Js

**1、JS中的前端性能优化原则：**

（1）减少http请求次数（css sprite，background-position:百分比，数值）及请求资源的大小（js/css代码压缩）打包工具：webpack　gulp　grunt

2、尽量所使用的字体图标或者SVG图标来代替传统png图，因为字体图标或者SVG是矢量图，代码编写出来的，放大不会变形，而且渲染速度快

3、采用图片的懒加载（延迟加载）  
　　目的为了，减少页面第一次加载过程中http的请求次数  
　　具体步骤：  
　　　　1、页面开始加载时不去发送http请求，而是放置一张占位图  
　　　　2、当页面加载完时，并且图片在可视区域再去请求加载图片信息

4、 能用css做的效果，不要用js做，能用原生js做的，不用第三方插件。只是用里面的一个小功能

5、避免使用iframe：不仅不好管控样式，而且相当于在本页面又嵌套其他页面，消耗性能会更大。因为还回去加载这个嵌套页面的资源

6、减少对cookie的使用（最主要的就是减少本地cookie存储内容的大小），因为客户端操作cookie的时候，这些信息总是在客户端和服务端传递。如果上设置不当，每次发送一个请求将会携带cookie

7、前端与后端进行数据交互时，对于多项数据尽可能基于json格式来进行传送。相对于使用xml  
　　来说传输有这个优势　目的：是数据处理方便，资源偏小

8、在基于ajax的get请求进行数据交互的时候，根据需求可以让其产生缓存（不是304），这样在下一次从相同地址获取是数据时，取得就是上一次缓存的数据。（注意：很少使用，一般都会清空。根据需求来做）

9、 用setTimeout来避免页面失去响应

10、 当需要设置的样式很多时设置className而不是直接操作style

11、 少用全局变量

12、在js中尽量减少闭包的使用，原因：使用闭包后，闭包所在的上下文不会被释放

13、减少对DOM操作，主要是减少DOM的重绘与回流（重排）

14、把css放在body上，把js放在body下面，让其先加载css（注意：这里关于优化没有多大关系）

15、减少css表达式的使用，拖慢网页的加载速度带来性能问题

16、在js封装过程中，尽量做到低耦合高内聚。减少页面的冗余代码

17、基于script标签下载js文件时，可以使用defer或者async来异步加载

18、在事件绑定中，尽可能使用事件委托，减少循环给DOM元素绑定事件处理函数。

**附：**

**1、Js放<body>js+html</body>与<body>html+js</body>区别**?

js+<body>html</body>:js是阻塞的，也就是说，<script>对应的js必须加载完+执行完，才会继续加载下面的html的部分,如果你的js很大、很慢，那么放在上面，会导致页面会长时间空白

<body>html</body>js:唯一的区别就是放在</body>之后不标准，现在可以，但不定哪天官方突然抽风就不好使了,因为按照标准来说</body>之后不应该出现任何东西

**2、 HTML5 有哪些不同类型的存储？**

之前Cookie 实现，localStorage：适用于长期存储数据，浏览器关闭后数据不丢失, 数据只能由用户删除；sessionStorage：存储的数据在浏览器关闭后自动删除。

**3、常见的 http 状态码以及代表的意义**

200：请求已成功，数据返回。\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303：告知客户端使用另一个 URL 来获取资源。303的使用场景几乎没有,参考302\400：请求格式错误。1)语义有误2)请求参数有误。\403：服务器拒绝请求\404：页面无法找到\500：服务器内部错误\502：服务器网关错误

304：服务器资源未变化

**4、简要描述你对 AJAX 的理解**

AJAX即异步的 JavaScript 和XML。它是指一种创建交互式网页应用的网页开发技术，可以实现页面的异步请求和局部刷新。

**5、请介绍一下 XMLHttprequest 对象**

AJAX 的核心是 JavaScript 对象 XmlHttpRequest。该对象在 Internet Explorer 5 中

首次引入，它是一种支持异步请求的技术。简而言之，XmlHttpRequest 可以使用 JavaScript

向服务器提出请求并处理响应，而不阻塞用户。实现页面的局部更新。readyState 属性：请求的状态，有 5 个可取值（0=未初始化，1=正在加载，2=以加载，3=交互中，4=完成）

