

前端学科面试宝典

H5 移动 web 开发.....	16
1、H5 的新特性有哪些？C3 的新特性有哪些？（必会）	16
2、如何使一个盒子水平垂直居中？（必会）	17
3、如何实现双飞翼（圣杯）布局？（必会）	20
4、CSS 的盒模型？（必会）	22
5、CSS 中选择器的优先级以及 CSS 权重如何计算？（必会）	23
6、列举 5 个以上的 H5input 元素 type 属性值？（必会）	24
7、CSS 中哪些属性可继承，哪些不可以？（必会）	25
8、CSS 单位中 px、em 和 rem 的区别？（必会）	25
9、rem 适配方法如何计算 HTML 根字号及适配方案？（必会）	26
10、display: none 与 visibility: hidden 的区别？（必会）	26
11、position 的值有哪些，分别有哪些作用？（必会）	27
12、为什么会出现浮动？浮动元素会引起什么问题？如何清除浮；动？（必会）	27
13、简述弹性盒子 flex 布局及 rem 布局？（必会）	28
14、如何解决 margin “塌陷”？（必会）	31
15、::before 和 ::after 中双冒号和单冒号有什么区别、作用？（必会）	31
16、CSS3 新增伪类，以及伪元素？（必会）	32
17、Bootstrap 栅格系统的工作原理？（必会）	32
18、BFC 是什么？（高薪常问）	34
19、什么是渐进增强和优雅降级？它们有什么不同？（了解）	34
20、iframe 有哪些优缺点？（了解）	35
21、使用 CSS 怎么让 Chrome 支持小于 12px 的文字比如 10px？（了解）	35
JavaScript 基础.....	36
1、 JavaScript 的基本类型有哪些？引用类型有哪些？null 和 undefined 的区别？（必会）	36
3、简述创建函数的几种方式？（必会）	37
4、Javascript 创建对象的几种方式？（必会）	38
5、请指出 JavaScript 宿主对象和原生对象的区别？（必会）	39
6、JavaScript 内置的常用对象有哪些？并列举该对象常用的方法？（必会）	40
7、=== 和 == 的区别？（必会）	43
8、null, undefined 的区别（必会）	43
9、JavaScript 中什么情况下会返回 undefined 值？（必会）	44
10、如何区分数组和对象？（必会）	44
11、多维数组降维的几种方法（必会）	44
12、怎么判断两个对象相等？（必会）	46
13、列举三种强制类型转换和两种隐式类型转换？（必会）	47
14、JavaScript 中怎么获取当前日期的月份？（必会）	47
15、什么是类数组（伪数组），如何将其转化为真实的数组？（必会）	48
16、如何遍历对象的属性？（必会）	48
17、如何使用原生 JavaScript 给一个按钮绑定两个 onclick 事件？（必会）	50
18、JavaScript 中的作用域、预解析与变量声明提升？（必会）	50
19、变量提升与函数提升的区别？（必会）	51
20、什么是作用域链？（必会）	52

