Js

1、JS中的前端性能优化原则：

多使用内存、缓存等方法

减少CPU计算、减少HTTP请求

(1)减少http请求次数：css精灵、JS、CSS源码压缩合并 a.js b.js c.js -> abc.js、图片大小控制合适；网页Gzip，CDN托管让资源加载更快，对不同区域的网络进行优化，data缓存

初始首屏之外图片的懒加载 （当用户下拉到该页面时再加载图片，而不是打开页面全部加载完毕。）

缓存Ajax

(2)CSS放head中,JS放body下面（JS会阻塞页面DOM的加载，放下面，不会阻塞渲染，先加载页面再执行JS）

(3)减少DOM操作查询、合并DOM操作、缓存DOM查询，缓存Ajax查询结果，

var frag = document.createDocumentFragment();DOM碎片

var pList = document.getElementByTagName('p');

for循环时缓存DOM对象长度

字符串拼接join代替+

(4)时间节流，setTimeout减少页面阻塞，减少事件操作，连续操作时会很多计算，影响性能。

(5)避免使用css表达式，标签闭合小写，不乱嵌套，一个h1标签，img加alt 属性，项目目录不要太深，

(6)document.ready代替window.onload

(7)innerHTML代替dom操作,className代替style,

(8)少用全局变量：作用域链最顶端 ，解决在赋值给局部变量

2、清除缓存方法：

(1)meta方法用客户端代码使浏览器不再缓存Web页面：

<meta http-equiv="Expires" CONTENT="0">

<meta http-equiv="Cache-Control CONTENT="no-cache">

<meta http-equiv="Pragma" CONTENT="no-cache">

(2)用随机数和随机数一样。

URL?参数后加上??ran="?+?Math.random();//当然这里参数?ran可以任意取了

在?URL?参数后加上??timestamp=+?new?Date().getTime();

(3)用php后端清理

在服务端加?header("Cache-Control:?no-cache,?must-revalidate");

以上是网上常用的清除方法,另外在java中可以通过:JAVA后端清除Session缓存方法来实现

session.removeAttribute("user");??

session.removeAttribute("constant\_cache");??

session.invalidate();

3.node

nodejs是一个javascript的运行环境 运行在服务器,作为web server 运行在本地,作为打包工具或者构建工具

Nodejs基于Javascript语言,实现前后端统一语言，利于前端代码整合, nodejs的性能是高于其他后台语言的，可以做缓冲来增加服务器端的总体性能。

NodeJS集成npm,所以npm也一并安装了

Npm是nodejs的包管理器 在github里进行下载、

应用场景：做与服务器的一些事、网站开发、im即时聊天（socket. io）、高并发

api（移动端，pc）

HTTP? Proxy

前端构建工具

4、HTTP

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

200 OK 正常返回信息

201 Created 请求成功并且服务器创建了新的资源

202 Accepted 服务器已接受请求，但尚未处理

301 Moved Permanently 请求的网页已永久移动到新位置。

302 Found 临时性重定向。

303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。

304 Not Modified 自从上次请求后，请求的网页未修改过。

400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

401 Unauthorized 请求未授权。

403 Forbidden 禁止访问。

404 Not Found 找不到如何与 URI 相匹配的资源。

500 Internal Server Error 最常见的服务器端错误。

503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

5、对前端界面工程师这个职位是怎么样理解的？它的前景会怎么样？

前端是最贴近用户的程序员，比后端、数据库、产品经理、运营、安全都近。

1、实现界面交互

2、提升用户体验

3、有了Node.js，前端可以实现服务端的一些事情

前端是最贴近用户的程序员，前端的能力就是能让产品从 90分进化到 100 分，甚至更好，

参与项目，快速高质量完成实现效果图，精确到1px；

与团队成员，UI设计，产品经理的沟通；

做好的页面结构，页面重构和用户体验；

处理hack，兼容、写出优美的代码格式；

针对服务器的优化、拥抱最新前端技术。

6、ajax过程

(1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.

(2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.

(3)设置响应HTTP请求状态变化的函数.

(4)发送HTTP请求.

(5)获取异步调用返回的数据.

(6)使用JavaScript和DOM实现局部刷新.

7、闭包

**外部读取函数可以**读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中，不会在调用后被自动清除。

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

使用场景：

闭包，内部函数使用外部函数的变量

function f1() {

    var n = 999;

    function f2() {

       console.log(n);

    }

    return f2;

}

var result = f1();

result();//999

使用场景

(1)setTimeout

原生的setTimeout传递的第一个函数不能带参数，通过闭包可以实现传参效果。

function f1(a) {

    function f2() {

        console.log(a);

    }

    return f2;

}

var fun = f1(1);

setTimeout(fun,1000);//一秒之后打印出1

(2)回调

定义行为，然后把它关联到某个用户事件上（点击或者按键）。代码通常会作为一个回调（事件触发时调用的函数）绑定到事件。　当点击数字时，字体也会变成相应的大小。

<a href="#" id="size-12">12</a>

    <a href="#" id="size-20">20</a>

    <a href="#" id="size-30">30</a>

    <script type="text/javascript">

        function changeSize(size){

            return function(){

                document.body.style.fontSize = size + 'px';

            };

        }

        var size12 = changeSize(12);

        var size14 = changeSize(20);

        var size16 = changeSize(30);

        document.getElementById('size-12').onclick = size12;

        document.getElementById('size-20').onclick = size14;

        document.getElementById('size-30').onclick = size16;

    </script>

(3)使用场景三:封装相关功能集

8、gzip 优点是减轻了带宽压力，缺点是加重了服务器的计算压力

9、Ajax中status状态码

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，已经接收到全部响应内容

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cookie实现用户信息，历史记录，

10、cookie来实现购物车功能

一、大概思路

　　1、从cookie中取商品列表

　　2、判断要添加的商品是否存在cookie中。

　　3、如果已添加过，则把对应的商品取出来，把要添加的商品的数量加上去。

　　4、如果没有添加过，则把改商品添加到商品列表中。

　　5、再把商品列表序列化，加入cookie中。

11、命名规范

可读性~能看懂，规范性~，匈牙利命名的：类型前缀，首字母大写a数组，b布尔，f浮点，fn函数，o对象

12、webpack：

前端本身不支持像后端那样文件引用，使用webpack就可以实现这种功能。另外打包还会对代码做检查和压缩，达到优化的目的。

13、typeof和instanceof区别

typeof:

1.返回值是一个字符串， 用来说明变量的数据类型。

2.typeof 一般只能返回如下几个结果： number, boolean, string, function, object, undefined。

instanceof:

1.返回值为布尔值;

2. instanceof 用于判断一个变量是否属于某个对象的实例。

14、PC端/移动端常见的兼容性问题总结

① 安卓浏览器看背景图片，有些设备会模糊，原因是手机的分辨率太小

解决方案： 用2X图片来代替img标签，然后background-size: contain

② 防止手机中页面放大或缩小： 在meta中设置viewport user-scalable = no

③ 上下拉滚动条卡顿： overflow-scrolling: touch;

④ 禁止复制选中文本： user-select: none;

⑤ 长时间按住页面出现闪退：-webkit-touch-callout: none;

⑥ 动画定义3D硬件加速： transform: translate 3d(0,0,0);

⑦ formate-detection 启动或禁止自动识别页面中的电话号码，content = "yes/no"

⑧ a标签添加tel是拨号功能

⑨ 安卓手机的圆角失效： background-clip: padding-box;

⑩ 手机端300ms延迟： fastclick

① 横平时字体加粗不一致： text-size-adjust: 100%;

PC端：

① rgba不支持IE8 用opacity属性代替rgba设置透明度

② 图片加a标签在IE9中出现边框 解决方案： img{border: none;}

③ IE6不支持display: inline-block 设置为： display: inline

④ position : fixed 不支持IE5/IE6

⑤ IE6，Firfox中，width = width + padding + border

⑥ min-height ie 6.0不支持；ie 7.0后的支持，但也可能会存在兼容性问题；

15、为什么要进行URL编码?

是因为Url中有些字符会引起歧义，Url的编码格式采用的是ASCII码

encodeURIComponent编码的字符范围要比encodeURI的大，encodeURIComponent对特殊字符编码解决

用ASCII吗表示为： **3D：= ，   26：&**

现在有这样一个问题，如果我的参数值中就包含=或&这种特殊字符的时候该怎么办。   
比如说“name1=value1”,其中value1的值是“va&lu=e1”二手字符串，那么实际在传输过程中就会变成这样“name1=va&lu=e1”。我们的本意是就只有一个键值对，但是服务端会解析成两个键值对，这样就产生了奇异。

URL编码只是简单的在特殊字符的各个字节前加上%，例如，我们对上述会产生奇异的字符进行URL编码后结果：“name1=va%26lu%3D”，这样服务端会把紧跟在“%”后的字节当成普通的字节，就是不会把它当成各个参数或键值对的分隔符。

console.log(encodeURIComponent("va&lu=e1"))// va%26lu%3De1

16、一个页面从输入URL到页面加载显示完成，这个过程都发生什么？

可以分为这几个大的过程：

DNS解析

TCP连接

客户端发送HTTP请求

服务器处理请求并返回HTTP报文

浏览器解析渲染页面

结束

17、http协议中302状态

302：临时重定向，理解为该资源原本确实存在，但已经被临时改变了位置

403：服务器拒绝请求

500：服务器内部错误

200：请求成功

404：页面无法找到

502：服务器网关错误

18、数组的常用方法如下:

push 尾部添加

pop尾部删除

unshift头部添加

shift头部删除

arr.splice(位置，个数，添加新项)

concat: 链接两个或者更多数据，并返回结果。

arr.every(): 数组里面所有的元素都要符合条件，才返回true

filter: 对数组中的每一项运行给定函数，返回改函数会返回true的项组成的数组。

forEach: 对数组中的每一项运行给定函数，这个方法没有返回值。

join: 将所有的数组元素链接成一个字符串。

indexOf: 返回第一个与给定参数相等的数组元素的索引，没有找到则返回-1。

lastIndexOf: 返回在数组中搜索到的与给定参数相等的元素的索引里最大的值。

map: 对数组中的每一项运行给定函数，返回每次函数调用的结果组成的数组。重新整理数据结构:

arr.reverse(): 颠倒数组中元素的顺序，原先第一个元素现在变成最后一个，同样原先的最后一个元素变成现在的第一个。

slice(start,end-1): 从已有数据中选中元素，返回新数组，含头不含尾，

arr.some(): 类似查找, 数组里面某一个元素符合条件，返回true/false

sort: 按照字母顺序对数组排序，支持传入指定排序方法的函数作为参数。

arr.toString: 将数组作为字符串返回。

arr.reduce() ：从左往右执行arr.reduce(prev,item,index,arr) ; prev上次计算的结果

arr.valueOf: 和toString相似，将数组作为字符串返回

for....of....： arr.keys() 数组下标 ,arr.entries() 数组某一项

arr.find(): 查找，找出第一个符合条件的数组成员，如果没返回有找到，undefined,return返回一个值

arr.findIndex(): 找的是位置， 没找到返回-1

arr.fill() 填充,arr.fill(填充的东西, 开始位置, 结束位置不包含);

arr.includes() 返回true/false

Array.from:

