20190213

grid布局 二维网格布局

20190214

移动端布局

rem 百分比

lib-flexible px2rem-loader

overflow:hidden的作用

calc()

\*\*\*运算符前后都需要保留一个空格，例如：width：calc(100% - 10px);否则属性会报错！

ios系统input blur事件，页面会被卡住的问题：

document.getElementsByTagName("body")[0].scrollTop += 1;

如何判断一个变量是Array类型？如何判断一个变量是Number类型？（都不止一种）

var ary = [1,2,3,4];

console.log(ary **instanceof** Array); //输出true

console.log(ary.**constructor**==Array) //输出true

**完美方法**

console.log(Object.prototype.toString.call(arr)) //输出"[object Array]"

??

obj.toString()的结果和Object.prototype.toString.call(obj)的结果不一样，这是为什么？

因为toString为Object的原型方法，而Array ，function等类型作为Object的实例，都重写了toString方法。需要使用call或者apply方法来改变toString方法的执行上下文。

0215

web前端图片懒加载实现原理

<https://blog.csdn.net/qq_18851347/article/details/73223566>

* 函数防抖原理：每次执行\_delay函数先清除上一次setTimeout生成的引用，阻止上一次的函数调用（如果还没执行的话），然后创建一个新的setTimeout，在time保存的时间间隔后调用函数

vue项目图片懒加载VueLazyload的使用 demo

【前端性能】高性能滚动 scroll 及页面渲染优化

<http://www.cnblogs.com/coco1s/p/5499469.html>

从输入url到获取页面的完整过程

http请求报文：请求行（方法 url 协议版本），请求头，请求主体

HTTP响应：状态行（协议版本 状态码）、消息报头、响应正文。

为什么要减少HTTP请求？

http请求过程和原理:

1.用户在浏览器中键入需要访问网页的URL或者点击某个网页中链接；

2.浏览器根据URL中的域名，通过DNS解析出目标网页的IP地址；

3.浏览器与网页所在服务器建立TCP连接；

4.浏览器发送HTTP请求报文，获取目标网页的文件；

5.服务器发送HTTP响应报文，将目标网页文件发送给浏览器；

6.释放TCP连接；

7.浏览器将网页的内容包括文本、图像、声音等显示呈现在用户计算机屏幕。

什么是MVVM，MVC

节前0128-0130

本周完成：

1.齿科微保渠道预约页面弹框bug问题方案查询

2. vue项目图片懒加载VueLazyload的使用 demo

下周计划：

图片预加载和懒加载原理及使用

0213-0215

本周：

1.判断一个变量是Array类型方法总结

2.学习了解 grid布局 ，二维网格布局

下周计划：

高性能滚动 scroll 及页面渲染优化

**20190218**

图片预加载

（齿科评价列表图片用的就是这种方法 new Imae() img.onload预载完成事件）

用法：

1. 默认情况下如果网站请求速度慢，所以会有一段时间的空白页面等等，用户体验效果不好，见到很多的页面都有预加载的效果，加载之前先加载一个动画，后台进程继续加载页面内容，当页面内容加载完之后再退出动画显示内容，这样的用户体验更好。

单纯使用CSS，未实现居中

强缓存和协商缓存

强缓存：[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)通过设置http中hdader的Expires和cache-control字段告诉浏览器换存的有效期。

协商缓存：通过服务器来判断缓存是否可用。

**sessionStorage 、localStorage 和 cookie 之间的区别**

共同点：都是保存在浏览器端，且同源的。

区别：

cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递。而sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下。

存储大小限制也不同，cookie数据不能超过4k，同时因为每次http请求都会携带cookie，所以cookie只适合保存很小的数据，如会话标识。sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。

数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。

作用域不同，sessionStorage不在不同的浏览器窗口中共享，即使是同一个页面；localStorage 在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的。

Web Storage 支持事件通知机制，可以将数据更新的通知发送给监听者。

Web Storage 的 api 接口使用更方便。

Cache-Control是关于浏览器缓存的最重要的设置，因为它覆盖其他设置，比如 Expires 和 Last-Modified

0219

前端性能优化：DOM操作性能优化建议

**网页的生成过程**，大致可以分成五步:

HTML代码转化成DOM

CSS代码转化成CSSOM（CSS Object Model）

结合DOM和CSSOM，生成一棵渲染树（包含每个节点的视觉信息）

生成布局（layout），即将所有渲染树的所有节点进行平面合成

将布局绘制（paint）在屏幕上

这五步里面，第一步到第三步都非常快，耗时的是第四步和第五步。

"生成布局"（flow）和"绘制"（paint）这两步，合称为"渲染"（render）。

1.查找元素的优化：尽可能的通过ID或者类来查找元素，避免通过类来查找元素。

2.改变DOM，包括添加，修改，删除DOM：避免不必要的渲染

3. 不要一条条地改变样式，而要通过改变class，一次性地改变样式。

4. 尽量使用离线DOM(Document Fragment)，而不是真实的网面DOM，来改变元素样式。

Vue 单页应用 的 首屏优化

1、压缩代码

对于vue-cli生成的项目，在Webpack配置文件中使用了UglifyJsPlugin进行代码的压缩

2、框架和插件按需加载

3、路由懒加载

路由懒加载也就是 把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件。

JQuery\_九大选择器

1.基础选择器：#id element class

2.层级选择器：子元素 子孙元素

3.过滤选择器

3.1基础过滤：$("tr:first") :not(selector) :even :odd :gt(index)

3.2内容过滤：:contains(text) :empty :has(selector)

3.3可见度过滤：:hidden :visible

3.4属性过滤：[attribute]

3.5子元素过滤：$("ul li:nth-child(2)")

3.6表单对象属性过滤： :enabled :disable :checked :selected

4.表单选择器：

:input :text :password :radio :checkbox :submit :image :reset :button :file :hidden

**组件之间的传值**

0222

# Js基础

## js数组去重方法：

var arr = [1,23,1,1,1,3,23,5,6,7,9,9,8,5];

1.利用空对象记录新数组中已经存储过的元素

var o={};

var new\_arr=[];

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

var k=arr[i];

if(!o[k]){

o[k]=true;

new\_arr.push(k);

}

}

console.log(new\_arr);

2.数组filter方法

var d=arr.filter(function(element,index,self){

return self.indexOf(element)===index

})

3. 元素首次在数组出现的下标是否与循环下标相等

3.1不相等 删除

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

if (arr.indexOf(arr[i]) != i) {

arr.splice(i,1);//删除数组元素后数组长度减1后面的元素前移

i--;//数组下标回退

}

}

3.2相等 push进新的数组

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

if (arr.indexOf(arr[i]) ==== i) {

newArr.push(arr[i])

}

}

4.new Set()

[...new Set(arr)] Array.from(new Set(arr))

5.当前项与后面每一项比较（留后面的一项）

var newArr3=[];

for(let i=0;i<ary.length;i++){

var item=ary[i];other=ary.slice(i+1);

if(other.indexOf(item)===-1){newArr3.push(item)}

}

for(let i=0;i<ary.length;i++){

var item=ary[i];other=ary.slice(i+1);

if(other.indexOf(item)>-1){ary.splice(i,1);i--}

}

## 数组排序：

**Vm sort.js**

冒泡排序：

function bubble(ary){

let temp=null;

for(let i=0;i<ary.length-1;i++){

//不用跟自己比，减去轮数（放在最末尾的最大数）

for(let j=0;j<ary.length-1-i;j++){

if(ary[j]>ary[j+1]){

temp=ary[j];

ary[j]=ary[j+1];

ary[j+1]=temp

}

}

}

}

插入排序：

快速排序：

## js数组方法：

arr.concact(arr1,arr2,…) arr.slice(start,end)

arr.splice(i,num…) arr.reverse() arr.push() arr.pop() arr.unshift() arr.shift() arr.sort()

arr.join() arr.toString() arr.indexOf(element,start)

ES6

Array.isArray(arr)

forEach() map every some any filter reduce find

## string字符串方法：

var str="Visit Microsoft!"

str.indexOf(e,fromindex) str.lastIndexOf() str.replace(/Microsoft/, "W3School")

str.slice() str.substring()

Str.toUpperCase() str.toLowerCase()

str.substr(start,length) str.split(‘’,num) str.charAt(index) str.charCodeAt(i)

str.search(/Microsoft /i)返回index

ES6

includes() startsWith endsWith repeat padStart padEnd trimStart trimEnd

## date方法：

var myDate=new Date()

myDate.getDate() myDate.getDay() myDate.getMonth() myDate.getFullYear() myDate.getHours() myDate.getMinutes() myDate.getSeconds() myDate.getTime()

## Math方法：

Math.round Math.ceil Math.floor Math.abs Math.random() Math.max() Math.min()

获取[n,m]之间的随机整数

Math.round(Math.random()\*(m-n)+n)

生成验证码最简方式

Math.radom().toString(32)

数组的最大值

Let arr = [1,35,78]

Math.max.apply(this,arr)

# BOM

Navigator.userAgent

# DOM

js对象深拷贝、浅拷贝

<http://www.ruanyifeng.com/blog/2010/05/object-oriented_javascript_inheritance_continued.html>

0225

1.vue生命周期如何阐述比较好

new Vue() init() 调用beforeCreate

inject 调用created 能拿到数据比如this.a，才能进行操作

去查找实例中是否有el option，接着判断是否含有template选项，会把template解析成一个render function。render函数是发生在beforeMount和mounted之间的，这也从侧面说明了，在beforeMount的时候，$el还只是我们在HTML里面写的节点，然后到mounted的时候，它就把渲染出来的内容挂载到了DOM节点上。

在使用.vue文件开发的过程当中，我们在里面写了template模板，在经过了vue-loader的处理之后，就变成了render function，最终放到了vue-loader解析过的文件里面。这样做有什么好处呢？原因是由于在解析template变成render function的过程，是一个非常耗时的过程，vue-loader帮我们处理了这些内容之后，当我们在页面上执行vue代码的时候，效率会变得更高

在mounted挂载完毕之后，这个实例就算是走完流程了。

后续的钩子函数执行的过程都是需要外部的触发才会执行。比如说有数据的变化，会调用beforeUpdate，然后经过Virtual DOM，最后updated更新完毕。当组件被销毁的时候，它会调用beforeDestory，以及destoryed。

**websocket协议**

websocket是允许web应用程序与服务器进行双向通信的协议，HTML5的websocket API主要为浏览器提供基于TCP协议实现全双工通信方法。

优点：

全双工通信 实时性 更少的数据传输和更少的请求次数

没有同源限制，客户端可以与任意服务器通信。

传统web应用中，浏览器和服务器之间是半双工通信的，同一时间数据流是单向的。websocket服务器可以主动向客户端推送信息，客户端也可以主动向服务器发送信息，是真正的双向平等对话，属于服务器推送技术的一种。

实例对象的 onmessage 属性，用于指定收到服务器数据后的回调函数。

心跳检测，就是为了检测客户端与服务端的连接是否还在。

实现心跳检测的方法思路，主要是通过定时向服务器send()消息，并且定下心跳包的超时时间，当收到服务器返回的消息时，清掉当前心跳计时器以及重连超时的定时器。若服务端未能够及时返回特定消息，超过设定的超时时间时，主动关闭当前websocket连接，并且尝试重新建立新的websocket连接。

小程序白屏

比较查询差异文件的作用：

package-lock.json是当 node\_modules 或 package.json 发生变化时自动生成的文件。这个文件主要功能是确定当前安装的包的依赖，以便后续重新安装的时候生成相同的依赖，而忽略项目开发过程中有些依赖已经发生的更新。

并未引用的js语法错误？？

**前端性能优化总结**

加载优化

1.减少http请求（合并css,js,多个icon使用精灵图）

2.缓存

3.压缩文件和图片

4.无阻塞（js放页面底部，避免使用style内部样式）

5.按需加载

6.img避免空的src（在img 对象的src 属性是空字符串(“”)的时候，浏览器认为这是一个缺省值，值的内容为当前网页的路径。浏览器会用当前路径进行再一次载入，并把其内容作为图像的二进制内容并试图显示。）

7.尽量避免重设图片大小，避免重绘

20190320

# **webpack**

js应用程序的静态模块打包器。当webpack处理程序时，会建立一个依赖关系图，包含程序需要的所有模块，然后将模块打包成一个或多个bundle。

四个核心概念：

入口 输出 loader 插件

入口

入口起点指示webpack应该使用哪个模块作为构建依赖图的开始

在build文件夹中的webpack.base.con.js配置。

loader

让webpack(自身只能理解js)能够处理非js文件。

html-webpack-plugin 打包html文件

extract-text-webpack-plugin

CommonJs ESMoudle区别

1. CommonJs模块输出的是值的拷贝，ES6输出的是值的引用，输出模块的内部改变会影响引用的改变
2. CommonJs运行时加载，ES6是编译时输出接口
3. CommonJs加载的是整个模块，将所有接口全部加载进来，ES6可以单独加载其中某个接口

AMD是CommonJs规范的草案，异步模块加载机制，只要实现RequireJS

CommonJs目的是弥补js在服务器端缺少模块化机制，nodejs,webpack都是基于该规范实现：

模块是同步加载的

模块在首次执行后会缓存，再次加载返回缓存结果

## Webpack 优化

压缩代码

压缩图片资源

组件懒加载

Tree-shaking删除无用代码（require不支持，要使用import）

require（CommandJS规范 ，在nodejs中使用）,运行时调用，可以随处引入

import（es6语法）,import是编译时调用，必须放在文件开头引入

Scope hosting 作用域提升--合并一些代码

Webpack 打包测试、生产域名自动匹配.env .env.production

process.env包含所有环境变量

1.配置文件有：

.env 全局默认配置文件，不论什么环境都会加载合并  
.env.development 开发环境下的配置文件  
.env.production 生产环境下的配置文件

