**前端面试题总结**

1. **浏览器适配**

**移动端适配解决方案-rem**

**media query媒体查询的方式**

**刘海屏适配 iphoneX** viewport-fit=cover padding一定的距离

function (doc, win) {

var docEl = doc.documentElement,

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

recalc = function () {

var clientWidth = docEl.clientWidth;

if (!clientWidth) return;

if (clientWidth >= 750) {

docEl.style.fontSize = '100px';

} else {

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

}

};

if (!doc.addEventListener) return;

win.addEventListener(resizeEvt, recalc, false);

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

})(document, window);

1. **浏览器兼容性**

不同浏览器各自的margin和padding差异较大，CCS里:   \*{margin:0; padding:0}

几个img标签放在一起的时候，有些浏览器会有默认的间距，使用float属性为img布局

1. **前端综合题**

## 移动设备忽略将页面中的数字识别为电话号码的方法

<meta name="format-detection" content="telephone=no">

<meta name="format-detection" content="email=no">

<meta name="format-detection" content="adress=no">

<meta name="format-detection" content="telephone=no,email=no,adress=no">

1. **Css面试题**

## Css3的新特性

圆角border-radiuis，文字特效：text-shadow，旋转：transform，线性渐变： gradient

## Css的居中方式

display: flex;

justify-content: center;　　/\*使垂直居中\*/

align-items:center;　　　　/\*使水平居中\*/

.div1{

width:500px;

height:500px;

border:1px solid black;

position: relative; /\*很重要,不能忘\*/

}

.div2{

background: yellow;

width:300px;

height:200px;

margin:auto;

bottom: 0;

top:0;

left:0;

right:0;

position: absolute;

}

## 写一个幻灯片效果

<body>

<div class="banner">

</div>

</body>

.banner {

width: 480px;

height: 320px;

margin: 150px auto;

overflow: hidden;

background-size: cover;

background-position: center;

animation-name: "banner"; // 动画名称

animation-duration: 20s; // 播放动画所需时间

animation-timing-function: linear; // 播放动画的速度

animation-iteration-count: infinite; // 动画播放次数

}

@-webkit-keyframes 'banner' {

0% {background: url('images/1.jpg') no-repeat;}

25% {background: url('images/2.jpg') no-repeat;}

50% {background: url('images/3.jpg') no-repeat;}

75% {background: url('images/4.jpg') no-repeat;}

100% {background: url('images/5.jpg') no-repeat;}

}

## [CSS中伪类及伪元素用法详解](https://www.cnblogs.com/ghost-xyx/p/3763669.html)

：hover 鼠标放上去的样式

：active 元素被激活的样式

：first-child 第一个元素的样式

：before ：after 气泡框的三角指示

## 行内元素有哪些？块级元素有哪些？

行内元素有：span img input select strong

块级元素有：div ul ol dl dt dd h1 h2 h3 h4 p...

## 相邻的两个inline-block节点为什么会出现间隔，该如何解决

元素被当成行内元素排版的时候，原来HTML代码中的回车换行被转成一个空白符，在字体不为0的情况下，空白符占据一定宽度，所以inline-block的元素之间就出现了空隙。这些元素之间的间距会随着字体的大小而变化，当行内元素font-size:16px时，间距为8px。

解决方法一：给父级元素设置font-size： 0；子元素设置相应的font-size

解决方法二：改变书写方式，元素间留白间距出现的原因就是标签段之间的空格，因此，去掉HTML中的空格，自然间距就消失了

1. Js面试题

## ==和===的区别

1. ”==”表示：equality ->等同的意思，”==”使用两个等号时，如果两边值的类型不同的时候，是要先进行类型转换后，才能做比较

2. “===”表示：identity -> 恒等的意思，“===”使用三个等号时，是不需要做类型转换的，如果两边值的类型不同，就表示一定是不等的。

3. 简单说明使用三个等号(===)的判断规则:

(1) 如果类型不同，就一定不相等。

(2) 如果两个都是数值，并且是同一个值，那么相等；如果其中至少一个是NaN那么不相等。（判断一个值是否是NaN,只能使用isNaN()来判断）

(3) 如果两个都是字符串，每个位置的字符都一样，那么相等，否则不相等

## 什么是闭包? 闭包有什么作用?

答:

闭包是指有权访问另一个函数作用域中的变量的函数. 创建闭包常见方式,就是在一个函数内部创建另一个函数.

作用:

匿名自执行函数  (function (){ ... })();   创建了一个匿名的函数，并立即执行它，由于外部无法引用它内部的变量，因此在执行完后很快就会被释放，关键是这种机制不会污染全局对象。

缓存, 可保留函数内部的值

实现封装

实现模板

## 事件冒泡机制:

从目标元素开始，往顶层元素传播。途中如果有节点绑定了相应的事件处理函数，这些函数都会被一次触发。如果想阻止事件起泡，可以使用e.stopPropagation()（Firefox）或者e.cancelBubble=true（IE）来组织事件的冒泡传播。

## js有哪些基本类型?

答: Undefined, Null, Boolean, Number, String [BigInt](https://segmentfault.com/a/1190000019912017)、Symbol

　　　　Object是复杂数据类型, 其本质是由一组无序的名值对组成的.

