大

学

全

栈

面 试 题 库

V120191220

目录

[1. 你做的页面在哪些流览器测试过？这些浏览器的内核分别是什么? 5](#_Toc27635538)

[2. 你简述盒模型 5](#_Toc27635539)

[3. CSS3的新特性 5](#_Toc27635540)

[4. CSS选择器及其优先级 6](#_Toc27635541)

[5. 说说BFC 6](#_Toc27635542)

[6. 说说定位相关的属性 7](#_Toc27635543)

[7. 谈谈flex布局 8](#_Toc27635544)

[8. 你怎么清楚浮动 10](#_Toc27635545)

[9. 两边宽度固定中间自适应的三栏布局(默写题,手写笔试题) 11](#_Toc27635546)

[10. 浏览器渲染机制 13](#_Toc27635547)

[11. 重绘和回流的区别 13](#_Toc27635548)

[12. JS数据类型 15](#_Toc27635549)

[13. JS判断数据类型的方法 15](#_Toc27635550)

[14. null和undefined区别 16](#_Toc27635551)

[15. JS作用域的理解 16](#_Toc27635552)

[16. call,apply和bind区别的页面在哪些流览器测 16](#_Toc27635553)

[17. 深拷贝和浅拷贝 16](#_Toc27635554)

[18. 谈谈防抖和节流 18](#_Toc27635555)

[19. 谈谈cookie,sessionStorage和localStorage 19](#_Toc27635556)

[20. 0.1+0.2!=0.3怎么处理 19](#_Toc27635557)

[21. 数组的常用方法(至少说出8个) 20](#_Toc27635558)

[22. new一个对象的过程中发生了什么吗? 24](#_Toc27635559)

[23. JS实现继承(至少会一种) 24](#_Toc27635560)

[24. get和post区别 26](#_Toc27635561)

[25. JSONP原理 26](#_Toc27635562)

[26. 缓存的理解 27](#_Toc27635563)

[27. XSS和CSRF区别 27](#_Toc27635564)

[28. HTTP与HTTPS的区别 27](#_Toc27635565)

[29. HTTP状态码 28](#_Toc27635566)

[30. 性能优化 28](#_Toc27635567)

[31. 性能优化 29](#_Toc27635568)

[32. mvc和mvvm理解 31](#_Toc27635569)

[33. vue2双向绑定原理(最简洁版) 32](#_Toc27635570)

[34. v-model双向绑定原理 32](#_Toc27635571)

[35. vue生命周期函数 33](#_Toc27635572)

[36. 组件data为什么返回函数 34](#_Toc27635573)

[37. vue给对象新增属性页面没有响应 34](#_Toc27635574)

[38. v-if和v-show区别 34](#_Toc27635575)

[39. vue中scoped属性作用 35](#_Toc27635576)

[40. vue中ref的作用 35](#_Toc27635577)

[41. vue中ref的作用 35](#_Toc27635578)

[42. 计算属性 computed 和事件 methods 有什么区别 35](#_Toc27635579)

[43. vue中computed和watch区别 35](#_Toc27635580)

[44. vue中父子组件传值 35](#_Toc27635581)

[45. axios拦截器怎么配(默写题) 36](#_Toc27635582)

[46. vue-router 有哪几种导航钩子? 36](#_Toc27635583)

[47. 谈谈vuex 37](#_Toc27635584)

[48. vue-router中$route 和 $router 的区别 38](#_Toc27635585)

[49. nextTick 是做什么的 38](#_Toc27635586)

[50. 你有对 Vue 项目进行哪些优化？ 38](#_Toc27635587)

HTML

CSS

HTML5

CSS3

## 你做的页面在哪些流览器测试过？这些浏览器的内核分别是什么?

IE: trident内核

Firefox：gecko内核

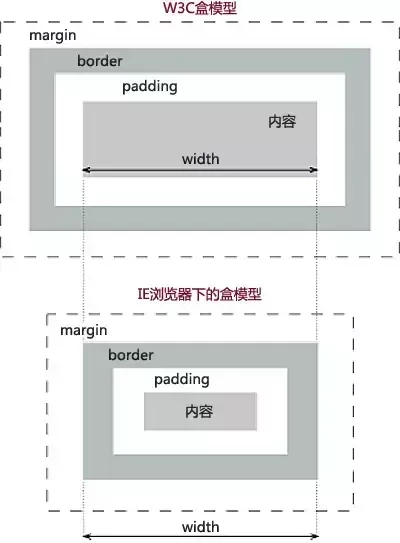
Safari:webkit内核

Opera:以前是presto内核，Opera现已改用Google Chrome的Blink内核

Chrome:基于webkit

只要能分清浏览器使用什么内核就得满分（safari和chrome使用webkit内核，Firefox使用gecko内核）

## 你简述盒模型

CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：边距margin，边框border，填充padding，和实际内容content。盒模型允许我们在其它元素和周围元素边框之间的空间放置元素。

box-sizing:content-box（W3C盒模型，又名标准盒模型）：元素的宽高大小表现为内容的大小。 box-sizing:border-box（IE盒模型，又名怪异盒模型）：元素的宽高表现为内容 + 内边距 + 边框的大小。背景会延伸到边框的外沿。

## CSS3的新特性

word-wrap 文字换行

text-overflow 超过指定容器的边界时如何显示

text-decoration 文字渲染

text-shadow文字阴影

gradient渐变效果

transition过渡效果 transition-duration：过渡的持续时间

transform拉伸，压缩，旋转，偏移等变换

animation动画

transition和animation的区别：

  Animation和transition大部分属性是相同的，他们都是随时间改变元素的属性值，他们的主要区别是transition需要触发一个事件才能改变属性，而animation不需要触发任何事件的情况下才会随时间改变属性值，并且transition为2帧，从from .... to，而animation可以一帧一帧的。