作用: 把类数组就是伪数组(获取一组元素、arguments...) 对象转成数组

addEventListener :true - 事件句柄在捕获阶段执行

map主要是对数组每个元素的操作，有return，

for/in主要是对象键值的一些遍历，对象

forEach的应用只要是数组的简单遍历

for..of遍历数组，有keys(),entries()

19、vue生命周期：

beforeCreate 组件实例刚刚被创建,属性都没有

created 实例已经创建完成，属性已经绑定

beforeMount 模板编译之前（准备）

mounted 模板编译之后，代替之前ready \*

beforeUpdate 组件更新之前 data数据变了 (用在$.watch('a',function){})

updated 组件更新完毕 \* (用在$.watch('a',function){})

beforeDestroy 组件销毁前

destroyed 组件销毁后

Vue钩子函数：created mounted updated Destroy

20、sessionStorage 和localStorage

(1)sessionStorage 存储系统存储数据的特点

\* 当浏览器窗口关闭时,数据全部丢失

\* 当再次打开浏览器窗口时,数据不能使用

(2)localStorage 存储系统存储数据的特点

\* 当浏览器窗口关闭时,数据依旧存储

\* 当再次打开浏览器窗口时,数据继续使用

\* 数据只能由用户删除

21、Get与Post的主要区别

get相对于post更不安全，虽然都可以加密

get的参数会显示在浏览器地址栏中，而post的参数不会显示在浏览器地址栏中；

使用post提交的页面在点击【刷新】按钮的时候浏览器一般会提示“是否重新提交”，而get则不会；

用get的页面可以被搜索引擎抓取，而用post的则不可以；

用post可以提交的数据量非常大，而用get可以提交的数据量则非常小(2k)，受限于网页地址的长度。

用post可以进行文件的提交，而用get则不可以。

22、CSS盒子模型:一个是标准模型，一个是IE模型。

CSS盒子模式都具备这些属性：内容(content)、内边距(padding)、边框(border)、外边距(margin)

标准模型 ：宽高只是内容（content）的宽高

box-sizing:content-box;

IE模型：宽高是内容(content)+填充(padding)+边框(border)的总宽高。

box-sizing:border-box;

23、. 前端开发的优化问题

（1） 减少http请求次数：css sprite,data uri

（2） JS，CSS源码压缩

（3） 前端模板 JS+数据，减少由于HTML标签导致的带宽浪费，前端用变量保存AJAX请求结果，每次操作本地变量，不用请求，减少请求次数

（4） 用innerHTML代替DOM操作，减少DOM操作次数，优化javascript性能

（5） 用setTimeout来避免页面失去响应

（6） 用hash-table来优化查找

（7） 当需要设置的样式很多时设置className而不是直接操作style

（8） 少用全局变量

（9） 缓存DOM节点查找的结果

（10） 避免使用CSS Expression

（11） 图片预载

（12） 避免在页面的主体布局中使用table，table要等其中的内容完全下载之后才会显示出来，显示比div+css布局慢

24、null和undefined区别

null 是javascript关键字，空值，typeof返回object,可以表示数字，字符串和对象“无值”

undefined是预定义的全局变量，为“未定义”，变量一种取值，表示没有初始化。

当查询对象属性，数组元素值时，返回undefined时表示属性或元素不存在；

如果函数没返回值也返回undefined

25\关于异步，解决方案：

a). 回调函数

b). 事件监听

c). 发布/订阅

d). Promise对象

26、new操作符具体干了什么呢?

（1）创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。

（2）属性和方法被加入到 this 引用的对象中。

（3）新创建的对象由 this 所引用，并且最后隐式的返回 this 。

27、CSS水平垂直居中常见方法总结

1、文本水平居中

line-height,text-align:center(文字)

元素水平居中 margin:0 auo

方案1：position 元素已知宽度

父元素设置为：position: relative;

子元素设置为：

position: absolute; left: 50%;top: 50%;margin: -50px 0 0 -50px;

距上50%，据左50%，减去元素自身宽度的距离

方案2：position transform 元素未知宽度

margin: -50px 0 0 -50px;替换为：transform: translate(-50%,-50%);

方案3：flex布局

父元素加：

display: flex; //flex布局

justify-content: center; //使子项目水平居中

align-items: center; //使子项目垂直居中

28、移动端设置

<meta name="viewport" content="width=device-width, initial-scale=1.0,maximum-scale=1.0, user-scalable=no"/>

29、JSONP原理

ajax 请求受同源策略影响，不允许进行跨域请求，而 script 标签 src 属性中的链接却可以访问跨域的js脚本，利用这个特性，

服务端不再返回JSON格式的数据，而是返回一段调用某个函数的js代码，在src中进行了调用，这样实现了跨域。

30. 如何规避javascript多人开发函数重名问题

参考答案：

(1) 可以开发前规定命名规范，根据不同开发人员开发的功能在函数前加前缀

(2) 将每个开发人员的函数封装到类中，调用的时候就调用类的函数，即使函数重名只要类名不重复就ok

31、回流，重绘

什么是回流

当render tree中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建。这就称为回流(reflow)。

什么是重绘

当render tree中的一些元素需要更新属性，而这些属性只是影响元素的外观，风格，而不会影响布局的，比如background-color。则就叫称为重绘。

32、什么是跨域，为什么浏览器会禁止跨域，实现跨域的几种方法

1、什么是跨域

跨域的产生来源于浏览器所的‘同源策略’，所谓同源策略，是指只有在地址的：

1. 协议名 https，http

2. 域名 http://a.study.cn http://study.cn

3. 端口名 http://study.cn:8080/json/jsonp/jsonp.html study.cn/json/jsonp/jsonp.html

均一样的情况下，才允许访问相同的cookie、localStorage或是发送Ajax请求等等。若在不同源的情况下访问，就称为跨域。

2、为什么浏览器会禁止跨域

跨域的访问会带来许多安全性的问题，比如，cookie一般用于状态控制，常用于存储登录的信息，如果允许跨域访问，那么别的网站只需要一段脚本就可以获取你的cookie，从而冒充你的身份去登录网站，造成非常大的安全问题，因此，现代浏览器均推行同源策略。

demo2: 基于script标签实现跨域

<script type="text/javascript">

var jshow = function(data){

alert(data.s);//// 爱奇艺,阿里云,阿里巴巴,安居客,apple,airpods2,apex,安徽人亊考试网,安全教育平台

};

var url = "https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=a&cb=jshow";

var script = document.createElement('script');

script.setAttribute('src', url);

document.getElementsByTagName('head')[0].appendChild(script);

</script>

JSONP

2.document.domain

这种方式只适合主域名相同，但子域名不同的iframe跨域。

比如主域名是http://crossdomain.com:9099，子域名是http://child.crossdomain.com:9099，这种情况下给两个页面指定一下document.domain即document.domain = crossdomain.com就可以访问各自的window对象了。

不同主域script

13、求和

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

var arr = [1,2,3,4,5];

var sum = 0;

arr.forEach(function(v){

    sum += v

})

console.log(sum);//15  map一样

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

var arr = [1,2,3,4,5];

var sum=0;

arr.forEach(v=>sum+=v)

console.log(sum);//15  map一样

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

var arr = [1,2,3,4,5];

var sum = 0;

for(let val of arr){

  sum+=val;

}

console.log(sum)

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

var arr = [1,2,3,4,5];var sum =0;

for(var i=0;i<arr.length;i++){

  sum+=arr[i];

}

console.log(sum)

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

var arr = [1,2,3,4,5];

var newArr = arr.reduce((pre,item)=>{

    return pre+item;

})

console.log(newArr)

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

二、程序：

结果：test is not a function

var test =1 ;

function test(index){

    console.log(index);

    index=3;

}

test(2)//test is not a function

13、获取URL地址参数

//方法1

url =location.href;// var url = 'http://baidu.com?a=1&b=55';

var theRequest = {};

if (url.indexOf("?") != -1) {

    var str = url.substr(url.indexOf("?")+1);//a=1&b=55

    strs = str.split("&");

    for(var i = 0; i < strs.length; i ++) {

        theRequest[strs[i].split("=")[0]] = strs[i].split("=")[1];

    }

    console.log(theRequest) //{a: "1", b: "55"}

}

//方法2

function GetRequest() {

    var url = location.search; //获取url中"?"符后的字串 ?a=2&b=33

    var theRequest = {};

    if (url.indexOf("?") != -1) {

        var str = url.substr(1);//a=2&b=33

        strs = str.split("&");//["a=2", "b=33"]

        for(var i = 0; i < strs.length; i ++) {

            theRequest[strs[i].split("=")[0]] = strs[i].split("=")[1];

        }

    }

    return theRequest;

}

Request = GetRequest();

console.log(Request)//{a: "2", b: "33"}

//方法3

function getQueryString(name) {

        var reg = new RegExp('(^|&)' + name + '=([^&]\*)(&|$)', 'i');

        var r = location.search.substr(1).match(reg);

        if (r != null) {

            return unescape(r[2]);

        }

        return null;

    }

Promise的队列与setTimeout的队列有何关联？

因此promise.then的回调比setTimeout先执行。

12354

setTimeout(function(){

    console.log(4)

},0);

new Promise(function(resolve){

    console.log(1)

    for( var i=0 ; i<10000 ; i++ ){

         i==9999 && resolve()

    }

    console.log(2)

    }).then(function(){

         console.log(5)

});

console.log(3);//1235

add 来替换 sub，add.call(sub,3,1) == add(3,1)

执行call前边的函数

function add(a,b)

{

    alert(a+b);

}

function sub(a,b)

{

    alert(a-b);

}

add.call(sub,3,1);//4

33、 使用JavaScript深度克隆一个对象。(百度)