引用类型则有: Object, Array, Date, RegExp, Function

## js实现数组去重怎么实现?

**方法1**. 创建一个新的临时数组来保存数组中已有的元素

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var a = new Array(1,2,2,2,2,5,3,2,9,5,6,3);  Array.prototype.unique1 = function(){      var n = [];     //一个新的临时数组      for(var i=0; i<this.length; i++){          //如果把当前数组的第i已经保存进了临时数组, 那么跳过          if(n.indexOf(this[i]) == -1){              n.push(this[i]);          }      }      return n;  }  console.log(a.unique1()); |

## 原型和原型链

原型到底是什么？原型是一个可以被复制（或者叫克隆）的一个类

原型链：

在javascript中，所有的对象都拥有一个\_\_proto\_\_属性指向该对象的原型（prototype)。

https://www.cnblogs.com/libin-1/p/5820550.html

调用一层一层查找指向

var Person = function(){};

Person.prototype.type = 'Person';

Person.prototype.maxAge = 100;

var p = new Person();

console.log(p.maxAge);

p.name = 'rainy';

Person.prototype.constructor === Person;  //=> true

p.\_\_proto\_\_ === Person.prototype;     //=> true

console.log(p.prototype);         //=> undefined

proto指向原始对象的prototype，new的对象的prototype为undefined

## 解释一下this指向

在全局环境下，this 始终指向全局对象（window）

### 对象中的this

对象内部方法的this指向调用这些方法的对象

#### 原型链中this

原型链中的方法的this仍然指向调用它的对象

### call & apply

当函数通过Function对象的原型中继承的方法 call() 和 apply() 方法调用时， 其函数内部的this值可绑定到 call() & apply() 方法指定的第一个对象上， 如果第一个参数不是对象，JavaScript内部会尝试将其转换成对象然后指向它。

Call和apply的不同在于传参的方式，call是，多个。Apply是一个数组

通过bind方法绑定后， 函数将被永远绑定在其第一个参数对象上， 而无论其在什么情况下被调用。

## 箭头函数中的 this

由于箭头函数不绑定this， 它会捕获其所在（即定义的位置）上下文的this值， 作为自己的this值，

https://www.cnblogs.com/dongcanliang/p/7054176.html

## 箭头函数需要注意的地方

\*当要求动态上下文的时候，就不能够使用箭头函数，也就是this的固定化。

1、在使用=>定义函数的时候，this的指向是定义时所在的对象，而不是使用时所在的对象；  
2、不能够用作构造函数，这就是说，不能够使用new命令，否则就会抛出一个错误；  
3、不能够使用arguments对象；

https://www.cnblogs.com/fengxiongZz/p/8191503.html

## **let和const**

let是更完美的var，不是全局变量，具有块级函数作用域,大多数情况不会发生变量提升。const定义常量值，不能够重新赋值，如果值是一个对象，可以改变对象里边的属性值。

1、let声明的变量具有块级作用域  
2、let声明的变量不能通过window.变量名进行访问

## Calss类

因为在ES6中，子类的构造函数必须含有super函数，super表示的是调用父类的构造函数，虽然是父类的构造函数，但是this指向的却是cat。

## Promise是异步编程的一种解决方案

三个状态 fulfill，reject，pending

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

if (操作成功) {

resolve(value)

} else {

reject(error)

}

})

promise.then(function (value) {

// success

}, function (value) {

// failure

})

Promise的实现过程，其主要使用了设计模式中的观察者模式

Promise.all，所有都执行完才执行

Promise.race其中一个执行完就开始执行

## Js数组方法改变原始数组

splice() 方法用于添加或删除数组中的元素。

pop() 方法用于删除数组的最后一个元素并返回删除的元素。

push() 方法可向数组的末尾添加一个或多个元素，并返回新的长度。

reverse() 方法用于颠倒数组中元素的顺序。

sort() 方法用于对数组的元素进行排序。

shift() 方法用于把数组的第一个元素从其中删除，并返回第一个元素的值。

unshift() 方法可向数组的开头添加一个或更多元素，并返回新的长度。

## Es6数组map方法

映射（一一对应）。[].map();基本用法跟forEach方法类似：

array.map(callback,[ thisObject]);

但是callback需要有return值（如果没有，就像会返回undefined）

1 var a1 = ['a', 'b', 'c'];

2 var a2 = a1.map(function(item) {

3 return item.toUpperCase();

4 });

5 console.log(a2); // logs A,B,C

## Es6数组去重

1、Array.from（）New Set(数组) var unique = (a) => [...new Set(a)]

2、function unique (arr) {

const seen = new Map()

return arr.filter((a) => !seen.has(a) && seen.set(a, 1))

}

## Es6不固定参数

使用方式：  前面加三个点 function(object, ...keys){}

注意事项： 1 只能有一个不定参数 2 只能使用在末尾 3 不能使用在setter中（setter只有一个参数

## Async/await

函数前面的async一词意味着一个简单的事情：这个函数总是返回一个promise，如果代码中有return <非promise>语句，JavaScript会自动把返回的这个value值包装成promise的resolved值。

关键词await可以让JavaScript进行等待，直到一个promise执行并返回它的结果，JavaScript才会继续往下执行。

## Js精度丢失问题

解决办法(0.1\*10 + 0.2\*10) / 10 == 0.3 // true

## 判断数据类型

Typeof判断基本类型和function类型

instanceof 只能用来判断两个对象是否属于实例关系， 而不能判断一个对象实例具体属于哪种类型。原理：var L = A.\_\_proto\_\_;var R = B.prototype;判断A的\_\_proto\_\_ 是否指向B的prototype

判断数据类型Object.prototype.toString.call

## js中异步问题的解决方案

主流的解决方法主要有以下几种：

1. 回调函数
2. 事件触发
3. 发布/订阅者模式
4. promise
5. generate

## generate函数

创建一个generate函数很简单

function\* gen(){

yield 1

yield 2

return 3

}

区别于普通函数的地方在于function后面的\*号，以及函数内部的yield。  
\*号是定义方式，带有 \* 号表示是一个generate函数，yield是其内部独特的语法。

function\* gen(){

yield 1

yield 2

return 3

}

let g=gen();

console.log(g.next())//{value:1,done:false}

console.log(g.next())//{value:2,done:false}

console.log(g.next())//{value:3,done:true}

console.log(g.next())//{value:undefined,done:true}

调用generate函数会生成一个遍历器对象，不会立即执行，需要调用next执行，执行到带有yield的那一步，next会返回一个对象，对象中value表示yield或return后的值，done表示函数是否已经执行结束（是否已经执行到return）。之后每次执行next都会从上一个yield开始继续执行