2.命名规则：

属性名必须以VUE\_APP\_开头，比如VUE\_APP\_XXX

Nodejs path模块

<https://www.jianshu.com/p/d683509f9dcd>

**自带优化功能：**

# 微信JS-SDK

用过的：

获取地理位置接口wx.getLocation

查看位置接口wx.openLocation

拍照或从手机相册中选图接口wx.chooseImage

上传图片接口wx.uploadImage（问题：返回选定照片的本地ID列表，localId可以作为img标签的src属性显示图片，苹果需要转换为base64数据）

分享wx.onMenuShareAppMessage

显示右上角菜单接口wx.showOptionMenu();

批量隐藏功能按钮接口

wx.ready(function(){   
  wx.hideMenuItems({})

})

vue-router

项目vue-reDiscover

微信小程序，weex框架

**20190516**

**z-index失效问题：**

<style>

div{position:abosolute; width:100px; height:100px;}

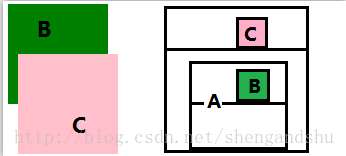
</style>

<div id="A" style="z-index:-1">

<div id="B" style="background:green; z-index:2"></div>

</div>

<div id="C" style="background:pink; left:20px; top:50px"></div>



由于z-index=-1,此时的A就像是在负1楼，即使B的z-index=2也会被C覆盖。

20190527

1扩展运算符将一个**数组**转为用逗号分隔的**参数序列**

console.log(...[a, b, c])

// a b c

2 使用扩展运算符展开数组代替apply方法，将数组转为函数的参数

//ES5 取数组最大值

console.log(Math.max.apply(this, [654, 233, 727]))；

//ES6 扩展运算符

console.log(Math.max(...[654, 233, 727]))

//相当于

console.log(Math.max(654, 233, 727))

3 合并数组

var arr1 = ['a', 'b'];

var arr2 = ['c'];

var arr3 = ['d', 'e'];

// ES5 的合并数组

arr1.concat(arr2, arr3);

// [ 'a', 'b', 'c', 'd', 'e' ]

// ES6 的合并数组

[...arr1, ...arr2, ...arr3]

5 将字符串转换为数组

[...'hello']

// [ "h", "e", "l", "l", "o" ]

//ES5

str.split('')

20190528

1.promise的链式操作及all的用法

20190530

es6

let不允许在相同作用域内，重复声明同一个变量。

为什么需要块级作用域?

内层变量可能会覆盖外层变量。

解构赋值

1.数组

let [a, b, c] = [1, 2, 3];

2.对象

let { foo, bar } = { foo: 'aaa', bar: 'bbb' };

对象的解构与数组有一个重要的不同。数组的元素是按次序排列的，变量的取值由它的位置决定；而对象的属性没有次序，变量必须与属性同名，才能取到正确的值。

3.字符串

const [a, b, c, d, e] = 'hello';

4.函数参数

function add([x, y]){

return x + y;

}

add([1, 2]); // 3

bootstrap

20190701

1.职业规划 <https://blog.csdn.net/varyall/article/details/81912301>

考察方面：自我认知 岗位任职 语言能力 职业稳定性 目标感 自我驱动

2.职业优势

逻辑思维强，功能代码完成效率高，vue框架组件库使用熟练，沟通能力较强

3.http常见状态码<https://blog.csdn.net/dufufd/article/details/53112184>

**2开头 （请求成功）表示成功处理了请求的状态代码。**

200   （成功）  服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。   
204   （无内容）  服务器成功处理了请求，但没有返回任何内容。

**4开头 （请求错误）这些状态代码表示请求可能出错，妨碍了服务器的处理。**

400 （错误请求） 服务器不理解请求的语法。

401 （未授权） 请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

403 （禁止） 服务器拒绝请求。

404 （未找到） 服务器找不到请求的网页。

408 请求超时

414 （请求的 URI 过长） 请求的 URI（通常为网址）过长，服务器无法处理。

**5开头（服务器错误）这些状态代码表示服务器在尝试处理请求时发生内部错误。 这些错误可能是服务器本身的错误，而不是请求出错。**

500   （服务器内部错误）  服务器遇到错误，无法完成请求。

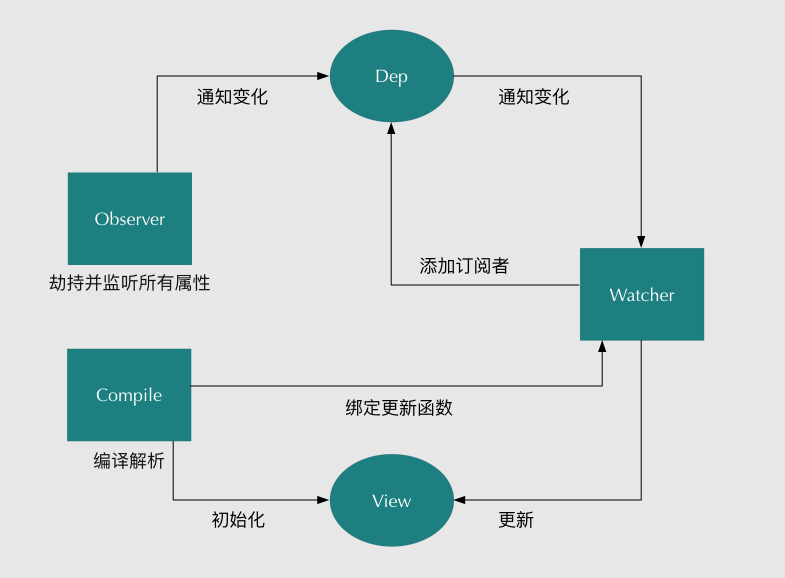
502   （错误网关） 服务器作为网关或代理，从上游服务器收到无效响应。

504   （网关超时）  服务器作为网关或代理，但是没有及时从上游服务器收到请求。

Vue的双向绑定原理及实现

深入的<https://blog.csdn.net/sinat_27346451/article/details/78315075>

28.5 855 6.5 -646



vue的数据双向绑定是通过数据劫持和发布-订阅者功能来实现的

实现步骤：1.实现一个监听者Oberver来劫持并监听所有的属性，一旦有属性发生变化就通知订阅者

2.实现一个订阅者watcher来接受属性变化的通知并执行相应的方法，从而更新视图

3.实现一个解析器compile，可以扫描和解析每个节点的相关指令，并根据初始化模板数据以及初始化相对应的订阅者

1.解析模板指令，并替换模板数据，初始化视图

2.将模板指令对应的节点绑定对应的更新函数，初始化相应的订阅器

vue相关面试题

一、对于MVVM的理解？

二、Vue的生命周期

三、vue的优点是什么？

四、组件之间的传值？

面试题

**字符串由空格字母数字组成，反转每个词的字符顺序，保持词的顺序不变，词间如果多个空格也需要保留**

var str='i am a student'

var arr=str.split(' ')

arr.forEach(function(val,i){if(val){arr[i]=val.split('').reverse().join('')}})

var newArr=arr.join(' ')

**js实现简单的双向绑定**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

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

<title>js简单双向数据绑定</title>

</head>

<body>

<div id="app">

<input type="text" id="txt"/>

<p id="show\_txt"></p>

</div>

</body>

<script>

let obj = {}

Object.defineProperty(obj,'txt',{

get:function(){

return obj

},

set:function(newValue){

document.getElementById('txt').value = newValue

document.getElementById('show\_txt').innerHTML = newValue

}

})

document.addEventListener('keyup',function(e){

obj.txt = e.target.value

})

</script>

</html>

### export 和 export default区别

1. 在一个文件或模块中，export、import可以有多个，export default仅有一个
2. 通过export方式导出，在导入时要加{ }，export default则不需要

**如何解释vue的生命周期才能令面试官满意？**

<https://juejin.im/post/5ad10800f265da23826e681e>

0730

**[Vue动态组件](https://www.cnblogs.com/shitaotao/p/7624657.html)**

<https://www.cnblogs.com/shitaotao/p/7624657.html>

Vue生命周期activated之实现返回上一页不重新请求数据

<https://blog.csdn.net/brand2014/article/details/81981548>

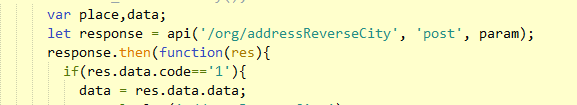
小结：适用于查看的页面，有操作的（例如状态变化）不适合

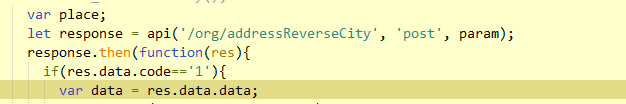
## [HTML5可预览多图片ajax上传(使用formData传递数据)](https://www.cnblogs.com/tugenhua0707/p/7599691.html)

<https://www.cnblogs.com/tugenhua0707/p/7599691.html>

FormData类型其实是在XMLHttpRequest 2级定义的，它是为序列化表以及创建与表单格式相同的数据（当然是用于XHR传输）提供便利。

0808





0903

退出forEach循环的方法：

1.抛出异常

2.跑空循环（添加if条件）

new Date(‘1990’, ‘11’, ‘19’)

1010

vuex

1.mapState 辅助函数, 帮助生成计算属性;对象展开运算符, 简化写法

2. 记住浏览过页面的位置scrollBehavior（一知半解）

1011

为什么进步太慢，因为你没有一个好习惯

vh是相对视口（viewport）的高度而定的，长度等于视口高度的1/100

1015

图片加载相关问题：

img.**complete**是Image对象的**属性**,返回一个布尔值;img.**onload**是在图片加载完成后**触发**的事件

javascript/html 禁止图片缓存

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

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

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

1031

函数柯里化：

所谓"柯里化"，就是把一个多参数的函数，转化为单参数函数。（还没找到应用场景）

js重写tostring,调用时机（？？没查到有用的资料）

面试题：

随机打乱数组

int temp=m+parseInt(Math.random()\*(n+1-m)); //生成从m到n的随机整数[m,n]

vue面试题

<https://juejin.im/post/5d59f2a451882549be53b170>

计算属性 vs 侦听属性

计算属性在大多数情况下更合适，当需要在数据变化时执行异步或开销较大的操作时，侦听器最有用。

在示例中，使用 watch 选项允许我们执行异步操作 (访问一个 API)，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

activated在组件第一次渲染时会被调用，之后在每次缓存组件被激活时调用。

activated调用时机：

第一次进入缓存路由/组件，在mounted后面，beforeRouteEnter守卫传给 next 的回调函数之前调用：

beforeMount=> 如果你是从别的路由/组件进来(组件销毁destroyed/或离开缓存deactivated)=>

mounted=> activated 进入缓存组件 => 执行 beforeRouteEnter回调

复制代码

因为组件被缓存了，再次进入缓存路由/组件时，不会触发这些钩子：

// beforeCreate created beforeMount mounted 都不会触发。

**Hash**

Hash 方法是在路由中带有一个 #，主要原理是通过监听 # 后的 URL 路径标识符的更改而触发的浏览器 hashchange 事件，然后通过获取 location.hash 得到当前的路径标识符，再进行一些路由跳转的操作

Vue 中的 key 有什么作用？

key的作用就是更新组件时判断两个节点是否相同。相同就复用，不相同就删除旧的创建新的。避免组件复用，能保证组件状态正确。

**Vue 项目性能优化**

[https://juejin.im/post/5d548b83f265da03ab42471d#heading-5](https://juejin.im/post/5d548b83f265da03ab42471d" \l "heading-5)

v-for 遍历必须为 item 添加 key，且避免同时使用 v-if

<ul>

<li

v-for="user in users"

v-if="user.isActive"

:key="user.id"

>

{{ user.name }}

</li>

</ul>

computed: {

activeUsers: function () {

return this.users.filter(function (user) {

return user.isActive

})

}

}

哪怕我们只渲染出一小部分用户的元素，也得在每次重渲染的时候遍历整个列表，修改后渲染更高效。

长列表性能优化

Object.freeze冻结对象，避免vue的数据劫持，提高性能。

Webpack 对图片进行压缩（用 image-webpack-loader来压缩图片）

减少 ES6 转为 ES5 的冗余代码：Babel 插件会在将 ES6 代码转换成 ES5 代码时会注入一些辅助函数，babel-plugin-transform-runtime 插件将相关辅助函数进行替换成导入语句，从而减小 babel 编译出来的代码的文件大小。

事件的销毁；图片资源懒加载（Vue 的 vue-lazyload 插件）；路由懒加载（() => **import**('./Foo.vue')）

setTimeOut, Promise,同步代码

同步-> Promise -> setTimeOut

掘金小册

**[Web 前端面试指南与高频考题解析](https://juejin.im/book/5a8f9ddcf265da4e9f6fb959)**

<https://juejin.im/book/5a8f9ddcf265da4e9f6fb959/section/5a8f9ddc6fb9a06340522253>

面试会被问到的

如果让你设计一个前端框架你会有什么思考

登录

Authorization（项目没用到，没搞懂）

token

单点登录（查看现有项目代码）

数组扁平化，排序，去重

Array.from(new Set(arr.flat(Infinity))).sort((a,b)=>{ return a-b})

