1907A面试宝典

1907A面试宝典

一、个人简历模版
二、自我介绍话术
三、面试官问题
四、项目介绍
五、面试官问题
六、vue面试题
6.1 . MVC、MVP与MVVM模式
6.2 MVVM模式的优点以及与MVC模式的区别
6.3 常见的实现MVVM数据绑定的做法有哪些?
6.4 Object.defineProperty()方法的作用是什么?
6.5 vue.js的两个核心是什么?
6.6 请详细说下你对vue生命周期的理解?
6.6.1 什么是vue生命周期?
6.6.2 vue生命周期钩子函数都有哪些？分别是什么意思?
6.6.3 vue生命周期的作用是什么?
6.6.4 第一次页面加载会触发哪几个钩子?
6.6.5 简述每个周期具体适合哪些场景?
6.6.6 created和mounted的区别?
6.6.7 vue获取数据在哪个周期函数?
6.7 说一下你对vue路由的理解吧
6.7.1 什么是vue路由?
6.7.2 vue路由的优点以及缺点是什么?
6.7.3 请简单说一下vue路由的原理?
6.7.4 怎么定义 vue-router 的动态路由?如何获取动态路由传过来的值?
6.7.5 请描述vue-router路由守卫的作用?
6.7.6 路由守卫使用的方式有几种?
6.7.7 路由守卫的钩子函数都有哪些？分别是什么意思?
6.7.8 路由守卫钩子函数里面的三个参数分别是什么?
6.7.9 路由守卫的解析流程?
6.7.10 vue-router路由传参的方式一共有几种？他们是如何就收传递过来的参数?
6.7.11 query传参和params方式的区别是什么?
6.7.12 什么是路由懒加载？以及路由懒加载是如何实现的?
6.8 说一下你对vuex的理解?
6.8.1 什么是vuex?
6.8.2 vuex的优点和缺点是什么?
6.8.3 一般什么情况下使用 vuex?
6.8.4 vuex的原理是什么?
6.8.5 请你说一下vuex的用法?
6.8.6 你在项目中哪些地方用到了vuex?
6.8.7 vuex的运行机制什么?
6.8.8 vuex都有哪些属性?
6.8.9 你是如何获取state的值，如何调用gettes里面定义的方法？如何调用mutations的方法？如何调用actions的方法?
6.8.10 vuex里面module属性的作用是什么?
6.8.11 不使用vuex会带来什么问题?
6.8.12 Vue.js中ajax请求代码应该写在组件的methods中还是vuex的actions中?
6.8.13 Vuex中如何异步修改状态?
6.8.14 Vuex中actions和mutations的区别?
6.8.15 页面刷新后vuex的state数据丢失怎么解决?
6.8.16 vuex怎么知道state是通过mutation修改还是外部直接修改的?
6.9 请你说一下你对vue组件通信的理解?

6.10 父组件如何与子组件怎么通信?
6.11 子组件如何与父组件进行通信?
6.12 非父子组件之间如何进行通信?
6.15 除了组件之间的这种通信方式以外，还是什么方式可以让组件的数据进行共享?
6.16 props接收父组件发送过来的数据有几种形式?
6.17 非父子组件之间通信的原理是什么?
6.18 请描述vue的优点是什么？缺点是什么?
6.19 请你描述一下vue中让元素隐藏的方式有几种？区别是什么?
6.20 你在vue中怎么样让你写的css样式只在你当前写的组件中显示?
6.21 请你说一下keep-alive？你在哪些地方用过它?
6.22 请你说一下在vue中如何获取dom元素?
6.23 请你说一下vue中常用的指令有哪些?
6.24 请你说一下为什么使用key?
6.25 说一下你对axios的理解?
6.26 说一下axios如何解决跨域?
6.27 请描述v-model是什么？以及他的作用？以及他的原理是什么?
6.28 请描述computed，watch和methods的区别？以及他们的使用场景?
6.29 请你描述一下你对$nextTick的理解?
6.30 说一下你对渐进式框架的理解?
6.31 说一下你对vue数据双向绑定的理解?
6.32 说一下vue单页面和多页面的区别?
6.33 请你说一下什么是vue的过滤器？你在项目中哪些地方使用过过滤器?
6.34 请你说一下你对vue指令的理解？以及他的使用场景？ 并描述你在项目中安歇地方使用过vue自定义指令?
6.35 请你说一下vue的核心是什么?
6.36 请你说一下vue和jquery的区别?
6.37 请你说一下你在vue打包项目的时候有没有出现什么问题？你是如何解决的?
6.38 请你描述一下react和vue的区别是什么?
6.39 请你说一下如何优化vue首屏加载的速度?
6.40 请你说一下你对slot的理解?
6.41 请你描述一下封装vue组件的过程?
6.42 如果说你在开发项目的时候，后台的接口还没有写完，请问这个时候你一般会怎么做
6.43 vue如何封装通用组件?
6.44 vue常用的ui组件库有哪些?
6.45 vue常用的修饰符一共有哪些?
6.46 请你说一下ajax和axios的区别是什么?
6.47 vue组件如何适配移动端?
6.48 说一下在vue中如何使用背景图片?
6.49 如何解决禁用表单后移动端样式不统一问题?
6.50 请你说一下数据双向绑定的原理是什么?
6.51 什么是请求拦截，什么响应拦截？ 拦截点分别是那几个?

七、ECMAScript6面试题

7.1 请描述let与const以及var的区别？以及什么是暂时性死区？什么是变量提升?
7.2 请说一下你对es6的模版字符串的理解？有什么特点?
7.3 请说一下箭头函数与普通函数的区别?
7.4 请说一下什么是函数的默认参数?
7.5 请说一下Object.assign()的有什么作用
7.6 请说一下你对promise的理解？并说一下promise你是如何使用的?
7.7 请说一下你对es6模块化的理解
7.8 请说一下 es5与es6的区别?
7.9 请说一下使用箭头函数应该要注意什么?
7.10 请说一下es6有哪些新增的特性?
7.11 请说一下你对es6 class类的理解?
7.12 请说一下**Promise 中reject 和 catch 处理上有什么区别**?
7.13 请说一下什么是深拷贝，什么是浅拷贝？以及如何实现深拷贝与浅拷贝？用es6如何实现深拷贝?
7.14 举一些ES6对String字符串类型做的常用升级优化?
7.15 举一些ES6对Array数组类型做的常用升级优化?
7.16 Map是什么，有什么作用?

7.17 Set是什么，有什么作用?
7.18 Proxy是什么，有什么作用?
7.19 Class、extends是什么，有什么作用?
7.20 常前端代码开发中，有哪些值得用ES6去改进的编程优化或者规范?
7.21 什么是 Babel?
八、微信小程序面试题
8.1 简单描述下微信小程序的相关文件类型
8.2 简述微信小程序原理
8.3 小程序的双向绑定和vue哪里不一样
8.4 小程序的wxss和css有哪些不一样的地方?
8.5 小程序页面间有哪些传递数据的方法
8.6 小程序的生命周期函数
8.7 怎么封装微信小程序的数据请求?
8.8 请说一下小程序的授权登录?
8.9 哪些方法可以用来提高微信小程序的应用速度?
8.10 怎么解决小程序的异步请求问题?
8.11 小程序关联微信公众号如何确定用户的唯一性?
8.12 小程序如何实现下拉刷新?
8.13 bindtap和catchtap的区别是什么?
8.14 简述下 `wx.navigateTo()`, `wx.redirectTo()`, `wx.switchTab()`, `wx.navigateBack()`, `wx.reLaunch()` 的区别
8.15 说一下小程序组件中如何进行通信?
8.16 说一下小程序中的behaviors的作用?
8.17 说一下小程序的observe的理解?
8.18 说一下微信小程序支付功能如何实现?
8.19 说一下什么是小程序云开发?
8.20 说一下小程序中wxs?
8.21 说一下在小程序中如何使用npm?以及如何使用echarts?
九、JavaScript面试题
9.1闭包
9.2 谈谈你对原型链的理解?
9.3说一下JS继承（含ES6的）--或者人家这样问有两个类A和B,B怎么继承A?
9.3 说一下JS原生事件如何绑定
9.4说一下JS原生常用dom操作方法?
9.5说一下ES6新增特性?
9.6 JS设计模式有哪些(单例模式观察者模式等)
9.7 说一下你对JS面试对象的理解
9.8 说一下JS数组常用方法（至少6个）
9.9 说一下JS数组内置遍历方法有哪些和区别
9.10 说一下JS作用域
9.11 说一下从输入URL到页面加载完中间发生了什么?
9.12 说一下JS事件代理（也称事件委托）是什么，及实现原理?
9.13 说一下js数据类型有哪些?
9.14 说一下 call,apply,bind区别
9.15 JavaScript的作用域链理解吗
9.16 ES6模块与CommonJS模块有什么区别?
9.17 null与undefined的区别是什么?
9.18 那么箭头函数的this指向哪里?
9.19 async/await是什么?
9.20 async/await相比于Promise的优势?
9.21 JavaScript的基本类型和复杂类型是储存在哪里的?
9.22 简述同步与异步的区别
9.23 JavaScript垃圾回收原理?
9.24 请描述值类型(基本数据类型)和引用类型的区别?
9.25 深拷贝和浅拷贝的区别？如何实现
实现方式
9.26 浏览器是如何渲染页面的?
9.27 什么是JavaScript原型，原型链？有什么特点?

9.28 json和jsonp的区别?
9.29 如何阻止冒泡?
9.30 如何阻止默认事件?
9.31 JavaScript事件流模型都有什么?
9.32 用 **js** 实现随机选取 **10–100** 之间的 **10** 个数字，存入一个数组，并排序。
9.33 有这样一个 **URL**:**http://item.taobao.com/item.htm?a=1&b=2&c=&d=xxx&e**， 请写一段 **JS** 程序提取 **URL** 中的各个 **GET** 参数(参数名和参数个数不确定)，将 其按 **key-value** 形式返回到一个 **json** 结构中，如**{a:'1', b:'2', c:'', d:' xxx', e:undefined}**。
9.34 请你谈谈cookie的弊端?
9.35 哪些操作会造成内存泄漏?
9.36 你如何优化自己的代码?
9.37 JavaScript 中的强制转型是指什么?
9.38 解释 JavaScript 中的相等性。
9.39 你能解释一下 ES5 和 ES6 之间的区别吗?
箭头函数和字符串插值:
常量
块作用域变量。
默认参数值
类定义和继承
for...of 操作符
用于对象合并的 Spread 操作
promise
模块导出和导入
9.40 解释 JavaScript 中“undefined”和“not defined”之间的区别
9.41 匿名和命名函数有什么区别?
9.42 什么是 JavaScript 中的提升操作?
十、WEB页面性能优化面试题
10.1 为什么要进行页面性能优化
10.2 减少HTTP请求
10.3 **合并CSS、合并javascript、合并图片。**
10.4 **合并CSS、合并javascript、合并图片**
10.5 **合理设置缓存**
10.6 **将更新频率比较低的CSS、javascript、logo、图标等静态资源文件缓存在浏览器中**
10.7 CSS放在页面最上部，javascript放在页面最下面
10.8 减少作用域链查找
10.9 css和注释
10.10 CSS Sprites（精灵图||雪碧图）
10.11减少dome请求
10.11**使用外部的JavaScript和CSS**
10.12 节流和防抖
10.12.1**防抖**（debounce）
10.12.2**节流**（throttle）
十一、WebPack面试题
11.01 webpack与grunt，gulp的不同?
11.02 webpack，rollup，parcel优劣?
11.03 有哪些常见的Loader?
11.04 有哪些常见的Plugin?
11.05 分别介绍bundle，chunk，module是什么
11.06 Loader和Plugin的不同?
11.07 webpack的构建流程是什么?
11.08 是否写过Loader和Plugin？描述一下编写loader或plugin的思路?
11.09 webpack的热更新是如何做到的？说明其原理?
11.10 如何用webpack来优化前端性能?
11.11 如何提高webpack的打包速度?
11.12 如何提高webpack的构建速度?
11.13 怎么配置单页应用？怎么配置多页应用?
十二、移动端、安卓、IOS兼容性面试题
12.1 IOS移动端click事件300ms的延迟相应?

12.2 一些情况下对非可点击元素（label，span）监听click事件，iso下不会触发?
12.3 说一下手机端如何做适配的?
12.4 键盘遮挡输入框?
12.5 安卓部分版本input里的placeholder位置偏上?
12.6 1px边框问题?
12.7 安卓浏览器看背景图片，有些设备会模糊
12.8 防止手机中页面放大和缩小
12.9 上下拉动滚动条时卡顿、慢
12.10 长时间按住页面出现闪退
12.11 响应式图片
十三、GIT面试题
13.1 列举工作中常用的几个git命令?
13.2 提交时发生冲突,你能解释冲突是如何产生的吗?你是如何解决的?
13.3 如果本次提交误操作,如何撤销?
13.4 如何查看分支提交的历史记录？查看某个文件的历史记录呢?
13.5 能不能说一下git fetch和git pull命令之间的区别?
13.6 git跟其他版本控制器有啥区别?
13.7 我们在本地工程常会修改一些配置文件，这些文件不需要被提交，而我们又不想每次执行git status时都让这些文件显示出来，我们该如何操作?
13.8 如何把本地仓库的内容推向一个空的远程仓库?
13.9 如果分支是否已合并为master，你可以通过什么手段知道?
13.10 描述一下你所使用的分支策略?
13.11 如何在Git中创建存储库?
13.12 请描述什么是工作区、暂存区和本地仓库?
13.13 请写出查看分支、创建分支、删除分支、切换分支、合并分支的命令以及写出解决冲突的思路?
13.14 请写出将工作区文件推送到远程仓库的思路？（两种情况都写出来）
13.15 请写出团队内部协作开发的流程?
13.16 请写出远程跨团队协作开发的流程?
13.17 请写出配置ssh的思路?
13.18 请描述什么是GitLab,或者说出你对GitLab的理解?
13.19 请写出你所参与的多人协同开发时候，项目都有哪些分支，分支名是什么，每个分支代表什么，以及分支是由谁合并?
十四、HTML与CSS面试题
14.01 HTML、XHTML、XML有什么区别
14.02 知道img的srcset的作用是什么?
14.03 link和@import的区别?
14.04 如何理解层叠上下文是什么?
14.05 谈谈对BFC的理解 是什么?
14.06 HTML5和css3的新标签
14.07 html5有哪些新特性，移除了哪些元素？如何处理HTML5新标签的浏览器谦容问题？如何区分HTML和HTML5?
14.08 为什么有时候人们用translate来改变位置 而不是定位?
14.09 浏览器的内核分别是什么？经常遇到的浏览器的谦容性有哪些？原因，解决方法是什么，常用hack的技巧?
14.10 html常见谦容性问题?
14.11 描述一个“reset”的css文件并如何使用它。知道normalize.css吗？你了解他们的不同之处?
14.12 什么是外边距重叠？重叠的结果是什么?
14.15 box-sizing常用的属性有哪些?分别有啥?
14.16 CSS中transition和animate有何区别?animate如何停留在最后一帧!
14.17 什么是css hack？ ie6,7,8的hack分别是什么?
14.18 谈谈以前端角度出发做好SEO需要考虑什么?
14.19 一个页面上有大量的图片(大型电商网站)，加载很慢，你有哪些方法优化这些图片的加载，给用户更好的体验。
14.20 你能描述一下渐进增强和优雅降级之间的不同吗?
14.21 为什么利用多个域名来储存网站资源会更有效?
14.22 css中可以通过哪些属性定义，使得一个dom元素不显示在浏览器可视范围内?
14.23 超链接访问过后hover样式就不出现的问题是什么？如何解决?

14.24 css中可以让文字在垂直和水平方向上重叠的两个属性是什么?
14.25 display:none 与visibility:hidden的区别是什么?
14.26 IE的双边BUG：块级元素float后设置横向margin，ie6显示margin比设置的较大。
14.27 absolute的containing block 计算方式跟正常流有什么不同?
14.28 知道css有个content属性吗？有什么作用？有什么应用?
14.29 css新增伪类有哪些?
14.30 sass，less是什么？大家为什么要使用他们?
14.31 css动画animation
十五、HTTP面试题
15.01 http/https协议
15.02 常见状态码
15.03 get/post
15.04 websocket
15.05 TCP三次握手
15.06 TCP四次挥手
15.07 Node 的 Event Loop: 6个阶段
15.08 跨域
15.09 安全
15.10 缓存机制
15.11 什么是HTTPS
15.12 为什么需要HTTPS

一、个人简历模版

https://a.lmongo.com/offer/resume.html

二、自我介绍话术

您好，我叫xxx，今年xxx岁，老家xxx。做前端开发已经xxx年了，期间换过xxx家公司，第一家是一个xxxx类型公司，叫xxxxx；上一家家叫xxxx，上一家公司是自己公司或外包公司，做自己的产品或者做别人的产品，在上一家公司呆了xxx(多长时间)，上一家公司一共xxx人，开发团队xxx人，一共xxx个小组，我们小组xxx人，前端xxx人，后端xxx人，大概做了xxx个项目，上一家公司前端主要是用xxx技术，后端是用xxx技术，从上一家公司离职主要是因为xxx

三、面试官问题

1. **你从上一家公司离职的原因是什么?**
   - 家里有人生病,回家需要照顾　为什么不请假呢?　因为请的时间比较久,所以选择了离职
   - 因为有家人或者女朋友/男朋友在这边,所以选择来这边发展
   - 公司资金链出现问题,你坚持了3个月还没有发工资,由于经济原因实在撑不住了,所以选择离职
2. **你上一家公司在那里?**

   找自己简历上面写的公司的地址
3. **那你现在住在那里?**

   北京-昌平区-天通苑　北京-昌平区-沙河地铁站附近
4. **你到你上一家公司需要多久,坐什么车,需要走多久,到那个地方下车?**

   自己去查
5. **你上一家是做什么的?**

   自己在网上查
6. **上一家公司多少人?或者你们公司技术团队多少人？ 前端几个人，后端几个人?**

公司具体多少人我记不清了,刚好钉钉群或者微信群离职的时候给退了,所以现在具体多少人我也不太清楚,但是我们那个开发小组大概个5人,1个前端,2个后端,1个产品,1个测试

7. **上一家公司给你的薪资是多少?**

   根据自己的需求

8. **你是那个学校毕业的?**

   你简历上写的学历

9. **你是什么学历?**

   本科

10. **你的学历是公办还是民办?**

    我上的这个学校是一个民办的学校

11. **你学的是那个专业?**

    根据自己办的学历的内容去说

12. **这个专业你都学过哪些课程或者你上大学都学过计算机哪些课程?你学的这些知识你还记得多少?你们的校训是什么?**

    - 我们的课程分为专业课和非专业课

      非专业课: 马克思主义 大学英语 高等数学

      专业课: 计算机基础 软件工程 数据库原理 c语音 java asp c#

    - 时间太久很多东西都忘了
    - 校训在网上查一下

13. **你为什么选择前端或者你是怎么接触到前端的?**

    - 我们公司当时也有开发部门,当时我和开发部门的领导关系比较好,他说让我自学一些技术,而且技术部门的工资比较高,所以我当时就转技术了
    - 我家里的亲人或者你哥你姐是做技术的,而且工资比较高,后来过年在一块的时候,他们就让我去跟着他们学技术,然后平时他们会让我帮忙做一些东西,后台他们觉得我感觉做的东西还行,就帮忙给我推荐了一家公司
    - 我大学就是计算机专业的,所以大学毕业后就做的是开发
    - 从培训机构

14. **你有没有在培训机构培训过?**

    - 没有
    - 有,我大概两年前在培训机构培训过,现在已经工作三年了

15. **你找工作找了多久了?**

    - 错误回答: 1个月 或者2几个月
    - 正确回答: 刚开始找,咱们公司是我第一面试的, 我已经面了两三家,目前已经收到了1家offer

16. **你感觉你的优点是什么? 缺点是什么?**

    - 努力,学习能力强,有毅力,做事比较大气
    - 我还真的没有去思考过这个问题

17. **谈谈你对前端的看法?**

    自己在网上搜相关的资料,自己去总结一套话术

18. **你上一家公司用的什么技术?**

    你擅长什么技术就说什么技术

19. **你感觉那一个项目是你最满意的项目?你为什么觉得她是你最满意的项目?**

    - 我觉得我里面写的项目都挺满意的
    - 可以选一个刚开始用vue做的项目

    - 可以选一个最近做的一个项目
20. **那你对我们公司有没有了解?**
    - 没有,您能不能给我介绍一下
    - 有,我在来咱们公司之前,我在网上搜索过咱们公司的一些内容
21. **你上一家公司什么类型公司? 是外包呢还是自己公司的产品?**

    自己简历上写的公司自己去查
22. **你的项目有没有上线地址?**

    结合实际情况去说, 有
23. **你感觉你上家公司怎么样?**

    公司团队氛围好,领导对你特别好,不管是在技术或者在其他生活方面都可以给你带来很多的帮助
24. **你在你上家公司的收获是什么?**

    收获了技术和同事,领导

四、项目介绍

- **项目背景**

  　　您好,我给您说一下最近这个公司做的项目。我去这家公司的时候公司项目刚起步(或者项目已经开发了一部分了),我去了之后主要是负责项目的切页面以及功能的开发. 我当时的这个项目小组,前端就我一个人或者两个人,后端3个人,产品和测试各1人. 我当时做的这个项目的名字叫x x x,这是一个什么样的项目(项目概述)
- **项目所使用的技术**

  在这个项目开发的时候,主要用到了(项目所使用的技术栈)
- **负责的内容**

  我在写这个项目的时候主要负责哪些模块(简历上负责模块)
- **项目功能**

  这个项目都有哪些功能(可以是简历上写的功能,也可以是简历上没写的功能,但是说到的功能必须自己会,并且能说出这个功能的思路)
- **项目难点**

  我在写这个项目的时候主要遇到了哪些难点(自己去总结)
- **解决方案**

  这些难点的解决方式是(自己去总结)
- **项目总结**

  做完这个项目之后,我的收获是(自己总结)