**6、AJAX 都有哪些优点和缺点**

优点：

页面局部刷新，提高用户体验度；

使用异步方式与服务器通信，具有更加迅速的响应能力；

减轻服务器负担；

基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。

缺点：不支持浏览器 back 按钮；安全问题；对搜索引擎的支持比较弱。

AJAX干掉了Back和History功能，即对浏览器机制的破坏。无法回到前一个页面状态

Ajax就如同对企业数据建立了一个直接通道。使开发者不经意间暴露比以前更多的数据和服务器逻辑

对搜索引擎的支持比较弱。如果使用不当，AJAX会增大网络数据的流量，从而降低整个系统的性能。

**7、介绍一下 XMLHttpRequest 对象的常用方法和属性**

参考答案：open(“method”,”URL”) ：建立对服务器的调用，第一个参数是 HTTP，请求方式（可以为 GET，POST 或任何服务器所支持的您想调用的方式）， 第二个参数是请求页面的 URL；send()方法：发送具体请求；abort()方法：停止当前请求；readyState 属性：请求的状态，有 5 个可取值（0=未初始化，1=正在加载，2=以加载，3=交互中，4=完成）；responseText 属性：服务器的响应，表示为一个串；

reponseXML 属性：服务器的响应，表示为 XML；status 属性：服务器的 HTTP 状态码。

**8、简要描述 JavaScript 的数据类型？**

JavaScript 的数据类型可以分为原始类型和对象类型。

原始类型包括 string、number 和 boolean 特殊原始值：null（空）和 undefined（未定义），

其中，字符串是使用一对单引号

或者一对双引号括起来的任意文本；而数值类型都采用 64 位浮点格式存储，不区分整数和

小数；布尔（逻辑）只能有两个值：true 或 false。

Object对象类型如 Function,Array、Date、Object 等。

**9、计算结果**

//1将对象赋值给变量指向同一个数组

var a = []; var b = a; b[0] = 1; console.log(a[0]);

**10、简要描述 null 和 undefined 的区别**

null：是JavaScript 的关键字，表示“空值”typeof返回“object”即为一个特殊的对象值，可以表示数字、字符串和对象是“无值”的。 null这不是类型，只能说是一个关键字，表示空指针

undefined：是预定义的全局变量，其值为“未定义”，它是变量的一种取值，表示变量没有初始化。

当查询对象属性、数组元素的值时，如果返回 undefined 则表示属性或者元素不存在；

如果函数没有任何返回值，也返回 undefined。

需要注意的是，虽然 null 和 undfined 是不同的，但是因为都表示“值的空缺”，两

者可以互换。因此＝＝为true,＝＝＝为false

**11、解释一下 JavaScript 中的局部变量与全局变量的区别**

全局变量拥有全局作用域，在 JavaScript 代码的任何地方都可以访问；在函数内声明

的变量只在函数体内有定义，即为局部变量，其作用域是局部性的。

需要注意的是，在函数体内声明局部变量时，如果不使用 var 关键字，则将声明全局

变量。前提函数得调用，声明提前赋值不提前

**12、什么是 JavaScript 中的函数作用域**

变量在声明它的函数体以及这个函数体嵌套的任意位置都是有定义。在函数体内声明的所有变量都是可见的，这种特性也被称为“声明提前”，即，函数内声明的所有变量（不涉及到赋值）都被提

前至函数的顶部声明。

function test() {

console.log(x);

var x = 10;

console.log(x);

}

test();//将先输出 undefined，再输出 10。X声明提前赋值不提前

i 在 for 循环中声明，在整个函数体内都有效（函数作用域）

function test() {var sum = 0;

for (var i = 0; i < 10; i++) {s);//45 10。um += i;}console.log(sum);console.log(i);}test(

**13、简述 arguments 对象的作用**

arguments 可以访问函数的参数。表示函数的参数数组。arguments.length参数个数，其次，可以通过下标（arguments[index]）来访问某个参数。

**14、简要描述 JavaScript 中定义函数的几种方式**

JavaScript 中，有三种定义函数的方式：

1、函数语句：即使用 function 关键字显式定义函数。如：

function f(x){return x+1;}

2、函数定义表达式：也称为“函数直接量”。形如：

var f = function(x){return x+1;};

3、使用 Function() 构造函数定义，形如：

Var f = new Function(“x”,”return x+1;”);

