# 面试题

# JS方面：

**typeof**操作符返回一个字符串，表示未经计算的操作数的类型。

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

**isPrototypeOf()** 方法用于测试一个对象是否存在于另一个对象的原型链上。

## Es6那些新特性？

Let,const Set Map Class 解构赋值，扩展运算符…，字符串模板，函数表达式

## vue双向绑定的原理

2.0的话是使用了defineProperty，

3.0有两种情况， reactive (proxy), ref(reflect)

## H5里面Websocket的作用

客户端需要一直发送请求，请求服务器端，查看数据发生变化，需要请求很多次，

使用websocket的话，只需要请求一次即可，当服务器有变化时，就是告诉服务器端发起请求。聊天室就可以用这个来做。

## Vue写插件的时候，比较核心的install方法是怎么实现的

// 注册一个挂载在实例上的use方法

1. 当插件中install是一个函数的时候，调用install方法，指向插件，并把一众参数传入
2. plugin.install.apply(plugin, args)
3. 当插件本身就是一个函数的时候，把它当做install方法，指向插件，并把一众参数传入
4. plugin.apply(null, args)

## vue里面slot，以及具名插槽是怎么使用的，有什么作用

子组件有个

<template slot="header">

<p>我是name为header的slot</p>

</template>

父组件：

<slot name="header"></slot>

我的理解在可以更加动态的修改子组件的代码

## H5里面的语义标签大概有哪些，经常使用什么

通常我们认为h5就是html5，也就是html5标准，

所谓的 H5 开发，通常指代在微信中，点开后会在微信内部展开的精美的移动端的 Web 页面。

HTML 语义化的核心是反对大篇幅的使用无语义化的 div + css + span，而鼓励使用 HTML 定义好的语义化标签。

Header， hgroup，footer，section，aside

## 我现有项目里面使用的微信开发者工具是比较原生的小程序开发方法，有没有了解过比较新的小程序写法

## es6中Promise、async await、generator分别是怎么使用的

Promise：



Generator：



## 她给了一个场景：按序分别请求三次数据，前一次调用结束才能开始下一次，怎么写

三个promise嵌套，

在promise里面，await+请求方法，resolve的时候回调下一个请求

## css里面浮动怎么清除？

1.添加 clearfix属性

.clearfix::after {

content: '';

display: block;

clear: both;

}

2.空div方法：<div style="clear:both;"></div>。

3. overflow: auto或overflow: hidden方法。设置overflow: hidden的方法可能使其子元素显示不完整，当子元素的高度大于父元素时。

## 高度塌陷一般出现在哪些场景？怎么解决？

当子元素有浮动，父元素没有高度的时候，父元素会出现高度塌陷

## 有没有学过webpack？配置或者loader了解过吗？

## keep-alive标签的原理是什么？有什么功能？

概念

    keep-alive 是 Vue 的内置组件，当它包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们。和 transition 相似，keep-alive 是一个抽象组件：它自身不会渲染成一个 DOM 元素，也不会出现在父组件链中。

作用

    在组件切换过程中将状态保留在内存中，防止重复渲染DOM，减少加载时间及性能消耗，提高用户体验性

原理

    在 created 函数调用时将需要缓存的 VNode 节点保存在 this.cache 中／在 render（页面渲染） 时，如果 VNode 的 name 符合缓存条件（可以用 include 以及 exclude 控制），则会从 this.cache 中取出之前缓存的 VNode 实例进行渲染。

    VNode：虚拟DOM，其实就是一个JS对象

Props

include - 字符串或正则表达式。只有名称匹配的组件会被缓存。

exclude - 字符串或正则表达式。任何名称匹配的组件都不会被缓存。

max - 数字。最多可以缓存多少组件实例。

## http里面cache-control了解吗？有什么作用？

## 在页面中的动画，设置一个定时器，间隔一段时间就动一次，考虑性能，以及考虑整个页面的刷新时间，应该设置多长时间？

## 跨域是怎么实现的

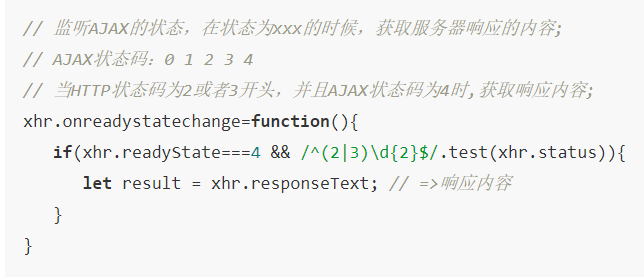
Nginx

## 在同源的标签里面要传递数据，会用什么方法？

## 简洁描述对原型链和闭包的理解

## ajax的原理

1. 创建ajax实例let xhr = new XMLHttpRequest;
2. 打开URL（配置发送请求的信息）xhr.open('GET','./json/xxx.json',true); /
3. 监听AJAX状态