五、面试官问题

1. **你们公司前端几个人?后端几个人?**

   自己去说,一般是1比2或者1比3的一个比例
2. **后端用的语言?**

   Java php python node
3. **你这个项目是前后端分离吗?**

   是
4. **什么是前后端分离?说说你对前后端分离的理解?**

- 什么是前后端分离?
- 什么是前后端不分离?
- 传统开发(前后端不分离)的缺点
- 前后端分离的优点

5. **你们的接口规范是什么?你知道RESTFUL吗?**

   自己去查

6. **那你说一下你和后台是如何交互的?或者说你和后台交互的方式有哪些**

   - 原生的ajax
   - jquery的ajax
   - axios
   - fetch
   - websocket

7. **ajax的通信原理是什么或者axios的原理是什么?**

   自己去查

8. **axios或者ajax或者vue中跨域是什么解决的,都有哪些方案?**

   - **为什么要有跨域?**

     跨域限制是服务端的一个行为，当开启对某些域名的访问限制后，只有同域或指定域下的页面可以调用，这样相对来说更安全，图片也可以防盗链 跨域限制一般只在浏览器端存在，对于服务端或 OS Android 等客户端是不存在的。

   - **什么时候跨域? 介绍同源策略**
   - **跨域的方式有哪几种?**
     - jsonp
     - vue中通过config.js在里面的proxy(代理)里面进行配置
     - iframe
     - node写一个代理
     - Cros(后端允许跨域)
   - **在项目开发中一般都用那种方式?**

     自己总结

9. **请你说一下jsonp的原理?**

   自己查

10. **请你说一下iframe是如何跨域的?**

    自己查

11. **你项目的难点是什么?**

    自己去总结

12. **你是怎么解决这些难点的?**

    自己去总结

13. **你这个项目是H5还是app?**

    根据自己简历写的项目去回答

14. **如果是app的话,请问你们的这个app是那种app? 是原生app?还是混合app?还是webapp?**

    根据自己找的项目来回答

15. **什么是原生app?**

    自己去查

16. **什么是混合app?**

自己去查

17. **什么是webapp?**

    自己去查

18. **app是如何打包的?**

    - 将写好的vue项目通过npm run build 命令进行打包,但是打包之前我们需要将路由配置hash,将vu e.config.js里面的publicPath配置为./
    - 打开hbuilderx,创建5+app项目
    - 保留manifest.json文件和unpackaged.json文件
    - 将vue打包后的dist目录的内容放到5+app目录里面
    - 在manifest.json文件里面可以配置打包的一些配置
    - 点击发行->选择云打包->最后生成打包后的链接或者.apk这样的一个文件

19. **你们项目是如何开发的?**

    - 需求 - 原型图 - 流程图 - 设计图 - 接口文档
    - 项目开发前的准备工作
    - 项目使用什么技术开发
    - 先开发的什么,后开发的什么
    - 开发完成之后做什么
    - 在公司里面一天

20. **什么是模块化开发?**

    自己去查

21. **项目里都用到了哪些插件?**

    自己总结

22. **项目都用到了哪些技术点?**

    自己去总结

23. **请你说一下你项目当中具体的某几个功能是怎么实现的?**

    自己去总结

24. **你是如何完成第三方登录?或者说一下微信登录和qq登录的思路?**

    自己去总结

25. **项目如何上线?**

    - 不是你负责的,公司的项目上线是运维负责的,但是你知道项目大概怎么上线的
        - 购买域名以及购买服务器(域名需要认证,将近1个月的时间)
        - 购买的服务器上面搭建nginx服务器
        - 通过ftp工具将我们的项目上传到nginx服务器就可以(配置nginx跨域/代理)

26. **项目开发了多久?**

    按项目的开发周期

27. **项目开发了几期?每一期都有哪些功能或者优化了哪些部分?**

    自己去总结

28. **你对后端了解多少?**

    使用过,但很久了,很多东西已经忘了,但是可以快速上手

29. **你们项目开发的时候有没有什么代码规范?如果有的话你们的规范是什么?**

30. **你们这个项目都有了哪些框架?**

    自己总结

31. **你们为什么用vue开发?为什么不用react?**
    - 学习成本
    - 踩坑成本
32. **在你写的项目中,你有没有封装过哪些模块或者组件?你是如何进行封装的?**

    自己总结
33. **你是如何进行组件话开发的?或者说一下你对最简化开发的理解?**

    自己总结
34. **说一下你在项目中是如何封装组件的?有没有什么需要注意的?**

    自己总结
35. **我们开发组件应该尊重什么原则?**

    你对组件开发的理解以及官方的描述:
    - 开放性
    - 封闭性
    - 粒度
36. **你觉你这个项目的亮点是什么?**

    代码层面
37. **你最近写的项目写完之后你的收获是什么?**

    自己总结
38. **你一共做过多少个项目?这些项目都有哪些类型的**

    自己总结
39. **最近做的这个项目?**

    自己根据自己简历写的项目去说
40. **你写的这个项目是h5还是app,还是pc端,还是安卓,还是ios?**

    自己总结
41. **为什么公司要做这个项目?**

    我刚去的时候项目都已经开始了,所以我也不太了解
42. **项目的开发周期多久?**

    自己总结
43. **项目的开始时间以及项目的结束时间?**

    自己根据简历上写的内容去总结
44. **项目一共分为多少期?**

    自己总结
45. **这几期每一期做到什么程度?每一期之间的区别是什么?或者描述一些每一期都做了耐饿功能和优化?**

    自己总结
46. **项目一共有多少个功能?那个功能你觉得最难?为什么?你是怎么解决的?**

    自己总结
47. **项目的开发流程?**

    开会讨论需求-> 出原型图-》流程图-〉设计图-》编码-〉测试-》上线
48. **项目是几个人开发的?**

    自己总结

49. **项目各自的分工是什么?你们是怎么划分任务的?**
    - 我都负责了哪些模块的开发
    - 领导划分任务
50. **公司的版本控制工具用的什么?**

    git svn -》 gitlab
51. **你们是如何进行多人协同开发的?**

    自己总结
52. **项目开发的前期你都在干什么?**

    自己总结
53. **如果让你重新再做这个项目,你觉的哪些 地方还可以再优化?**

    代码层面去分析
54. **你在做这个项目的时候,你是如何去排查bug?**

    自己总结
55. **你们这个项目前后端是怎么交互的?**

    ajax axios fetch ifrmae
56. **项目是如何上线的?项目上线流程是什么?**

    不是我负责的,但是我知道大概的流程

六、vue面试题

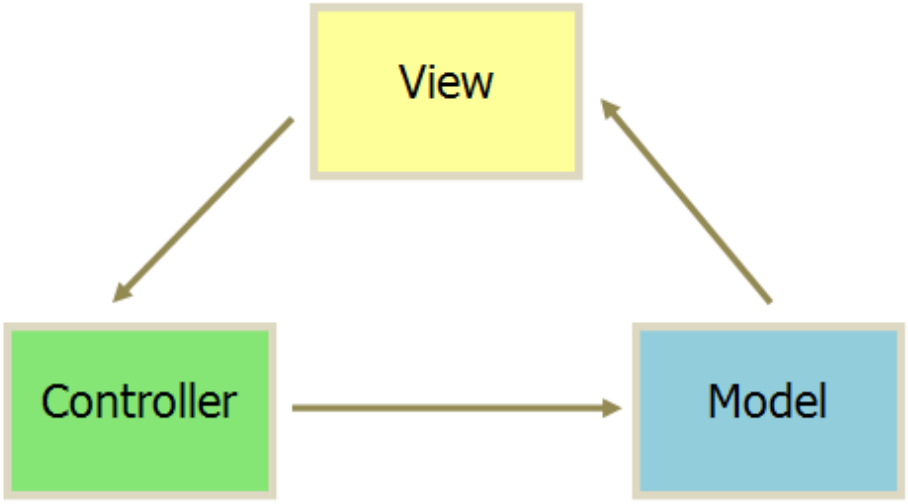
6.1 . MVC、MVP与MVVM模式

MVC:

MVC是应用最广泛的软件架构之一，一般`MVC`分为:

`Model（ 模型 ）`、`Controller（ 控制器 ）`、`View（ 视图 ）`。

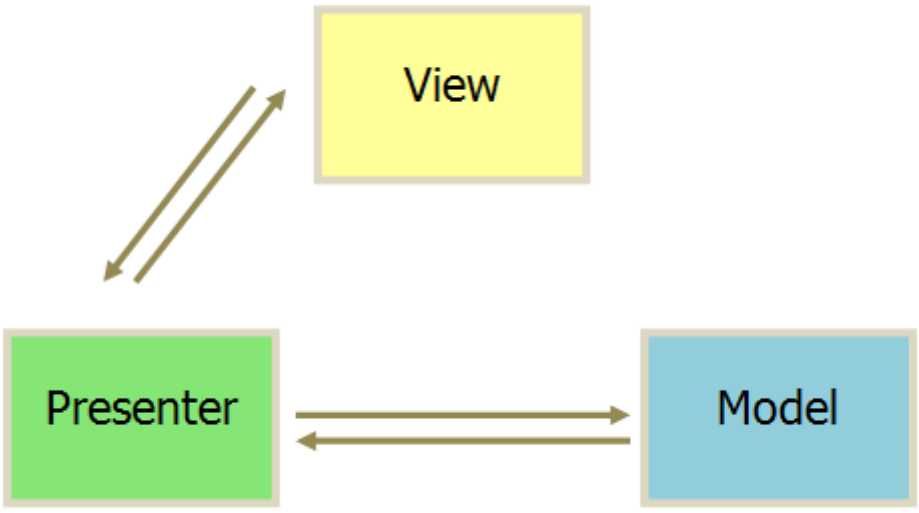
这主要是基于分层的目的，让彼此的职责分开。`View` 一般通过 `Controller` 来和 `Model` 进行联系。`Controller`是 `Model` 和 `View` 的协调者，`View`和`Model`不直接联系。基本联系都是单向的。



> 1、View 传送指令到 Controller
> 2、Controller 完成业务逻辑后，要求 Model 改变状态
> 3、Model 将新的数据发送到 View，用户得到反馈

MVP：

MVP 模式将 Controller 改名为 `Presenter`，同时改变了通信方向。

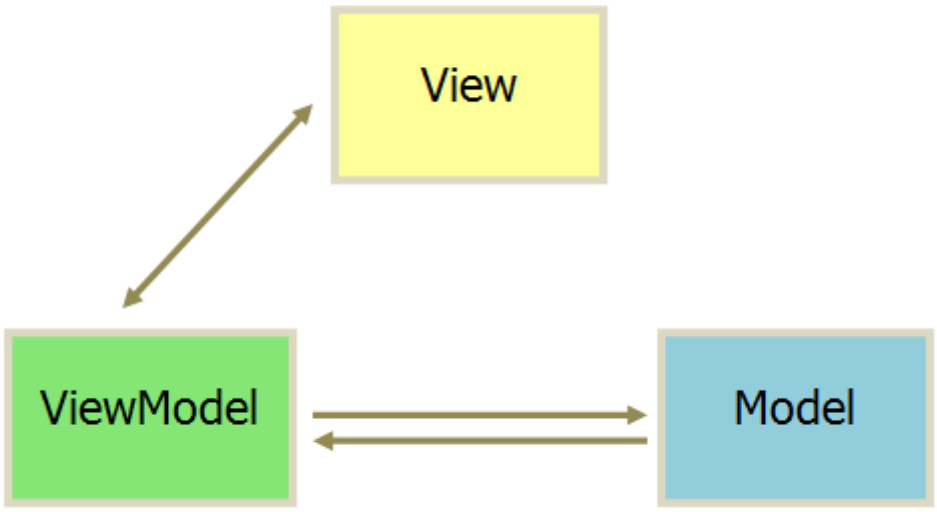


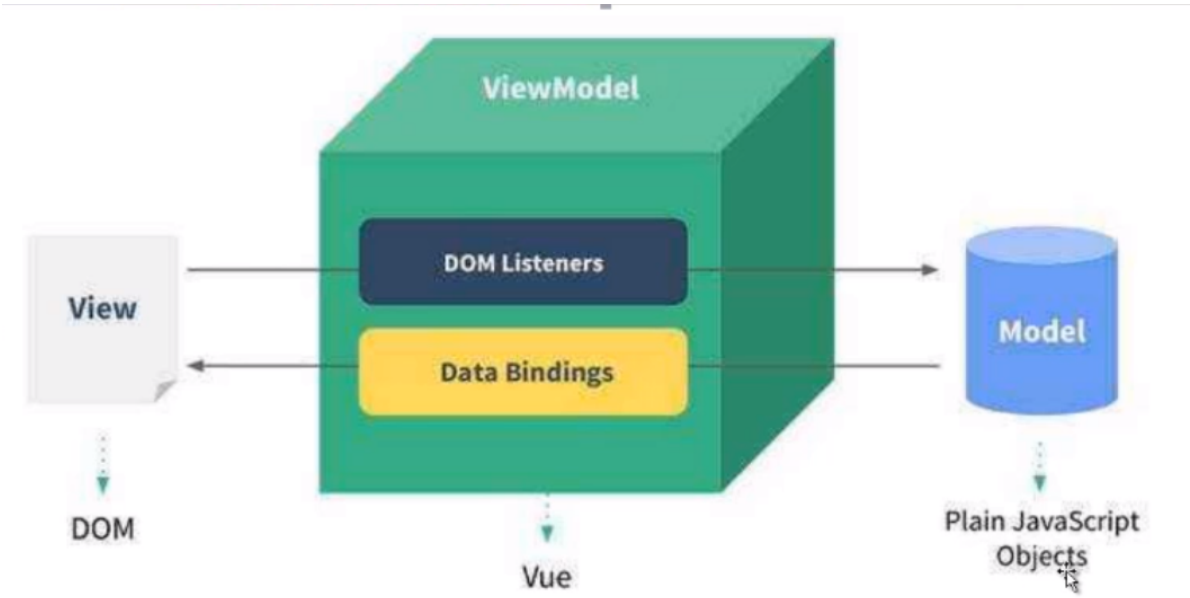
> 1、各部分之间的通信，都是双向的。
> 2、View 与 Model 不发生联系，都通过 Presenter 传递。
> 3、View 非常薄，不部署任何业务逻辑，称为"被动视图"（Passive View），即没有任何主动性，而 Presenter非常厚，所有逻辑都部署在那里。

MVVM

`MVVM` 是把 `MVC` 的 `Controller` 和 `MVP` 的 `Presenter` 改成了 `ViewModel`。

`View` 的变化会自动更新到 `ViewModel`，`ViewModel` 的变化也会自动同步到 `View` 上显示。这种自动同步是因为 `ViewModel` 中的属性实现了 `Observer`，当属性变更时都能触发对应的操作。





6.2 MVVM模式的优点以及与MVC模式的区别

MVVM模式的优点：

`1、低耦合：` 视图（View）可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同的"View"上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。

`2、可重用性：` 你可以把一些视图逻辑放在一个ViewModel里面，让很多 view 重用这段视图逻辑。

`3、独立开发：` 开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。

`4、可测试：` 界面素来是比较难于测试的，而现在测试可以针对ViewModel来写。

MVVM 和 MVC 的区别：

`mvc` 和 `mvvm` 其实区别并不大。都是一种设计思想。

主要区别

- mvc 中 Controller演变成 mvvm 中的 viewModel，
- mvvm 通过数据来显示视图层而不是节点操作。
- mvvm主要解决了: mvc中大量的DOM 操作使页面渲染性能降低，加载速度变慢，影响用户体验。

6.3 常见的实现MVVM数据绑定的做法有哪些?

实现数据绑定的做法有大致如下几种：

> 发布者-订阅者模式（backbone.js）
> 脏值检查（angular.js）
> 数据劫持（vue.js）

1、发布者-订阅者模式:

一般通过 `sub`，`pub` 的方式实现数据和视图的绑定监听，
更新数据方式通常做法是 `vm.set('property', value)`。

这种方式现在毕竟太low了，我们更希望通过 `vm.property = value` 这种方式更新数据，同时自动更新视图，于是有了下面两种方式。

2、脏值检查:

---

angular.js 是通过脏值检测的方式比对数据是否有变更，来决定是否更新视图，

最简单的方式就是通过 setInterval() 定时轮询检测数据变动，

angular只有在指定的事件触发时进入脏值检测，大致如下：

```
1、DOM事件，譬如用户输入文本，点击按钮等。( ng-click )

2、XHR响应事件 ( $http )

3、浏览器Location变更事件 ( $location )

4、Timer事件( $timeout , $interval )

5、执行 $digest() 或 $apply()
```

3、数据劫持:

---

vue.js 则是采用 `数据劫持` 结合 `发布者-订阅者` 模式的方式，

通过 `Object.defineProperty()` 来劫持各个属性的 `setter`，`getter`，

在数据变动时发布消息给订阅者，触发相应的监听回调。

6.4 Object.defineProperty()方法的作用是什么?

> `Object.defineProperty() 方法`会直接在一个对象上定义一个新属性，或者修改一个对象的现有属性， 并返回这个对象。

语法:

```
Object.defineProperty(obj, prop, descriptor)
```

参数说明:

```
obj：必需。目标对象
prop：必需。需定义或修改的属性的名字
descriptor：必需。目标属性所拥有的特性
```

返回值:

```
传入函数的对象。即第一个参数obj
```

针对属性，我们可以给这个属性设置一些特性，比如是否只读不可以写；是否可以被for..in或Object.keys()遍历。

> 给对象的属性添加特性描述，目前提供两种形式：数据描述和存取器描述。

6.5 vue.js的两个核心是什么?

1、数据驱动，也叫双向数据绑定。

> Vue.js数据观测原理在技术实现上，利用的是ES5Object.defineProperty和存储器属性: getter和setter（所以只兼容IE9及以上版本），可称为基于依赖收集的观测机制。核心是VM，即ViewModel，保证数据和视图的一致性。

2、组件系统。

.vue组件的核心选项:

> 1、模板（template）：模板声明了数据和最终展现给用户的DOM之间的映射关系。
> 2、初始数据（data）：一个组件的初始数据状态。对于可复用的组件来说，这通常是私有的状态。
> 3、接受的外部参数(props)：组件之间通过参数来进行数据的传递和共享。
> 4、方法（methods）：对数据的改动操作一般都在组件的方法内进行。
> 5、生命周期钩子函数（lifecycle hooks）：一个组件会触发多个生命周期钩子函数，最新2.0版本对于生命周期函数名称改动很大。
> 6、私有资源（assets）：Vue.js当中将用户自定义的指令、过滤器、组件等统称为资源。一个组件可以声明自己的私有资源。私有资源只有该组件和它的子组件可以调用。
> 等等。

6.6 请详细说下你对vue生命周期的理解?

6.6.1 什么是vue生命周期?

Vue实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模板、挂载Dom、渲染→更新→渲染、销毁等一系列过程，我们称这是Vue的生命周期。通俗说就是Vue实例从创建到销毁的过程，就是生命周期。

6.6.2 vue生命周期钩子函数都有哪些？分别是什么意思?

- 组件通过new Vue() 创建出来之后会初始化事件和生命周期，然后就会执行beforeCreate钩子函数，这个时候，数据还没有挂载呢，只是一个空壳，无法访问到数据和真实的dom，一般不做操作
- 挂载数据，绑定事件等等，然后执行created函数，这个时候已经可以使用到数据，也可以更改数据,在这里更改数据不会触发updated函数，在这里可以在渲染前倒数第二次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取
- 接下来开始找实例或者组件对应的模板，编译模板为虚拟dom放入到render函数中准备渲染，然后执行beforeMount钩子函数，在这个函数中虚拟dom已经创建完成，马上就要渲染,在这里也可以更改数据，不会触发updated，在这里可以在渲染前最后一次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取下来开始render，渲染出真实dom，然后执行mounted钩子函数，此时，组件已经出现在页面中，数据、真实dom都已经处理好了,事件都已经挂载好了，可以在这里操作真实dom等事情...
- 当组件或实例的数据更改之后，会立即执行beforeUpdate，然后vue的虚拟dom机制会重新构建虚拟dom与上一次的虚拟dom树利用diff算法进行对比之后重新渲染，一般不做什么事儿
- 当更新完成后，执行updated，数据已经更改完成，dom也重新render完成，可以操作更新后的虚拟dom
- 经过某种途径调用$destroy方法后，立即执行beforeDestroy，一般在这里做一些善后工作，例如清除计时器、清除非指令绑定的事件等等,组件的数据绑定、监听...去掉后只剩下dom空壳，这个时候，执行destroyed，在这里做善后工作也可以

> 如果觉得上面的太长,也可以如下回答:
>
> 总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。
>
> 创建前/后： 在beforeCreated阶段，vue实例的挂载元素el还没有。在created阶段,vue实例的数据对象data有了,el还没有.
>
> 载入前/后：在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。
>
> 更新前/后：当data变化时，会触发beforeUpdate和updated方法。
>
> 销毁前/后：在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

6.6.3 vue生命周期的作用是什么?

生命周期中有多个事件钩子，让我们在控制整个 vue 实例的过程时更容易形成好的逻辑

6.6.4 第一次页面加载会触发哪几个钩子?

第一次加载会触发 beforeCreate、created、beforeMount、mounted

6.6.5 简述每个周期具体适合哪些场景?

生命周期钩子的一些使用方法：

- beforecreate : 可以在这加个loading事件，在加载实例时触发
- created : 初始化完成时的事件写在这里，如在这结束loading事件，异步请求也适宜在这里调用
- mounted : 挂载元素，获取到DOM节点 updated : 如果对数据统一处理，在这里写上相应函数
- beforeDestroy : 可以做一个确认停止事件的确认框 nextTick : 更新数据后立即操作dom

6.6.6 created和mounted的区别?