## Js对象的深拷贝和浅拷贝

**基本数据类型，拷贝是直接拷贝变量的值，而引用类型拷贝的其实是变量的地址**

而浅拷贝和深拷贝就是在这个基础之上做的区分，如果在拷贝这个对象的时候，只对基本数据类型进行了拷贝，而对引用数据类型只是进行了引用的传递，而没有重新创建一个新的对象，则认为是浅拷贝。反之，在对引用数据类型进行拷贝的时候，创建了一个新的对象，并且复制其内的成员变量，则认为是深拷贝

浅拷贝  
let o2 = Object.assign({},o1)

深拷贝

let o2 = JSON.parse(JSON.stringify(o1))

对象带function的操作

var s=JSON.stringify(json, function(key, val) {

if (typeof val === 'function') {

return val + '';

}

return val;

});

JSON.parse(s,function(k,v){

if(v.indexOf&&v.indexOf('function')>-1){

return eval("(function(){return "+v+" })()")

}

return v;

});

## 立即执行函数和使用场景

(function(){

//code

}())

立即执行函数的作用：

1、不必为函数命名，避免了污染全局变量

2、立即执行函数内部形成了一个单独的作用域，可以封装一些外部无法读取的私有变量

## JS函数柯里化

把接受多个参数的函数变换成接受一个单一参数（最初函数的第一个参数）的函数，并且返回接受余下的参数而且返回结果的新函数的技术。

好处：参数复用、延迟执行 bind方法的底层实现



## 怎么判断两个对象是否相等

//定义一个深度对比方法

function deepCompare(x, y) {

//声明所需变量

var i, l, leftChain, rightChain;

//具体对比函数

function compare2Objects(x, y) {

var p;

// 记得 NaN === NaN 返回 false

// isNaN(undefined) returns true

if (isNaN(x) && isNaN(y) && typeof x === 'number' && typeof y === 'number') {

return true;

}

// Compare primitives and functions.

// Check if both arguments link to the same object.

// Especially useful on the step where we compare prototypes

if (x === y) {

return true;

}

// Works in case when functions are created in constructor.

// Comparing dates is a common scenario. Another built-ins?

// We can even handle functions passed across iframes

if ((typeof x === 'function' && typeof y === 'function') ||

(x instanceof Date && y instanceof Date) ||

(x instanceof RegExp && y instanceof RegExp) ||

(x instanceof String && y instanceof String) ||

(x instanceof Number && y instanceof Number)) {

return x.toString() === y.toString();

}

// At last checking prototypes as good as we can

if (!(x instanceof Object && y instanceof Object)) {

return false;

}

if (x.isPrototypeOf(y) || y.isPrototypeOf(x)) {

return false;

}

if (x.constructor !== y.constructor) {

return false;

}

if (x.prototype !== y.prototype) {

return false;

}

// Check for infinitive linking loops

if (leftChain.indexOf(x) > -1 || rightChain.indexOf(y) > -1) {

return false;

}

// Quick checking of one object being a subset of another.

// todo: cache the structure of arguments[0] for performance

for (p in y) {

if (y.hasOwnProperty(p) !== x.hasOwnProperty(p)) {

return false;

} else if (typeof y[p] !== typeof x[p]) {

return false;

}

}

for (p in x) {

if (y.hasOwnProperty(p) !== x.hasOwnProperty(p)) {

return false;

} else if (typeof y[p] !== typeof x[p]) {

return false;

}

switch (typeof(x[p])) {

case 'object':

case 'function':

leftChain.push(x);

rightChain.push(y);

if (!compare2Objects(x[p], y[p])) {

return false;

}

leftChain.pop();

rightChain.pop();

break;

default:

if (x[p] !== y[p]) {

return false;

}

break;

}

}

return true;

}

if (arguments.length < 1) {

return true; //Die silently? Don't know how to handle such case, please help...

// throw "Need two or more arguments to compare";

}

for (i = 1, l = arguments.length; i < l; i++) {

leftChain = []; //Todo: this can be cached

rightChain = [];

if (!compare2Objects(arguments[0], arguments[i])) {

return false;

}

}

return true;

}

1. **Html面试题**

## [script标签中的async和defer](https://www.cnblogs.com/jiasm/p/7683930.html)

defer

如果script标签设置了该属性，则浏览器会异步的下载该文件并且不会影响到后续DOM的渲染；  
如果有多个设置了defer的script标签存在，则会按照顺序执行所有的script；  
defer脚本会在文档渲染完毕后，DOMContentLoaded事件调用前执行。

async

async的设置，会使得script脚本异步的加载并在允许的情况下执行  
async的执行，并不会按着script在页面中的顺序来执行，而是谁先加载完谁执行。

## 从前端角度出发谈谈做好seo需要考虑什么?

答:

* 语义化html标签
* 合理的title, description, keywords;
* 重要的html代码放前面
* 少用iframe, 搜索引擎不会抓取iframe中的内容
* 图片加上alt

## 请描述一下 cookies，sessionStorage 和 localStorage 的区别？

答:

* **cookie:**
  + cookie是网站为了**标示用户身份**而储存在用户本地终端（Client Side）上的数据（通常经过加密）。
  + cookie数据**始终在同源的http请求中携带**（即使不需要），记会在浏览器和服务器间来回传递。
* **sessionStorage和localStorage**不会自动把数据发给服务器，仅在本地保存。
* **存储大小：**
  + cookie数据大小不能超过4k。
  + 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。
* **有期时间**：
  + localStorage    存储**持久数据**，浏览器关闭后数据不丢失除非主动删除数据；
  + sessionStorage  数据在当前浏览器窗口关闭后自动删除。
  + cookie          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。

## src和href的区别?

答:

* src指向外部资源的位置, 用于替换当前元素, 比如js脚本, 图片等元素
* href指向网络资源所在的位置, 用于在当前文档和引用资源间确定联系, 加载css

## 说一下什么是Http协议？

对器客户端和 服务器端之间数据传输的格式规范，格式简称为“超文本传输协议”。

## 什么是Http协议无状态协议？怎么解决Http协议无状态协议？

无状态协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息

* 无状态协议解决办法： 通过1、Cookie 2、通过Session会话保存。

## Http状态码

* 200("OK")  
  一切正常。实体主体中的文档（若存在的话）是某资源的表示。
* 500("Bad Request")  
  客户端方面的问题。实体主题中的文档（若存在的话）是一个错误消息。希望客户端能够理解此错误消息，并改正问题。
* 500("Internal Server Error")  
  服务期方面的问题。实体主体中的文档（如果存在的话）是一个错误消息。该错误消息通常无济于事，因为客户端无法修复服务器方面的问题。
* 503 服务器维护
* 301重定向。永久移动
* 302重定向。临时移动

这种情况下，服务器返回的头部信息中会包含一个 Location 字段，内容是重定向到的url。

400 客户端请求的语法错误，服务器无法理解

401 请求要求用户的身份认证

403 服务器理解请求客户端的请求，但是拒绝执行此请求

* 404("Not Found") 和410("Gone")  
  当客户端所请求的URI不对应于任何资源时，发送此响应代码。404用于服务器端不知道客户端要请求哪个资源的情况；410用于服务器端知道客户端所请求的资源曾经存在，但现在已经不存在了的情况。

## Http协议有什么组成？

  请求报文包含三部分：

* 请求行：包含请求方法、URI、HTTP版本信息
* 请求首部字段
* 请求内容实体

  响应报文包含三部分：

* 状态行：包含HTTP版本、状态码、状态码的原因短语
* 响应首部字段
* 响应内容实体

## Http协议首部字段？

a、通用首部字段（请求报文与响应报文都会使用的首部字段）

* Date：创建报文时间
* Connection：连接的管理
* Cache-Control：缓存的控制
* Transfer-Encoding：报文主体的传输编码方式

b、请求首部字段（请求报文会使用的首部字段）

* Host：请求资源所在服务器
* Accept：可处理的媒体类型
* Accept-Charset：可接收的字符集
* Accept-Encoding：可接受的内容编码
* Accept-Language：可接受的自然语言

c、响应首部字段（响应报文会使用的首部字段）

* Accept-Ranges：可接受的字节范围
* Location：令客户端重新定向到的URI
* Server：HTTP服务器的安装信息

d、实体首部字段（请求报文与响应报文的的实体部分使用的首部字段）

* Allow：资源可支持的HTTP方法
* Content-Type：实体主类的类型
* Content-Encoding：实体主体适用的编码方式
* Content-Language：实体主体的自然语言
* Content-Length：实体主体的的字节数
* Content-Range：实体主体的位置范围，一般用于发出部分请求时使用

## Http协议中Http1.0与1.1区别？

* 在http1.0中，当建立连接后，客户端发送一个请求，服务器端返回一个信息后就关闭连接，当浏览器下次请求的时候又要建立连接，显然这种不断建立连接的方式，会造成很多问题。
* 在http1.1中，引入了持续连接的概念，通过这种连接，浏览器可以建立一个连接之后，发送请求并得到返回信息，然后继续发送请求再次等到返回信息，也就是说客户端可以连续发送多个请求，而不用等待每一个响应的到来。

## Http协议中有那些请求方式？

* GET： 用于请求访问已经被URI（统一资源标识符）识别的资源，可以通过URL传参给服务器
* POST：用于传输信息给服务器，主要功能与GET方法类似，但一般推荐使用POST方式。
* PUT： 传输文件，报文主体中包含文件内容，保存到对应URI位置。
* HEAD： 获得报文首部，与GET方法类似，只是不返回报文主体，一般用于验证URI是否有效。
* DELETE：删除文件，与PUT方法相反，删除对应URI位置的文件。
* OPTIONS：查询相应URI支持的HTTP方法。

## Http与Https优缺点？

* 通信使用明文不加密，内容可能被窃听，也就是被抓包分析。
* 不验证通信方身份，可能遭到伪装
* 无法验证报文完整性，可能被篡改
* HTTPS就是HTTP加上加密处理（一般是SSL安全通信线路）+认证+完整性保护

## Http优化

* 利用负载均衡优化和加速HTTP应用
* 利用HTTP Cache来优化网站

## URL访问网站的网络传输全过程

整个过程基本分做下面几个部分：

* 1、域名解析成IP地址；
* 2、与目的主机进行TCP连接（三次握手）；
* 3、发送与收取数据；
* 4、与目的主机断开TCP连接（四次挥手）；

具体的三次握手建立连接的过程如下表述，其中数据包的传输过程类似上文请求DNS服务器时的过程，就简单的表示一下：

**第一次握手：**主机A发送位码为syn＝1,随机产生seq number=x的数据包到服务器，客户端进入SYN\_SEND状态，等待服务器的确认；主机B由SYN=1知道，A要求建立联机；

**第二次握手：**主机B收到请求后要确认联机信息，向A发送ack number(主机A的seq+1),syn=1,ack=1,随机产生seq=y的包,此时服务器进入SYN\_RECV状态;

**第三次握手：**主机A收到后检查ack number是否正确，即第一次发送的seq number+1,以及位码ack是否为1，若正确，主机A会再发送ack number(主机B的seq+1),ack=1，主机B收到后确认seq值与ack=1则连接建立成功。客户端和服务器端都进入ESTABLISHED状态，完成TCP三次握手。