运行结果：

function f(){console.log("function")}

function test() {

  console.log(f)  //f() {console.log("function")}

  f()       //function

f = "hello"

  console.log(f)  //hello

  f()       //f is not a function

}test();

**15、 列举几个 JavaScript 中常用的全局函数，并描述其作用**

1. parseInt：解析一个字符串并返回一个整数；

2. parseFloat：解析一个字符串并返回一个浮点数；

3. isNaN：检查某个值是否是数字,表示 不是一个数字，返回 true 或者 false不是一个数字如a=’a’isNaN(a)true

4. encodeURI ：把字符串作为 URI 进行编码；

5. decodeURI ：对 encodeURI() 函数编码过的 URI 进行解码；

6. eval：计算某个字符串，以得到结果，或者用于执行其中的 JavaScript 代码。

instanceof来检测某个对象是不是另一个对象的实例。

instanceof **运算符**用于检测构造函数的 prototype 属性是否出现在某个实例对象的原型链上。

function Car(make, model, year) {

    this.make = make;

    this.model = model;

    this.year = year;

}

const auto = new Car('Honda', 'Accord', 1998);

console.log(auto instanceof Car);//true

console.log(auto instanceof Object);//true

语法: object instanceof constructor

object: 某个实例对象

constructor: 某个构造函数

for...in 语句来遍历数组内的元素。

**17、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，删除几个，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() 方法可返回一个指定的字符串值最后出现的位置,如果指定第二个参数 start,则在一个字符串中的指定位置从后向前搜索。”

var arr = ["a", "c", "b", "c", "d"];

x.innerHTML = arr.lastIndexOf('c')//3

 x.innerHTML = arr.indexOf('c')//1

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()方法就是将一个类数组对象或者可遍历对象转换成一个真正的数组

let arrayLike = {0: 'tom',1: '65', 2: '男',3: ['jane', 'john', 'Mary'],

'length': 4}let arr = Array.from(arrayLike) console.log(arr) // ['tom','65','男',['jane','john','Mary']]

所谓类数组对象，最基本的要求就是具有length属性的对象。

addEventListener :true - 事件句柄在捕获阶段执行

map返回一个新数组主要是对数组每个元素的操作，有return，

var array1 = [1, 4, 9, 16]; const map1 = array1.map(x => x \* 2);

console.log(map1);// [2, 8, 18, 32

for/in主要是对象键值的一些遍历，对象,

 不要使用 for/in 语句来循环数组的索引，你可以使用 [for](https://www.runoob.com/jsref/jsref-for.html) 语句替代。

forEach的应用只要是数组的简单遍历

var arr = [1, 2, 3, 4, 5];

arr.forEach(function (currentValue, index, arr) {//当前，索引，所有值

        console.log(currentValue)//1 2 3 4 5

 })

for..of遍历数组，for of无法循环遍历对象

var arr = ['nick', 'freddy', 'mike', 'james'];

for (var key in arr) { console.log(key);//0123}

for (var item of arr) {console.log(item);//nick freddy mike james}

**18、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">

1. 用随机数或时间戳方法

在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();

开发中：Network-Disable cache

<script>document.write('<script src="index.js?t=' + new Date().getTime() + '"><\/script>')</script>

**19、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

**20、闭包的用途**

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

闭包：外部可以读取函数内部的变量，可以让变量的值始终保持在内存中。

缺点：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。

闭包使用场景:子函数可以使用父函数的变量

（1）采用函数引用方式的setTimeout调用。?例子

（2）将函数关联到对象的实例方法。

（3）封装相关的功能集。

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

或

function f1() {

      var n = 999;

      function f2() { console.log(n);}

      return f2();

}f1();//999

闭包使用场景:子函数可以使用父函数的变量

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

**21、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.

**21、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。

22、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

**23、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

-D 添加开发依赖 在开发周期有用 ，发布没用 ——devDependencies

-S 添加 生产依赖 在生产环境下发布以后还需要的依赖

**24**、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。

使用webpack过程：

1. 生成Packjson.json 执行命令 npm init –y；

文件作用：是node的项目描述文件，如项目依赖谁有哪些scripts 常用

1. 手动创建webpack.config.js

css用的 loader  css-loader style-loader

img用的 loader(file-loader  或 url-loader)