- created:在模板渲染成html前调用，即通常初始化某些属性值，然后再渲染成视图。
- mounted:在模板渲染成html后调用，通常是初始化页面完成后，再对html的dom节点进行一些需要的操作。

6.6.7 vue获取数据在哪个周期函数?

- 看实际情况，一般在 created（或beforeRouter） 里面就可以，如果涉及到需要页面加载完成之后的话就用 mounted。
- 在created的时候，视图中的html并没有渲染出来，所以此时如果直接去操作html的dom节点，一定找不到相关的元素
- 而在mounted中，由于此时html已经渲染出来了，所以可以直接操作dom节点，（此时 document.getelementById 即可生效了）

6.7 说一下你对vue路由的理解吧

6.7.1 什么是vue路由?

“Vue路由就是指vue-router,其中router是指根据url分配到对应的处理程序,所以说路由就是用来解析URL以及调用对应的控制器并返回从视图对象中提取好的网页代码给web服务器,最终返回给客户端。

6.7.2 vue路由的优点以及缺点是什么?

- 优点：
  - 不需要每次都从服务器获取，渲染页面更快速
- 缺点:
  - 不利于SEO
  - 使用浏览器前进、后退按键时重新发送请求，未合理利用缓存
  - 单页面无法记住之前滚动的位置

6.7.3 请简单说一下vue路由的原理?

Vue的路由实现：hash模式 和 history模式

- hash模式：在浏览器中符号“#”，#以及#后面的字符称之为hash，用window.location.hash读取；
- 特点：
  - hash虽然在URL中，但不被包括在HTTP请求中；用来指导浏览器动作，对服务端安全无用，hash不会重加载页面。
  - hash 模式下，仅 hash 符号之前的内容会被包含在请求中，如 http://www.xxx.com，因此对于后端来说，即使没有做到对路由的全覆盖，也不会返回 404 错误。
- history模式：history采用HTML5的新特性；且提供了两个新方法：pushState（），replaceState（）可以对浏览器历史记录栈进行修改，以及popState事件的监听到状态变更。
- 特点:
  - history 模式下，前端的 URL 必须和实际向后端发起请求的 URL 一致，如 http://www.xxx.com/items/id。后端如果缺少对 /items/id 的路由处理，将返回 404 错误。Vue-Router 官网里如此描述：“不过这种模式要玩好，还需要后台配置支持……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。”

6.7.4 怎么定义 vue-router 的动态路由?如何获取动态路由传过来的值?

- 定义动态路由:
  - 在 router 目录下的 index.js 文件中，对 path 属性加上 /:id。
- 获取动态路由传过来的值:
  - 使用 router 对象的 `params.id` 获取

```
//全局获取动态路由传递过来的值
$route.params.id
//局部或者是在方法内获取
this.$route.params.id
```

6.7.5 请描述vue-router路由守卫的作用?

vue-router 的导航钩子，主要用来作用是拦截导航，让他完成跳转或取消。

6.7.6 路由守卫使用的方式有几种?

- 全局的
- 单个路由独享的
- 组件级的

6.7.7 路由守卫的钩子函数都有哪些？分别是什么意思?

- vue-router全局有三个守卫:
  - router.beforeEach 全局前置守卫 进入路由之前
  - router.beforeResolve 全局解析守卫(2.5.0+) 在beforeRouteEnter调用之后调用
  - router.afterEach 全局后置钩子 进入路由之后
- 组件内的守卫:
  - beforeRouteEnter
  - beforeRouteUpdata(2.2新增)
  - beforeRouteLeave

6.7.8 路由守卫钩子函数里面的三个参数分别是什么?

- to,from,next 这三个参数:
  - to和from是将要进入和将要离开的路由对象,路由对象指的是平时通过this.$route获取到的路由对象。
  - next:Function 这个参数是个函数，且必须调用，否则不能进入路由(页面空白)。
    - next() 进入该路由。
    - next(false): 取消进入路由，url地址重置为from路由地址(也就是将要离开的路由地址)。
    - next 跳转新路由，当前的导航被中断，重新开始一个新的导航。

- 我们可以这样跳转：next('path地址')或者next({path:''})或者next({name:''})
- 且允许设置诸如 replace: true、name: 'home' 之类的选项以及你用在router-link或router.push的对象选项。

6.7.9 路由守卫的解析流程?

- 导航被触发
- 在失活的组件里调用离开守卫
- 调用全局的 beforeEach 守卫
- 在重用的组件里调用 beforeRouteUpdate 守卫
- 在路由配置里调用 beforEnter
- 解析异步路由组件
- 在被激活的组件里调用 beforeRouteEnter
- 调用全局的 beforeResolve 守卫
- 导航被确认
- 调用全局的 afterEach 钩子
- 触发 DOM 更新
- 在创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数

6.7.10 vue-router路由传参的方式一共有几种？他们是如何就收传递过来的参数?

- 三种：
  - 分别是query，params，动态路由传参
- 接收：
  - 通过query方式传递过来的参数一般是通过this.$route.query接收
  - 通过params方式传递过来的参数一般是通过this.$route.params接收
  - 通过动态路由传参方式传递过来的参数一般是通过this.$route.params接收

6.7.11 query传参和params方式的区别是什么?

- query使用path和name传参跳转都可以，而params只能使用name传参跳转。
- 传参跳转页面时，query不需要再路由上配参数就能在新的页面获取到参数，params也可以不用配，但是params不在路由配参数的话，当用户刷新当前页面的时候，参数就会消失。
- 也就是说使用params不在路由配参数跳转，只有第一次进入页面参数有效，刷新页面参数就会消失。

6.7.12 什么是路由懒加载？以及路由懒加载是如何实现的?

- 按需加载
- 当打包构建应用时，Javascript 包会变得非常大，影响页面加载。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。
- 结合 Vue 的`异步组件`和 Webpack 的`代码分割`功能，轻松实现`路由组件的懒加载。`

6.8 说一下你对vuex的理解?

6.8.1 什么是vuex?

- Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式**。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。
- 我的个人理解是vuex其实就是一个管理数据的工具，通过vuex我们可以解决组件之间数据共享的问题，后期也方便我们管理以及维护

6.8.2 vuex的优点和缺点是什么?

- 优点：
  - 解决了非父子组件的消息传递（将数据存放在state中）
  - 减少了AJAX请求次数，有些情景可以直接从内存中的state获取
  - 数据方便管理以及维护
- 缺点：

- 小型项目使用的话，vuex会显得有点繁琐冗余
- 刷新浏览器，vuex中的state会重新变为初始状态，我们如何要解决这个问题就可能需要用本地存储或者vuex的一个插件

6.8.3 一般什么情况下使用 vuex?

- 官方说的是在大型项目中推荐使用vuex，但是我个人的理解是当页面的组件比较多，业务比较复杂时，数据难以维护，这个时候我一般会使用vuex

6.8.4 vuex的原理是什么?

- 每个Vuex应用的本质是store(仓库），包含应用中大部分的状态。
- state, getters,mutations,actions,module

6.8.5 请你说一下vuex的用法?

- 安装vuex
- 在src目录下创建store文件夹，在该文件夹内创建index.js
- 在store文件夹内的index.js文件内引入vuex
- 然后在引入vue
- 调用Vue.use()方法注册vuex
- 对vuex进行实例化
- 进行实例化之后通过export default导出
- 在main.js文件内引入store文件夹内的index.js文件
- 挂载到new Vue实例上面
- 初始化vuex的状态和属性

6.8.6 你在项目中哪些地方用到了vuex?

- 登录模块，购物车模块，订单模块，商品模块。。。。

6.8.7 vuex的运行机制什么?

- 在vue组件里面，通过dispatch来触发actions提交修改数据的操作。
- 然后再通过actions的commit来触发mutations来修改数据。
- mutations接收到commit的请求，就会自动通过Mutate来修改state（数据中心里面的数据状态）里面的数据。

最后由store触发每一个调用它的组件的更新

6.8.8 vuex都有哪些属性?

- State、Getter、Mutation 、Action、Module 五种
  - state => 基本数据
  - getters => 从基本数据派生的数据
  - mutations => 提交更改数据的方法，同步!
  - actions => 像一个装饰器，包裹mutations，使之可以异步。
  - modules => 模块化Vuex

6.8.9 你是如何获取state的值，如何调用gettes里面定义的方法？如何调用mutations的方法？如何调用actions的方法?

- state的值获取的方式有两种：
  - 第一种是组件里面进行获取 this.$store.state.状态
  - 第二种是在vuex内部进行获取
    - 函数参数里面会有一个state参数，通过这个state参数我们可以直接拿到state的值

    - getters的方法在组件内调用的话是通过this.$store.getters来进行获取，而getters的作用一般是用来获取state的值
    - mutations的方法在组件内调用时一般是通过this.$store.commit()来进行调用，而mutations的作用一般是用来改变state里面状态，只不过通过同步的方式去改变
    - actions的方法在组件内调用的话是通过this.$store.dispatch()来进行调用，而actions的作用一般是用来异步的改变状态，actions也支持promise

6.8.10 vuex里面module属性的作用是什么?

- module属性相当于是vuex里面的模块化方法，module属性可以让每一个模块拥有自己的state、mutation、action、getters,使得结构非常清晰，方便管理。
- 比如：购物车模块，订单模块，商品模块...每个模块都有自己的数据，建立多个模块文件来保存各自对应模块的数据，最后，在module属性里面进行合并

6.8.11 不使用vuex会带来什么问题?

- 可维护性会下降，想修改数据要维护三个地方；
- 可读性会下降，因为一个组件里的数据，根本就看不出来是从哪来的；
- 增加耦合，大量的上传派发，会让耦合性大大增加，本来Vue用Component就是为了减少耦合，现在这么用，和组件化的初衷相背

6.8.12 Vue.js中ajax请求代码应该写在组件的methods中还是vuex的actions中?

- 如果请求来的数据是不是要被其他组件公用，仅仅在请求的组件内使用，就不需要放入vuex 的state里。
- 如果被其他地方复用，这个很大几率上是需要的，如果需要，请将请求放入action里，方便复用，并包装成promise返回，在调用处用async await处理返回的数据。如果不要复用这个请求，那么直接写在vue文件里很方便。

6.8.13 Vuex中如何异步修改状态?

actions去异步的改变state的状态，mutations是同步改变状态，调用actions内定义的方法，需要通过this.$store.dispatch(),mutations方法是通过this.$store.commit()来进行调用,而在actions要调用mutations的方法，通过commit来进行调用

6.8.14 Vuex中actions和mutations的区别?

- Action 提交的是 mutation，而不是直接变更状态。
- Action 可以包含任意异步操作
- mutations只能是同步操作

6.8.15 页面刷新后vuex的state数据丢失怎么解决?

localStorage 或者就是sessionStorage ， 或者借用辅助插vuex-persistedstate

6.8.16 vuex怎么知道state是通过mutation修改还是外部直接修改的?

通过$watch监听mutation的commit函数中_committing是否为true

6.9 请你说一下你对vue组件通信的理解?

vue组件的通信是为了解决组件之间数据传递的问题，分为

- 父子组件之间的通信
- 非父子组件的通信

6.10 父组件如何与子组件怎么通信?

- 父组件将数据绑定在子组件上
- 子组件通过props属性来进行接收，props的接收方式有两种，分别是数组的方式接收，以及对象的方式接收，他们两个的不同是对象接收的方式可以设置默认值以及传递过来的类型

6.11 子组件如何与父组件进行通信?

- 在子组件里用`$emit`向父组件触发一个事件，父组件监听这个事件就行了

6.12 非父子组件之间如何进行通信?

- 非父子组件之间通信我们可以使用vuex或者event bus，而这个event bus我们把它称之为中央时间总线，vue中央事件总线这种方法适用于任何情况的父子组件通信，同级别组件通信，相当于组件通信间的万金油。但是碰到多人合作时，代码的维护性较低，代码可读性低（这个缺点可以忽略）。

6.15 除了组件之间的这种通信方式以外，还是什么方式可以让组件的数据进行共享?

路由，vuex，本地存储

6.16 props接收父组件发送过来的数据有几种形式?

- 两种，一种是数组，另外一种是对象

6.17 非父子组件之间通信的原理是什么?

- 非父子组件之间通信我们一般使用event bus，中央时间总线来进行解决，而中央事件总线的鱼哪里是通过vue实例化之后的对象调用bus.emit来进行数据的发送,通过bus.$on来进行接收

6.18 请描述vue的优点是什么？缺点是什么?

- vue的优点:
    - 简单易用
    - 灵活渐进式
    - 轻量高效
        - 压索之后20KB大小
        - 虚拟DOM
    - MVVM
        - 数据驱动视图
        - 常规的操作方式都是DOM
        - 普通的javascript数据
    - 组件化
        - 组件化优点
            - 提高开发效率
            - 方便重复使用
            - 简化调试步骤
            - 提升整个项目的可维护性
            - 便于协同开发
- vue的缺点:
    - VUE不支持IE8

6.19 请你描述一下vue中让元素隐藏的方式有几种？区别是什么？

- v-show v-if
  - **共同点**:
    - `v-if` 和 `v-show` 都是动态显示DOM元素。
  - **区别**
    - `编译过程：` v-if 是 真正 的 条件渲染，因为它会确保在切换过程中条件块内的`事件监听器`和`子组件`适当地`被销毁`和`重建`。`v-show` 的元素`始终会`被渲染并保留`在 DOM 中`。`v-show` 只是简单地`切换`元素的 CSS 属性`display`。
    - `编译条件：` v-if 是惰性的：如果在初始渲染时条件为假，则什么也不做。直到条件第一次变为真时，才会开始渲染条件块。`v-show`不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换
    - 性能消耗：`v-if有更高的切换消耗。v-show有更高的`初始渲染消耗`。
    - `应用场景：` v-if适合运行时条件很少改变时使用。v-show适合频繁切换

6.20 你在vue中怎么样让你写的css样式只在你当前写的组件中显示?

-

6.21 请你说一下keep-alive？你在哪些地方用过它?

- `keep-alive：`主要用于`保留组件状态`或`避免重新渲染`。
- **比如：** 有一个`列表页面`和一个`详情页面`，那么用户就会经常执行打开`详情=>返回列表=>打开详情`这样的话 列表 和 详情 都是一个`频率很高`的页面，那么就可以`对列表组件`使用`进行缓存，这样用户每次`返回列表`的时候，都能`从缓存中快速渲染，`而`不是重新渲染`

6.22 请你说一下在vue中如何获取dom元素?

`ref`

6.23 请你说一下vue中常用的指令有哪些?

`自己总结`

6.24 请你说一下为什么使用key?

- key值：用于 管理可复用的元素。因为`Vue`会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染。这么做使 Vue 变得非常快，但是这样也不总是符合实际需求。

6.25 说一下你对axios的理解?

- 什么是axios?
- axios一般什么时候用?
- 使用场景?
- 使用的时候遇到过什么问题?

6.26 说一下axios如何解决跨域?

`自己总结`

6.27 请描述v-model是什么？以及他的作用？以及他的原理是什么?

- v-model就是vue的双向绑定的指令,能将页面上控件输入的值同步更新到相关绑定的data属性，也会在更新data绑定属性时候，更新页面上输入控件的值
- v-model主要提供了两个功能，view层输入值影响data的属性值，data属性值发生改变会更新view层的数值变化

6.28 请描述computed，watch和methods的区别？以及他们的使用场景?

自己总结

6.29 请你描述一下你对$nextTick的理解?

- nextTick是vue里面提供的一个方法,当dom更新循环结束之后执行延迟回调,在修改数据之后可以使用 nextTick,那么我们可以在回调中获取更新后的dom，我们写项目的时候，当时点击按钮要获取一个元素的内容，但是发现了第二次点击的时候才回去到了，后台在网上查了一下，发现是vue异步更新队列的问题，后来是通过$nextTick解决的

6.30 说一下你对渐进式框架的理解?

- 就是主张最少，可以只用一部分功能，而不必使用全部，而且可以跟其他框架结合使用，

  没有多做职责之外的事

6.31 说一下你对vue数据双向绑定的理解?

- 就是利用了Object.defineProperty()这个方法重新定义了对象获取属性get和设置属性set来操作实现的

6.32 说一下vue单页面和多页面的区别?

- 单页面就是组件之间来回跳转，跳转速度快，不需要请求数据 缺点：首屏加载慢，跳转快
- 多页面就是页面之间来回跳转，跳转速度慢，每次跳转都需要向后台请求数据 缺点：首屏加载快，跳转速度慢

6.33 请你说一下什么是vue的过滤器？你在项目中哪些地方使用过过滤器?

自己总结

6.34 请你说一下你对vue指令的理解？以及他的使用场景？ 并描述你在项目中安歇地方使用过vue自定义指令?

自己总结

6.35 请你说一下vue的核心是什么?

- vue的核心是：数据驱动，组件化开发
  - 数据驱动：
    - mvvm模式
  - 组件化开发:
    - 就是内聚性和藕合度（高内聚，低藕合）

6.36 请你说一下vue和jquery的区别?

- jquery是直接操作DOM的而vue是操作数据的
- vue做到了数据和视图完全分离，他首先把值和JS对象进行绑定，然后在修改JS对象的值，vue框架就会自动把DOM的值进行更新，对数据进行操作不在需要引用相应的DOM对象，他们通过Vue对象实现数据和视图的相互绑定
- jquery则是先使用选择器($)来选取Dom对象，然后对Dom对象进行操作（如赋值、取值、事件绑定等）

6.37 请你说一下你在vue打包项目的时候有没有出现什么问题？你是如何解决的?

自己总结

6.38 请你描述一下react和vue的区别是什么?

自己总结

6.39 请你说一下如何优化vue首屏加载的速度?

自己总结

6.40 请你说一下你对slot的理解?

自己总结

6.41 请你描述一下封装vue组件的过程?

自己总结

6.42 如果说你在开发项目的时候，后台的接口还没有写完，请问这个时候你一般会怎么做

自己总结

6.43 vue如何封装通用组件?

自己总结

6.44 vue常用的ui组件库有哪些?

自己总结

6.45 vue常用的修饰符一共有哪些?

自己总结

6.46 请你说一下ajax和axios的区别是什么?

自己总结

6.47 vue组件如何适配移动端?

自己总结

6.48 说一下在vue中如何使用背景图片?

自己总结

6.49 如何解决禁用表单后移动端样式不统一问题?

自己总结

6.50 请你说一下数据双向绑定的原理是什么?

自己总结

6.51 什么是请求拦截，什么响应拦截？ 拦截点分别是那几个?

自己总结

七、ECMAScript6面试题

7.1 请描述let与const以及var的区别？以及什么是暂时性死区？什么是变量提升?

- 区别:
  - let 具有块级作用 不能重复声明 可以重复赋值
  - const 具有块级作用域 不能重复声明 不能重复赋值
  - var 全局作用域 可以重复声明 可以重复赋值
- 暂时性死区:
  - 我个人理解，所谓的暂时性死区就是在会计作用域内使用let声明了变量，那么这个变量就不会受外部的影响，这个我把它理解为暂时性死区。
- 变量提升:
  - 我个人理解，所谓的变量提升就是为了先事先声明变量，然后在进行赋值

7.2 请说一下你对es6的模版字符串的理解？有什么特点?

- 我个人理解，所谓的模版字符串其实指的是我们拼接字符串的时候，是通过连接符"+"来接的，并且如果换行等需要使用转义字符，否则就会报错。这样让我们书写十分不便。所以，ES6中就引入了模板字符串，帮助我们解决这一问题，并且，在模版字符串内使用${}包裹变量，就是将声明的变量进行解析
- 使用方式为 反单引号 可以直接插入变量 可以进行换行们不需要使用转义符号进行换行

7.3 请说一下箭头函数与普通函数的区别?

- 普通函数是很早就提出的，而箭头函数是es6提出的，他们两个在语法上不一样，并在普通函数与箭头函数他们this的指向也不要一样，普通函数内的this指向是如果没有绑定事件元素的时候，this指向的window，或者在闭包中this指向的也是window，如果函数绑定了事件，但并没有产生闭包，这个this指向的是当前调用的事件对象，箭头函数内this的指向是父作用域
- 箭头函数不能使用arguments，普通函数可以使用，arguments是以集合的方式获取函数传递的参数
- 箭头函数不能实例化为构造函数，而普通函数可以进行实例化

7.4 请说一下什么是函数的默认参数?

- 所谓的函数的默认参数其实指的就是当没有给函数参数进行传参的时候，可以给函数的形参制定默认值

7.5 请说一下Object.assign()的有什么作用

- Object.assign()方法主要是用于将源对象复制到目标对象，Object.assign()方法有两个参数，第一个参数的表示目标对象，第二个参数表示源对象。
- 在项目一般使用object.assign()用来对对象进行合并

7.6 请说一下你对promise的理解？并说一下promise你是如何使用的?