TCP位码,有6种标示:SYN(synchronous建立联机) ACK(acknowledgement 确认) PSH(push传送) FIN(finish结束) RST(reset重置) URG(urgent紧急)

Sequence number(顺序号码) Acknowledge number(确认号码)

三次握手的过程就是一去一回一去，互相确认一下，就建立连接啦。这个过程中任何一个报文出错或者超时，都要进行重传。

如果Connection:Keep-Alive就表示持续连接，多个HTTP请求都是这一个网络连接，如果是Connection:close，在下一次HTTP请求时就会有另外一次HTTP连接，也就会再有一个三次握手的过程。

1. **Vue面试题**

## Vue的生命周期

**beforeCreate**（创建前） 在数据观测和初始化事件还未开始  
**created**（创建后） 完成数据观测，属性和方法的运算，初始化事件，$el属性还没有显示出来  
**beforeMount**（载入前） 在挂载开始之前被调用，相关的render函数首次被调用。实例已完成以下的配置：编译模板，把data里面的数据和模板生成html。注意此时还没有挂载html到页面上。  
**mounted**（载入后） 在el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用。实例已完成以下的配置：用上面编译好的html内容替换el属性指向的DOM对象。完成模板中的html渲染到html页面中。此过程中进行ajax交互。  
**beforeUpdate**（更新前） 在数据更新之前调用，发生在虚拟DOM重新渲染和打补丁之前。可以在该钩子中进一步地更改状态，不会触发附加的重渲染过程。  
**updated**（更新后） 在由于数据更改导致的虚拟DOM重新渲染和打补丁之后调用。调用时，组件DOM已经更新，所以可以执行依赖于DOM的操作。然而在大多数情况下，应该避免在此期间更改状态，因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不被调用。  
**beforeDestroy**（销毁前） 在实例销毁之前调用。实例仍然完全可用。  
**destroyed**（销毁后） 在实例销毁之后调用。调用后，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。  
1.什么是vue生命周期？  
答： Vue 实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。

**2.vue生命周期的作用是什么？**  
答：它的生命周期中有多个事件钩子，让我们在控制整个Vue实例的过程时更容易形成好的逻辑。

**3.vue生命周期总共有几个阶段？**  
答：它可以总共分为8个阶段：创建前/后, 载入前/后,更新前/后,销毁前/销毁后。

**4.第一次页面加载会触发哪几个钩子？**  
答：会触发 下面这几个beforeCreate, created, beforeMount, mounted 。

**5.DOM 渲染在 哪个周期中就已经完成？**  
答：DOM 渲染在 mounted 中就已经完成了。

## Vue实现数据双向绑定的原理：

Object.defineProperty（）

当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty 将它们转为 getter/setter。

vue的数据双向绑定 将MVVM作为数据绑定的入口，整合Observer，Compile和Watcher三者，通过Observer来监听自己的model的数据变化，通过Compile来解析编译模板指令（vue中是用来解析 {{}}），最终利用watcher搭起observer和Compile之间的通信桥梁，达到数据变化

## js实现简单的双向绑定

<body>

<div id="app">

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

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

</div>

</body>

<script type="text/javascript">

var obj = {}

Object.defineProperty(obj, 'txt', {

get: function () {

return obj

},

set: function (newValue) {

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

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

}

})

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

obj.txt = e.target.value

})

</script>

**1.父组件与子组件传值**  
父组件传给子组件：子组件通过props方法接受数据;  
子组件传给父组件：$emit方法传递参数  
**2.非父子组件间的数据传递，兄弟组件传值**

Vuex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

**state**  
Vuex 使用单一状态树,即每个应用将仅仅包含一个store 实例，但单一状态树和模块化并不冲突。存放的数据状态，不可以直接修改里面的数据。  
**mutations**  
mutations定义的方法动态修改Vuex 的 store 中的状态或数据。  
**getters**  
类似vue的计算属性，主要用来过滤一些数据。  
**action**   
actions可以理解为通过将mutations里面处里数据的方法变成可异步的处理数据的方法，简单的说就是异步操作数据。view 层通过 store.dispath 来分发 action。

## vue-cli如何新增自定义指令？

1.创建局部指令

var app = new Vue({

el: '#app',

data: {

},

// 创建指令(可以多个)

directives: {

// 指令名称

dir1: {

inserted(el) {

// 指令中第一个参数是当前使用指令的DOM

console.log(el);

console.log(arguments);

// 对DOM进行操作

el.style.width = '200px';

el.style.height = '200px';

el.style.background = '#000';

}

}

}

})

<div id="app">

<div v-dir1></div>

<div v-dir2></div>

</div>

## 2.v-if 和 v-show 区别 答：v-if按照条件是否渲染，v-show是display的block或none；

## 3.$route和$router的区别 答：$route是“路由信息对象”，包括path，params，hash，query，fullPath，matched，name等路由信息参数。而$router是“路由实例”对象包括了路由的跳转方法，钩子函数等。