20191021

call和aplly的第一个参数都是要改变上下文的对象，而call从第二个参数开始以参数列表的形式展现，apply则是把除了改变上下文对象的参数放在一个数组里面作为它的第二个参数。

用途：

求数组最大/最小值

伪数组（ **数据结构必须是以数字为下标而且一定要有length属性**）转化为数组

数组追加

判断数据类型

1022

我接触过的前端数据结构与算法[https://juejin.im/post/5958bac35188250d892f5c91#heading-5](https://juejin.im/post/5958bac35188250d892f5c91" \l "heading-5)

在 JavaScript 中学习数据结构与算法

<https://juejin.im/post/594dfe795188250d725a220a>

入坑出坑的过程~~~

选择排序/快速排序：手写

Array flat方法实现过程：手写

1023

es6数组新方法：

arr.filter()

1024

cors:跨域，前端不需要操作，后台添加响应头

阻止默认事件：

**return false;**

**event.preventDefault( );**

数组遍历：

1. for (初始化; 循环执行条件; 每遍历一个元素后做的事情;){}
2. forEach map every (这3个不相上下,可认为运行速度差不多)

new的实现过程，源码

<https://rockjins.js.org/2017/01/18/2017-01-18-new-object/>



简单的new 过程：伪代码表示：

var a = new myFunction("Li","Cherry");

new myFunction{

var obj = {};

obj.\_\_proto\_\_ = myFunction.prototype;

var result = myFunction.call(obj,"Li","Cherry");

return typeof result === 'obj'? result : obj;

}

创建一个空对象 obj;

将新创建的空对象的隐式原型指向其构造函数的显示原型。

使用 call 改变 this 的指向

如果无返回值或者返回一个非对象值，则将 obj 返回作为新对象；如果返回值是一个新对象的话那么直接直接返回该对象

**vscode快捷键**

全局搜索ctrl + shift +F 复制上一行Shift+Alt + ↓ / ↑

F12 转到定义文件 Ctrl+Shift+K 删除当前行

[].shift.call( arguments )，**让类数组调用数组的方法，删除数组第一项，并返回删除项**

**es6-es10**

<https://juejin.im/post/5ca2e1935188254416288eb2>

ECMAScript中所有的函数的参数都是按值传递的。

函数参数传递的并不是变量的引用，而是变量拷贝的副本，当变量是原始类型时，这个副本就是值本身，当变量是引用类型时，这个副本是指向堆内存的地址。

装箱和拆箱

装箱转换：把基本类型转换为对应的包装类型

拆箱操作：把引用类型转换为基本类型

每当我们操作一个基础类型时，后台就会自动创建一个包装类型的对象，从引用类型到基本类型的转换，也就是拆箱的过程中，会遵循ECMAScript规范规定的toPrimitive原则，一般会调用引用类型的valueOf和toString方法，你也可以直接重写toPeimitive方法。一般转换成不同类型的值遵循的原则不同，例如：

引用类型转换为Number类型，先调用valueOf，再调用toString

引用类型转换为String类型，先调用toString，再调用valueOf

**如何写出一个惊艳面试官的深拷贝?**

[https://juejin.im/post/5d6aa4f96fb9a06b112ad5b1#heading-5](https://juejin.im/post/5d6aa4f96fb9a06b112ad5b1" \l "heading-5)

JavaScript中的拷贝方法

JSON.parse(JSON.stringify());

concat ， slice，Object.assgn ，... ---🡪首层浅拷贝

20191101

**需要查的资料：更改代码上生产，需要手动清缓存，改进方案**

**1105**

**一个合格(优秀)的前端都应该阅读这些文章**

<https://juejin.im/post/5d387f696fb9a07eeb13ea60>

缓存：



在做前端缓存时，我们尽可能设置长时间的强缓存，通过文件名加hash的方式来做版本更新。在代码分包的时候，应该将一些不常变的公共库独立打包出来，使其能够更持久的缓存。

hash 计算与整个项目的构建相关；

chunkhash 计算与同一 chunk 内容相关；

contenthash 计算与文件内容本身相关。

反向代理，负载均衡

1109

README规范：<https://juejin.im/post/5aaa06e4f265da237b21cf63>

1121

图片请求不到，防盗链。可以加：

<meta name="referrer" content="never">

1122

面向对象：

封装 继承 多态

isPrototypeOf **instanceof** hasOwnProperty **in**

**1203**

# 需要背的

1. 根据手机分辨率，设置倍数的背景图片，防止模糊

@media screen and (-webkit-min-device-pixel-ratio:2){}

1. H5禁止微信浏览器调整字体大小

IOS:text-size-adjust:100%

Android:

// 设置网页字体为默认大小

**WeixinJSBridge.invoke('setFontSizeCallback', { 'fontSize' : 0 });**

// 重写设置网页字体大小的事件

WeixinJSBridge.on('menu:setfont', function() {

WeixinJSBridge.invoke('setFontSizeCallback', { 'fontSize' : 0 });

});

1. 滚动卡顿

Body{

Overflow-scrolling:touch;

}

1. ios input默认内阴影

-webkit-appearance:none;

1. 圆角bug

有些android圆角失效，background-clip:padding-box;

## 常用css组合

...

单行

overflow: hidden; text-overflow:ellipsis; white-space: nowrap;

多行

display: -webkit-box; -webkit-box-orient: vertical; -webkit-line-clamp: 3; overflow: hidden;

平移，缩放，旋转

transform:translate(x,y) rotate() scale()

**[前端制作动画的几种方式（css3，js）](https://www.cnblogs.com/zhangwenjiajessy/p/6154703.html)**

CSS

transition: property duration timing-function delay;

animation: name duration timing-function delay iteration-count direction;

Js

Jquery animate函数

$(selector).animate(styles,options)

终极方法：

canvas制作动画

栗子：canvas将上传图片压缩

const canvas = document.createElement('canvas')

const ctx = canvas.getContext('2d')

drawImage(image, sourceX, sourceY, sourceWidth, sourceHeight, destX, destY, destWidth, destHeight)

let compressed = canvas.toDataURL('image/jpeg', 0.7)

//Blob对象

var fileBlob = b64toBlob(compressed.substr(23), 'image/jpeg')

var fd = new FormData()

fd.append("file", fileBlob)

var xhr = new XMLHttpRequest()

xhr.open("POST", "upload.php")

xhr.send(fd)

基础API

let canvas = document.getElementById(‘canvas’)

let ctx = canvas.getContext(‘2d’)

线：

ctx.beginPath()

Ctx.moveTo(0,0)

Ctx.lineTo(10,10)

Ctx.lineWidth=2

Ctx.strokeStyle=’#333’

Ctx.stroke()

Ctx.closePath()

矩形：

ctx.fillRect(20,20,20,20)

Ctx.strokeRect(,,,)

Let myGradient=Ctx.createLinearGradient()

myGradient.addColorStop(0.5,color)

Ctx.fillStyle=myGradient

Ctx.clearRect()

文本：

Ctx.font=’bold 20px Arial’

Ctx.textAlign=’left’

Ctx.fillStyle=’#333’

Ctx.fillText(‘hello canvas’,x,y)//填充

Ctx.strokeText(‘hello’,x,y)//描边

圆：

Ctx.arc(0,0,10,0,Math.PI\*2,true)//逆时针、顺时针

Ctx.fillStyle=’orange’

Ctx.fill()

//圆弧

ctx.strokeStyle = 'orange'

ctx.stroke()

记录滚动位置的方法和属性

$(window).scroll(）

window.onscroll= function(){}

$(window).scrollTop()

window.scrollTop

$(‘body,html’).animate({scrollTop:0},{duration:1000,easing:’swing’})

window.scroll(0,Y)

canvas签名、刮刮卡、上传图片转为base64显示（canvas.toDataURL（））、转为blob（b64toBlob(imgBase64.substr(23), 'image/jpeg')）上传服务器

## 软键盘兼容性

示例：vue-reDiscover css.vue

iOS 上拉边界下拉出现空白

IOS 键盘弹起挡住原来的视图

IOS 键盘收起时页面没有回落，底部会留白

IOS 下 fixed 失效的原因

Zepto点透的解决方案

在上层显示以后加入对应的class名控制，截断显示层下方可获取焦点元素的事件获取

**解决移动端适配的问题：**

vue项目中使用lib-flexible px2rem-loader(可以解决1px问题)

例子：vue\_component

**lib-flexible**会自动在html的head中添加一个meta name="viewport"的标签，同时会自动设置html的font-size为屏幕宽度除以10，也就是1rem等于html根节点的font-size。假如设计稿的宽度是750px，此时1rem应该等于75px。但是当分辨率大于某个特定值时，它便不再生效。

检查一下html文件的head中，如果有 meta name="viewport"标签，需要将他注释掉，因为如果有这个标签的话，lib-flexible就会默认使用这个标签。而我们要使用lib-flexible自己生成的 meta name="viewport"来达到高清适配的效果。

核心做法：

~~根据设备的 dpr 动态改写 <meta> 标签，设置 viewport 的缩放~~

~~给 <html> 元素添加 data-dpr 属性，并且动态改写 data-dpr 的值~~

根据 document.documentElement.clientWidth 动态修改 <html> 的 font-size ，页面其他元素使用 rem 作为长度单位进行布局，从而实现页面的等比缩放

**px2rem-loader**自动将css中的px转成rem（需要配置基准字体remUint: 75）

1.此方法只能将.vue文件style标签中的px转成rem，不能将script标签和元素style里面定义的px转成rem

2.如果在.vue文件style中的某一行代码不希望被转成rem，只要在后面写上注释 /\* no\*/就可以了

不同多倍屏的图片选取

.btn-android {

background-image: url("../img/@2x/android.png?v=@@version");

[data-dpr="3"] & {

background-image: url("../img/@3x/android.png?v=@@version");

}

}

**postcss-px-to-viewport**插件主要用来把px单位转换为vw、vh、vmin或者vmax这样的视窗单位，也是vw适配方案的核心插件之一。

Package.json配置（vuecli3）

plugins: {

autoprefixer: {},

"postcss-px-to-viewport": {

viewportWidth: 375, // 视窗的宽度，对应的是我们设计稿的宽度.

viewportHeight: 667, // 视窗的高度，对应的是我们设计稿的高度.(也可以不配置)

unitPrecision: 3, // 指定`px`转换为视窗单位值的小数位数（很多时候无法整除）

viewportUnit: 'vw', // 指定需要转换成的视窗单位，建议使用vw

selectorBlackList: ['ignore', 'tab-bar', 'tab-bar-item'], // 指定不需要转换的类,后面再讲.

minPixelValue: 1, // 小于或等于`1px`不转换为视窗单位.

mediaQuery: false // 允许在媒体查询中转换`px`,

},

}

**postcss-viewport-units**插件主要是给CSS的属性添加content的属性，配合viewport-units-buggyfill库给vw、vh、vmin和vmax做适配的操作。

Git push后的版本回退

Git reflog查看回退版本号 git reset --hard 版本号

**预防XSS 跨域脚本攻击**

<meta http-equiv="Content-Security-Policy" content="script-src 'self'; style-src nos.netease.com kaola.com;">

对于脚本资源只信任本域下的，对于样式资源，除了本域还会加载nos.netease.com和kaola.com这两个域名下的。

==============1227==================

制作知识点思维导图

<https://mm.edrawsoft.cn/files>

账号：wangkunx@163.com 密码：

0224

移动端弹出键盘时遮挡输入框，让输入框与顶部平齐，但是安卓某些机型因为底部固定定位的元素被键盘顶起还会遮挡，故监听键盘是否弹出控制安卓底部元素的隐藏显示；顶部元素固定定位会失效的问题，尝试解决，效果不佳;

**Vue**

**钩子函数：**

beforeCreate created beforeMount mounted beforeUpdate updated beforeDestory destoryed

特殊的（keep-alive）activated deactivated

**路由守卫：**

全局 next需要被主动调用

beforeEach afterEach beforeEnter

组件内：

beforeRouteEnter beforeRouteUpdate beforeRouteLeave

beforeRouteEnter

此时组件实例还未被创建，不能访问this

传一个回调给next来访问组件实例，在导航被确认时执行回调，并把实例作为回调的参数；

可以在这个守卫中请求服务端获取数据，当成功获取并能进入路由时，调用next并在回调中通过 vm访问组件实例进行赋值等操作

beforeRouteEnter触发在导航确认、组件实例创建之前：beforeCreate之前；而next中函数的调用在mounted之后：为了确保能对组件实例的完整访问

beforeRouteEnter(to, from, next){

next(vm => {vm.clearFun()})

}

路由配置

一个路由匹配到的所有路由记录会暴露为 $route 对象的$route.matched 数组，搭配路由守卫 进行验证

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

if (to.matched.length ===0) { //如果未匹配到路由

from.name ? next({ name:from.name }) : next('/'); //如果上级也未匹配到路由则跳转错误页面，如果上级能匹配到则转上级路由

} else {

next(); //如果匹配到正确跳转

}

});

0226

$router和$route的区别：

//$router : 是路由操作对象，只写对象

//$route : 路由信息对象，只读对象

**Vue Router 的params和query传参的使用和区别**

使用params传参只能用name来引入路由，即push里面只能是name:’xxxx’,不能是path:’/xxx’,因为params只能用name来引入路由，如果这里写成了path，接收参数页面会是undefined

