# 关于服务器渲染与客户端渲染的区别

#### 服务器渲染：

也称 SSR。当浏览器发送请求时，在服务器上渲染完整的第一个屏幕 dom 结构，即将数据与 html 整合，之后返回给浏览器。

#### 客户端渲染：

当浏览器发送请求后，服务器只会发送 html 框架，浏览器解析后从上到下依次解析，如果发现 ajax（Web 数据交互方式）请求就再发送新的请求，多次 ajax 请求后，将数据和 html 框架拼接，之后便是浏览器显示的页面。

#### 区别：

1. 哪一方完成了 html 和数据的拼接，称之为 XX 渲染
2. 服务器渲染只会进行一次请求，客户端渲染至少进行两次请求。所以服务器渲染的加载速度会快一些，对于网站的首页，一般为服务器渲染。
3. 服务器渲染查看源代码是可以直接看到数据内容；客户端渲染查看源代码时，只能看到 html 的框架，无法看到数据内容。

# 移动端设计稿多大尺寸

一般移动端设计稿是640或者750的尺寸

# 你们做移动端平时在什么浏览器上测试？

Chrome，Safari，微信X5，UC，其他手机自带浏览器

# 移动端是如何调试的？

（1）模拟手机调试chrome://inspect

（2）真机调试之android手机+Chrome

（3）真机调试之iphone + safari

（4）UC浏览器

（5）微信内置浏览器调试

（6）debuggap

（7）抓包

# 移动端怎么固定横屏显示？

（1） 检测手机竖屏的时候，元素给元素添加90deg的旋转

（2） 检测手机横屏的时候，元素给元素的旋转再改为0

# 移动端用过那些meta标签？

<!--1:设置视口宽度 缩放比例-->

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

<!--2:忽略将数字变为电话号码-->

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

<!--3:忽略识别邮箱-->

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

<!--4:IOS中Safari允许全屏浏览-->

<meta content="yes" name="apple-mobile-web-app-capable">

# 为什么移动端click会有300毫秒延迟？

因为移动端浏览器会有一些默认的行为，比如双击缩放、双击滚动。这些行为尤其是双击缩放，主要是为桌面网站在移动端的浏览体验设计的。而在用户对页面进行操作的时候，移动端浏览器会优先判断用户是否要触发默认的行为。

# 固定定位布局键盘挡住输入框内容？

分析：

* 在软件盘弹出之后（在focus中加个延迟时间），获取input的坐标
* 判断input是否遮挡
* 判断 input 是否在可视区的高度以下
* 如果被遮挡了，就向上移动整个页面

（1）通过绑定窗口改变事件,监听键盘的弹出。然后去改变固定定位元素的位置。默认键盘的宽度应该是页面的2分之一。所以我们位移的距离改成键盘的二分之一就可以。

（2）通过定时器实时监听是否触发input。如果触发input框，就把固定定位改变成静态定位。这样就会浏览器会总动把内容顶上去。

# 为什么安卓浏览器看背景图片，有些设备会模糊？

因为手机分辨率太小，如果按照分辨率来显示网页，字会非常小，安卓手机devicePixoRadio比较乱，有1.5的，有2的也有3的。想让图片在手机里显示更为清晰，必须使用2x的背景图来代替img标签（一般情况下都是2倍的），或者指定background-size:contain;都可以。

用-webkit-min-device-pixel-ratio可以做到不同倍数不同尺寸的图片

# 如何防止手机中页面放大和缩小？

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

# 如何解决上下拉动滚动条时出现卡顿的现象？

body{

-webkit-overflow-scrolling:touch;

overflow-scrolling:touch;}

Android3+和iOSi5+支持CSS3的新属性为overflow-scrolling

# 移动端为什么会有一像素问题？如何解决？

因为在移动端，由于屏幕分辨率的不同，现在分为一倍屏、二倍屏、三倍屏。在不同的分辨率上，有可能一像素被渲然成二个像素点或者三个像素点，所以在实际写代码的时候，我们写的border: 1px solid #000; 可能实际被渲然为 2px/3px；

（1）先使用伪类元素实现边框效果，然后通过媒体查询来操控transform: scale来适配不同分辨率

（2）使用border-image来代替border

# 响应式图片

在移动端中，图片的处理应该是很谨慎的。假设有一张图片本身的尺寸是X宽，设置和包裹它的div一样宽，如果是div宽度小于图片宽度没有问题，但是如果div宽度大于图片的宽度，图片则会被拉伸失真

**解决方法：**让图片最大只能是自己的宽度

img{

max-width: 100%;

display: block;

margin: 0 auto;

}

# 使用了3d做动画之后，3d元素下边的文字失真的问题

给上面动画的幻灯片什么的外层加上绝对定位，定位里面一层加上相对定位。

# 移动端布局怎么做适配

1. **移动端布局方式**百分比（%）

弹性盒布局

rem 布局

#### 弹性盒子（display-flex） 影响：

* 1. 子元素默认横向排列
  2. 行内元素，转换成块级元素
  3. 只有一个子元素，在子元素添加 margin:auto，子元素水平垂直居中。

#### em 与 rem

em：相对单位，相对于父元素的字体大小

rem：相对单位，相对于根元素（html）字体大小

#### vh 与 vw

vh：view-hright 100vh === 视口高度vw：view-width 100vw === 视口宽度100vw 包含滚动条，窗口大小

100% 刨除滚动条，剩余的空间占满

375px = 100vw

1px = 100/375 vw 0.2667

#### meta 标签

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

用户不允许双击缩放： user-scalable=no

1. **多列布局（瀑布流效果）** 属性 用法 属性值

column-count 显示的列数 数值

column-gap 调整列间距 px

column-rule 列边框 2px solid #ff6600

column-fill 列高度统一 balance/auto(把父盒子占满)

column-span（设置在子元素） 横跨所有列 none/all(横跨所有列)

break-inside 禁止盒子内部折行 avoid

1. **rem的计算公式**

(clientWidth/实际图尺寸如640）\*设计图规定大小，如设计图字体20 + “px”

# UI 组件库样式重置---优先级/不加样式作用域/样式穿透

1. /deep/ (Vue 可能不兼容这种写法) 或者 ::v-deep（切记必须是双冒号）

在 style 经常用 scoped 属性实现组件的私有化,所以才需要样式穿透

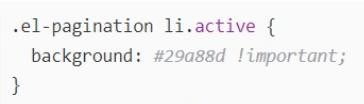
>>> 只作用于 css

::v-deep 只作用于 sass

/deep/ 只作用于 less

1. 在单个 xx.vue 文件最后多写一对<style> </style>标签，即有两对 style 标签，一对有 scoped,一对没有 scoped。

如果写在有 scoped 属性的 style 标签里：覆盖的样式不会生效！

1. 新建 global.css,并在 main.js 中引入 import "./assets/style/global.css";
2. ！important 修饰符提高层级

# Vue计算属性 computed 和 watch 的区别

**computed（计算属性)**

监听其所有依赖的变化，如果有变化会执行，没有变化不执行。有缓存，不用每次重新算。不支持异步

详解：在vue的 模板内（{{}}）是可以写一些简单的js表达式的 ，很便利。但是如果在页面中使用大量或是复杂的表达式去处理数据，对页面的维护会有很大的影响。这个时候就需要用到computed 计算属性来处理复杂的逻辑运算。

1.优点：

在数据未发生变化时，优先读取缓存。computed 计算属性只有在相关的数据发生变化时才会改变要计算的属性，当相关数据没有变化是，它会读取缓存。而不必想 motheds方法 和 watch 方法是否每次都去执行函数。

2.setter 和 getter方法：（注意在vue中书写时用set 和 get）

setter 方法在设置值是触发。

getter 方法在获取值时触发

**watch（侦听属性）:**

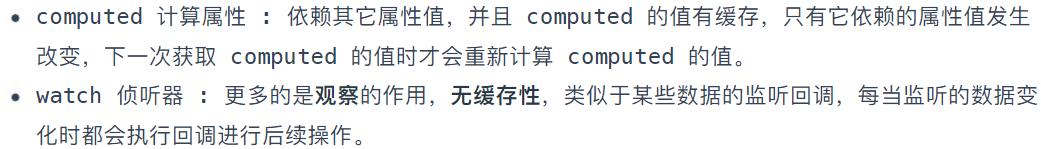
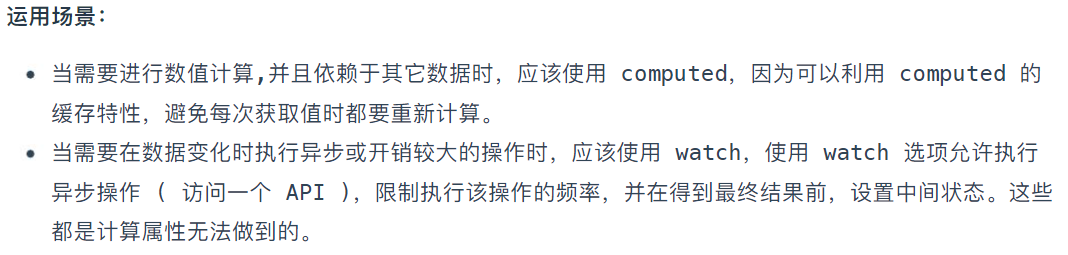
观察某一个变量，发生变化会执行。支持异步。Vue 实例将会在实例化时调用 $watch()，遍历 watch 对象的每一个属性

一个对象，键是需要观察的表达式，值是对应回调函数。值也可以是方法名，或者包含选项的对象。

小结：

1.主动调用的方法写在methods里，依据某些变量的更新进行某种操作用computed或者watch。

2.computed和watch：如果要异步，只能用watch。如果是计算某个值推荐用computed，比如购物车全选单选功能，购物车计算总价小计功能



# 插槽slot是什么？

slot又名插槽，是Vue的内容分发机制，组件内部的模板引擎使用slot元素作为承载分发内容的出口。插槽slot是子组件的一个模板标签元素，而这一个标签元素是否显示，以及怎么显示是由父组件决定的。slot又分三类，默认插槽，具名插槽和作用域插槽。

* 默认插槽∶又名匿名查抄，当slot没有指定name属性值的时候一个默认显示插槽，一个组件内只有有一个匿名插槽。
* 具名插槽︰带有具体名字的插槽，也就是带有name属性的slot，一个组件可以出现多个具名插槽。
* 作用域插槽∶默认插槽、具名插槽的一个变体，可以是匿名插槽，也可以是具名插槽，该插槽的不同点是在子组件渲染作用域插槽时，可以将子组件内部的数据传递给父组件，让父组件根据子组件的传递过来的数据决定如何渲染该插槽。

实现原理︰当子组件vm实例化时，获取到父组件传入的slot标签的内容，存放在vm. $slot中，默认插槽为vm.$slot.default，具名插槽为vm.$slot.xxx，XXx为插槽名，当组件执行渲染函数时候遇到slot标签，使用$slot中的内容进行替换，此时可以为插槽传递数据，若存在数据，则可称该插槽为作用域插槽。

# 字符串逆序

Array对象的reverse()方法

递归

# 数组逆序

通过split和数组的循环

通过split和reverse

通过for循环数组前后调换数据

# 字符串转换为数组的4 个方法

split() 方法

const text = "abc";

const chars = text.split('');

console.log(chars);

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

展开运算符

const text = "abc😎";

const chars = [ ...text ];

console.log(chars);

//["a", "b", "c", "😎"]

解构赋值

const text = "abc😎";

const [ ...chars ] = text;

console.log(chars);

//["a", "b", "c", "😎"]

Array.from

const text = "abc😎";

const chars = Array.from(text);

console.log(chars);

//["a", "b", "c", "😎"]

# 数组转字符串

toString()

join()

var a = [1,2,3,4,5]; //定义数组

var s = a.join(""); //指定分隔符

console.log(s); //返回字符串“12345”

# element-UI二次封装

**InputNumber**

el-input-number 是一个很好用的组件，它只允许用户输入数字值。但是这个组件会有个默认值，给他赋予一个null 或""的时候会显示0

这对于有些业务来说并不是很友好，例如添加页面和编辑页面

并且它这个组件的值是居中显示的，和普通的input 框居左显示不同，这就导致了样式不太统一

**改造：**让 InputNumber 可以居左显示且没有默认值，用法保持和el-input-number组件相似

**OptionPlus**

select 组件用在有较多选项时，但是有些选项的长度难免比较长，就会把选项框整个给撑大，要展示的字段过长时就不太友好。

**改造：**固定选项框的大小，让选项显示更加合理

**FormPlus**

后台系统肯定会有查找功能，搜索条件大部分都是这三种，输入框、下拉框和日期选择。所以可以整合这三个常用的元素，将它们封装成一个易于使用的组件

这三个组件是用来过滤条件的，因此一般与查询和重置按钮在一起

# 按钮的封装

1.所有组件的封装就是接收父组件传递的参数和将数据传递给父组件；

2.单独创建一个vue文件来装封装的组件，组件里面使用到button按钮和作用域插槽；

3.按钮的样式和大小通过使用者传递参数来达到相应效果；

传递不同的属性值，按钮名称和按钮颜色不一样

props: {//调用该组件时需要传递过来的属性

name:{

type:String

},

type:{

type:String

}

# 组件封装

vue组件的功能

1．能够把页面抽象成多个相对独立的模块

2．实现代码重用，提高开发效率和代码质量，使得代码易于维护

Vue组件封装过程

1．首先,使用Vue.extend()创建一个组件2．然后，使用Vue.component()方法注册组件

3．接着，如果子组件需要数据，可以在props中接受定义

4．最后，子组件修改好数据之后，想把数据传递给父组件，可以使用emit()方法

# 浏览器兼容

安装babel包，就是当使用es6语法的代码时，不仅仅可以让谷歌浏览器看，还可以使用ie8以上的浏览器看，其实就是把我们的代码，把es6语法转化为es5语法

# W3C标准

万维网联盟（外语缩写：W3C）标准不是某一个标准，而是一系列标准的集合。网页主要由三部分组成：结构（Structure）、表现（Presentation）和行为（Behavior）。

web可以简单分为：结构、表现、行为。三部分独立开来使其模块化

1. 结构化标准语言主要包括XHTML和XML
2. 表现标准语言主要包括CSS
3. 行为标准主要包括对象模型（如W3C DOM）、ECMAScript

w3c是对web做出规范，使代码更严谨，做出来的网页更易使用，维护。

w3c做出的规范可分为如下：

**结构上：**（标签规范对页面的搜索权重有很大关系，写的越规范网站在搜索排名越靠前）

1. 标签闭合、标签小写、不乱嵌套
2. 需要声明（DOCTYPE）
3. 需要定义语言编码<meta http-equiv=“Content-Type” content=“text/html; charset=gb2312” />

**表现、行为上：**

1. 使用外链的css和js脚本，提高页面渲染效率。
2. Js必须要用<script language=“javascript” type=“text/javascript”>来开头定义，而不是原来的<script language=javascript>或干脆直接<script>
3. 并且需要加个注释符<!-- -->，以保证不在不支持js的浏览器上直接显示出代码来（做了解）
4. 少使用行内样式，类名要做到见名知意
5. CSS必须要用<style type=“text/css”>开头来定义，而不是原来的直接<style>
6. 也不建议直接写在内容代码里（行内样式）如：<div style=“padding-left:20px;”></div>
7. 为保证各浏览器的兼容性，在写CSS时请都写上数量单位
8. 其他参考https://blog.csdn.net/weixin\_48594833/article/details/121326752

# 前端调试的方法

1. log(...)
2. vscode 调试
3. chrome 插件调试【debugger】
4. 浏览器中断点
   1. console.log()打印的时候，后面会出现打印的具体文件位置
   2. 点击之后会跳到这个文件的具体位置。之后设置断点
   3. 之后按F5刷新断点调试了，程序运行到这个地方会被卡住，然后你就可以查看当前的环境中的一些信息
5. 代码中的断点
   1. 在代码中直接使用 debugger 关键字可以快速的实现断点。
   2. 效果和浏览器中的断点效果是一样的。
6. 性能调试工具performance，使用Chrome DevTools的performance面板可以记录和分析页面在运行时的所有活动

# 虚拟 dom 比对--用的是 diff 算法

Diff 算法的作用:diff 算法**就是对虚拟 dom 进行对比**，并**返回一个 patch 对象**，这个对象的作用是存储两个节点不同的地方， 最后**用 patch 里记录的信息去局部更新真实的 dom**

Diff 算法主要首先通过：

* setter，中捕获数据的更新
* 随后通过 Dep.notify，通知对应的订阅者 watcher，
* 调用 patch 方法 ，给真实 DOM 打补丁，更新对应的试图
* patch 方法会判断同层节点是否为相同类型
* 否就直接返回新节点
* 是就进行 patchVnode 方法 进行深层对比
* patchVnode 中又会进行五层对比
* 找到对应的真实 DOM，称为 el
* 判断 newVnode 和 oldVnode 是否指向同一个对象，如果是，那么直接 return
* 如果他们都有文本节点并且不相等，那么将 el 的文本节点设置为 newVnode 的文本节点。
* 如果 oldVnode 有子节点而 newVnode 没有，则删除 el 的子节点
* 如果 oldVnode 没有子节点而 newVnode 有，则将 newVnode 的子节点真实化之后添加到 el
* 如果两者都有子节点，则执行 updateChildren 函数比较子节点，这一步很重要
* updateChildren 方法利用首尾指针进行比较
* 若有 key 时则使用 key 进行比较

# 难点总结

## 大文件上传和断点续传

### 思路：

**前端：**

大文件上传网上的大部分文章已经给出了解决方案，核心是利用Blob.prototype.slice 方法，和数组的 slice 方法相似，调用的 slice 方法可以返回原文件的某个切片

这样我们就可以根据预先设置好的切片最大数量将文件切分为一个个切片，然后借助 http 的可并发性，同时上传多个切片，这样从原本传一个大文件，变成了同时传多个小的文件切片，可以大大减少上传时间

另外由于是并发，传输到服务端的顺序可能会发生变化，所以我们还需要给每个切片记录顺序

**服务端:**

服务端需要负责接受这些切片，并在接收到所有切片后合并切片

### 实现：

当点击上传按钮时，调用createFileChunk 将文件切片，切片数量通过文件大小控制，这里设置10MB，也就是说100 MB 的文件会被分成10个切片，在createFileChunk 内使用 while 循环和 slice 方法将切片放入数组中返回。

在生成文件切片时，需要给每个切片一个标识作为 hash ，可以使用文件名＋下标，这样后端可以知道当前切片是第几个切片，用于之后的合并切片

随后将所有的文件切片，将文件切片，切片 hash ，以及文件名放入 FormData 中上传，再调用上一步的 request 函数返回一个 proimise ，最后调用 Promise.all 并发上传所有的切片。前端发送合并请求，通知后端进行切片合并

**上传进度条**

整个文件的上传进度是基于每个切片上传进度计算而来的，将每个切片已上传的部分累加，除以整个文件的大小，就能得出当前文件的上传进度

**断点续存**

断点续传的原理在于前端／服务端需要记住已上传的切片，这样下次上传就可以跳过之前已上传的部分，有两种方案实现记忆的功能

●前端使用 localStorage 记录已上传的切片hash

●服务端保存已上传的切片 hash ，前端每次上传前向服务端获取已上传的切片

第一种是前端的解决方案，第二种是服务端，而前端方案有一个缺陷，如果换了个浏览器就失去了记忆的效果，所以这里选取后者

**文件秒传**

所谓的文件秒传，即在服务端已经存在了上传的资源，所以当用户再次上传时会直接提示上传成功

文件秒传需要依赖上一步生成的 hash ，即在上传前，先计算出文件 hash ，并把 hash 发送给服务端进行验证，由于 hash 的唯一性，所以一旦服务端能找到 hash 相同的文件，则直接返回上传成功的信息即可

秒传其实就是给用户看的障眼法，实质上根本没有上传

## 数据埋点

google analytics 和 自定义埋点，调用api

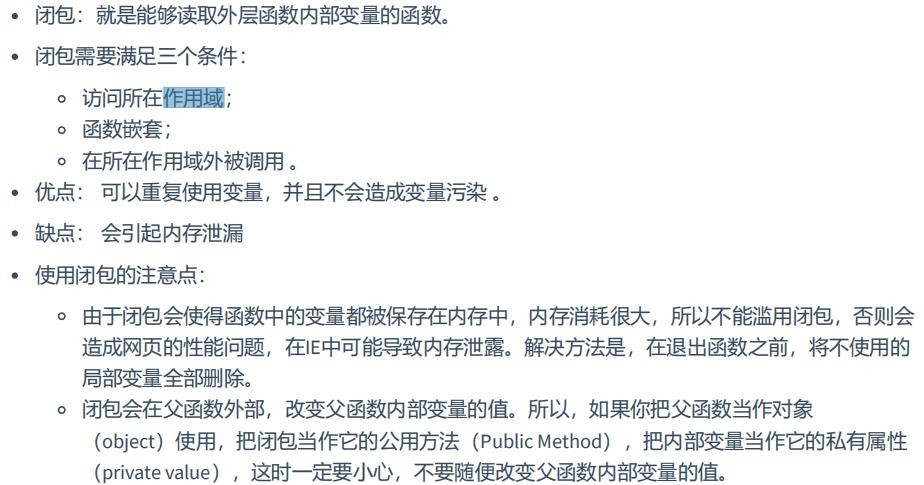
按钮埋点 把actions 和 类型传给后端，大数据对类型进行分类，用户用户转换率优化

自定义埋点：就是定义一个方法。然后把action 和 类型（和大数据约定好） 传给后端，然后后端进行统计

**埋点如何进行数据收集：**

单页面中可以通过鼠标的移入移出来处理，移入时定义一个时间，移出又是一个时间，这中间就是我们想要收集的用户的停留时间，在多页面中我们可以通过监听路由来进行判断用户停留的时间来收集用户的留存度