参考答案：

Javascript中的对像赋值与Java中是一样的，都为引用传递。就是说，在把一个对像赋值给一个变量时，那么这个变量所指向的仍就

是原来对像的地址。那怎么来做呢？答案是“克隆”。

克隆有两种方法：一种是“浅克隆”，一种是“深克隆”（深度克隆）。

浅克隆：基本类型为值传递，对象仍为引用传递。数组和对象

深克隆（深度克隆）：所有元素均完全复制，并于原对象完全独立（原对象的修改不影响新对象）。

深度克隆的代码如下（深克隆去掉递归即为浅克隆）：

Object.prototype.clone = function() {

    var o = (this.constructor === Array ? [] : {});//{}

    for(var key in this) {  //this ->{name: "johnny"}

         o[key] = typeof this[key] === "object" ? this[key].clone() : this[key];

    }

    return o;

}

obj1 = {

    name : "johnny"

};

obj2 = obj1.clone();

console.log(obj2)

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

34、rem自动计算

 (function(doc, win) {

    var docEl = doc.documentElement,

        resizeEvt = 'orientationchange' in window ? 'orientationchange' : 'resize',

        recalc = function() {

            var clientWidth = docEl.clientWidth;

            if (!clientWidth) return;

            docEl.style.fontSize = 100 \* (clientWidth / 750) + 'px';

        };

    if (!doc.addEventListener) return;

    win.addEventListener(resizeEvt, recalc, false);

    doc.addEventListener('DOMContentLoaded', recalc, false);

})(document, window);

35、冒泡排序：

function sort(arr){

    for(var i=0;i<arr.length;i++){

        for(var j=0;j<arr.length-i-1;j++){

            if(arr[j]>arr[j+1]){

                var temp = arr[j];

                    arr[j]=arr[j+1];

                    arr[j+1]=temp;

            }

        }

    }

}

var arr=[1,2,5,32,54,33];

sort(arr);

console.log(arr); //[1, 2, 5, 32, 33, 54]

4、数组去重

Array.prototype.unique=function(){

    var newArr=[];

    for(var i=0;i<this.length;i++){

        if(newArr.indexOf(this[i])==-1){

            newArr.push(this[i]);

        }

    }

    return newArr;

}

var arr= [1,2,3,'3','3',2,3,4,2];

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

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

Array.prototype.unique=function(){

    var res=[];

    for(var i=0,newArr=[];i<this.length;i++){

        if(!newArr[this[i]]){

            res.push(arr[i]);

            newArr[this[i]]=1;

        }

    }

    return res;

}

var arr= [1,2,3,'3','3',2,3,4,2];

console.log(arr.unique());//[1, 2, 3, 4]

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

function unique(arr){

    for(var i=0,hash=[],result =[];i<arr.length;i++){

        if(hash[arr[i]]===undefined){

            result[result.length]=arr[i];

            hash[arr[i]]=true;

        }

    }

    return result;

}

 console.log(unique([2,3,2,1,3,4,1,5]));  //[2, 3, 1, 4, 5]

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

**36、JSON对象**

JSON.stringify() 序列化,给一个json,出来一个字符串 ，字符串写法JSON.stringify()结果有单引号如：‘{"a":12，"b":5，"name":"blue"} ’

JSON.parse() 给一个字符串 进行解析，还原成json

JSON.stringify({a:12,b:5}) =>'{"a":12,"b":5}'

JSON.parse('{"a":12,"b":5}') =>{a:12,b:5}

ES6

1、关于定（声明）变量：

let和const

(1)var 可重复声明，不可限制修改，函数级

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

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

不可以：let a=12; let a=5; //不能重复声明定义，报错has already been declare

可以：let a=12;a=5; alert(a) //5，变量可改

Let 能替代闭包

点abc分别 输出 3

<input type="button" value='a'>

<input type="button" value='b'>

<input type="button" value='c'>

<script>

for(var i=0;i<aBtn.length;i++){

    aBtn[i].onclick = function(){

        alert(i)

   }

}

</script>

点abc分别   输出 0 1 2，

解决方法一:let

for(let i=0;i<aBtn.length;i++){

    aBtn[i].onclick = function(){

        alert(i)

   }

}

解决方法二：闭包

说明：var作用域是函数，为了强行弄出三个 i 包个函数，for循环几次就有几个函数执行就有几个i，这样每个i就都属性自己独立函数里的 ，不是window.load的

点abc分别 输出 0 1 2，

for(var i=0;i<aBtn.length;i++){

   (function(i){

        aBtn[i].onclick = function(){

          alert(i)

        }

   })(i)

}

2、解构赋值:

(1)两边的结构必须一样

(2)右边必须是个合法东西

(3)赋值和解构必须同时完成

针对上述错误写法：

let {a,b} = [12,6]; console.log(a,b);   //X 两边结构不一样,

let {a,b} = {12,5}; console.log(a,b);   //X 右边不是json也不是数组, let {a,b};{a,b} = {a:12,b:5}           //X 需要同时完成let {a,b}= {a:12,b:5}

**解构扩展**（参数扩展(收集，展开)、数组展开、json展开）**展开用表示用 ...**

**3、箭头函数**

作用：

1. 简写, (2)修正this，固定this,this=>当前的环境（在哪个对象环境里执行或取决于你这句话所执行时this是什么） 普通函数function写法的this跟着执行人走谁执行就是谁， 箭头函数的this固定不动，取决于在哪声明的这个函数（除非用call,apply,bind强行改变this指向） 普通function：this跟着执行人走

箭头函数注意:

1. this问题, 定义函数所在的对象或是父级对象是谁，不在是运行时所在的对象如window

2. 箭头函数里面没有arguments, 用 ‘...’

3. 箭头函数不能当构造函数

JavaScript 中，this 关键字的作用是什么？

参考答案：

笼统的说，关键字 this 指向当前对象。比如，顶级代码中的 this 指向全局对象；在

指定元素事件的时候，this 指定当前发生事件的元素对象。

对于嵌套函数，普通this：this作为方法被调用，指向调用它的对象。作为函数调用指向window,或undefined(严格模式)

 var o = {

        m: function () {

            console.log(this);  //{m: ƒ}

            f();

            function f() {

                console.log(this);    //window

            }

        }

    };

    o.m();

**4、Array与JSON方法**

**原生对象扩展**

Array扩展：map,reduce,filter,forEach

模板字符串

json写法，

**map()** 映射--对应：返回新数组，原数组处理后的值。按原数组顺序依次处理（简单说就是进去10个出来还是10个，一一对应）

let arr = [68,53,12,98,65];

var result = arr.map((item)=>{

  return item>60?'及格':'不及格'

})

console.log(result)                //结果: ["及格", "不及格", "不及格", "及格", "及格"]

**reduce()**作为累加器，数组中的每个值（从左到右）缩减成一个值。一般用在求和，n=>1(多个值变成一个值)

let arr = [1,2,3,4];

// 第一次：tatal是1,当前值是下一个2,依次；结果：求和1,当前2,索引1

let result = arr.reduce((total,curVal,curIdx,arr)=>{

  return total+curVal;

})

console.log(result)   //结果：10

filter 根据条件来筛选过滤，进去多个出来几个不一定

let arr = [1,2,3,4,5,6,7,8];

var result = arr.filter(item=>{

  return item%2==0;

})

console.log(result);              //结果：[2, 4, 6, 8]

forEach 遍历就是循环 突出所有的都走一遍，进去几个出来几个，没返回值，

let arr = [1,2,3];

var result = arr.forEach((item,index)=>{

    console.log(`第${index}个是${item}`);

})

/\*

//结果：

 第0个是1

 第1个是2

 第2个是3

\*/

7、字符串模板

`${变量}` ``反单引号 代替字符串拼接，可以换行

**8、json 写法、JSON对象**

JSON.stringify() 序列化,给一个json,出来一个字符串 ，字符串写法JSON.stringify()结果有单引号如：‘{"a":12，"b":5，"name":"blue"} ’

JSON.parse() 给一个字符串 进行解析，还原成json

JSON.stringify({a:12,b:5}) =>'{"a":12,"b":5}' console.log中''省略了，出来的里边都有双引号

JSON.parse('{"a":12,"b":5}') =>结果：{a: 12, b: 5} parse()里必须是里层双引外层单引号

**错误写法**

let arr = '{a:12,b:5}'; console.log(JSON.parse(arr)) //报错，里边必须加双引号"a" “b”

JSON.parse("{a:12,b:5,'name':'blue'}") //报错，必须是外边单引，里边双引

**附：call与apply**

作用：相同都是应用某一对象的一个方法，用另一个对象替换当前对象，一般用在es5的继承上

**call()方法** 第一个参数和apply()方法的一样，但是传递给函数的参数必须列举出来。

**语法：**call(对象，值必须列举出来);

**apply()方法** 接收两个参数，一个是函数运行的作用域（this），另一个是参数数组。

**语法：**apply(对象， [必须是数组]);

不同点例

function add(c,d){

     return this.a + this.b + c + d;

}

var s = {a:1, b:2};

console.log(add.call(s,3,4)); // 1+2+3+4 = 10

console.log(add.apply(s,[5,6])); // 1+2+5+6 = 14

**9、面向对象**ES5与ES6的区别?

ES5面象对象--假的

ES5没有系统统一的写法，处于自己摸索的状态（例如两个人写自己的库，一互用就会有问题了）ES5中没有class这一说法 ，它是用函数完成的功能，使用函数声明类，Person即是类也是构造函数，

**ES6面象对象-优点**

完全解决了统一的问题

提供了四个新的关键字，用于解决上面的问题

class ：类声明

constructor：构造函数/构造器

extends：继承

super：超类/父类

//有单独的类声明，构造函数声明

class Person{ constructor(){ } }

继承

Class Worker extends Person { constructor(){ super() } }

例：Es5继承

//父类Person 子类Worker 有单独的showJob在单写，其他的继承Person

function Person(name,age){

    this.name = name;this.age = age;

}

Person.prototype.showName = function(){ console.log(this.name)}

Person.prototype.showAge = function(){ console.log(this.age)}

function Worker(name,age,job){

    Person.call(this,name,age); //call方法继承person属性和方法

    this.job = job;

}

Worker.prototype = new Person(); // worker类也就继承了

Worker.prototype.constructor = Worker;

Worker.prototype.showJob = function(){

    console.log(this.job)

}

// 继承父类方法

var w = new Worker('YJUI',18,'随便');

w.showName();

w.showAge();

w.showJob();

例：Es6继承

// class类声明，构造函数声明constructor

class Person{

    constructor(name,age){

        this.name =name; this.age = age;

    }

    showName(){console.log(this.name)}

    showAge(){console.log(this.age)}

}

class Worker extends Person {

    constructor(name,age,job){

        super(name,age);  //继承的属性放里

        this.job = job; //子类自己的属性

    }

    // 子类自己的方法

    showJob(){console.log(this.job)}

}

var w= new Worker('YJUI',18,'随便');

w.showName();   //结果  YJUI

w.showAge();    //结果  18

w.showJob()     //结果  随便

**10、ES6模块系统**

export (输出) import(导入)