使用query传参使用path来引入路由。

params是路由的一部分,必须要在路由后面添加参数名。query是拼接在url后面的参数，没有也没关系。

滚动行为

<https://www.jianshu.com/p/c805b74e1f14?utm_campaign>

1px

<https://blog.csdn.net/bbnbf/article/details/51580569>

0302

要想给元素设置z-index样式，必须先让它变成定位元素；

用伪类写选中的底部边框样式（CSS边框长度控制）；

0311

**Git**

新建本地分支

Git branch dev

新建分支并切换

Git checkout -b dev

将本地分支推送远程分支（若远程仓库不存在，会新建）

Git push origin dev:dev

Git pull

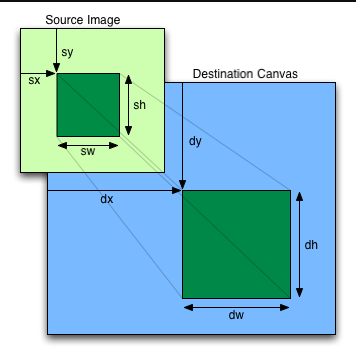
Git fetch --all

Git merge dev

Git push

canvas.getContext('2d').drawImage()方法参数

drawImage(image, sourceX, sourceY, sourceWidth, sourceHeight, destX, destY, destWidth, destHeight)



0318

canvas压缩图片上传

<https://xts.so/fe/compress-the-picture-with-canvas-before-uploading.html?ver=20150930>

0415

**哀悼日网站变灰色**

Html{webkit-filter:grayscale(100%);filter:grayscale(100%)}

0416

Chrome浏览器设置允许跨域

--disable-web-security --user-data-dir=E:\cors

0426

# 异步：

单线程，避免DOM渲染冲突

主线程（同步代码）执行结束， 轮询（event loop）异步队列

# Vue

MVVM

## jquery 、框架的区别：

数据、视图分离、解耦（开发、封闭原则）

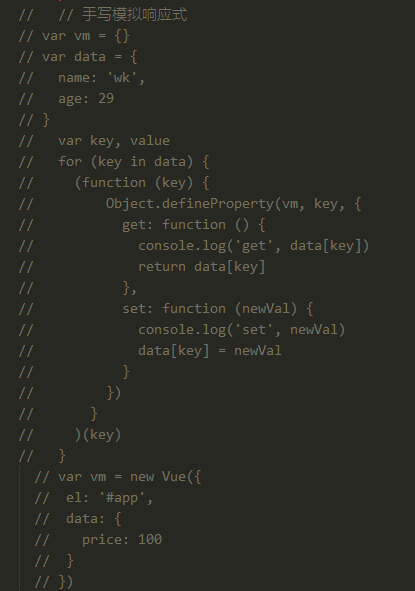
以数据驱动视图，只关心数据变化，DOM操作被封装

## vue三要素

### 响应式：

vue监听data属性的变化;data属性被代理到vm

Object.defineProperty



### 模板引擎：

vue模板如何被解析，指令如何处理

模板是什么 本质：字符串

最终转化为js函数（render函数）？：有逻辑，最终转化为html来显示

与html格式很像，但有很大区别

### 渲染：

Render函数

With

返回vnode

模板中的所有元素都包含在render函数中

Render函数与vdom--js模拟dom结构（节点，属性，子元素）

UpdateComponent中实现了vdom的patch

页面首次渲染执行UpdateComponent

Data中每次修改属性，执行UpdateComponent

## Vue实现流程：

1. 解析模板成render函数
2. 响应式开始监听
3. 首次渲染，显示页面，且绑定依赖

初次渲染，执行**UpdateComponent**，执行**vm.\_render()**

执行render函数，会访问vm.list和vm.title

会被响应式的get方法监听到

为何要监听get,直接监听set不行吗？

Data中有很多属性，有些被用到，有些可能用不到

被用到的走get，不被用到的不会走get

未走get的属性,set的时候也无需关心（必须走get才能走set）

避免不必要的渲染

执行UpdateComponent，会走到**vdom的patch**方法

patch将vnode渲染成DOM,初次渲染完成

1. Data属性变化，触发rerender

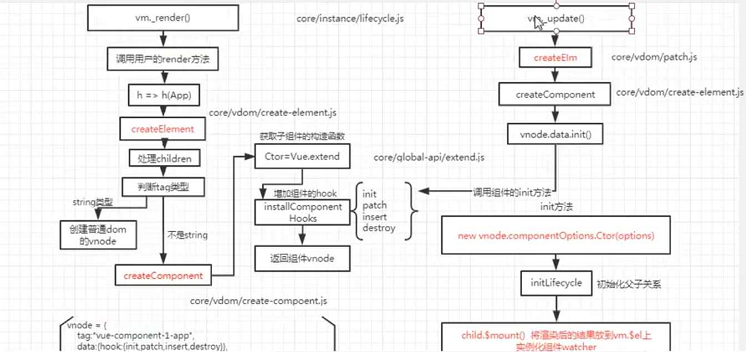
修改属性，被响应式的set监听到

Set中执行UpdateComponent

UpdateComponent重新执行vm.\_render()

生成的vnode和preVnode，通过patch进行对比

渲染到html中



## 响应式原理

### 检测变化的注意事项

#### 对于对象

1.Vue 无法检测 property 的添加或移除。

Vue.set(vm.someObject, 'b', 2)

2.为已有对象赋值多个新 property

this.someObject = Object.assign({}, this.someObject, { a: 1, b: 2 })

#### 对于数组

不能检测两种变化

1. 利用索引设置数组项

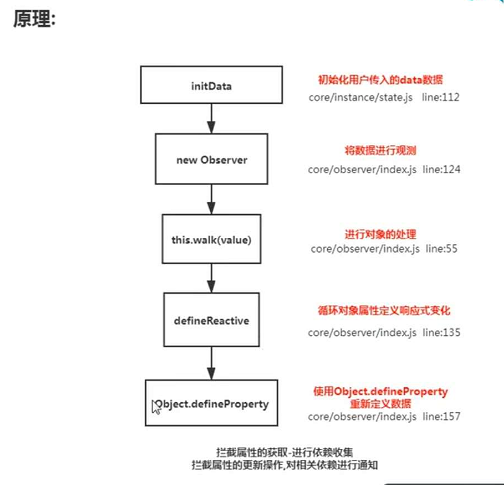
Vue.set(vm.items, indexOfItem, newValue)

vm.items.splice(indexOfItem, 1, newValue)

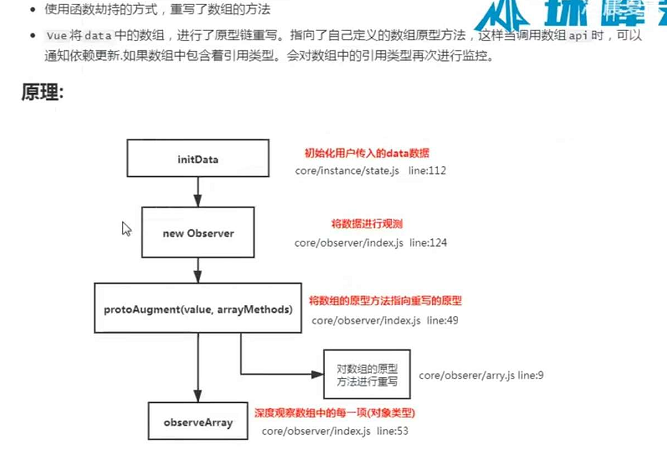
1. 改变数组长度

vm.items.splice(newLength)

对象



数组



### 异步更新队列

updateMessage: function () {

this.message = '已更新'

console.log(this.$el.textContent) // => '未更新'

this.$nextTick(function () {

console.log(this.$el.textContent) // => '已更新'

})

}

$nextTick() 返回一个 Promise 对象，所以你可以使用新的 ES2017 async/await 语法完成相同的事情

updateMessage: async function () {

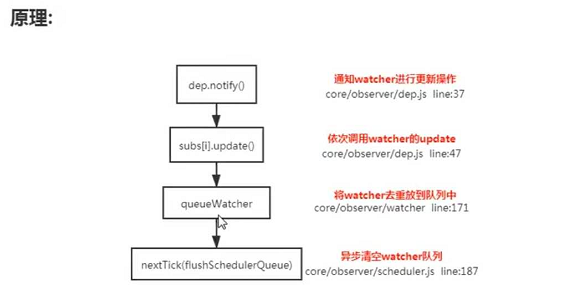
this.message = '已更新'

console.log(this.$el.textContent) // => '未更新'

await this.$nextTick()

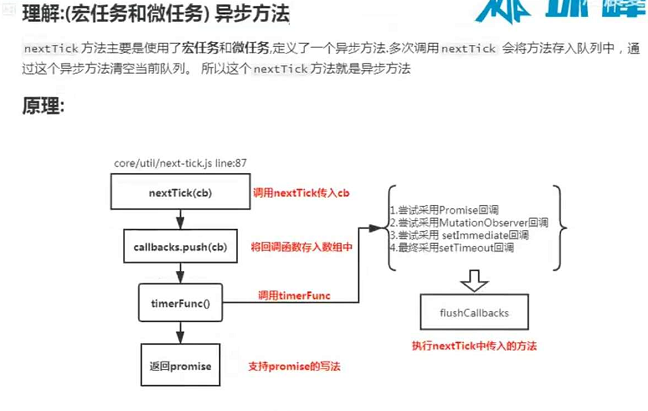
console.log(this.$el.textContent) // => '已更新'

}



NextTick

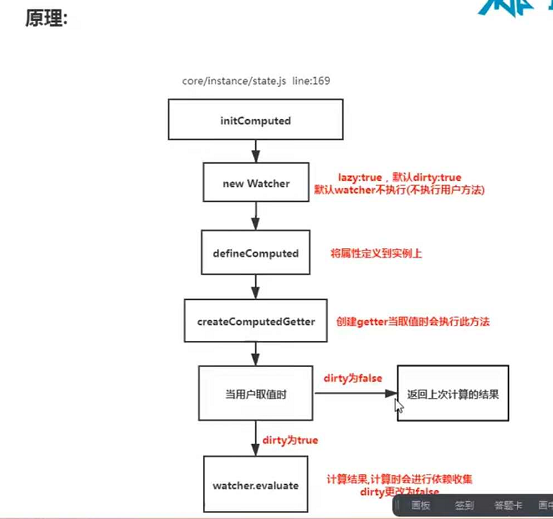
Vue 在内部对异步队列尝试使用原生的 Promise.then、MutationObserver 和 setImmediate，如果执行环境不支持，则会采用 setTimeout(fn, 0) 代替。



Computed

dirty实现缓存机制

写在模板中的数据{{computed}}调用JSON.Stringify()，默认对所有属性进行取值



## watch deep true如何实现

默认只监听外层函数，加deep把所有属性从头遍历到尾

## Vuex

**State**

唯一公共数据源，所有共享的数据统一放在Store中的State中存储。

访问State

This.$store.state.count

**mapState**

将当前组件需要的全局数据，映射为当前组件的computed计算属性

computed:{...mapState([])}

**Mutation**

用于变更State中的数据

调用mutation:

this.$store.commit(‘add’)

mapMutation将需要的mutation函数映射为当前组件的methods方法

**Action**异步任务

用来触发action:

this.store.dispatch()

mapAction将需要action函数映射为当前组件的methods方法

**Getter**

对store中的数据进行加工处理形成新的数据。

## vue中ref的作用

vm.$refs：持有已注册过的ref的所有子组件的，注册在父组件上的对象

Ref：给元素或子组件注册引用信息。如果在普通元素上使用，引用指向DOM元素，在子组件上，引用指向组件实例。

获取DOM元素

this.$refs.child

获取子组件的data，调用子组件方法

this.$refs.child.msg

## 父组件调用子组件方法：

1. 监听

This.$refs.child.$emit(‘childMethod’,’msg’)

1. this.$refs.child.callMethod()

子组件：

Methods:{

Listening () {

This.$on(‘childMethod’,(res) => {

Console.log(res)

})

},

callMethod () {}

}

父组件向子组件传值：

Props

## vue配置环境变量

Vue2中通过cross-env插件配置三种环境(开发，测试，生产)打包

1.config文件夹增加各种环境prod.env.js参数配置文件(创建项目时已存在)

2.index.js 引入build:{prodEnv:require(./prod.env)}

3.修改package.json配置打包命令：’build:prod’:’cross-env NODE\_ENV=production env\_config=prod node build/build.js’

4.webpack.prod.conf.js配置全局变量

const env = config.build[process.env.env\_config + 'Env']

new webpack.DefinePlugin({

'process.env': env

})

DefinPlugin里面定义的必要性是   你可以在你的src 目录下的文件里直接使用process.env.NODE\_ENV

5.使用：process.env.NODE\_ENV === ‘production’

Vuecli3

环境变量的使用 :

根路径下配置文件.env

.env.production

只有以 VUE\_APP\_ 开头的变量会被 webpack.DefinePlugin 静态嵌入到客户端侧的包中(即在项目代码中使用)。console.log(process.env.VUE\_APP\_SECRET)