21、如何延长作用域链？（必会）	52
22、判断一个值是什么类型有哪些方法？（必会）	52
23、如何实现数组的随机排序？（必会）	53
24、src 和 href 的区别是？（了解）	53
WebAPI.....	54
1、什么是 dom？（必会）	54
2、dom 节点的 Attribute 和 Property 有何区别？（必会）	54
3、dom 结构操作怎样添加、移除、移动、复制、创建和查找节点？（必会）	54
4、dom 事件模型？（必会）	55
5、什么是事件冒泡，它是如何工作的？如何阻止事件冒泡、默认行为？（必会）	55
6、JavaScript 动画和 CSS3 动画有什么区别？（必会）	56
7、event 对象的常见应用？（必会）	57
8、通用事件绑定/ 编写一个通用的事件监听函数？（必会）	57
9、DOM 和 BOM 的区别（必会）	57
10、事件三要素（必会）	58
11、事件执行过程（必会）	58
12、获取元素位置（必会）	58
13、封装运动函数（必会）	59
14、绑定事件和解除事件的区别（必会）	60
15、谈谈事件委托的理解？（必会）	60
16、JavaScript 中的定时器有哪些？他们的区别及用法是什么？（必会）	61
17、比较 attachEvent 和 addEventListener？（必会）	61
18、document.write 和 innerHTML 的区别？（必会）	61
19、什么是 window 对象？什么是 document 对象？（必会）	61
20、Js 拖动的原理？（必会）	63
21、描述浏览器的渲染过程，DOM 树和渲染树的区别（必会）	64
22、如何最小化重绘(repaint)和回流(reflow)（必会）	64
23、Js 延迟加载的方式有哪些？（了解）	65
24、IE 与标准事件模型有哪些差别？（了解）	67
JavaScript 高级.....	67
1、typeof 和 instanceof 区别（必会）	67
2、js 使用 typeof 能得到的哪些类型？（必会）	67
3、解释一下什么是回调函数，并提供一个简单的例子？（必会）	68
4、什么是闭包？（必会）	68
5、什么是内存泄漏（必会）	69
6、哪些操作会造成内存泄漏？（必会）	69
7、JS 内存泄漏的解决方式（必会）	69
8、说说你对原型（prototype）理解（必会）	71
9、介绍下原型链（解决的是继承问题吗）（必会）	71
10、常见的 js 中的继承方法有哪些（必会）	71
11、介绍 this 各种情况（必会）	72
12、数组中的 forEach 和 map 的区别？（必会）	73
13、for in 和 for of 的区别（必会）	73

14、call 和 apply, bind 的区别（必会）	73
15、EventLoop 事件循环机制（必会）	74
16、js 防抖和节流（必会）	75
17、new 操作符具体干了什么呢？（必会）	76
18、用 JavaScript 实现冒泡排序。数据为 23、45、18、37、92、13、24（必会）	76
19、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（必会）	77
20、已知数组 var stringArray = [“This”，“is”，“Baidu”，“Campus”]，Alert 出” This is Baidu Campus”（必会）	77
21、已知有字符串 foo=” get-element-by-id”，写一个 function 将其转化成驼峰表示法” getElementById”（必会）	77
22、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}（必会）	78
23、输出今天的日期，以 YYYY-MM-DD 的方式，比如今天是 2014 年 9 月 26 日，则输出 2014-09-26（必会）	78
24、把两个数组合并，并删除第二个元素。（必会）	78
25、写一个 function，清除字符串前后的空格。（兼容所有浏览器）（必会）	79
26、截取字符串 abcdefg 的 efg（必会）	79
27、判断一个字符串中出现次数最多的字符，统计这个次数（必会）	79
28、将数字 12345678 转化成 RMB 形式 如： 12,345,678（必会）	79
29、split（）和 join（）的区别？（必会）	80
30、JavaScript 中如何对一个对象进行深度 clone？（必会）	80
31、js 数组去重，能用几种方法实现（必会）	81
32、谈谈你对 Javascript 垃圾回收机制的理解？（高薪常问）	82
33、class 和普通构造函数有何区别？（高薪常问）	82
34、JS 里垃圾回收机制是什么，常用的是哪种，怎么处理的？（高薪常问）	83
35、什么是进程、什么是线程、它们之间是什么关系（了解）	83
36、什么是任务队列？（了解）	84
37、栈和队列的区别？（了解）	84
38、栈和堆的区别？（了解）	84
jQuery	85
1、jQuery 的\$(document).ready(function () {}), \$(function () {})与原生 JS 的 window.onload 有什么不同？（必会）	85
2、jQuery 和 Zepto 的区别？各自的使用场景？（必会）	85
3、你是如何使用 jQuery 中的 ajax 的？（必会）	86
4、jQuery 的常用的方法增、删、复制、改、查（必会）	86
5、jQuery 中\$.get()提交和\$.post()提交的区别？（必会）	87
6、简单的讲叙一下 jQuery 是怎么处理事件的，你用过哪些事件？（必会）	87
7、你使用过 jQuery 中的动画吗，是怎样用的？（必会）	88
8、你在 jQuery 中使用过哪些插入节点的方法，它们的区别是什么？（必会）	88
9、jQuery 中如何来获取或设置属性？（必会）	88