## CSS选择器及其优先级

!important

内联样式style=""

ID选择器#id

类选择器/属性选择器/伪类选择器.class.active[href=""]

元素选择器/关系选择器/伪元素选择器html+div>span::after

通配符选择器\*

## 说说BFC

BFC（Block Formatting Context）格式化上下文，是Web页面中盒模型布局的CSS渲染模式，指一个独立的渲染区域或者说是一个隔离的独立容器。

BFC应用

防止margin重叠

清除内部浮动

自适应两（多）栏布局

防止字体环绕

触发BFC条件

根元素

float的值不为none

overflow的值不为visible

display的值为inline-block、table-cell、table-caption

position的值为absolute、fixed

BFC的特性

内部的Box会在垂直方向上一个接一个的放置。

垂直方向上的距离由margin决定

bfc的区域不会与float的元素区域重叠。

计算bfc的高度时，浮动元素也参与计算

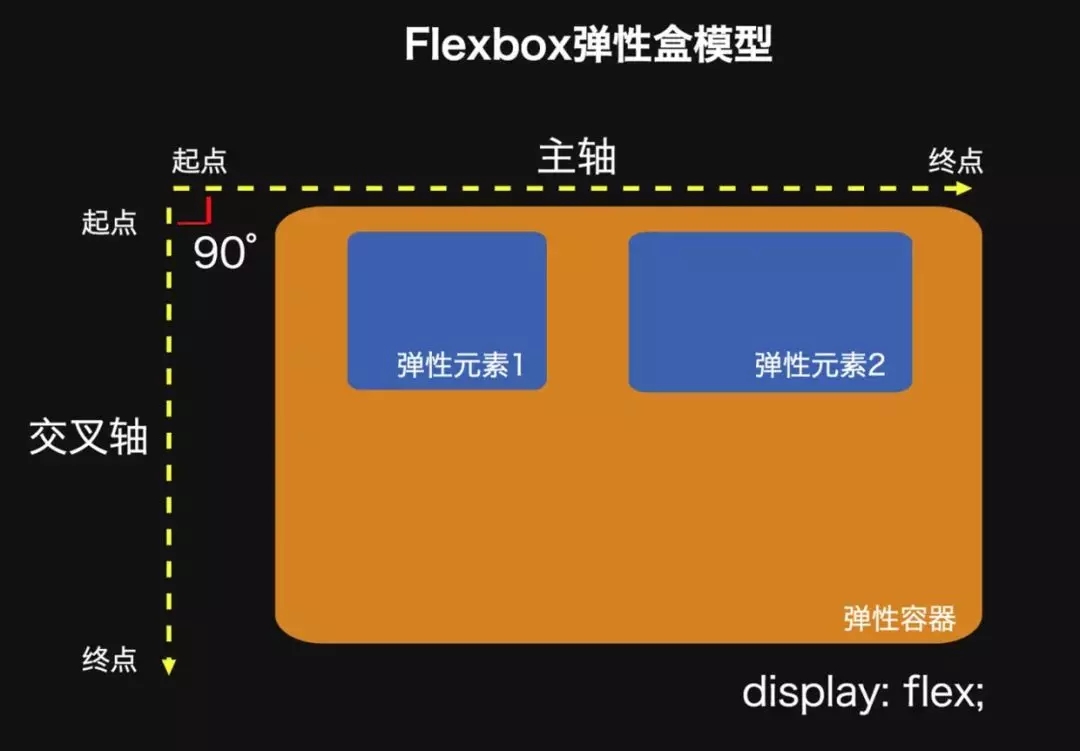
bfc就是页面上的一个独立容器，容器里面的子元素不会影响外面元素。

## 说说定位相关的属性

position 属性的五个值：

|  |  |
| --- | --- |
| **值** | **描述** |
| static | 默认。位置设置为 static 的元素，  它始终会处于页面流给予的位置（static 元素会忽略任何 top、bottom、left 或 right 声明）。HTML 元素的默认值，即没有定位，遵循正常的文档流对象。从上到下,从左到右  静态定位的元素不会受到 top, bottom, left, right影响。 |
| absolute | 位置设置为 absolute 的元素，可定位于相对于第一个已定位（非静态的）的包含它的元素的指定坐标。  绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>  此元素的位置可通过 "left"、"top"、"right" 以及 "bottom" 属性来规定。  absolute 定位使元素的位置与文档流无关，因此不占据空间。  absolute 定位的元素和其他元素重叠。  会将原来的块元素变成行内块元素 |
| fixed | 位置被设置为 fixed 的元素，可定位于相对于浏览器窗口的指定坐标。元素的位置相对于浏览器窗口是固定位置。即使窗口是滚动的它也不会移动  此元素的位置可通过 "left"、"top"、"right" 以及 "bottom" 属性来规定。  不论窗口滚动与否，元素都会留在那个位置。工作于 IE7（strict 模式）。  会将原来的块元素变成行内块元素 |
| relative | 位置被设置为 relative 的元素，可将其定位于相对于其正常位置的地方，  因此 "left:20" 会将元素移至元素正常位置左边 20 个像素的位置。  相对定位元素的定位是相对其自身正常位置。  移动相对定位元素，但它原本所占的空间不会改变。 |
| sticky | sticky 英文字面意思是粘，粘贴，所以可以把它称之为粘性定位。  position: sticky; 基于用户的滚动位置来定位。  粘性定位的元素是依赖于用户的滚动，在 position:relative 与 position:fixed 定位之间切换。  它的行为就像 position:relative; 而当页面滚动超出目标区域时，它的表现就像 position:fixed;，它会固定在目标位置。  元素定位表现为在跨越特定阈值前为相对定位，之后为固定定位。  这个特定阈值指的是 top, right, bottom 或 left 之一，换言之，指定 top, right, bottom 或 left 四个阈值其中之一，才可使粘性定位生效。否则其行为与相对定位相同。 |
| inherit | position 属性的值从父元素继承。 |