less用的loader less-loader

vue用的loader vue-loader webpack webpack-cli

vue-style-loader vue-html-loader vue-template-compiler

vue-loader：编译vue文件必备的

webpack webpack-cli ：编译过程中需要用到，

vue-style-loader编译vue里的样式，是style-loader的子级，有style-loader功能也有自己功能

vue-html-loader ：编译vue里html代码，编译xxx.vue文件里template中div开始的东西。

vue-template-compiler：编译vue模板，

基本配置：

module.exports = {

            mode: 'development',//开发模式

            entry: [],  //入口文件

            output: {},    //出口文件

            module: {//模块规则

                rules: [

              {test: /\.css$/i, use: ['style-loader', 'css-loader'] },

                ]

            }

}

文件作用:是node的一个模块要对外输出json,所有的webpack配置都在这

1、webpack与vue-cli区别:webpack是自己配置。vue-cli脚手架 构建项目不需要自己配置

**25、typeof和instanceof区别**

typeof: 返回值是一个字符串， 用来说明变量的数据类型。 一般只能返回如下几个结果： number, boolean, string, function, object, undefined。

instanceof: 返回值为布尔值;用于判断一个变量是否属于某个对象的实例。

**26、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；

**27、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

**29、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

**30、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。

“一个汉字字符是2个字节。1kb=1024字节， 2k就1024个字

各种浏览器和服务器的最大处理能力如下：

IE：对URL的最大限制为2083个字符，若超出这个数字，提交按钮没有任何反应。

**31、请描述一个网页从开始请求到最终显示的完整过程？**

1. 在浏览器中输入网址；

2. 发送至 DNS 服务器并获得域名对应的 WEB 服务器的 IP 地址；

3. 与 WEB 服务器建立 TCP 连接；

4. 浏览器向 WEB 服务器的 IP 地址发送相应的 HTTP 请求；

5. WEB 服务器响应请求并返回指定 URL 的数据，或错误信息，如果设定重定向，则重

定向到新的 URL 地址。

6. 浏览器下载数据后解析 HTML 源文件，解析的过程中实现对页面的排版，解析完成

后在浏览器中显示基础页面。

7. 分析页面中的超链接并显示在当前页面，重复以上过程直至无超链接需要发送，完

成全部显示。

**32、如何理解 html 标签语义化？**

语义化的主要目的在于，直观的认识标签(markup)和属性(attribute)的用途和作用。

可以概括为：用正确的标签做正确的事情。

html 语义化可以让页面的内容结构化，便于浏览器解析，便于搜索引擎解析，并提高

代码的可维护度和可重用性。

比如，尽可能少的使用无语义的标签 div，使用结构化标签<header>、<section>、

<footer> 。

**33、锚点的作用是什么？如何创建锚点？**

锚点是文档中某行的一个记号，类似于书签，用于链接到文档中的某个位置。当定义了

锚点后，我们可以创建直接跳至该锚点（比如页面中某个小节）的链接，这样使用者就无需

不停地滚动页面来寻找他们需要的信息了。

在使用 <a> 元素创建锚点时，需要使用 name 属性为其命名，代码如下所示：

<a name=”anchorname1”>锚点一</a><a href=”#anchorname1”>回到锚点一</a>

**34、超级链接有哪些常见的表现形式？**

<a> 元素用于创建超级链接，常见的表现形式有：

1、普通超级链接，语法为：

<a href="" target="">文本</a>

2、下载链接，即目标文档为下载资源，语法如：

<a href="DAY02.zip">下载</a>

3、电子邮件链接，用于链接到 email，语法如：

<a href="mailto:tarena@tarena.com.cn">联系我们</a>

4、空链接，用于返回页面顶部，语法如：

<a href="#">...</a>

5、链接到 JavaScript，以实现特定的代码功能，语法如：

<a href="javascript : …">JS 功能</a>

**35、简要描述行内元素和块级元素的区别。**

块级元素的前后都会自动换行，如同存在换行符一样。默认情况下，块级元素会独占一

行。例如，<p>、<hn>、<div> 都是块级元素。在显示这些元素中间的文本时，都将从

新行中开始显示，其后的内容也将在新行中显示。

行内元素可以和其他行内元素位于同一行，在浏览器中显示时不会换行。例如，<a>、

<span> 等。对于行内元素，不能设置其高度和宽度。

还有一种元素，为行内块级元素，比如 <img> 、<input> 元素等。这些元素可以和

其他行内元素位于同一行，同时可以设置其高度和宽度。

**36、link 和@import 都可以为页面引入 CSS 文件，其区别是？**

将样式定义在单独的.css 的文件里，link 和@import 都可以在 html 页面引入 css 文

件。有 link 和@import 两种方式，导入方式如下

link 方式：<link rel="stylesheet" type="text/css" href="aa.css">

@import 方式:<style type="text/css">@import "aa.css";</style>