# 什么是闭包



应用场景：

节流和防抖

发送ajax请求成功|失败的回调

setTimeout的延时回调

一个函数内部返回另一个匿名函数

**原型和原型链**



**reactive与ref区别**

**1、 从定义数据方面：**

**ref通常用来定义基本类型数据**

**reactive用来定义：对象（或者数组）类型数据**

**ref也可以用来定义对象或者数组类型的数据，内部会通过reactive转为代理对象**

**2、从原理方面：**

**ref通过Object.defineProperty()的get和set实现数据代理。**

**reactive使用Proxy实现数据代理，并且通过Reflect操作源对象内部的数据。**

**3、从使用方面：**

**ref操作数据需要.value，template模板中不需要。**

**reactive都不需要,value**

**————————————————**

**版权声明：本文为CSDN博主「一路向阳~负责的男人」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。**

**原文链接：https://blog.csdn.net/weixin\_43416349/article/details/120360330**

**Js 作用域**

作用域就是在代码执行过程中，形成一个独立的空间，让空间内的变量不会邪泄露在空间外，也让独立空间内的变量函数在独立空间内运行，而不会影响到外部的环境。

作用域分为全局作用域和局部作用域，也就是本来有一个巨大的空间，空间内定义的函数内部，就形成了一个独立的小空间， 全局作用域是最大的作用域。

但是当独立空间内的数据不能满足需求时，是可以从外部获取数据的，也就是说这样的独立空间之间是可以有层级关系的，外部的空间不可以从内部的空间获取数据，但内部的空间可以。当子级空间在父级空间中获取数据的时，父级空间没有的话，父级空间也会到他的父级空间中查找数据，这样形成的链式结构叫作用域链。

当将一个变量当做值使用时，会先在当前作用域中查找这个变量的定义和数据，如果没有定义的话，就会去父级作用域中查找， 如果父级作用域中有的话就使用这个值，如果父级作用域中也没有的话，就通过父级作用域查找他的父级作用域，直到找到最大的作用域-全局，如果全局也没有就报错。

当将一个变量当做数据容器存储，也就是给变量赋值的时候，也要先在自己作用域中查找变量的定义，如果没有就在上一级作用域中查找，直到全局，如果全局作用域中也没有这个变量的定义，就在全局定义这个变量并赋值。

# var、let、const的区别

#### var 声明作用域

使用 var 操作符定义的变量会成为包含它的函数的局部变量。比如，使用 var 在一个函数内部定义一个变量，就意味着该变量将在函数退出时被销毁

#### let

let 跟 var 的作用差不多，但有着非常重要的区别。最明显的区别是，let 声明的范围是块作用域， 而 var 声明的范围是函数全局作用域。这也是 JavaScript 中的新概念。块 级作用域由最近的一对包含花括号{}界定。换句话说，if 块、while 块、function 块，甚至连单独的块也是 let 声明变量的作用域。

#### Const

const 的行为与 let 基本相同，唯一一个重要的区别是用它声明变量时必须同时初始化变量，且 尝试修改 const 声明的变量会导致运行时错误,可以修改对象

# 跨域是什么？有哪些解决跨域的方法和方案？

## 话术：掘金优秀文章看到过其他解决办法,但是我一般用主流的解决办法 cors 和代理我一般在开发时用代理,上线时用 cors

#### 什么是跨域?

所谓的同源是指，域名、协议、端口均为相同。 所谓的跨域，不同的域名、协议、端口皆为不同域

一个域与另一个域名、协议或者端口不同的域的之间访问都叫跨域

**解决跨域的方法和方案：**

1. **CORS**：跨域资源共享，是一种实现跨源请求数据的技术。这就是跨源问题的解决方案之一。也是广泛 的解决方案。

工作原理： 服务器 在返回响应报文的时候，在响应头中 设置一个允许跨域[header](https://so.csdn.net/so/search?q=header&amp;spm=1001.2101.3001.7020)

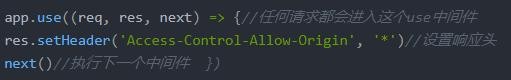
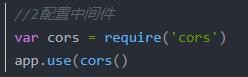
CORS 是一个 W3C 标准，全称是"跨域资源共享"（Cross-origin resource sharing）

它允许浏览器向跨源服务器，发出 XMLHttpRequest 请求，从而克服了 AJAX 只能同源使用的限制整个CORS 通信过程，都是浏览器自动完成，不需要用户参与

对于开发者来说，CORS 通信与同源的 AJAX 通信没有差别，代码完全一样

实现CORS 通信的关键是服务器，只要服务器实现了 CORS 接口，就可以跨源通信epress 有一个自带的中间件 cors，它的作用是自动给每一个 res 设置默认请求头

这样就不用我们自己每一个接口都要设置一次了



1. **设置 proxy 代理** ，在代码里写需要代理的网址，在文件里写代码



1. jsonp 跨域
   1. jsonp 跨域就是利用 script 标签的跨域能力请求资源
   2. jsonp 与 ajax 没有半毛钱关系！！
   3. 浏览器的同源策略限制了 js 的跨域能力，但没有限制 link img iframe script 的跨域行为实现方式：
2. 利用 js 创建一个 script 标签，把 json 的 url 赋给 script 的 scr 属性，
3. 2. 把这个 script 插入到页面里，让浏览器去跨域获取资源
4. JS 先声明好回调函数，插入页面后会代为执行该函数，并且传入 json 对象为其参数。注意：
5. jsonp 只针对 get 请求
6. script 标签加载回来的资源会被当成 js 在全局执行

4：nginx 代理跨域

通过 nginx 服务器转发跨域请求，达到跨域的目的

**区别：**

1. CORS：

服务器返回响应头，前端无需任何处理简单快捷，支持所有请求方式

1. JSONP

浏览器：自定义响应回调函数，使用 script 标签的 src 请求利用浏览器的 src 属性没有跨域这一限制特点

服务器：接收 callback 参数，返回函数调用处理复杂，并且只支持 get 请求

原因：get 请求参数直接在 url 后面拼接，而 post 请求参数是放在请求体中

# 防抖和节流

**防抖(debounce):**

防抖触发高频率事件时n秒后只会执行一次,如果n秒内再次触发,则会重新计算。

概述:每次触发时都会取消之前的延时调用。

**节流(thorttle):**

高频事件触发,每次触发事件时设置一个延迟调用方法，并且取消之前的延时调用方法。

概述:每次触发事件时都会判断是否等待执行的延时函数。

**区别:**降低回调执行频率,节省计算资源。

函数防抖一定时间连续触发的事件,只在最后执行一次,而函数节流一段时间内只执行一次。

# settimeout和setinterval的区别

setTimeout只运行一次，也就是说设定的时间到后就触发运行指定代码，运行完后即结束；

setinterval是一直循环运行下去，即每到设定时间间隔就触发指定代码，要想停止，需要使用clearInterval()函数

# 常见的 http 网络请求方式

#### GET（SELECT）：从服务器取出资源（一项或多项）。

* **POST（CREATE）：在服务器新建一个资源。**
* **PUT（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源）。**
* PATCH（UPDATE）：在服务器更新资源（客户端提供改变的属性）。

#### DELETE（DELETE）：从服务器删除资源。

* HEAD：获取资源的元数据。
* OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的。

#### 1、GET 方法

用于使用给定的 URI 从给定服务器中检索信息，即从指定资源中请求数据。使用 GET 方法的请求应该只是检索数据,并且不应对数据产生其他影响。

在 GET 请求的 URL 中发送查询字符串(名称/值对)，需要这样写：/test/demo\_form.php?name1=value1&name2=value2 说明：GET 请求是可以缓存的，我们可以从浏览器历史记录中查找到 GET 请求，还可以把它收藏到书签中;且 GET 请求有长度限制，仅用于请求数据(不修改)。

注：因 GET 请求的不安全性，在处理敏感数据时，绝不可以使用 GET 请求。

2、HEAD 方法

与 GET 方法相同，但没有响应体，仅传输状态行和标题部分。这对于恢复相应头部编写的元数据非常有用，而无需传输整个内容。

#### 3、POST 方法

用于将数据发送到服务器以创建或更新资源，它要求服务器确认请求中包含的内容作为由 URI 区分的 Web 资源的另一个下属。

POST 请求永远不会被缓存，且对数据长度没有限制;我们无法从浏览器历史记录中查找到 POST 请求。

#### 4、PUT 方法

用于将数据发送到服务器以创建或更新资源，它可以用上传的内容替换目标资源中的所有当前内容。

它会将包含的元素放在所提供的 URI 下，如果 URI 指示的是当前资源，则会被改变。如果 URI 未指示当前资源，则服务器可以使用该 URI 创建资源。

#### 5、DELETE 方法

用来删除指定的资源，它会删除 URI 给出的目标资源的所有当前内容。

6、CONNECT 方法

用来建立到给定 URI 标识的服务器的隧道;它通过简单的 TCP/IP 隧道更改请求连接，通常实使用解码的 HTTP 代理来进行 SSL 编码的通信(HTTPS)。

7、OPTIONS 方法

用来描述了目标资源的通信选项，会返回服务器支持预定义 URL 的 HTTP 策略。

8、TRACE 方法

用于沿着目标资源的路径执行消息环回测试;它回应收到的请求，以便客户可以看到中间服务器进行了哪些(假设任何)进度或增量

以上介绍了 HTTP 的 8 种请求方式，其中常用的是 GET 和 POST。可以说，GET 是从服务器上获取数据，POST 是向服务器传送数据，至于选择哪种，就需要根据实际情况来选择了

#### get 和 post 的区别

1. GET 把参数包含在 URL 中，POST 通过 request body 传递参数所以 get 的参数长度有限制，而 post 无限制
2. GET 在浏览器回退时是无害的，而 POST 会再次提交请求。
3. GET 请求只能进行 url 编码，而 POST 支持多种编码方式。
4. GET 请求参数会被完整保留在浏览器历史记录里，而 POST 中的参数不会被保留。
5. 对参数的数据类型，GET 只接受 ASCII 字符，而 POST 没有限制
6. GET 比 POST 更不安全，因为参数直接暴露在 URL 上，所以不能用来传递敏感信息（相对安全，因为安全更加不加密有关）

# promise 重构

* Pending---Promise 对象实列创建时候的初始状态
* Fulfilled---可以理解为成功的状态
* Rejected---可以理解为失败的状态

#### promise 的链式调用

每次调用返回的都是一个新的 Promise 实列（这就是 then 可用,链式调用的原因)如果 then 中返回的是一个结果的话，会把这个结果传递下一次 then 中的成功回调；如果 then 中出现异常，会走下一个 then 的失败回调，在 then 中使用 return,那么 return 的值会被 Promise.resolve()包装then 中可以不传递参数，如果不传递会透到下一个 then 中catch 会捕获到没有捕获的异常

Promise构造函数的原型对象上，有then()和catch()等方法，then()第一个参数接收resolved()传来的数据，catch()第一个参数接收rejected()传来的数据

# promise 所有的静态方法

### 我只常用两个,其他知道,但不太了解,区别

实例方法

promise.then()：获取异步任务的正常结果。promise.catch()：获取异步任务的异常结果。promise.finaly()：异步任务无论成功与否，都会执行。 对象方法

#### resolve 方法

(1). Promise.resolve 的用法相当于 new Promise，并且执行 resolve 操作. (2). resolve 方法参数问题：

A：参数是一个普通的值或者对象B：参数本身是 Promise

C：参数是一个 thenable

#### reject 方法

Promise.reject 的用法相当于 new Promise，只是会调用 reject 操作

特别注意：Promise.reject 传入的参数无论是什么形态，都会直接作为 reject 状态的参数传递到 catch 的

#### promise.all (): 并发处理多个异步任务，所有任务都执行成功，才能得到结果

Promise.all 方法接受一个参数，参数类型必须是一个数组，数组里面每一项都是一个 promise 实例，该方法会返回一个新的 promise 实例

每一个实例的状态如果都是成功 fulfilled 状态，新返回的 promise 实例 newProm 会执行 then 方法，then 方法里面res 是每一个实例的终值，res 是一个数组，数组里面每一个终值的顺序由我们在调用 promise.all 传递的参数在数组顺序决定，不受每个实例获取终值的时间决定。

只要有一个实例的状态是失败 reject，新返回的 promise 实例会执行 catch 方法，该方法里面能够获取 err,err 就是最快出错实例的拒因

应用场景

项目中某个操作依赖于两个或者多个请求的结果再进行处理

比如：首页需要等所有接口数据都返回才会显示页面，否则都会显示骨架图。**promise.race (): 并发处理多个异步任务，只要有一个任务执行成功，就能得到结果**语法

Promise.race 方法接受一个参数，参数类型必须是一个数组，数组里面每一项都是一个 promise 实例，该方法会返回一个新的 promise 实例

race 方法不会关注 promise 实例的状态，只会关注哪个实例先有结果，这个结果可能是一个终值，也有可能是一个拒因

应用场景

项目里面测试 cdn 的速度，就是项目里面接口会部署几个服务器，我们获取最快的服务器地址

fetch 请求超时设置取消请求

#### Promise.allSettled

语法

Promise.allSettled 方法接受一个参数，参数类型必须是一个数组，数组里面每一项都是一个 promise 实例，该方法会返回一个新的 promise 实例

**allSettled 方法无论每一个实例的状态是成功还是失败，都会执行 then 方法，拿到所有实例的结果，返回的 res 也是一个数组，顺序也是由参数的位置决定，不受实例执行时间影响。**

返回的 res 里面包含每一个实例结果的状态 status，如果是成功返回 fulfilled，如果是失败返回 reject

如果是成功返回终值 value，如果是失败返回拒因 reason

应用场景

项目里面同时请求多个接口，无论接口是成功还是失败我们都要拿到多个接口的返回值

#### Promise.any

语法

#### Promise.any 方法接受一个参数，参数类型必须是一个数组，数组里面每一项都是一个 promise 实例，该方法会返回一个新的 promise 实例

注意：此方法需要 node 高版本支持，建议在浏览器环境测试。

any 方法只要有一个实例状态成功就会执行 then 方法，得到这个实例的终值，这一点和 race 相同和 race 不同的是它不会因为一个实例失败就执行 catch，它会等所有实例都失败才执行 catch 方法。应用场景

#### Promise.any 作用和 race 几乎相同，暂未发现异同之处

# 什么是Async/Await

① async/await是ES7新特性

② async/await是写异步代码的新方式，以前的方法有回调函数和Promise

③ async/await是基于Promise实现的，它不能用于普通的回调函数

④ async/await与Promise一样，是非阻塞的

⑤ async/await使得异步代码看起来像同步代码，这正是它的魔力所在

await 操作符用于等待一个 Promise 对象。它只能在异步函数 async function 中使用。

await 右侧的表达式一般为 promise 对象, 但也可以是其它的值

如果表达式是 promise 对象, await 返回的是 promise 成功的值

如果表达式是其它值, 直接将此值作为 await 的返回值

await 必须写在 async 函数中, 但 async 函数中可以没有 await，如果 await 的 Promise 失败了, 就会抛出异常, 需要通过 try...catch 捕获处理

# promise和async/await的区别

1. 使用async函数可以让代码简洁很多，不需要像Promise一样需要些then，不需要写匿名函数处理Promise的resolve值，也不需要定义多余的data变量，还避免了嵌套代码。
2. Promise的出现解决了传统callback函数导致的“地域回调”问题，但它的语法导致了它向纵向发展行成了一个回调链，遇到复杂的业务场景，这样的语法显然也是不美观的。而async await代码看起来会简洁些，使得异步代码看起来像同步代码，await的本质是可以提供等同于”同步效果“的等待异步返回能力的语法糖，只有这一句代码执行完，才会执行下一句。
3. async await与Promise一样，是非阻塞的。
4. async await是基于Promise实现的，可以说是改良版的Promise，它不能用于普通的回调函数。

**Event Loop ( 事件循环 )**

事件循环分为**同步任务**和**异步任务**；所有同步任务都在主线程上执行，形成一个函数调用栈（执行栈），而异步则先放到**任务队列**（**task queue**）里，任务队列又分为**宏任务**（macro-task）与**微任务**（micro-task）。

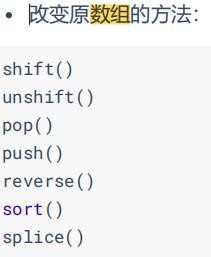
**宏任务大概包括**：：script（整块代码）、setTimeout、setInterval、I/O、UI 交互事件、setImmediate（node 环境）

**微任务大概包括**：：new promise().then(回调)、MutationObserver(html5 新特新)、Object.observe(已废弃)、process.nextTick

（node 环境）

若同时存在promise 和 nextTick，则先执行 nextTick

# Es6 中的数组方法



shift删头 un增头

pop删尾 push增尾

**ES6 新特性**

1. const 与 let
2. 模板字符串
3. 解构赋值
4. 对象简写法
5. for...of 循环
6. 展开运算符
7. 剩余参数(可变参数)
8. ES6 箭头函数
9. 数组去重

ES6提供了特别简单的去重方法Set对象

var arrSet=[1,1,1,3,3,2,2]

let arry1=Array.from(new Set([1,1,1,3,3,2,2]))

console.log(arry1)

//输出 [1,3,2]

1. 参数默认
2. 类和继承
3. 模块化规范

# 箭头函数和普通函数的区别

1. 相比普通函数更简洁的语法
2. 没有 this,捕获其所在上下文的 this 值，作为自己的 this 值
3. 不能使用 new,箭头函数作为匿名函数,是不能作为构造函数的,不能使用 new
4. 不绑定 arguments，用 rest 参数...解决

let test3=(...a)=>{console.log(a[1])} //22

1. 使用 call()和 apply()调用:由于 this 已经在词法层面完成了绑定，通过 call() 或 apply() 方法调用一个函数时，只是传入了参数而已，对 this 并没有什么影响：
2. 箭头函数没有原型属性
3. 不能简单返回对象字面量

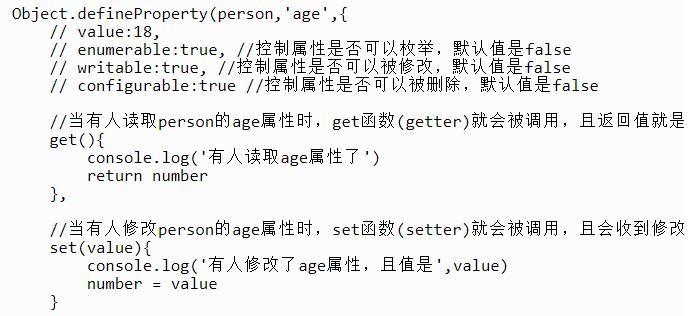
let fun5 = ()=>({ foo: x }) //如果 x => { foo: x } //则语法出错

1. 箭头函数不能当做 Generator 函数,不能使用 yield 关键字

# 数据代理

通过一个对象代理对另一个对象中的属性的操作（读/写），就是数据代理。

#### Object.defineProperty 方法



**Vue 中数据代理**

vue 数据代理： 通过 vm 对象来代理 data 中所有属性的操作，更方便操作 data 中的数据

1. 通过 Object.defineProperty()给 vm 添加与 data 对象的属性对应的属性描述符
2. 所有添加的属性都包含 getter/setter，getter/setter 内部去操作 data 中对应的属性数据

# Vue3 中的反射reflact----有了解 但没用过 ,在vue 源码中看到过,之前找一个源码意外看到过

Reflect 其实和 Proxy 一样都是属于 ES6 的高级API，Reflect 也是属于 window 的一个内置类，可以通过 window.Reflect

访问到

Proxy 要配合 Reflect 一起使用，用来 触发代理对象的劫持时 保证正确的 this 上下文指向

## 面试官问的,回答不能停顿的问题:

1. 代码规范
   1. 严格按照公司开发文档
   2. 按照eslint 开发规范，差不多
2. 公司用的开发工具
3. git 上传

**webpack 五个核心概念**

1. Entry

入口(Entry)指示 webpack 以哪个文件为入口起点开始打包，分析构建内部依赖图。

1. Output

输出(Output)指示 webpack 打包后的资源 bundles 输出到哪里去，以及如何命名。

1. Loader

webpack 只能理解 JavaScript 和 JSON 文件，这是 webpack 开箱可用的自带能力。loader 让 webpack 能够去处理其他类型的文件，并将它们转换为有效 模块，以供应用程序使用，以及被添加到依赖图中。

1. Plugins

插件(Plugins)可以用于执行范围更广的任务。插件的范围包括，从打包优化和压缩，一直到重新定义环境中的变量等。

1. Mode

模式(Mode)指示 webpack 使用相应模式的配置。

# webpack 流程

1. 局部安装 webpack
2. 在 package.json 文件中配置 webpack 运行脚本命令
3. 在项目根目录下面创建一个 webpack.config.js 文件，此文件是 webpack 的默认配置文件名称，后续就可以在此目录下面对 webpack 进行运行打包配置。

const path = require('path')

module.exports = {

// 打包模式 development | production mode: 'development',

// 项目入口

entry: './src/index.js',

// 项目出口

output: {

// 路径一定要用绝对路径

path: path.resolve('dist'),

// [name]默认的名称为 main

// [hash]随机生成 21 位字符串

filename: 'js/[name].js'

}

}

1. 在项目根目录下面创建一个 public 目录， 并在此目录中创建一个 index.html 文件， 使用 webpack 的

html-webpack-plugin 插件()，可以将此文件用于项目 web 入口文件模块

# 安装

npm i -D html-webpack-plugin # 引入

const HtmlWebpackPlugin = require('html-webpack-plugin') # 使用 webpack.config.js

plugins: [

new HtmlWebpackPlugin({

template: path.resolve('public/index.html'),

})

]

# 删除打包的 dist 目录

npm i -D clean-webpack-plugin@4

// 在下次打包时主动删除上一次的打包目录

const { CleanWebpackPlugin } = require('clean-webpack-plugin')

plugins: [

new CleanWebpackPlugin()

]

## Ts相比js的优点（ts的优点）

* 继承了js的所有编程类型，js代码可在ts环境很好的运行
* 为构建大型应用而生，但小程序同样适用
* 强大的类型系统，拥有静态类型检查能力
* 新增类型注解和类型推断
* 拥有丰富的class扩展功能
* 添加了系统级设计能力，设计模式由顶层由下进行设计