**注意：**

./必须要写，因为webpack是nodejs写的东西，必须遵循nodejs的规定

**多用在**webpack.config.js中module.exports = {}

需要对外输出json ——>{},CMD写法主要给nodejs、nodejs遵循CMD，

./ 用webpack就必须写，指当前文件（按ES6本身标准可以不写）但现在用webpack编译，webpack是nodejs写的要遵循nodejs约定，也就是说./是nodejs的规定，

import \* as mod1 from './mod1'; as 导入所有成员取个共同的名字叫mod1,从当前目录下mod1.js取

导出（export）的几个情况：可以导出变量、常量、一堆变量、函数，class

例：import 导入mod1.js中export内容

Export:

    mod1.js: export let a=2

    index.js :  import  {a} from '@/assets/mod1' 取a

    或import  \* as name from '@/assets/mod1'  取name.a

 Export default:

    Mod1.js:let a=4;let b=6;export default b;

    Index.js: import  mod from '@/assets/mod1';console.log(mod) //6

（1）export 和 export default 的区别

Export： import输出要加{} 一个也加{},export名与import名必须一致对应，不加{}可以起别名用as,import   \* as name from './mod1.js';

export default： import输出名随便，，一个模块只有一个默认输出，只能用一次，所以import后不用{}，唯一对应

import \* as mod1 from 'xxx' //export 与export default都可以as后名随意，区别在于export取值mod1.变量名；export defaul取值mod1.default.变量名

import ./ 用webpack就必须写，指当前文件（按ES6本身标准可以不写）但现在用webpack编译，webpack是nodejs写的要遵循nodejs约定，

9、Promise：解决异步回调问题

传统方式解决异步回调问题方式：大部分用回调函数，事件，相当于ajax嵌套

11、异步同步？

异步操作：同时进行多个操作，用户体验好（如用户名检查，输入完就检查）. 异步缺点： 好用但写起来麻烦

同步操作：一次只能进行一次操作，用户体验不好，按顺序执行,优点：清晰，

**Promise**

即有异步操作优势（不用卡住可同时进行多个操作）也可以像同步一样简单的写法

异步：then成功回调-> resolve=>解决，失败回调->reject=>拒绝

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

    $.ajax({

        url:'data/1.txt',

        dataType:'json',

        success(arr){

            resolve(arr);

        },

        error(res){

            reject(res);

        }

    })

 })

 p.then(function(arr){

      alert('成功')

      console.log(arr)   //12,5,8

 },function(res){

     alert('失败')

 })

总结：Promise 本身不能算是对异步操作的处理只是一个封装，因为不同的异步操作的表现形式不一样，Promise只是给一个统一的格式，统一模板，按这个封装，不管是什么，都有resolve和reject，调取就可以

Jquery的$.ajax本身就是一个Promise，直接用then的写法

 $.ajax({

   "url":'data/1.txt',

   dataType:"json"

}).then(arr=>{  //成功

    alert(arr)

},res=>{        //失败

    alert('失败')

})

**Promise.all([])：**统一做一个then。必须全都成功，有一个失败了全失败。要求所有东西都读取完了之后会给一个统一的结果,里边是数组。Promise.all()虽然好但不能解决所有的问题，

Promise.all([

    $.ajax({url:"data/1.txt",dataType:"json"}),

    $.ajax({url:"data/2.txt",dataType:"json"}),

    $.ajax({url:"data/3.txt",dataType:"json"})

]).then(arr=>{

    console.log(arr)

    // alert('成功')

},res=>{

    alert('失败')

})

**Promise.all()不能处理的情况：**Promise.all()一门心思读到底，这几个全都读下来。并不是说第一个读完判断一下，然后在读后边的，这种promise.all()处理不了；

有一种情况不能用Promise.all()，就是根据前一请求数据来读后一请求数据时不适合,利用前边数据来指导后边的数据:X

如非要用Promise处理这样的逻辑只能这样表示，但和原始的ajax请求没什么太大的区别

ajax('http://taobao.com/api/user').then(user\_data=>{

    if(user\_data.vip){

      ajax('http:/taobao.com/api/vip\_items').then()

    }

},error=>{

  alert('error')

})

**async/await特点：**

async虽然本身很特殊但调用的时候当普通函数用就行

async是函数的一个特殊形式，是一个语法，表示声明函数中是包含异步操作的，

await哪个是异步的哪个是同步的，程序不知道，只要加上await表明就是异步的，是等待的意思，有await标注那一行要等待操作结束后再往下走 ，也可以顺便把数据收集起来 let data = await $.ajax();

只是给函数加了个修饰， 告诉编译器这不是普通的函数，是有一些需要暂停的操作，会用await标记出来

现在写代码习惯中所有的异步操作都用了async/await

普通函数：一旦开始运行就不会停，直到代码执行完，

async函数-能够“暂停” ，是语法糖，会把这个大函数拆分成很多的小函数，执行第一个也是从头到尾 ，执行完要等着操作完成在执行第二个小函数，

async会暂停在执行，碰着await就暂停一会，到哪执行到哪等待是由开发人决定的

语法糖：往往给程序员提供了更实用的编码方式，有益于更好的编码风格，更易读。

写法和同步一样方便，但是异步的，$.ajax执行完之后也执行了then

async function show(){

    xxx;

    xxx;

    let data =  await $.ajax();

    xxx;

}

 show();

10、数组:

arr.forEach() ：代替普通for，forEach接收两个参数：第二个是this指向，很少用

arr.map() 重新整理数据结构: return 做数据交互 "映射",和后台打交到都会用到

arr.filter(): 过滤，过滤一些不合格“元素”， 如果回调函数返回true，就留下来return

arr.some(): 类似查找, 数组里面某一个元素符合条件，返回true

arr.every(): 数组里面所有的元素都要符合条件，才返回true

arr.reduce() ：从左往右执行

arr.reduce(prev,item,index,arr) ; prev上次计算的结果

arr.reduceRight() ： 从右往左

for...of 语句创建一个循环来迭代可迭代的对象,ES6替代 for...in 和 forEach()

for...of 允许你遍历 Arrays（数组）, Strings（字符串）, Maps（映射）, Sets（集合）等可迭代的数据结构等。

arr.keys() 数组下标

arr.entries() 数组某一项

Array.from:

作用: 把类数组就是伪数组(获取一组元素、arguments...) 对象转成数组

类数组点型的特点有length

Array.of(): 把一组值，转成数组

arr.find(): 查找，找出第一个符合条件的数组成员，如果没返回有找到，undefined,return返回一个值

arr.fill() 填充 arr.fill(填充的东西, 开始位置, 结束位置不包含);

arr.indexOf() 返回位置，没找到返-1

arr.includes() 返回true/false

Object.is(): 用来比较两个值是否相等

Object.assign(): 用来合并对象

语法： let json = Object.assign({}, defaults, options);

Object.keys()

Object.entries();

Object.values();

Node.js

1. 介绍

nodejs运行在服务器端，前后端代码整合

Node.js有自己的模块系统，因为Node.js早于ES6出现，nodejs是遵循CMD规范，所以对CMD比较熟悉的就好上手很多【cmd有requirejs等】

使用：安node.js 自带npm,

npm install –g xxx

2、npm和cnpm 的区别？

npm的源在国外

cnpm在国内 【c不是china】

npm和cnpm装的包是不能混用的，同一个项目就用其中一个，用注了，一混用就会有问题

VUE

**1、vue指令**

**v-bind**: 用于属性的单向绑定，可简写为 ':' v-bind:title="a"

**v-on**: 用于事件的绑定，可简写为'@';  v-on:click ==@click

v-model 双向绑定 多用于input select

**v-text**: 同插值表达是作用一样，但是会覆盖原本的内容；

**v-html**: 将内容以html元素渲染；

v-for循环输出 v-for ="(val,key) in data" ，key作用：区分元素、提高性能

v-if/v-else/v-else-if

v-once只会渲染解析一次

v-prev 预编译指令让vue跳过这个节点不编译原样输出。写书文档能用到

v-if 元素真的被删掉 ,只剩下一行注释 ，<!-------->占位符

v-show 元素只是隐藏了，display

v-cloak 防止页面加载时出现vuejs的变量名而设计。解决vue代码加载闪烁问题

[v-cloak]{

display: none;

}

2、v-if与v-show区别？

v-if 元素真的被删掉 ,只剩下一行注释 ，<!-------->占位符

v-show 元素只是隐藏了，display

v-show 用于频繁显示隐藏,【隐藏在显示比直接删除显示快】v-if用的更多。大量的隐藏-也会影响性能。v-show某些元素隐藏了也会起作用-比如：表单，【v-if不会有这个问题】

**2、事件**

v-on v-on:click="xxx" 等价于 @click="xxx"

**3、事件修饰符**

(1).stop—阻止冒泡 @click.stop

(2).prevent—阻止默认事件:

按键盘这个行为要干掉它，@keydown.prevent

如表单提交按钮submit或在页面上点右键，自动出现下拉菜单

(3).self—只接收自身的事件（冒泡上来的不要）嵌套点击中父级加.self

阻止冒泡二选一像.self放div或.stop放button上

<div @click.self=‘divFn’>

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

</div>

总结： .self和.stop

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

**.once**: 事件只触发一次；

.passive 告诉浏览器你不想阻止事件的默认行为

4、**filters--过滤器**

作用：接收输入的数据->转换->输出的结果 有return

filters:{} {{a | Upper }}

Upper: function (value) { return value.toUpperCase() },

##### 例：时间戳转日期格式，千位分隔符、转成万单位、首字母大写、保留两位小数

解答：为什么这里用filter要比用methods好？

（1）filters本身专事专用，仅是用来做转换的，用filters有个预期是要开始转数据了；methods本身可以做任何事情，

（2）filters语法更简洁

**4、computed ——计算后的数据——在真实的数据之外包一层有return**

**总结：**

**应用场景**

(1)简单的一些小计算可以直接用模板内的表达式计算，比较复杂一点的就建议使用“计算属性来运算了”，也方便后期的维护；

(2)computed适合比较单纯的数据改动，处理完后返回一个新的数据return,页中使用新变量

(3)每个计算属性都有一个getter函数 和 setter函数，下面的示例只是用了computed的唯一默认属性，就是getter , setter一般用来手动修改数据