**37、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;

**38、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

**39、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

**40、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

**41、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 。

**42、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

移动端设置

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

**43、JSONP原理**

其背后原理就是利用了script标签不受同源策略的限制，在页面中动态插入了script，script标签的src属性就是后端api接口的地址,并且以get的方式将前端回调处理函数名称告诉后端，后端在响应请求时会将回调返还，并且将数据以参数的形式传递回去。

**44、回流，重绘**

回流：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)。

重绘：当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。

**45、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。

实现跨域：

1. Jsonp: 其背后原理就是利用了script标签不受同源策略的限制，在页面中动态插入了script，script标签的src属性就是后端api接口的地址,并且以get的方式将前端回调处理函数名称告诉后端，后端在响应请求时会将回调返还，并且将数据以参数的形式传递回去。

基于script标签实现跨域

<script type="text/javascript">

    var jshow = function (data) { alert(data.s); };

    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>

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。

substr() 方法可在字符串中抽取从 *start* 下标开始的指定数目的字符。

stringObject.substr(start,length)

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

**46、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 。

**47、 简要描述 JavaScript 中的作用域链**

**参考答案：**任何一段 JavaScript 代码都对应一个作用域链，作用域链中存放一系列对象，代码中

声明的变量将作为对象的属性存放。

在 JavaScript 的顶层代码中，作用域链由一个全局对象组成；当定义一个函数时，它

保存一个作用域链，作用域链上有两个对象，一个是函数对象，一个是全局对象。

每当一个函数被调用时，会创建一个活动对象（也叫上下文对象），函数中的局部变量

将作为该对象的属性存放。

当需要使用一个变量时，将从作用域链中逐个查找对象的属性。比如：要使用变量 a，

将先查找作用域中的第一个对象是否有属性 a，如果有就使用；如果没有就查找作用域链中

下一个对象的属性，以此类推。如果作用域链上没有任何一个对象含有属性 x，则认为这段

代码的作用域链上不存在 x，将抛出引用错误异常。

当函数调用完成后，如果没有其他引用指向为此次调用所创建的上下文对象，该对象将

被回收。

**48、JavaScript 中，this 关键字的作用是什么？**

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，如果嵌套函数作为方法被调用，其 this 指向调用它的对象；如果作为函数调用，this 是全局对象或者为 undefined（严格模式下）。

var o = {a: 1,

      m: function () {console.log(this);

            f();//Window  作这函数调用

            function f() {

               console.log(this);

            }

       }

  };

 o.m();//{a: 1, m: ƒ}作为方法调用

**49、简要描述 JavaScript 中的自有属性和原型属性**

参考答案：

自有属性是指，通过对象的引用添加的属性，此时，其它对象可能无此属性。对于自有

属性，是各个对象所特有的、彼此独立的属性。比如：emp1.job = 'Coder';原型属性是指从原型对象中继承来的属性，一旦原型对象中属性值改变，所有继承自该

原型的对象属性均改变。比如：Emp.prototype.dept = '研发部';

当需要检测对象的自有属性时，可以使用 hasOwnProperty()方法。另，还可以使用 in

操作检测对象及其原型链中是否具备指定属性。需要注意的是，在检测对象属性时，先检测自有属性，再检测原型属性。

**50、移动端rem计算使用**

<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0" />

<meta http-equiv="cache-control" content="max-age=0" />

<meta http-equiv="cache-control" content="no-cache" />

**支持横竖屏切换改字号**

//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);

**51、http和https区别**

http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

**52、使用JavaScript深度克隆一个对象。(百度)**

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）。

深度拷贝：数组与对象通用

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;

}

**53、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

二、程序：

1）var和function同名并不冲突，在部分浏览器中，var会遮蔽同名function，导致function失效”，只要test被赋值就会not a function ，只var test;显示2

var test = 1;

function test(

console.log(index);

    index = 3;