# 自定义指令

vue2中自定义指令常用钩子函数有：

* bind：在指令第一次绑定到元素时调用
* inserted：被绑定元素插入父节点时调用 (仅保证父节点存在，但不一定已被插入文档中)
* update：数据更新时调用，自己的数据更新后立刻执行
* componentUpdate 当它的子元素中所有的数据都更新后执行
* unbind 解绑

请注意：不管在定义全局还是局部自定义指令时，所提及的指令名均是不带v-前缀的名称。

在vue中以v-开头属性，都为指令，如果不是内置的，则需要自定义

自定义指令分为全局和局部

Vue.directive( 参数1，参数2 ）

参数1：自定义指令的名称，不用写v-，直接写后面的名称

参数2: 两种写法

○ 对象：{ } [标准方式，有5个钩子函数]

○ 回调函数： ()=>{ } 【它是bind和update钩子函数的简写】

bind(el,binds) //el 当前元素 binds 传的数据

# vue路由钩子函数

全局前置守卫 router.beforeEach

全局解析守卫 router.beforeResolve

在 ****每次导航****时都会触发，但是确保在导航被确认之前，****同时在所有组件内守卫和异步路由组件被解析之后，解析守卫就被正确调用****。

全局后置钩子 router.afterEach

路由独享的守卫 beforeEnter

只在进入路由时触发，不会在 params、query 或 hash 改变时触发

只有在 ****从一个不同的**** 路由导航时，才会被触发

组件内的守卫 beforeRouteEnter beforeRouteUpdate beforeRouteLeave

# 图片懒加载

懒加载也叫做延迟加载、按需加载，指的是在长网页中延迟加载图片数据，是一种较好的网页性能优化的方式

图片的加载是由src引起的，当对src赋值时，浏览器就会请求图片资源。根据这个原理，我们使用HTML5 的data-xxx属性来储存图片的路径，在需要加载图片的时候，将data-xxx中图片的路径赋值给src，这样就可以实现图片的按需加载，即懒加载

**知识点**

1. window.innerHeight 是浏览器可视区的高度。
2. document.body.scrollTop || document.documentElement.scrollTop 是浏览器滚动的过的距离。
3. imgs.offsetTop 是元素顶部距离文档顶部的高度（包括滚动条的距离）。
4. 图片加载条件：img.offsetTop < window.innerHeight + document.body.scrollTop。

# 前端页面白屏的解决方案

1. 路由懒加载

// 1、Vue异步组件技术：

{

path: '/home',

name: 'Home',

component: resolve => require(['../views/home.vue'], resolve)

}

// 2、es6提案的import()

{

path: '/',

name: 'home',

component: () => import('../views/home.vue')

}

// 3、webpack提供的require.ensure()

{

path: '/home',

name: 'Home',

component: r => require.ensure([],() => r(require('../views/home.vue')), 'home')

}

1. 组件懒加载

// import 方式

components:{

"dailyModal":()=>import("./dailyModal.vue")

},

// require 方式

components:{

"dailyModal":resolve=>require(['./dailyModal.vue'],resolve)

},

1. CDN 资源优化

将 vue、vue-router、vuex、axios 这些 vue 全家桶的资源，全部改为通过 CDN 链接获取，在 index.html 里插入相应链接。

1. 首页加 loading或 骨架屏

# ts中遇到的问题

1. ts开发过程中类型指定问题

例如：

const c: number = statistics.getLikeCount()

这样的类型指定是不允许的，提示后面返回值有可能是undefined，所以不能直接指定为number,所以需要考虑到undefined类型

解决方法：

const c: number = statistics.getLikeCount() || 0  – 这样即可

或

const c = statistics.getLikeCount()

let c2 = c == null ? c : 0

1. ts开发小程序中设置data值

js版本设置data中的数据的语法：

this.setData({})

但是在ts中，这样写的话，会一直会有报错，表示可能会设置undefined值到data中，所以需要指定设置的data值为非空，才可以通过编译

解决方法：

//第一种方法： 在设置值的时候，加上！标示非空的数据

this.setData!({})

// 第二种方法：在上这一行上面加上`//@ts-ignore` 表示跳过ts检查， 注意：前面的// 是需要的

//@ts-ignore

\_this.setData({

})

# TS 中 any 和 unkown 的区别

unknown 和 any 的主要区别是 unknown 类型会更加严格：在对 unknown 类型的值执行大多数操作之前，我们必须进行某种形式的检查。而在对 any 类型的值执行操作之前，我们不必进行任何检查。

总结： any 和 unknown 都是顶级类型，但是 unknown 更加严格，不像 any 那样不做类型检查，反而 unknown 因为未知性质，不允许访问属性，不允许赋值给其他有明确类型的变量

# mounted 和 nextTick 的区别

Vue.nextTick()

在Vue[生命周期](https://so.csdn.net/so/search?q=%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/tempta36/article/details/_blank)的created()函数中进行DOM操作是要放在Vue.nextTick()回调函数中。原因就是created()在执行的时候DOM只是创建出来并没有渲染在页面上，因此页面上没有这个元素也就无法对其进行操作。

当页面进行某个操作后，部分DOM随数据的改变而改变，这个时候就应该将这个操作放在Vue.nextTick()的[回调](https://so.csdn.net/so/search?q=%E5%9B%9E%E8%B0%83&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/tempta36/article/details/_blank)函数中执行。

mounted()

mounted()函数执行时所有的DOM的创建，布局和渲染都已完成

# vue2和vue3的区别

1. 响应式系统升级

vue2 中的 object.defineProperty(响应式系统的核心)

初始化的时候，会遍历data中的所有成员，使用defineProperty把对象的属性转换成get和set，如果该成员中的属性还是对象的话，需要递归处理每一个子对象的属性。这些都是在初始化的时候进行的。即使你没有使用到这个属性，那么也是会将该属性进行响应式处理。

**缺点：**无法监听对象或数组新增、删除的元素

**Vue2 方案：**针对常用数组原型方法push、pop、shift、unshift、splice、sort、reverse进行了hack处理；提供Vue.set监听对象/数组新增属性。对象的新增/删除响应，还可以new个新对象，新增则合并新属性和旧对象；删除则将删除属性后的对象深拷贝给新对象

基本用法：直接在一个对象上定义新的属性或修改现有的属性，并返回对象

****Tips：****writable 和 value 与 getter 和 setter 不共存

vue3 proxy

proxy的性能本身就比defineProperty要好， 另外，代理对象可以拦截属性的访问、赋值、删除等操作，不需要初始化时遍历所有的属性。另外，如果有多层属性嵌套的话，只有访问某个属性的时候，才会递归处理下一级的属性。使用proxy默认就可以监听到动态新增的属性，可以监听到删除的属性。可以监听数组的索引和length属性。

IE11以下无法兼容

总结： vue3使用的proxy对象提升了响应式系统的性能和功能

1. 模板语法--多根节点

打开App.vue文件，唯一和Vue2不同的是：Template不再强调一定要有根标签了

1. 生命周期函数

Created和beforecreate=》

1. setup函数

Vue2 是 选项式API（Option API），一个逻辑会散乱在文件不同位置（data、props、computed、watch、生命周期函数等），导致代码的可读性变差，需要上下来回跳转文件位置

Vue3 组合式API（Composition API）则很好地解决了这个问题，可将同一逻辑的内容写到一起。

但是在vue3中都是在setup上,setup函数可以说是Vue3的属性和方法入口

* setup(props,context):接收两个参数

(1) props ：接收来自父组件传来的参数

(2) context ：上下文，主要包含3个使用参数：attrs,emits,slots，相当于Vue2中this的attrs,emits,slots

* 有返回值
* 函数内部没有this
* 当内部有异步函数，需要使用到await的时候，可以直接使用，不需要在setup前面加async

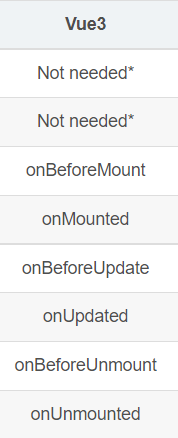
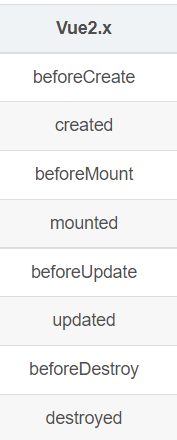
1. Vue3 由TS重写，相对于 Vue2 有更好地TypeScript支持。

Vue2 Option API中 option 是个简单对象，而TS是一种类型系统，面向对象的语法，不是特别匹配。

Vue2 需要vue-class-component强化vue原生组件，也需要vue-property-decorator增加更多结合Vue特性的装饰器，写法比较繁琐

# vue3生命周期的变化

整体来看，变化不大，只是名字大部分需要 + on，功能上类似。使用上 Vue3 组合式 API 需要先引入；Vue2 选项 API 则可直接调用，如下所示。



# 多根节点

Vue3 支持了多根节点组件，也就是fragment。

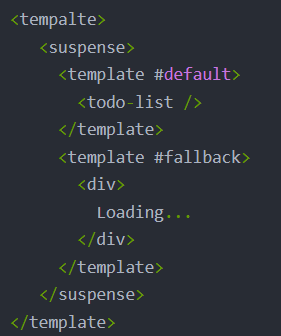
Vue2中，编写页面的时候，在template中，我们需要去将组件包裹在div中，否则报错警告。

# 异步组件

Vue3 提供 Suspense组件，允许程序在等待异步组件时渲染兜底的内容，如 loading ，使用户体验更平滑。使用它，需在模板中声明，并包括两个命名插槽：default和fallback。Suspense确保加载完异步内容时显示默认插槽，并将fallback插槽用作加载状态

真实的项目中踩过坑，若想在 setup 中调用异步请求，需在 setup 前加async关键字。这时，会受到警告async setup() is used without a suspense boundary。

解决方案：在父页面调用当前组件外包裹一层Suspense组件



# vue-router 怎么实现history配置

1. 前端配置

首先要设置路由的mode和base两个值，如下：

const routes = [...]

const router = new VueRouter({

mode: "history",

base: process.env.BASE\_URL, // 如果使用history模式，必须设置base参数

routes

});

export default router;

1. 其次要设置vue.config.js里的publicPath，如下：

module.exports = {

// publicPath默认值是'/'，即你的应用是被部署在一个域名的根路径上

// 设置为'./'，可以避免打包后的静态页面空白

// 当在非本地环境时，这里以项目test-daily为例，即打包后的h5项目部署服务器的test-daily目录下

// 那么这里就要把publicPath设置为/test-daily/，表示所有的静态资源都在/test-daily/里

// 打包部署后，会发现index.html引用的静态资源都添加了路径/test-daily/

publicPath: process.env.NODE\_ENV == 'development' ? './' : '/test-daily/',

......

}

1. 服务端配置(这里以nginx为例）

# 浏览器的渲染

1. 浏览器中的5个进程

**浏览器进程**。主要负责界面显示、用户交互、子进程管理、同时提供存储等功能。

**渲染进程**。 核心任务是将HTML、CSS和JavaScript转换为用户可以与之交互的网页，排版引擎Blink和JavaScript引擎V8都运行在该进程中，默认情况下，Chrome为每一个Tab标签页创建一个渲染进程。出于安全考虑，渲染进程都是运行在沙箱模式下的。

**GPU进程**。\*\*GPU图形处理器（英语：graphics processing unit，缩写：GPU）,负责3D CSS效果，网页，Chrome ui的绘制。

**网络进程**。主要负责页面的网络资源加载，之前是作为一个模块运行在浏览器进程里面的，直至最近才独立处理，成为单独一个进程。

**插件进程**。主要负责插件的运行，因为插件易崩溃，所以通过插件进程来隔离，以保证插件进程崩溃不会对浏览器和页面造成影响。每一种类型的插件对应一个进程，仅当使用该插件时才创建。

所以我们开启一个页面，至少会启动4个进程。

# 回流和重绘以及优化的办法

1. 回流（reflow）

有时也叫做重排或者布局（layout）。通过构造渲染树，将可见DOM节点以及它对应的样式结合起来，可是还需要计算它们在页面内的确切位置和大小，这个计算的阶段就是回流。

或者说：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)。

**触发条件：**

1. 页面一开始渲染的时候（无法避免）
2. 添加或删除可见的DOM元素
3. 元素的位置发生变化
4. 元素的尺寸发生变化（包括外边距、内边框、边框大小、高度和宽度等）
5. 内容发生变化（文本变化或图片被另一个不同尺寸的图片所替代等）
6. 浏览器的窗口尺寸变化（因为回流是根据视口的大小来计算元素的位置和大小的）
7. 重绘（repaint）

通过构造渲染树和回流阶段，知道了哪些节点是可见的，以及可见节点的样式和具体的几何信息(位置、大小)，那么我们就可以将渲染树的每个节点都转换为屏幕上的实际像素，这个阶段就叫做重绘节点。

重绘是当节点需要更改外观而不会影响布局的，比如改变 color 就会引起重绘

****回流必定会发生重绘，重绘不一定会引发回流****

1. **优化**

**如果想设定元素的样式，通过改变元素的 class 类名 (尽可能在 DOM 树的最里层)**

**避免设置多项内联样式**

**应用元素的动画，使用 position 属性的 fixed 值或 absolute 值(如前文示例所提)**

**权衡平滑和速度**

**避免使用 table 布局**

**避免使用 CSS 的 JavaScript 表达式 (仅 IE 浏览器)**

# websocket原理

实现了浏览器与服务器全双工通信，能更好的节省服务器资源和带宽并达到实时通讯的目的，是一个持久化的协议

websocket约定了一个通信的规范，通过一个握手的机制，客户端和服务器之间能建立一个类似tcp的连接，从而方便它们之间的通信

在websocket出现之前，web交互一般是基于http协议的短连接或者长连接

# tcp三次握手四次挥手

1. 三次握手
   1. 第一次：发送端发送一个SYN=1，ACK=0数据包给接收端，请求进行连接 早上吃了吗
   2. 第二次：接收端收到请求并允许连接，发送一个SYN=1，ACK=1数据包给发送端，并告诉可以通信了，并让发送端发送一个确认数据包 吃了
   3. 第三次：发送端发送一个SYN=0，ACK=1数据包给接收端，告诉连接已经被确认 你呢
   4. 之后，tcp连接建立成功，开始通信 我也吃了
2. 四次挥手
   1. 第一次：主动关闭方告诉被动关闭方：我要停止发数据了，但主动关闭方还可以收数据 （A对B：再见）
   2. 被动关闭方收到消息，发送一个ACK给对方，确认序号为收到序号+1 （B：好的）
   3. 被动关闭方发送一个消息:表明我不会给你发数据了 （B对A：再见）
   4. 第四次：主动关闭方收到消息，发送一个确认信号 （A：好的）

# websocket和http区别

相同点：

都是基于tcp的，都是可靠性传输协议

都是应用层协议

不同点：

WebSocket是双向通信协议，模拟Socket协议，可以双向发送或接受信息

HTTP是单向的

WebSocket是需要浏览器和服务器握手进行建立连接的

而http是浏览器发起向服务器的连接，服务器预先并不知道这个连接

# 几种与服务端实时通信的方法

1. AJAX轮询

AJAX轮询也就是定时发送请求，也就是普通的客户端与服务端通信过程，只不过是无限循环发送，这样，可以保证服务端一旦有最新消息，就可以被客户端获取。

ajax轮询的原理非常简单，让浏览器隔个几秒就发送一次请求，询问服务器是否有新信息

1. Long Polling长轮询

长轮询是客户端和浏览器保持一个长连接，等服务端有消息返回，断开。

然后再重新连接，也是个循环的过程，无穷尽也...

其实原理跟 ajax轮询 差不多，都是采用轮询的方式，不过采取的是阻塞模型（一直打电话，没收到就不挂电话），也就是说，客户端发起连接后，如果没消息，就一直不返回Response给客户端。直到有消息才返回，返回完之后，客户端再次建立连接，周而复始。

1. websocket

需要经过一次HTTP请求，就可以做到源源不断的信息传送了。（在程序设计中，这种设计叫做回调，即：你有信息了再来通知我，而不是我每次跑来问你 ）

只需要一次HTTP握手，所以说整个通讯过程是建立在一次连接/状态中，也就避免了HTTP的非状态性，服务端会一直知道你的信息，直到你关闭请求，这样就解决了接线员要反复解析HTTP协议，还要查看identity info的信息。

同时由客户主动询问，转换为服务器（推送）有信息的时候就发送（当然客户端还是等主动发送信息过来的。。），没有信息的时候就交给接线员（Nginx），不需要占用本身速度就慢的客服（Handler）了

# Grid布局

Grid布局即网格布局

Grid布局和flex布局是有实质性的区别的，flex是一维布局，只能处理一个维度上的布局，一行或者是一列。但是Grid布局是二维布局 ，将容器划分成了“行”和“列”，产生了一个个的网格，可以将网格元素放在行和列相关的位置上，从而达到了布局的目的

# 流式布局

流式布局也叫百分比布局，也称非固定像素布局或自适应布局

这种布局的特点是随着屏幕的改变，页面的布局没有发生大的变化，可以进行适配调整，这个正好与响应式布局相补

优点：元素的宽高用百分比做单位，元素宽高按屏幕分辨率调整，布局不发生变化

缺点：屏幕尺度跨度过大的情况下，页面不能正常显示

左侧固定+右侧自适应

左右固定宽度+中间自适应

# Rem布局

rem布局的本质是等比缩放，一般是基于屏幕宽度

先按定高宽设计出来页面，然后转换为rem单位,配合js查询屏幕大小来改变html的font-size,达到整个页面的缩放

rem是弹性布局的一种实现方式，弹性布局可以算作响应式布局的一种，但响应式布局不是弹性布局，弹性布局强调等比缩放，100%还原；响应式布局强调不同屏幕要有不同的显示，比如媒体查询

# Vue中computed原理

computed是vue中的计算属性，在依赖的值发生变化的时候进行重新计算，否则使用缓存

通过Watcher中的dirty属性从而得知computed的值发生了变化，dirty属性为true时，说明computed中的值需要重新计算，dirty为false时，则说明依赖没有变化，不需要重新计算

当计算属性的值发生变化时，计算属性的watcher和组件的watcher都会得到通知。

计算属性的watcher会将dirty置为true,组件的Watcher得到通知，同样将dirty属性置为false,重新计算值，用于本次渲染

computed就是定义在vm上的一个getter属性，这个getter属性被触发时会做两件事

1. 计算当前属性的值，此时会使用Watcher去观察计算属性中用到的所有其他数据的变化。同时将计算属性的Watcher的dirty属性设置为false.

2.当计算属性中用到的数据发生变化时,将得到通知从而进行重新渲染

# http常见[状态码](https://so.csdn.net/so/search?q=%E7%8A%B6%E6%80%81%E7%A0%81&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/nbsl_/article/details/_blank)

##### 1xx ，1表示消息

100 （继续）请求者应当继续提出请求。 服务器返回此代码表示已收到请求的第一部分，正在等待其余部分。

101（切换协议） 请求者已要求服务器切换协议，服务器已确认并准备切换。

##### 2xx，2表示成功

200 成功处理了请求，一般情况下都是返回此状态码；

201 请求成功并且服务器创建了新的资源。  
202 接受请求但没创建资源；  
203 返回另一资源的请求；  
204 服务器成功处理了请求，但没有返回任何内容；  
205 服务器成功处理了请求，但没有返回任何内容；  
206 处理部分请求

##### 3xx，3表示重定向

300 （多种选择） 针对请求，服务器可执行多种操作。 服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择。  
301 （永久移动） 请求的网页已永久移动到新位置。 服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。  
302 （临时移动） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。  
303 （查看其他位置） 请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。  
304 （未修改） 自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。  
305 （使用代理） 请求者只能使用代理访问请求的网页。 如果服务器返回此响应，还表示请求者应使用代理。  
307 （临时重定向） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

##### 4xx, 4表示请求错误

400 服务器不理解请求的语法。  
401 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 服务器拒绝请求。  
404 服务器找不到请求的网页。  
405 禁用请求中指定的方法。  
406 无法使用请求的内容特性响应请求的网页。  
407 此状态代码与 401类似，但指定请求者应当授权使用代理。  
408 服务器等候请求时发生超时。  
409 服务器在完成请求时发生冲突。 服务器必须在响应中包含有关冲突的信息。  
410 如果请求的资源已永久删除，服务器就会返回此响应。  
411 服务器不接受不含有效内容长度标头字段的请求。  
412 服务器未满足请求者在请求中设置的其中一个前提条件。  
413 服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。  
414 请求的 URI（通常为网址）过长，服务器无法处理。  
415 请求的格式不受请求页面的支持。  
416 如果页面无法提供请求的范围，则服务器会返回此状态代码。  
417 服务器未满足”期望”请求标头字段的要求。

##### 5xx, 5表示服务器错误

500 （服务器内部错误） 服务器遇到错误，无法完成请求。  
501 （尚未实施） 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。  
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。  
503 （服务不可用） 服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。  
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。  
505 （HTTP 版本不受支持） 服务器不支持请求中所用的 HTTP 协议版本。

# 虚拟dom原理

当用原生js或jquery等库去操作DOM时，浏览器会从构建DOM树开始讲整个流程执行一遍，所以频繁操作DOM会引起不需要的计算，导致页面卡顿，影响用户体验，而Virtual DOM能很好的解决这个问题

# 深拷贝浅拷贝的区别

## 浅拷贝（浅复制）

浅拷贝是创建一个新对象，这个对象有着原始对象属性值的一份精确拷贝。

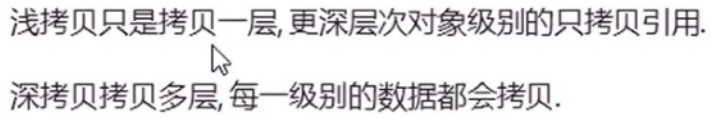
如果属性是基本类型，拷贝的就是基本类型的值，

如果属性是引用类型，拷贝的就是指向内存的地址 ，所以如果其中一个对象改变了这个引用类型的值，就会影响到另一个对象

## 深拷贝（深复制）

深拷贝需要不但对指针进行拷贝，并对指针指向的内容进行拷贝，经过深拷贝后的指针是指向两个不同地址的指针

深拷贝是将一个对象从内存中完整的拷贝一份出来,从堆内存中开辟一个新的区域存放新对象,且修改新对象不会影响原对象



## 实现深拷贝的方式

1. **递归调用**

const deepCopy = (obj) => {

// 判断传入的值是否为一个对象

if (obj === null && typeof obj !== "object") {

return obj;