## 4.vue.js的两个核心是什么？ 答：数据驱动、组件系统

**5.vue几种常用的指令**  
答：v-for 、 v-if 、v-bind、v-on、v-show、v-else

**10.vue等单页面应用及其优缺点**  
答：优点：Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件，核心是一个响应的数据绑定系统。MVVM、数据驱动、组件化、轻量、简洁、高效、快速、模块友好。  
缺点：不支持低版本的浏览器，最低只支持到IE9；不利于SEO的优化（如果要支持SEO，建议通过服务端来进行渲染组件）；第一次加载首页耗时相对长一些；不可以使用浏览器的导航按钮需要自行实现前进、后退。

## 过度动画

transition 的封装组件，name="fade"样式名称

enter和leave

:duration="1000"过度持续时间

1. **webpack**

## 如何利用webpack来优化前端性能？（提高性能和体验）

用webpack优化前端性能是指优化webpack的输出结果，让打包的最终结果在浏览器运行快速高效。

* 压缩代码。删除多余的代码、注释、简化代码的写法等等方式。可以利用webpack的UglifyJsPlugin和ParallelUglifyPlugin来压缩JS文件， 利用cssnano（css-loader?minimize）来压缩css
* 利用CDN加速。在构建过程中，将引用的静态资源路径修改为CDN上对应的路径。可以利用webpack对于output参数和各loader的publicPath参数来修改资源路径
* 删除死代码（Tree Shaking）。将代码中永远不会走到的片段删除掉。可以通过在启动webpack时追加参数--optimize-minimize来实现
* 提取公共代码。

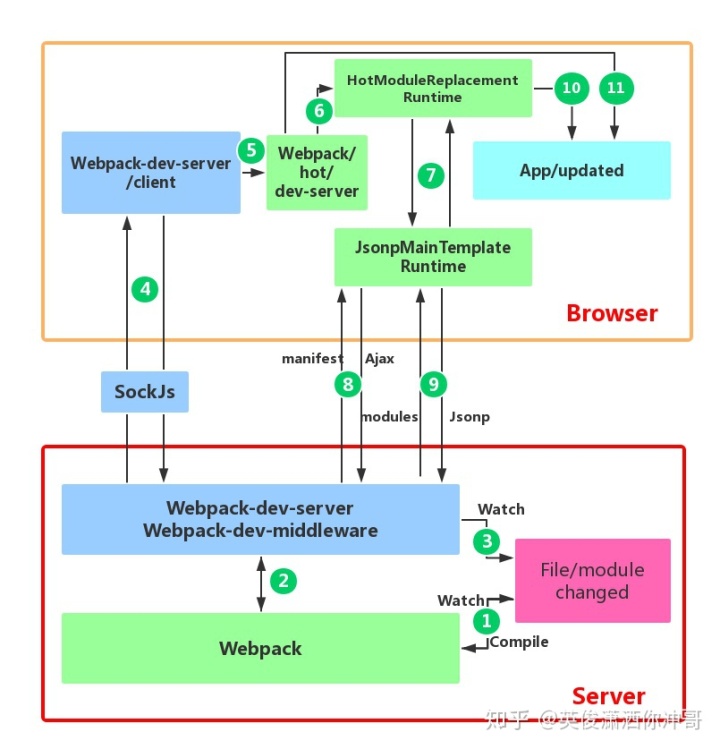
## 如何提高webpack的构建速度？

1. 多入口情况下，使用CommonsChunkPlugin来提取公共代码
2. 通过externals配置来提取常用库
3. 利用DllPlugin和DllReferencePlugin预编译资源模块 通过DllPlugin来对那些我们引用但是绝对不会修改的npm包来进行预编译，再通过DllReferencePlugin将预编译的模块加载进来。
4. 使用Happypack 实现多线程加速编译
5. 使用webpack-uglify-parallel来提升uglifyPlugin的压缩速度。 原理上webpack-uglify-parallel采用了多核并行压缩来提升压缩速度
6. 使用Tree-shaking和Scope Hoisting来剔除多余代码