10、jQuery 如何设置和获取 HTML、文本和值? (必会)	88
11、有哪些查询节点的选择器? (必会)	88
12、jQuery 中的 hover() 和 toggle() 有什么区别? (必会)	89
13、jQuery 中 detach() 和 remove() 方法的区别是什么? (必会)	89
14、\$(this) 和 this 关键字在 jQuery 中有何不同? (必会)	89
15、jQuery 中 attr() 和 prop() 的区别 (必会)	89
16、jQuery 库中的 \$() 是什么? (必会)	90
17、jQuery 的属性拷贝(extend)的实现原理是什么, 如何实现深浅拷贝? (高薪常问)	90
18、jQuery 的实现原理? (高薪常问)	90
数据可视化.....	91
1、echarts 的基本用法 (必会)	91
2、如何使用 echarts (必会)	91
3、echarts 如何画图? (必会)	91
4、echarts 绘制条形图 (必会)	91
5、切换其他组件统计图时, 出现卡顿问题如何解决 (必会)	92
6、echarts 图表自适应 div resize 问题 (必会)	92
7、echarts 在 vue 中怎么引用? (必会)	92
8、echarts 支持哪些图标? (了解)	93
Ajax/计算机网络相关.....	93
1、什么是 Ajax, Ajax 的原理, Ajax 都有哪些优点和缺点? (必会)	93
2、常见的 HTTP 状态码以及代表的意义 (必会)	94
3、请介绍一下 XMLHttpRequest 对象及常用方法和属性 (必会)	94
4、Ajax 的实现流程是怎样的? (必会)	95
5、Ajax 接收到的数据类型有哪些, 数据如何处理? (必会)	96
6、请解释一下 JavaScript 的同源策略 (必会)	97
7、为什么会有跨域的问题出现, 如何解决跨域问题 (必会)	97
8、Get 和 Post 的区别以及使用场景 (必会)	98
9、解释 jsonp 的原理 (必会)	98
10、封装好的 Ajax 里的常见参数及其代表的含义 (必会)	98
11、jQuery 中 \$.ajax 与 fetch、axios 有什么区别? (必会)	99
12、Ajax 注意事项及适用和不适用场景 (必会)	100
13、HTTP 与 HTTPS 的区别 (必会)	100
14、localStorage、sessionStorage、cookie 的区别 (必会)	100
15、简述 web 前端 Cookie 机制, 并结合该机制说明会话保持原理? (必会)	101
16、一个页面从输入 URL 到页面加载显示完成, 这个过程中都发生了什么 (高薪常问)	102
17、你知道的 HTTP 请求方式有几种 (高薪常问)	102
18、什么是 TCP 连接的三次握手 (高薪常问)	103
19、为什么 TCP 连接需要三次握手四次挥手 (高薪常问)	104
20、TCP 与 UDP 的区别有哪些 (高薪常问)	104
21、介绍一下 websocket (高薪常问)	104
22、拆解一下 URL 的各个部分, 分别是什么意思 (高薪常问)	105
23、HTTP 缓存机制 (高薪常问)	105

ES6.....	106
1、ES5 和 ES6 的区别，说几个 ES6 的新增方法（必会）	106
2、ES6 的继承和 ES5 的继承有什么区别（必会）	109
3、var、let、const 之间的区别（必会）	109
4、class、extends 是什么，有什么作用（必会）	109
5、module、export、import 有什么作用（必会）	109
6、使用箭头函数应注意什么/箭头函数和普通函数的区别（必会）	110
7、ES6 的模板字符串有哪些新特性？并实现一个类模板字符串的功能（必会）	110
8、介绍下 Set、Map 的区别（必会）	110
9、setTimeout、Promise、Async/Await 的区别（必会）	111
10、Promise 有几种状态，什么时候会进入 catch？（必会）	111
11、ES6 怎么写 class，为何会出现 class（必会）	111
12、如何获取多个 Promise 最后整体结果？（必会）	111
13、ES6 如何转化为 ES5，为什么要转化（必会）	112
14、日常前端代码开发中，有哪些值得用 ES6 去改进的编程优化或者规范（必会）	114
15、ES6 和 node 的 commonjs 模块化规范的区别（高薪常问）	114
16、Promise 中 reject 和 catch 处理上有什么区别（高薪常问）	114
17、理解 async/await 以及相对 Generator 的优势	114
18、手写一个 Promise（高薪常问）	115
19、Promise 如何封装一个 Ajax（高薪常问）	116
20、下面的输出结果是多少（高薪常问）	116
21、以下代码依次输出的内容是（高薪常问）	117
22、分析下列程序代码，得出运行结果，解释其原因（高薪常问）	117
23、分析下列程序代码，得出运行结果，解释其原因（高薪常问）	117
24、使用结构赋值，实现两个变量的值的交换（高薪常问）	118
25、说一下 ES6 的导入导出模块（高薪常问）	118
git.....	119
1、git 的基本使用方法（必会）	119
2、git 工作流程（必会）	119
3、我们如何使用 git 和开源的码云或 github 上面的远端仓库的项目进行工作呢（必会）	120
4、git, github, gitlab 三者之间的联系以及区别（必会）	123
5、github 和码云的区别（必会）	123
6、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（必会）	124
7、如果本次提交误操作，如何撤销（必会）	124
8、git 修改提交的历史信息（必会）	124
9、如何删除 github 和 gitlab 上的文件夹（必会）	125
10、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（必会）	125
11、git 跟 svn 有什么区别（必会）	125
12、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（必会）	126