}

// 判断对象的类型 注意这里不考虑包装类对象

if (Object.prototype.toString.call(obj) === "[object Date]") {

return new Date(obj);

}

if (Object.prototype.toString.call(obj) === "[object RegExp]") {

return new RegExp(obj);

}

if (Object.prototype.toString.call(obj) === "[object Undefined]") {

return new Error(obj);

}

// 判断对象是类

let newObj = Array.isArray(obj) ? [] : {}

for(let item in obj){

if(typeof obj[item] === 'object') {

newObj[item] = deepCopy(obj[item])

}else {

newObj[item] = obj[item]

}

}

return newObj

};

const foo = {

name: '张三',

info: {

age: 24

}

}

const newFoo = deepCopy(foo)

console.log(foo, newFoo)

foo.info.age = 25

console.log(foo, newFoo)

1. **JSON.stringify + JSON.parse**

当对象内存在undefined、symbol、function类型的属性时，在序列化过程中会被忽略。

当属性为NaN 和 Infinity 格式的数值及 null 都会被当做 nul

const foo = {

name: "张三",

age: undefined,

height: 183,

gender: Symbol("男"), //

say: () => {},

aa: NaN,

};

const newFoo = JSON.parse(JSON.stringify(foo));

console.log(foo, newFoo);

// {aa: NaN, age: undefined, gender: Symbol(男), height: null, name: "张三"} { {name: '张三', height: null, aa: null} }

JSON.stringify() 第二个参数为可选参数，可以是一个函数或者数组，

当是数组时表示需要被转化的属性列表，但undeinfed,symbol,function类型的属性依然会被忽略

const newFoo = JSON.parse(JSON.stringify(foo, ['name', 'age', 'gender', 'say', 'aa']));

console.log(newFoo)

// {name: '张三', aa: null}

1. **Object.assgin**

**只有当对象中没有嵌套对象时，才可以实现深拷贝**

// 对象中有内嵌的对象时

const foo = {

name: '张三',

info: {

age: 24

}

}

const newFoo = Object.assign({}, foo)

foo.info.age = 25

console.log(foo, newFoo)

## 浅拷贝实现方式：

1. Object.assign()

let obj2 = Object.assign({}, obj1);

1. 展开运算符...

let obj2= {... obj1}

1. Concat

let arr = [1, 2, 3];

let arr2 = arr.concat();

1. slice

arr.slice(begin, end);

let arr3 = arr.slice()

# Vue和React组件间的传值方式

1. 父子之间
   1. Vue

父->子：通过props进行传递数据给子组件

子->父：通过emit向父组件传值，ref

* 1. React

父->子：通过props将数据传递给子组件

子->父：通过父组件向子组件传递函数，然后子组件中调用这些函数，利用回调函数实现数据传递

1. 兄弟之间
   1. Vue

在Vue中，可以通过查找父组件下的子组件实例，然后进行组件进行通信。如this.parent.children，在$children中，可以通过组件的name找到要通信的组件，进而进行通信

* 1. React

在React中，需要现将数据传递给父组件，然后父组件再传递给兄弟组件

1. 多层级之间(孙子祖父等)
   1. Vue

可以通过不断的获取$parent找到需要传递的祖先级组件，然后进行通信，但是这样繁琐易错，并不推荐

provide/inject的方式进行传递数据。即在祖先组件提供数据，在需要使用的组件中，注入数据，这样就可以在子组件中使用数据了

* 1. React

和Vue类似的处理多层级组件通信的方式——context

只需要将子组件包裹在生产者的Provider中，在需要用到数据的子组件中，通过Consumer包裹，就可以拿到生产者的数据

1. 任意组件之间
   1. Vue--vuex
   2. React--redux

# Vue组件通信

1、props和$emit

父组件向子组件传递数据是通过prop传递的，子组件传递数据给父组件是通过$emit触发事件

2、$attrs和$listeners

3、中央事件总线 bus

上面两种方式处理的都是父子组件之间的数据传递，而如果两个组件不是父子关系呢？这种情况下可以使用中央事件总线的方式。新建一个Vue事件bus对象，然后通过bus.$emit触发事件，bus.$on监听触发的事件。

4、provide和inject

父组件中通过provider来提供变量，然后在子组件中通过inject来注入变量。不论子组件有多深，只要调用了inject那么就可以注入provider中的数据。而不是局限于只能从当前父组件的prop属性来获取数据，只要在父组件的生命周期内，子组件都可以调用。

5、v-model

父组件通过v-model传递值给子组件时，会自动传递一个value的prop属性，在子组件中通过this.$emit(‘input’,val)自动修改v-model绑定的值

6、$parent和$children

7、boradcast和dispatch

8、vuex处理组件之间的数据交互 如果业务逻辑复杂，很多组件之间需要同时处理一些公共的数据，这个时候才有上面这一些方法可能不利于项目的维护，vuex的做法就是将这一些公共的数据抽离出来，然后其他组件就可以对这个公共数据进行读写操作，这样达到了解耦的目的。

# state和props的区别

props：一般用于父组件向子组件通信，在组件之间通信使用。

state：一般用于组件内部的状态维护，更新组建内部的数据，状态，更新子组件的props等。

this.setState({ childData: ‘Child Data’ });

在父组件的render()方法里面，可以将父组件的state，作为子组件的props传递下去

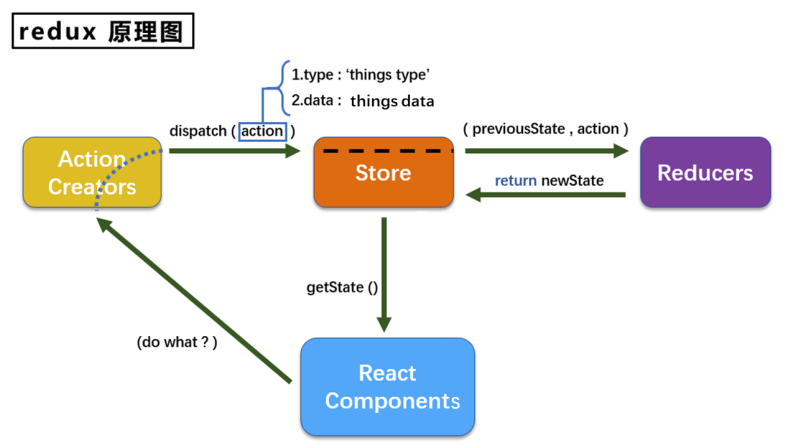
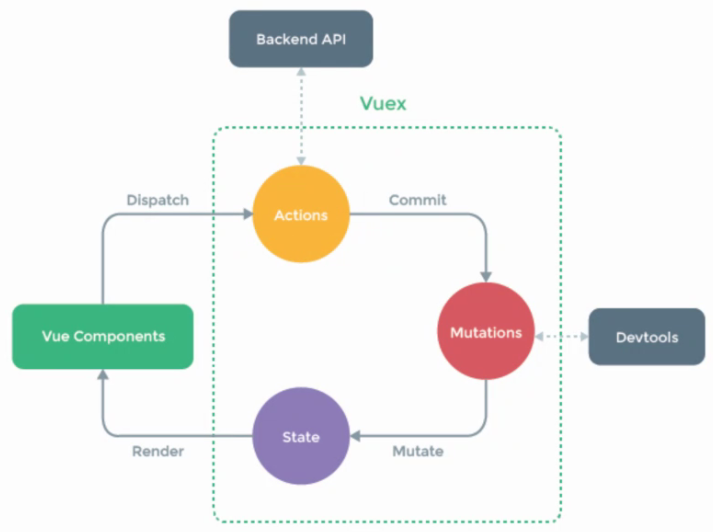
# vuex和redux的区别

Vuex改进了Redux中的Action和Reducer函数，以mutations变化函数取代Reducer，无需switch，只需在对应的mutation函数里改变state值即可

Vuex由于Vue自动重新渲染的特性，无需订阅重新渲染函数，只要生成新的State即可

Vuex数据流的顺序是∶View调用store.commit提交对应的请求到Store中对应的mutation函数->store改变（vue检测到数据变化自动渲染）

通俗点理解就是，vuex 弱化 dispatch，通过commit进行 store状态的一次更变；取消了action概念，不必传入特定的 action形式进行指定变更；弱化reducer，基于commit参数直接对数据进行转变，使得框架更加简易;



# 数据代理

1.Vue中的数据代理：

通过vm对象来代理data对象中属性的操作（读/写）

2.Vue中数据代理的好处：

更加方便的操作data中的数据

3.基本原理：

通过Object.defineProperty()把data对象中所有属性添加到vm上。

为每一个添加到vm上的属性，都指定一个getter/setter。

在getter/setter内部去操作（读/写）data中对应的属性。

# jsx语法规则

1. 定义虚拟DOM时，不要写引号

2. 标签中混入 JS表达式 时要用{ }

3. 样式的类名指定不要用class，要用className

4. 内联样式，要用style= {{ key:'value' }} 的形式去写

5. 虚拟DOM必须只有一个根标签

6. 标签必须闭合

7. 关于标签首字母

a. 若小写字母，则将标签改为html中同名元素，若html中无该标签对应的同名元素，则报错

b. 若大写字母开头，react就去渲染对应的组件，若组件没有定义，则报错

# vue双向绑定的原理

1. mvvm

MVVM（Model-View-ViewModel）是对 MVC（Model-View-Control）和 MVP（Model-View-Presenter）的进一步改进

MVVM 将数据双向绑定（data-binding）作为核心思想，View 和 Model 之间没有联系，它们通过 ViewModel

这个桥梁进行交互

Model 和 ViewModel 之间的交互是双向的，因此 View 的变化会自动同步到 Model，而 Model

的变化也会立即反映到 View 上显示

当用户操作 View，ViewModel 感知到变化，然后通知 Model 发生相应改变；反之当 Model 发生改变，ViewModel也能感知到变化，使 View 作出相应更新

MVVM框架的的核心就是双向绑定, 其原理是通过数据劫持+发布订阅模式相结合的方式来是实现的

1. 双向绑定的核心： Object.defineProperty()

obj：要定义属性的对象

prop：要定义或修改的属性的名称或 Symbol

descriptor：要定义或修改的属性描述符

返回值：被传递给函数的对象

通过Object.defineProperty的get方法用来获取值 set方法用来拦截设置值

1. object.defineProperty的缺点：
   1. 因为es5的object.defineProperty无法监听对象属性的删除和添加
   2. 不能监听数组的变化，除了push/pop/shift/unshift/splice/spObject.definert/reverse，其他都不行
   3. Object.defineProperty只能遍历对象属性直接修改(需要深拷贝进行修改)
2. proxy的优点：
   1. 直接监听对象而非属性
   2. 直接监听数组的变化
   3. 拦截的方式有很多种(有13种，set,get,has)
   4. Proxy返回一个新对象，可以操作新对象达到目的

# cookie、sessionStorage和localStorage的区别

1. webstorage本地存储

webstorage是本地存储，存储在客户端，包括localStorage和sessionStorage

localStorage和sessionStorage的存储数据大小一般都是：5MB

localStorage和sessionStorage都保存在客户端，不与服务器进行交互通信，所以安全性相对于cookie来说比较高一些

localStorage和sessionStorage只能存储字符串类型

sessionStorage为临时保存,仅在当前会话下有效，关闭页面或浏览器后被清除

localStorage生命周期是永久，这意味着除非用户显示在浏览器提供的UI上清除localStorage信息，否则这些信息将永远存在

1. Cookie

在HTTP请求发送Set-Cookie HTTP头作为响应的一部分。通过name=value的形式存储

cookie的作用:主要用于保存登录信息

cookie的作用:主要用于保存登录信息

cookie的长度和数量的限制。每个domain最多只能有20条cookie，每个cookie长度不能超过4KB

安全性问题。如果cookie被人拦掉了，那个人就可以获取到所有session信息。加密的话也不起什么作用

失效时间:一般默认是浏览器关闭失效,可以自己设置失效时间

# call、apply、bind三者的用法和区别

call、apply、bind都是改变this指向的方法

1. call

call可以调用函数,call可以改变函数中this的指向

参数一：Function的调用者将会指向这个对象

参数二：将参数一个一个传入，每个参数都会映射到Function的参数，若以数组传入，该数组将对应参数的第一位，之后参数都为undefined

1. apply

apply方法和call方法有些相似，它也可以改变this的指向，也可以有多个参数，但是不同的是，

第二个参数必须是一个数组

1. bind

bind方法返回的是一个修改过后的函数,

bind也可以有多个参数，并且参数可以执行的时候再次添加，但是要注意的是，参数是按照形参

的顺序进行的。

call和apply都是对函数进行直接调用，而bind方法返回的仍是一个函数

# 设计模式有哪些, 分别说一说 ? 共23种设计模式，介绍其中6种应用较为广泛的模式。

1. 发布订阅模式:

这种设计模式可以大大降低程序模块之间的耦合度，便于更加灵活的扩展和维护。

1. 中介者模式 :

观察者模式通过维护一堆列表来管理对象间的多对多关系，中介者模式通过统一接口来维护一对多

关系，且通信者之间不需要知道彼此之间的关系，只需要约定好API即可。

1. 代理模式 :

为其他对象提供一种代理以控制对这个对象的访问。

代理模式使得代理对象控制具体对象的引用。代理几乎可以是任何对象：文件，资源，内存中的对

象，或者是一些难以复制的东西。

1. 单例模式 :

保证一个类只有一个实例，并提供一个访问它的全局访问点（调用一个类，任何时候返回的都是同

一个实例）。

1. 工厂模式 :

工厂模式定义一个用于创建对象的接口，这个接口由子类决定实例化哪一个类。该模式使一

个类的实例化延迟到了子类。而子类可以重写接口方法以便创建的时候指定自己的对象类型

1. 装饰者模式 : 装饰者(decorator)模式能够在不改变对象自身的基础上，在程序运行期间给对像动态的

添加职责（方法或属性）。与继承相比，装饰者是一种更轻便灵活的做法。

# Promsie 和 async/await 的区别和使用

区别：

1）函数前面多了一个async关键字。await关键字只能用在async定义的函数内。async函数会隐式地

返回一个promise，该promise的reosolve值就是函数return的值。

2）第1点暗示我们不能在 外层代码中使用await，因为不在async函数内。使用：

1.async和await是配对使用的，await存在于async的内部。否则会报错 。

2.await表示在这里等待一个promise返回，再接下来执行。

3.await后面跟着的应该是一个promise对象，（也可以不是，如果不是接下来也没什么意义了…）

# 如何确定 this 指向？改变 this 指向的方式有哪些？

this 指向：

1. 全局上下文（函数外）

无论是否为严格模式，均指向全局对象。注意:严格模式下全局对象为undifined

1. 函数上下文（函数内）

默认的，指向函数的调用对象，且是最直接的调用对象：

简单调用，指向全局对象注意:严格模式下全局对象为undifined，某些浏览器未实现此标准也可能会是window

1. 改变this指向的方式
   1. new改变this指向
   2. Apply
   3. Bind
   4. Call

# this.$set怎么用

当你发现你给对象加了一个属性，在控制台能打印出来，但是却没有更新到视图上时，也许这个时候就需要用到this.set（ ）这个方法了，简单来说this.set（）这个方法了

怎么用？

在标签中写一个@click点击事件，在data中放一个list数组的数据，点击按钮调用方法，

this.$set( target, key, value )

target：要更改的数据源(可以是对象或者数组)

key：要更改的具体数据

value ：重新赋的值

用this.$set方法把更改的数据更新到视图上

# v-model实现原理

v-model 本质上是语法糖，可以用 v-model 指令在表单 <input>、<textarea> 及 <select> 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。它负责监听用户的输入事件以更新数据

**实现原理**

v-model只不过是一个语法糖而已,真正的实现靠的还是

1. v-bind:绑定响应式数据

2. 触发oninput 事件并传递数据

# 文本省略怎么实现？

1、首先给用于放文本的标签元素设置一个宽度值，并设置溢出属性overflow为溢出隐藏。

width: 245px;/\*一定要设置固定宽度\*/

overflow: hidden;/\*不显示超过对象尺寸的内容，就是把超出的部分隐藏了\*/

2、设置文本缩略的样式为"..."

white-space: nowrap;/\*表示文本不会换行，在同一行继续，直到遇到标签为止\*/

text-overflow: ellipsis;/\*当文本对象溢出是显示...，当然也可是设置属性为clip不显示...\*/

# flex布局-弹性盒子

flex-direction属性 决定主轴的方向，即项目的排列方向

row:默认值，主轴为水平方向，起点在左端

row-reverse:主轴为水平方向，起点在有段

column:主轴为垂直方向，起点在上方

column-reverse:主轴为垂直方向，起点在下方

flex-wrap属性 决定一条轴线排不下该如何换行-----有3个值

nowrap:默认值，不换行

wrap：换行，第一行在上方

wrap-reverse：换行，第一行在下方

flex-flow属性 是flex-direction属性和flex-wrap属性的简写形式

justify-content属性 定义了项目在主轴上的对齐方式

flex-start：默认值左对齐

flex-end：右对齐

center：居中

space-between：两端对齐，项目之间的间隔都相等

space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍

# javascript实现继承的方式：

1. 原型链继承；将父类的实例作为子类的原型。
2. 构造继承；使用父类的构造函数来增强子类实例。
3. 实例继承；为父类实例添加新特性，作为子类实例返回。
4. 拷贝继承。
5. 组合继承。
6. 寄生组合继承。

# fontSize最小字体12px，如何还能再小？

通过缩放来解决

color:rgb(98, 98, 98);

transform: scale(0.9); /\* 用缩放来解决 \*/

transform-origin: 0 0; /\* 左对齐 \*/

# js获取子节点的方式

# Vue优缺点

**优点：**

虚拟DOM（类似ReactJS）

首先，将传统DOM结构变为JSON结构，用操作js的方式代替操作DOM，性能更快；

其次，虚拟dom同步到真实dom，使用sync()方法，分批渲染。

双向绑定（类似AngularJS）

data和DOM任何一方修改，另一方都会同步修改。

**缺点：**

虚拟DOM需要在内存中的维护一份DOM的副本。在DOM更新速度和使用内存空间之间取得平衡。

简单页面不适用

如果虚拟DOM大量更改，这是合适的。但是单一的，频繁的更新的话，虚拟DOM将会花费更多的时间处理计算的工作。

所以，如果你有一个DOM节点相对较少页面，用虚拟DOM，它实际上有可能会更慢。

但对于大多数单页面应用，这应该都会更快。

二、单页面应用（vue-router）

优点：

把页面当组件加载，加载更快

共用API和资源，不再重复加载

缺点：

首次加载稍慢

所有JS和CSS会在首次加载完成（需要通过webpack合并压缩，减小资源大小和请求次数）

不利于 SEO

页面是异步加载，不利于搜索引擎抓取能（如信用分）暂时保持不变

# js获取节点的方式

获取子节点

通过获取dom方式直接获取子节点

通过childNodes获取子节点--是一个数组的格式

通过children来获取子节点

获取父节点

firstElementChild获取第一个子节点

parentNode获取父节点

parentElement获取父节点

offsetParent获取所有父节点

获取兄弟节点

通过获取父亲节点再获取子节点来获取兄弟节点

获取上一个兄弟节点

previousSibling会匹配字符，包括换行和空格，而不是节点

previousElementSibling则直接匹配节点

获取下一个兄弟节点

nextSibling和nextElementSibling同上

# 从地址栏输入URL到页面展示经历了什么？

输入URL后->先查看浏览器的缓存->没有则去系统缓存->路由器缓存->有则显示页面内容->无则发送http请求前域名解析->解析获取相应的IP地址->浏览器向服务器发送tcp链接，进行3次握手->握手成功后，浏览器向服务器发送http请求->请求数据包->服务器收到请求后，将数据返回到浏览器->浏览器读取渲染页面

# vue如何配置多环境变量？

process.env 会根据当前的环境,加载对应的环境变量文件中的变量

# pinia和vuex的区别？

（1）它没有mutation,他只有state，getters，action【同步、异步】使用他来修改state数据

（2）他默认也是存入内存中，如果需要使用本地存储，在配置上比vuex麻烦一点

（3）语法上比vuex更容易理解和使用，灵活。

（4）pinia没有modules配置，没一个独立的仓库都是definStore生成出来的

1. state是一个对象返回一个对象和组件的data是一样的语法

# webpack优化

webpack 性能优化可分为开发环境优化和生产环境优化

开发环境优化分为：优化代码构建速度，和优化代码调试

生产环境优化：分为优化打包构建速度和优化代码运行性能

<https://zhuanlan.zhihu.com/p/490076369>

# koa流程--中间件

koa的中间件机制是一个剥洋葱式的模型，多个中间件通过use放进一个数组队列然后从外层开始执行，遇到next后进入队列中的下一个中间件，所有中间件执行完后开始回帧，执行队列中之前中间件中未执行的代码部分，这就是剥洋葱模型。koa的中间件机制,是基于async/await + Promise实现的