## 谈谈flex布局

采用 Flex 布局的元素，称为 Flex容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为 Flex项目（flex item），简称“项目”。

1、父元素属性

| **属性名** | **属性值** | **备注** |
| --- | --- | --- |
| display | flex | 定义了一个flex容器，它的直接子元素会接受这个flex环境 |
| flex-direction | row,  row-reverse,  column,  column-reverse | 决定主轴的方向 |
| flex-wrap | nowrap,  wrap,  wrap-reverse | 如果一条轴线排不下，如何换行 |
| flex-flow | [flex-direction] , [flex-wrap] | 是 flex-direction属性和  flex-wrap属性的简写形式，  默认值为 row nowrap |
| justify-content | flex-start,  flex-end,  center,  space-between,  space-around | 设置或检索弹性盒子元素在主轴（横轴）方向上的对齐方式 |
| align-items | flex-start,  flex-end,  center,  baseline,  stretch | 设置或检索弹性盒子元素在侧轴（纵轴）方向上的对齐方式 |
|  |  |  |
| 子元素属性 | | |
|  |  |  |
| 属性名 | 属性值 | 备注 |
| order | [int] | 默认情况下flex order会按照书写顺训呈现，  可以通过order属性改变，  数值小的在前面，还可以是负数。 |
| flex-grow | [number] | 设置或检索弹性盒的扩展比率,  根据弹性盒子元素所设置的扩展因子作为比率来分配剩余空间 |
| flex-shrink | [number] | 设置或检索弹性盒的收缩比率,  根据弹性盒子元素所设置的收缩因子作为比率来收缩空间 |
| flex-basis | [length], auto | 设置或检索弹性盒伸缩基准值 |
| align-self | auto,flex-start,flex-end,center,baseline,stretch | 设置或检索弹性盒子元素在侧轴（纵轴）方向上的对齐方式，  可以覆盖父容器align-items的设置 |

## 你怎么清楚浮动

1、在浮动元素后面添加 clear:both 的空 div 元素

<div class="container">

<div class="left"></div>

<div class="right"></div>

<div style="clear:both"></div>

</div>

2、给父元素添加 overflow:hidden 或者 auto 样式，触发BFC。

<div class="container">

<div class="left"></div>

<div class="right"></div>

</div>

.container{

width: 300px;

background-color: #aaa;

overflow:hidden;

zoom:1; /\*IE6\*/

}

3、使用伪元素，也是在元素末尾添加一个点并带有 clear: both 属性的元素实现的。

<div class="container clearfix">

<div class="left"></div>

<div class="right"></div>

</div>

.clearfix{

zoom: 1; /\*IE6\*/

}

.clearfix:after{

content: ".";

height: 0;

clear: both;

display: block;

visibility: hidden;

}

推荐使用第三种方法，不会在页面新增div，文档结构更加清晰。

## 两边宽度固定中间自适应的三栏布局(默写题,手写笔试题)

圣杯布局和双飞翼布局是前端工程师需要日常掌握的重要布局方式。两者的功能相同，都是为了实现一个两侧宽度固定，中间宽度自适应的三栏布局。

圣杯布局

<style>

body{

min-width: 550px;

}

#container{

padding-left: 200px;

padding-right: 150px;

}

#container .column{

float: left;

}

#center{

width: 100%;

}

#left{

width: 200px;

margin-left: -100%;

position: relative;

right: 200px;

}

#right{

width: 150px;

margin-right: -150px;

}

</style>

<div id="container">

<div id="center" class="column">center</div>

<div id="left" class="column">left</div>

<div id="right" class="column">right</div>

</div>

双飞翼布局

<style>

body {

min-width: 500px;

}

#container {

width: 100%;

}

.column {

float: left;

}

#center {

margin-left: 200px;

margin-right: 150px;

}

#left {

width: 200px;

margin-left: -100%;

}

#right {

width: 150px;

margin-left: -150px;

}

</style>

<div id="container" class="column">

<div id="center">center</div>

</div>

<div id="left" class="column">left</div>

<div id="right" class="column">right</div>

## 浏览器渲染机制

构建DOM树（parse）：渲染引擎解析HTML文档，首先将标签转换成DOM树中的DOM node

构建渲染树（construct）：解析对应的CSS样式文件信息

布局渲染树（reflow/layout）：从根节点递归调用，计算每一个元素的大小、位置等，给出每个节点所应该在屏幕上出现的精确坐标；

绘制渲染树（paint/repaint）：遍历渲染树，使用UI后端层来绘制每个节点

## 重绘和回流的区别

重绘（repaint或redraw）：当盒子的位置、大小以及其他属性，例如颜色、字体大小等都确定下来之后，浏览器便把这些原色都按照各自的特性绘制一遍，将内容呈现在页面上。重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。

重绘发生在元素的可见的外观被改变，但并没有影响到布局的时候。比如，仅修改DOM元素的字体颜色（只有Repaint，因为不需要调整布局）

回流（重构/重排/reflow）：当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建, 这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。

触发回流的条件：任何页面布局和几何属性的改变都会触发回流：

页面渲染初始化(无法避免)

添加或删除可见的DOM元素

元素位置的改变，或者使用动画

元素尺寸的改变——大小，外边距，边框

浏览器窗口尺寸的变化

填充内容的改变，比如文本的改变或图片大小改变而引起的计算值宽度和高度的改变

回流必定会引发重绘，但重绘不一定会引发回流。

JavaScript

jQuery

Ajax

## 2JS数据类型

1基本数据类型:string,number,Boolean,null,undefined;