默认的写法是get()，set()可以改变新数据值

(4)写法：

**computed与watch都可以这样表示：a(){} 同于 a:function(){} 同于 'a'(){}同于'a':function(){}**

作用：

1、可以控制对数据的操作

2、缓存：当数据不变，get每次都不用重新计算，如算税有很多没变用会上，快很多（避免无效的计算），只有在相数据发生改变时才会重新求值执行函数。

3、想改变a的值，要设置set()，不加set的a只有get()是只读的。例中加set后vm.a设置成5倍数{{a}}就会变了。不加set改vm.a值会提示没设setting报错。

直接这样写a(){return this.true\_a+5},同于get()里的，

应用场景：加set才能改变a值，在console.log改vm.a只有5的倍数才能改变页中a

<div id ="app">{{a}}</div>

var vm =new Vue({

            el:'#app',

            data：{true\_a:5},

            computed:{

                a(){return  this.true\_a+5 } //同于get()里的

                a:{

                    get(){ return this.true\_a+5 },

                    set(val){

                        if(val%5==0){

                            this.true\_a = val;

                        }

                    }

                }

            }

        })

缓存：Price不变不执行total函数这就叫缓存{{total}}。改vm.price可与页面同步。改vm.total报错，加上set可以set(val){this.price=val;}能同步vm.total

var vm =new Vue({

      el:'#app',

      data:{ price:10,},

      computed:{

          total:function(){return this.price+100; }

      }

})

5、watch

**watch ——监听数据的修改-没return**

说明：

watch函数名必须和data名一样，监听data里数据。接收两个参数值，（newval,oldval）变化后值和变化前值

类似于事件——当某个数据被修改了，可以得到通知，**有深度监听和浅度监听**

浅度监听：watch在默认的情况下只能监听表层vm.json=?会出现变了，vm.json.a监听不到不现在变了，

data:{ json:{a:12,b:5} },watch:{ json(){ console.log('json变了') } }

深度监听vm.json与vm.json.a都触发watch：deep:true；也监听内部；性能不高

immediate讲解作用：true在程序初始化之后watch就立即发生一次执行 “json变了”

用在页码上初始handler(){ ajax(...) //请求数据操作 }

watch:{

json:{

       deep:true,

immediate:true,

       handler(){

        console.log('json变了')

      }

 }

}

如果即希望深度监听也希望性能高些：选择性去做，浅层的够用就用浅层的，精确盯着某个要修改的值，用浅度监听找监听的值 ,vm.json与vm.json.a都会监听到

Watch:{

'json.a':function(){

      console.log('json变了')

}

}

**watch监听数组**

data:{ arr:[1,2,3,4,5] }, watch:{ 'arr.2'(){ console.log('数组变了') } },

watch监听数组某个下标值vm.arr.2=？2是下标也不会触发watch，只有vm.arr.push(6)有变化。因为vue对json【可以精准某个值用浅度写法vm.json.a】和数组【精准浅度写法也无用】。要用vm.$set(数据，key，val) Vue.set(数据，key，val)方法

vm.$set(vm.arr,2,33)-> [1,2,33,4,5]

**watch需要注意的:不要循环watch**

watch:{ a(){this.a++;} }, 这样不可

6、组件

1. 全局组件——任何地方：Vue.component('xxx',{ data(){ return {}; }, })
2. 局部组件——父组件之内components:{}
3. 类声明组件——var cmp = Vue.extend({})挂局部上/全局上Vue.component(' cmp ', cmp)
4. 动态组件——is
5. 组件传参

组件注意：

* 1. 组件要写在vm实例之上，
  2. 中划线组件名 页中<login-dialog></login-dialog> 组件js引入可以loginDialog
  3. new Vue能用的，组件也能用——filters、computed、watch、\*
  4. 只有在全局组件中能用props

注意 页中不能是<myButton></myButton>当然对应的 js中myButton与 my-button都不行

页中能是<my-button></my-button> 对应js中myButton、my-button都行

总结html中组件名不能是中间大写的形式（如：myButton）要用横岗代替（如：my-button）。

对应js都行。

全局与局部组件写法：

<login-dialog :aa='a'><aaa></aaa></login-dialog>

结果：这是全局组件12——我是局部组件12

Vue.component('loginDialog',{

       props:['aa'],

       template:`<h1 >这是全局组件{{aa}}<aaa :bb = "aa"></aaa></h1>`,

       components:{

         'aaa':{

                props:['bb'],

                template:`<span>——我是局部组件{{bb}}</span>`,

          }

       }

 })

局部组件写法：

1、局部组件放vm实例的components中

2、全局组件里放局部组件数据用:aa='a' 的props传，全局与局部组件都能用props

3、用类的方式var cmp =Vue.extend({}) 把cmp放vm的局部components

 let vm = new Vue({

        el:'#root',

        data:{ a:12},

        components:{

        'aaa':{ props:['aa'],

             template:`<span>我是局部组件{{aa}}</span>`,

          }

        }

})

var child={

      template:`<button @click="add">我是局部组件{{a}}</button>`,

methods:{ add(){ this.a++;}}

};

let vm = new Vue({

     el:'#root',

     components:{

          'loginDialog':child    //loginDialog:child加不加引号都行

     }

})

类声有组件Vue.extend({})

let name= Vue.extend({ //类声明组件

    template:`<span>abc</span>`

})

//挂成全局组件

Vue.component('name',name)

let vm = new Vue({el:'#root',

     // 挂局部上

components:{ cmp}

})

//导出写法用 Vue.component('vue-head',Vue.header)或components:{'vue-head':Vue.header } <vue-head :aa="a"><vue-head>

(function(vue){

   var template = `<div>{{aa}}</div>`

    //head组件名

   var head = vue.extend({

       template:template,

       props:['aa'],

   })\

   vue.header = head; //vue上放header

})(Vue)

动态组件

**作用：**

**1、**改变挂载的组件，用 is 属性来切换组件。 <component :is="组件名"></component> 可以用v-for

<component :is="组件名"> </component>

:class的几种写法

(1)字符串 ： :class ="abb"

1. 数组 ： :class="['box','aa','bb']"
2. json ： :class ="{ box:true, aaa:false, bbb:true }"

(4)组合 ：:class ="['btn','bnt-dark',{active:index==i}]"

(5)三目 ： :class="i==value?'active':''"

*Props两种使用方法：*

*1、子组件接收父组件传的值props:[‘msg’]*

*2、参数约束：*具体约束类型，是否必传，范围操作用

default:默认值如:leixing不传时required:false

<my-button :leixing="a"></my-button>

props:{

            leixing:{

                type:String,   //a类型必须为String

                required:true, //为true必须传:leixing='a'

                validator(arg){ //接收leixing值的范围判断

                    //这里arg是default输出子组件default,不是就false报错

                    if(arg=='default'){

                        return true;

                    }else {

                        return false;

                    }

                }

            }

        },

V-model

父组件 v-model="a"等同于 ：value="a" @input="fn"

子组件是@input ="$emit('input',$event,target.value)"

4-2 、.sync

(1).sync作用

v-model是实现一个属性双向绑定的，那多个时就用.synce

.sync是vue中用于实现简单的“双向绑定”

vue的prop是单向下行绑定：父级的prop的更新会向下流动到子组件中，但是反过来不行。可是有些情况，我们需要对prop进行“双向绑定”。这个时候，就可以用.sync来解决

说明：

v-model是@input写死的一个标签上就一个，实现一个tab，@update:xx要多少有多少，只要xx变了就有。.sync一个标签能有多个实现多个tab

.sync 的原理和v-model差不多，只是在v-model的基础上扩展了要多少有多少。

(2).sync语法：

即然什么名都行，那就往语法上靠

:xx.sync = :xx="value"+@update:xx ="赋值"

这里update:a/update:b就是事件

父:a.sync=cur1 :b.sync=cur2

等同于 :a="a" :b="b" @update:a="val=>a=val" @update:b="val=>b=val"

子：this.$emit('update:a',this.a+1);  this.$emit('update:b',this.b+1);

(3).sync用法用例：

<text-document :title.sync="doc.title"></text-document> 当子组件需要更新 title 的值时，它需要显式地触发一个更新事件：this.$emit('update:title', newValue)

这样title的属性在子组件内部更新，父组件也能感知的到，实现了“双向绑定”。

例.在子组件点击每次都+1，实现双向绑定

<div id="root" v-cloak>a={{cur1}} b={{cur2}}

<cmp1 :a="cur1" :b="cur2" @update:a = "fn1" @update:b = "fn2"></cmp1>

<cmp1 :a="cur1" :b="cur2"  @update:a="val=>cur1=val" @update:b="val=>cur2=val" ></cmp1>

同于<cmp1 :a.sync="cur1" :b.sync="cur2" ></cmp1>

    </div>

       <script src="vue2.js"></script>

       <script type="text/javascript">

       var cmp1 = Vue.extend ({

           props:['a','b'],

           template:`<div><input type="button" @click="fn" value="+1按钮"/></div>`,

           methods:{

              fn(){

                this.$emit('update:a',this.a+1);

                this.$emit('update:b',this.b+1);

              }

           }

       })

       let vm = new Vue({

           el:'#root',

           data:{

               cur1:0,cur2:0

           },

           components:{ cmp1},

           methods:{//其它俩种写示不加fn1/fn2

                fn1(val){ this.cur1= val; },

                fn2(val){this.cur2= val; }

           }

       })

       </script>

**小结**

一个组件需要提供多个双向绑定的属性时使用，只能选用一个属性来提供 v-model 功能，但如果有其他属性也要提供双向绑定，就需要.sync

slot插槽应用扩展：占位符

**作用**：vue中，经常需要向一个组件传递内容。为了解决这个问题，官方引入了插槽(slot)的概念。不用插槽内容会丢失

<cmp1>直接这么写接收不到</cmp1>

**插槽分类**

匿名插槽：它不需要设置name属性，也叫它单个插槽或者默认插槽。与具名插槽相对，。（它隐藏的name属性为default。） <slot>默认</slot>

具名插槽：当一个组件中需要定义多个插槽时，就需要用到具名插槽。 有name

需要在slot标签中添加 name属性，属性值任意写；

在引用组件时，通过添加 slot = “属性值” 来关联对应的插槽。

作用域插槽

**一个slot区分，**

*当组件渲染的时候，<slot></slot> 将会被替换为“Your Profile”。*

<cmp1> Your Profile</cmp1> <slot>111</slot>

父组件不提供任何插槽内容时 默认显示Submit,不认slot有没有name，只要cmp组件是空就取默认值

<cmp1></cmp1> <slot>Submit</slot>

显示默认值情况:

<cmp1>aaa</cmp1>//找不找name为title的就默认

<cmp1></cmp1>

<cmp1 #title></cmp1>

<slot name="title">默认值</slot>

**slot写法，**

语法：a:12

父:slot=”名“

子：<slot name="名"></slot>

父级:<cmp1>12</cmp1>           //a:12 结果：12   默认组件传值就用<slot>不加name

子级写法:<slot></slot>

父级:    <cmp1><div slot="名字">abc</div></cmp1>

子级写法:<slot name="名字"></slot>    //结果：abc

父级:    <cmp1><template v-slot:名字>abc1111</template></cmp1>   同于 <cmp1><template #名字>abc</template></cmp1>同于<cmp1  #user><template>abc</template></cmp1>

子级写法:<slot name="名字"></slot>    //结果：abc1111

父级:   <cmp1 #title>abc</cmp1 >   //同于： <cmp1 v-slot:title>abc</cmp1>

子级写法:<slot name="title"></slot>    //结果：abc

父 <cmp1 title="用户标题"></cmp1>   //结果：子组件用户标题

子 <slot name="title">{{title}}</slot> props:['title'], //直接写{{title}} props:['title']，可不写slot 这里已实践

父 <cmp1>abc</cmp1>    //父没定义插槽取slot默认值，结果：默认值

子 <slot name="title">默认值</slot>

父 <cmp1 #title>123</cmp1>    //父定义slot取组件值，结果：123

子 <slot name="title">默认值</slot>

slot插槽作用域

什么：可以给插槽template里传参，也就从里边往外边传参数子组件slot——>父组件template

父：<template slot-scope="scope">{{scope}}</template> //同于 <div slot-scope="scope">{{scope}}</div>

子：<slot a="12" :b="55"></slot>   //加:是变量，不加是String

结果:{ "a": "12", "b": 55 }

Router

（1）配路由在router/index.js

import Vue from 'vue'

import Router from 'vue-router'

Vue.use(Router);

import index from '@/components/index';

import cmp1 from '@/components/cmp1';

let router = new Router({

mode:hash', //默认有#

    routes:[

        {path:'/',component:index},

        {path:'/cmp1',component:cmp1},

    ]

})

export default router;

最外层index.js 把配置的路由引进来

import router from './router';

APP.vue

<button @click="fn1">首页</button> 函数里内容this.$router.push('/')

<button @click="fn2">新闻</button> 函数里内容this.$router.push('/cmp1')

<router-view/>

同于

<a href="#/">首页</a>

<a href="#/news">新闻页</a>

<router-view/>

同于

<router-link to="/">首页</router-link>

<router-link to="/cmp1">新闻</router-link>

<router-view/>

(2)<router-link></router-link>精确匹配选中样式.router-link-exact-active{}

(3)路由三种模式（默认hash）

访问首页 / 与 /cmp1

1. history——地址变页面不刷新

history下必须加上如下，否则在<http://localhost:3002/cmp1下f5>刷新会真像服务器请求，服务器没有get就是404报错，加上true可正常

devServer:{

    historyApiFallback:true,

}

没#号 例：<http://localhost:3002/> <http://localhost:3002/cmp1>

2、hash——页面不刷新

有#号 例：http://localhost:3002/#[/](http://localhost:3002/) <http://localhost:3002/#/cmp1>

——像描点，默认模式如： <http://localhost:8080/#/news>

3、abstract——不存在地址栏、不改地址也不改hash，一般受于后台给前端

不存在有没有#号,只要是当前链接就能切换，如：保留输入的地址切换地址不在变化 如http://localhost:3002， 一直这个下切换

(4)**处理404访问不存在页面**

{path: '\*'} path中\*通吃，通配 找不到的页面都走\*的组件，代替404，router/index.js

let router = new Router({

    routes:[

        {path:'/',component:index}, {path:'/cmp1',component:cmp1},

        {path: '\*',component: notfound} //找不到cmp3显示notfound东西

    ]

})

App.vue

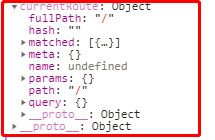
这里的找不到页面指路由配置中没引入cmp3没加：{path:'/cmp3',component:cmp3}

App.vue中用了就会空白就表示404如<router-link to="/cmp3">新闻</router-link>

等同关系：

this.$router.currentRoute == this.$route //内是路由相关参数

$router == new Router({}) this.$router.push('/')



**(4) 路由传参**—**params与query**

**Params方式**

如：<http://localhost:3001/news/1323/yjui> //对应写法:id/:name

获参params二者同：

 this.$route.params == s.$router.currentRoute.params

C:\Users\Administrator\AppData\Local\YNote\data\qqE684D2BB0B04DBEB1BF21C045B8F3FA5\a00ccdbbba864981847ceca45a7fa54e\1598928707(1).png

传参：{path: '/news/:id/:name',component: news},            //index.js

传参：<router-link to="/news/1323/yjui">新闻</router-link>  //news.vue

取参与赋值

let {id,name} =  this.$route.params/this.$router.currentRoute.params

console.log(id,name)        //1323 yjui

**query方式—**?a=12&b=5

url格式：<http://localhost:3001/news/1323/yjui?a=12&b=5> //这种写法都是字符串

格式 :

:to="{path:'',query:{}}" 等同于 this.$router.push({path:'',query:{}})

网址都为：http://localhost:3002/#/cmp1/3123/yjui?a=33&b=55

<router-link :to="{path:'/news/3123/news',query:{a:33,b:55}}">新闻</router-link>

 等价于this.$router.push({path: '/news/3123/news',query: {a:33,b:55}}) //$router->new Router())

取值与获参query：

this.$router.currentRoute.query == this.$route.query

**(5) 组件相同参数不同获取当前参数问题\_用watch**

**App.vue**

**改成这样结果也一样：**

**<router-link :to="{path:'/news/1323/news'}">首页</router-link>**

**<router-link :to="{path:'/news/123/hot'}">hot</router-link>**

<template>

  <div id="app">

    <router-link to="/news/1323/news">首页</router-link>

    <router-link to="/news/123/hot">hot</router-link>

    <router-view />

  </div>

</template>

<script>

export default {

  created() {

    let { id, name } = this.$router.currentRoute.params;

    console.log(id, name, this.$route.params);

  },

};

</script>

**这种形式只能取到第一次初始加载的参数不能实时触发改变：**

created(){

let {id,name} = this.$router.currentRoute.params

console.log([id,name])

}

**Router/index.js**

import Vue from 'vue'

import Router from 'vue-router'

Vue.use(Router);

import cmp1 from '@/components/cmp1';

export default new Router({

    mode: 'history',

    routes: [

        { path: '/news/:id/:name', component: cmp1 },

    ]

});

总结组件相同参数不同实时监听的三种写法：

**（1）**直接把created改成   updated() {}

**（2）**身上某个东西变了能得到通知watch， 这里的数据是路由参数数据为$route进行监听

随着<http://localhost:3002/news/1323/news> 与<http://localhost:3002/news/123/hot>的切换

它显示：

**C:\Users\ADMINI~1\AppData\Local\Temp\1602319573(1).png**

updated(){

  let {id,name} = this.$router.currentRoute.params

   console.log([id,name])

},//或

*created(){*

*let updateData=()=>{*

*let {id,name} = this.$router.currentRoute.params*

*console.log([id,name])//  ["11", "news"]  ["22", "hot"]*

*}*

*updateData();*

*this.$watch('$route',updateData)*

*}//或*

export default {

  watch: {

    $route() {

      let { id, name } = this.$router.currentRoute.params;

      console.log(id, name, this.$route.params);

    },

  },

};

附加：组件什么时候会更新/重新渲染（updated）：总结就是data变了，自己的或是父级

1、data变了

2、props变了会更新也就是父级参数变了

3、强制更新——几乎用不上

**(6) Router-path两种模式与嵌套**

1. 绝对路径——path写法有绝对路径/。推荐绝对路径 默认mode:hash

访问时path加在/#/后面

访问：一层：<http://localhost:3002/#/index> 二层：<http://localhost:3002/#/aaa>

分别结果：App CMP1 AppCMP1CMP2

{path:'/index',component:cmp1,children:[

    {path:'/aaa',component:cmp2}       //绝对路径直接找

]}

（2）相对路径

访问：二层：<http://localhost:3002/#/index/aaa>

{path:'/index',component: cmp1,children:[

    {path:'aaa',component:cmp2} //相对路径逐级找

]}

嵌套主要用到：

（1）path相对或绝对路径访问与children

（2）<router-view/>

App.vue:<div id="app">App<router-view/></div>

cmp1: <div>CMP1<router-view/></div>

**(7) 命名路由**

**作用**:路由配置是vue使用的基础，采用传统方式麻烦且不清晰，而命名路由无论path多长多繁琐，都能直接通过name就匹配到了，十分方便，所以，强烈推荐使用命名路由的方式

访问：

<http://localhost:3001/#/index/12?a=1&b=2>

Router/index.js

{path:'/index/:id',name:'index',component:cmp1},

App.vue

<router-link :to="{path:'/index/12',query:{a:1,b:2}}">首页</router-link>

等同于

<router-link :to="{name:'index',params:{id:12},query:{a:1,b:2}}">首页</router-link>

注：

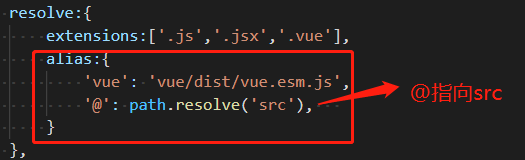
path中路径要与to中path一致如：

正确写法

routes:[{path:'/index/:id'}]一致于：to="{path:'/index/12'}"

错误写法

routes:[{path:'/index/:id'}]不能：to="{path:'/12'}"



**(8)Vue-Router路由钩子函数(导航守卫)**

守卫作用：过来新的地址要先经过守卫同意了才能进去 也就是next()

分类：全局和局部

路由钩子函数有三种：

  1：全局钩子： beforeEach、afterEach

  2：单个路由里面的钩子：beforeEnter、beforeLeave

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

语法：

**全局钩子函数：** 加在router/index.js上

//进入路由前触发

全局前置守卫【先函数在跳页】: router.beforeEach((to,from,next)=>{})

//进入路由后触发

全局后置钩子【先跳页在执行】: router.afterEach((to,from)=>{})

全局形式如：

//to 去哪

//from 目前在哪

//next 如果同意了进入链接页就调用next()作用证明，加next()可跳到对应页,不加随便点哪url都不变没反应

var router = new Router({});

router.beforeEach((to,from,next)=>{});//router.afterEach((to,from)=>{})

export default router;