- 我个人对promise的理解是，promise是异步编程的一种解决方案，他比传统的回调函数加事件更加合理和强大，目前我用promise除了使用他的异步操作外，还使用promise在项目中解决了回调地狱等问题。
- 接下来，我在说一下promise的特点，promise一共有两个特点:
    - 对象不受外界影响，并且promise一共有三个状态，分别是进行中，成功，或者失败，只有异步操作的结果，可以决定是哪一种状态，任何其他的操作都无法改变这个状态
    - 一旦状态改变，就不会再变，任何时候都可以得到这个结果，promise的状态改变只有两种可能，要么是成功，要么失败。
- 如果要使用promise必须还要对promise进行实例化，实例化之后promise内有一个回调函数，这个函数里面有两个参数，分别是resolve和reject，当我们的状态发生变化的时候，如果是成功则会通过resolve将成功的结果返回出去，如果失败，那么可以通过reject将错误的信息也返回出去，我们可以通过.then方法接收返回的是成功的结果，通过catch可以接受失败的结果。
- promise常用的方法还有promise.all方法，主要是用来将多个实例包装成一个新的实例，以及promise.rece()方法
- 那么在项目中我一般使用promise来对api接口进行封装，以及一些异步的操作，都会使用到promise

7.7 请说一下你对es6模块化的理解

> 在说这个es6模块化之前，我觉得我个人有必要先介绍什么是模块化，模块化开发是一种管理方式，=是一种生产方式，也可以理解为是一种解决方案，一个模块其实就是代表一个实现特定功能文件，那么在这里大家也可以理解为一个js文件就是一个模块，有了模块之后我们就可以方便的使用别人的代码，想要什么功能，就加载什么模块，但是模块开发需要一定的方案，所以后来慢慢衍生出了amd，cmd规范，不过es6也给我们提供了一种模块化的方案，分别是import和export，也就是模块的导入和导出

> amd和cmd规范，其实amd规范是浏览器端的一种规范，而cmd规范是服务器端模块化的一种规范，之间amd规范的主要代表是sea.js，不过sea.js目前淘汰了，在前台开发中使用模块化的开发，我们使用的都是es6提供模块化方法，而cmd规范目前在node中我使用的cmd规范

> es6模块的语法是 import export

> cmd规范的语法是 require("") module.export

7.8 请说一下 es5与es6的区别?

- es5其实指的是ecmascript第五个版本
- es6其实指的是ecmascript第六个版本
- es6在es5的基础上新增了很多的新特性

7.9 请说一下使用箭头函数应该要注意什么?

`自己总结`

7.10 请说一下es6有哪些新增的特性?

- 变量的声明方式，分别是let与const
- 变量的解构赋值
- 增加了一些字符串与数组的方法，比如：模版字符串，includes方法，startsWith方法,endsWith方法，还是其他等等，数组方法有map方法，filter方法，foreach方法，find方法，findindex()方法
- 增加了symbol类型
- 箭头函数，函数参数默认值
- promise
- 模块化
- class类
- async await
- set和map数据结构
- 正则

7.11 请说一下你对es6 class类的理解?

> 所谓的class类其实就是es5构造函数的语法糖，所谓的语法糖其实就是构造函数的另外一种写法而已

7.12 请说一下Promise 中reject 和 catch 处理上有什么区别?

- reject 是用来抛出异常，catch 是用来处理异常
- reject 是 Promise 的方法，而 catch 是 Promise 实例的方法
- reject后的东西，一定会进入then中的第二个回调，如果then中没有写第二个回调，则进入catch
- 网络异常（比如断网），会直接进入catch而不会进入then的第二个回调

7.13 请说一下什么是深拷贝，什么是浅拷贝？以及如何实现深拷贝与浅拷贝？用es6如何实现深拷贝?

`自己总结`

7.14 举一些ES6对String字符串类型做的常用升级优化?

- 优化部分：
  - `ES6`新增了字符串模板，在拼接大段字符串时，用反斜杠`()`取代以往的字符串相加的形式，能保留所有空格和换行，使得字符串拼接看起来更加直观，更加优雅
- 升级部分：
  - `ES6`在`String`原型上新增了`includes()`方法，用于取代传统的只能用`indexOf`查找包含字符的方法(`indexOf`返回`-1`表示没查到不如`includes`方法返回`false`更明确，语义更清晰),此外还新增了`startsWith()`，`endsWith()`，`padStart()`,`padEnd()`,`repeat()`等方法，可方便的用于查找，补全字符串

7.15 举一些ES6对Array数组类型做的常用升级优化?

`ES6`在`Array`原型上新增了`find()`方法，用于取代传统的只能用`indexOf`查找包含数组项目的方法,且修复了`indexOf`查找不到`NaN的bug([NaN].indexOf(NaN) === -1)`.此外还新增了`copyWithin()`,`includes()`,`fill()`,`flat()`等方法，可方便的用于字符串的查找，补全,转换等

7.16 Map是什么，有什么作用?

`Map`是`ES6`引入的一种类似`Object`的新的数据结构，`Map`可以理解为是`Object`的超集，打破了以传统键值对形式定义对象，对象的`key`不再局限于字符串，也可以是`Object`。可以更加全面的描述对象的属性

7.17 Set是什么，有什么作用?

`Set`是`ES6`引入的一种类似`Array`的新的数据结构，`Set`实例的成员类似于数组`item`成员，区别是`Set`实例的成员都是唯一，不重复的。这个特性可以轻松地实现数组去重

7.18 Proxy是什么，有什么作用?

Proxy`是`ES6`新增的一个构造函数，可以理解为JS语言的一个代理，用来改变JS默认的一些语言行为，包括拦截默认的`get/set`等底层方法，使得JS的使用自由度更高，可以最大限度的满足开发者的需求。比如通过拦截对象的`get/set`方法，可以轻松地定制自己想要的`key`或者`value`。下面的例子可以看到，随便定义一个`myOwnObj`的`key`,都可以变成自己想要的函数

7.19 Class、extends是什么，有什么作用?

`ES6`的`class`可以看作只是一个`ES5`生成实例对象的构造函数的语法糖。它参考了`java`语言，定义了一个类的概念，让对象原型写法更加清晰，对象实例化更像是一种面向对象编程。`Class`类可以通过`extends`实现继承。它和ES5构造函数的不同点

7.20 常前端代码开发中，有哪些值得用ES6去改进的编程优化或者规范?

- 常用箭头函数来取代`var self = this`;的做法。
- 常用`let`取代`var`命令。
- 常用数组/对象的结构赋值来命名变量，结构更清晰，语义更明确，可读性更好
- 在长字符串多变量组合场合，用模板字符串来取代字符串累加，能取得更好地效果和阅读体验
- 用`Class`类取代传统的构造函数，来生成实例化对象
- 在大型应用开发中，要保持`module`模块化开发思维，分清模块之间的关系，常用`import`、`export`方法。

7.21 什么是 Babel?

> Babel 是一个 JS 编译器，自带一组 ES6 语法转化器，用于转化 JS 代码。

- 这些转化器让开发者提前使用最新的 JS语法(ES6/ES7)，而不用等浏览器全部兼容
- Babel 默认只转换新的 JS 句法(syntax)，而不转换新的API。

八、微信小程序面试题

8.1 简单描述下微信小程序的相关文件类型

- 微信小程序项目结构主要有四个文件类型
  - WXML（WeiXin Markup Language）是框***架**设计***的一套标签语言，结合基础组件、事件系统，可以构建出页面的结构。内部主要是微信自己定义的一套组件
  - WXSS (WeiXin Style Sheets)是一套样式语言，用于描述 WXML 的组件样式
  - js 逻辑处理，网络请求
  - json 小程序设置，如页面注册，页面标题及tabBar
- 主要文件
  - · app.json 必须要有这个文件，如果没有这个文件，项目无法运行，因为微信框架把这个作为配置文件入口，整个小程序的全局配置。包括页面注册，网络设置，以及小程序的 window

背景色，配置导航条样式，配置默认标题

- · app.js 必须要有这个文件，没有也是会报错！但是这个文件创建一下就行 什么都不需要写以后我们可以在这个文件中监听并处理小程序的生命周期函数、声明全局变量
- · app.wxss 可选

8.2 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 UI ， 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

8.3 小程序的双向绑定和vue哪里不一样

- 小程序直接 this.data 的属性是不可以同步到视图的，必须调用：

```
this.setData({
    // 这里设置
})
```

8.4 小程序的wxss和css有哪些不一样的地方?

WXSS 和 CSS 类似，不过在 CSS 的基础上做了一些补充和修改

- 尺寸单位 rpx

  rpx 是响应式像素,可以根据屏幕宽度进行自适应。规定屏幕宽为 750rpx。如在 iPhone6 上，屏幕宽度为 375px，共有 750 个物理像素，则 750rpx = 375px = 750 物理像素
- 使用 @import 标识符来导入外联样式。@import 后跟需要导入的外联样式表的相对路径，用;表示语句结束

```
/** index.wxss **/
@import './base.wxss';

.container{
    color: red;
}
```

8.5 小程序页面间有哪些传递数据的方法

- · 使用全局变量实现数据传递 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使用的时候，直接使用 getApp() 拿到存储的信息

```
App({
     // 全局变量
  globalData: {
    userInfo: null
  }
})
```

- · 使用 wx.navigateTo 与 wx.redirectTo 的时候，可以将部分数据放在 url 里面，并在新页面 onLoad 的时候初始化

```
/pageA.js

// Navigate
wx.navigateTo({
  url: '../pageD/pageD?name=raymond&gender=male',
})

// Redirect
wx.redirectTo({
  url: '../pageD/pageD?name=raymond&gender=male',
})


// pageB.js
...
Page({
  onLoad: function(option){
    console.log(option.name + 'is' + option.gender)
    this.setData({
      option: option
    })
  }
})
```

- 需要注意的问题：

  wx.navigateTo 和 wx.redirectTo 不允许跳转到 tab 所包含的页面

  onLoad 只执行一次

- · 使用本地缓存 Storage 相关

8.6 小程序的生命周期函数

- 小程序生命周期函数
- 页面生命周期函数
- 组件生命周期函数

`具体的钩子函数自己去查`

8.7 怎么封装微信小程序的数据请求?

`自己总结`

8.8 请说一下小程序的授权登录?

自己总结

8.9 哪些方法可以用来提高微信小程序的应用速度?

- 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

8.10 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

8.11 小程序关联微信公众号如何确定用户的唯一性?

如果开发者拥有多个移动应用、网站应用、和公众帐号（包括小程序），可通过 unionid 来区分用户的唯一性，因为只要是同一个微信开放平台帐号下的移动应用、网站应用和公众帐号（包括小程序），用户的 unionid 是唯一的。换句话说，同一用户，对同一个微信开放平台下的不同应用，unionid 是相同的

8.12 小程序如何实现下拉刷新?

- 首先在全局 config 中的 window 配置 enablePullDownRefresh
- 在 Page 中定义 onPullDownRefresh 钩子函数,到达下拉刷新条件后，该钩子函数执行，发起请求方法
- 请求返回后，调用 wx.stopPullDownRefresh 停止下拉刷新

> 参考 这里：（https://juejin.im/post/5a781c756fb9a063606eb742）

> 下拉刷新和上拉加载是业务上一个很常见的需求，在微信小程序里，提供了下拉刷新的方法 onPullDownRefresh 。而实现上拉加载相对来说就比较不方便了

***下拉刷**新**

虽然微信的官方文档有很多坑，但下拉刷新介绍的还是很全面的。在这里稍稍带过。

· 首先在全局 config 中的 window 配置 enablePullDownRefresh .

· 在 Page 中定义 onPullDownRefresh 钩子函数。到达下拉刷新条件后，该钩子函数执行，发起请求方法。

· 请求返回后，调用 wx.stopPullDownRefresh 停止下拉刷新。

```
### *config\****

config = {

  pages: [

   'pages/index'

  ],

  window: {
```

```
    backgroundTextStyle: 'light',
    navigationBarBackgroundColor: '#ccc',
    navigationBarTitleText: 'WeChat',
    navigationBarTextStyle: '#000',
    enablePullDownRefresh: true
  }
 }
```

```
### *page\****
onPullDownRefresh() {
 wepy.showNavigationBarLoading()
 setTimeout(()=>{
  this.getData = '数据拿到了'
  wepy.stopPullDownRefresh()
  wepy.hideNavigationBarLoading()
  this.$apply()
 },3000)
}
```

效果如下：

你会发现下拉的过程有些僵硬。这实际上是没有添加背景色的原因，加上背景色后再试试。

现在感觉好多了吧。下拉刷新有现成的配置和方法，很容易实现，可上拉加载就不同了。

上拉加载

首先看一下要实现的效果，这是3g端的上拉加载。小程序要实现同样的效果。

首先功能有

· 点击回到顶部 这个很好实现，有对应的回到顶部函数

· 滑动屏幕记录当前页数 这个也很好实现，主要是监听滚动事件，判断对应滚动条高度，去计算其与子容器的高度即可。

· 上拉加载动画

这里有两个实现的方案。一个是 page 自带的下拉触底钩子事件 onReachBottom 能做的只是下拉到底部的时候通知你触底了，一个是 scroll-view 标签自带事件。现在用两个方法分别实现一下上拉加载。

上拉触底事件 onReachBottom

模板

```
<template>
 <view class="loading"></view>
 <view class="container"
•    @touchmove="moveFn"
•    @touchstart="startFn"
•    @touchend="endFn"
•    style="transform:translate3d(0,{{childTop}}px,0)">
  <repeat for="{{list}}"
•      key="index"
•      index="index"
•      item="item">
•    <view>{{ item }}<text>{{index}}</text></view>
  </repeat>
  </view>
</template>复制代码
钩子函数
data = {
 getData: '',
 top: 0,
 lastTop: 0,
 canDrag: false,
 list: []
}
onReachBottom() {
 this.canDrag = true
}
```

```
methods = {
 moveFn(ev) {
  let nowY = ev.changedTouches[0].clientY
  nowY = nowY-this.lastTop
  if(nowY > 0 )
   this.canDrag = false
  if( nowY<=0 && this.canDrag ) {
   this.top = nowY
  }
  if( -this.top>= this.maxTop  )
   this.top = -this.maxTop
 },
 startFn(ev) {
  this.lastTop = ev.changedTouches[0].clientY
 },
 endFn() {
  if(this.top <= -this.maxTop) {
   this.text = "去请求数据了"
   setTimeout(()=>{
    this.text = "请求回来了"
    this.canDrag = false
    this.list.push(...["数据","数据","数据"])
    this.$apply()
    this.top = 0;
    return
   },1000)
  }
 },
 gotoTop() {
```

```
  wepy.pageScrollTo({

   scrollTop: 0

  })

 }

}
```

完成后看一下效果：


滚动容器实现上拉加载

scroll-view： 可滚动视图区域。
它的具体用法不赘述，看官方文档就行了。这里提解决上述问题的方法即可。

· bindscrolltolower 类比原生全局钩子 onReachBottom
模板

```
<scroll-view   scroll-y

•        id="content"

•        @scroll="scroll"

•        @scrolltolower="lower"

•        scroll-top="{{gotoTopNum}}"

•        lower-threshold="100"

•        style="transform:translate3d(0,{{childTop}}px,0)">

  <view  class="sty-search"

•      @touchmove="moveContent"

•      @touchstart="startContent"

•      @touchend="endContent">...</view>

</scroll-view>
```

以上就是最终的模板，你可能在想为什么这么复杂。虽然复杂，但每个属性都是有用的，当然这其中有几个坑在等着我们。
首先节点分为滚动容器和子容器。

8.13 bindtap和catchtap的区别是什么?

- 相同点：首先他们都是作为点击事件函数，就是点击时触发。在这个作用上他们是一样的，可以不做区-分
- 不同点：他们的不同点主要是bindtap是不会阻止冒泡事件的，catchtap是阻值冒泡的

8.14 简述下 `wx.navigateTo()`，`wx.redirectTo()`，`wx.switchTab()`，`wx.navigateBack()`，`wx.reLaunch()`的区别

- 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 abBar 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，打开到应用内的某个页面

8.15 说一下小程序组件中如何进行通信?

`自己总结`

8.16 说一下小程序中的behaviors的作用?

`自己总结`

8.17 说一下小程序的observe的理解?

`自己总结`

8.18 说一下微信小程序支付功能如何实现?

`自己总结`

8.19 说一下什么是小程序云开发?

`自己总结`

8.20 说一下小程序中wxs?

`自己总结`

8.21 说一下在小程序中如何使用npm?以及如何使用echarts?

`自己总结`

九、JavaScript面试题

9.1闭包

- 什么是闭包?

  MDN的解释：闭包是函数和声明该函数的词法环境的组合。

  按照我的理解就是：闭包 =『函数』和『函数体内可访问的变量总和』

  说白了就是函数嵌套函数，内部函数能够访问外部函数的变量

```
(function() {
    var a = 1;
    function add() {
        var b = 2

        var sum = b + a
        console.log(sum); // 3
    }
    add()
})()
```

`add`函数本身，以及其内部可访问的变量，即 `a = 1`，这两个组合在一起就被称为闭包，仅此而已。

- 闭包的作用
  - 闭包最大的作用就是隐藏变量，闭包的一大特性就是**内部函数总是可以访问其所在的外部函数中声明的参数和变量，即使在其外部函数被返回（寿命终结）了之后**
  - 基于此特性，JavaScript可以实现私有变量、特权变量、储存变量等
  - 我们就以私有变量举例，私有变量的实现方法很多，有靠约定的（变量名前加_），有靠Proxy代理的，也有靠Symbol这种新数据类型的。
- 闭包的优点
  - 可以隔离作用域，不造成全局污染
- 闭包的缺点
  - 由于闭包长期驻留内存，则长期这样会导致内存泄露
  - 如何解决内存泄露：将暴露全外部的闭包变量置为null
- 适用场景：封装组件，for循环和定时器结合使用,for循环和dom事件结合.可以在性能优化的过程中,节流防抖函数的使用,导航栏获取下标的使用

写一个返回闭包的函数

```
function outer(){
     var val = 0;
     return function (){
           val += 1;
           document.write(val + "<br />");
     };
}
var outObj = outer();
outObj();//1，执行val += 1后，val还在
outObj();//2
outObj = null;//val 被回收
var outObj1 = outer();
outObj1();//1
outObj1();//2
```

9.2 谈谈你对原型链的理解?

答：原型链是理解JS面向对象很重要的一点，这里主要涉及到两个点，一是_ *proto* ，二是prototype,举个例子吧，这样还好说点，例如：我用function创建一个Person类，然后用new Person创建一个对象的实例假如叫p1吧，在Person类的原型 prototype添加一个方法，例如：play方法,那对象实例p1如何查找到play这个方法呢，有一个查找过程，具体流程是这样的：

首先在p1对象实例上查找是否有有play方法，如果有则调用执行，如果没有则用p1.**proto**(*proto*是一个指向的作用,指向上一层的原型)往创建p1的类的原型上查找，也就是说往Person.prototype上查找，如果在Person.prototype找到play方法则执行，否则继续往上查找，则用Person.prototye.**proto**继续往上查找，找到Object.prototype，如果Object.prototype有play方法则执行之，否则用Object.prototype.**proto**继续再往上查找，但Object.prototpye.**proto**上一级是null,也就是原型链的顶级，结束原型链的查找，这是我对原型链的理解

9.3说一下JS继承（含ES6的）--或者人家这样问有两个类A和B,B怎么继承A?

- JS继承实现方式也很多，主要分ES5和ES6继承的实现

  先说一下ES5是如何实现继承的

  ES5实现继承主要是基于prototype来实现的，具体有三种方法

  一是**原型链继承**：即 B.prototype=new A()

  二是借**用构造函数继承(call或者apply的方式继承)**

```
function B(name,age) {

 A.call(ths,name,age)

}
```

  三是组合继承

  组合继承是结合第一种和第二种方式

  再说一下ES6是如何实现继承的

- ES6继承是目前比较新，并且主流的继承方式，用class定义类，用extends继承类，用super()表示父类,【下面代码部分只是熟悉，不用说课】

```
      例如：创建A类

class A {

  constructor() {

   //构造器代码，new时自动执行

  }



 方法1( ) { //A类的方法 }

 方法2( ) { //A类的方法 }



}



创建B类并继承A类
```

```
class B extends A {

  constructor() {

   super() //表示父类

  }

}
```

实例化B类： var b1=new B( )

- b1.方法1( )

9.3 说一下JS原生事件如何绑定

- JS原生绑定事件主要为三种：

  一是html事件处理程序

  二是DOM0级事件处理程序

  三是DOM2级事件处理程序

  其中：html事件现在早已不用了，就是在html各种标签上直接添加事件，类似于css的行内样式，缺点是不好维护，因为散落在标签中,也就是耦合度太高

例如：

点我< /button>

第二类是DOM0级事件，目前在PC端用的还是比较多的绑定事件方式，兼容性也好，主要是先获取dom元素，然后直接给dom元素添加事件

```
例如：var btn=document.getElementById('id元素')

   btn.onclick=function() {

•      //要处理的事件逻辑

   }
```

DOM0事件如何移除呢？很简单：btn.onclick=null;置为空就行

优点：兼容性好

缺点：只支持冒泡，不支持捕获

第三类是DOM2级事件，移动端用的比较多，也有很多优点，提供了专门的绑定和移除方法

例如： var btn=document.getElementById('id元素')

//绑定事件

btn.addEventListener('click',绑定的事件处理函数名,false)

//移除事件

btn.removeEventListener('click',要移除的事件处理函数名，false)

优点：支持给个元素绑定多个相同事件，支持冒泡和捕获事件机制

9.4说一下JS原生常用dom操作方法?

js原生dom操作方法有

- 查找：
  - getElementByid,
  - getElementsByTagName,
  - querySelector,
  - querySelectorAll
- 插入：
  - appendChild,insertBefore
- 删除：
  - removeChild
- 克隆：
  - cloneNode
- 设置和获取属性：
  - setAttribute("属性名","值")
  - getAttibute("属性名")

9.5说一下ES6新增特性?