2引用数据类型Object,Array,function

3 ES6新增 symbol

## JS判断数据类型的方法

typeof

console.log(typeof 2); // number

console.log(typeof true); // boolean

console.log(typeof 'str'); // string

console.log(typeof undefined); // undefined

console.log(typeof []); // object

console.log(typeof {}); // object

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

console.log(typeof null); // object

优点：能够快速区分基本数据类型

缺点：不能将Object、Array和Null区分，都返回object

instanceof

console.log(2 instanceof Number); // false

console.log(true instanceof Boolean); // false

console.log('str' instanceof String); // false

console.log([] instanceof Array); // true

console.log(function(){} instanceof Function); // true

console.log({} instanceof Object); // true

优点：能够区分Array、Object和Function，适合用于判断自定义的类实例对象

缺点：Number，Boolean，String基本数据类型不能判断

Object.prototype.toString.call()

console.log(Object.prototype.toString.call(2)); //[object Number]

console.log(Object.prototype.toString.call(true)); //[object Boolean]

console.log(Object.prototype.toString.call('str')); //[object String]

console.log(Object.prototype.toString.call([])); //[object Array]

console.log(Object.prototype.toString.call(function(){})); //[object Function]

console.log(Object.prototype.toString.call({})); //[object Object]

console.log(Object.prototype.toString.call(undefined)); //[object Undefined]

console.log(Object.prototype.toString.call(null)); //[object Null]

优点：精准判断数据类型

缺点：写法繁琐不容易记，推荐进行封装后使用

## null和undefined区别

Undefined类型只有一个值，即undefined。当声明的变量还未被初始化时，变量的默认值为undefined。用法：

变量被声明了，但没有赋值时，就等于undefined。

调用函数时，应该提供的参数没有提供，该参数等于undefined。

对象没有赋值的属性，该属性的值为undefined。

函数没有返回值时，默认返回undefined。

Null类型也只有一个值，即null。null用来表示尚未存在的对象，常用来表示函数企图返回一个不存在的对象。用法

作为函数的参数，表示该函数的参数不是对象。

作为对象原型链的终点。

## JS作用域的理解

JS中的作用域分为两种：全局作用域和函数作用域。函数作用域中定义的变量，只能在函数中调用，外界无法访问。没有块级作用域导致了if或for这样的逻辑语句中定义的变量可以被外界访问，因此ES6中新增了let和const命令来进行块级作用域的声明。

## call,apply和bind区别的页面在哪些流览器测

三个函数的作用都是将函数绑定到上下文中，用来改变函数中this的指向；三者的不同点在于语法的不同。

fun.call(thisArg[, arg1[, arg2[, ...]]])

fun.apply(thisArg, [argsArray])

所以 apply和 call的区别是 call方法接受的是若干个参数列表，而 apply接收的是一个包含多个参数的数组。

而bind()方法创建一个新的函数, 当被调用时，将其this关键字设置为提供的值，在调用新函数时，在任何提供之前提供一个给定的参数序列。

var bindFn = fun.bind(thisArg[, arg1[, arg2[, ...]]])

bindFn()

## 深拷贝和浅拷贝

浅拷贝

function simpleClone(obj) {

var result = {};

for (var i in obj) {

result[i] = obj[i];

}

return result;

}

深拷贝，遍历对象中的每一个属性

测试案例

var obj={

oName:"kky",

oAge:18,

oAddress:{

province:"beijing"

},

ofavorite:[

"swimming",

{reading:"history book"}

],

skill:function(){

console.log("bob is coding");

}

};

var arr=[1,[2,3],{user:"kky"}];

var obj2=deepClone(obj);

//深度克隆

function deepClone(obj){

var o = obj instanceof Array ? [] : {};

for(var k in obj){

o[k] = typeof obj[k] === "object" ? arguments.callee(obj[k]) : obj[k];

}

return o;

}

## 谈谈防抖和节流

说白了，防抖节流就是使用定时器来实现我们的目的。

防抖(debounce)：

在事件被触发n秒后再执行回调，如果在这n秒内又被触发，则重新计时。

典型的案例就是输入框搜索：输入结束后n秒才进行搜索请求，n秒内又输入的内容，则重新计时。

节流(throttle)：

规定在一个单位时间内，只能触发一次函数，如果这个单位时间内触发多次函数，只有一次生效。

典型的案例就是鼠标不断点击触发，规定在n秒内多次点击只生效一次。

为什么要掌握防抖和节流

函数节流（throttle）与函数防抖（debounce）都是可以限制函数的执行频次，根据不同的场景来对执行频率进行限制，避免了函数触发频率过高导致的响应速度跟不上触发频率，出现延迟，假死或卡顿的现象。

防抖

function debounce(fn, delay) {

let timer = null;

return function () {

if (timer) clearTimeout(timer);

timer = setTimeout(() => {

fn.apply(this, arguments);

}, delay);

}

}

节流

function throttle(fn, cycle) {

let start = Date.now();

let now;

let timer;

return function () {

now = Date.now();

clearTimeout(timer);

if (now - start >= cycle) {

fn.apply(this, arguments);

start = now;

} else {

timer = setTimeout(() => {

fn.apply(this, arguments);

}, cycle);

}

}

}

## 谈谈cookie,sessionStorage和localStorage

cookie用来保存登录信息，大小限制为4KB左右

localStorage是Html5新增的，用于本地数据存储，保存的数据没有过期时间，一般浏览器大小限制在5MB

sessionStorage接口方法和localStorage类似，但保存的数据的只会在当前会话中保存下来，页面关闭后会被清空。

| **名称** | **生命期** | **大小** | **与服务器通信** |
| --- | --- | --- | --- |
| cookie | 一般由服务器生成，可设置失效时间。  如果在浏览器端生成Cookie，  默认是关闭浏览器后失效 | 4KB | 每次都会携带在HTTP头，  如果使用cookie保存过多数据会带来性能问题 |
| localStorage | 除非被清除，  否则永久保存 | 5MB | 仅在浏览器中保存，不与服务器通信 |
| sessionStorage | 仅在当前会话下有效，  关闭页面或浏览器后被清除 | 5MB | 仅在浏览器中保存，不与服务器通信 |