# css拖拽

**实现拖拽的三大事件**

onmousedown （鼠标按下的时候）、onmousemove（鼠标移动的时候）、onmouseup（鼠标松开的时候）

给目标元素加上onmousedown事件，记录鼠标按下的时候，鼠标距离所在元素的位置。要移动的是这个元素，就要算出鼠标坐标与这个元素边界的距离

# w3c标准

**web标准**主要分为结构、表现、行为

**结构：**指我们平时在body里面写的标签，主要是由HTML标签组成

**表现：**指更加丰富HTML标签样式，主要由CSS样式组成

**行为：**指页面和用户的交互，主要由JS部分组成

W3C对web标准提出了代码规范

对结构的规范：

标签字母要小写

标签要闭合

标签不允许随意嵌套

对表现和行为的要求：

建议使用外链CSS和js脚本，实现结构与表现分离、结构与行为分离，能提高页面的渲染效率，更快地显示网页内容

# vue监视属性

watch对象中的属性，该属性名表示data或者computed对象中的同名属性被监视了

**作用：**用于监视data或者computed对象中的属性

**用法：**当需要在数据变化时执行异步或开销较大的操作时，用它

**监视属性的配置属性：**

handler(newValue,oldValue)：当监视属性监视的属性的值的发生改变时调用

newValue：被监视的属性修改之后的值

oldValue： 被监视的属性被修改之前的值

immediate：控制初始化时否是调用hander，默认值为false

deep：控制是否开启深度监视，默认值为false

# vue拖拽指令之offsetX、clientX、pageX、screenX

offset：偏移；client：客户；page：页面；screen：屏幕；

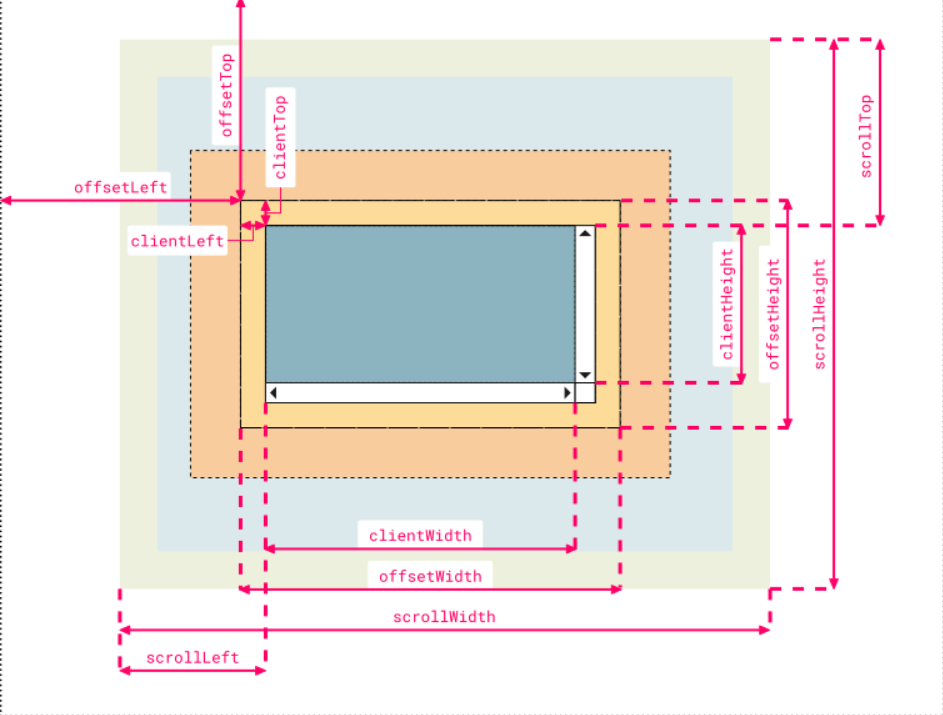
offsetX：相对于触发元素的偏移距离；

clientX：相对于客户端浏览器的距离，可以理解成相对于地址栏以下的区域；

pageX：相对于整个页面的距离，比clientX要加上滚动条移动的距离；

screenX：相对于整个屏幕的距离，包含地址栏；

# Clientwidth



# 四种post中的参数请求方式

1、application/x-www-form-urlencoded

First name:

Last name:

此时看到提交的数据按照 key1=val1&key2=val2 的方式进行编码，key 和 val 都进行了 URL 转码。大部分服务端语言都对这种方式很好的支持，常用的如jQuery中的ajax请求，Content-Type 默认值都是「application/x-www-form-urlencoded;charset=utf-8

2、multipart/form-data

这也是常见的post请求方式，一般用来上传文件，各大服务器的支持也比较好。所以我们使用表单 上传文件 时，必须让表单的enctype属性值为 multipart/form-data.

注意：以上两种方式：application/x-www-form-urlencoded和multipart/form-data都是浏览器原生支持的

3、application/json

application/json，除了低版本的IE，基本都支持。除了低版本的IE都支持JSON.stringify（）的方法，服务端也有处理JSON的函数，使用json不会有任何麻烦。

4、text/xml

它是一种使用 HTTP 作为传输协议，XML 作为编码方式的远程调用规范

**怎么回退到某一个历史节点**

git log 查询历次提交记录，得到版本号

得到的版本号是一个很长的编码

git reset --hard ef582da9e1dbcf34f939fdb0fa2db897ae4887b1 //git命令行回退到某个指定的版本

git push -f -u origin master // 如果需要push到远端,则需要强制提交

# BFC

**概念**

块级格式化上下文，简单来说，BFC定义了一个规则，让空间内的子元素不会影响到外部的布局。

**触发**

overflow: hidden

display: inline-block

position: absolute

position: fixed

display: table-cell

display: flex

采用这些就可以完成bfc的触发

**规则**

BFC就是一个块级元素，块级元素会在垂直方向一个接一个的排列

BFC就是页面中的一个隔离的独立容器，容器里的标签不会影响到外部标签

垂直方向的距离由margin决定， 属于同一个BFC的两个相邻的标签外边距会发生重叠

计算BFC的高度时，浮动元素也参与计算

**BFC解决的问题**

1. 解决float脱离文档流后，父元素高度塌陷问题

由于两个子元素都浮动，无法撑开父盒子，此时将父盒子设置为BFC规则，就可以了

1. Margin边距重叠

两个盒子之间存在margin，但是实际显示并不是两个margin的和，造成边距重叠，此时可以在最后一个子盒子包裹一层块级元素就可以

1. 两栏布局

左侧盒子出现左浮动时，会导致右侧盒子图里文档流覆盖上去了，将第二个div设置成一个BFC就行

# vue中封装自定义组件

我用vue开发的所有项目，都是采用组件化的思想开发的。一般我在搭建项目的时候，会创建一个views目录和一个commen目录和一个feature目录，views目录中放页面级的组件，commen中放公共组件（如：head（公共头组件），foot（公共底部组件）等），feature目录内放功能组件（如：swiper(轮播功能组件)，tabbar(切换功能组件)、list(上拉加载更多功能组件)）

首先，组件可以提升整个项目的开发效率。能够把页面抽象成多个相对独立的模块，解决了我们传统项目开发：效率低、难维护、复用性低等问题。

**使用组件的好处**

组件是可以复用性的

易于维护

有封装性，易于使用

大型项目中降低组件之间重复性

### **全局组件**

引用全局组件my-header<div id="main2"> *<!-- 自定义组件具有复用性 -->* <my-header></my-header> </div>

注册全局组件my-header

//全局组件：在任何地方，任何方式，任何地点都可以使用的标签组件

vue.component("my-header", {

// h1标签是我们在自定义主键里面写的参数的标签

template: '<h1>标题{{msg}}</h1>',

data(){

return {msg:100}

}

})

new Vue({

el: '#main'

});

局部组件：只是在我们规定的范围之内才会生效

**注：**

当使用 kebab-case(中划线命名) 定义一个组件时，你也必须在引用这个自定义元素时使用 kebab-case，例如<my-component-name>

当使用 PascalCase (驼峰式命名) 定义一个组件时，你在引用这个自定义元素时两种命名法都可以使用。也就是说

<my-component-name>和<MyComponentName> 都是可接受的。注意，尽管如此，直接在 DOM(即非字符串的模板) 中使用时只有kebab-case 是有效的

**脚手架vue-cli中的组件**

我们一般通过脚手架vue-cli开发项目，每个 .vue单文件就是一个组件。在父组件中使用import 导入一个子组件，并在components中注册子组件，子组件需要数据，可以在props中接受定义。而子组件修改好数据后，想把数据传递给父组件。可以采用$emit方法触发自定义事件传递参数。

所以在prop 的定义应该尽量详细，至少需要指定其类型

通过 $on 传递父组件方法，通过$on传递父组件方法是组件通信中常用的方法传递方式。它可以与通过props传递方法达到相同的效果。相比于props传递function,它更加的直观和显示的表现出了调用关系

通过$emit传递父组件数据

与父组件到子组件通讯中的$on配套使用，可以向父组件中触发的方法传递参数供父组件使用

# 判断数组或对象中是否包含指定元素

1. in

console.log('2 in arr', 2 in arr)

针对的是key值，而非value值.

对于数组，key值是隐藏的，表示索引值(index)

对于对象，key值显而易见。

返回值：存在true，不存在false

1. indexOf

console.log('arr.indexOf("a")',arr.indexOf("a"))

arr.indexOf(find,start)

仅用于字符串、数组。不能用于对象。

find:要找的内容，必须；

start:查找开始下标，可选；

返回：查找数据所在的下标，如果没找到，返回-1

1. includes

console.log('arr.includes("a")',arr.includes("a"))

仅用于字符串、数组。不能用于对象。

返回值：存在true，不存在false

# 介绍一下标准的CSS的盒子模型？与低版本IE的盒子模型有什么不同的？？

标准盒子模型：宽度=内容的宽度（content）+ border + padding + margin

低版本IE盒子模型：宽度=内容宽度（content+border+padding）+ margin

# box-sizing属性？

用来控制元素的盒子模型的解析模式，默认为content-box

context-box：W3C的标准盒子模型，设置元素的 height/width 属性指的是content部分的高/宽

border-box：IE传统盒子模型。设置元素的height/width属性指的是border + padding + content部分的高/宽

# CSS选择器有哪些？优先级顺序？哪些属性可以继承？

CSS选择器：

id选择器(#myid)、

类选择器(.myclassname)、

标签选择器(div, h1, p)、

相邻选择器(h1 + p)、

子选择器（ul > li）、

后代选择器（li a）、

通配符选择器（\*）、

属性选择器（a[rel=“external”]）、

伪类选择器（a:hover, li:nth-child）

CSS选择器优先级顺序：

-!important > 行内样式（比重1000）> ID 选择器（比重100） > 类选择器（比重10） > 标签（比重1） > 通配符 > 继承 > 浏览器默认属性

CSS可以继承属性：

font-size, font-family, color

CSS不可以继承属性：

border, padding, margin, width, height

什么是 !important：

CSS 中的 !important 规则用于增加样式的权重；

!important 与优先级无关，但它与最终的结果直接相关，使用一个 !important 规则时，此声明将覆盖任何其他声明。

# 垂直居中几种方式有哪些？

单行文本: line-height = height

图片: vertical-align: middle;

absolute 定位: top: 50%;left: 50%;transform: translate(-50%, -50%);

flex: display:flex; margin:auto(自由水平居中）

# display:none和visibility:hidden的区别？

display:none ：隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。

visibility:hidden ：隐藏对应的元素，但是在文档布局中仍保留原来的空间。

# position的值， relative和absolute分别是相对于谁进行定位的？

relative:相对定位，相对于自己本身在正常文档流中的位置进行定位。

absolute:生成绝对定位，相对于最近一级定位不为static的父元素进行定位

fixed: 固定定位，相对于浏览器窗口或者frame进行定位。（老版本IE不支持）

static:默认值，没有定位，元素出现在正常的文档流中。

sticky:生成粘性定位的元素，容器的位置根据正常文档流计算得出。

# 常见兼容性问题？

浏览器默认的margin和padding不同。解决方案是加一个全局的\*{margin:0;padding:0;}来统一。

Chrome 中文界面下默认会将小于 12px 的文本强制按照 12px 显示,

可通过加入 CSS 属性 -webkit-text-size-adjust: none; 解决.

# CSS3新增伪类有那些?

p:first-of-type 选择属于其父元素的首个元素

p:last-of-type 选择属于其父元素的最后元素

p:only-of-type 选择属于其父元素唯一的元素

p:only-child 选择属于其父元素的唯一子元素

p:nth-child(2) 选择属于其父元素的第二个子元素

:enabled :disabled 表单控件的禁用状态。

:checked 单选框或复选框被选中

# display有哪些值？说明他们的作用?

inline（默认）–内联

none – 隐藏

block – 块显示

table – 表格显示

list-item – 项目列表

inline-block – 与 display: inline 相比，主要区别在于 display: inline-block 允许在元素上设置宽度和高度。

# CSS3有哪些新特性？

文字阴影：text-shadow： 5px 5px 5px #FF0000;（水平阴影，垂直阴影，模糊距离，阴影颜色）

font-face属性：定义自己的字体

圆角（边框半径）：border-radius 属性用于创建圆角

边框图片：border-image: url(border.png) 30 30 round

盒阴影：box-shadow: 10px 10px 5px #888888

# position跟display、overflow、float这些特性相互叠加后会怎么样？

display属性规定元素应该生成的框的类型；

position属性规定元素的定位类型；

float属性是一种布局方式，定义元素在哪个方向浮动。

类似于优先级机制：position：absolute/fixed优先级最高，有他们在时，float不起作用，display值需要调整。float 或者absolute定位的元素，只能是块元素或表格。

# BFC 是什么？

BFC 即 Block Formatting Contexts (块级格式化上下文)，它属于普通流，即：元素按照其在 HTML 中的先后位置至上而下布局，在这个过程中，行内元素水平排列，直到当行被占满然后换行，块级元素则会被渲染为完整的一个新行，除非另外指定，否则所有元素默认都是普通流定位，也可以说，普通流中元素的位置由该元素在 HTML 文档中的位置决定。

可以把 BFC 理解为一个封闭的大箱子，箱子内部的元素无论如何翻江倒海，都不会影响到外部。

只要元素满足下面任一条件即可触发 BFC 特性

body 根元素

浮动元素：float 除 none 以外的值

绝对定位元素：position (absolute、fixed)

display 为 inline-block、table-cells、flex

overflow 除了 visible 以外的值 (hidden、auto、scroll)

# DIV布局的三种方式？

常规流式布局：

元素按照自身HTML元素定义的位置显示（怎么写的怎么显示）

元素按照自身的常规显示特性显示

浮动布局：

左浮动 float:left;

右浮动 float:right;

定位布局：

静态定位：position:static;（默认值），不写position相当于写上position:static;以前没学定位的时候其实都是静态定位，元素在它原本的位置显示，即使加了top、left等也不起作用。

相对定位：position: relative;相对定位是相对于自身的原始位置进行平移，如果设置position：relative；表示相对定位。z-index:值越大越在上面

注意：z-index必须加在已经定位的元素上才起作用。

绝对定位： position: absolute;子容器相对于父容器的定位，如果没有父容器，则相对于body定位。

# 设置元素浮动后，该元素的display值是多少？

自动变成display:block

13、全屏滚动的原理是什么？用到了CSS的哪些属性？

原理：有点类似于轮播，整体的元素一直排列下去，假设有5个需要展示的全屏页面，那么高度是500%，只是展示100%，剩下的可以通过transform进行y轴定位，也可以通过margin-top实现

overflow：hidden；

transition：all 1000ms ease；

# ::before 和 :after中双冒号和单冒号有什么区别？解释一下这2个伪元素的作用？

单冒号(:)用于CSS3伪类，双冒号(::)用于CSS3伪元素。

::before就是以一个子元素的存在，定义在元素主体内容之前的一个伪元素。并不存在于dom之中，只存在在页面之中。

:before 和 :after 这两个伪元素，是在CSS2.1里新出现的。起初，伪元素的前缀使用的是单冒号语法，但随着Web的进化，在CSS3的规范里，伪元素的语法被修改成使用双冒号，成为::before ::after

# 你对line-height是如何理解的？

行高是指一行文字的高度，具体说是两行文字间基线的距离。

CSS中起高度作用的是height和line-height，没有定义height属性，最终其表现作用一定是line-height。

单行文本垂直居中：把line-height值设置为height一样大小的值可以实现单行文字的垂直居中，其实也可以把height删除。

多行文本垂直居中：需要设置display属性为inline-block。

# 小程序 xcx

# 微信的小程序的主要文件

* WXML——模板文件
* JSON——配置/设置文件，如标题，tabbar，页面注册
* WXSS——样式文件，样式可直接用import 导入
* JS——脚本逻辑文件，逻辑处理，网络请
* 求
* app.json——配置文件入口，整个小程序的全局配置，网络超时时间、底部tab、页面路径，window字段是小程序所有页面的顶部背景颜色、文字颜色
* app.js——可以没有内容，可以在里边监听

生命周期函数、声明全局变量· app.wxss——全局配置样式文件

# uni-app微信小程序封装全局判断是否登录方法结合全局变量

新建的uni-app项目会有个app.vue文件，在此文件下 onLaunch应用生命周期封装一个方法

onLaunch: function() {

console.log('App Launch')

//是否判断登录

wx.checkSession({

success: () => {

//没用到token可以直接忽略以下步骤:

console.log('登录未过期');

const user\_token = uni.getStorageSync('user\_token');

if(user\_token){

this.globalData.userLogin = true;

}else{

//解决token消失但登录未过期问题

this.globalData.userLogin = false;

//没登录就异步删除验证token

uni.removeStorage({

key: 'user\_token',

success: function (res) {

console.log('user\_token移除成功');

}

});

}

//没用到token走这个步骤:

//this.globalData.userLogin = false;

},

fail: () => {

console.log('登录已过期');

this.globalData.userLogin = false;

//没用到token可以直接忽略以下步骤:

//没登录就异步删除验证token

uni.removeStorage({

key: 'user\_token',

success: function (res) {

console.log('user\_token移除成功');

}

});

}

})

},

# 简述小程序原理

小程序分为两个部分webview和

appService，webview用来展现UI，appService用来处理业务逻辑、数据及接口调用，它们在两个进程中运行，通过系统层JSBridge实现通信，完成UI渲染、事件处理。

# 小程序的优点和缺点

小程序的优点

·无需下载●打开速度快● 开发成本低

为用户提供良好的安全保障。发布有一套严格的审查流程，不能通过审查的程序无法发布上线●服务请求快

小程序的缺点

依托微信，不能开发后台管理功能●大小限制不能超过2M，不能打开超过5个层级的页面

# 小程序怎么跟随事件传值

给HTML元素添加data-\*属性来传递我们需要的值，然后通过e.currentTarget.dataset或 onload的 param参数获取。但data -名称不能有大写字母和不可以存放对象

<button bindtap="get" data-name="测试"> 拿到传值</button>

get(e){

console.log(e.currentTarget.dataset.name)

},

# 小程序页面之间有哪些（传值）传递数据的方法

使用全局遍历实现数据传递,在 app.js 文件中定义全局变量globalData，将需要存储的信息存放在里面

页面跳转或重定向wx.navigateTo与wx.redirectTo时，可以将部分数据放在url里面，并在新页面onLoad的时候初始化

使用组件模板 template传递参数

使用缓存传递参数

使用数据库传递参数

给html元素添加data-属性来传递值，然后通过e.currentTarget.dataset或onload的param参数获取（data- 名称不能有大写字母，不可以存放对象）

设置id 的方法标识来传值，通过e.currentTarget.id获取设置的id值，然后通过设置全局对象的方式来传递数据

在navigator中添加参数数值

# 小程序简单介绍下三种事件对象的属性列表?

**基础事件(BaseEvent)**

type:事件类型

timeStamp:事件生成时的时间戳target:触发事件的组件的属性值集合currentTarget:当前组件的一些属性集合

**自定义事件(CustomEvent)**

detail

**触摸事件(TouchEvent)**

touches

changedTouches

# 小程序运行机制

热启动 ：假如用户已经打开了某个小程序，在一定时间内再次打开小程序的话，这个时候我们就不再需要重新启动了，这需要把我们的后台打开的小程序切换到前台来使用。

冷启动：用户首次打开小程序或被微信主动销毁再次打开的情况，此时小程序需要重新加载启动。

# 小程序什么时候会主动销毁？

小程序在进入后台之后，客户端会帮我们在一定时间内维持我们的一个状态，超过五分钟后，会被微信主动销毁.

官方也没有明确说明 什么时候销毁， 在不同机型表现也不一样，

2019年开发时：时间官方文档没有说明，但是经过询问一般指5分钟内

2020年开发时：时间官方文档没有说明，实测安卓没有固定时间，内存足够情况下，有时候一天了还在，有时候几分钟就没了。

# 怎么解决小程序的异步请求问题

小程序支持大部分ES6语法

在返回成功的回调里面处理逻辑Promise异步

# 小程序数据请求怎么封装

1. 将所有的接口放在统一的js文件中并导出（或者将请求地址、头、方法在一个js文件里统一定义为一个常量并导出）
2. 在app.js创建封装请求数据的方法在
3. 子页面中调用封装的方法请求数据

# 参数传递

给HTML元素中添加data-\*属性来传递需要的值，之后通过e.currentTarget.dataset或onload的param参数获取。注意不能有大写字母，不可以存放对象

跳转页面时通过navigator传递需要的参数值

设置id的方法标识，通过e.currentTarget.id

获取设置的id值，然后通过设置全局变量的方法来传递数值

# 小程序有哪些传递数据的方法

1. 使用全局变量

在 app.js 中的 this.globalData = { } 中放入要存储的数据。

在 组件.js 中， 头部 引入 const app = getApp(); 获取到全局变量

直接使用 app.globalData.key 来进行赋值和获取值。

2. 使用 路由

wx.navigateTo 和 wx.redirectTo 时，可以通过在 url 后 拼接 + 变量， 然后在 目标页面 通过在 onLoad 周期中，通过参数来获取传递过来的值。

3. 使用本地缓存

# 生命周期函数

onLoad : 页面加载时触发。一个页面只会调用一次，可以在 onLoad的参数中获取打开当前页面路径中的参数

onShow : 页面显示 / 切入前台时触发调用。

onReady : 页面初次渲染完成时触发,一个页面只会调用一次。

onHide : 页面隐藏 / 切入后台时触发，如 navigateTo 或底部 tab切换到其他页面，小程序切入后台等

onUnload : 页面卸载时触发。如 redirectTo或 navigateBack 到其他页面时.