1. 新增了块级作用域(let,const)
2. 提供了定义类的语法糖(class)
3. 新增了一种基本数据类型(Symbol)
4. 新增了变量的解构赋值
5. 函数参数允许设置默认值，引入了rest参数，新增了箭头函数
6. 数组新增了一些API，如 isArray / from / of 方法;数组实例新增了 entries()，keys() 和 values() 等方法
7. 对象和数组新增了扩展运算符
8. ES6 新增了模块化(import/export)
9. ES6 新增了 Set 和 Map 数据结构
10. ES6 原生提供 Proxy 构造函数，用来生成 Proxy 实例
11. ES6 新增了生成器(Generator)和遍历器(Iterator)

9.6 JS设计模式有哪些(单例模式观察者模式等)

JS设计模式有很多，但我知道的有单例模式，观察者模式

- 单例模式：

  就是保证一个类只有一个实例，实现的方法一般是先判断实例存在与否，如果存在直接返回，如果不存在就创建了再返回，这就确保了一个类只有一个实例对象。在JavaScript里，单例作为一个命名空间提供者，从全局命名空间里提供一个唯一的访问点来访问该对象。

- 观察者模式:

  观察者的使用场合就是：当一个对象的改变需要同时改变其它对象，并且它不知道具体有多少对象需要改变的时候，就应该考虑使用观察者模式。

总的来说，观察者模式所做的工作就是在解耦，让耦合的双方都依赖于抽象，而不是依赖于具体。从而使得各自的变化都不会影响到另一边的变化

9.7 说一下你对JS面试对象的理解

JS面向对象主要基于function来实现的，通过function来模拟类，通过prototype来实现类方法的共享，跟其他语言有着本质的不同，自从有了ES6后，把面向对象类的实现更像后端语言的实现了，通过class来定义类，通过extends来继承父类，其实ES6类的实现本质上是一个语法糖，不过对于开发简单了好多

9.8 说一下JS数组常用方法（至少6个）

在开发中，数组使用频率很频繁，JS数组常用方法有

- push
- pop
- unshift
- shift
- splice
- join
- concat
- forEach
- filter
- map
- sort

- some
- every

好多，不过都是平时开发中很常用的方法,大家可以补充一点儿es6的

9.9 说一下JS数组内置遍历方法有哪些和区别

JS数组内置遍历（遍历就是循环的意思）方法主要有：

- **forEach**
  - 这个方法是为了取代for循环遍历数组的，返回值为undefined例如：
  - ```
    let arrInfo=[4,6,6,8,5,7,87]

    arrInfo.forEach((item,index,arr)=>{

      //遍历逻辑

    })
    ```

    其中：

    item代码遍历的每一项,

    index:代表遍历的每项的索引,

    arr代表数组本身
- **filter**
  - 是一个过滤遍历的方法，如果返回条件为true，则返回满足条件为true的新数组

    ```
      let arrInfo=[4,16,6,8,45,7,87]

    •       let resultArr=arrInfo.filter((item,index,arr)=>{

      //例如返回数组每项值大于9的数组

      return item>9

    })
    ```

- **map**
  - 这个map方法主要对数组的复杂逻辑处理时用的多，特别是react中遍历数据，也经常用到，写法和forEach类似
- **some**
  - 这个some方法用于只要数组中至少存在一个满足条件的结果，返回值就为true,否则返回fasel, 写法和forEach类似
- **every**
  - 这个every方法用于数组中每一项都得满足条件时，才返回true，否则返回false, 写法和forEach类似
    ```

9.10 说一下JS作用域

JS作用域也就是JS识别变量的范围，作用域链也就是JS查找变量的顺序

- 先说作用域，JS作用域主要包括全局作用域、局部作用域和ES6的块级作用域
    - 全局作用域：也就是定义在window下的变量范围，在任何地方都可以访问，
    - 局部作用域：是只在函数内部定义的变量范围
    - 块级作用域：简单来说用let和const在任意的代码块中定义的变量都认为是块级作用域中的变量，例如在for循环中用let定义的变量，在if语句中用let定义的变量等等

注: 1. 尽量不要使用全局变量，因为容易导致全局的污染，命名冲突，对bug查找不利。

2. 而所谓的作用域链就是由最内部的作用域往最外部,查找变量的过程.形成的链条就是作用域链

9.11 说一下从输入URL到页面加载完中间发生了什么?

大致过程是这样的：

1. 通过DNS服务器：url=>ip地址；
2. 到达ip地址对应的服务器；
3. 服务器接收用户的请求；
4. 把处理后的结果返回给客户端；
5. 客户端把结果渲染到浏览器即可，最后页面显示出来

输入了一个域名,域名要通过DNS解析找到这个域名对应的服务器地址(ip),通过TCP请求链接服务,通过WEB服务器(apache)返回数据,浏览器根据返回数据构建DOM树,通过css渲染引擎及js解析引擎将页面渲染出来,关闭tcp连接

9.12 说一下JS事件代理（也称事件委托）是什么，及实现原理?

JS事件代理就是通过给父级元素（例如：ul）绑定事件，不给子级元素(例如：li)绑定事件，然后当点击子级元素时，通过事件冒泡机制在其绑定的父元素上触发事件处理函数，主要目的是为了提升性能，因为我不用给每个子级元素绑定事件，只给父级元素绑定一次就好了,在原生js里面是通过event对象的targe属性实现

```
var ul = document.querySelector("ul");

 ul.onclick = function(e){//e指event,事件对象

 var target = e.target || e.srcElement; //target获取触发事件的目标(li)

   if(target.nodeName.toLowerCase() == 'li'){//目标(li)节点名转小写字母，不转的话是大写字母

•     alert(target.innerHTML)

   }

 }
```

jq方式实现相对而言简单 $(“ul”).on(“click”,“li”,function(){//事件逻辑}) 其中第二个参数指的是触发事件的具体目标，**特别是给动态添加的元素绑定事件，这个特别起作用**

9.13 说一下js数据类型有哪些?

js数据类型有：

基本数据类型

- umber
- string
- Boolean
- null
- undefined
- symbol（ES6新增）

复合类型有

- Object
- function

9.14 说一下 call,apply,bind区别

call,apply,bind主要作用都是改变this指向的，但使用上略有区别，说一下区别

- call和apply的主要区别是在传递参数上不同，**call后面传递的参数是以逗号的形式分开的，apply传递的参数是数组形式 [Apply是以A开头的,所以应该是跟Array(数组)形式的参数]**
- bind返回的是一个函数形式，如果要执行，则后面要再加一个小括号 例如：bind(obj,参数1,参数2,)(),bind只能以逗号分隔形式，不能是数组形式

9.15 JavaScript的作用域链理解吗

JavaScript属于静态作用域，即声明的作用域是根据程序正文在编译时就确定的，有时也称为词法作用域。

其本质是JavaScript在执行过程中会创造可执行上下文，可执行上下文中的词法环境中含有外部词法环境的引用，我们可以通过这个引用获取外部词法环境的变量、声明等，这些引用串联起来一直指向全局的词法环境，因此形成了作用域链。

9.16 ES6模块与CommonJS模块有什么区别?

ES6 Module和CommonJS模块的区别：

- CommonJS是对模块的浅拷贝，ES6 Module是对模块的引用,即ES6 Module只存只读，不能改变其值，具体点就是指针指向不能变，类似const
- import的接口是read-only（只读状态），不能修改其变量值。 即不能修改其变量的指针指向，但可以改变变量内部指针指向,可以对commonJS对重新赋值（改变指针指向），但是对ES6 Module赋值会编译报错。

ES6 Module和CommonJS模块的共同点：

- CommonJS和ES6 Module都可以对引入的对象进行赋值，即对对象内部属性的值进行改变。

9.17 null与undefined的区别是什么?

null表示为空，代表此处不应该有值的存在，一个对象可以是null，代表是个空对象，而null本身也是对象。

undefined表示『不存在』，JavaScript是一门动态类型语言，成员除了表示存在的空值外，还有可能根本就不存在（因为存不存在只在运行期才知道），这就是undefined的意义所在

9.18 那么箭头函数的this指向哪里?

箭头函数不同于传统JavaScript中的函数,箭头函数并没有属于自己的this,它的所谓的this是捕获其所在上下文的 this 值，作为自己的 this 值,并且由于没有属于自己的this,而箭头函数是不会被new调用的，这个所谓的this也不会被改变.

9.19 async/await是什么?

async 函数，就是 Generator 函数的语法糖，它建立在Promises上，并且与所有现有的基于Promise的API兼容。

1. Async—声明一个异步函数(async function someName(){...})

- 自动将常规函数转换成Promise，返回值也是一个Promise对象
- 只有async函数内部的异步操作执行完，才会执行then方法指定的回调函数
- 异步函数内部可以使用await

2. Await—暂停异步的功能执行(var result = await someAsyncCall()😉

- 放置在Promise调用之前，await强制其他代码等待，直到Promise完成并返回结果
- 只能与Promise一起使用，不适用与回调
- 只能在async函数内部使用

9.20 async/await相比于Promise的优势?

- 代码读起来更加同步，Promise虽然摆脱了回调地狱，但是then的链式调用也会带来额外的阅读负担
- Promise传递中间值非常麻烦，而async/await几乎是同步的写法，非常优雅
- 错误处理友好，async/await可以用成熟的try/catch，Promise的错误捕获非常冗余
- 调试友好，Promise的调试很差，由于没有代码块，你不能在一个返回表达式的箭头函数中设置断点，如果你在一个.then代码块中使用调试器的步进(step-over)功能，调试器并不会进入后续的.then代码块，因为调试器只能跟踪同步代码的『每一步』。

9.21 JavaScript的基本类型和复杂类型是储存在哪里的?

基本类型储存在栈中，但是一旦被闭包引用则成为常住内存，会储存在内存堆中。

复杂类型会储存在内存堆中。

9.22 简述同步与异步的区别

- 同步：
    - 浏览器访问服务器请求，用户看得到页面刷新，重新发请求,等请求完，页面刷新，新内容出现，用户看到新内容,进行下一步操作
    - 代码从上往下依次执行，执行完当前代码，才能执行下面的代码。（阻塞）
- 异步：

- 浏览器访问服务器请求，用户正常操作，浏览器后端进行请求。等请求完，页面不刷新，新内容也会出现，用户看到新内容
- 代码从上往下依次执行，没执行完当前代码，也能执行下面的代码。（非阻塞）

9.23 JavaScript垃圾回收原理?

- 在javascript中，如果一个对象不再被引用，那么这个对象就会被GC回收；
- 如果两个对象互相引用，而不再被第3者所引用，那么这两个互相引用的对象也会被回收。

9.24 请描述值类型(基本数据类型)和引用类型的区别?

- 值类型
- - 占用空间固定，保存在栈中（当一个方法执行时，每个方法都会建立自己的内存栈，在这个方法内定义的变量将会逐个放入这块栈内存里，随着方法的执行结束，这个方法的内存栈也将自然销毁了。因此，所有在方法中定义的变量都是放在栈内存中的；栈中存储的是基础变量以及一些对象的引用变量，`基础变量的值是存储在栈中`，而引用变量存储在栈中的是`指向堆中的数组或者对象的地址`，这就是为何修改引用类型总会影响到其他指向这个地址的引用变量。）
- - 保存与复制的是值本身
- - 使用typeof检测数据的类型
- - 基本类型数据是值类型
- 引用类型
- - 占用空间不固定，保存在堆中（当我们在程序中创建一个对象时，这个对象将被保存到运行时数据区中，以便反复利用（因为对象的创建成本通常较大），这个运行时数据区就是堆内存。堆内存中的对象不会随方法的结束而销毁，即使方法结束后，这个对象还可能被另一个引用变量所引用（方法的参数传递时很常见），则这个对象`依然不会被销毁`，只有当一个对象没有任何引用变量引用它时，系统的垃圾回收机制才会在核实的时候回收它。）
- - 保存与复制的是指向对象的一个指针
- - 使用instanceof检测数据类型
- - 使用new()方法构造出的对象是引用型

9.25 深拷贝和浅拷贝的区别？如何实现

浅拷贝只复制指向某个对象的指针，而不复制对象本身，新旧对象还是共享同一块内存。浅拷贝只复制对象的第一层属性
但**深拷贝**会另外创造一个一模一样的对象，新对象跟原对象不共享内存，修改新对象不会改到原对象。对对象的属性进行递归复制

实现方式

- 浅拷贝
  - 使用Object.assign({},obj)第一个参数是一个空对象，第二个参数是你要复制的对象；通过这个方法我们知道浅拷贝不能修改基础的数据类型，可以修改引用的数据类型；
  - ES6中的...扩展运算符来进行浅拷贝的实现；
  - Object.assign()实现

```
Object.assign() 方法可以把任意多个的源对象自身的可枚举属性拷贝给目标对象，然后返回目标对象。但是 Object.assign() 进行的是浅拷贝，拷贝的是对象的属性的引用，而不是对象本身。

var obj = { a: {a: "hello", b: 21} };

var initalObj = Object.assign({}, obj);

initalObj.a.a = "changed";

console.log(obj.a.a); //  "changed"
```

注意：当object只有一层的时候，是深拷贝，例如如下

```
var obj1 = { a: 10, b: 20, c: 30 };
var obj2 = Object.assign({}, obj1);
obj2.b = 100;
console.log(obj1);
// { a: 10, b: 20, c: 30 } <-- 沒被改到
console.log(obj2);
// { a: 10, b: 100, c: 30 }
```

- 深拷贝
  - **对象只有一层的话可以使用上面的：Object.assign()函数**
  - **转成 JSON 再转回来**

```
var obj1 = { body: { a: 10 } };
var obj2 = JSON.parse(JSON.stringify(obj1));
obj2.body.a = 20;
console.log(obj1);
// { body: { a: 10 } } <-- 沒被改到
console.log(obj2);
// { body: { a: 20 } }
console.log(obj1 === obj2);
// false
console.log(obj1.body === obj2.body);
// false
```

用JSON.stringify把对象转成字符串，再用JSON.parse把字符串转成新的对象。

  - **使用Object.create()方法**

直接使用var newObj = Object.create(oldObj)，可以达到深拷贝的效果。

```
function deepClone(initalObj, finalObj) {
  var obj = finalObj || {};
  for (var i in initalObj) {
    var prop = initalObj[i];         // 避免相互引用对象导致死循环，如
initalObj.a = initalObj的情况
    if(prop === obj) {
      continue;
    }
```

```
    if (typeof prop === 'object') {
      obj[i] = (prop.constructor === Array) ? [] : Object.create(prop);
    } else {
      obj[i] = prop;
    }
  }
  return obj;
}
```

9.26 浏览器是如何渲染页面的?

渲染的流程如下：

1.解析HTML文件，创建DOM树。

自上而下，遇到任何样式（link、style）与脚本（script）都会阻塞（外部样式不阻塞后续外部脚本的加载）。

2.解析CSS。优先级：浏览器默认设置<用户设置<外部样式<内联样式<HTML中的style样式；

3.将CSS与DOM合并，构建渲染树（Render Tree）

4.布局和绘制，重绘（repaint）和重排（reflow）

9.27 什么是JavaScript原型，原型链？有什么特点?

- 每个对象都会在其内部初始化一个属性，就是`prototype`(原型)，当我们访问一个对象的属性时
- 如果这个对象内部不存在这个属性，那么他就会去`prototype`里找这个属性，这`个prototype`又会有自己的`prototype`，于是就这样一直找下去，也就是我们平时所说的原型链的概念
- 关系：`instance.constructor.prototype = instance.__proto__`
- 特点：
-
  - `JavaScript`对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变
- 当我们需要一个属性的时，`Javascript`引擎会先看当前对象中是否有这个属性， 如果没有的
- 就会查找他的`Prototype`对象是否有这个属性，如此递推下去，一直检索到`Object`内建对象

9.28 json和jsonp的区别?

json返回的是一串json格式数据；而jsonp返回的是脚本代码（包含一个函数调用）

jsonp的全名叫做json with padding，就是把json对象用符合js语法的形式包裹起来以使其他的网站可以请求到，也就是将json封装成js文件传过去。

9.29 如何阻止冒泡?

冒泡型事件：事件按照从最特定的事件目标到最不特定的事件目标(document对象)的顺序触发。

w3c的方法是e.stopPropagation()，IE则是使用e.cancelBubble = true。

```
//阻止冒泡行为
function stopBubble(e) {
//如果提供了事件对象，则这是一个非IE浏览器
if ( e && e.stopPropagation )
    //因此它支持W3C的stopPropagation()方法
    e.stopPropagation();
else
    //否则，我们需要使用IE的方式来取消事件冒泡
    window.event.cancelBubble = true;
}
```

9.30 如何阻止默认事件?

w3c的方法是e.preventDefault()，IE则是使用e.returnValue = false

```
//阻止浏览器的默认行为
function stopDefault( e ) {
    //阻止默认浏览器动作(W3C)
    if ( e && e.preventDefault )
        e.preventDefault();
    //IE中阻止函数器默认动作的方式
    else
        window.event.returnValue = false;
    return false;
}
```

9.31 JavaScript事件流模型都有什么?

“事件冒泡”:事件开始由最具体的元素接受，然后逐级向上传播

“事件捕捉”:事件由最不具体的节点先接收，然后逐级向下，一直到最具体的

“DOM 事件流”:三个阶段:事件捕捉，目标阶段，事件冒泡

9.32 用 js 实现随机选取 10–100 之间的 10 个数字，存入一个数组，并排序。

```
function randomNub(aArray, len, min, max) {

  if (len >= (max - min)) {
      return '超过' + min + '-' + max + '之间的个数范围' + (max - min - 1)
+ '个的总数';
  }

  if (aArray.length >= len) {
    aArray.sort(function(a, b) {
      return a - b
```

```
    });
    return aArray;
  }

  var nowNub = parseInt(Math.random() * (max - min - 1)) + (min + 1);

  for (var j = 0; j < aArray.length; j++) {
      if (nowNub == aArray[j]) {
        randomNub(aArray, len, min, max);
        return;
    }
  }

  aArray.push(nowNub);

  randomNub(aArray, len, min, max);

  return aArray;
}
var arr=[];

randomNub(arr,10,10,100);
```

9.33 有这样一个 URL:http://item.taobao.com/item.htm?a=1&b=2&c=&d=xxx&e，请写一段 JS 程序提取 URL 中的各个 GET 参数(参数名和参数个数不确定)，将 其按 key-value 形式返回到一个 json 结构中，如{a:'1', b:'2', c:'', d:' xxx', e:undefined}。

```
function serilizeUrl(url) {
  var urlObject = {};
  if (/\?/.test(url)) {

      var urlString = url.substring(url.indexOf("?") + 1);

      var urlArray = urlString.split("&");

  for (var i = 0, len = urlArray.length; i < len; i++) {
      var urlItem = urlArray[i];
      var item = urlItem.split("=");
      urlObject[item[0]] = item[1];
    }
    return urlObject;
  }
  return null;
}
```

####

9.34 请你谈谈cookie的弊端?

缺点:
1.Cookie 数量和长度的限制。每个 domain 最多只能有 20 条 cookie，每个 cookie 长度 不能超过 4KB，否则会被截掉。

2.安全性问题。如果 cookie 被人拦截了，那人就可以取得所有的 session 信息。即使加密也与事无补，因为拦截者并不需要知道 cookie 的意义，他只要原样转发 cookie 就可以达到目的了。

3.有些状态不可能保存在客户端。例如，为了防止重复提交表单，我们需要在服务器端保存 一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

9.35 哪些操作会造成内存泄漏?

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。 垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用 数量为 0(没有其他对象引用过该对象)，或对该对象的惟一引用是循环的，那么该对象 的内存即可回收。

1. setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。
2. 闭包
3. 控制台日志
4. 循环(在两个对象彼此引用且彼此保留时，就会产生一个循环)

9.36 你如何优化自己的代码?

代码重用

避免全局变量(命名空间，封闭空间，模块化 mvc..)

拆分函数避免函数过于臃肿

9.37 JavaScript 中的强制转型是指什么?

在 JavaScript 中，两种不同的内置类型间的转换被称为强制转型。强制转型在 JavaScript 中有两种形式：显式和隐式。

这是一个显式强制转型的例子：

```
var a = "42";
var b = Number( a );
a; // "42"
b; // 42 -- 是个数字!
```

这是一个隐式强制转型的例子：

```
var a = "42";
var b = a * 1; // "42" 隐式转型成 42
a; // "42"
b; // 42 -- 是个数字!
```

9.38 解释 JavaScript 中的相等性。

JavaScript 中有严格比较和类型转换比较：

- 严格比较（例如 ===）在不允许强制转型的情况下检查两个值是否相等；
- 抽象比较（例如 ==）在允许强制转型的情况下检查两个值是否相等。

```
var a = "42";
var b = 42;
a == b; // true
a === b; // false
```

一些简单的规则：

- 如果被比较的任何一个值可能是 true 或 false，要用 ===，而不是 ==；
- 如果被比较的任何一个值是这些特定值（0、“”或 []），要用 ===，而不是 ==；
- 在其他情况下，可以安全地使用 ==。它不仅安全，而且在很多情况下，它可以简化代码，并且提升代码可读性。

9.39 你能解释一下 ES5 和 ES6 之间的区别吗?

- ECMAScript 5（ES5）：ECMAScript 的第 5 版，于 2009 年标准化。这个标准已在所有现代浏览器中完全实现。
- ECMAScript 6（ES6）或 ECMAScript 2015（ES2015）：第 6 版 ECMAScript，于 2015 年标准化。这个标准已在大多数现代浏览器中部分实现。

以下是 ES5 和 ES6 之间的一些主要区别：

箭头函数和字符串插值：

```
const greetings = (name) => {
 return `hello ${name}`;
}
const greetings = name => `hello ${name}`;
```

常量

常量在很多方面与其他语言中的常量一样，但有一些需要注意的地方。常量表示对值的“固定引用”。因此，在使用常量时，你实际上可以改变变量所引用的对象的属性，但无法改变引用本身。

```
const NAMES = [];
NAMES.push("Jim");
console.log(NAMES.length === 1); // true
NAMES = ["Steve", "John"]; // error
```

块作用域变量。

新的 ES6 关键字 let 允许开发人员声明块级别作用域的变量。let 不像 var 那样可以进行提升。

默认参数值

默认参数允许我们使用默认值初始化函数。如果省略或未定义参数，则使用默认值，也就是说 null 是有效值。

```
// 基本语法
function multiply (a, b = 2) {
 return a * b;
}
multiply(5); // 10
```

类定义和继承

ES6 引入了对类（关键字 class）、构造函数（关键字 constructor）和用于继承的 extend 关键字的支持。

for...of 操作符

for...of 语句将创建一个遍历可迭代对象的循环。

用于对象合并的 Spread 操作

```
const obj1 = { a: 1, b: 2 }
const obj2 = { a: 2, c: 3, d: 4}
const obj3 = {...obj1, ...obj2}
```

promise

promise 提供了一种机制来处理异步操作结果。你可以使用回调来达到同样的目的，但是 promise 通过方法链接和简洁的错误处理带来了更高的可读性。

```
const isGreater = (a, b) => {
return new Promise ((resolve, reject) => {
 if(a > b) {
 resolve(true)
 } else {
 reject(false)
 }
 })
}
isGreater(1, 2)
.then(result => {
 console.log('greater')
})
.catch(result => {
```

```
 console.log('smaller')
})
```

模块导出和导入

```
const myModule = { x: 1, y: () => { console.log('This is ES5') }}
export default myModule;
import myModule from './myModule';
```

9.40 解释 JavaScript 中“undefined”和“not defined”之间的区别

在 JavaScript 中，如果你试图使用一个不存在且尚未声明的变量，JavaScript 将抛出错误“var name is not defined”，让后脚本将停止运行。但如果你使用 typeof undeclared_variable，它将返回 undefined。

在进一步讨论之前，先让我们理解声明和定义之间的区别。

“var x”表示一个声明，因为你没有定义它的值是什么，你只是声明它的存在。

```
var x; // 声明 x
console.log(x); // 输出: undefined
```

“var x = 1”既是声明又是定义（我们也可以说它是初始化），x 变量的声明和赋值相继发生。在 JavaScript 中，每个变量声明和函数声明都被带到了当前作用域的顶部，然后进行赋值，这个过程被称为提升（hoisting）。

当我们试图访问一个被声明但未被定义的变量时，会出现 undefined 错误。

```
var x; // 声明
if(typeof x === 'undefined') // 将返回 true
```

当我们试图引用一个既未声明也未定义的变量时，将会出现 not defined 错误。

```
console.log(y); // 输出: ReferenceError: y is not defined
```

9.41 匿名和命名函数有什么区别?