## 0.1+0.2!=0.3怎么处理

把需要计算的数字升级（乘以10的n次幂）成计算机能够精确识别的整数，等计算完成后再进行降级（除以10的n次幂），即：

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

## 数组的常用方法(至少说出8个)

改变原数组的方法

1. splice() 添加/删除数组元素

语法：arrayObject.splice(index,howmany,item1,.....,itemX)

参数:

1.index：必需。整数，规定添加/删除项目的位置，使用负数可从数组结尾处规定位置。

2.howmany：可选。要删除的项目数量。如果设置为 0，则不会删除项目。

3.item1, ..., itemX：可选。向数组添加的新项目。

返回值: 如果有元素被删除,返回包含被删除项目的新数组。

1. sort() 数组排序

语法：arrayObject.sort(sortby)

参数:

1.sortby 可选。规定排序顺序。必须是函数。。

返回值: 返回包排序后的新数组。

1. pop() 删除一个数组中的最后的一个元素

语法：arrayObject.pop()

参数:无

返回值: 返回被删除的元素。

1. shift() 删除数组的第一个元素

语法：arrayObject.shift()

参数:无

返回值: 返回被删除的元素。

1. push() 向数组的末尾添加元素

语法：arrayObject.push(newelement1,newelement2,....,newelementX)

参数:

1.newelement1 必需。要添加到数组的第一个元素。

2.newelement2 可选。要添加到数组的第二个元素。

3.newelementX 可选。可添加若干个元素。

返回值: 返回被删除的元素。

1. unshift() 向数组的开头添加一个或更多元素

语法：arrayObject.unshift(newelement1,newelement2,....,newelementX)

参数:

1.newelement1 必需。要添加到数组的第一个元素。

2.newelement2 可选。要添加到数组的第二个元素。

3.newelementX 可选。可添加若干个元素。

返回值: arrayObject 的新长度。。

1. reverse() 颠倒数组中元素的顺序

语法：arrayObject.reverse()

参数:无

返回值: 颠倒后的新数组。

1. copyWithin() 指定位置的成员复制到其他位置

语法：array.copyWithin(target, start = 0, end = this.length)

参数:

1.target（必需）：从该位置开始替换数据。如果为负值，表示倒数。

2.start（可选）：从该位置开始读取数据，默认为 0。如果为负值，表示倒数。

3.end（可选）：到该位置前停止读取数据，默认等于数组长度。如果为负值，表示倒数。

返回值: 返回当前数组。

1. fill() 填充数组

语法：array.fill(value, start, end)

参数:

1.value 必需。填充的值。

2.start 可选。开始填充位置。

3.end 可选。停止填充位置 (默认为 array.length)

返回值: 返回当前数组。

不改变原数组的方法

1. slice() 浅拷贝数组的元素

语法：array.slice(begin, end);

参数:

1.begin(可选): 索引数值,接受负值，从该索引处开始提取原数组中的元素,默认值为0。

2.end(可选):索引数值(不包括),接受负值，在该索引处前结束提取原数组元素，默认值为数组末尾(包括最后一个元素)。

返回值: 返回一个从开始到结束（不包括结束）选择的数组的一部分浅拷贝到一个新数组对象，且原数组不会被修改。

1. join() 数组转字符串

语法：array.join(str)

参数:

1.str(可选): 指定要使用的分隔符，默认使用逗号作为分隔符。

返回值: 返回生成的字符串。

1. concat() 合并两个或多个数组

语法：var newArr =oldArray.concat(arrayX,arrayX,......,arrayX)

参数:

1.arrayX（必须）：该参数可以是具体的值，也可以是数组对象。可以是任意多个。

返回值: 返回返回合并后的新数组。

1. indexOf() 查找数组是否存在某个元素

语法：array.indexOf(searchElement,fromIndex)

参数:

1.searchElement(必须):被查找的元素

2.fromIndex(可选):开始查找的位置(不能大于等于数组的长度，返回-1)，接受负值，默认值为0。

返回值: 返回下标

1. lastIndexOf() 查找指定元素在数组中的最后一个位置

语法：arr.lastIndexOf(searchElement,fromIndex)

参数:

1.searchElement(必须): 被查找的元素

2.fromIndex(可选): 逆向查找开始位置，默认值数组的长度-1，即查找整个数组。

返回值: 方法返回指定元素,在数组中的最后一个的索引，如果不存在则返回 -1。（从数组后面往前查找）

1. includes() 查找数组是否包含某个元素

语法：array.includes(searchElement,fromIndex=0)

参数:

1.searchElement(必须):被查找的元素

2.fromIndex(可选):默认值为0，参数表示搜索的起始位置，接受负值。正值超过数组长度，数组不会被搜索，返回false。负值绝对值超过长数组度，重置从0开始搜索。

返回值: 返回布尔

## new一个对象的过程中发生了什么吗?

1. 创建空对象；

var obj = {};

2. 设置新对象的constructor属性为构造函数的名称，设置新对象的\_\_proto\_\_属性指向构造函数的prototype对象；

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

3. 使用新对象调用函数，函数中的this被指向新实例对象：

ClassA.call(obj);//{}.构造函数();

4. 将初始化完毕的新对象地址，保存到等号左边的变量中

## JS实现继承(至少会一种)

首先创建一个父类

// 定义一个动物类

function Animal(name, color) {

// 属性

this.name = name || 'Animal';

this.color = color || ['black'];