## webpack的热更新是如何做到的？说明其原理？

webpack的热更新又称热替换（Hot Module Replacement），缩写为HMR。 这个机制可以做到不用刷新浏览器而将新变更的模块替换掉旧的模块。

**原理：**

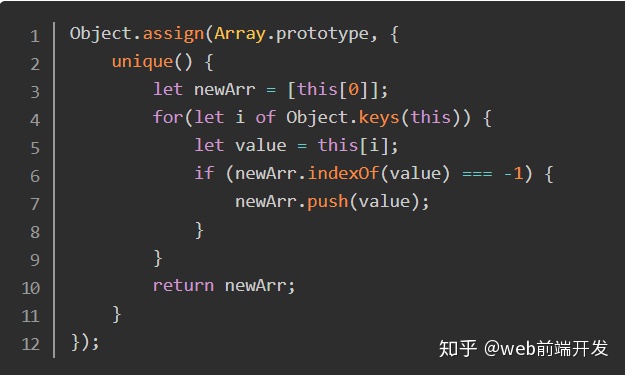
图片来自饿了么前端@知乎专栏

首先要知道server端和client端都做了处理工作

1. 第一步，在 webpack 的 watch 模式下，文件系统中某一个文件发生修改，webpack 监听到文件变化，根据配置文件对模块重新编译打包，并将打包后的代码通过简单的 JavaScript 对象保存在内存中。
2. 第二步是 webpack-dev-server 和 webpack 之间的接口交互，而在这一步，主要是 dev-server 的中间件 webpack-dev-middleware 和 webpack 之间的交互，webpack-dev-middleware 调用 webpack 暴露的 API对代码变化进行监控，并且告诉 webpack，将代码打包到内存中。
3. 第三步是 webpack-dev-server 对文件变化的一个监控，这一步不同于第一步，并不是监控代码变化重新打包。当我们在配置文件中配置了devServer.watchContentBase 为 true 的时候，Server 会监听这些配置文件夹中静态文件的变化，变化后会通知浏览器端对应用进行 live reload。注意，这儿是浏览器刷新，和 HMR 是两个概念。
4. 第四步也是 webpack-dev-server 代码的工作，该步骤主要是通过 sockjs（webpack-dev-server 的依赖）在浏览器端和服务端之间建立一个 websocket 长连接，将 webpack 编译打包的各个阶段的状态信息告知浏览器端，同时也包括第三步中 Server 监听静态文件变化的信息。浏览器端根据这些 socket 消息进行不同的操作。当然服务端传递的最主要信息还是新模块的 hash 值，后面的步骤根据这一 hash 值来进行模块热替换。
5. webpack-dev-server/client 端并不能够请求更新的代码，也不会执行热更模块操作，而把这些工作又交回给了 webpack，webpack/hot/dev-server 的工作就是根据 webpack-dev-server/client 传给它的信息以及 dev-server 的配置决定是刷新浏览器呢还是进行模块热更新。当然如果仅仅是刷新浏览器，也就没有后面那些步骤了。
6. HotModuleReplacement.runtime 是客户端 HMR 的中枢，它接收到上一步传递给他的新模块的 hash 值，它通过 JsonpMainTemplate.runtime 向 server 端发送 Ajax 请求，服务端返回一个 json，该 json 包含了所有要更新的模块的 hash 值，获取到更新列表后，该模块再次通过 jsonp 请求，获取到最新的模块代码。这就是上图中 7、8、9 步骤。
7. 而第 10 步是决定 HMR 成功与否的关键步骤，在该步骤中，HotModulePlugin 将会对新旧模块进行对比，决定是否更新模块，在决定更新模块后，检查模块之间的依赖关系，更新模块的同时更新模块间的依赖引用。
8. 最后一步，当 HMR 失败后，回退到 live reload 操作，也就是进行浏览器刷新来获取最新打包代码。
9. **前端常见算法题**