1. send中放的是请求主体的内容;

## This指向问题？



## Apply，call ，bind

他们可以改变this的指向

Call（）：调用函数，并且改变函数内的this的指向

Apply（） 调用函数，传递的参数必须是一个数组，应用于求数组的最大值

Bind（）不会调用函数，会改变函数内部的this指向，返回的是改变this之后的新函数

## 原型，原型链，构造函数

一个对象,我们称一个对象的prototype为原型对象。

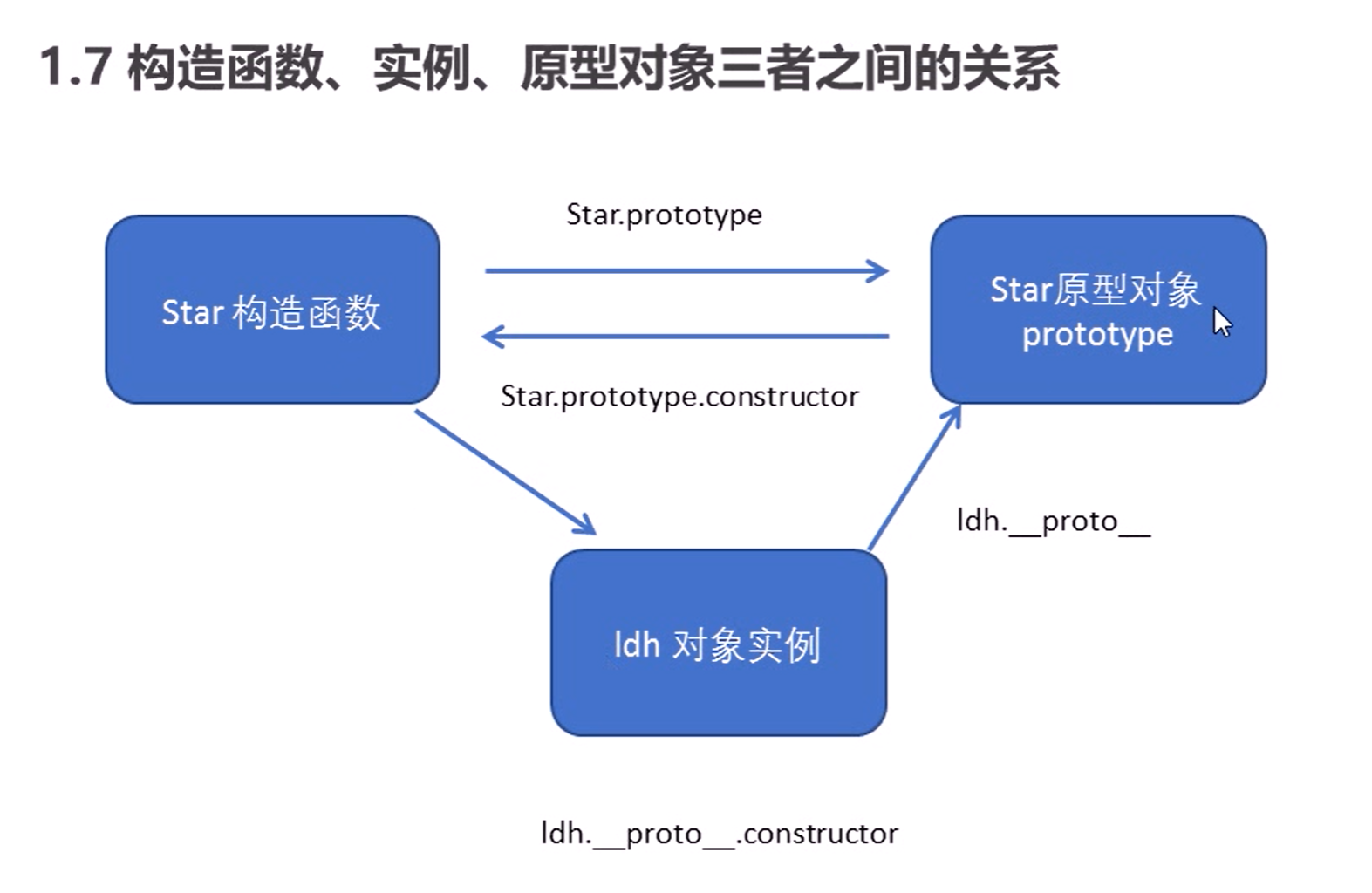
作用是共享方法，减少内存资源。

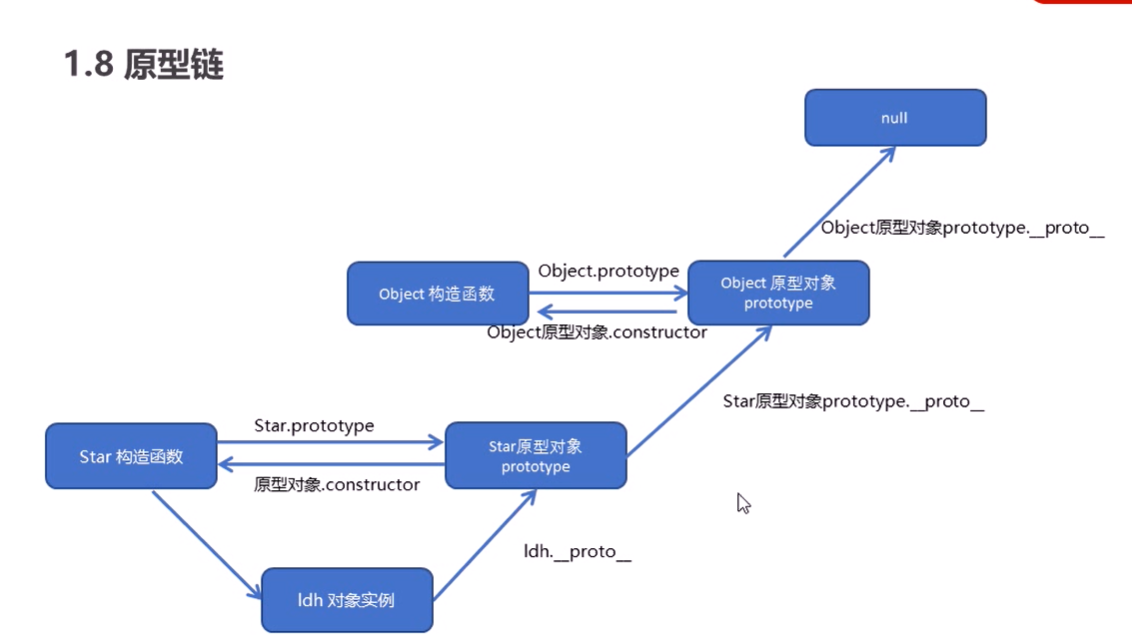
对象有一个\_\_proto\_\_属性，指向构造函数的原型对象（不能修改，只是指明了一条线路）。 \_\_proto\_\_和prototype是等价的