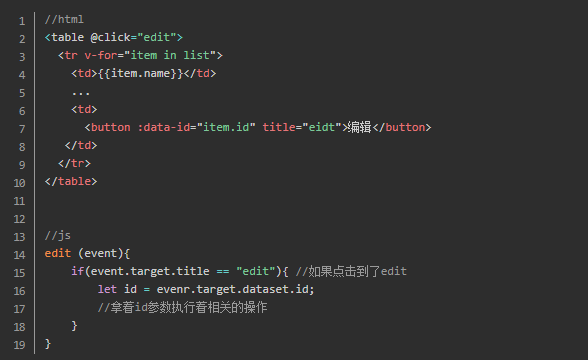
package.json 配置"build:prod": "vue-cli-service build --mode production",

## Vue项目优化

编码优化：

V-for v-if不连用（v-for优先级更高）若必须连用，用computed先过滤

V-for绑定事件使用事件代理



Keep-alive缓存组件

Key保证唯一性（vue默认就地复用（同标签tag、同key）策略）

v-for 不加key，默认加索引（静态组件可以）

Object.freeze冻结数据（避免被监听）

加载性能优化：

Vue-virtual-scroll-list

Vue-lazyload

用户体验：

App-skeleton骨架屏

App-shellapp壳

# Virtual dom

Vdom 是vue和react的核心

Vdom是啥？为啥存在？

用js模拟DOM结构

DOM变化的对比，放在js层来做（图灵完备语言）

提高重绘性能

如何应用，API？

H函数、Patch函数

# Diff算法

为啥要用？

Dom操作昂贵，减少DOM操作

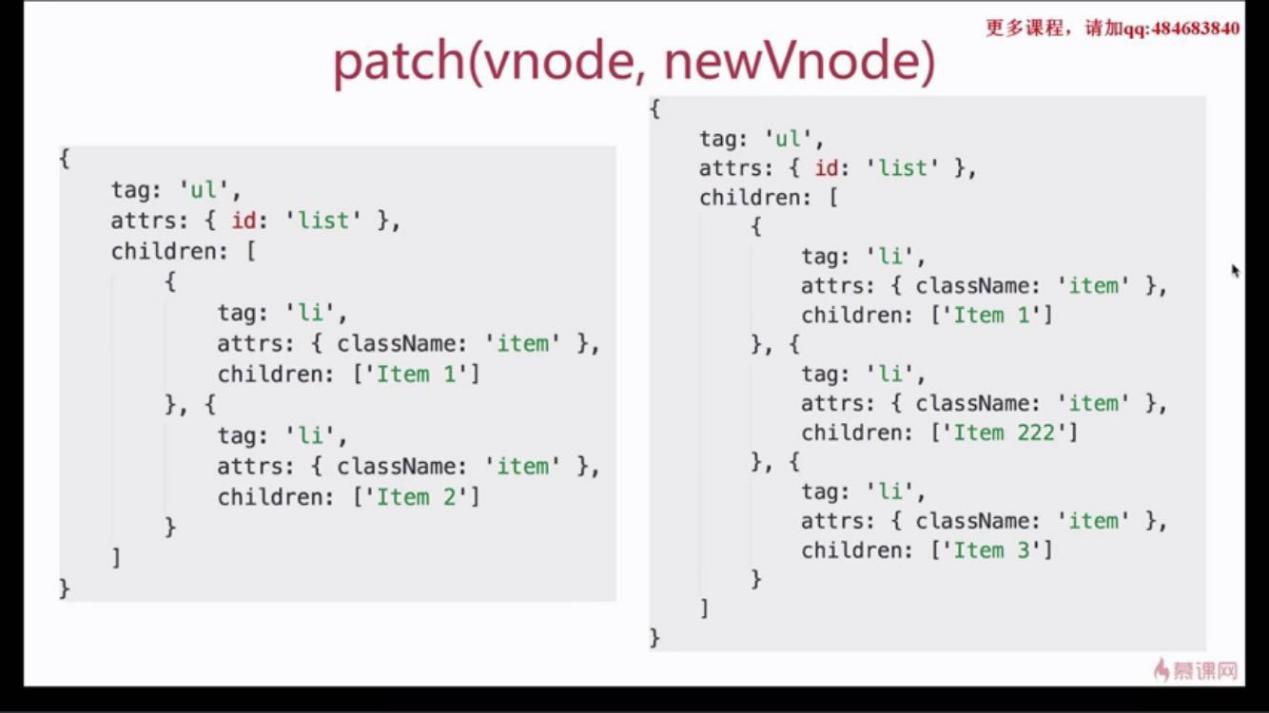
找出本次DOM必须更新的节点更新，其他不动

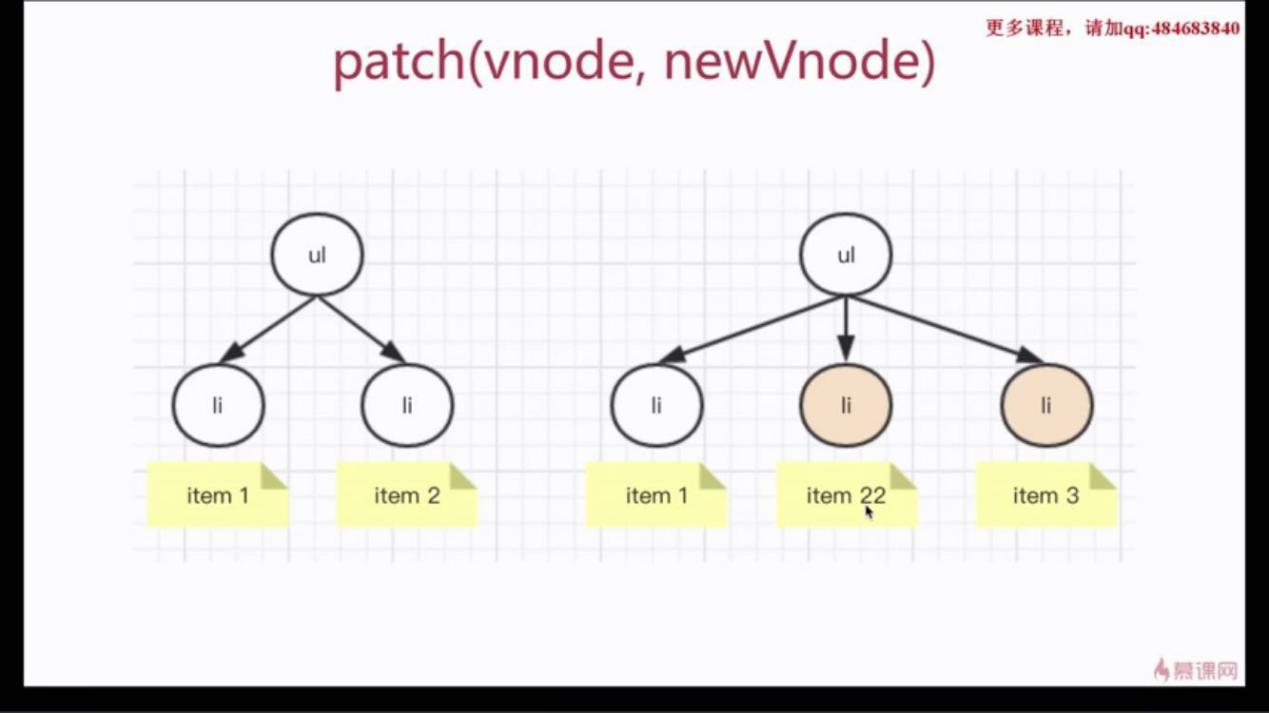
找出的过程，需要diff

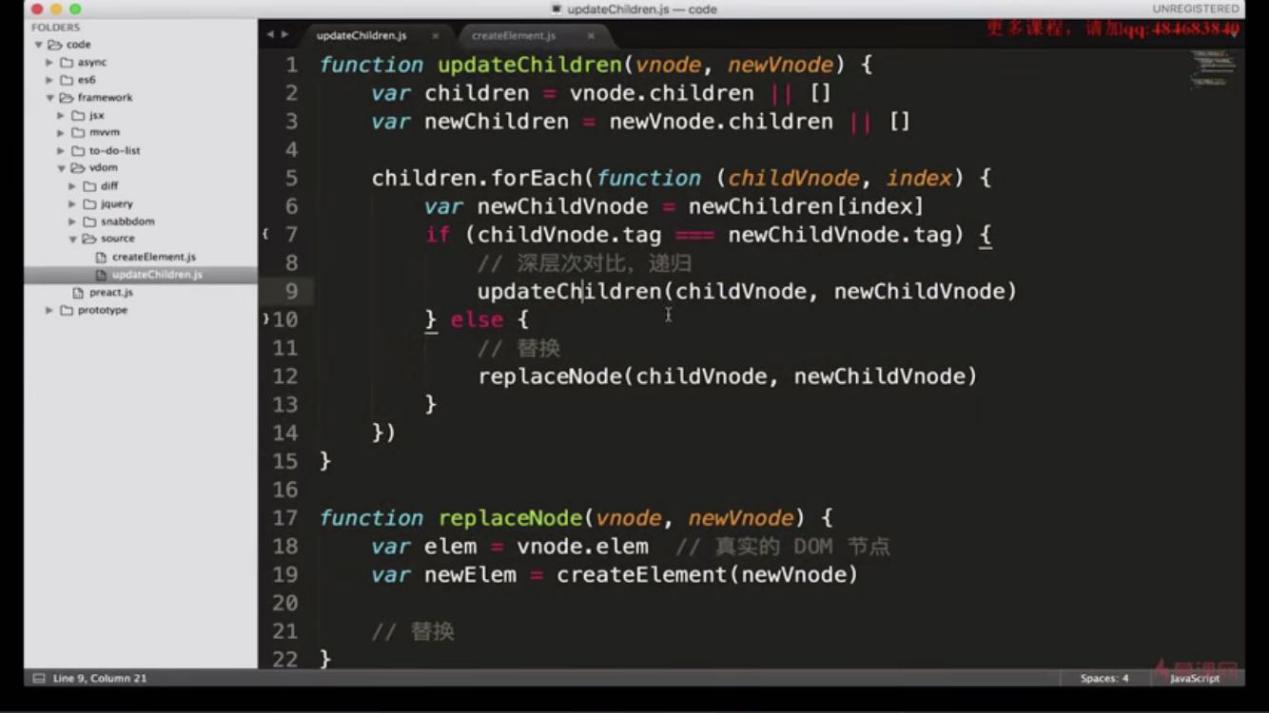
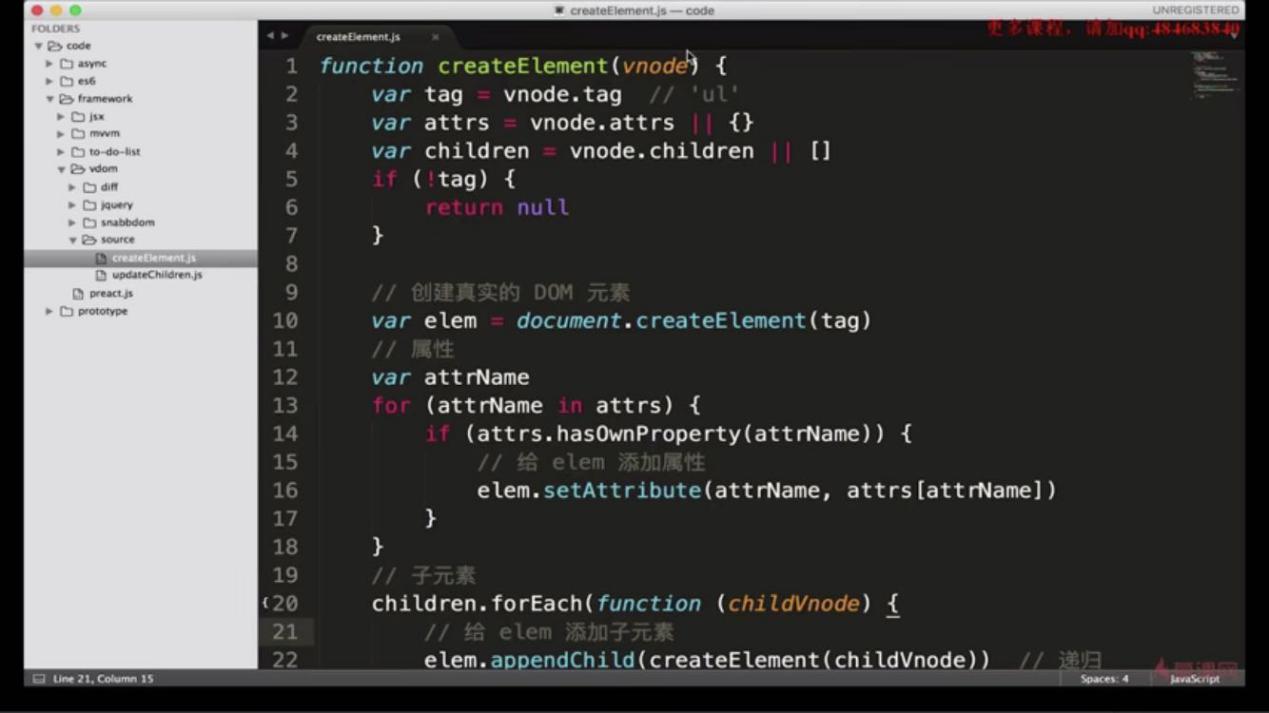
实现过程： 核心逻辑：