```
var foo = function() { // 赋给变量 foo 的匿名函数
 // ..
};
var x = function bar(){ // 赋给变量 x 的命名函数 bar
 // ..
};
foo(); // 实际执行函数
x();
```

9.42 什么是 JavaScript 中的提升操作?

提升（hoisting）是 JavaScript 解释器将所有变量和函数声明移动到当前作用域顶部的操作。有两种类型的提升：

- 变量提升——非常少见
- 函数提升——常见

无论 var（或函数声明）出现在作用域的什么地方，它都属于整个作用域，并且可以在该作用域内的任何地方访问它。

```
var a = 2;
foo(); // 因为`foo()`声明被"提升"，所以可调用
function foo() {
 a = 3;
 console.log( a ); // 3
 var a; // 声明被"提升"到 foo() 的顶部
}
console.log( a ); // 2
```

十、WEB页面性能优化面试题

10.1 为什么要进行页面性能优化

- “网站页面的快速加载，能够建立用户对网站的信任，增加回访率，大部分的用户其实都期待页面能够在2秒内加载完成，而当超过3秒以后，[就会有接近40%的用户离开你的网站]
- 因此,我们要强调即使没有对性能有实质的优化,通过设计提高用户体验的这个过程,也算是性能优化,因为 GUI 开发直面用户,你让用户有了性能快的 **错觉**,这也叫性能优化了,毕竟用户觉得快,才是真的快...

10.2 减少HTTP请求

数据量不大的页面，就减少http请求数量，一次性返回过来，对于数据量大的页面，可以分段异步请求，让用户先看到一部分，再继续加载另外一部分

http协议是**无状态**的应用层协议，意味着**每次http请求都需要建立通信链路、进行数据传输**，而在**服务器端**，每个http都需要启动**独立的线程**去处理。这些通信和服务的开销都很昂贵，减少http请求的数目可有效提高访问性能。

10.3 合并CSS、合并javascript、合并图片。

将浏览器一次访问需要的javascript和CSS合并成一个文件，这样浏览器就只需要一次请求。图片也可以合并，多张图片合并成一张，如果每张图片都有不同的超链接，可通过CSS偏移响应鼠标点击操作，构造不同的URL。

10.4 合并CSS、合并javascript、合并图片

将浏览器一次访问需要的javascript和CSS合并成一个文件，这样浏览器就只需要一次请求。图片也可以合并，多张图片合并成一张，如果每张图片都有不同的超链接，可通过CSS偏移响应鼠标点击操作，构造不同的URL。

10.5 合理设置缓存

很少变化的图片资源可以直接通过 HTTP Header中的Expires设置一个很长的过期头 ;变化不频繁而又可能会变的资源可以使用 Last-Modifed来做请求验证。尽可能的让资源能够在缓存中待得更久。

10.6 将更新频率比较低的CSS、javascript、logo、图标等静态资源文件缓存在浏览器中

避免频繁的http请求。通过设置http头中的cache-control和expires的属性，可设定浏览器缓存，缓存时间可以是数天，甚至是几个月。

10.7 CSS放在页面最上部，javascript放在页面最下面

浏览器会在下载完成全部CSS之后才对整个页面进行渲染，因此最好的做法是将CSS放在页面最上面，让浏览器尽快下载CSS。如果将 CSS放在其他地方比如 BODY中，则浏览器有可能还未下载和解析到 CSS就已经开始渲染页面了，这就导致页面由无 CSS状态跳转到 CSS状态，用户体验比较糟糕，所以可以考虑将CSS放在HEAD中。
Javascript则相反，浏览器在加载javascript后立即执行，有可能会阻塞整个页面，造成页面显示缓慢，因此javascript最好放在页面最下面。但如果页面解析时就需要用到javascript，这时放到底部就不合适了。

10.8 减少作用域链查找

前文谈到了作用域链查找问题，这一点在循环中是尤其需要注意的问题。如果在循环中需要访问非本作用域下的变量时请在遍历之前用局部变量缓存该变量，并在遍历结束后再重写那个变量，这一点对全局变量尤其重要，因为全局变量处于作用域链的最顶端，访问时的查找次数是最多的。

低效率的写法：

```
<span style="font-size:14px;">// 全局变量
var globalVar = 1;
function myCallback(info){
    for( var i = 100000; i--;){
        //每次访问 globalVar 都需要查找到作用域链最顶端，本例中需要访问 100000 次
        globalVar += i;
    }
}
</span>
```

10.9 css和注释

尽量减少页面中的空格和注释，从而减少页面的大小。对于css、js可以使用压缩工具进行压缩后再发布

10.10 CSS Sprites（精灵图 || 雪碧图）

很多的小图片整合成一张大图，通过css来移动位置显示，从而减少向服务器请求数据的次数

10.11减少dome请求

页面的dom层级尽量的减少，没有用的，多余的dom层级都移除掉，越简洁运行越快

10.11使用外部的JavaScript和CSS

10.12 节流和防抖

防抖是控制次数，节流是控制频率

函数防抖和节流是优化**高频率**执行js代码的一种手段，js中的一些事件如浏览器的`resize、scroll`，鼠标的`mousemove、mouseover`，input输入框的`keypress`等事件在触发时，会不断地调用绑定在事件上的回调函数，极大地浪费资源，降低前端性能。为了优化体验，需要对这类事件进行调用次数的限制。

10.12.1防抖（debounce）

防抖和节流的作用都是防止函数多次调用。区别在于，假设一个用户一直触发这个函数，且每次触发函数的间隔小于wait，防抖的情况下只会调用一次，而节流的 情况会每隔一定时间（参数wait）调用函数。

- 延迟的抖动函数
  - 该实现思路很简单，就是将执行函数放到一个定时器中，如果在定时器触发之前没有事件执行，那么就触发该执行函数，否则清空定时器
  - 这是一个简单版的防抖，但是有缺陷，这个防抖只能在最后调用。一般的防抖会有immediate选项，表示是否立即调用。这两者的区别，举个栗子来说：

```
// func是用户传入需要防抖的函数
// wait是等待时间
const debounce = (func, wait = 50) => {
  // 缓存一个定时器id
  let timer = 0
  // 这里返回的函数是每次用户实际调用的防抖函数
  // 如果已经设定过定时器了就清空上一次的定时器
  // 开始一个新的定时器，延迟执行用户传入的方法
  return function(...args) {
    if (timer) clearTimeout(timer)
    timer = setTimeout(() => {
      func.apply(this, args)
    }, wait)
  }
}
// 不难看出如果用户调用该函数的间隔小于wait的情况下，上一次的时间还未到就被清除了，并不会执行函数
```

- 改进版的（立即执行的防抖动函数）

```
// 这个是用来获取当前时间戳的
function now() {
  return +new Date()
}
/**
 * 防抖函数，返回函数连续调用时，空闲时间必须大于或等于 wait，func 才会执行
 *
 * @param  {function} func         回调函数
 * @param  {number}   wait         表示时间窗口的间隔
 * @param  {boolean}  immediate    设置为ture时，是否立即调用函数
 * @return {function}              返回客户调用函数
 */
function debounce (func, wait = 50, immediate = true) {
  let timer, context, args

  // 延迟执行函数
  const later = () => setTimeout(() => {
    // 延迟函数执行完毕，清空缓存的定时器序号
    timer = null
    // 延迟执行的情况下，函数会在延迟函数中执行
    // 使用到之前缓存的参数和上下文
    if (!immediate) {
      func.apply(context, args)
      context = args = null
    }
  }, wait)

  // 这里返回的函数是每次实际调用的函数
  return function(...params) {
    // 如果没有创建延迟执行函数（later），就创建一个
    if (!timer) {
      timer = later()
      // 如果是立即执行，调用函数
      // 否则缓存参数和调用上下文
      if (immediate) {
        func.apply(this, params)
      } else {
        context = this
        args = params
      }
    // 如果已有延迟执行函数（later），调用的时候清除原来的并重新设定一个
    // 这样做延迟函数会重新计时
    } else {
      clearTimeout(timer)
      timer = later()
    }
  }
}
```

- 总结
  - 对于按钮防点击来说的实现：如果函数是立即执行的，就立即调用，如果函数是延迟执行的，就缓存上下文和参数，放到延迟函数中去执行。一旦我开始一个定时器，只要我定时器还在，你每次点击我都重新计时。一旦你点累了，定时器时间到，定时器重置为 `null`，就可以再次点击了。
  - 对于延时执行函数来说的实现：清除定时器ID，如果是延迟调用就调用函数

10.12.2节流（throttle)

防抖动和节流本质是不一样的。防抖动是将多次执行变为最后一次执行，节流是将多次执行变成每隔一段时间执行。

所谓节流，就是指连续触发事件但是在 n 秒中只执行一次函数。节流会稀释函数的执行频率。

- 使用时间戳的节流方案

```
        const debounce=(func,delay=1000)=>{
            let context,args;
            var prev=Date.now();
            return function(...parms){
                var now=Date.now();
                if((now-prev)>=delay){
                    context=this;
                    args=parms
                    func.apply(context,args);
                    prev=Date.now();
                }else{
                    context=args=null;
                }
            }
        }
```

- 使用时间戳+定时器版的节流方案

```
/**
 * underscore 节流函数，返回函数连续调用时，func 执行频率限定为 次 / wait
 *
 * @param  {function}   func      回调函数
 * @param  {number}     wait      表示时间窗口的间隔
 * @param  {object}     options   如果想忽略开始函数的的调用，传入{leading: false}。
 *                                如果想忽略结尾函数的调用，传入{trailing: false}
 *                                两者不能共存，否则函数不能执行
 * @return {function}             返回客户调用函数
 */
_.throttle = function(func, wait, options) {
    var context, args, result;
    var timeout = null;
    // 之前的时间戳
    var previous = 0;
    // 如果 options 没传则设为空对象
    if (!options) options = {};
    // 定时器回调函数
    var later = function() {
      // 如果设置了 leading，就将 previous 设为 0
      // 用于下面函数的第一个 if 判断
      previous = options.leading === false ? 0 : _.now();
      // 置空一是为了防止内存泄漏，二是为了下面的定时器判断
      timeout = null;
      result = func.apply(context, args);
      if (!timeout) context = args = null;
    };
    return function() {
```

```
      // 获得当前时间戳
      var now = _.now();
      // 首次进入前者肯定为 true
      // 如果需要第一次不执行函数
      // 就将上次时间戳设为当前的
      // 这样在接下来计算 remaining 的值时会大于0
      if (!previous && options.leading === false) previous = now;
      // 计算剩余时间
      var remaining = wait - (now - previous);
      context = this;
      args = arguments;
      // 如果当前调用已经大于上次调用时间 + wait
      // 或者用户手动调了时间
      // 如果设置了 trailing，只会进入这个条件
      // 如果没有设置 leading，那么第一次会进入这个条件
      // 还有一点，你可能会觉得开启了定时器那么应该不会进入这个 if 条件了
      // 其实还是会进入的，因为定时器的延时
      // 并不是准确的时间，很可能你设置了2秒
      // 但是他需要2.2秒才触发，这时候就会进入这个条件
      if (remaining <= 0 || remaining > wait) {
        // 如果存在定时器就清理掉否则会调用二次回调
        if (timeout) {
          clearTimeout(timeout);
          timeout = null;
        }
        previous = now;
        result = func.apply(context, args);
        if (!timeout) context = args = null;
      } else if (!timeout && options.trailing !== false) {
        // 判断是否设置了定时器和 trailing
        // 没有的话就开启一个定时器
        // 并且不能不能同时设置 leading 和 trailing
        timeout = setTimeout(later, remaining);
      }
      return result;
    };
  };
```

十一、WebPack面试题

11.01 webpack与grunt，gulp的不同?

Grunt、Gulp是基于任务运行的工具：

它们会自动执行指定的任务，就像流水线，把资源放上去然后通过不同插件进行加工，它们包含活跃的社区，丰富的插件，能方便的打造各种工作流。

Webpack是基于模块化打包的工具:

自动化处理模块,webpack把一切当成模块，当 webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle。

因此这是完全不同的两类工具,而现在主流的方式是用npm script代替Grunt、Gulp,npm script同样可以打造任务流.

11.02 webpack，rollup，parcel优劣?

- webpack适用于大型复杂的前端站点构建: webpack有强大的loader和插件生态,□打包后的文件实际上就是一个立即执行函数，这个立即执行函数接收一个参数，这个参数是模块对象，键为各个模块的路径，值为模块内容。立即执行函数内部则处理模块之间的引用，执行模块等,这种情况更适合文件依赖复杂的应用开发.
- rollup适用于基础库的打包，如vue、d3等: Rollup 就是将各个模块打包进一个文件中，并且通过 Tree-shaking 来删除无用的代码,可以最大程度上降低代码体积,但是rollup没有webpack如此多的的如代码分割、按需加载等高级功能，其更聚焦于库的打包，因此更适合库的开发.
- parcel适用于简单的实验性项目: 他可以满足低门槛的快速看到效果,但是生态差、报错信息不够全面都是他的硬伤，除了一些玩具项目或者实验项目不建议使用

11.03 有哪些常见的Loader?

- file-loader：把文件输出到一个文件夹中，在代码中通过相对 URL 去引用输出的文件
- url-loader：和 file-loader 类似，但是能在文件很小的情况下以 base64 的方式把文件内容注入到代码中去
- source-map-loader：加载额外的 Source Map 文件，以方便断点调试
- image-loader：加载并且压缩图片文件
- babel-loader：把 ES6 转换成 ES5
- css-loader：加载 CSS，支持模块化、压缩、文件导入等特性
- style-loader：把 CSS 代码注入到 JavaScript 中，通过 DOM 操作去加载 CSS。
- eslint-loader：通过 ESLint 检查 JavaScript 代码

11.04 有哪些常见的Plugin?

- define-plugin：定义环境变量
- html-webpack-plugin：简化html文件创建
- uglifyjs-webpack-plugin：通过`UglifyES`压缩`ES6`代码
- webpack-parallel-uglify-plugin: 多核压缩,提高压缩速度
- webpack-bundle-analyzer: 可视化webpack输出文件的体积
- mini-css-extract-plugin: CSS提取到单独的文件中,支持按需加载

11.05 分别介绍bundle，chunk，module是什么

- bundle：是由webpack打包出来的文件
- chunk：代码块，一个chunk由多个模块组合而成，用于代码的合并和分割
- module：是开发中的单个模块，在webpack的世界，一切皆模块，一个模块对应一个文件，webpack会从配置的entry中递归开始找出所有依赖的模块

11.06 Loader和Plugin的不同?

不同的作用:

- **Loader**直译为"加载器"。Webpack将一切文件视为模块，但是webpack原生是只能解析js文件，如果想将其他文件也打包的话，就会用到`loader`。 所以Loader的作用是让webpack拥有了加载和解析*非JavaScript文件*的能力。
- **Plugin**直译为"插件"。Plugin可以扩展webpack的功能，让webpack具有更多的灵活性。 在 Webpack 运行的生命周期中会广播出许多事件，Plugin 可以监听这些事件，在合适的时机通过 Webpack 提供的 API 改变输出结果。

不同的用法:

- **Loader**在`module.rules`中配置，也就是说他作为模块的解析规则而存在。 类型为数组，每一项都是一个`Object`，里面描述了对于什么类型的文件（`test`），使用什么加载(`loader`)和使用的参数（`options`）

- **Plugin**在`plugins`中单独配置。 类型为数组，每一项是一个`plugin`的实例，参数都通过构造函数传入。

11.07 webpack的构建流程是什么?

Webpack 的运行流程是一个串行的过程，从启动到结束会依次执行以下流程：

1. 初始化参数：从配置文件和 Shell 语句中读取与合并参数，得出最终的参数；
2. 开始编译：用上一步得到的参数初始化 Compiler 对象，加载所有配置的插件，执行对象的 run 方法开始执行编译；
3. 确定入口：根据配置中的 entry 找出所有的入口文件；
4. 编译模块：从入口文件出发，调用所有配置的 Loader 对模块进行翻译，再找出该模块依赖的模块，再递归本步骤直到所有入口依赖的文件都经过了本步骤的处理；
5. 完成模块编译：在经过第4步使用 Loader 翻译完所有模块后，得到了每个模块被翻译后的最终内容以及它们之间的依赖关系；
6. 输出资源：根据入口和模块之间的依赖关系，组装成一个个包含多个模块的 Chunk，再把每个 Chunk 转换成一个单独的文件加入到输出列表，这步是可以修改输出内容的最后机会；
7. 输出完成：在确定好输出内容后，根据配置确定输出的路径和文件名，把文件内容写入到文件系统。

在以上过程中，Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑，并且插件可以调用 Webpack 提供的 API 改变 Webpack 的运行结果

11.08 是否写过Loader和Plugin？描述一下编写loader或plugin的思路?

Loader像一个"翻译官"把读到的源文件内容转义成新的文件内容，并且每个Loader通过链式操作，将源文件一步步翻译成想要的样子。

编写Loader时要遵循单一原则，每个Loader只做一种"转义"工作。 每个Loader的拿到的是源文件内容（`source`），可以通过返回值的方式将处理后的内容输出，也可以调用`this.callback()`方法，将内容返回给webpack。 还可以通过`this.async()`生成一个`callback`函数，再用这个callback将处理后的内容输出出去。 此外`webpack`还为开发者准备了开发loader的工具函数集——`loader-utils`。

相对于Loader而言，Plugin的编写就灵活了许多。 webpack在运行的生命周期中会广播出许多事件，Plugin 可以监听这些事件，在合适的时机通过 Webpack 提供的 API 改变输出结果。

11.09 webpack的热更新是如何做到的？说明其原理?

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

原理：

首先要知道server端和client端都做了处理工作

1. 第一步，在 webpack 的 watch 模式下，文件系统中某一个文件发生修改，webpack 监听到文件变化，根据配置文件对模块重新编译打包，并将打包后的代码通过简单的 JavaScript 对象保存在内存中。
2. 第二步是 webpack-dev-server 和 webpack 之间的接口交互，而在这一步，主要是 dev-server 的中间件 webpack-dev-middleware 和 webpack 之间的交互，webpack-dev-middleware 调用 webpack 暴露的 API对代码变化进行监控，并且告诉 webpack，将代码打包到内存中。
3. 第三步是 webpack-dev-server 对文件变化的一个监控，这一步不同于第一步，并不是监控代码变化重新打包。当我们在配置文件中配置了devServer.watchContentBase 为 true 的时候，Server 会监听这些配置文件夹中静态文件的变化，变化后会通知浏览器端对应用进行 live reload。注意，这儿是浏览器刷新，和 HMR 是两个概念。
4. 第四步也是 webpack-dev-server 代码的工作，该步骤主要是通过 sockjs（webpack-dev-server 的依赖）在浏览器端和服务端之间建立一个 websocket 长连接，将 webpack 编译打包的各个阶段

的状态信息告知浏览器端，同时也包括第三步中 Server 监听静态文件变化的信息。浏览器端根据这些 socket 消息进行不同的操作。当然服务端传递的最主要信息还是新模块的 hash 值，后面的步骤根据这一 hash 值来进行模块热替换。

5. webpack-dev-server/client 端并不能够请求更新的代码，也不会执行热更模块操作，而把这些工作又交回给了 webpack，webpack/hot/dev-server 的工作就是根据 webpack-dev-server/client 传给它的信息以及 dev-server 的配置决定是刷新浏览器呢还是进行模块热更新。当然如果仅仅是刷新浏览器，也就没有后面那些步骤了。
6. HotModuleReplacement.runtime 是客户端 HMR 的中枢，它接收到上一步传递给他的新模块的 hash 值，它通过 JsonpMainTemplate.runtime 向 server 端发送 Ajax 请求，服务端返回一个 json，该 json 包含了所有要更新的模块的 hash 值，获取到更新列表后，该模块再次通过 jsonp 请求，获取到最新的模块代码。这就是上图中 7、8、9 步骤。
7. 而第 10 步是决定 HMR 成功与否的关键步骤，在该步骤中，HotModulePlugin 将会对新旧模块进行对比，决定是否更新模块，在决定更新模块后，检查模块之间的依赖关系，更新模块的同时更新模块间的依赖引用。
8. 最后一步，当 HMR 失败后，回退到 live reload 操作，也就是进行浏览器刷新来获取最新打包代码

11.10 如何用webpack来优化前端性能?

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

- 压缩代码:删除多余的代码、注释、简化代码的写法等等方式。可以利用webpack的 `UglifyJsPlugin` 和 `ParallelUglifyPlugin` 来压缩JS文件， 利用 `cssnano` （css-loader?minimize）来压缩css
-
- 利用CDN加速: 在构建过程中，将引用的静态资源路径修改为CDN上对应的路径。可以利用webpack对于 `output` 参数和各loader的 `publicPath` 参数来修改资源路径
- Tree Shaking: 将代码中永远不会走到的片段删除掉。可以通过在启动webpack时追加参数 `--optimize-minimize` 来实现
- Code Splitting: 将代码按路由维度或者组件分块(chunk),这样做到按需加载,同时可以充分利用浏览器缓存
- 提取公共第三方库: SplitChunksPlugin插件来进行公共模块抽取,利用浏览器缓存可以长期缓存这些无需频繁变动的公共代码

11.11 如何提高webpack的打包速度?

- happypack: 利用进程并行编译loader,利用缓存来使得 rebuild 更快,遗憾的是作者表示已经不会继续开发此项目,类似的替代者是thread-loader
- 外部扩展(externals): 将不怎么需要更新的第三方库脱离webpack打包，不被打入bundle中，从而减少打包时间,比如jQuery用script标签引入
- dll: 采用webpack的 DllPlugin 和 DllReferencePlugin 引入dll，让一些基本不会改动的代码先打包成静态资源,避免反复编译浪费时间
- 利用缓存: `webpack.cache` 、babel-loader.cacheDirectory、 `HappyPack.cache` 都可以利用缓存提高rebuild效率
- 缩小文件搜索范围: 比如babel-loader插件,如果你的文件仅存在于src中,那么可以 `include: path.resolve(__dirname, 'src')` ,当然绝大多数情况下这种操作的提升有限,除非不小心build了node_modules文件

11.12 如何提高webpack的构建速度?

1. 多入口情况下，使用 `CommonsChunkPlugin` 来提取公共代码
2. 通过 `externals` 配置来提取常用库
3. 利用 `DllPlugin` 和 `DllReferencePlugin` 预编译资源模块 通过 `DllPlugin` 来对那些我们引用但是绝对不会修改的npm包来进行预编译，再通过 `DllReferencePlugin` 将预编译的模块加载进来。

4. 使用`Happypack`实现多线程加速编译
5. 使用`webpack-uglify-parallel`来提升`uglifyPlugin`的压缩速度。 原理上`webpack-uglify-parallel`采用了多核并行压缩来提升压缩速度
6. 使用`Tree-shaking`和`Scope Hoisting`来剔除多余代码

11.13 怎么配置单页应用？怎么配置多页应用?

单页应用可以理解为webpack的标准模式，直接在`entry`中指定单页应用的入口即可，这里不再赘述

多页应用的话，可以使用webpack的`AutoWebPlugin`来完成简单自动化的构建，但是前提是项目的目录结构必须遵守他预设的规范。 多页应用中要注意的是：

- 每个页面都有公共的代码，可以将这些代码抽离出来，避免重复的加载。比如，每个页面都引用了同一套css样式表
- 随着业务的不断扩展，页面可能会不断的追加，所以一定要让入口的配置足够灵活，避免每次添加新页面还需要修改构建配置

十二、移动端、安卓、IOS兼容性面试题

12.1 IOS移动端click事件300ms的延迟相应?

移动设备上的web网页是有300ms延迟的，往往会造成按钮点击延迟甚至是点击失效。

这是由于区分单机事件和双击屏幕缩放的历史原因造成的。

- 解决方式：
  - fastclick可以解决在手机上点击事件的300ms延迟
  - zepto的touch模块，tap事件也是为了解决在click的延迟问题
  - 触摸屏的相应顺序为touchstart-->touchmove-->touchend-->click，也可以通过绑定ontouchstart事件，加快　事件的响应，解决300ms的延迟问题

12.2 一些情况下对非可点击元素（label，span）监听click事件，iso下不会触发?

css增加cursor：pointer

为元素设置上cursor:pointer属性后，会导致元素点击时出现一个蓝色的背景。

为元素设置`-webkit-tap-highlight-color: transparent`;可以解决这个问题。

transparent:透明值

12.3 说一下手机端如何做适配的?

前端做适配没有最好的方法，只有适合的方法，目前前端主要做适配的方法有：百分比，em,rem,媒体查询(即media query),flex布局（即弹性盒），vw,vh等

目前我在项目中用的多的是rem，flex布局，有时会用到媒体查询，在做pc响应式布局时用

主要是用了一个手淘的js库flexible.js,在页面变化时,检测页面宽度,除以10份,动态的赋值给font-size.属性.;而页面的布局我是通过rem来进行布局的,所以就可以适配所有的移动端设备了

12.4 键盘遮挡输入框?