// 实例方法

this.sleep = function () {

console.log(this.name + '正在睡觉！');

}

}

// 原型方法

Animal.prototype.eat = function (food) {

console.log(this.name + '正在吃：' + food);

};

原型链继承

new了一个空对象，这个空对象指向Animal并且Cat.prototype指向了这个空对象，这种就是基于原型链的继承。

function Cat(name) {

this.name = name || 'tom'

}

Cat.prototype = new Animal()

var cat = new Cat()

cat.color.push('red')

cat.sleep() //tom正在睡觉！

cat.eat('fish') //tom正在吃：fish

console.log(cat.color) //["black", "red"]

console.log(cat instanceof Animal) //true

console.log(cat instanceof Cat) //true

var new\_cat = new Cat()

console.log(new\_cat.color) //["black", "red"]

特点：基于原型链，既是父类的实例，也是子类的实例。

缺点：1.无法实现多继承；2.所有新实例都会共享父类实例的属性。

构造继承

function Dog(name) {

Animal.call(this)

this.name = name || 'mica'

}

var dog = new Dog()

dog.color.push('blue')

dog.sleep() // mica正在睡觉！

dog.eat('bone') //Uncaught TypeError: dog.eat is not a function

console.log(dog.color) //["black", "blue"]

console.log(dog instanceof Animal) //false

console.log(dog instanceof Dog) //true

var new\_dog = new Dog()

console.log(new\_dog.color) //["black"]

特点：可以实现多继承（call多个），解决了所有实例共享父类实例属性的问题。

缺点：1.只能继承父类实例的属性和方法；2.不能继承原型上的属性和方法。

组合继承

function Mouse(name){

Animal.call(this)

this.name = name || 'jerry'

}

Mouse.prototype = new Animal()

Mouse.prototype.constructor = Mouse

var mouse = new Mouse()

mouse.color.push('yellow)

mouse.sleep() //jerry正在睡觉！

mouse.eat('carrot') //jerry正在吃：carrot

console.log(mouse instanceof Animal)//true

console.log(mouse instanceof Mouse)//true

var new\_mouse = new Mouse()

console.log(new\_mouse.color) //["black"]

特点：可以继承实例属性/方法，也可以继承原型属性/方法

缺点：调用了两次父类构造函数，生成了两份实例

## get和post区别

| **请求方式** | **GET** | **POST** |
| --- | --- | --- |
| 参数位置 | 参数拼接到url的后面 | 参数在请求体中 |
| 参数大小 | 受限于浏览器url大小，一般不超过32K | 1G |
| 服务器数据接收 | 接收1次 | 根据数据大小，可分多次接收 |
| 适用场景 | 从服务器端获取数据 | 向服务器提交数据 |
| 安全性 | 参数携带在url中，安全性低 | 相对于GET请求，安全性更高 |

## JSONP原理

由于浏览器的同源策略限制，不允许跨域请求；但是页面中的 script、img、iframe标签是例外，不受同源策略限制。

Jsonp 就是利用 script标签跨域特性进行请求。

JSONP 的原理就是，先在全局注册一个回调函数，定义回调数据的处理；与服务端约定好一个同名回调函数名，服务端接收到请求后，将返回一段 Javascript，在这段 Javascript 代码中调用了约定好的回调函数，并且将数据作为参数进行传递。当网页接收到这段 Javascript 代码后，就会执行这个回调函数。

JSONP缺点：它只支持 GET请求，而不支持 POST请求等其他类型的HTTP请求。

## 缓存的理解

缓存分为强缓存和协商缓存。强缓存不过服务器，协商缓存需要过服务器，协商缓存返回的状态码是304。两类缓存机制可以同时存在，强缓存的优先级高于协商缓存。当执行强缓存时，如若缓存命中，则直接使用缓存数据库中的数据，不再进行缓存协商。

强缓存

Expires

cache-control

协商缓存

Last-Modified 和 If-Modified-Since

Etag 和 If-None-Match

## XSS和CSRF区别

跨站脚本攻击（Cross Site Scripting)，为了不和层叠样式表 CSS 混淆，故将跨站脚本攻击缩写为 XSS。恶意攻击者往 Web 页面里插入恶意 Script 代码，当用户浏览该页之时，嵌入其中 Web 里面的 Script 代码会被执行，从而达到恶意攻击用户的目的。

跨站请求伪造（Cross-site request forgery），是伪造请求，冒充用户在站内的正常操作。我们知道，绝大多数网站是通过 cookie 等方式辨识用户身份，再予以授权的。所以要伪造用户的正常操作，最好的方法是通过 XSS 或链接欺骗等途径，让用户在本机（即拥有身份 cookie 的浏览器端）发起用户所不知道的请求。