**局部钩子函数：放组件中**

//加next()后才跳页。在渲染组件的对应路由被confirm前调用。不能访问组件this

进入这个路由改Url：beforeRouteEnter(to,from,next){}

//加next()后才跳页。当前路由即同样的Foo组件，组件被复用时调用。能访问组件this。需要路由参数用watch获取路由地址变化的东西，还可以用beforeRouteUpdate也改变内容与链接跳转

当前路由改变改Url：beforeRouteUpdate(to,from,next){}

//加next()后才跳页。离开该组件的对应路由时调用。能访问组件this

离开该组件路由改Url ： beforeRouteLeave(to,from,next){}

**全局钩子函数与局部区别**

全局写法router.xxx 挂在router实例上； 全局写在 router/index.js中. 除了afterEach都有next()

局部不需挂router，局部写在组件中路由

Ref

作用：ref除了可以获取本页面的dom元素，还可以拿到子组件中的data和去调用子组件中的方法

父子组件事件监听写法两种

方法一：$emit写法

子:

<button @click="$emit('add', 12, 5)">add</button>

父：

<cmp1 @add="fn"/> methods: {fn(a,b){console.log(a, b);//12 5},},

方法二：ref写法$on接收，ref就是给这个组件起个名字

子:

<button @click="$emit('add', 12, 5)">add</button>

父：

$on用在mounted生命周期为加载完，created是dom未挂载不可

<cmp1 ref="cmp1" />

 mounted(){this.$refs.cmp1.$on('add',function(a,b){console.log(a,b);//12 5})}

这里证明 箭头函数可以合保证this不变

$once注意如父组件数据实现x++,data(){return x=0;}要指定this

//这里this组件是cmp1,这里点击是会没返应this不是App,惹祸的function(){})

实现x++：this.$refs.cmp1.$once('add',function(a,b){this.x++}

//这里this组件是App,this指针是App的

可实现x++：this.$refs.cmp1.$once('add',(a,b)=>{this.x++}

总结：

父组件中用@add接收和在js中用$on效果是一样的

有两种props是不一样的，如下：

给类传参用propsData 【new Blue】

**组件接收参数用props**

**1、vue生命周期：**

**beforeCreate 组件实例刚刚被创建,属性都没有**

**created 实例已经创建完成，属性已经绑定**

**beforeMount 模板编译之前（准备）**

**mounted 模板编译之后，代替之前ready \***

**beforeUpdate 组件更新之前 data数据变了 (用在$.watch('a',function){})**

**updated 组件更新完毕 \* (用在$.watch('a',function){})**

**beforeDestroy 组件销毁前**

**destroyed 组件销毁后**

2、在vue-router中要使用选中样式的方法有两种：

1、直接在路由main.js文件中配置linkActiveClass,active样式有定义

const router =new VueRouter({

linkActiveClass:'active',

routerConfig

});

2、在router-link中写入active-class

<router-link to="/home" tag="li" active-class="active">

<a href="javascript:;">首页</a>

</router-link>

1、active-class是哪个组件的属性？嵌套路由怎么定义？

答：vue-router模块的router-link组件。

.slide-down-enter-active, .slide-down-leave-active { }

.slide-down-enter ,

.slide-down-leave {

}

2、怎么定义vue-router的动态路由？怎么获取传过来的动态参数？

答：在router目录下的index.js文件中，对path属性加上/:id。 使用router对象的params.id

路由其他信息:

/detail/:id/age/:age

{{$route.params | json}} -> 当前参数 {'id':'001'}

{{$route.path}} -> 当前路径 /news/detail/001?a=1&b=2

{{$route.query | json}} -> 数据 { "a": "1", "b": "2" }

vue-router钩子函数

beforeRouteEnter ,beforeRouteLeave,

1.简要介绍Vuex原理

Vuex实现了一个单向数据流，在全局拥有一个State存放数据，当组件要更改State中的数据时，必须通过Mutation进行，Mutation同时提供了订阅者模式供外部插件调用获取State数据的更新。而当所有异步操作(常见于调用后端接口异步获取更新数据)或批量的同步操作需要走Action，但Action也是无法直接修改State的，还是需要通过Mutation来修改State的数据。最后，根据State的变化，渲染到视图上。

2.简要介绍各模块在流程中的功能：

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

3、vuex是什么，怎么使用，场景

答：vue框架中状态管理。在main.js引入store，注入。新建了一个目录store，….. export 。场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

App.vue

import {mapActions,mapGetters} from 'vuex'

computed:mapGetters([

//名字和getters.js返回的函数名要一样

'tabbarShow'

]),

computed:mapGetters([

'headShow',

'loading',

'footerShow'

]),

index.js

import Vuex from 'vuex'

Vue.use(Vuex);

export default new Vuex.Store({

actions,

//mutations是个组件，

modules:{

mutations

}

})

自己搭建过vue开发环境吗？

简单的目录结构:

|-index.html

|-main.js 入口文件

|-App.vue vue文件，官方推荐命名法,可以小写，为了方便都大写了

|-package.json 工程文件(项目依赖、名称、配置) 这个可以生成，其他手动建立

package.json npm init --yes 生成，用于对团队开发简单需要用到它，

|-webpack.config.js webpack配置文件

1、下载webpack webpack-dev-server

npm install webpack webpack-dev-server --save-dev

（1）webpack.config.js配置

配置webpack.config.js入口文件main.js；出口文件加上当前路径和统一打包build.js；相关loader把.vue,.js.style，图片，转换成正常代码，

babel把es6,78代码编译成ES5浏览器可以识别的

安装下载时最好加上--save-dev，好处：当同事拿到代码后直接install就可以用了，不紧把模块下到本地，而且会写在配置文件里边

（2）npm init 生成package.json

需求一边改一边刷新(加上端口避免占用)在里边配置一个，package.json中：

"scripts": {

"dev": "webpack-dev-server --inline --hot --port 8082" //一边改一边刷新

},

解析es6安装：npm install babel-loader babel-core babel-plugin-transform-runtime babel-preset-es2015 babel-runtime --save-dev

解析app.vue的安装：npm install vue-html-loader css-loader vue-style-loader vue-hot-reload-api@1.3.2 --save-dev

安装VUE:npm install vue@1.0.28 --save

说明--save放dependencies中, 项目必须依赖的框架，devD...开发依赖框架loader,bable

[d??p?nd?nsiz]依赖

安装所有依赖：npm install

(3)index.html

<script src="build.js"></script>

(4)main.js要引入vue

import Vue from 'vue'

import App from './App.vue'

new Vue({

el:'body',

components:{

app:App

}

});

3、vue-router有哪几种导航钩子？

1 、全局导航钩子两种钩子：前置守卫、后置钩子，

beforeEach

afterEach

2 、某个路由独享的导航钩子

beforeEnter

3 、路由组件上的导航钩子

beforeRouteEnter

beforeRouteUpdate (2.2 新增)

beforeRouteLeave

答：三种，

一种是全局导航钩子, 直接挂载在router实例上的：router.beforeEach(to,from,next)，

参数：有to（去的那个路由）、from（离开的路由）、next（一定要用这个函数才能去到下一个路由，如果不用就拦截）

//生成路由实例

const router=new VueRouter({

routes

});

// 全局导航钩子 直接挂载在router实例上的,跟踪路由

router.beforeEach((to, from, next) => {

console.log(to.path)

// next()

});

router.afterEach((to, from) => {

// do someting

});

不同于前置守卫，后置钩子并没有 next 函数，也不会改变导航本身

第二种：组件内的钩子App.vue；

const File = {

template: `<div>This is file</div>`,

//进入home页后显示

beforeRouteEnter(to, from, next) {

// do someting

// 在渲染该组件的对应路由被 confirm 前调用

},

beforeRouteUpdate(to, from, next) {

// do someting

// 在当前路由改变，但是依然渲染该组件是调用

},

//离开home页后显示

beforeRouteLeave(to, from ,next) {

// do someting

// 导航离开该组件的对应路由时被调用

}

}

第三种：单个路由独享的导航钩子，它是在路由配置上直接进行定义的：

在router.config.js中配置

进入主页时显示

//配置路由

cont router = new VueRouter({

routes: [

{

path: '/home',

component: Home,

beforeEnter: (to, from ,next) => {

// do someting

}

}

]

});

Vue 监听路由变化

watch:{

$route( to , from ){

// to , from 分别表示从哪跳转到哪，都是一个对象

// to.path ( 表示的是要跳转到的路由的地址 eg: /home );

}

}

4、scss是什么？安装使用的步骤是？有哪几大特性？

答：css的预编译。

使用步骤：

第一步：用npm 下三个loader（sass-loader、css-loader、node-sass）

第二步：在build目录找到webpack.config.js，在那个extensions属性中加一个拓展.scss

第三步：还是在同一个文件，配置一个module属性test,loader

第四步：然后在组件的style标签加上lang="scss"

有哪几大特性:

1、可以用变量，例如（$变量名称=值）；

2、可以用混合器，例如（）

3、可以嵌套

module.exports = {

module: {

rules: [

{

test: /\.(scss|css)$/,

use:[ 'style-loader','css-loader','sass-loader'],

},

],

},

};

5、mint-ui是什么？怎么使用？说出至少三个组件使用方法？

答：基于vue的前端组件库。

npm安装，然后import(import MintUI from 'mint-ui'

import 'mint-ui/lib/style.css')样式和js，vue.use（mintUi）全局引入。在单个组件局部引入：import {Toast} from ‘mint-ui’。

组件一：Toast(‘登录成功’)；组件二：mint-header；组件三：mint-Swipe

import { Button } from 'mint-ui';

Vue.component(Button.name, Button);

<mt-button type="default">default</mt-button>

import { Header } from 'mint-ui';

Vue.component(Header.name, Header);

<mt-header fixed title="固定在顶部"></mt-header>

6、v-model是什么？怎么使用？ vue中标签怎么绑定事件？

答：可以实现双向绑定，指令（v-class、v-for、v-if、v-show、v-on）。vue的model层的data属性。绑定事件：<input @click=doLog() />

7、axios是什么？怎么使用？描述使用它实现登录功能的流程？

https://chuansongme.com/n/394228451820

答：请求后台资源的模块。

Axios 是一个基于 promise 的 HTTP 库，可以工作于浏览器中，也可以在 node.js 中使用，提供了一个API用来处理 XMLHttpRequests 和 node 的 http 接口

可能很多人会疑问：用 jquery 的 get/post 不就很好了，为什么要用 Axios？原因主要有：

（1）Axios 支持 node.js，jquery 不支持

（2）Axios 基于 promise 语法标准，jquery 在 3.0 版本中才全面支持

（3）Axios 是一个小巧而专业的 HTTP 库，jquery 是一个大而全的库，如果有些场景不需要使用jquery的其他功能，只需要HTTP相关功能，这时使用 Axios 会更适合

除了 get/post，还可以请求 delete,head,put,patch

23、聊聊你对Vue.js的template编译的理解？

答：简而言之，就是先转化成AST树，再得到的render函数返回VNode（Vue的虚拟DOM节点）

详情步骤：

首先，通过compile编译器把template编译成AST语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile是createCompiler的返回值，createCompiler是用以创建编译器的。另外compile还负责合并option。

然后，AST会经过generate（将AST语法树转化成render funtion字符串的过程）得到render函数，render的返回值是VNode，VNode是Vue的虚拟DOM节点，里面有（标签名、子节点、文本等等）

4. 简述Vue的响应式原理

当一个Vue实例创建时，vue会遍历data选项的属性，用 Object.defineProperty 将它们转为 getter/setter并且在内部追踪相关依赖，在属性被访问和修改时通知变化。

每个组件实例都有相应的 watcher 程序实例，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 setter 被调用时，会通知 watcher 重新计算，从而致使它关联的组件得以更新。

jsonp原理

通过jquery的ajax进行跨域，采用jsonp方式实现的，它允许在服务器端生成script,tags返回至客户端，也就是动态生成javascript标签，通过javascript，callback的形式实现数据读取

typeof 返回类型与以上依次对应为 number object boolean string object object undefined

一、Less、Sass/Scss是什么?