```
Element.scrollIntoView()  //让当前元素滚动到浏览器窗口的可视区域内
解决方法：
var oHeight = $(document).height(); //浏览器当前的高度

  $(window).resize(function(){

       if($(document).height() < oHeight){

       $("#footer").css("position","static");
   }else{

       $("#footer").css("position","absolute");
   }

  });
```

12.5 安卓部分版本input里的placeholder位置偏上?

把input的line-height设为norma

12.6 1px边框问题?

transform: scale(calc(1/1));

####

12.7 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：

```
.icon-logo
{
    background-image: url(src/assets/logo.png);
    width: 24px;
    height: 24px;
    background-size: contain;
}
@media screen and (-webkit-min-device-pixel-ratio: 2)
{
    .icon-logo { background-image: url(src/assets/logo@2.png); }
}
@media screen and (-webkit-min-device-pixel-ratio: 3)
{
    .icon-logo { background-image: url(src/assets/logo@3.png); }
}
@media screen and (-webkit-min-device-pixel-ratio: 4)
{
    .icon-logo { background-image: url(src/assets/logo@4.png); }
```

```
}
```

12.8 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" />
```

12.9 上下拉动滚动条时卡顿、慢

```
body
{
    -webkit-overflow-scrolling:touch;
    overflow-scrolling:touch;
}
```

12.10 长时间按住页面出现闪退

```
element
{
    -webkit-touch-callout:none;
}
```

12.11 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;

}

十三、GIT面试题

13.1 列举工作中常用的几个git命令?

新增文件的命令：git add file或者git add .
提交文件的命令：git commit –m或者git commit –a
查看工作区状况：git status –s
拉取合并远程分支的操作：git fetch/git merge或者git pull
查看提交记录命令：git reflog

13.2 提交时发生冲突,你能解释冲突是如何产生的吗?你是如何解决的?

- 开发过程中，我们都有自己的特性分支，所以冲突发生的并不多，但也碰到过。诸如公共类的公共方法，我和别人同时修改同一个文件，他提交后我再提交就会报冲突的错误.
- 发生冲突，在IDE里面一般都是对比本地文件和远程分支的文件，然后把远程分支上文件的内容手工修改到本地文件，然后再提交冲突的文件使其保证与远程分支的文件一致，这样才会消除冲突，然后再提交自己修改的部分。特别要注意下，修改本地冲突文件使其与远程仓库的文件保持一致后，需要提交后才能消除冲突，否则无法继续提交。必要时可与同事交流，消除冲突.
- 发生冲突，也可以使用命令:
    - 通过git stash命令，把工作区的修改提交到栈区，目的是保存工作区的修改；
    - 通过git pull命令，拉取远程分支上的代码并合并到本地分支，目的是消除冲突；
    - 通过git stash pop命令，把保存在栈区的修改部分合并到最新的工作空间中；

13.3 如果本次提交误操作,如何撤销?

如果想撤销提交到索引区的文件，可以通过git reset HEAD file；如果想撤销提交到本地仓库的文件，可以通过git reset –soft HEAD^n恢复当前分支的版本库至上一次提交的状态，索引区和工作空间不变更；可以通过git reset –mixed HEAD^n恢复当前分支的版本库和索引区至上一次提交的状态，工作区不变更；可以通过git reset –hard HEAD^n恢复当前分支的版本库、索引区和工作空间至上一次提交的状态。

13.4 如何查看分支提交的历史记录？查看某个文件的历史记录呢?

- 查看分支的提交历史记录：
    - 命令git log –number：表示查看当前分支前number个详细的提交历史记录；
    - 命令git log –number –pretty=oneline：在上个命令的基础上进行简化，只显示sha-1码和提交信息；
    - 命令git reflog –number: 表示查看所有分支前number个简化的提交历史记录；
    - 命令git reflog –number –pretty=oneline：显示简化的信息历史信息；

      如果要查看某文件的提交历史记录，直接在上面命令后面加上文件名即可。
- 注意：如果没有number则显示全部提交次数。

13.5 能不能说一下git fetch和git pull命令之间的区别?

简单来说：git fetch branch是把名为branch的远程分支拉取到本地；

而git pull branch是在fetch的基础上，把branch分支与当前分支进行merge；因此pull = fetch + merge。

> git pull 命令从中央存储库中提取特定分支的新更改或提交，并更新本地存储库中的目标分支。

git fetch 也用于相同的目的，但它的工作方式略有不同。当你执行 git fetch 时，它会从所需的分支中提取所有新提交，并将其存储在本地存储库中的新分支中。如果要在目标分支中反映这些更改，必须在 git fetch 之后执行git merge。只有在对目标分支和获取的分支进行合并后才会更新目标分支。为了方便起见，请记住以下等式：

```
git pull = git fetch + git merge
```

简单的说，git merge和git rebase都是合并分支的命令。

git merge branch会把branch分支的差异内容pull到本地，然后与本地分支的内容一并形成一个committer对象提交到主分支上，合并后的分支与主分支一致；

git rebase branch会把branch分支优先合并到主分支，然后把本地分支的commit放到主分支后面，合并后的分支就好像从合并后主分支又拉了一个分支一样，本地分支本身不会保留提交历史。

13.6 git跟其他版本控制器有啥区别?

- GIT是分布式版本控制系统，其他类似于SVN是集中式版本控制系统。
- 分布式区别于集中式在于：每个节点的地位都是平等，拥有自己的版本库，在没有网络的情况下，对工作空间内代码的修改可以提交到本地仓库，此时的本地仓库相当于集中式的远程仓库，可以基于本地仓库进行提交、撤销等常规操作，从而方便日常开发。

13.7 我们在本地工程常会修改一些配置文件，这些文件不需要被提交，而我们又不想每次执行git status时都让这些文件显示出来，我们该如何操作?

首先利用命令touch .gitignore新建文件：

```
touch .gitignore
```

然后往文件中添加需要忽略哪些文件夹下的什么类型的文件：

```
/target/class
.settings
.imp
*.ini
```

注意：忽略/target/class文件夹下所有后缀名为.settings，.imp的文件，忽略所有后缀名为.ini的文件。

13.8 如何把本地仓库的内容推向一个空的远程仓库?

首先确保本地仓库与远程之间是连同的。如果提交失败，则需要进行下面的命令进行连通：

```
git remote add origin xxxx
```

注意：XXXX是你的远程仓库地址。
如果是第一次推送，则进行下面命令：

```
git push -u origin master
```

注意：-u 是指定origin为默认主分支
之后的提交，只需要下面的命令：

```
git push origin master
```

13.9 如果分支是否已合并为master，你可以通过什么手段知道?

答案很直接。

要知道某个分支是否已合并为master，你可以使用以下命令：

`git branch –merged` 它列出了已合并到当前分支的分支。

`git branch –no-merged` 它列出了尚未合并的分支。

13.10 描述一下你所使用的分支策略?

这个问题被要求用Git来测试你的分支经验，告诉他们你在以前的工作中如何使用分支以及它的用途是什么，你可以参考以下提到的要点：

功能分支（Feature branching）
要素分支模型将特定要素的所有更改保留在分支内。当通过自动化测试对功能进行全面测试和验证时，该分支将合并到主服务器中。

任务分支（Task branching）
在此模型中，每个任务都在其自己的分支上实现，任务键包含在分支名称中。很容易看出哪个代码实现了哪个任务，只需在分支名称中查找任务键。

发布分支（Release branching）
一旦开发分支获得了足够的发布功能，你就可以克隆该分支来形成发布分支。创建该分支将会启动下一个发布周期，所以在此之后不能再添加任何新功能，只有错误修复，文档生成和其他面向发布的任务应该包含在此分支中。一旦准备好发布，该版本将合并到主服务器并标记版本号。此外，它还应该再将自发布以来已经取得的进展合并回开发分支。

最后告诉他们分支策略因团队而异，所以我知道基本的分支操作，如删除、合并、检查分支等。

13.11 如何在Git中创建存储库?

这可能是最常见的问题，答案很简单。

要创建存储库，先为项目创建一个目录（如果该目录不存在），然后运行命令 **git init**。通过运行此命令，将在项目的目录中创建 .git 目录。

13.12 请描述什么是工作区、暂存区和本地仓库?

`自己总结`

13.13 请写出查看分支、创建分支、删除分支、切换分支、合并分支的命令以及写出解决冲突的思路?

自己总结

13.14 请写出将工作区文件推送到远程仓库的思路? （两种情况都写出来）

自己总结

13.15 请写出团队内部协作开发的流程?

自己总结

13.16 请写出远程跨团队协作开发的流程?

自己总结

13.17 请写出配置ssh的思路?

自己总结

13.18 请描述什么是GitLab,或者说出你对GitLab的理解?

自己总结

13.19 请写出你所参与的多人协同开发时候，项目都有哪些分支，分支名是什么，每个分支代表什么，以及分支是由谁合并?

自己总结

十四、HTML与CSS面试题

14.01 HTML、XHTML、XML有什么区别

- HTML(超文本标记语言): 在html4.0之前HTML先有实现再有标准，导致HTML非常混乱和松散
- XML(可扩展标记语言): 主要用于存储数据和结构，可扩展，大家熟悉的JSON也是相似的作用，但是更加轻量高效，所以XML现在市场越来越小了
- XHTML(可扩展超文本标记语言): 基于上面两者而来，W3C为了解决HTML混乱问题而生，并基于此诞生了HTML5，开头加入`<!DOCTYPE html>`的做法因此而来，如果不加就是兼容混乱的HTML，加了就是标准模式。

14.02 知道img的srcset的作用是什么?

> 可以设计响应式图片，我们可以使用两个新的属性srcset 和 sizes来提供更多额外的资源图像和提示，帮助浏览器选择正确的一个资源。
>
> srcset 定义了我们允许浏览器选择的图像集，以及每个图像的大小。
>
> sizes 定义了一组媒体条件（例如屏幕宽度）并且指明当某些媒体条件为真时，什么样的图片尺寸是最佳选择。
>
> 所以，有了这些属性，浏览器会：
>
> - 查看设备宽度
> - 检查 sizes 列表中哪个媒体条件是第一个为真
> - 查看给予该媒体查询的槽大小
> - 加载 srcset 列表中引用的最接近所选的槽大小的图像

> srcset提供了根据屏幕条件选取图片的能力

14.03 link和@import的区别?

- link属于XHTML标签，而@import是CSS提供的。
- 页面被加载时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载。
- import只在IE 5以上才能识别，而link是XHTML标签，无兼容问题。
- link方式的样式权重高于@import的权重。
- 使用dom控制样式时的差别。当使用javascript控制dom去改变样式的时候，只能使用link标签，因为@import不是dom可以控制的。

14.04 如何理解层叠上下文是什么?

- 层叠上下文是HTML元素的三维概念，这些HTML元素在一条假想的相对于面向（电脑屏幕的）视窗或者网页的用户的z轴上延伸，HTML元素依据其自身属性按照优先级顺序占用层叠上下文的空间

  ## 如何产生?

  触发一下条件则会产生层叠上下文：

  - 根元素 (HTML),
  - z-index 值不为 "auto"的 绝对/相对定位，
  - 一个 z-index 值不为 "auto"的 flex 项目 (flex item)，即：父元素 display: flex|inline-flex，
  - opacity 属性值小于 1 的元素（参考 the specification for opacity），
  - transform 属性值不为 "none"的元素，
  - mix-blend-mode 属性值不为 "normal"的元素，
  - filter值不为“none”的元素，
  - perspective值不为“none”的元素，
  - isolation 属性被设置为 "isolate"的元素，
  - position: fixed
  - 在 will-change 中指定了任意 CSS 属性，即便你没有直接指定这些属性的值（参考 这篇文章）
  - -webkit-overflow-scrolling 属性被设置 "touch"的元素

14.05 谈谈对BFC的理解 是什么?

书面解释：BFC(Block Formatting Context)这几个英文拆解

- Block: Block在这里可以理解为Block-level Box,指的是块级盒子的标准
- Formatting context：块级上下文格式化，它是页面中的一块渲染区域，并且有一套渲染规则，它决定了其子元素将如何定位，以及和其他元素的关系和相互作用

BFC是指**一个独立的渲染区域，只有Block-level Box参与， 它规定了内部的Block-level Box如何布局，并且与这个区域外部毫不相干**.

它的作用是在**一块独立的区域，让处于BFC内部的元素与外部的元素互相隔离**.

如何形成?

BFC触发条件:

- 根元素，即HTML元素
- position: fixed/absolute
- float 不为none
- overflow不为visible
- display的值为inline-block、table-cell、table-caption

- 作用是什么?
    - 防止margin发生重叠
    - 两栏布局，防止文字环绕等
    - 防止元素塌陷

14.06 HTML5和css3的新标签

	HTML5： nav, footer, header, section, hgroup, video, time, canvas, audio...CSS3: RGBA, opacity, text-shadow, box-shadow, border-radius, border-image, border-color, transform...;

14.07 html5有哪些新特性，移除了哪些元素？如何处理HTML5新标签的浏览器谦容问题？如何区分HTML和HTML5?

* HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

* 绘画 canvas

用于媒介回放的 video 和 audio 元素

本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；

sessionStorage 的数据在浏览器关闭后自动删除

语意化更好的内容元素，比如 article、footer、header、nav、section

表单控件，calendar、date、time、email、url、search

新的技术webworker, websockt, Geolocation

* 移除的元素

纯表现的元素：basefont，big，center，font, s，strike，tt，u；

对可用性产生负面影响的元素：frame，frameset，noframes；

支持HTML5新标签：

* IE8/IE7/IE6支持通过document.createElement方法产生的标签，

可以利用这一特性让这些浏览器支持HTML5新标签，

浏览器支持新标签后，还需要添加标签默认的样式：

* 当然最好的方式是直接使用成熟的框架、使用最多的是html5shim框架

14.08 为什么有时候人们用translate来改变位置 而不是定位?

translate()是transform的一个值。改变transform或opacity不会触发浏览器重新布局（reflow）或重绘（repaint），只会触发复合（compositions）。而改变绝对定位会触发重新布局，进而触发重绘和复合。transform使浏览器为元素创建一个 GPU 图层，但改变绝对定位会使用到 CPU。 因此translate()更高效，可以缩短平滑动画的绘制时间。

而translate改变位置时，元素依然会占据其原始空间，绝对定位就不会发生这种情况。

14.09 浏览器的内核分别是什么？经常遇到的浏览器的谦容性有哪些？原因，解决方法是什么，常用hack的技巧?

* IE浏览器的内核Trident、 Mozilla的Gecko、google的WebKit、 Opera内核Presto；

* png24为的图片在iE6浏览器上出现背景，解决方案是做成PNG8.

* 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。

* IE6双边距bug:块属性标签float后，又有横行的margin情况下，在ie6显示margin比设置的大。

浮动ie产生的双倍距离 #box{ float:left; width:10px; margin:0 0 0 100px;}

这种情况之下IE会产生20px的距离，解决方案是在float的标签样式控制中加入 ——*display:inline;将其转化为行内属性。(*这个符号只有ie6会识别)

渐进识别的方式，从总体中逐渐排除局部。

首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。

接着，再次使用“+”将IE8和IE7、IE6分离开来，这样IE8已经独立识别。

css

.bb{

background-color:#f1ee18;/*所有识别*/

.background-color:#00deff\9; /*IE6、7、8识别*/

+background-color:#a200ff;/*IE6、7识别*/

_background-color:#1e0bd1;/*IE6识别*/

}

* IE下,可以使用获取常规属性的方法来获取自定义属性,

也可以使用getAttribute()获取自定义属性;

Firefox下,只能使用getAttribute()获取自定义属性.

解决方法:统一通过getAttribute()获取自定义属性.

* IE下,even对象有x,y属性,但是没有pageX,pageY属性;

Firefox下,event对象有pageX,pageY属性,但是没有x,y属性.

* （条件注释）缺点是在IE浏览器下可能会增加额外的HTTP请求数。

* 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.

超链接访问过后hover样式就不出现了 被点击访问过的超链接样式不在具有hover和active了解决方法是改变CSS属性的排列顺序:

L-V-H-A： a:link {} a:visited {} a:hover {} a:active {}

14.10 html常见谦容性问题?