方法的查找规则：

查看对象上是否有该方法，

没有的话去prototype上寻找该方法





成员为一级一级往上查找

## promise和async await有什么区别

## 防抖，节流

防抖：

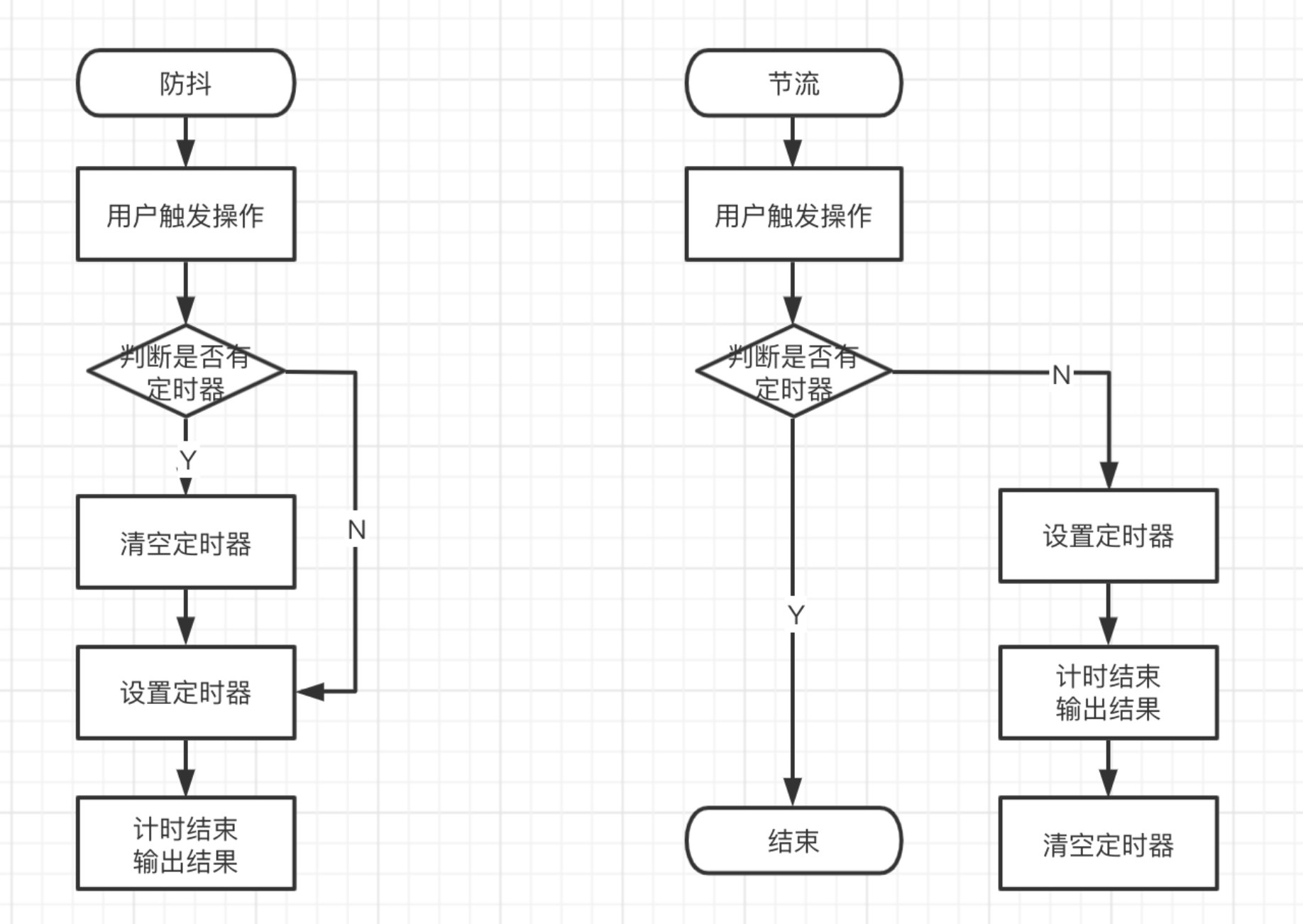
在高频触发回调函数时，防抖操作使回调函数在一定时间间隔内，再次触发会清空定时器，并重新计时；计时结束后输出一次结果。

核心在于，在短时间内大量触发同一事件时，只会执行一次回调函数。避免把一次事件误认为多次。

节流：

在高频触发回调函数时，节流操作使回调函数在每隔一段时间定期执行一次，时间间隔内再触发，不会重新执行。

核心在于让一个函数不要执行的太频繁，减少一些过快的操作。



### 常见场景

* 监听 scroll、mousemove 等事件 - 节流（每隔一秒计算一次位置）
* 监听浏览器窗口 resize 操作 - 防抖（只需计算一次）
* 键盘文本输入的验证 - 防抖（连续输入文字后发送请求进行验证，验证一次就好）
* 提交表单 - 防抖（多次点击变为一次）