# 小程序怎么实现下拉刷新

两种方案

**方案 一 ：**

通过在 app.json 中， 将 "enablePullDownRefresh": true, 开启全局下拉刷新。

或者通过在 组件 .json ， 将 "enablePullDownRefresh": true, 单组件下拉刷新。

在下拉函数执行时发起数据请求，请求返回后，调用wx.stopPullDownRefresh停止下拉刷新的状态

**方案二：**

scroll-view ：使用该滚动组件 自定义刷新，通过 bindscrolltoupper 属性， 当滚动到顶部/左边，会触发 scrolltoupper事件，所以我们可以利用这个属性，来实现下拉刷新功能。

# 1px=2rpx

# 如何自定义组件（弹窗）

先创建一个components文件夹，用来存放所有自定义组件的，目录结构依然是js,wxml,json,wxss 基本配置

json———进行自定义组件声明

{"component": true 3 使用组件}

假如在index.wxml中使用这个自定义的组件，首先在index.json中进行声明

{"usingComponents":{

"toastdemo":"/components/toas}}

接着在index.wxml中引用● 然后在index.js进行配置●使用时直接执行

this.toastdemo.showToast（'弹框组件调用成功，2000）就可以了

# bindtap 和 catchtap 区别

相同点： 都是点击事件

不同点： bindtap 不会阻止冒泡， catchtap 可以阻止冒泡

# 小程序内的页面跳转

wx.navigateTo() : 保留当前页面，跳转到应用内的某个页面。但是不能跳到 tabbar 页面

wx.redirectTo() : 关闭当前页面，跳转到应用内的某个页面。但是不允许跳转到 tabbar 页面

wx.switchTab() : 跳转到 TabBar 页面，并关闭其他所有非 tabBar 页面

wx.navigateBack() : 关闭当前页面，返回上一页面或多级页面。可通过 getCurrentPages() 获取当前的页面栈，决定需要返回几层

wx.reLaunch() : 关闭所有页面，打开到应用的某个页面。

# 小程序 wx:if 和 hidden 的区别

wx:if : 有更高的切换消耗。

hidden : 有更高的初始渲染消耗。

**使用**

频繁切换使用 hidden, 运行时条件变化使用 wx: if

# app.json 全局配置文件描述

pages : 用于存放当前小程序的所有页面路径

window : 小程序所有页面的顶部背景颜色，文字颜色配置。

tabBar : 小程序底部的 Tab ,最多5个，最少2个

# 小程序和Vue的区别

**小程序的双向绑定和vue的区别：**

小程序 直接使用this.data.key = value 是 不能更新到视图当中的。

必须使用 this.setData({ key ：value }) 来更新值。

**循环遍历∶** 小程序是wx∶for="list"，vue是v-for="inforin list"

**调用data模型（赋值）的时候：**

小程序：this.data.item // 调用，this.setDate({item:1})//赋值

Vue：this.item //调用，this.item=1 //赋值

生命周期不一样，微信小程序生命周期比较简单

数据绑定也不同，微信小程序数据绑定需要使用()， vue直接:就可以显示与隐藏元素， vue中，使用v-if和v-show控制元素的显示和隐藏,小程序中，使用wx-if和hidden控制元素的显示和隐藏

事件处理不同，小程序中，全用 bindtap(bind+event)，或者catchtap(catch+event)绑定事件,vue:使用v-onevent绑定事件，或者使用@event绑定事件

数据双向绑定也不也不一样在vue中.只需要再表单元素上加上 v-model,然后再绑定 data中对应的一个值，当表单元素内容发生变化时, data中对应的值也会相应改变，这是vue非常 nice的一点。微信小程序必须获取到表单元素，改变的值，然后再把值赋给一个data中声明的变量。

# 提高小程序的应用速度的方法

提高页面加载速度

用户行为预测

减少默认data的大小

组件化方案

# setData的回调函数

微信小程序的setData实现是和react的setData 实现类似的，所以它也是一个异步函数，并且有回调函数的参数，当然平时小量数据我们可能并没有感觉到它的异步，但是为了确保逻辑的正确执行，在需要用到setData后data里的数据的步骤，请写入setData的回调函数中，如下示例

this.setData(d

a: this.data.a++},()=>{})

# 微信小程序与H5的区别

①运行环境不同（小程序在微信运行，h5在浏览器运行）

传统的HTML5的运行环境是浏览器，包括webview，而微信小程序的运行环境并非完整的浏览器，是微信开发团队基于浏览器内核完全重构的一个内置解析器，针对小程序专门做了优化，配合自己定义的开发语言标准，提升了小程序的性能。

②开发成本不同（h5需要兼容不同的浏览器）

只在微信中运行，所以不用再去顾虑浏览器兼容性，不用担心生产环境中出现不可预料的奇妙BUG

③获取系统权限不同（系统级权限可以和小程序无缝衔接）

④应用在生产环境的运行流畅度（h5需不断对项目优化来提高用户体验）

# 小程序onPageScroll方法的使用注意什么?

由于此方法调用频繁，不需要时，可以去掉，不要保留空方法，并且使用onPageScroll时，尽量避免使用setDatalq尽星减少setData4的使用频次。

# 小程序关联微信公众号如何确定用户的唯一性

使用wx.getUserlnfo方法withCredentials为true时，可获取encryptedData，里面有union\_id.后端需要进行对称解密。

# 小程序如何确定用户的唯一性

小程序可以通过微信官方提供的登录能力方便地获取微信提供的用户身份标识，快速建立小程序内的用户体系

然而因为小程序中的openid不可以直接使用需要用code（登录凭证）去换取openid

**思路**

获取openid首先需要调用小程序的login方法获取小程序的登录凭证code,然后使用code向微信换取登录态信息,包括用户的唯一标识(openid)及本次登录的会话密钥(session\_key)

我这里是用一个点击事件来触发一个函数

wx.getUserInfo 会获取到你的一些信息 比如名字

info(){

wx.getUserInfo({

//成功后会返回

success:(res)=>{

console.log(res);

}

})

},

wx.login 可以获取到你的code值

wx.login({

//成功放回

success:(res)=>{

console.log(res);

let code=res.code

}

})

通过wx.request 来让code换取openid

在微信开发平台获取APPID,SECRET

这里面需要一段很长的url地址

https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js\_code=JSCODE&grant\_type=authorization\_code

**完整代码**



# webview中的页面怎么跳回小程序中

先在管理后台配置域名白名单，● 然后引入jweixin-1.3.2.js

(https://res.wx.qq.com/open/js/jweixin-1.3.0.js)

wx.miniProgram.navigateTo(furl: '/pa wx.miniProgram.navigateTo(furl: '/pa

# webview的页面怎么跳转到小程序导航的页面

小程序导航的页面可以通过switchTab，但默认情况是不会重新加载数据的。若需加载新数据，则在success属性中加入以下代码即可∶

success: function (e) {

var page = getCurrentPages().pop()if (page == undefined ll page == n page.onLoad();1

//webview的页面，则通过wx.miniProgram.switchTab(d

url: '/pages/index/index'})

# 小程序和小程序之间的跳转

在同一主体公众号上关联2个小程序appid,

·用navigator，对应设置一些属性即可·target∶miniProgram——其他小程序·target∶self——当前小程序

# 简述微信小程序原理？

微信小程序采用JavaScript、WXML、WXSS 三种技术进行开发.本质就是一个单页面应用，所有的页面渲染和事件处理，都在一个页面内进行,但又可以通过微信客户端调用原生的各种接口

微信的架构，是数据驱动的架构模式，它的UI和数据是分离的，所有的页面更新，都需要通过对数据的更改来实现

小程序分为两个部分webview和 appService。其中 webview主要用来展现Ul, appService有来处理业务逻辑、数据及接口调用。它们在两个进程中运行，通过系统层JSBridge 实现通信，实现UI的渲染、事件的处理

# 请谈谈微信小程序主要目录和文件的作用？

project.config.json：项目配置文件，用的最多的就是配置是否开启https校验

App.js：设置一些全局的基础数据等

App.json：底部tab，标题栏和路由等设置

App.wxss：公共样式，引入iconfont等

pages：里面包含一个个具体的页面

index.json：配置当前页面标题和引入组件

index.wxml：页面结构

index.wxss：页面样式表

index.js：页面的逻辑，请求和数据处理

# 请谈谈wxml与标准的html的异同？

都是用来描述页面的结构

都由标签，属性等构成

标签名字不一样，且小程序标签更少，单一标签更多

多了一些 wx:if 这样的属性以及{{}} 这样的表达式

WXML仅能在微信小程序开发者工具中预览，而HTML可以在浏览器内预览

组件封装不同，WXML对组件进行了重新封装

小程序运行在JS Core内，没有DOM树和windiw对象，小程序中无法使用window对象和document对象

# WXSS和CSS的区别？

wxss的图片引入需使用外链地址

没有Body;样式可直接使用import导入

都是用来描述页面的样子

WXSS具有CSS大部分的特性，也做了一些扩充和修改

WXSS新增了尺寸单位，WXSS在底层支持新的尺寸单位rpx

WXSS仅支持部分CSS选择器

WXSS提供全局样式与局部样式

# 分析微信小程序的优劣势？

**优势：**

容易上手，基础组件库比较全，基本不需要考虑兼容问题

开发文档比较完善，开发社区比较活跃，支持插件式开发

良好的用户体验，无需下载，通过搜索和扫一扫就可以打开，打开速度快，安卓上可以添加到桌面，与原生APP差不多

开发成本比APP要低

为用户提供良好的保障（小程序发布，严格是审查流程）

**劣势：**

限制较多，页面大小不能超过1M，不能打开超过5个层级的页面

样式单一，部分组件已经是成型的，样式不可修改，例如：幻灯片，导航

推广面窄，不能分享朋友圈，只能通过分享给朋友，附加小程序推广

依托与微信，无法开发后台管理功能

后台调试麻烦，因为api接口必须https请求且公网地址

真机测试，个别安卓和苹果表现迥异，例如安卓的定位功能加载很慢

# 小程序与原生App那个好？

各有各自的优点，都又有缺点

**小程序的优点：**

基于微信平台开发，享受微信自带的流量，这个优点最大

无需安装，只要打开微信就能用，不占手机内存，体验好

开发周期段，一般最多一个月就可以上线完成

开发所需的资金少，所需资金是开发原生APP的一半不到

小程序名称是唯一的，在微信的搜索里权重很高

容易上手，只要之前有HTML+CSS+JS基础知识，写小程序基本没有大问题

基本不需要考虑兼容性问题，只要微信可以正常运行的机器，就可以运行小程序

发布，审核高效，基本上午发布审核，下午就审核通过，升级简单，支持灰度发布

开发文档完善，社区活跃

支持插件式开发，一些基本功能可以开发成插件，供多个小程序使用

**缺点：**

局限性很强（比如页面大小不能超过1M，不能打开超过5个层级的页面，样式单一，小程序的部分组件已经是成型的了，样式不能修改，比如幻灯片，导航）只能依赖于微信依托与微信，无法开发后台管理功能

不利于推广，推广面窄，不能分享朋友圈，只能分享给朋友，附近小程序推广，其中附加小程序也收到微信限制

后台调试麻烦，因为API接口必须https请求，且公网地址，也就是说后台代码必须发布到远程服务器上；当然我们可以修改host进行dns映射把远程服务器转到本地，或者开启tomcat远程调试；不管怎么说终归调试比较麻烦

前台测试有诸多坑，最头疼莫过于模拟器与真机显示不一致

js引用只能使用绝对路径，不能操作DOM

**原生App优点**

原生的相应速度快

对于有无网络操作时，譬如离线操作基本选用原生开发

需要调用系统硬件的功能（摄像头，拨号，短信蓝牙..）

在无网络或者弱网情况下体验好

**原生App缺点**

开发周期长，开发成本高，需要下载

# 小程序的发布流程（开发流程）

注册微信小程序账号

获取微信小程序的AppID

下载微信小程序开发者工具

创建demo项目

去微信公众号配置域名

手机浏览

代码上传

提交审核

小程序发布

# 如何封装小程序请求

**思路：**封装 wx.request 请求传递需要的参数( url , data , method , success 成功回调 ， fail 失败回调 ) , 封装常用方法 POST , GET , DELETE , PUT .... 最后导出这些方法；然后新建一个 api.js 文件，导入封装好的方法，然后调取相应的方法，传递数据。

### **wx.request 封装**

var app = getApp(); //获取小程序全局唯一app实例

var host = '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'; //接口地址

//POST请求

function post(url, data, success,fail) {

request(url, postData, "POST", doSuccess, doFail);

}

//GET请求

function get(url, data, success, fail) {

request(url, postData, "GET", doSuccess, doFail);

}

function request(url, data, method, success, fail) {

wx.showLoading({

title: "正在加载中...",

})

wx.request({

url: host + url, //请求地址

method: method, //请求方法

header: { //请求头

"Content-Type": "application/json;charset=UTF-8"

},

data: data, //请求参数

dataType: 'json', //返回数据格式

responseType: 'text', //响应的数据类型

success: function(res) {

wx.hideLoading();

//成功执行方法，参数值为res.data,直接将返回的数据传入

success(res.data);

},

fail: function() {

//失败执行方法

fail();

},

})

}

module.exports = {

postRequest: post,

getRequest: get,

}

### **组件使用 封装好的请求**

var http = require('../../utils/request.js'); //相对路径

var params = {//请求参数

id:this.data.userId

}

http.postRequest("user/delUser", params, function(res) {

console.log("修改成功！");

}, function(res) {

console.log("修改失败！！！")

})

# WEB 前端项目开发流程

**项目需求分析**

这个环节是由项目经理完成，项目经理首先和客户进行交流，了解客户的需求，然后分析项目的可行性，如果项目可以被实现，项目经理写出项目需求文档交给设计师完成后续的开发。

**页面设计/项目选型**

这个环节主要是 UI 设计师参与，UI 设计师根据产品需求分析文档，对产品的整体美术风格、交互设计、界面结构、操作流程等做出设计。负责项目中各种交互界面、图标、LOGO、按钮等相关元素的设计与制作。并且确定使用技术

**编码**

这个部分由程序员来实现。（程序员分为 WEB 前端开发工程师和后台开发工程师。前端开发人员主要做我们可以在网页上看的见的页面，后台就做一些我们看不见的管理系统以及功能的实现。）程序员根据 UI 设计师的设计，用编码来完成整个项目的各个功能。

**测试**

这部分由程序测试员来完成。程序测试员主要就是测试寻找程序还存在的 bug，一般来说刚编码完成的程序都是存在问题的，就需要测试人员反复不断的测试并将存在问题的测试结果交给编码人员进行bug 的修复。等到几乎所有 bug 修复完成，这个项目差不多就可以上线了。

**维护**

程序的维护是整个项目的最后一个阶段，但也是耗时最多，成本最高最高的的一个阶段。程序的维护包括程序上线后后续 bug 的修复和程序版本的更新。

**更换接口域名**

就是在开发的时候调用的后台接口是后台测试的接口 项目上线后要把请求的接口替换成上线的域名

# 混入的使用场景

当我们存在多个组件中的数据或者功能很相近时，我们就可以利用mixins将公共部分提取出来，通过 mixins封装的函数，组件调用他们是不会改变函数作用域外部的。

作用:减少data、methods、钩子的重复

当组件和mixin定义一个同名变量时，组件的值会覆盖mixin的值

# Axios拦截器的作用

比如一些网站过了一定的时间不进行操作，就会退出登录让你重新登陆页面，当然这不用拦截器你或许也可以完成这功能，但是会很麻烦而且代码会产生大量重复，所以我们需要用到拦截器

应用场景：

1：每个请求都带上的参数，比如token，时间戳等。

2：对返回的状态进行判断，比如token是否过期

# vue性能优化

1. 代码模块化
2. for循环设置key值，在用v-for进行数据遍历渲染的时候，为每一项都设置唯一的key值，为了让Vue内部核心代码能更快地找到该条数据，当旧值和新值去对比的时候，可以更快的定位到diff
3. Vue路由设置成懒加载，当首屏渲染的时候，能够加快渲染速度
4. 可以使用keep-alive，keep-alive是Vue提供的一个比较抽象的组件，用来对组件进行缓存，从而节省性能
5. 减少图片使用
6. 按需引入，咱们使用的一些第三方库可以通过按需引入的方式加载。避免引入不需要使用的部分

# react中类组件和函数组件的区别

1.首先函数组件是一个函数，他接受一个props对象，返回一个jsx，类组件是需要继承component这个类并且创建render函数返回一个jsx，

2.函数组件没有状态和生命周期，他需要使用hooks来模拟状态和生命周期

3.函数组件不用考虑this

# react的优化

1.像类组件中的key使用purecomponent

2.使用生命周期函数componentDidupdate来做优化，他有两个参数，一个是更新后的props和一个更新后的state，他返回true继续渲染，返回false就是停止渲染

3.像函数组件可以使用usemome缓存计算后的值，使用usecallback缓存一些函数，或者子组件啥的

4.列表渲染的时候使用key，增加diff对比速率

5.使用react.fragments避免额外标记

6.像类组件可以使用immutable对象

7.避免使用内联样式

8.在类组件中避免在will系列的生命周期中做异步请求，操作dom

# uni-app和taro是怎么选型的

uni-app和taro都可以做多端开发，uni-app支持vue的语法，taro可以使用react的语法

像uni-app可以使用#ifdef来判断平台环境，在# ifdef platform（比如写：h5） #endif 在这两个之间写代码他就只在h5页面中显示，通过这种方式可以做到多个平台共用一套代码

像taro的话if（process.env.TARO.ENV===weapp）{}

taro就是使用process.env.TARO.ENV来进行判断，通过这种方式来做到多个平台共用一套代码

像小程序的生命周期 比如onload，onReady，onShow，onHide，onUnLoad，我们常用的就是一个onLoad

其他的就是使用hbuiledrX来书写代码

# 介绍盒模型（盒子模型）

一个盒模型由四个部分组成：content,padding,border,margin

标准盒模型：盒子总宽度：w+p+m+b

怪异盒模型：w(包括p+b)+m

开启怪异盒模型：box-sizin：border-box

# set和Map区别

1.Map是键值对，Set是值的集合，map类似于对象是以键值对形式存储数据，区别是对象只能存储字符串或者symbol类型的数据，set不能重复，可以用来数去重。

2.Map可以通过get方法获取值，而set不能因为它只有值

3.都是迭代器，可以用for...of遍历

# for in 和 for of的区别

1、推荐在循环对象属性的时候使用 for...in，在遍历数组的时候的时候使用 for...of

2、for...in 循环出的是 key，for...of 循环出的是 value

3、注意，for...of 是 ES6 新引入的特性。修复了 ES5 引入的 for...in 的不足

4、for...of 不能循环普通的对象（如通过构造函数创造的），需要通过和 Object.keys()搭配使用

# react中间件thunk和saga的区别

* redux-thunk

优点：库小，代码就几行，采用的是async/await,代码比较优雅

缺点：代码臃肿，reducer不再是纯粹函数 直接返回对象，违背了当初的设计原则；action的形式不统一，异步操作太为分散，分散在了各个action中

* redux-saga

优点：采取generate函数，将异步与reducer区分开了，更加优雅，适合大量APi请求，而且每个请求之间存在复杂的以来关系，它的action总是一个对象

缺点：学习成本较高，需要理解generate；而且库也比较大，即使发布的最小也有25kb,gzip压缩后也有7KB,React压缩后才45kb

* 区别

thunk异步采取 async/awaitsaga采取generate函数

# vue中$route和$router的区别

$route用来获取路由的信息的，它是路由信息的一个对象，里面包含路由的一些基本信息，包括name、meta、path、hash、query、params、fullPath、matched、redirectedFrom等。

$router主要是用来操作路由的，它是VueRouter的实例，包含了一些路由的跳转方法，钩子函数等

# 封装echarts

1.npm i echarts 安装依赖包

2.xAxis表示的是X轴数据，yAxis表示的是Y轴数据，series.data表示的是折线数据

3.封装组件要多考虑一下，不能局限于现有的需求，需求随时都在变化，所以我们封装的时候要注意代码的可扩展性

# web常用协议

DNS协议

DNS协议的作用是将域名解析为IP

TCP/IP协议

1、TCP和IP协议，通常会把他们放在一起，其实他们是不同的两种协议，作用也不一样。

2、IP协议：是用来查找地址，对应网际互联层，TCP协议：是用来规范传输规则的，对应的传输层。IP只负责找到地址，具体怎么传输是由TCP来完成。

# 常见浏览器兼容性问题及解决办法总结

1. 不同浏览器的标签默认的外补丁( margin )和内补丁(padding)不同；

解决办法：在CSS文件开头，加\*{margin: 0; padding: 0;}，把所有标签的margin和padding设置为0即可

1. IE6及更低版本浮动元素，浮动边双倍边距

解决方案：不使用margin ，使用 padding

1. JS兼容问题
   1. 滚动事件的兼容
   2. 阻止冒泡的兼容
   3. 阻止默认行为的兼容

办公所需的软件

开发 vs code

任务进度查看，日志填报：jira--领导安排任务

查看需求：confluence

日常打卡，请假：北森

如果加班：分贝通

# grid和flex区别是什么?

1.Flexbox是一维布局系统，适合做局部布局，比如导航栏组件。2.Grid是二维布局系统，通常用于整个页面的规划。

3.二者从应用场景来说并不冲突。虽然 Flexbox也可以用于大的页面布局，但是没有Grid 强大和灵活。二者结合使用更加轻松。

4.flex布局和Grid布局有实质的区别，那就是flex布局是一维布局，Grid布局是二维布局。flex布局一次只能处理一个维度上的元素布局，一行或者一列。Grid布局是将容器划分成了“行"和“列"，产生了一个个的网格，我们可以将网格元素放在与这些行和列相关的位置上，从而达到我们布局的目的。