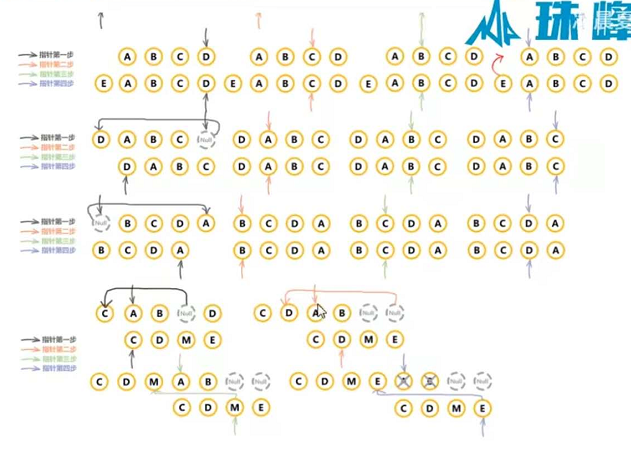
Patch(container,vnode) CreateElement

Patch(vnode,newVnode) updateChildren









比较顺序：头头 尾尾 头尾 尾头

NextTick应用场景

在Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中

在数据变化后要执行的某个操作，而这个操作需要使用随数据改变而改变的DOM结构的时候，这个操作都应该放进Vue.nextTick()的回调函数中。

# 原型：

vm demo

jQuery 如何使用原型？

入口函数、构造函数、构造函数的原型

扩展：$.fn = function () {}

优点：只开放了$，统一规范

## \_\_proto\_\_ constructor prototype

总结一下：

1.我们需要牢记两点：①\_\_proto\_\_和constructor属性是对象所独有的；② prototype属性是函数所独有的，因为函数也是一种对象，所以函数也拥有\_\_proto\_\_和constructor属性。

2.\_\_proto\_\_属性的作用就是当访问一个对象的属性时，如果该对象内部不存在这个属性，那么就会去它的\_\_proto\_\_属性所指向的那个对象（父对象）里找，一直找，直到\_\_proto\_\_属性的终点null，再往上找就相当于在null上取值，会报错。通过\_\_proto\_\_属性将对象连接起来的这条链路即我们所谓的原型链。

3.prototype属性的作用就是让该函数所实例化的对象们都可以找到公用的属性和方法，即f1.\_\_proto\_\_ === Foo.prototype。

4.constructor属性的含义就是指向该对象的构造函数，所有函数（此时看成对象了）最终的构造函数都指向Function。

# Es6

## Es6模块化如何使用，开发环境如何打包

模块化的基本语法

开发环境配置

Babel

电脑有node环境，运行npm init

npm install babel-core babel-preset-es2015 babel-preset-latest

创建.babellrc文件

npm install --global babel-cli

babel --version

Webpack

http-server 是一个简单的零配置的命令行 http服务器，它足够强大便于生产和使用，用于本地测试和开发。

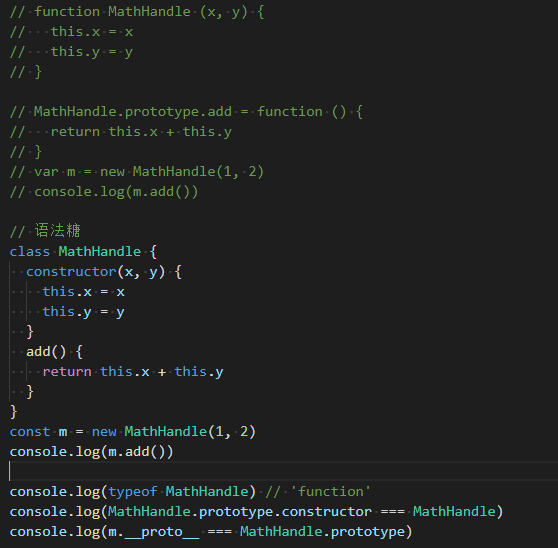
Js众多模块化标准

没有模块化

AMD成为标准，require.js(也有CMD)

## Class

和普通构造函数有何区别



继承---js

语法糖的本质！！！会用（表面）------背后

## Promise

的基本使用和原理

Callback Hell

### promise语法

new Promise实例，而且要return

new Promise时要传入函数，函数有resolve reject两个函数

成功时执行resolve() 失败执行reject()

then监听结果

## ES6其他常用功能

let/const

多行字符串/模板变量

解构赋值

块级作用域

函数默认参数

箭头函数

# 组件化

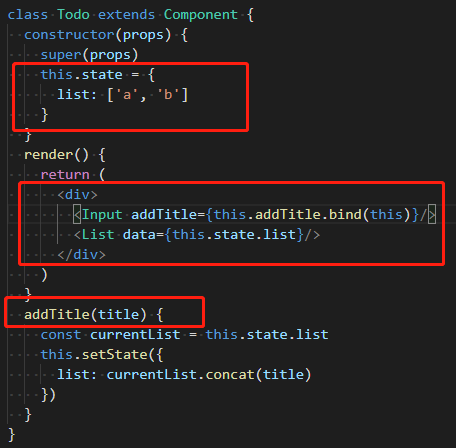
的理解

## 封装

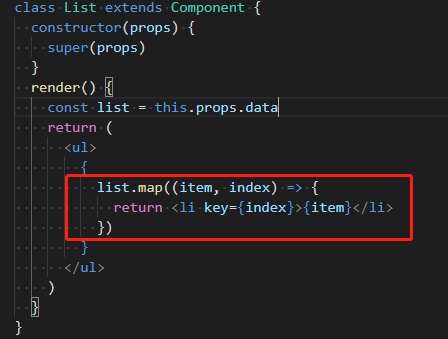
视图

数据

数据变化逻辑（数据驱动视图变化）



## Props传递、复用



# JSX

本质是啥

## JSX语法

html形式

引入JS变量和表达式

If...else...

循环

Style className

事件

JSX语法根本无法被浏览器所解析

## JSX解析成JS

JSX其实是语法糖

开发环境会将JSX编译成JS代码

JSX的写法大大降低了学习成本和编码工作量

同时，JSX也会增加debug成本

## 为何需要vdom

JSX就是模板，最终渲染成html

初次渲染+修改state后的rerender

React.createElement()

独立的标准

JSX是React引入的，但不是React独有的

## setState的过程：

每个组件实例，都有renderComponent方法

执行renderComponent会重新执行实例的render

render函数返回newVnode，然后拿到preVnode

执行patch(preVnode, newVnode)

setState vue修改属性 异步的，目的是避免重复修改数据，提供效率

# React

Render()书写规则

Class App extends Component{

constructor(props){

super()

this.state={

list:[]

}

}

render(){

Return ( //（ 可以换行

<div className> //< html

{false?1:2} // { js

</div>

)

}

}

备注：快捷键

事件绑定onClick，箭头函数避免this指向问题

数据存储（setState），setState是异步操作，需要在回调函数获取更新的DOM

this.setState({

list: [...this.state.list,this.state.inputValue],

inputValue: ''

},()=>{

console.log(this.ul.querySelectorAll('li').length)

})

this.$nextTick(function () {

console.log(this.$el.textContent) // => '已更新'

})

生命周期

父子组件传值、事件调用

Ref

有状态组件、无状态组件：

class关键字创建的，有私有属性state、生命周期函数

构造函数创建的，只有props

props、state区别：

props是从外界传递的；只读的，不能重新赋值

State私有数据；数据可读可写

React没有指令

css模块化modules:避免样式污染，利用名称转换，让css拥有作用域，避免全局样式污染



# React vs vue

## 本质区别：

### 模板的区别

Vue--使用模板（最初由angular提出）

React--使用JSX

模板语法上，倾向JSX

Vue模板指令需要学习

JSX render()直接写JS逻辑

模板分离上，倾向vue

JSX模板和JS混子一起，未分离（开发封闭原则）

### 组件化的区别

React本身就是组件化，更清晰

Vue也支持组件化，不过是在MVVM上的扩展

### 共同点：

组件化

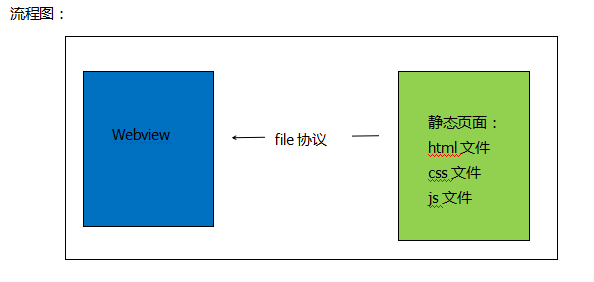
国内使用，首推vue。文档更易读、易学、社区够大

团队水平较高，推荐使用React。组件化和JSX

# hybrid

是什么，为何使用

更新和上线的流程



## 完整流程：

分版本，有版本号，如202004301330

将静态文件压缩成zip包，上传到服务器

客户端每次启动，都去服务器检查版本号

如果服务器端版本号大于客户端版本号，就去下载最新的zip包

下载完成之后解压包，然后将现有的文件覆盖

## hybird vs h5

优点：

体验好，可快速迭代

缺点：开发成本高，运维成本高

使用场景：hybird适合产品型，h5适合运营型

# 热爱编程：

书：构建知识体系

博客：合格程序员必备，总结、交流方式

开源：github的star是硬通货

协议：

HTTP

新知：

vue构造器:Vue.extend()

vnode 指令背后的原理 mixins

0506

对象的属性名，不一定是字符串（基本类型：字符串，数字，布尔，null,undefined 引用类型 Symbol），运行时可能会转化成字符串

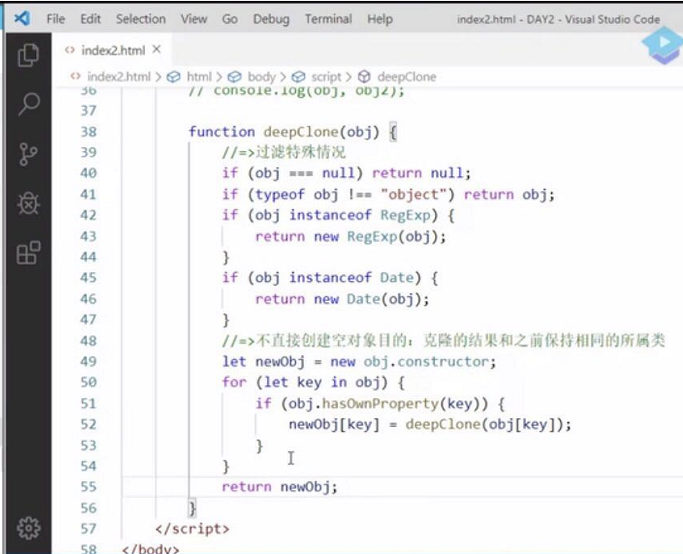


引用类型都是堆

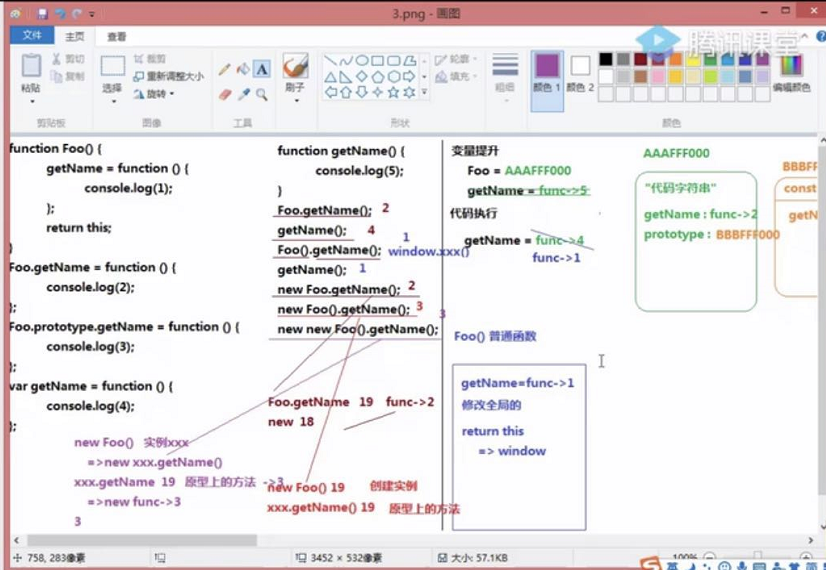
eval()函数把含有转义字符的字符串转换成Object对象

头疼的面试题

深度遍历：



变量提升，代码执行：



**vue.config.js配置**

**publicPath**

Default: '/'

这个值也可以被设置为空字符串 ('') 或是相对路径 ('./')，这样所有的资源都会被链接为相对路径，这样打出来的包可以被部署在任意路径。

## 移动端click事件300ms的延迟响应

1. 浏览器开发商的解决方案
   1. **禁用缩放**

<meta name="viewport" content="user-scalable=no"> <meta name="viewport" content="initial-scale=1,maximum-scale=1">

1.2**更改默认的视口宽度**

<meta name="viewport" content="width=device-width">

1.3**方案三：CSS touch-action**

Fastclick bug

引发的问题：fastclick影响el-upload组件调用

fastclick影响input focus/click事件，多次点击才能唤起

zepto的touch模块，tap事件也是为了解决在click的延迟问题

# Node.js

单线程

非阻塞I/O

事件驱动

安装nodemon，完毕后，需要以管理员身份运行

设置set-ExecutionPolicy RemoteSigned

nodemon express.js

Mongodb express数据库增删改查

windows环境下启动mongodb服务

mongodb安装目录下的“bin”目录(C:\Program Files\MongoDB\Server\4.2\bin)中，打开命令窗口，输入命令：mongod --dbpath F:\mongdb\data