1.双边距BUG float引起的 使用display

2.3像素问题 使用float引起的 使用dislpay:inline -3px

3.超链接hover 点击后失效 使用正确的书写顺序 link visited hover active

4.IE z-index问题 给父级添加position:relative

5.Png 透明 使用js代码 改

6.Min-height 最小高度！ Important 解决’

7.select 在ie6下遮盖 使用iframe嵌套

8.为什么没有办法定义1px左右的宽度容器（IE6默认的行高造成的，使用over:hidden,zoom:0.08 line-height:1px）

9.IE5-8不支持opacity，解决办法：

.opacity {

  opacity: 0.4

 filter: alpha(opacity=60); /* for IE5-7 */

  -ms-filter: "progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"; /* for IE 8*/

}

\10. IE6不支持PNG透明背景，解决办法: IE6下使用gif图片

14.11 描述一个“reset”的css文件并如何使用它。知道normalize.css吗？你了解他们的不同之处？

重置样式非常多，凡是一个前端开发人员肯定有一个常用的重置CSS文件并知道如何使用它们。他们是盲目的在做还是知道为什么这么做呢？原因是不同的浏览器对一些元素有不同的默认样式，如果你不处理，在不同的浏览器下会存在必要的风险，或者更有戏剧性的性发生。

你可能会用Normalize来代替你的重置样式文件。它没有重置所有的样式风格，但仅提供了一套合理的默认样式值。既能让众多浏览器达到一致和合理，但又不扰乱其他的东西（如粗体的标题）。

在这一方面，无法做每一个复位重置。它也确实有些超过一个重置，它处理了你永远都不用考虑的怪癖，像HTML的audio元素不一致或line-height不一致。

14.12 什么是外边距重叠？重叠的结果是什么？

外边距重叠就是margin-collapse。

在CSS当中，相邻的两个盒子（可能是兄弟关系也可能是祖先关系）的外边距可以结合成一个单独的外边距。这种合并外边距的方式被称为折叠，并且因而所结合成的外边距称为折叠外边距。

折叠结果遵循下列计算规则：

两个相邻的外边距都是正数时，折叠结果是它们两者之间较大的值。

两个相邻的外边距都是负数时，折叠结果是两者绝对值的较大值。

两个外边距一正一负时，折叠结果是两者的相加的和。

14.15 box-sizing常用的属性有哪些?分别有啥?

box-sizing有两个值:content-box(W3C标准盒模型),border-box(怪异模型),

这个css 主要是改变盒子模型大小的计算形式

可能有人会问padding-box,这个之前只有 Firefox 标准实现了,目前50+的版本已经废除;

用一个栗子来距离,一个div的宽高分别100px,border为5px,padding为5px

```
 .test {
  box-sizing: content-box;
  border: 5px solid #f00;
  padding:5px;
  width: 100px;
  height: 100px;
 }
```

```
<!--
content-box的计算公式会把宽高的定义指向 content,border和 padding 另外计算,
也就是说 content + padding + border = 120px(盒子实际大小)

而border-box的计算公式是总的大小涵盖这三者, content 会缩小,来让给另外两者
content(80px) + padding(5*2px)* + *border(5*2px) = 100px
-->
```

14.16 CSS中transition和animate有何区别?animate如何停留在最后一帧!

这种问题见仁见智,我的回答大体是这样的..待我捋捋.

transition一般用来做过渡的, 没时间轴的概念, 通过事件触发(一次),没中间状态(只有开始和结束)

而animate则是做动效,有时间轴的概念(帧可控),可以重复触发和有中间状态;

过渡的开销比动效小,前者一般用于交互居多,后者用于活动页居多;

至于如何让animate停留在最后一帧也好办,就它自身参数的一个值就可以了

animation-fill-mode: forwards;

让我们来举个栗子....自己新建一个 html 跑一下....

```
.test {
 box-sizing: border-box;
 border: 5px solid #f00;
 padding: 5px;
 width: 100px;
 height: 100px;
 position:absolute;
 /*
 简写的姿势排序
 @keyframes name : 动画名
 duration 持续时间
 timing-function 动画频率
 delay 延迟多久开始
 iteration-count 循环次数
 direction 动画方式,往返还是正向
 fill-mode  一般用来处理停留在某一帧
 play-state running 开始,paused 暂停 ....
 更多的参数去查文档吧..我就不一一列举了
 */
 animation: moveChangeColor  ease-in 2.5s 1  forwards running;
}

@keyframes moveChangeColor {
 from {
  top:0%;
  left:5%;
  background-color:#f00
 }
 to{
```

```
  top:0%;
  left:50%;
  background-color:#ced;
 }
```

}

14.17 什么是css hack？ ie6,7,8的hack分别是什么？

答案：针对不同的浏览器写不同的CSS code的过程，就是CSS hack。

示例如下：

```
 #test{
   background-color:yellow; /ie8/
   +background-color:pink;  /ie7/
   _background-color:orange;  /ie6/ }
```

更好的方式是使用IE条件判断语句：

<!–[if lte IE 6]>

内容

<![endif]–>

等

14.18 谈谈以前端角度出发做好SEO需要考虑什么？

搜索引擎主要以:

外链数量和质量

网页内容和结构

来决定某关键字下的网页搜索排名。

前端应该注意网页结构和内容方面的情况：

Meta标签优化

主要包括主题（Title），网站描述(Description)。还有一些其它的隐藏文字比如Author（作者），Category（目录），Language（编码语种）等。

符合W3C规范的语义性标签的使用。

如何选取关键词并在网页中放置关键词

搜索就得用关键词。关键词分析和选择是SEO最重要的工作之一。首先要给网站确定主关键词（一般在5个上下），然后针对这些关键词进行优化，包括关键词密度（Density），相关度（Relavancy），突出性（Prominency）等等。

14.19 一个页面上有大量的图片(大型电商网站)，加载很慢，你有哪些方法优化这些图片的加载，给用户更好的体验。

图片懒加载，在页面上的未可视区域可以添加一个滚动条事件，判断图片位置与浏览器顶端的距离与页面的距离，如果前者小于后者，优先加载。

如果为幻灯片、相册等，可以使用图片预加载技术，将当前展示图片的前一张和后一张优先下载。

如果图片为css图片，可以使用CSSsprite，SVGsprite，Iconfont、Base64等技术。

如果图片过大，可以使用特殊编码的图片，加载时会先加载一张压缩的特别厉害的缩略图，以提高用户体验。

如果图片展示区域小于图片的真实大小，则因在服务器端根据业务需要先行进行图片压缩，图片压缩后大小与展示一致。

14.20 你能描述一下渐进增强和优雅降级之间的不同吗?

渐进增强 progressive enhancement：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。

优雅降级 graceful degradation：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

区别：优雅降级是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要。降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带。

“优雅降级”观点

“优雅降级”观点认为应该针对那些最高级、最完善的浏览器来设计网站。而将那些被认为“过时”或有功能缺失的浏览器下的测试工作安排在开发周期的最后阶段，并把测试对象限定为主流浏览器（如 IE、Mozilla 等）的前一个版本。

在这种设计范例下，旧版的浏览器被认为仅能提供“简陋却无妨 (poor, but passable)” 的浏览体验。你可以做一些小的调整来适应某个特定的浏览器。但由于它们并非我们所关注的焦点，因此除了修复较大的错误之外，其它的差异将被直接忽略。

“渐进增强”观点

“渐进增强”观点则认为应关注于内容本身。

内容是我们建立网站的诱因。有的网站展示它，有的则收集它，有的寻求，有的操作，还有的网站甚至会包含以上的种种，但相同点是它们全都涉及到内容。这使得“渐进增强”成为一种更为合理的设计范例。这也是它立即被 Yahoo! 所采纳并用以构建其“分级式浏览器支持 (Graded Browser Support)”策略的原因所在。

14.21 为什么利用多个域名来储存网站资源会更有效?

CDN缓存更方便

突破浏览器并发限制

节约cookie带宽

节约主域名的连接数，优化页面响应速度

防止不必要的安全问题

14.22 css中可以通过哪些属性定义，使得一个dom元素不显示在浏览器可视范围内?

设置display属性为none，或者设置visibility属性为hidden

设置宽高为0，设置透明度为0，设置z-index位置在-1000em

设置text-indent:-9999px;

14.23 超链接访问过后hover样式就不出现的问题是什么？如何解决?

答案：被点击访问过的超链接样式不再具有hover和active了,解决方法是改变CSS属性的排列顺序: L-V-H-A（link,visited,hover,active）

14.24 css中可以让文字在垂直和水平方向上重叠的两个属性是什么?

垂直方向：line-height

水平方向：letter-spacing

那么问题来了，关于letter-spacing的妙用知道有哪些么?

答案:可以用于消除inline-block元素间的换行符空格间隙问题。

14.25 display:none 与visibility:hidden的区别是什么?

display : 隐藏对应的元素但不挤占该元素原来的空间。

visibility: 隐藏对应的元素并且挤占该元素原来的空间。

即是，使用CSS display:none属性后，HTML元素（对象）的宽度、高度等各种属性值都将“丢失”;而使用visibility:hidden属性后，HTML元素（对象）仅仅是在视觉上看不见（完全透明），而它所占据的空间位置仍然存在。

14.26 IE的双边BUG：块级元素float后设置横向margin，ie6显示margin比设置的较大。

解决：加入_display：inline

14.27 absolute的containing block 计算方式跟正常流有什么不同?

lock-level boxes
一个 block-level element ('display' 属性值为 'block', 'list-item' 或是 ‘table’) 会生成一个 block-level box，这样的盒子会参与到 block-formatting context (一种布局的方式) 中。
block formatting context
在这种布局方式下，盒子们自所在的 containing block 顶部起一个接一个垂直排列，水平方向上撑满整个宽度 (除非内部的盒子自己内部建立了新的 BFC)。
containing block
一般来说，盒子本身就为其子孙建立了 containing block，用来计算内部盒子的位置、大小，而对内部的盒子，具体采用哪个 containing block 来计算，需要分情况来讨论：

若此元素为 inline 元素，则 containing block 为能够包含这个元素生成的第一个和最后一个 inline box 的 padding box (除 margin, border 外的区域) 的最小矩形；
否则则由这个祖先元素的 padding box 构成。
根元素所在的 containing block 被称为 initial containing block，在我们常用的浏览器环境下，指的是原点与 canvas 重合，大小和 viewport 相同的矩形；
对于 position 为 static 或 relative 的元素，其 containing block 为祖先元素中最近的 block container box 的 content box (除 margin, border, padding 外的区域)；
对于 position:fixed 的元素，其 containing block 由 viewport 建立；
对于 position:absolute 的元素，则是先找到其祖先元素中最近的 position 属性非 static 的元素，然后判断：
如果都找不到，则为 initial containing block。

14.28 知道css有个content属性吗？有什么作用？有什么应用?

知道。css的content属性专门应用在 before/after 伪元素上，用来插入生成内容。最常见的应用是利用伪类清除浮动。

```
//一种常见利用伪类清除浮动的代码

.clearfix:after {

  content:"."; //这里利用到了content属性

  display:block;

  height:0;

  visibility:hidden;

  clear:both; }

.clearfix {

  *zoom:1;

}
```

after伪元素通过 content 在元素的后面生成了内容为一个点的块级素，再利用clear:both清除浮动。

那么问题继续还有，知道css计数器（序列数字字符自动递增）吗？如何通过css content属性实现css计数器?

答案：css计数器是通过设置counter-reset 、counter-increment 两个属性 、及 counter()/counters() 一个方法配合after / before 伪类实现。

14.29 css新增伪类有哪些?

p:first-of-type 选择属于其父元素的首个

元素的每个

元素。

p:last-of-type 选择属于其父元素的最后

元素的每个

元素。

p:only-of-type 选择属于其父元素唯一的

元素的每个

元素。

p:only-child 选择属于其父元素的唯一子元素的每个

元素。

p:nth-child(2) 选择属于其父元素的第二个子元素的每个

元素。

:enabled、:disabled 控制表单控件的禁用状态。

:checked，单选框或复选框被选中。

14.30 sass，less是什么？大家为什么要使用他们?

他们是CSS预处理器。他是CSS上的一种抽象层。他们是一种特殊的语法/语言编译成CSS。

例如Less是一种动态样式语言. 将CSS赋予了动态语言的特性，如变量，继承，运算， 函数. LESS 既可以在客户端上运行 (支持IE 6+, Webkit, Firefox)，也可一在服务端运行 (借助 Node.js)。

为什么要使用它们?

结构清晰，便于扩展。

可以方便地屏蔽浏览器私有语法差异。这个不用多说，封装对浏览器语法差异的重复处理，减少无意义的机械劳动。

可以轻松实现多重继承。

完全兼容 CSS 代码，可以方便地应用到老项目中。LESS 只是在 CSS 语法上做了扩展，所以老的 CSS 代码也可以与 LESS 代码一同编译。

14.31 css动画animation

`transition`: 过渡动画

- `transition-property`: 属性
- `transition-duration`: 间隔
- `transition-timing-function`: 曲线
- `transition-delay`: 延迟
- 常用钩子: `transitionend`

```
animation / keyframes
```

- `animation-name`: 动画名称，对应`@keyframes`
- `animation-duration`: 间隔
- `animation-timing-function`: 曲线
- `animation-delay`: 延迟
- `animation-iteration-count`

  : 次数
  - `infinite`: 循环动画
- `animation-direction`

  : 方向
  - `alternate`: 反向播放
- `animation-fill-mode`

  : 静止模式
  - `forwards`: 停止时，保留最后一帧
  - `backwards`: 停止时，回到第一帧
  - `both`: 同时运用 `forwards / backwards`
- 常用钩子: `animationend`

动画属性: 尽量使用动画属性进行动画，能拥有较好的性能表现

- `translate`
- `scale`
- `rotate`
- `skew`
- `opacity`
- `color`

十五、HTTP面试题

15.01 http/https协议

1.0 协议缺陷:

- 无法复用链接，完成即断开，**重新慢启动和 TCP 3次握手**
- head of line blocking: **线头阻塞**，导致请求之间互相影响

1.1 改进:

- **长连接**(默认 keep-alive)，复用
- host 字段指定对应的虚拟站点
- 新增功能:
  - 断点续传
  - 身份认证
  - 状态管理

    - cache 缓存
        - Cache-Control
        - Expires
        - Last-Modified
        - Etag

2.0:

- 多路复用
- 二进制分帧层: 应用层和传输层之间
- 首部压缩
- 服务端推送

https: 较为安全的网络传输协议

- 证书(公钥)
- SSL 加密
- 端口 443

TCP:

- 三次握手
- 四次挥手
- 滑动窗口: 流量控制
- 拥塞处理
    - 慢开始
    - 拥塞避免
    - 快速重传
    - 快速恢复

缓存策略: 可分为 **强缓存** 和 **协商缓存**

- Cache-Control/Expires: 浏览器判断缓存是否过期，未过期时，直接使用强缓存，**Cache-Control 的 max-age 优先级高于 Expires**
- 当缓存已经过期时，使用协商缓存
    - 唯一标识方案: Etag(response 携带) & If-None-Match(request携带，上一次返回的 Etag): 服务器判断资源是否被修改，
    - 最后一次修改时间: Last-Modified(response) & If-Modified-Since (request，上一次返回的 Last-Modified)
        - 如果一致，则直接返回 304 通知浏览器使用缓存
        - 如不一致，则服务端返回新的资源
- Last-Modified 缺点：
    - 周期性修改，但内容未变时，会导致缓存失效
    - 最小粒度只到 s， s 以内的改动无法检测到
- Etag 的优先级高于 Last-Modified

15.02 常见状态码

1xx: 接受，继续处理

200: 成功，并返回数据

201: 已创建

202: 已接受

203: 成为，但未授权

204: 成功，无内容

205: 成功，重置内容

206: 成功，部分内容

301: 永久移动，重定向

302: 临时移动，可使用原有URI

304: 资源未修改，可使用缓存

305: 需代理访问

400: 请求语法错误

401: 要求身份认证

403: 拒绝请求

404: 资源不存在

500: 服务器错误

15.03 get/post

- get: 缓存、请求长度受限、会被历史保存记录
  - 无副作用(不修改资源)，幂等(请求次数与资源无关)的场景
- post: 安全、大数据、更多编码类型
-

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
书签	可收藏为书签	不可收藏为书签
缓存	能被缓存	不能缓存
编码类型	application/x-www-form-urlencoded	application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。
历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。 在发送密码或其他敏感信息时绝不要使用 GET ！	POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。

15.04 websocket

Websocket 是一个 **持久化的协议**， 基于 http ， 服务端可以 **主动 push**

- 兼容:
  - FLASH Socket
  - 长轮询： 定时发送 ajax
  - long poll： 发送 --> 有消息时再 response
- `new WebSocket(url)`
- `ws.onerror = fn`
- `ws.onclose = fn`
- `ws.onopen = fn`
- `ws.onmessage = fn`
- `ws.send()`

作者：郭东东
链接：https://juejin.im/post/6844903776512393224
来源：掘金
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

15.05 TCP三次握手

建立连接前，客户端和服务端需要通过握手来确认对方:

- 客户端发送 syn(同步序列编号) 请求，进入 syn_send 状态，等待确认
- 服务端接收并确认 syn 包后发送 syn+ack 包，进入 syn_recv 状态
- 客户端接收 syn+ack 包后，发送 ack 包，双方进入 established 状态

15.06 TCP四次挥手

- 客户端 -- FIN --> 服务端， FIN—WAIT
- 服务端 -- ACK --> 客户端， CLOSE-WAIT
- 服务端 -- ACK,FIN --> 客户端， LAST-ACK
- 客户端 -- ACK --> 服务端，CLOSED

15.07 Node 的 Event Loop: 6个阶段

- timer 阶段: 执行到期的 `setTimeout / setInterval` 队列回调
- I/O 阶段: 执行上轮循环残流的 `callback`
- idle, prepare
- poll: 等待回调
  - 1. 执行回调
  - 1. 执行定时器
    - 如有到期的 `setTimeout / setInterval`， 则返回 timer 阶段
    - 如有 `setImmediate`，则前往 check 阶段
- check
  - 执行 `setImmediate`
- close callbacks

###

15.08 跨域

- JSONP: 利用`<script>`标签不受跨域限制的特点，缺点是只能支持 get 请求

```
function jsonp(url, jsonpCallback, success) {
  const script = document.createElement('script')
  script.src = url
  script.async = true
  script.type = 'text/javascript'
  window[jsonpCallback] = function(data) {
    success && success(data)
  }
  document.body.appendChild(script)
}
复制代码
```

- 设置 CORS: Access-Control-Allow-Origin：*
- postMessage

15.09 安全

- XSS攻击: 注入恶意代码
  - cookie 设置 httpOnly
  - 转义页面上的输入内容和输出内容
- CSRF: 跨站请求伪造，防护:
  - get 不修改数据
  - 不被第三方网站访问到用户的 cookie
  - 设置白名单，不被第三方网站请求
  - 请求校验

15.10 缓存机制

1. 浏览器发送请求前，根据请求头的expires和cache-control判断是否命中（包括是否过期）强缓存策略，如果命中，直接从缓存获取资源，并不会发送请求。如果没有命中，则进入下一步。
2. 没有命中强缓存规则，浏览器会发送请求，根据请求头的last-modified和etag判断是否命中协商缓存，如果命中，直接从缓存获取资源。如果没有命中，则进入下一步。
3. 如果前两步都没有命中，则直接从服务端获取资源。

15.11 什么是HTTPS

HTTPS是在HTTP上建立SSL加密层，并对传输数据进行加密，是HTTP协议的安全版。现在它被广泛用于万维网上安全敏感的通讯，例如交易支付方面。

HTTPS主要作用是：

（1）对数据进行加密，并建立一个信息安全通道，来保证传输过程中的数据安全;

（2）对网站服务器进行真实身份认证。

我们经常会在Web的登录页面和购物结算界面等使用HTTPS通信。使用HTTPS通信时，不再用`http://`，而是改用`https://`。另外，当浏览器访问HTTPS通信有效的Web网站时，浏览器的地址栏内会出现一个带锁的标记。对HTTPS的显示方式会因浏览器的不同而有所改变。

15.12 为什么需要HTTPS

在HTTP协议中有可能存在信息窃取或身份伪装等安全问题。使用HTTPS通信机制可以有效地防止这些问题，接下来，我们先来了解下 HTTP协议存在的哪些问题：

- 通信使用明文（不加密），内容可能被窃听

由于HTTP本身不具备加密的功能，所以也无法做到对通信整体（使用HTTP协议通信的请求和响应的内容）进行加密。即，**HTTP报文使用明文（指未经过加密的报文）方式发送。**

HTTP明文协议的缺陷是导致数据泄露、数据篡改、流量劫持、钓鱼攻击等安全问题的重要原因。HTTP协议无法加密数据，所有通信数据都在网络中明文“裸奔”。通过网络的嗅探设备及一些技术手段，就可还原HTTP报文内容。

- 无法证明报文的完整性，所以可能遭篡改

所谓完整性是指信息的准确度。若无法证明其完整性，通常也就意味着无法判断信息是否准确。由于HTTP协议无法证明通信的报文完整性，因此，在请求或响应送出之后直到对方接收之前的这段时间内，即使请求或响应的内容遭到篡改，也没有办法获悉。 换句话说，**没有任何办法确认，发出的请求/响应和接收到的请求/响应是前后相同的。**

- 不验证通信方的身份，因此有可能遭遇伪装

HTTP协议中的请求和响应不会对通信方进行确认。在HTTP协议通信时，由于不存在确认通信方的处理步骤，任何人都可以发起请求。另外，服务器只要接收到请求，不管对方是谁都会返回一个响应（但也仅限于发送端的IP地址和端口号没有被Web服务器设定限制访问的前提下）

HTTP协议无法验证通信方身份，任何人都可以伪造虚假服务器欺骗用户，实现“钓鱼欺诈”，用户无法察觉。

反观HTTPS协议，它比HTTP协议相比多了以下优势（下文会详细介绍）：

- 数据隐私性：内容经过对称加密，每个连接生成一个唯一的加密密钥
- 数据完整性：内容传输经过完整性校验
- 身份认证：第三方无法伪造服务端（客户端）身份