## Token，JWT

## SSO，SEO

## Js常用内置对象？

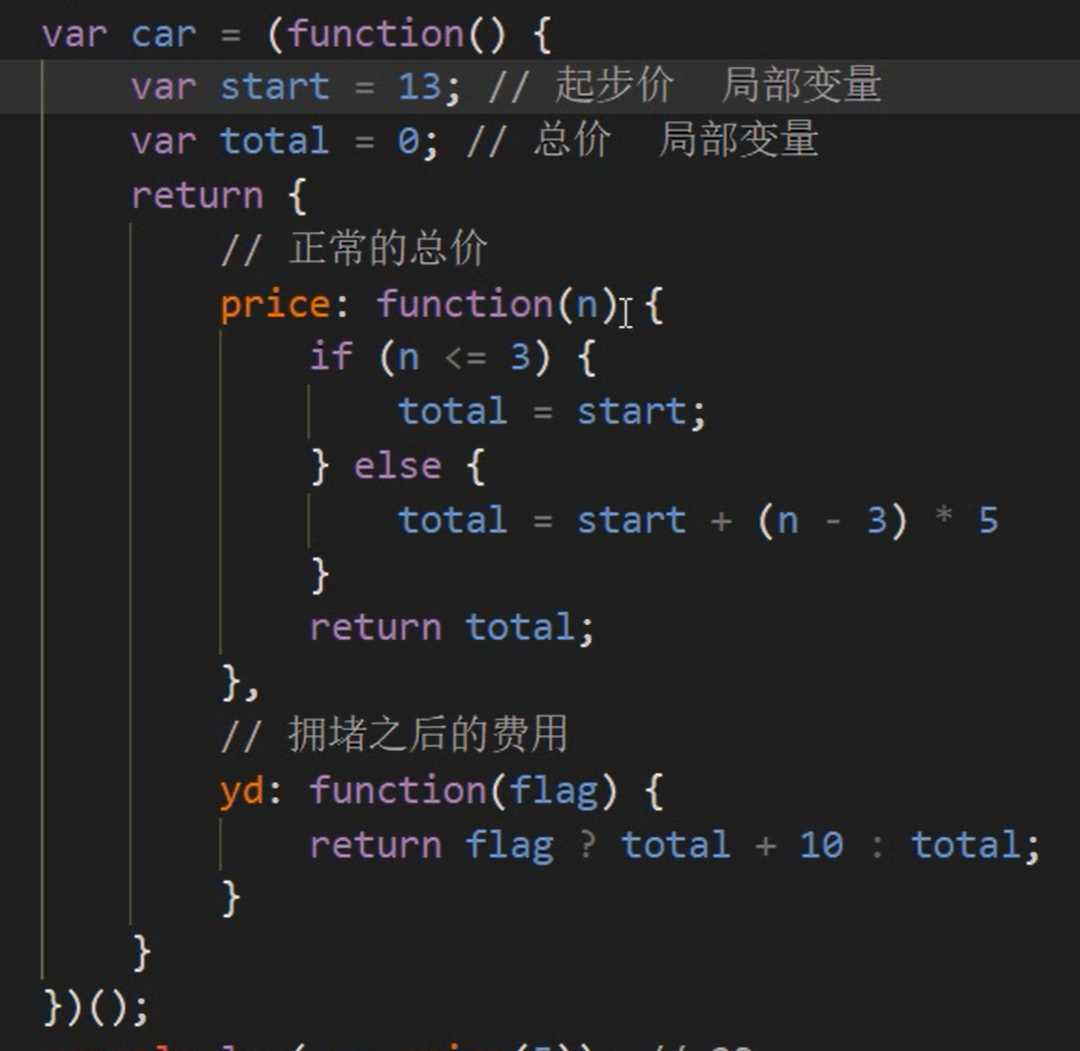
Array数组对象、Math方法、String方法、日期对象的方法。

## 闭包

是指有权使用另一个函数作用域中的变量的函数。

在一个函数中，返回另一个函数。

作用：延伸了变量的作用范围



## 深浅拷贝

浅拷贝：

var target = {}; for(var i in source) { if (source.hasOwnProperty(i)) { target[i] = source[i]; } } return target;

深拷贝：

JSON.parse(JSON.Stringify(obj))

## 功能检测（feature detection）、功能推断（feature inference）

功能检测：

if ('geolocation' in navigator) {

// 可以使用 navigator.geolocation

} else {

// 处理 navigator.geolocation 功能缺失

}

功能推断

if (document.getElementsByTagName) {

element = document.getElementById(id);