13、git fetch 和 git merge 和 git pull 的区别（必会）	126
14、如何把本地仓库的内容推向一个空的远程仓库（高薪常问）	126
大事件项目 PC 端.....	127
1、开发背景.....	127
1.1 项目介绍.....	127
2、系统架构.....	127
2.1 关键技术.....	127
2.2 API 文档.....	128
2.3 人员配置.....	128
2.4 开发流程.....	128
3、登录模块.....	129
3.1 业务实现思路.....	129
3.2 技术亮点.....	129
4、首页模块.....	129
4.1 业务实现思路.....	129
4.2 技术亮点.....	129
5、文章类别管理.....	130
5.1 业务实现思路.....	130
5.2 技术亮点.....	130
6、文章列表管理.....	130
6.1 业务实现思路.....	130
6.2 技术亮点.....	130
7、前台页面文章获取.....	130
7.1 业务实现思路.....	130
7.2 技术亮点.....	130
8、项目介绍话术.....	130
Node.js.....	131
1、对 Node.js 有没有了解，它有什么特点，适合做什么业务（必会）	131
2、Node 和前端项目怎么解决跨域的（必会）	131
3、Node 的优点是什么？缺点是什么（必会）	132
4、commonJS 中的 require/exports 和 ES6 中 import/export 的区别是什么（必会）	132
5、简述同步和异步的区别，如何避免回调地狱，Node 的异步问题是如何解决的（必会）	132
6、dependencies 和 devDependencies 两者区别（必会）	133
7、什么是前后端分离的项目？什么是 JS 渲染的项目，前端渲染和后端渲染的区别（高薪常问）	133
8、mysql 和 mongodb 有什么区别（高薪常问）	134
9、什么是中间件（高薪常问）	135
10、为什么要进行模块化（高薪常问）	135
11、谈谈你对 AMD 和 CMD 的理解（高薪常问）	135
12、Node 怎么跟 MongoDB 建立连接（高薪常问）	136
13、请介绍一下 require 的模块加载机制（高薪常问）	136
14、express 中如何获取路由的参数（高薪常问）	136