# vue权限管理

**路由权限控制：**

方案一：初始化即挂载全部路由，并且在路由上标记相应的权限信息，每次路由跳转前做校验

这种方式存在以下四种缺点：

加载所有的路由，如果路由很多，而用户并不是所有的路由都有权限访问，对性能会有影响。

全局路由守卫里，每次路由跳转都要做权限判断。

菜单信息写死在前端，要改个显示文字或权限信息，需要重新编译

菜单跟路由耦合在一起，定义路由的时候还有添加菜单显示标题，图标之类的信息，而且路由不一定作为菜单显示，还要多加字段进行标识

方案二：初始化的时候先挂载不需要权限控制的路由，比如登录页，404等错误页。如果用户通过URL进行强制访问，则会直接进入404，相当于从源头上做了控制

登录后，获取用户的权限信息，然后筛选有权限访问的路由，在全局路由守卫里进行调用addRoutes添加路由

这种方式也存在了以下的缺点：

全局路由守卫里，每次路由跳转都要做判断

菜单信息写死在前端，要改个显示文字或权限信息，需要重新编译

菜单跟路由耦合在一起，定义路由的时候还有添加菜单显示标题，图标之类的信息，而且路由不一定作为菜单显示，还要多加字段进行标识

**菜单权限控制：**

菜单权限可以理解成将页面与理由进行解耦

方案一：菜单与路由分离，菜单由后端返回

前端定义路由信息之后，再在全局路由守卫里做判断

{

name: "login",

path: "/login",

component: () => import("@/pages/Login.vue")

}

这种方式的缺点：

菜单需要与路由做一一对应，前端添加了新功能，需要通过菜单管理功能添加新的菜单，如果菜单配置的不对会导致应用不能正常使用

全局路由守卫里，每次路由跳转都要做判断

方案二：菜单和路由都由后端返回

前端统一定义路由组件

|  |
| --- |
| const Home = () => import("../pages/Home.vue");  const UserInfo = () => import("../pages/UserInfo.vue");  export default {  home: Home,  userInfo: UserInfo  }; |

后端路由组件返回以下格式

|  |
| --- |
| [  {  name: "home",  path: "/",  component: "home"  },  {  name: "home",  path: "/userinfo",  component: "userInfo"  }  ] |

这种方法也会存在缺点：

全局路由守卫里，每次路由跳转都要做判断

前后端的配合要求更高

# 前端性能优化

1、降低请求量

​ ① 合并资源，减少http请求数量。

​ ② lazyLoad，如图片懒加载。分批加载，每次只加载一部分。

​ ③ 使用字体图标或CSS绘制，来代替部分图片。

2、加快请求速度

​ ① 预解析DNS

​ ② 使用HTTP2.0

​ ③ 并行加载

​ ④ CDN 分发

​ ⑤ webP，对图片进行压缩，减少图片体积。

​ ⑥ minify/gzip 压缩，对css、js等文件进行压缩（去除空格、回车等），减少文件体积

补充知识：

​ webP 的优势体现在它具有更优的图像数据压缩算法，能带来更小的图片体积，而且图像质量几乎无差异。同时具备了无损和有损的压缩模式、Alpha 透明以及动画的特性，在 JPEG 和 PNG 上的转化效果都相当优秀。

​ Minify把 CSS 和 JS 压缩和削减(Minify：去掉空格回车符等)，以及把多个CSS，JS文件整合到一个文件里。

3、缓存

​ ① HTTP协议缓存请求

​ ② 离线缓存 manifest

​ ③ 本地缓存 localStorage

补充知识：

​ GET请求可以缓存，POST请求不能缓存。GET请求后退/刷新无害，POST后退/刷新则会致使重新提交数据

4、渲染

​ ① JS优化，如防抖、节流、事件委托、减少重排重绘等。

​ ② CSS优化，如提取公共样式减少代码量、减少选择器嵌套层数、精灵图等。

​ ③ 服务器端渲染

​ ④ 使用Web Workers

​ ⑤ CSS写在文件头部，JS写在文件底部。

# 发布订阅和观察者模式的区别

实际上观察者模式是包含了订阅发布模式，发布订阅模式只是观察者模式中的一种。观察者模式是观察者和被观察者之间的通信，而发布订阅模式中间增加了一个中转层

vue中数据劫持中就用到了观察者模式

vue中的兄弟组件通信bus的原理就是发布订阅模式，该模式有个缺点，当你订阅一个消息后，也许此消息最后都未发生，但这个订阅者会始终存在于内存中。所以该消息不使用的时候，调用$of销毁，以避免内存泄漏

# vue父子组件生命周期执行顺序

**加载渲染过程**

父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount->子mounted->父mounted

**更新过程**

父beforeUpdate->子beforeUpdate->子updated->父updated

**销毁过程**

父beforeDestroy->子beforeDestroy->子destroyed->父destroyed

# 前端项目打包上线部署流程

服务器安装Nginx

先查看服务器是否有nginx yum install -y nginx

启动nginx

重启nginx服务，一般用于修改配置文件之后，命令如下：

使用vue打包命令,生成dist文件夹：

（2）上传至服务器

解决刷新路由404问题

这是因为我们vue项目采用了history的路由方式，至于原因主要是vue是单页面应用，细节部分大家可自行下来了解。

（1）将路由模式改为hash

（2）修改nginx配置：

# v-if和v-show的区别

v-if直接对 节点/组件实例 进行卸载；

v-show 对节点进行disPlay:none;

**细节**

控制手段：

v-show隐藏则是为该元素添加css--display:none，dom元素依旧还在。v-if显示隐藏是将dom元素整个添加或删除

编译过程

v-if切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件；

v-show只是简单的基于css切换

编译条件：

v-if是真正的条件渲染，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。只有渲染条件为假时，并不做操作，直到为真才渲染

**共同点**

我们都知道在 vue 中 v-show 与 v-if 的作用效果是相同的(不含v-else)，都能控制元素在页面是否显示

# dist和lib和es区别

es：es5版本 lib：es6版本

# 所以说为什么会使用虚拟Dom

1. 减少不必要的重排和重绘（提高性能）
2. 提高项目的可维护性（会降低部分性能，但是对于可维护性的角度来说，这部分性能的损耗是能够接受的）

# ES6模块化与commonJS区别：

1. commonJS require的加载方式为‘运行时加载’，即require资源之后，生成一个对象\_fs，然后再去\_fs中读取需要的内容，只有在运行时才能得到这个引入的对象
2. ES6的这种加载称为“编译时加载”或者静态加载，即 ES6 可以在编译时就完成模块加载，效率要比 CommonJS 模块的加载方式高。因为是编译时加载，所以使用import命令时不能使用表达式和变量，这些只有运行时才能的得到结果的语法结构

**es6**

**export命令**

一个模块就是一个独立的文件，该文件内部的所有变量，在外部都是访问不到的，如果想要被外部访问到，那么就需要使用export命令导出

export命令不能用于块级作用域中

**export defaut 命令**

使用import命令时，我们需要知道需要加载的变量名和函数名，否则无法正确加载。

使用export default命令，可以使我们为模块指定默认输出，在导入时在为其命名

**import命令**

import载入的变量都是只读的

import语句存在变量提升，即会提升到模块头部，优先执行

export import命令是编译时执行的，只能运行在模块顶层，不能运行在块级作用域中

# 浏览器如何渲染网页

浏览器渲染的过程主要包括以下五步：

1. 浏览器将获取的HTML文档解析成DOM树。
2. 处理CSS标记，构成层叠样式表模型CSSOM(CSS Object Model)。
3. 将DOM和CSSOM合并为渲染树(rendering tree)，代表一系列将被渲染的对象。
4. 渲染树的每个元素包含的内容都是计算过的，它被称之为布局layout。浏览器使用一种流式处理的方法，只需要一次绘制操作就可以布局所有的元素。
5. 将渲染树的各个节点绘制到屏幕上，这一步被称为绘制painting。

需要注意的是，以上五个步骤并不一定一次性顺序完成，比如DOM或CSSOM被修改时，亦或是哪个过程会重复执行，这样才能计算出哪些像素需要在屏幕上进行重新渲染。而在实际情况中，JavaScript和CSS的某些操作往往会多次修改DOM或者CSSOM。

# for in 和 for of区别

**区别一：**

for in 和 for of 都可以循环数组，for in 输出的是数组的index下标，而for of 输出的是数组的每一项的值。

for in 简要说明: 这个循环很多人爱用，但实际上，经分析测试，在众多的循环遍历方式中它的效率是最低的

|  |
| --- |
| const arr = [1,2,3,4]  // for ... in  for (const key in arr){  console.log(key) // 输出 0,1,2,3  }  // for ... of  for (const key of arr){  console.log(key) // 输出 1,2,3,4 |

**区别二：**

for in 可以遍历对象，for of 不能遍历对象，只能遍历带有iterator接口的，例如Set,Map,String,Array

|  |
| --- |
| const object = { name: 'lx', age: 23 }  // for ... in  for (const key in object) {  console.log(key) // 输出 name,age  console.log(object[key]) // 输出 lx,23  }  // for ... of  for (const key of object) {  console.log(key) // 报错 Uncaught TypeError: object is not iterable |

3.数组对象

|  |
| --- |
| for (const val of list) {  console.log(val) // 输出{ name: 'lx' }, { age: 23 }  for (const key in val) {  console.log(val[key]) // 输出 lx,23  } |

for in适合遍历对象，for of适合遍历数组

for in遍历的是数组的索引，对象的属性，以及原型链上的属性

for of这种方式是es6里面用到的，性能要好于for in，但仍然比不上普通for循环

# forEach和map区别

1. 相同点
   1. 都是循环遍历数组中的每一项。
   2. 每次执行匿名函数都支持三个参数，参数分别为item(当前每一项)，index(索引值)，arr(原数组)。
   3. 匿名函数中的this都是指向window。
   4. 只能遍历数组。
   5. map和forEach都不会影响原数组
2. 不同点
   1. map()会分配内存空间存储新数组并返回，forEach()不会返回数据。
   2. forEach()允许callback更改原始数组的元素。map()返回新的数组。
3. map这种方式也是用的比较广泛的，虽然用起来比较优雅，但实际效率还比不上foreach
4. map会分配内存空间存储新数组并返回，而forEach不会返回执行结果，会返回undefined，若想返回需要抛出异常或者try catch；
5. 由于forEach是Array型自带的，对于一些非这种类型的，无法直接使用(如NodeList)，所以才有了这个变种，使用这个变种可以让类似的数组拥有foreach功能。

**使用场景：**

map适用于你要改变数据值的时候，不仅在于它更快，而且返回一个新的数组，这样可以提高复用性（map(),filter(),reduce()）等组合使用。

# vue实现图片上传

**elementui自带的el-upload上传**

实现思路：

1.通过点击事件打开文件夹，选取需要上传的图片

2.拿到图片数据，将其添加到FromData实例对象中

3.将图片数据上传至服务器

<el-upload action="" :on-change="handleelchange" :auto-upload="false" list-type="picture-card">

<i class="el-icon-plus"></i>

</el-upload>

handleelchange(file, fileList) {

console.log(file, fileList)

let formdata = new FormData()

fileList.map(item => { //fileList本来就是数组，就不用转为真数组了

formdata.append("file", item.raw) //将每一个文件图片都加进formdata

})

axios.post("接口地址", formdata).then(res => { console.log(res) })

},



# js实现图片上传

**利用原始input标签form表单上传**

通常使用FormData对象来存放上传图片的数据流，然后上传FormData对象

<template>

<div>

<input type="file" @change="changeFile">

<button @click="upload()">上传</button>

</div>

</template>

<script>

import axios from 'axios'

export default {

data(){

return{

file:null,

}

},

methods:{

upload(){

// 上传图像通常要将文件参数放在fromdata 对象中

// 创建formData实例

let formData=new FormData()

// 向formData实例中追加属性 (参数1：属性名 参数2：属性值)

formData.append('files',this.file)

console.log(formData);//这里输出的是一个空的formData对象，因为formData是加密处理的我们看不到其内容，但的的确确添加进去了。

axios.post('上传路径',formData).then(res=>{

console.log(res);

})

},

changeFile(e){

// 获取文件信息 e.target.files

console.log(e.target.files[0]);

this.file=e.target.files[0];

}

}

}

</script>



**总结：**

1.获取文件对象 ：e.target.files[0]

2.将文件对象存储在FormData实例中：

let formData=new FormData()

// 向formData实例中追加属性 (参数1：属性名 参数2：属性值)

formData.append('参数1','参数2')

3.发生请求，请求参数为创建的FormData实例

axios.post('请求路径',formData).then(res=>{

console.log(res);

})

# 进程与线程

都是对操作系统的基本概念：两个名词都是CPU工作时间片的一个描述；

1. 进程是CPU资源分配的最小单位；（是拥有资源和独立运行的最小单位）
2. 线程是CPU调度的最小单位（是建立在进程基础上的一次程序运行单位）

**关系**

1. 每一个线程至少要做一件事，所以一个进程至少有一个线程，
2. 系统会给每个线程分配独立的内存，因此该线程有独立的资源；
3. 同一进程内的各个线程之间共享该进程的内存空间

进程可以理解为一个工厂不不同车间，相互独立。线程是车间里的工人，可以自己做自己的事情,也可以相互配合做同一件事情。

# webpack将es6生成es5

1. 安装三个包
2. 然后是配置webpack.config.js



上面的配置重点在两个地方，一个是exclude:/(node\_modules| bower\_components)

它的意思就是说node\_modules或者bower\_components文件夹中的js文件如果有ES6的语法，不要给我转为ES5。

还有一个是presets: ['es2015'], 它的意思就是babel-loader的配置是根据es2015来，具体这个作用我不是很懂

# 定位position



# http三个版本的区别

1.http1

加入了TCP常连接通信，每当建立起连接后，只要任意端口没有明确提出断开连接，TCP就会保持链接状态，大大减少了频繁建立连接的开销；

2.http2

基于http1做了一些内容：

头部压缩，http2中会压缩头，如果客户端发送多个请求，头是相似或者相同，那么协议就会消除重复的部分，减少不必要的浪费；

http1中报文是文本格式，在http2中报文是二进制格式，大大加快了传输效率；

http2中使用了多路复用，解决了队头阻塞的问题，后面的请求无法进行请求（因为TCP是可靠链接，顺序执行）。多路复用是多个http复用一个TCP链接，那么多路复用后，请求和响应不再是按照顺序一一对应，如果一个请求被堵塞，其他请求可以继续；

http2优化http1，但也出现了一些问题

3.http3

http2在多路复用时是复用了一个TCP链接，那么TCP一旦发送丢包，那么就会阻塞复用它的http，这里也是因为TCP是可靠运输，所以加入了基于UDP，UDP是不可靠传输，所以基于UDP的QUIC协议，类似实现了TCP的可靠传输；

http3将TCp三次握手四次挥手直接合并成了三次握手，，减少了链接次数；

# 当await等待过程中有多个错误，你如何去捕获，不一定全是resolve的。

**方法一：try catch 代码冗余---不建议**

try {

let res= await getData()

}catch (){

console.log(res)

}

**方法二：链式调用**

let res=await getData().then(()=>{}).catch(err=>[err,null])

**方法三：封装一个函数（传入一个promise）返回一个值**

function awaitWraper(promise){

return promise.then(res=>[null,res]))

.catch(err=>[err,null])

}

let [res,err]=await awaitWraper(getData())

# 谈谈Promise中断或者取消。

中止：通过返回一个新的Promise实例

Promise.resolve().then(()=>{

console.log("1")

return new Promise(()=>{})

}).then(()=>{

//上述返回一个新的promise对象后，不会执行下面的，达到中断

console.log("2")

}).catch(()=>{

console.log("4")

})

通过promise.race竞速方法，可以实现，不在执行

let p1 = newPromise((resolve, reject) => { resolve('ok1') })

let p1 = newPromise((resolve, reject) => { resolve('ok1') })

Promise.race([p1,p2]).then((result)=>{

console.log(result) // ko1

}).catch(()=>{

})

当Promise链中抛出一个错误时，错误信息沿着链路向后传递，直至被捕获。

# vue中使用history模式后，会出现404问题？

分析：由于vue-router设置的路径不是真实存在的路径，故页面刷新后访问的资源在服务端找不到

**解决方法一：webpack配置文件修改( for 本地调试）**

在devServer里面有个historyApiFallback的属性，用于找不到页面时返回默认目录的index.html。

module.exports={

devServer:{

historyApiFallback:{

rewrites:[

from:/.\*/g,

to:'/pages/index.html'

]

}

}

}

**生产环境下：解决404的方法**

所以呢，你要在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面，这个页面就是你 app 依赖的页面。

vue路由模式设置为history时再部署到nginx刷新或者路由地址错误的时候会报nginx的404错误，只要在nginx的配置文件中加入这个就可以啦

修改nginx文件：

// ...

location / {

root /opt/nginx/html/;

try\_files $uri $uri/ /index.html; // 按指定的file顺序查找存在的文件，并使用第一个找到的文件进行请求处理;

}

# vue封装组件的过程

**具体步骤：**

1）使用 Vue.extend（）方法创建一个组件

2）然后使用 Vue.component （）方法注册组件

3）如果 子组件需要数据，可以在 props 中接受定义

4）最后，子组件修改好数据后，想把数据传递给父组件， 可以采用$emit（） 方法

# 阐述一下em, rem, px 的区别

1.px是固定的像素，一旦设置了就无法因为适应页面大小而改变。

2，em,rem相对于px 更具有灵活性，他们是相对长度单位。意思是长度不是定死了的，更实用于响应式布局。

3，em 是相对于其父元素来设置字体大小的，一般都是以<body> 的“font-size"为基准。这样就会存在一个问题，进行任何元素设置，都有可能需要知道他父元素的大小。而rem 是相对于根元素《html》，这样就意味着，我们只需要在根元素确定一个参考值。 总之，em 相对于父元素，rem 相对于根元素。

# 如何实现一个dom元素从无到有的过度

使用transition 对元素的不透明度 opacty进行过度，或者是对visibility的visible进行进行过度

# 柯里化

**1，柯里化的定义：**

把接受多个参数的函数变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。

**2，柯里化的优点**

**1，参数复用**

// 正常正则验证字符串 reg.test(txt)

// 函数封装后

function check(reg, txt) {

return reg.test(txt)

}

check(/\d+/g, 'test') //false

check(/[a-z]+/g, 'test') //true

// Currying后

function curryingCheck(reg) {

return function(txt) {

return reg.test(txt)

}

}

var hasNumber = curryingCheck(/\d+/g)

var hasLetter = curryingCheck(/[a-z]+/g)

hasNumber('test1') // true

hasNumber('testtest') // false

hasLetter('21212') // false

上面的示例是一个正则的校验，正常来说直接调用check函数就可以了，但是如果我有很多地方都要校验是否有数字，其实就是需要将第一个参数reg进行复用，这样别的地方就能够直接调用hasNumber,hasLetter等函数，让参数能够复用，调用起来也更方便。

**2，提前确认**

var on = function(element, event, handler) {

if (document.addEventListener) {

if (element && event && handler) {

element.addEventListener(event, handler, false);

}

} else {

if (element && event && handler) {

element.attachEvent('on' + event, handler);

}

}

}

var on = (function() {

if (document.addEventListener) {

return function(element, event, handler) {

if (element && event && handler) {

element.addEventListener(event, handler, false);

}

};

} else {

return function(element, event, handler) {

if (element && event && handler) {

element.attachEvent('on' + event, handler);

}

};

}

})();

//换一种写法可能比较好理解一点，上面就是把isSupport这个参数给先确定下来了

var on = function(isSupport, element, event, handler) {

isSupport = isSupport || document.addEventListener;

if (isSupport) {

return element.addEventListener(event, handler, false);

} else {

return element.attachEvent('on' + event, handler);

}

}

我们在做项目的过程中，封装一些dom操作可以说再常见不过，上面第一种写法也是比较常见，但是我们看看第二种写法，它相对一第一种写法就是自治性然后返回一个新的函数，这样其实就是提前确定了会走哪一个方法，避免每次都进行判断。

**3，延迟执行**

Function.prototype.bind = function (context) {

var \_this = this

var args = Array.prototype.slice.call(arguments, 1)

return function() {

return \_this.apply(context, args)

}

}

像我们js中经常使用的bind，实现的机制就是Currying

说了这几点好处之后，发现还有个问题，难道每次使用Curring都要对底层函数去做修改

有没有什么通用的封装方法？

// 初步封装

var currying = function(fn) {

// args 获取第一个方法内的全部参数

var args = Array.prototype.slice.call(arguments, 1)

return function() {

// 将后面方法里的全部参数和args进行合并

var newArgs = args.concat(Array.prototype.slice.call(arguments))

// 把合并后的参数通过apply作为fn的参数并执行

return fn.apply(this, newArgs)

}

}

这边首先是初步封装，通过闭包把初步参数给保存下来，然后通过获取剩下的arguments进行拼接，最后执行需要curring的函数

这样返回的话其实只能多扩展一个参数，curring(a)(b)(c)这样的话，貌似就不支持了（不支持多参数调用），一般这种情况都会想到使用递归再进行封装一层。

// 支持多参数传递

function progressCurrying(fn, args) {

var \_this = this

var len = fn.length;

var args = args || [];

return function() {

var \_args = Array.prototype.slice.call(arguments);

Array.prototype.push.apply(args, \_args);

// 如果参数个数小于最初的fn.length，则递归调用，继续收集参数

if (\_args.length < len) {

return progressCurrying.call(\_this, fn, \_args);

}

// 参数收集完毕，则执行fn

return fn.apply(this, \_args);

}

}

这边其实就是在初步的基础上，加了递归的调用，只要参数小于最初的fn.length,就会继续执行递归

**curry的一些性能问题：**

1，存取arguments对象通常要比存取命名参数要慢一点

2，一些老版本的浏览器在argument.length的实现上是相当慢的

3，使用fn.apply(...)和fn.call(...)通常比直接调用fn(...)稍微慢点

4，创建大量嵌套作用域和闭包函数会带来花销，无论是在内存还是速度上

其实大部分应用中，主要的性能瓶颈是在操作DOM节点上，这js的性能损耗基本是可以忽略不计的所以curry是可以直接放心使用的。

最后在扩展一道经典面试题

// 实现一个add方法，使计算结果能够满足如下预期：

add(1)(2)(3) = 6;

add(1, 2, 3)(4) = 10;

add(1)(2)(3)(4)(5) = 15;

function add() {

// 第一次执行时，定义一个数组专门用来存储所有的参数

var \_args = Array.prototype.slice.call(arguments);

// 在内部声明一个函数，利用闭包的特性保存\_args并收集所有的参数值

var \_adder = function() {

\_args.push(...arguments);

return \_adder;

};

// 利用toString隐式转换的特性，当最后执行时隐式转换，并计算最终的值返回

\_adder.toString = function () {

return \_args.reduce(function (a, b) {

return a + b;

});

}

return \_adder;

}

add(1)(2)(3) // 6

add(1, 2, 3)(4) // 10

add(1)(2)(3)(4)(5) // 15

add(2, 6)(1) // 9

# http 和 https 的区别

HTTP协议：

全称是HyperText Transfer Protocol,中文意思是超文本传输协议，是一种发布和接收HTML页面的方法。服务端口号是80端口

HTTPS协议：

是HTTP协议的加密版本，在HTTP下加入了SSL层。服务 器端口号是443端口

1,https协议需要到ca申请证书，一般免费证书很少，需要交费。

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

3，http和https使用完全不同的连接方式用的端口也不一样，前者是80，后者是443.