区别：

原理不同，CSRF是利用网站A本身的漏洞，去请求网站A的api；XSS是向目标网站注入JS代码，然后执行JS里的代码。

CSRF需要用户先登录目标网站获取cookie，而XSS不需要登录

CSRF的目标是用户，XSS的目标是服务器

XSS是利用合法用户获取其信息，而CSRF是伪造成合法用户发起请求

## HTTP与HTTPS的区别

HTTP的URL由 http://起始且默认使用端口80，而HTTPS的URL由 https://起始且默认使用端口443

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

HTTP的连接很简单，是无状态的，HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全

## HTTP状态码

1xx表示客户端应该继续发送请求

2xx表示成功的请求

200表示OK，正常返回信息

201表示请求成功且服务器创建了新的资源

202表示服务器已经接受了请求，但还未处理

3xx表示重定向

301表示永久重定向，请求的网页已经永久移动到新位置

302表示临时重定向

304表示自从上一次请求以来，页面的内容没有改变过

4xx表示客户端错误

401表示服务器无法理解请求的格式

402表示请求未授权

403表示禁止访问

404表示请求的资源不存在，一般是路径写错了

5xx表示服务器错误

500表示最常见的服务器错误

503表示服务器暂时无法处理请求

## 性能优化

使用CDN

gzip压缩

文本压缩

合并请求

雪碧图

图片懒加载

缓存资源

减少DOM操作

## 性能优化

大致流程

URL 解析

DNS 查询

TCP 连接

处理请求

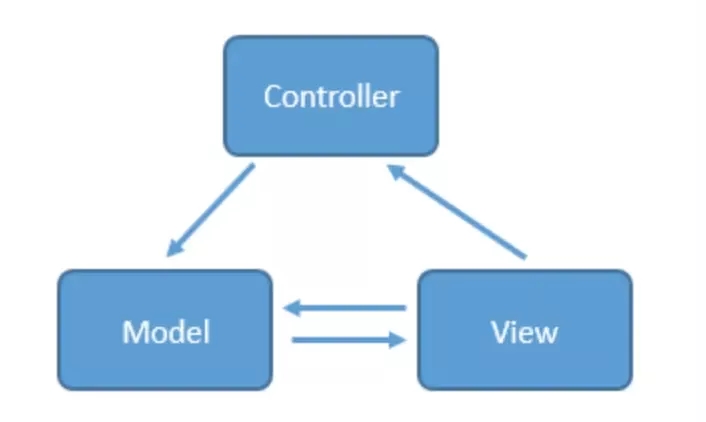
接受响应

渲染页面

Vue

## mvc和mvvm理解

MVC

MVC即Model View Controller，简单来说就是通过controller的控制去操作model层的数据，并且返回给view层展示

View 接受用户交互请求

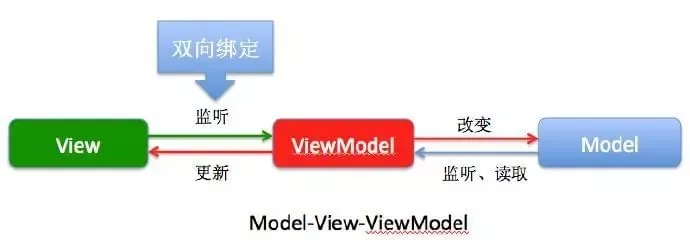
View 将请求转交给Controller处理

Controller 操作Model进行数据更新保存

数据更新保存之后，Model会通知View更新

View 更新变化数据使用户得到反馈

MVVM

MVVM即Model-View-ViewModel，将其中的 View 的状态和行为抽象化，让我们可以将UI和业务逻辑分开。MVVM的优点是低耦合、可重用性、独立开发。

View 接收用户交互请求

View 将请求转交给ViewModel

ViewModel 操作Model数据更新

Model 更新完数据，通知ViewModel数据发生变化

ViewModel 更新View数据

MVVM模式和MVC有些类似，但有以下不同

ViewModel 替换了 Controller，在UI层之下

ViewModel 向 View 暴露它所需要的数据和指令对象

ViewModel 接收来自 Model 的数据

概括起来，MVVM是由MVC发展而来，通过在 Model之上而在 View之下增加一个非视觉的组件将来自 Model的数据映射到 View中。

## vue2双向绑定原理(最简洁版)

vue采用数据劫持结合发布者-订阅者模式的方式，通过 Object.defineProperty劫持data属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

## v-model双向绑定原理

v-model本质上是语法糖，v-model 在内部为不同的输入元素使用不同的属性并抛出不同的事件。

text 和 textarea 元素使用 value 属性和 input 事件

checkbox 和 radio 使用 checked 属性和 change 事件

select 字段将 value 作为 prop 并将 change 作为事件

所以我们可以v-model进行如下改写：

1. <input v-model="sth"/>
2. // 等同于
3. <input :value="sth" @input="sth = $event.target.value"/>

这个语法糖必须是固定的，也就是说属性必须为value，方法名必须为：input。

知道了v-model的原理，我们可以在自定义组件上实现v-model。

//Parent

<template>

{{num}}

<Child v-model="num">

</template>

export default {

data(){

return {

num: 0

}

}

}

//Child

<template>

<div @click="add">Add</div>

</template>

export default {

props: ['value'],

methods:{

add(){

this.$emit('input', this.value + 1)

}

}

}

## vue生命周期函数

|  |  |
| --- | --- |
| beforeCreate | 组件实例被创建之初，组件的属性生效之前 |
| created | 组件实例已经完全创建，属性也绑定，但真实 dom 还没有生成，$el 还不可用 |
| beforeMount | 在挂载开始之前被调用：相关的 render 函数首次被调用 |
| mounted | el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子 |
| beforeUpdate | 组件数据更新之前调用，发生在虚拟 DOM 打补丁之前 |
| updated | 组件数据更新之后 |
| activited | keep-alive 专属，组件被激活时调用 |
| deactivated | keep-alive 专属，组件被销毁时调用 |
| beforeDestroy | 组件销毁前调用 |
| destroyed | 组件销毁后调用 |

## 组件data为什么返回函数

组件中的data写成一个函数，数据以函数返回值形式定义，这样每复用一次组件，就会返回一份新的data。如果单纯的写成对象形式，就使得所有组件实例共用了一份data，造成了数据污染。

## vue给对象新增属性页面没有响应

由于Vue会在初始化实例时对属性执行 getter/setter转化，所以属性必须在 data对象上存在才能让Vue将它转换为响应式的。Vue提供了$set方法用来触发视图更新。

export default {

data(){

return {

obj: {

name: 'fei'

}

}

},

mounted(){

this.$set(this.obj, 'sex', 'man')