}

## AJSX

Ajax（asynchronous JavaScript and XML）是使用客户端上的许多 Web 技术，创建异步 Web 应用的一种 Web 开发技术。借助 Ajax，Web 应用可以异步（在后台）向服务器发送数据和从服务器检索数据，而不会干扰现有页面的显示和行为。通过将数据交换层与表示层分离，Ajax 允许网页和扩展 Web 应用程序动态更改内容，而无需重新加载整个页面。实际上，现在通常将 XML 替换为 JSON，因为 JavaScript 对 JSON 有原生支持优势。

### 使用 Ajax 的优缺点分别是什么？

**优点**

* 交互性更好。来自服务器的新内容可以动态更改，无需重新加载整个页面。
* 减少与服务器的连接，因为脚本和样式只需要被请求一次。
* 状态可以维护在一个页面上。JavaScript 变量和 DOM 状态将得到保持，因为主容器页面未被重新加载。
* 基本上包括大部分 SPA 的优点。

**缺点**

* 动态网页很难收藏。
* 如果 JavaScript 已在浏览器中被禁用，则不起作用。
* 有些网络爬虫不执行 JavaScript，也不会看到 JavaScript 加载的内容。
* 基本上包括大部分 SPA 的缺点。

# CSS方面：

## 盒模型有哪些？有什么区别？Bootstrap里面用的是什么盒模型

盒模型的组成：是由content（内容区）、padding（填充区）、border（边框区）、margin（外边距区）这四个部分组成！

### 说说你对盒模型的理解，以及如何告知浏览器使用不同的盒模型渲染布局。

CSS 盒模型描述了以文档树中的元素而生成的矩形框，并根据排版模式进行布局。每个盒子都有一个内容区域（例如文本，图像等）以及周围可选的padding、border和margin区域。

CSS 盒模型负责计算：

* 块级元素占用多少空间。
* 边框是否重叠，边距是否合并。
* 盒子的尺寸。

盒模型有以下规则：

* 块级元素的大小由width、height、padding、border和margin决定。
* 如果没有指定height，则块级元素的高度等于其包含子元素的内容高度加上padding（除非有浮动元素，请参阅下文）。
* 如果没有指定width，则非浮动块级元素的宽度等于其父元素的宽度减去父元素的padding。
* 元素的height是由内容的height来计算的。
* 元素的width是由内容的width来计算的。
* 默认情况下，padding和border不是元素width和height的组成部分。

## rem是基于什么原理进行适配的

rem是全部的长度都相对于根元素<html>元素。通常做法是给html元素设置一个字体大小，然后其他元素的长度单位就为rem。

## Div垂直水平居中，至少说出三种：

.parent{

display: flex;

align-items: center;*/\*让项目在侧轴方向居中\*/*

justify-content: center;*/\*让项目在主轴方向居中\*/*

}

---

.parent{

position: relative;

}

.son{

position: absolute;

top: 0;

bottom: 0;

left: 0;

right: 0;

margin: auto;

}

https://juejin.cn/post/6904888109737967624

## BFC

BFC(Block formatting context)，块级格式化上下文。

BFC有什么用？

* 具备BFC特性的元素，即独立的渲染区域，或者看成是一个容器，容器内的不会影响到外面的容器。
* 避免外间距重叠（overflow：hidden）
* 可以清楚浮动（overflow：hidden）
* 阻止元素被浮动元素覆盖

怎么样触发BFC？

* 根元素或其它包含它的元素
* 浮动元素 (元素的 float 不是 none)
* 绝对定位元素 (元素具有 position 为 absolute 或 fixed)
* 内联块 (元素具有 display: inline-block)
* 表格单元格 (元素具有 display: table-cell，HTML表格单元格默认属性)
* 表格标题 (元素具有 display: table-caption, HTML表格标题默认属性)
* 具有overflow 且值不是 visible 的块元素，
* display: flow-root
* column-span: all 应当总是会创建一个新的格式化上下文，即便具有 column-span: all 的元素并不被包裹在一个多列容器中。

## Flex怎么使用

## 清除浮动的几种方式，及原理？

.clearfix::after {

content: '';

display: block;

clear: both;

}

* 空div方法：<div style="clear:both;"></div>。
* Clearfix 方法：上文使用.clearfix类已经提到。
* overflow: auto或overflow: hidden方法：上文已经提到。

在大型项目中，我会使用 Clearfix 方法，在需要的地方使用.clearfix。设置overflow: hidden的方法可能使其子元素显示不完整，当子元素的高度大于父元素时。

