scroll-view>复制代码

以上就是最终的模板，你可能在想为什么这么复杂。虽然复杂，但每个属性都是有用的，当然这其中有几个坑在等着我们。  
首先节点分为滚动容器和子容器。

Q：为什么滚动容器里嵌套一个子容器，并且将拖动的三个方法绑定在它上面。  
A：这是第一个坑，因为 scroll-view 容器不能绑定 touchmove 事件,那如果绑定了会怎么样呢？不会怎么样，事件钩子不会调用。（这个坑在官方文档查不出来，当时绑定了不调用，在社区找到了解决方法，就是将touchmove事件绑定到子容器）  
再来看代码

methods = {

async lower() {

this.canDrag = true

},

scroll (ev) {

this.scrollTop = ev.detail.scrollTop

if (ev.detail.deltaY > 0) {

this.canDrag = false

}

let nowSet = this.documentHeight+this.scrollTop-this.contentHeader

let num = Math.ceil(nowSet/this.listHeight) - 1

num = Math.floor(num / this.pageBean.pageSize) + 1

num = (num > this.pageBean.pageNo) ? this.pageBean.pageNo : num

if(num != this.page) {

this.page = num

this.$apply()

}

},

startContent(ev) {

this.lastTop = ev.changedTouches[0].clientY

if(!this.documentHeight){

this.documentHeight = wx.getSystemInfoSync().windowHeight

}

/\* 这句是解决回到顶部的bug \*/

if (this.gotoTopNum || this.gotoTopNum==0) { this.gotoTopNum = undefined }

},

moveContent (ev) {

let {

pageNo,

pageSize,

totalCount

} = this.pageBean

let nowY = ev.changedTouches[0].clientY

nowY = nowY-this.lastTop

if (this.canDrag && nowY) {

this.state = 1;

if (nowY <= -this.maxMove) {

nowY = -this.maxMove

}

if (nowY <= 0) {

this.childTop = nowY

}

}

},

async endContent(ev) {

let {

pageNo,

pageSize,

totalCount

} = this.pageBean

if (this.childTop === -this.maxMove) {

/\* 状态 \*/

if (pageNo >= this.maxPage || pageNo \* pageSize >= totalCount) {

this.state = 0

} else {

this.pageBean.pageNo++

await this.fillData()

this.childTop = 0

this.canDrag = false

this.$apply()

}

}

/\* 如果没超过刷新高度则重置 \*/

this.childTop = 0

},

gotoTop() {

this.gotoTopNum = 0

},

}

复制代码

Q： 为什么要在 touchStart 的时候 将 gotoTopNum 置为 undefined?  
A： 因为这个页面有一个回到顶部的功能，当回到顶部时，gotoTopNum 置为0，再次下翻时，虽然实际的 scrollTop 改变了，但是 gotoTopNum 还为0，再次点击回到顶部时，因为数据未改变，视图层就不会去更新。所以在 touchStart 的时候给 gotoTopNum 一个无效的值，再次点击回到顶部时，视图层也就更新了。

## 原生滚动 OR scroll-view

## 

## END...了吗......

并没有。

## 真机测试

实现的上拉加载在模拟器上跑的很流畅，不存在问题。可是。  
如果是苹果机的话（暂时测试iphone5 和 iPhone7），存在这样一个问题，上拉或下拉回弹效果，这个效果会影响上拉的距离。  
这个问题想了很久，目前不能优雅的解决。  
所以就找**产品经理修改了需求**，去掉了上拉动画效果 所以最终的效果就变成：  
[](https://link.juejin.im/?target=https%3A%2F%2Fgithub.com%2Fcheckmind%2FmyNote%2Fblob%2Fmaster%2Fwepy%2Fimages%2F7.gif)

## 总结

1. 在微信小程序里操作节点是昂贵的，比在浏览器里操作还昂贵（这是通过比较上拉加载功能在3g端和微信小程序的流畅度得来的），在 1.4.0 版本发布之后，虽然给出了很多操作节点的方法，比如得到一个节点的宽高、或者通过 id 选择器得到一个节点。请尽量减少这些方法的调用频率( 函数节流 )或 缓存结果
2. 动手之前先动脑！！！不然会走很多弯路..

**13. bindtap和catchtap的区别是什么**

相同点：首先他们都是作为点击事件函数，就是点击时触发。在这个作用上他们是一样的，可以不做区分

不同点：他们的不同点主要是bindtap是不会阻止冒泡事件的，catchtap是阻值冒泡的

**14. 简述下 `wx.navigateTo()`, `wx.redirectTo()`, `wx.switchTab()`, `wx.navigateBack()`, `wx.reLaunch()`的区别**

* wx.navigateTo()：保留当前页面，跳转到应用内的某个页面。但是不能跳到 tabbar 页面
* wx.redirectTo()：关闭当前页面，跳转到应用内的某个页面。但是不允许跳转到 tabbar 页面
* wx.switchTab()：跳转到 abBar 页面，并关闭其他所有非 tabBar 页面
* wx.navigateBack()关闭当前页面，返回上一页面或多级页面。可通过 getCurrentPages() 获取当前的页面栈，决定需要返回几层
* wx.reLaunch()：关闭所有页面，打开到应用内的某个页面