Rest client 代码形式发起请求

点击send request，可以看到响应信息

请求之间用###隔开

请求路径

Get/post uri

express mongodb实现功能：注册、登录、授权

使用express，快速开发

使用mongoose创建、操作本地数据库

使用jsonwebtoken加密id生成token、解密token验证身份信息

使用bcrypt加密密码，比较加密后的密码是否一致

小程序：原生，mpvue

Nodejs：adonis.js

**项目经历：**

自己做的、技术难点、亮点

1.评价模块：new Imae() img.onload预载完成事件，

.cover { object-fit: cover; }未实现居中效果

1. **z-index失效问题：**

问题标签无position属性（不包括static）

祖先标签的z-index值比较小

1. 服务端缓存问题（需刷新页面，清除缓存）

<https://www.cnblogs.com/qiu-Ann/p/11394504.html>

**日期选择组件**

闰年能被4整除且不能被100整除，或能被400整除。

**使用iconfont字体图标**

优点：1、轻量性：一个图标字体比一系列的图像（特别是在Retina屏中使用双倍图像）要小。一旦图标字体加载了，图标就会马上渲染出来，不需要下载一个图像。可以减少HTTP请求，还可以配合HTML5离线存储做性能优化。

2、灵活性：图标字体可以用过font-size属性设置其任何大小，还可以加各种文字效果，包括颜色、Hover状态、透明度、阴影和翻转等效果。可以在任何背景下显示。使用位图的话，必须得为每个不同大小和不同效果的图像输出一个不同文件。

3、兼容性：网页字体支持所有现代浏览器，包括IE低版本。

Async/await 方法改写嵌套调用的接口（极度舒适）

async function getDatas() {

let result1 = await that.getXYPackageIds(id);

await that.getCitys(result1);

await that.getOrgListByCity()

}

getDatas()

await 操作符用于等待一个Promise 对象。await 表示在这里等待promise返回结果了，再继续执行。它只能在异步函数 async function 中使用。

await 等待的虽然是promise对象，但不必写.then(..)，直接可以得到返回值。

Promise

.all([that.runAsync1(), that.runAsync2()])

.then(function (results) {

})

**普通函数、promise 、setTimeout执行顺序**

创建Promise实例时是同步的

promise.then()、setTimeout都是异步执行的，都会在异步队列中等待执行。

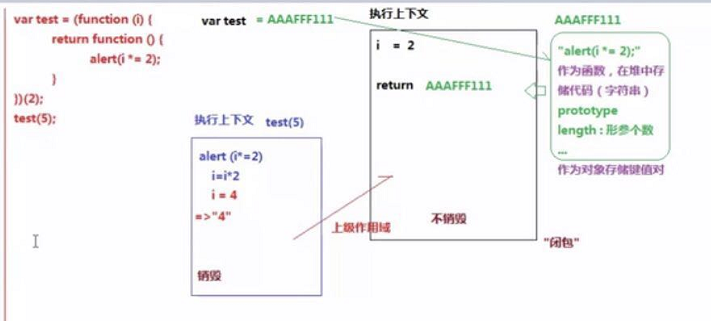
而promise.then()异步会放到microtask queue（微任务队列）中，microtask queue队列中的内容是在当前脚本执行完毕后立即发生的事。所以当同步Promise创建实例完成后，就执行promise.then()，然后把setTimeout放入执行堆栈中执行，故先执行Promise，再执行setTimeout。

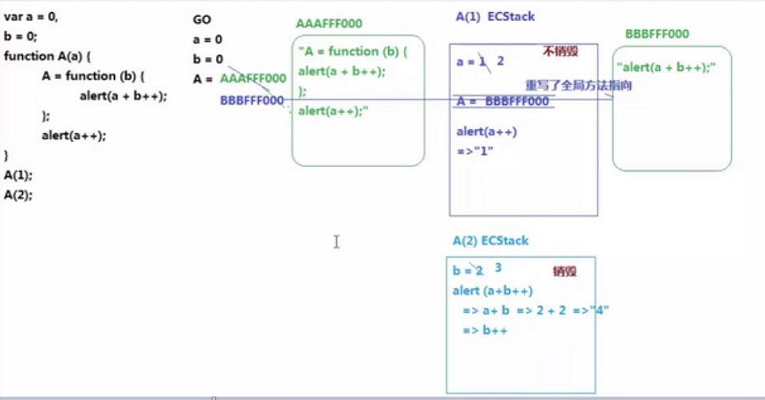
数组拼接的3中方式：

堆：存储引用类型的空间

栈：存储基本类型值和执行代码的环境

执行上下文：闭包





# http

一次完整的http请求

DNS解析，得到IP，找到对应的服务器发起TCP三次握手，建立TCP连接，发起http请求，服务器响应请求，浏览器得到html代码，浏览器解析html代码，并请求其中的资源，浏览器渲染页面，服务器关闭TCP连接

状态码：

200请求成功

403禁止访问 - 服务器已了解请求，但将拒绝满足该请求，如token过期

405 (Method Not Allowed)

500服务器错误，如后台上服务

对非Array对象类型的遍历

Object.getOwnPropertyNames(object)

Object.keys(object)

Object.entries(object)

# 珠峰架构

实现Vue高级树形组件

手写promise

离职原因：

你在现在的团队处于什么样的角色，起到了什么明显的作用？

你认为怎样才是全端工程师（Full Stack developer）？

介绍一个你最得意的作品吧？

项目中遇到什么问题？如何解决？

你的优点是什么？缺点是什么？

如何管理前端团队?

最近在学什么？能谈谈你未来3，5年给自己的规划吗？

1.你未来一到三年的一个职业规划是什么？

2.你都是怎么去学习和关注新技术的？

3.你近几年工作中有哪些心得或总结？

4.你觉得你在工作中的优缺点是什么？

5.你过来我们公司，你的优势是什么？

6.你有那些你觉得你写的不错的代码块，分享一下。

7.有些过开源项目吗？

8.写过 npm 包吗，写过 webpack 插件吗？

1. 看过哪些框架或者类库的源码，有什么收获？

<https://blog.csdn.net/u012207345/article/details/78188450>

面试全面的

<https://www.cnblogs.com/autismtune/p/5210116.html>

vuex存储与本地储存（localstorage、sessionstorage）的区别

前端内存泄漏检查判断及处理

浏览器信息navigator.userAgent

薪资问题

<http://www.ctoutiao.com/2216747.html>

# HTTP

灵魂拷问

## 报文结构

起始行 头部 空行 实体

请求起始行：GET /home HTTP/1.1

响应起始行：HTTP/1.1 200 OK

## 请求方法

**get post区别**

get 请求会被浏览器主动cache （缓存），post 则不会，除非手动设置

get 只接受ASCII 码字符，而post 没有限制

get 请求只能进行url 编码，而post 支持多种编码方式(application/json multipart/form-data)

get 产生一个tcp 数据包，浏览器会把http header 和data 一并发送出去，服务器响应200（返回数据）

 post 产生两个tcp 数据包，浏览器会先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200（返回数据）

## 状态码

403 token过期

405 请求方法错误

500服务器出错， 上服务

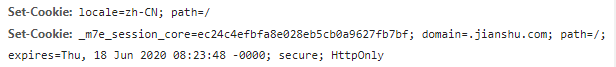
502 Bad Gateway

504 Gateway Time-out

## Cookie

**对 Cookie 了解多少？**

HTTP 是一个无状态的协议，HTTP 为此引入了 Cookie，用来保存状态



如果带上Secure，说明只能通过 HTTPS 传输 cookie。

如果 cookie 字段带上HttpOnly，那么说明只能通过 HTTP 协议传输，不能通过 JS 访问，这也是预防 XSS 攻击的重要手段。

### 作用域

关于作用域也有两个属性: ****Domain****和****path****, 给 ****Cookie**** 绑定了域名和路径，在发送请求之前，发现域名或者路径和这两个属性不匹配，那么就不会带上 Cookie。值得注意的是，对于路径来说，/表示域名下的任意路径都允许使用 Cookie。

### 生存周期

Cookie 的有效期可以通过**Expires**和**Max-Age**两个属性来设置。

* **Expires**即过期时间
* **Max-Age**用的是一段时间间隔，单位是秒，从浏览器收到报文开始计算。

若 Cookie 过期，则这个 Cookie 会被删除，并不会发送给服务端。

### Cookie 的缺点

容量4kb、性能、安全

## HTTP/2

改进：

头部压缩（HPACK算法：压缩请求头）

多路复用：HTTP 队头阻塞（根本原因在于HTTP 基于请求-响应的模型，在同一个 TCP 长连接中）

Tmall.com使用，如何区分

:authority"头部代替"Host"头部。

Chrome F12里面，HTTP/1.1有"view source"，而HTTP/2是binary，没有"view source

# Axios

Axios.get(url,config)

Axios.post(url,data,{})

Axios({baseUrl:’’, url:’’, method:’’, data:’’,cancelToken:new CancelToken()})

基于Promise的HTTP 请求库

## **特性**

* 从浏览器中创建 [XMLHttpRequests](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest" \t "http://www.axios-js.com/zh-cn/docs/_blank)
* 从 node.js 创建 [http](http://nodejs.org/api/http.html" \t "http://www.axios-js.com/zh-cn/docs/_blank) 请求
* 支持 [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise" \t "http://www.axios-js.com/zh-cn/docs/_blank) API
* 拦截请求和响应
* 转换请求数据和响应数据
* 取消请求
* 自动转换 JSON 数据
* 客户端支持防御 [XSRF](http://en.wikipedia.org/wiki/Cross-site_request_forgery" \t "http://www.axios-js.com/zh-cn/docs/_blank)

# Js高级

## 闭包

当嵌套的内部函数引用了外部函数的变量时，就产生了闭包

闭包：是嵌套的内部函数（存在于嵌套的内部函数中）

作用：

外部函数的变量在函数执行后，仍存活在内存（延长了局部变量的生命周期）；

函数外部可以操作函数内部变量

生命周期：

产生：嵌套内部函数定义执行时

死亡：嵌套内部函数成为垃圾对象

内存溢出：当程序运行需要的内存超过了剩余内存时出现的程序错误

内存泄漏：占用的内存没有及时释放

内存泄漏累积多了，容易引起内存溢出

常见的内存泄漏：意外的全局变量，没有及时清理掉的定时器或回调函数，闭包

终极闭包题：

function fun(n,o){

console.log(o)

return {

fun:function(m){

return fun(m,n)

}

}

}

var a = fun(0)

a.fun(1)

a.fun(2)

a.fun(3)

var b = fun(0).fun(1).fun(2).fun(3)

o {

fun:function(m){

return fun(m,0)

}

}

fun(1,0) 0 fun(1,o)

fun(m,n) fun(2,1)

var c = fun(0).fun(1)

c.fun(2)

c.fun(3)

<https://m.imooc.com/article/68898?block_id=tuijian_wz>

啃不动，放着吧~

定时器引发的思考：

## 原型链

访问一个对象的属性时，先在自身属性中查找，找到返回，若没有，沿着\_\_proto\_\_链向上找，找到返回，如最终没找到，返回undefined

别名：隐式原型链

作用：查找对象的属性/方法

## Prototype \_\_proto\_\_

prototype

每个函数都有一个prototype属性，默认指向一个Object空对象（原型对象）

原型对象都有一个属性constructor，指向函数对象

给原型对象添加属性（一般是方法）

作用：函数的所有实例对象自动拥有原型中的属性/方法

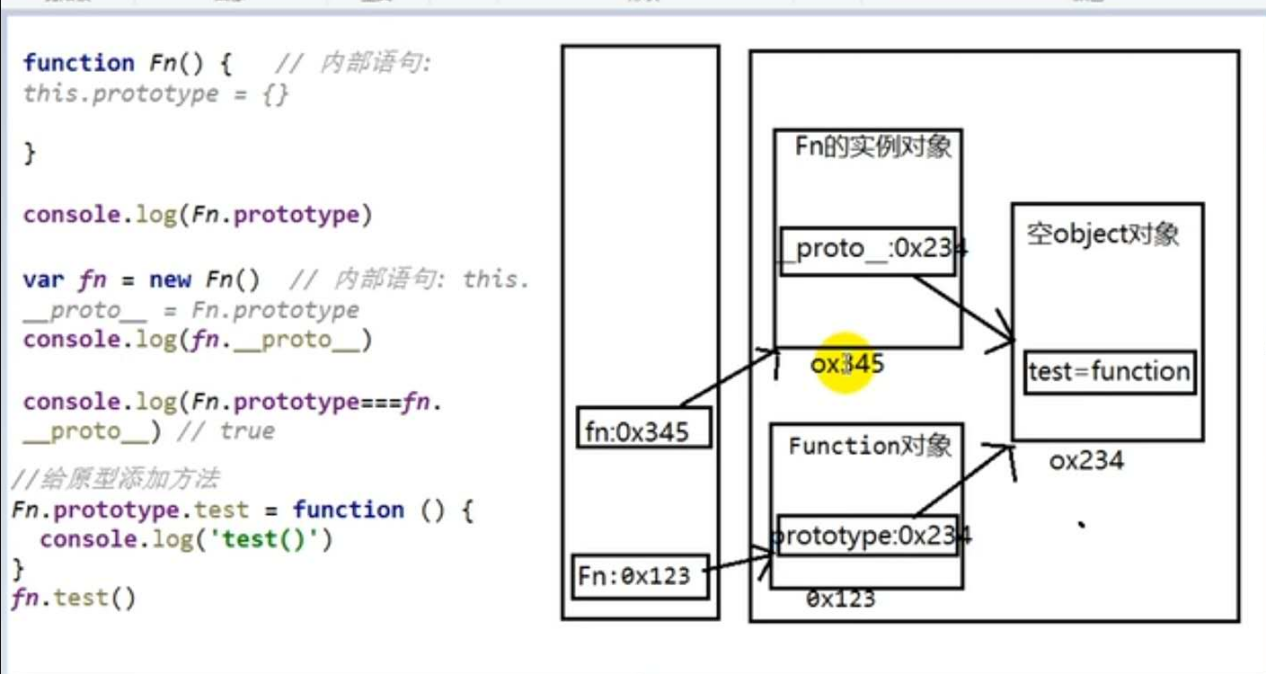
\_\_proto\_\_

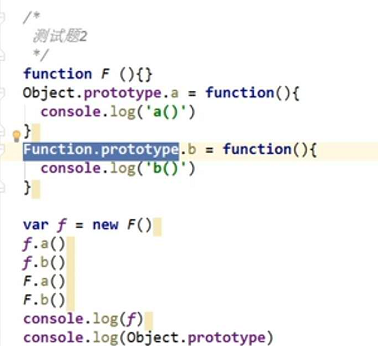
对象的\_\_proto\_\_属性：创建对象时自动添加的，默认值为构造函数的prototype属性值