}

test(2)//结果：test is not a function

2）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; // search 取"?"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

// url?a=1&b=2

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;

 }// console.log(getQueryString('a'))

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

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]

new Set方法

let arr = [1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 2, 3, 4, 4];

let newArr = [...new Set(arr)];

console.log(newArr); //[1, 2, 3, 4, 5, 6, 7]

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

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)

   }

}

Console.log(i)//3

</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)

   }

Console.log(i)//012

}

Console.log(i)//i is not defined

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

**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

\*/

**5、字符串模板**

`${变量}` ``反单引号 代替字符串拼接，可以换行

**6、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

**7、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，

8、**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

1）class ：类声明 2）constructor：构造函数/构造器3）extends：继承 4）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();s

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

**9、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，

10、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

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，

**10、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();

Object.assign(target,source)**;**

一个或多个源对象分配到目标对象，第一个值是目标对象，最后合并值都放target里

let json = Object.assign({}, defaults, options); //{}是target，后两个值合并后还是以前值

const target = { a: 1, b: 2 };

const source = { b: 4, c: 5 };

const result = Object.assign(target, source);

console.log(target, result)//都是 {a: 1, b: 4, c: 5}

console.log(source)// { b: 4, c: 5 };

-------

Object.is(null, null);       // true

Object.is([], []);           // false

Object.is(0, -0);            // false

Object.is('foo', 'foo');     // true

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

**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**

**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**

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 ——监听数据的修改-没return**

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

1. **组件**

全局组件——任何地方：Vue.component('xxx',{ data(){ return {}; }, })

局部组件——父组件之内components:{}

类声明组件——var cmp = Vue.extend({})挂局部上/全局上Vue.component(' cmp ', cmp)

动态组件——is

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>

*7、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;

                    }

                }

            }

        },

8、: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':''"

(6)多个动态class :class="`${'left'+key} ${key<signinDays?'active':''}`"

signinDays:[0,1,2,3,4,5,6] key是索引值left0 active

9、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)"

**10、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

**11、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 }

**12、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](http://localhost:3002/" \l "/cmp1)

——像描点，默认模式如： [http://localhost:8080/#/news](http://localhost:8080/" \l "/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

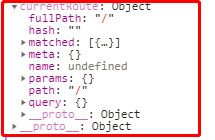
这里的找不到页面指路由配置中没引入cmp3没加：{path:'/cmp3',component:cmp3}

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**

**1、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

**2、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](http://localhost:3002/" \l "/index) 二层：[http://localhost:3002/#/aaa](http://localhost:3002/" \l "/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](http://localhost:3002/" \l "/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](http://localhost:3001/" \l "/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>

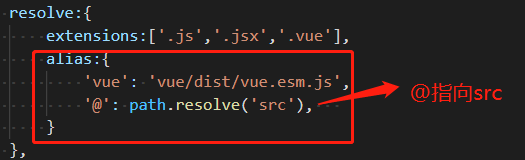
注：path中路径要与to中path一致如：

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**

**13.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。

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

**13、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

**14、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

**15、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）

**16. 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。

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

HTML与css

1、英文单词不发生词内换行

word-break: break-word;

2、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；

3、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;

4、css3 三角箭头

<http://www.jb51.net/css/41448.html>

.up{width:0;height:0;border-left:50px solid transparent;border-right:50px solid transparent;border-bottom:100px solid red}

.down{width:0;height:0;border-left:50px solid transparent;border-right:50px solid transparent;border-top:100px solid red}

.right{width:0;height:0;border-top:50px solid transparent;border-left:100px solid red;border-bottom:50px solid transparent}箭头向右

.left{width:0;height:0;border-top:50px solid transparent;border-right:100px solid red;border-bottom:50px solid transparent}箭头向左

5、em/px/rem区别？

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

如父24px, 子30px等于1.25em = 1/24\*30

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

Rem：相对于根元素<html>默认16px

需要转换值÷16=rem值

px稳定和精确。问题就是缩放页面时布局会打破

适配各种移动设备，使用rem

***注意：****任意浏览器的默认字体高都是16px。所有未经调整的浏览器都符合: 1em=16px。body选择器中声明Font-size=62.5%相于于10px，这样12px=1.2em, 10px=1em, 也就是说只需要将你的原来的px数值除以10，然后换上em作为单位就行了。*