4,http的连接是简单的，是无状态的

1. 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，要比http协议安全。

# 如何保持 接口安全和数据安全

1，保持接口安全

使用安全协议https

使用黑白名单

测试环境和生产环境分开

2，数据安全

请求头token验证

对请求频率的限制

对敏感数据进行加密

# 请简述虚拟dom转换为真实dom的过程

虚拟DoM是一个轻量级的javascript对象，它最初知识真实DOM的一个副本，并不是完全拥有真实DOM的所有属性以及方法，它是一个节点数，它将元素，属性和内容作为对象以及属性，vue/react的渲染函数从组件里创建一个节点树，然后响应数据的变化来更新这个节点数，这个变化是用户或者系统完成的动作引起的

虚拟DOM工作三个重要步骤

1，底层数据发生变化时，整个UI都将在虚拟DOM中重新渲染

2，计算之前DOM和更改数据的虚拟DOM之间不一样。

3，计算出不同时，真实DOM只渲染更改的差异，数据相同时不会进行重新渲染，节约渲染时间。

# ajax从0-4的五个状态分别是什么？

0：（未初始化）还没有调用open方法

1：（初始化）已经调用open方法，未派发请求（调用send)

2: (发送数据) send 已经调用，但是还没有接收到来自服务器的响应

3： （接收）已经接受部分数据，因为响应及http头并不完全，这时使用responseText获取部分数据会出现错误

4：（解析完成）数据接收完毕，可以在客户端响应

# 说一下$root，$parent，$refs区别

$root，和$parent都能访问父组件的属性和方法，区别在于如果存在多级子组件，通过parent 访问得到的是它最近一级的父组件，通过root 访问得到的是根父组件。通过在子组件标签定义 ref 属性，在父组件中可以使用$refs 访问子组件实例

# for、for in、for of、forEach的区别

迭代（iterate），指的是按照某种顺序反复多次执行一段程序，通常会有明确的终止条件。

遍历（traversal），指的是按照一定的规则访问树形结构中的每个节点，而且每个节点都只访问一次。

1. **for循环可以用于遍历数组，可以使用break，continue语句跳出循环**
2. **for in类似于增强for循环，我们可以简单理解为它适合遍历对象。应用场景:遍历数组以及遍历对象的属性**

let arr = {

name : '关羽',

sex:'男',

yrs:'33',

sleep:function(){},

eat :function(){}

}

for(attr in arr){//遍历arr对象的所有属性

console.log(attr)

}

1. **for of语句是一种严格的迭代语句，用于遍历可迭代对象的元素**

let arr1=[12,22,32,42,52]

let m= arr1.keys()

for(m1 of m){//遍历数组对象中的所有元素

console.log(m1)

}

1. **foreach 语句是对数组的每个元素执行一次提供的函数。是for语句的特殊简化版本，不能完全取代for语句，但任何foreach语句都可以改写为for语句版本**

let arr = [123,456,789,269,258]

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

console.log(arr[i])

}

var str = ''

arr.forEach(function(arr){//自循环：数组中的每个元素都执行forEach给定的函数

str+=arr+'\t'

})

1. **优缺点**
2. for:

优点：程序简洁，结构清晰，循环初始化，循环变量化和循环条件位置突出。

缺点：结构比while循环复杂，容易出编码错误。

1. for in:

优点：避免了for in的所有缺点，可以使用break,continue和return，不仅支持数组的遍历，还可以遍历类似数组的对象，支持字符串的遍历最简洁。

缺点：不适用于处理原有的原生对象。

1. for of:

优点：避免了for…in的所有缺点，支持break,continue,return。支持遍历map，object,array,set string等。

缺点：不适用于处理原有的原生对象。

1. forEach:

优点：便利的时候更加简洁，效率和for循环相同，不用关心集合下标的问题，减少了出错的效率

循环，或者使用return从函数体返回，对于空数组不会执行回调函数

1. 区别

四个算法语句区别主要体现在响应break, continue, return上和使用的对象上。

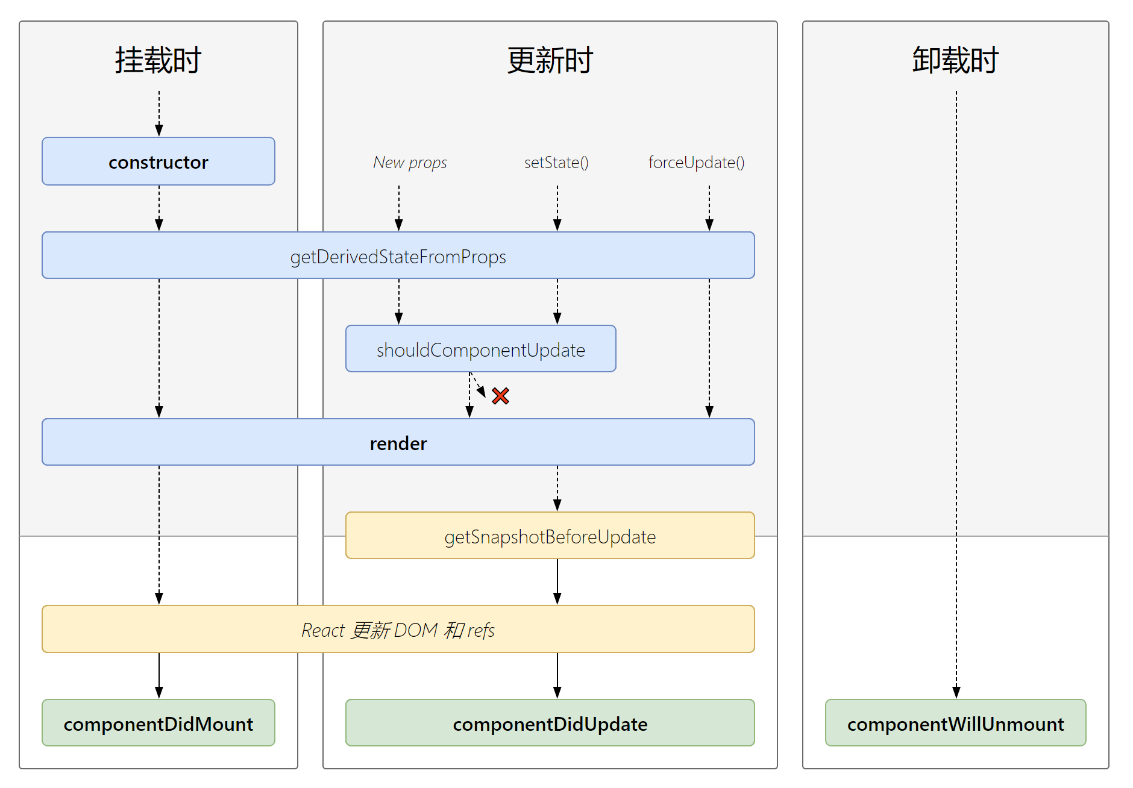
for 语句性能最好；能响应break, continue, return控制循环。

forEach 无法响应break, continue, return控制循环。

for in 无法响应break, continue, return控制循环；for in 主要针对对象，它不仅会循环对象本身的属性，还会查找循环原型上的属性；循环的顺序不确定。

for of 能响应break, continue, return控制循环，还能遍历map、set 等类数组，但是不能循环普通的对象

# react生命周期



初始化阶段：

getDefaultProps:获取实例的默认属性

getInitialState:获取每个实例的初始化状态

componentWillMount：组件即将被装载、渲染到页面上

render:组件在这里生成虚拟的 DOM 节点

componentDidMount:组件真正在被装载之后

运行中状态：

componentWillReceiveProps:组件将要接收到属性的时候调用

shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以返回 false，接收数据后不更新，阻止 render 调用，后面的函数不会被继续执行了）

componentWillUpdate:组件即将更新不能修改属性和状态

render:组件重新描绘

componentDidUpdate:组件已经更新

销毁阶段：

componentWillUnmount:组件即将销毁

# react生命周期中，最适合与服务端进行数据交互的是哪个函数

componentDidMount：在这个阶段，实例和dom已经挂载完成，可以进行相关的dom操作

# react组件生命周期的执行次数是什么样子的？？？

只执行一次： constructor、componentWillMount、componentDidMount

执行多次：render 、子组件的componentWillReceiveProps、componentWillUpdate、componentDidUpdate

有条件的执行：componentWillUnmount（页面离开，组件销毁时）

不执行的：根组件（ReactDOM.render在DOM上的组件）的componentWillReceiveProps（因为压根没有父组件给传递props）

# react组件的生命周期执行顺序是什么样子的？

**如果不涉及到setState更新，第一次渲染的顺序如下：**

App： constructor --> componentWillMount --> render -->

parent: constructor --> componentWillMount --> render -->

child: constructor --> componentWillMount --> render -->

componentDidMount (child) --> componentDidMount (parent) --> componentDidMount (App)

**这时候触发App的setState事件**

App： componentWillUpdate --> render -->

parent: componentWillReceiveProps --> componentWillUpdate --> render -->

child: componentWillReceiveProps --> componentWillUpdate --> render -->

componentDidUpdate (child) --> componentDidUpdate (parent) --> componentDidUpdate (App)

# v-if和v-for优先级

在vue2中,v-for的优先级高于v-if

在vue3中,v-if的优先级高于v-for

# 重置data的所有属性

1.拿到data原始属性

this.$options.data()

2.拿到当前的data

this.$data

3.复制到当前$data里面

Object.assign(this.$data, this.$options.data());

# h5新特性

1、拖拽释放（drag and drop）API

2、语义化更好的内容标签（header footer nav aside article section）

3、音频、视频（audio video）API

4、画布（Canvas）API

5、地理（Geolocation）API

6、localstorage 和 sessionstorage 缓存方式

7、表单控件（calendar date time email ul search）

8、新技术（webworker websocket Geolocation）

# css中实现元素隐藏的方法

1、display:none;

display翻译成中文是显示、展览的意思；将display的属性改为none可以实现元素的隐藏，元素和盒子模型也不生成，被隐藏的元素不占位置，看不见摸不着，它会导致浏览器的重排和重绘。

2、visibility:hidden;

visibility翻译成中文是能见、可见性的意思；hidden翻译成中文的是隐藏、不易察觉的意思。将visibility的属性改成hidden可以实现元素的隐藏，和display:none的区别是它占位置，看不见但是摸得着，所以它只会导致浏览器重绘而不会重排。

3、opacity:0;

opacity翻译成中文是透明度、不透明的意思；将opacity的属性改成0那就是透明度等于0看不见元素，但是这种方法也是只能隐藏看不见元素，和visibility:hidden一样，元素依然存在页面中。

4、position；

利用定位将元素的top和left值设为足够大的负数，使它移出屏幕在屏幕上看不见。

5、overflow:hidden；

overflow翻译成中文是漫出、溢出的意思；将overflow的属性设置hidden可以实现元素隐藏，但是这个是超出盒子的部分隐藏，有局限性。

# 判断一个对象是空对象

1. 将对象转换成字符串，再判断是否等于“{}”

let obj = {}

if(JSON.stringify(obj)=='{}'){

console.log('空对象')

}

1. es6新增的方法：Object.keys() ,返回对象的属性名组成的一个数组，若长度为0，则为空对象

let obj={}

if(Object.keys(obj).length==0){

console.log('空对象')

}

1. for...in...

let result=function(obj){

for(let key in obj){

return false;//若不为空，可遍历，返回false

}

return true;

}

console.log(result(obj));//返回true

# 浅谈canvas和svg的区别

（1）SVG： SVG可缩放矢量图形（Scalable Vector Graphics）是基于可扩展标记语言XML描述的2D图形的语言，SVG基于XML就意味着SVG DOM中的每个元素都是可用的，可以为某个元素附加Javascript事件处理器。在 SVG 中，每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化，那么浏览器能够自动重现图形。

其特点如下：

不依赖分辨率

支持事件处理器

最适合带有大型渲染区域的应用程序（比如谷歌地图）

复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）

不适合游戏应用

（2）Canvas： Canvas是画布，通过Javascript来绘制2D图形，是逐像素进行渲染的。其位置发生改变，就会重新进行绘制。

其特点如下：

依赖分辨率

不支持事件处理器

弱的文本渲染能力

能够以 .png 或 .jpg 格式保存结果图像

最适合图像密集型的游戏，其中的许多对象会被频繁重绘

矢量图，也称为面向对象的图像或绘图图像，在数学上定义为一系列由线连接的点。矢量文件中的图形元素称为对象。每个对象都是一个自成一体的实体，它具有颜色、形状、轮廓、大小和屏幕位置等属性。

**所以Canvas和SVG的区别可以总结为以下几点：**

1、Canvas是使用javaScript程序绘制动态生成的，SVG是使用xml文档描述来绘图。从这点来看：canvas更适合用来做动态交互，而且SVG绘图更容易编辑

1、SVG 不能绘制图片，而 canvas 可以。SVG是通过 XML 绘制，而Canvas通过 js 绘制

2、Canvas绘制的方式，是通过 js 逐像素渲染的。也就是说，它绘制一个复杂的图形和一个简单的图形的性能是差不多的。

SVG 是通过 XML 的方式渲染。它的本质是DOM，而复杂的图形，就会降低其渲染性能。

3、Canvas 是依赖分辨率，是一种标量图。所以在放缩的时候，存在失真的问题。

SVG 绘制的时候，不依赖分辨率，是一种矢量图。所以当SVG放缩的时候，不会使得图像失真。

4、SVG 适合带有大型渲染区域的应用程序：比如谷歌地图、百度地图。

Canvas 适合对象密集型的游戏应用场景。

# 清除浮动

1. 额外标签法

额外标签法也称为隔墙法，是W3C推荐的做法

额外标签会在浮动元素末尾添加一个空的标签。例如<div style="clear:both" > </div>，或者其他标签

优点：通俗易懂，书写方便

缺点：添加许多无意义的标签，结构化较差

1. 父级添加overflow

可以给父级添加overflow属性，将其属性值设置为hidden、auto或scroll

优点：代码简洁

缺点：无法显示溢出的部分

1. **清除浮动之:after伪元素法**

:after方式是额外标签法的升级版。也是给父元素添加

.clearfix:after {

content: "";

display: block;

height: 0;

clear: both;

visibility: hidden;

}

.clearfix { /\* IE6、7 专有 \*/

\*zoom: 1;

}

优点：没有增加标签，结构更简单

缺点：照顾低版本浏览器

1. 双伪元素清除浮动

也是给父元素添加

.clearfix:before,.clearfix:after {

content:"";

display:table;

}

.clearfix:after {

clear:both;

}

.clearfix {

\*zoom:1;

}

优点：代码更简洁

缺点：照顾低版本浏览器

# 在页面添加一个视频并且自动播放

<video width="320" height="240" controls autoplay>

<source src="movie.ogg" type="video/ogg">

您的浏览器不支持 video 标签。

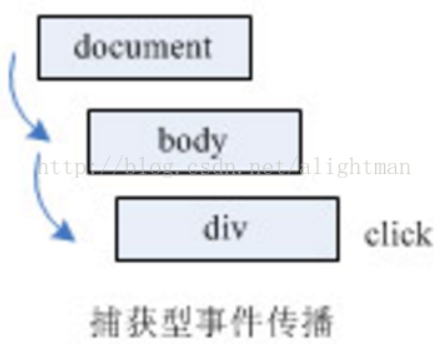
</video>

说明：<source>标签允许您规定可替换的视频/音频文件供浏览器根据它对媒体类型或者编解码器的支持进行选择。

**<video> 标签的autoplay 属性后，打开页面或者刷新页面时，视频会自动播放。**

# 事件流的三个阶段

1. 事件的捕获阶段

事件的处理将从DOM层次的根开始，而不是从触发事件的目标元素开始，事件被从目标元素的所有祖先元素依次往下传递

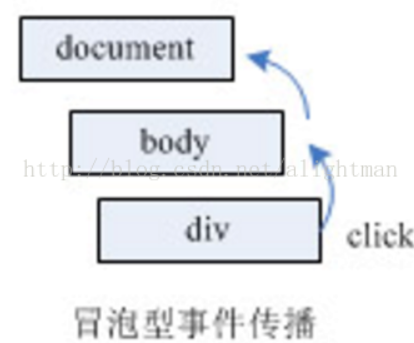
1. 事件的目标阶段

触发自己的事件

1. 事件的冒泡阶段

当我们触发了子元素的某个事件后，父元素对应的事件也会触发

阻止事件冒泡，event.stopPropagation()



# 阻止浏览器的默认行为

e.preventDefault() : 非 IE 使用

e.returnValue = false ：IE 使用

# instanceof和typeof区别

typeof 和 instanceof 常用来判断一个变量是否为空， 或者是什么类型的

**typeof**

typeof判断所有变量的类型，返回值有number、string、boolean、function、object、undefined，返回值是一个字符串object， 用来说明变量的数据类型

对于 Array, Null 等特殊对象使用 typeof 一律返回 object

**instanceof**

instanceof用来判断对象是否属于某个对象的实例，代码形式（obj1 instanceof obj2）（判断obj1是否为obj2的实例），obj2必须为对象，否则会报错

返回的是布尔值

instanceof可以对不同的实例对象进行判断，判断方法是根据对象的原型链依次向下查询，如果obj2的原型属性存在于obj1的原型链上，（obj1 instanceof obj2）值为true

# hook

## useState

1. State Hook让函数组件也可以有state状态, 并进行状态数据的读写操作

2. 语法: const [xxx, setXxx] = React.useState(initValue)

3. useState()说明:

参数: 第一次初始化指定的值在内部作缓存

返回值: 包含2个元素的数组, 第1个为内部当前状态值, 第2个为更新状态值的函数

4. setXxx() 2种写法:

setXxx(newValue): 参数为非函数值, 直接指定新的状态值, 内部用其覆盖原来的状态值

setXxx(value => newValue): 参数为函数, 接收原本的状态值, 返回新的状态值, 内部用其覆盖原来的状态值

## useEffect

[ ]：相当于 Didmount ，初始化阶段执行，只执行一次

[ a , b ]：相当于 watch ，当数组中的值改变时，才会执行

1. Effect Hook 可以让你在函数组件中执行副作用操作(用于模拟类组件中的生命周期钩子)

2. React中的副作用操作:

发ajax请求数据获取

设置订阅 / 启动定时器

手动更改真实DOM

3. 语法和说明:

useEffect(() => {

// 在此可以执行任何带副作用操作

return () => { // 在组件卸载前执行

// 在此做一些收尾工作, 比如清除定时器/取消订阅等

}

}, [stateValue]) // 如果指定的是[], 回调函数只会在第一次render()后执行

4. 可以把 useEffect Hook 看做如下三个函数的组合

componentDidMount()

componentDidUpdate()

componentWillUnmount()

## useReducer

const [state,dispatch] = useReducer(reducer, initialState)，它接收一个形如(state, action) => newState的 reducer，并返回当前 state 以及发起 action 的 dispath 方法

与Redux用法相同

useReducer 和 useState 里才是数组const [ state, dispatch ] = useReducer/useState

以{(开头的需要在(前加;：{;(

## useContext

可以在跨越组件层级直接传递变量，实现数据共享

这里要注意的是，很多同学觉得可以使用useContext结合useReducer来替代redux，其实两者的作用是不同的。

useContext：解决组件间传值的问题。

redux：统一管理应用状态。

所以，我们可以使用useContext结合useReducer来模拟一个小型redux场景，而无法替代redux

Context的作用就是对它所包含的组件树提供全局共享数据的一种技术

# CSS实现水平垂直居中

1. 利用margin:auto

position: absolute; margin: auto; top: 0; left: 0; right: 0; bottom: 0;

1. 利用position: absolute

position: absolute; left: 50%; top: 50%; margin-left: -25px; margin-top: -25px;

1. 弹性盒子

display: flex; justify-content: center; align-items: center;

1. 利用水平对齐和行高

text-align: center; line-height: 200px;

# 聊天功能发送图片或表情

1. 表情图片发送出去的实际上是文字，通过文字和图片匹配，从而显示表情消息
2. 中间聊天部分，用数组保存消息数据即可，同时，为了显示表情图片，数组对象中保存有消息数据，拿到数据，替换成图片，然后使用v-html显示即可

# 面试话术：

**为什么选择前端？**

前端市场越来越广

前端发展前景好

喜欢前端的实时交互的感觉

**部门：**

项目研发中心：产品设计部，交互设计部UE，视觉设计部UI，智能硬件部，硬件维修部

**排期：**研发-联调-测试

**职业规划参考话术：**

近几年肯定是主要以技术为主 扩大自己的技术面和技术深度 然后逐渐开始向前端or前端相关的底层基础设施上去做 提高自己的管理能力和团队协作能力 再之后可能就要看那时候的状态是否向技术leader或者其他方向转移了.期望薪资不高的可以不提leader方向，

想学习更多东西，想往全栈方向发展

**欠缺**:

就我应聘的工作是可以胜任的,但就整个前端来说,认识的深度和广度还不够

既然选择来本公司面试,但如果公司觉得我不匹配,我会回去整理一下欠缺的地方,查漏补缺

**遇到不会的问题**:IT经常会遇到大大小小的问题,我相信我可以解决；技术小组，博客，最后实在解决不出来再问同事问组长