function Fn() {} //内部语句：this.prototype={}

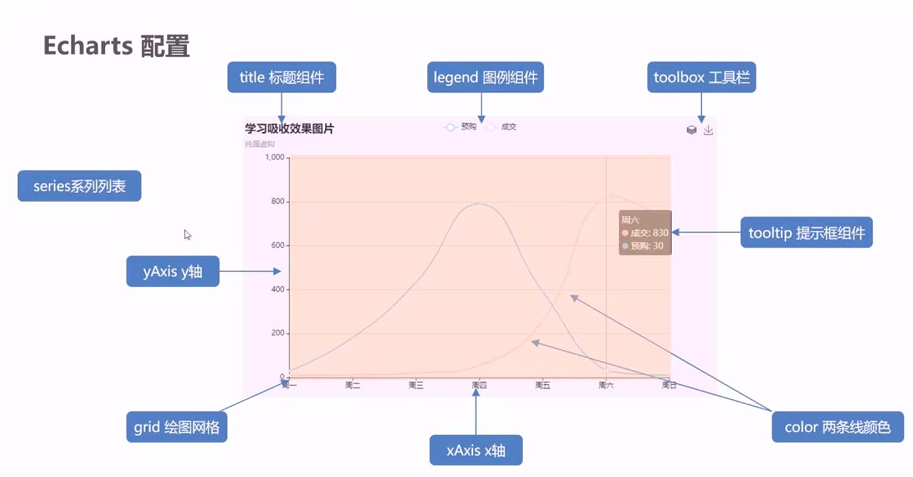
var fn=new Fn() //内部语句：this.\_\_proto\_\_=Fn.prototype

fn.\_\_proto\_\_===Fn.prototype //true





# Echarts.js



RESTful API架构风格

<https://segmentfault.com/a/1190000020029993?utm_source=tag-newest>

mpvue美团小程序框架

# **vue computed watch区别**

子组件中需要同时显示改变前和改变后的值。

=================================

// 缓存 userInfo

computed: {

userInfo () {

return { ...this.user }

}

},

watch: {

userInfo: {

handler (val, oldVal) {

this.oldUser = oldVal || val

},

deep: true,

immediate: true

}

}

=================================

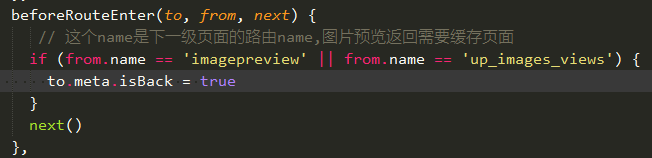
{ ...this.user } 或者使用 Object.assign({}, this.user) 来创建新的引用

# 项目技术点：

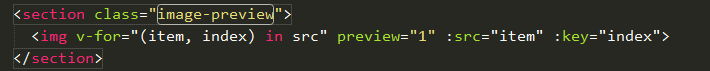
全局前置守卫、scrollBehavior缓存浏览位置



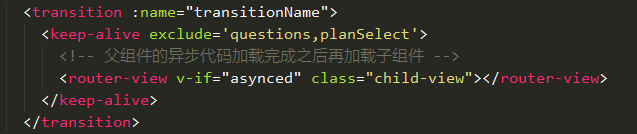
组件缓存==meta:{isBack:false}控制 activated this.$route.meta.isBack



vue-photo-preview大图预览



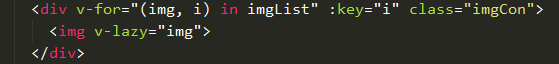
选择性缓存组件



图片批量上传：递归调用接口

图片懒加载

用法：



F12，审查元素：先将图片地址放在data-src，滚动到位置放置src，获取图片



请求列表返回图片id,循环根据id拿到真实图片地址

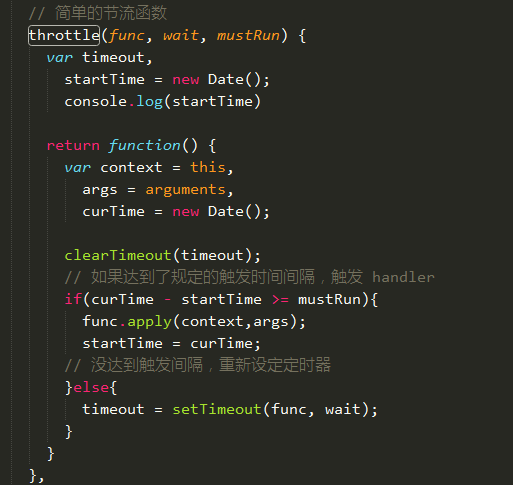


绑定src



节流模式：单位时间内只执行最后一次





$(window).scroll(() => {

$(window).scrollTop()

this.refs.child.offsetTop

})

var reg = /([^\u0020-\u007E\u00A0-\u00BE\u2E80-\uA4CF\uF900-\uFAFF\uFE30-\uFE4F\uFF00-\uFFEF\u0080-\u009F\u2000-\u201f\u2026\u2022\u20ac\r\n])|(\s)/g

改变数组的方法：

push pop shift unshift splice sort reverse

Inflection

通用CRUD接口

node使用express+multer文件上传

// 腾讯云COS

let COS = require('cos-nodejs-sdk-v5');

let cos = new COS({

SecretId: 'AKIDOYS6Wh9w1qQjT1HJAnAG4VgZEk294w2r',

SecretKey: 'iDZhZKo7WeWBcmhQrEMDoKZXDVJ1UkkX'

});

app.post('/admin/api/upload', authMiddleware(), upload.single('file'), async (req, res) => {

//之所以可以用req.file获取到文件数据，是因为用multer库的upload.single('file')将file参数赋值到req上

let file = req.file

//文件路径

let FilePath = file.path;

//文件名

let fileName = file.filename

var time = new Date().getTime();

//上传文件至腾讯云COS

cos.sliceUploadFile({

Bucket: 'wangkun-1302563356',

Region: 'ap-beijing',

Key: `${fileName}${time}.jpg`,

FilePath

}, (err, data) => {

err && res.status(403).send({ code: 0, msg: err.errmsg });

res.send({ code: 1, data: `https://${data.Location}` });

});

})

vue和微信小程序的区别、比较

<https://segmentfault.com/a/1190000015684864>

Vue-Socket.io的使用步骤

<https://blog.csdn.net/lily2016n/article/details/102483090>

JS垃圾回收机制

<https://segmentfault.com/a/1190000018605776?utm_source=tag-newest>

标记清除chrome 引用计数IE

运用**惰性删除**和定时删除实现可过期的localStorage缓存

大文件上传

18310612838

color: #1d2847;

DOM Ready

DOM加载完成，图片等外部文件加载之前

document.addEventListener(‘DOMContentLoaded’,function(){})

$(document).ready()

Ie8:document.onreadystateChange=funtion(){}

DOM Load

window.onload=funtion(){}

$(window).load()

1px边框

解决方法：使用伪类元素模拟边框，使用transform缩放

@media screen and (-webkit-min-device-pixel-ratio: 4){

.button:after{}

}

稍后再看

Fetch

<https://www.jianshu.com/p/7762515f8d1a>

监听文档滚动位置

$(window).scroll(function(){$(window).scrollTop()})

$("html, body").animate({scrollTop: 0}, { duration: 500, easing: "swing" })

微信小程序实操：

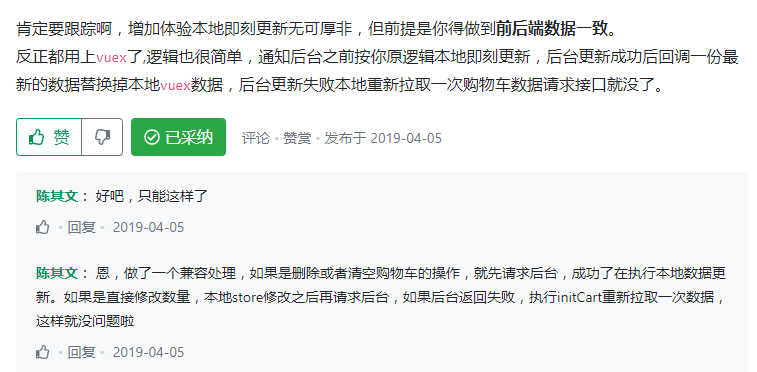
绑定事件：bindinput

无法在事件中直接传参，通过自定义参数data-num ，e.currentTarget.dataset.num

保存数据this.setDtata() 读取数据：this.data.

缓存：

wx.setStorageSync(‘key’,’value’) wx.getStorageSync() 类型不做转换



# H5

表单输入类型：Input type:number search tel time date month week

表单属性：autocomplete placeholder form

语义化标签：header main footer aside section nav

音视频：video audio

画布：canvas

事件：resize input

web存储：sessionStorage localStorage

C3:

1、选择器：nth-child（）、nth-of-type（）、：checked、：disabled

2、盒模型：box-sizing:

content-box (盒模型宽度=width+padding+border)

 border-size(盒模型宽度=width

padding+border包含在width)

3、背景：background-image、background-size、background-origin

1. 渐变：linear-gradient、radial-gradient

*background-image: linear-gradient(direction, color-stop1, color-stop2, ...);*

5、边框：border-radius、border-image

1. 阴影：box-shadow、text-shadow

box-shadow: *h-shadow v-shadow blur spread color* inset;

7、2D/3D转换：transform：translate rotate scale

1. 过渡：transition

*transition: property duration timing-function delay;*

1. 动画：keyframes、animation

*animation: name duration timing-function delay iteration-count direction;*

*@keyframes mymove*

*{*

*0% {top:0px;}*

*100% {top:200px;}*

*}*

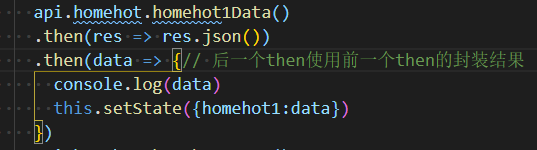
10：弹性盒子：flex

11：媒体查询：@media

Utils 工具类

React http-proxy-middleware前端解决跨域

Fetch().then().then()



Redux的使用，一顿操作猛如虎，没消化

**生命周期函数：**

组件将要挂载时触发的函数：componentWillMount

组件挂载完成时触发的函数：componentDidMount

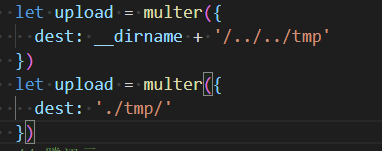
是否要更新数据时触发的函数：shouldComponentUpdate

将要更新数据时触发的函数：componentWillUpdate

数据更新完成时触发的函数：componentDidUpdate

组件将要销毁时触发的函数：componentWillUnmount

父组件中改变了props传值时触发的函数：componentWillReceiveProps



绝对地址





0916

Less设置公共样式文件，提升效率

颜色，flex

0917

# 面试题

什么是事件冒泡和事件捕获，区别是什么

##### **注意： addEventListener中有三个属性，第三个属性是布尔值。**

{默认属性是 false为事件冒泡，true 为事件捕获}

### **跨域**

协议，域名，端口，三者有一不一样，就是跨域

### **如何解决跨域**

目前有两种最常见的解决方案：

1. CORS，在服务器端设置几个响应头
2. Reverse Proxy，在 nginx/traefik/haproxy 等反向代理服务器中设置为同一域名

# **深入理解浏览器的缓存机制**

<https://www.jianshu.com/p/54cc04190252>

源码，啃不动，放着~

<https://github.com/DDFE/DDFE-blog/issues>

## **为什么使用 Virtual DOM**

但是模板引擎没有解决跟踪状态变化的问题，即当数据发生变化的时候，无法获取上一次的状态。只好把页面中的元素删除，然后重新创建。无法最小范围更新视图。

# **vue 中的动态传参和query传参**

params：/router1/:id，这里的 id 叫做 params。例如/router1/123, /router1/789

this.$router.push({ path:'/xxx' query:{ id:id } })

接收参数: this.$route.query.id

query：/router1?id=123，这里的 id 叫做 query。例如/router1?id=456

this.$router.push({ name:'xxx' params:{ id:id } })

接收参数: this.$route.params.id