}

}

## v-if和v-show区别

v-if 是真正的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建；也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 的 “display” 属性进行切换。

所以，v-if 适用于在运行时很少改变条件，不需要频繁切换条件的场景；v-show 则适用于需要非常频繁切换条件的场景。

## vue中scoped属性作用

在Vue文件中的style标签上有一个特殊的属性，scoped。当一个style标签拥有scoped属性时候，它的css样式只能用于当前的Vue组件，可以使组件的样式不相互污染。如果一个项目的所有style标签都加上了scoped属性，相当于实现了样式的模块化。

scoped属性的实现原理是给每一个dom元素添加了一个独一无二的动态属性，给css选择器额外添加一个对应的属性选择器，来选择组件中的dom。

## vue中ref的作用

获取dom元素 this.$refs.box

获取子组件中的data this.$refs.box.msg

调用子组件中的方法 this.$refs.box.open()

## keep-alive的作用

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，避免重新渲染 ，其有以下特性：

一般结合路由和动态组件一起使用，用于缓存组件；

提供 include 和 exclude 属性，两者都支持字符串或正则表达式， include 表示只有名称匹配的组件会被缓存，exclude 表示任何名称匹配的组件都不会被缓存 ，其中 exclude 的优先级比 include 高；

对应两个钩子函数 activated 和 deactivated ，当组件被激活时，触发钩子函数 activated，当组件被移除时，触发钩子函数 deactivated。

## 计算属性 computed 和事件 methods 有什么区别

我们可以将同一函数定义为一个 method 或者一个计算属性。对于最终的结果，两种方式是相同的。

不同点：

computed：计算属性是基于它们的依赖进行缓存的，只有在它的相关依赖发生改变时才会重新求值。

method：只要发生重新渲染， method 调用总会执行该函数。

## vue中computed和watch区别

当页面中有某些数据依赖其他数据进行变动的时候，可以使用计算属性computed。

watch用于观察和监听页面上的vue实例，如果要在数据变化的同时进行异步操作或者是比较大的开销，那么watch为最佳选择。

## vue中父子组件传值

父传子

1. 在该组件(该子组件)内通过props声明一些自定义属性
2. 在该组件(该子组件)上通过v-bind缩写为:,将父组件中要传的数据绑定到刚才声明的那些自定义属性上
3. 在该组件(该子组件)内,刚才自定义的属性就是从父元素接收过来的数据

子传父

1. 在该组件(该子组件)上通过v-on(缩写@),监听(绑定)一个自定义是事件,并添加一个与之对应父组件中的一个函数fn
2. 在该组件(该子组件)内,通过$emit触发刚才那个自定义事件,并传值,即$meit(自定义事件名称,参数1,参数2)
3. 父组件的那个关联的函数fn的形参就是要接收的数据

## axios拦截器怎么配(默写题)

// 添加请求拦截器

axios.interceptors.request.use(function (config) {

// 在发送请求之前做些什么

return config;

}, function (error) {

// 对请求错误做些什么

return Promise.reject(error);

});

// 添加响应拦截器

axios.interceptors.response.use(function (response) {

// 对响应数据做点什么

return response;

}, function (error) {

// 对响应错误做点什么

return Promise.reject(error);

});

## 0vue-router 有哪几种导航钩子?

三种

全局导航钩子（跳转前进行判断拦截）

router.beforeEach(to, from, next),

router.beforeResolve(to, from, next),

router.afterEach(to, from ,next)

组件内钩子

beforeRouteEnter

beforeRouteUpdate

beforeRouteLeave

单独路由独享组件

beforeEnter

## 谈谈vuex

vuex是一个能方便vue实例及其组件传输数据的插件 方便传输数据，作为公共存储数据的一个库

state: 状态中心

getters: 获取状态

mutations: 更改状态，同步的

actions: 异步更改状态

modules: 将state分成多个modules，便于管理

应用场景：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。

网上找的一个通俗易懂的了解vuex的例子

公司有个仓库

1.State（公司的仓库）

2.Getter（只能取出物品，包装一下，不能改变物品任何属性）

3.Muitation（仓库管理员，只有他可以直接存储到仓库）

4.Action（公司的物料采购员，负责从外面买东西和接货， 要往仓库存东西，告诉仓库管理员要存什么）

非常要注意的地方：只要刷新或者退出浏览器，仓库清空。

## vue-router中$route 和 $router 的区别

$router 为 VueRouter 实例，想要导航到不同 URL，则使用 $router.push 方法。

$route 为当前 router 跳转对象里面可以获取 name 、 path 、 query 、 params 等。

## nextTick 是做什么的

$nextTick 是在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后使用 $nextTick，则可以在回调中获取更新后的 DOM。

## 你有对 Vue 项目进行哪些优化？

如果没有对 Vue 项目没有进行过优化总结的同学，可以参考本文作者的另一篇文章《 Vue 项目性能优化 — 实践指南 》，文章主要介绍从 3 个大方面，22 个小方面详细讲解如何进行 Vue 项目的优化。（1）代码层面的优化

v-if 和 v-show 区分使用场景

computed 和 watch 区分使用场景

v-for 遍历必须为 item 添加 key，且避免同时使用 v-if

长列表性能优化

事件的销毁

图片资源懒加载

路由懒加载

第三方插件的按需引入

优化无限列表性能

服务端渲染 SSR or 预渲染

（2）Webpack 层面的优化

Webpack 对图片进行压缩

减少 ES6 转为 ES5 的冗余代码

提取公共代码

模板预编译

提取组件的 CSS

优化 SourceMap

构建结果输出分析

Vue 项目的编译优化

（3）基础的 Web 技术的优化

开启 gzip 压缩

浏览器缓存

CDN 的使用

使用 Chrome Performance 查找性能瓶颈