**6.垂直居中的方法？**

（1）用 position和负边距：父节点相对定位。子节position:absolute;top:50%;margin-身高度的一半;（2）多行文本居中： vertical-align: middle;display: table-cell;（3）文本居中line-height

**Display:flex弹性布局”, 用来为盒子模型提供最大的灵活性;**

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;

margin各减去上下距离的一半

方案2：position transform 元素未知宽度

子元素：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

**7、哪些属性可以继承？**

CSS 中可以继承的属性如下：

1）文本相关属性：font-family、 font-size、 font-style、font-variant, font-weight、

font、 letter-spacing、line-height、text-align、 text-indent、text-transform、

word-spacing、color;

**8、display:none 和visibility:hidden 区别inline inline-block block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；

block元素会独占一行，width,height边距都可控制　<div>, <p>, <h1>, <form>, <ul> 和 <li>是块元素的例子。

inline行内元素不独占一行，内联对象会被排列在同一行内、排列不下新换一行。width,height属性无效span>, <a>, <label>, <input>, <img>, <strong> 和<em>是inline元素的例子。

inline-block将对象呈递为内联对象，但是对象的内容作为块对象呈递。旁边的内联对象会被呈递在同一行内，允许空格。

**9、如何清除浮动元素所带来的影响？**Clear:both,

10、谈谈你对浏览器兼容性问题的理解

background-color: #f1ee18; /\*所有识别\*/

.background-color: #00deff\9; /\*IE6、7、8 识别\*/

+background-color: #a200ff;/\*IE6、7 识别\*/

 \_background-color: #1e0bd1; /\*IE6 识别\*/

属性分为固有属性property和自定义属性attribute

**10、 简述你对 JavaScript 中原型的理解**

参考答案：

在 JavaScript 中，函数本身就是一个包含了方法和属性的对象。每个函数中都有一个

prototype 属性，该属性所存储的就是原型对象。

原型对象用来保存共享属性和方法，可以通过原型来实现为对象扩展属性，实现继承。

**11、 简述 JavaScript 中创建自定义对象的方式**

参考答案：

自定义对象（user-defined object）指由用户创建的对象，兼容性问题需要由编写者

注意。创建自定义对象的方式有：

1、对象直接量

2、new Object()

3、function 对象模板

4、Object.create()

**12、简要描述 JavaScript 中的作用域链**

参考答案：

任何一段 JavaScript 代码都对应一个作用域链，作用域链中存放一系列对象，代码中

声明的变量将作为对象的属性存放。

在 JavaScript 的顶层代码中，作用域链由一个全局对象组成；当定义一个函数时，它

保存一个作用域链，作用域链上有两个对象，一个是函数对象，一个是全局对象。

每当一个函数被调用时，会创建一个活动对象（也叫上下文对象），函数中的局部变量

将作为该对象的属性存放。

当需要使用一个变量时，将从作用域链中逐个查找对象的属性。比如：要使用变量 a，

将先查找作用域中的第一个对象是否有属性 a，如果有就使用；如果没有就查找作用域链中

下一个对象的属性，以此类推。如果作用域链上没有任何一个对象含有属性 x，则认为这段

代码的作用域链上不存在 x，将抛出引用错误异常。

当函数调用完成后，如果没有其他引用指向为此次调用所创建的上下文对象，该对象将

被回收。

**13、简要描述 JavaScript 中定义函数的几种方式**

参考答案：

JavaScript 中，有三种定义函数的方式：

1、函数语句：即使用 function 关键字显式定义函数。如：

function f(x){

return x+1;

}

2、函数定义表达式：也称为“函数直接量”。形如：

var f = function(x){return x+1;};

3、使用 Function() 构造函数定义，形如：

Var f = new Function(“x”,”return x+1;”);

**14、 什么是 JavaScript 中的函数作用域**

参考答案：

JavaScript 中的函数作用域是指：变量在声明它的函数体以及这个函数体嵌套的任意

函数体内都是有定义的。这意味着，在函数体内声明的所有变量在整个函数体内始终是可见

的，这种特性也被称为“声明提前”，即，函数内声明的所有变量（不涉及到赋值）都被提

前至函数的顶部声明。