### 如何解决不同浏览器的样式兼容性问题？

* 在确定问题原因和有问题的浏览器后，使用单独的样式表，仅供出现问题的浏览器加载。这种方法需要使用服务器端渲染。
* 使用已经处理好此类问题的库，比如 Bootstrap。
* 使用 autoprefixer 自动生成 CSS 属性前缀。
* 使用 Reset CSS 或 Normalize.css。

### 如何为功能受限的浏览器提供页面？ 使用什么样的技术和流程？

* 优雅的降级：为现代浏览器构建应用，同时确保它在旧版浏览器中正常运行。
* 渐进式增强：构建基于用户体验的应用，但在浏览器支持时添加新增功能。
* 利用 [caniuse.com](https://caniuse.com/) 检查特性支持。
* 使用 autoprefixer 自动生成 CSS 属性前缀。
* 使用 [Modernizr](https://modernizr.com/)进行特性检测。

## CSS 选择器的优先级是如何计算的？

浏览器通过优先级规则，判断元素展示哪些样式。优先级通过 4 个维度指标确定，我们假定以a、b、c、d命名，分别代表以下含义：

a表示是否使用内联样式（inline style）。如果使用，a为 1，否则为 0。

b表示 ID 选择器的数量。

c表示类选择器、属性选择器和伪类选择器数量之和。

d表示标签（类型）选择器和伪元素选择器之和。

优先级的结果并非通过以上四个值生成一个得分，而是每个值分开比较。a、b、c、d权重从左到右，依次减小。判断优先级时，从左到右，一一比较，直到比较出最大值，即可停止。所以，如果b的值不同，那么c和d不管多大，都不会对结果产生影响。比如0，1，0，0的优先级高于0，0，10，10。

当出现优先级相等的情况时，最晚出现的样式规则会被采纳。如果你在样式表里写了相同的规则（无论是在该文件内部还是其它样式文件中），那么最后出现的（在文件底部的）样式优先级更高，因此会被采纳。

在写样式时，我会使用较低的优先级，这样这些样式可以轻易地覆盖掉。尤其对写 UI 组件的时候更为重要，这样使用者就不需要通过非常复杂的优先级规则或使用!important的方式，去覆盖组件的样式了。

# Vue方面：

## Vue3.0新特性

## 生命周期函数，

# 其他：

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

## 三次握手，四次挥手？

## 浏览器输入url，浏览器发生哪些操作？

## 页面渲染流程？

## Webpack，打包以及优化？

## 组件传值？

## keep-alive？

## 双向绑定原理？

## .跨域是有哪几种方式实现的（什么是跨域，限制了什么，不同源的话会进行什么策略）

## 页面渲染html的过程？

浏览器渲染页面的一般过程：

1.浏览器解析html源码，然后创建一个 DOM树。并行请求 css/image/js在DOM树中，每一个HTML标签都有一个对应的节点，并且每一个文本也都会有一个对应的文本节点。DOM树的根节点就是 documentElement，对应的是html标签。

2.浏览器解析CSS代码，计算出最终的样式数据。构建CSSOM树。对CSS代码中非法的语法它会直接忽略掉。解析CSS的时候会按照如下顺序来定义优先级：浏览器默认设置 < 用户设置 < 外链样式 < 内联样式 < html中的style。

3.DOM Tree + CSSOM --> 渲染树（rendering tree）。渲染树和DOM树有点像，但是是有区别的。

DOM树完全和html标签一一对应，但是渲染树会忽略掉不需要渲染的元素，比如head、display:none的元素等。而且一大段文本中的每一个行在渲染树中都是独立的一个节点。渲染树中的每一个节点都存储有对应的css属性。

4.一旦渲染树创建好了，浏览器就可以根据渲染树直接把页面绘制到屏幕上。

以上四个步骤并不是一次性顺序完成的。如果DOM或者CSSOM被修改，以上过程会被重复执行。实际上，CSS和JavaScript往往会多次修改DOM或者CSSOM。

## 大数据?

做过类似的，是预处理，跟后端协商好，数据尽量提前下载再本地，展示时尽量分期

使用vue-virtual-listview 插件

使用[umy-ui](http://www.umyui.com/) 插件

<https://github.com/livelyPeng/pl-table>

## webpack

<https://www.jianshu.com/p/216ed82a3e49>

## 简介

我叫王鑫