## Q1. 去掉一组整型数组中重复的值

该题目的解法有好几种，下面列举几种常见的解法。



上面那种解法是我们最容易想到的解法，在查找重复元素的过程中使用了indexOf方法，如果不使用的话，也可以使用循环解决如下：



除了使用indexOf和循环去重，我们还可以使用Object对象判断是否重复，下面这种解法是作者比较推荐的一种解法。

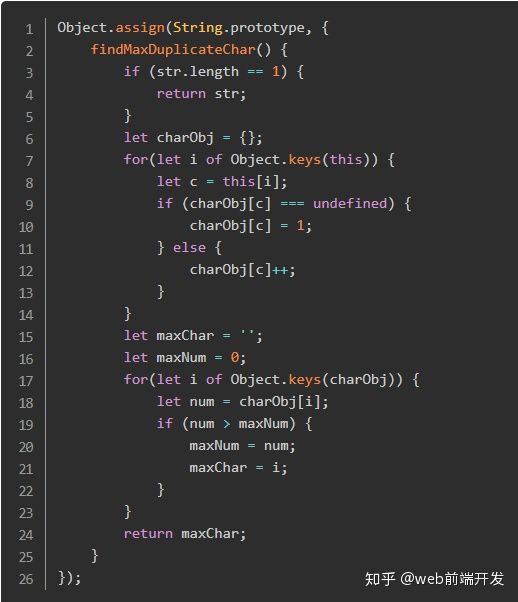


最后还有一种方法，使用了sort（）对数组进行了排序，这样去重之后原数组的索引对应元素的值都发生的变化，如果不考虑这一点，可以使用这种解决方案。



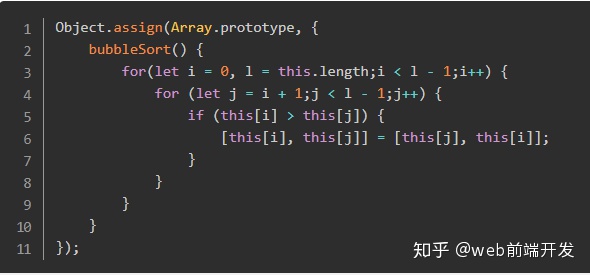
## Q2. 统计一个字符串出现最多的字母

其实就是统计字符串中每个字符出现的次数，最后从中找到出现次数最多的字符。

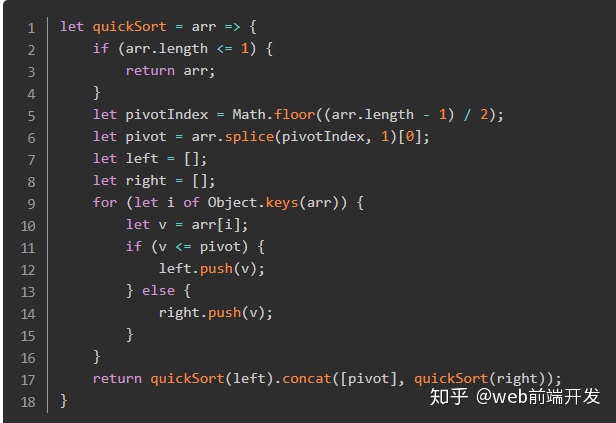


## Q3. 排序算法

冒泡排序



快速排序

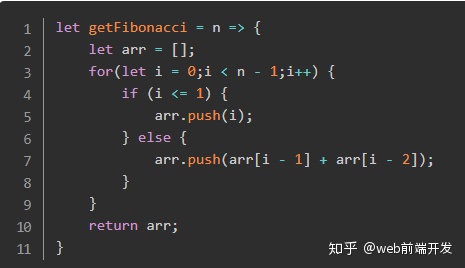


插入排序



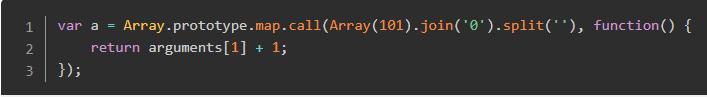
## Q4. 斐波那契数列

斐波那契数列就是形如0,1,1,2,3,5,8.....的数列。

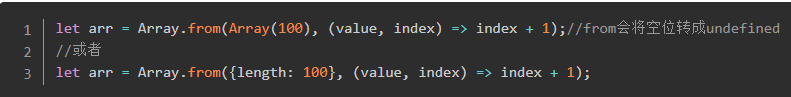


## Q5. 速度创建1-100的数组

ES5的写法

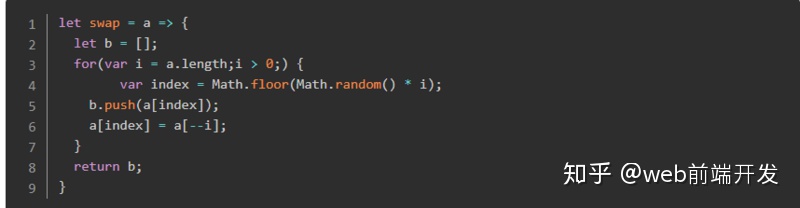


ES6的写法



## Q6. 洗牌算法

这道题是我去富途面试CTO问我的一道题，我听到之后一脸懵比（后面的题目也做的不好），结果直接game over，面试之后查阅了下百度自己写了出来



1. **typescript**

属于 JavaScript 的超集，可以编译为 JavaScript 执行。 它的最大特点就是支持强类型和 [ES6 Class](http://es6.ruanyifeng.com/#docs/class)。

<https://ts.xcatliu.com/introduction/what-is-typescript.html>

定义类型：加类型，任意类型any，字符串数组String[],联合类型string | number;

元组 ：多个类型的数据

可以枚举类，接口和对象的继承

1. **React面试题**

教程https://hulufei.gitbooks.io/react-tutorial/content/introduction.html

React 的核心思想是：封装组件。

React 发明了 JSX 让 JS 支持嵌入 HTML

凡是使用 JSX 的地方，<script> 标签都要加上 type="text/babel"

## 哪个生命周期可以调用ajax

可以使用 componentDidMount 方法设置 Ajax 请求

## Virtual DOM

当组件状态 state 有更改的时候，React 会自动调用组件的 render 方法重新渲染整个组件的 UI。

当然如果真的这样大面积的操作 DOM，性能会是一个很大的问题，所以 React 实现了一个*Virtual DOM*，组件 DOM 结构就是映射到这个 Virtual DOM 上，React 在这个 Virtual DOM 上实现了一个 diff 算法，当要重新渲染组件的时候，会通过 diff 寻找到要变更的 DOM 节点，再把这个修改更新到浏览器实际的 DOM 节点上，所以实际上不是真的渲染整个 DOM 树。这个 Virtual DOM 是一个纯粹的 JS 数据结构，所以性能会比原生 DOM 快很多。

## Data Flow

“单向数据绑定”是 React 推崇的一种应用架构的方式。

组件

props 是组件包含的两个核心概念之一，另一个是 state，一旦状态（数据）更改，组件就会自动调用 render 重新渲染 UI，这个更改的动作会通过 this.setState方法来触发。

无状态组件没有状态，没有生命周期，只是函数，所以它没有实例返回，例如箭头函数定义。

## 组件生命周期

一般来说，一个组件类由 extends Component 创建，并且提供一个 render 方法以及其他可选的生命周期函数、组件相关的事件或方法来定义。

getInitialState初始化 this.state 的值，设置默认属性getDefaultProps

### 装载组件触发

componentWillMount

只会在装载之前调用一次，在 render 之前调用，你可以在这个方法里面调用 setState 改变状态，并且不会导致额外调用一次 render

componentDidMount

只会在装载完成之后调用一次，在 render 之后调用，从这里开始可以通过 ReactDOM.findDOMNode(this)获取到组件的 DOM 节点。

### 更新组件触发

这些方法不会在首次 render 组件的周期调用

* componentWillReceiveProps
* shouldComponentUpdate
* componentWillUpdate
* componentDidUpdate

### 卸载组件触发

* componentWillUnmount

### 组件间通信

父子组件就是通过 props 属性传递

非父子组件通信

## Flux

一个 Flux 应用主要包含四个部分：

* the dispatcher

*处理动作分发，维护 Store 之间的依赖关系*

* the stores

*数据和逻辑部分*

* the views

*React 组件，这一层可以看作 controller-views，作为视图同时响应用户交互*

* the actions

*提供给 dispatcher 传递数据给 store*

整个流程如下：

* 首先要有 action，通过定义一些 action creator 方法根据需要创建 Action 提供给 dispatcher
* View 层通过用户交互（比如 onClick）会触发 Action
* Dispatcher 会分发触发的 Action 给所有注册的 Store 的回调函数
* Store 回调函数根据接收的 Action 更新自身数据之后会触发一个 change 事件通知 View 数据更改了
* View 会监听这个 change 事件，拿到对应的新数据并调用 setState 更新组件 UI

## Redux 的基础概念

### 三个基本原则

* 整个应用只有唯一一个可信数据源，也就是只有一个 Store
* State 只能通过触发 Action 来更改
* State 的更改必须写成纯函数，也就是每次更改总是返回一个新的 State，在 Redux 里这种函数称为 Reducer

### Redux-Saga

同步没有副作用UI——>action(plain)——>reducer——>state——>UI

异步有副作用UI——>action(side function)——>middleware——>action(plain)——>reducer——>state——>UI

Redux-Saga就是处理的副作用的middleware