1、Less：

是一种动态样式语言. 对CSS赋予了动态语言的特性，如变量、继承、运算、函数。

Less 既可以在客户端上运行 (支持IE 6+, Webkit, Firefox)，也可在服务端运行。

1.良好的CSS架构目标是什么？

预测、重用、扩展、维护

注意： 不管是什么方法，书写的顺序都是firefox的写在前面，IE7的写在中间，IE6的写在最后面。补充：IE6能识别\* ，但不能识别 !important,IE7能识别 \*，也能识别!important;FF不能识别 \*，但能识别!important;下划线” \_ “,IE6支持下划线，IE7和firefox均不支持下划线。

4、HTML5新页面元素5个以上

（1）figure元素：figure是个组合元素，可以带标题figcaption，一个figure只允许放置一个figcaption。

(2)details元素:details提供了一种替代Javascript的、将画面上局部区域进行展开或收缩的方法

(3）mark元素：mark元素表示页面需要突出显示或高亮显示的部分。

(4）progress元素：一般用于写进度条

(5） meter元素：定义度量衡。

5、针对不同浏览器写出box-shadow格式

-moz-box-shadow:3px 5px 5px #969696;

-webkit-box-shadow:3px 5px 5px #969696;

box-shadow:3px 5px 5px #969696;

filter: progid:DXImageTransform.Microsoft.Shadow(color='#969696', Direction=125, Strength=9);

6、UL ,OL,DL三种列表html结构

ul（无序列表）ol（有序列表）dl（自定义列表）

7、不换行设置哪些

white-space:nowrap

8、英文单词不发生词内换行

word-break: break-word;

9、display:none 和visibility:hidden 区别

.display:none是彻底消失，不在文档流中占位，浏览器也不会解析该元素；visibility:hidden是视觉上消失了，可以理解为透明度为0的效果，在文档流中占位，浏览器会解析该元素；

10、文字超出 。。。

word-break: break-word;

overflow: hidden;

text-overflow: ellipsis;

两行：

overflow: hidden;

text-overflow: ellipsis;

display: -webkit-box;

-webkit-line-clamp: 2;

-webkit-box-orient: vertical;

11、编写10000个随机整数排序

12，编写当前访问的设备类型

13，最优方式向<body>中添加10个<p>hello</p>,然后删除所有p标签

14，给html元素绑定click事件，使绑定事件后的li元素依然可以触发处理函数

<ul><li>1</li><li>2</li><li>3</li></ul>

15、结果

for(var i=0;i<5;i++){

setTimeout(function(){

console.log(i);

},1000)

}

55555

16、Function.bind函数相同功能

1运行:

var a=1;

function test(){

console.log(a); //undefined

var a=2;

console.log(this.a);//1 window

setTimeout(function(){

console.log(a);//3

},0)

return function(){

console.log(a);//2

a=3;

console.log(a);//3

}

}

test()();

2运行:

var a = 5;

function test(){

a=0;

console.log(a);

console.log(this.a)

console.log(this)

var a;

console.log(a)

}

test(); //0 5 Window 0

new test(); //0 undefined test{} 0

主要的区别就是this的指向变了

直接调用test()时，this的指向是window或global。

使用new调用 new test()时， this的指向变成被实例化的对象。

当作为构造函数来使用的时候，函数内的这个 this 就指向一个新的对象，这个对象是这个构造函数的实例，

以这种方式调用构造函数实际上会经历以下4个步骤：

(1) 创建一个新对象；

(2) 将构造函数的作用域赋给新对象（因此this 就指向了这个新对象）；

(3) 执行构造函数中的代码（为这个新对象添加属性）；

(4) 返回新对象。

3运行:

异步、作用域、闭包。

结果是：0 1 2 3 3 3

for (var i = 0; i < 3; i++) {

setTimeout(function() {

console.log(i);

}, 0);

console.log(i);

}

知识点：

因为是setTimeout是异步的。

正确的理解setTimeout的方式(注册事件)： 有两个参数，第一个参数是函数，第二参数是时间值。 调用setTimeout时，把函数参数，放到事件队列中。等主程序运行完，再调用。

setTimeout(function() {

console.log(1);

}, 0);

console.log(2); //先打印2，再打印1

这么写完，会弹出1吗。不会！！只是绑定事件而已！ 必须等我们去触发事件，比如去点击这个按钮，才会弹出1。

setTimeout也是这样的！只是绑定事件，等主程序运行完毕后，再去调用。

btn.onclick = function() {

alert(1);

};

for (var i = 0; i < 3; i++) {}

console.log(i); //3，也就说i可以在for循环体外访问到。所以是没有块级作用域。

运行程序看结果：

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global\_Objects/Promise/then

面试题：

//then() 方法返回一个 Promise 。它最多需要有两个参数：Promise 的成功和失败情况的回调函数。

// 这道题应该考察我JavaScript的运行机制的，让我理一下思路。首先先碰到一个 setTimeout，于是会先设置一个定时，在定时结束后将传递这个函数放到任务队列里面，因此开始肯定不会输出 1 。然后是一个 Promise，里面的函数是直接执行的，因此应该直接输出 2 3 。然后，Promise 的 then 应当会放到当前 tick 的最后，但是还是在当前 tick 中。因此，应当先输出 5，然后再输出 4 。最后在到下一个 tick，就是 1 。“2 3 5 4 1”

setTimeout(function(){

console.log(1)

},0)

new Promise(function executor(resolve) {

console.log(2);

for(var i=0;i<10000;i++){

i==9999&&resolve();

}

console.log(3);

}).then(function(){

console.log(4);

})

console.log(5);

//2 3 5 4 1

面试题：

var name = "w";

(function(){

//var name; 变量声明提升到此处 undefined

if(typeof name==='undefined'){

var name='j';//声明操作会被提升到它所在执行环境的顶部，并赋值 undefined而赋值操作则会原地待命。

console.log('G'+name); //Gj

}else {

console.log('h'+name);

}

})();

面试题：

var json = {

val:10,

dbl: function () {

this.val\*=2;

}

};

json.dbl(); //db1作为方法调用，此时this为json json.val=20 (10\*2)

var dbl = json.dbl; //db1函数赋值给db1变量后，此时的db1是一个普通的函数

dbl(); //普通函数调用，this指向window ，window.val为 2 (1\*2)

json.dbl.call(window);

//call方法调用，指定this为window，window.val为4 (2\*2)

alert(window.val+json.val) //所以结果为 24

总结：

页面中所有由setTimeout定义的操作，都将放在同一个队列中依次执行。而这个队列的执行时间需要等到函数调用栈执行完毕后才会执行，也就是等待所有的可执行代码执行完毕，才会轮到setTimeout执行其内部操作，并且按照其时延时间长短顺序执行代码！

setTimeout(function(){

console.log(4);

},0);

setTimeout(function(){

console.log(5);

},0);

console.log(1);

console.log(2);

console.log(3);

//输出顺序：1，2，3，4，5

声明提前 ：

console.log(x); //function(){}

var x = 1;

console.log(x); //1

function x(){}

console.log(x); //1

相当于:

var x

function x(){}

console.log(x); //function(){}

x = 1;

console.log(x); //1

console.log(x); //1

首先需要知道函数表达式和函数声明的区别。

函数声明具有变量声明提升(即所有声明变量或声明函数都会被提升到当前函数的顶部)

var getName = function(){alert(4)}

function getName(){alert(5)}

getName(); //4

相当于：

var getName;

getName = function(){alert(4)}

function getName(){alert(5)}

getName();//4

一、什么是跨域，为什么浏览器会禁止跨域，实现跨域的几种方法

1、什么是跨域

跨域的产生来源于现代浏览器所通用的‘同源策略’，所谓同源策略，是指只有在地址的：

1. 协议名 https，http

2. 域名 http://a.study.cn http://study.cn

3. 端口名 http://study.cn:8080/json/jsonp/jsonp.html study.cn/json/jsonp/jsonp.html

均一样的情况下，才允许访问相同的cookie、localStorage或是发送Ajax请求等等。若在不同源的情况下访问，就称为跨域。

2、为什么浏览器会禁止跨域

跨域的访问会带来许多安全性的问题，比如，cookie一般用于状态控制，常用于存储登录的信息，如果允许跨域访问，那么别的网站只需要一段脚本就可以获取你的cookie，从而冒充你的身份去登录网站，造成非常大的安全问题，因此，现代浏览器均推行同源策略。

em 相对父元素，需要转换的像素值 % 父元素

rem 相对根元素 需要转的像素值 %根元素

平时如果管理项目

先期确定好全局样式，编码模式（utf-8）

编写习惯一致

标注样式编写人，页面进行标注

CSS和html分文件夹并行存放，命名统一，英文翻译为准

图片采用整合

事件委托利用事件冒泡的原理，自己所触发的事件，让父级代替执行

阻止事件冒泡：e.stopPropagation, cancelBubble=true

阻止默认形为：e.preventDefault returnvalue=false

== === 判断值是否相等，后者值和类型是否相等

link和@import：

页面加载时link同时加载，@import引用CSS用等页面加载完后再加载，

link是XHTML提供的，@import是css提供的