Webpack.....	136
1、什么是 Webpack（必会）	136
2、Webpack 的优点是什么？（必会）	136
3、Webpack 的构建流程是什么?从读取配置到输出文件这个过程尽量说全（必会）	137
4、说一下 Webpack 的热更新原理(必会)	137
5、Webpack 与 grunt、gulp 的不同？（必会）	137
6、有哪些常见的 Loader？他们是解决什么问题的？（必会）	138
7、Loader 和 Plugin 的不同？（必会）	138
8、如何利用 Webpack 来优化前端性能（高薪常问）	138
9、是否写过 Loader 和 Plugin？描述一下编写 loader 或 plugin 的思路？（高薪常问）	139
10、使用 Webpack 开发时，你用过的哪些可以提高效率的插件？（高薪常问）	139
11、什么是长缓存？在 Webpack 中如何做到长缓存优化？（高薪常问）	139
12、如何提高 Webpack 的构建速度？（高薪常问）	140
13、怎么实现 Webpack 的按需加载？什么是神奇注释?(高薪常问)	140
Vue 面试题.....	140
1、Vue 的最大的优势是什么？（必会）	140
2、Vue 和 jQuery 两者之间的区别是什么？	141
3、mvvm 和 mvc 区别是什么？哪些场景适合？（必会）	141
4、Vue 数据双向绑定的原理是什么？（必会）	142
5、Object.defineProperty 和 Proxy 的区别（必会）	142
6、Vue 生命周期总共分为几个阶段？（必会）	143
7、第一次加载页面会触发哪几个钩子函数？（必会）	144
8、请说下封装 Vue 组件的过程？（必会）	144
9、Vue 组件如何进行传值的？（必会）	144
10、组件中写 name 选项有什么作用？（必会）	144
11、Vue 组件 data 为什么必须是函数（必会）	144
12、讲一下组件的命名规范（必会）	145
13、怎么在组件中监听路由参数的变化？（必会）	145
14、怎么捕获 Vue 组件的错误信息？（必会）	145
15、Vue 组件里的定时器要怎么销毁？（必会）	145
16、Vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？（必会）	146
18、Vue 该如何实现组件缓存？（必会）	146
19、跟 keep-alive 有关的生命周期是哪些？（必会）	147
20、Vue 常用的修饰符都有哪些？（必会）	147
21、Vue 常用的指令都有哪些？并且说明其作用（必会）	147
22、自定义指令(v-check、v-focus)的方法有哪些?它有哪些钩子函数?还有哪些钩子函数参数?（必会）	148
24、v-show 和 v-if 指令的共同点和不同点?（必会）	148
25、为什么避免 v-if 和 v-for 用在一起（必会）	148
26、watch、methods 和 computed 的区别?（必会）	149
27、怎么在 watch 监听开始之后立即被调用？（必会）	149
27、watch 怎么深度监听对象变化？（必会）	149

29、computed 中的属性名和 data 中的属性名可以相同吗？（必会）	150
30、什么是 Vue 的计算属性（必会）	150
31、Vue 中 key 值的作用是什么？（必会）	150
32、Vue-loader 是什么？使用它的用途有哪些？（必会）	151
33、Vue 中怎么自定义过滤器（必会）	151
34、你是怎么认识 Vuex 的？（必会）	152
35、Vuex 的 5 个核心属性是什么？（必会）	152
36、Vuex 的出现解决了什么问题？（必会）	153
38、Vuex 的 Mutation 和 Action 之间的区别是什么？（必会）	153
39、Vue-Router 是干什么的，原理是什么？（必会）	153
40、路由之间是怎么跳转的？有哪些方式？（必会）	154
41、Vue-Router 怎么配置路由（必会）	154
42、Vue-Router 有哪几种路由守卫？（必会）	154
43、Vue-Router 的钩子函数都有哪些？（必会）	155
45、怎么定义 Vue-Router 的动态路由？怎么获取传过来的动态参数？	156
46、query 和 params 之间的区别是什么？（必会）	156
47、\$route 和 \$router 的区别是什么？（必会）	156
48、active-class 属于哪个组件中的属性？该如何使用？	156
49、Vue 的路由实现模式：hash 模式和 history 模式（必会）	157
52、Vue 怎么实现跨域（必会）	158
54、你对 Vue.js 的 template 编译的理解？（必会）	159
55、Vue 渲染模板时怎么保留模板中的 HTML 注释呢？（必会）	159
56、Vue2.0 兼容 IE 哪个版本以上吗？（必会）	159
57、Vue 如何去除 URL 中的#（必会）	159
58、说一下你在 Vue 中踩过的坑（必会）	160
59、在 Vue 中使用插件的步骤（必会）	160
60、Vue 项目优化的解决方案都有哪些？（必会）	160
63、你知道 style 上加 scoped 属性的原理吗？（必会）	161
64、说说你对 SPA 单页面的理解，它的优缺点分别是什么？（必会）	161
65、怎样理解 Vue 的单向数据流？（必会）	162
66、VNode 是什么？什么是虚拟 DOM？（高薪常问）	162
67、Vue 中如何实现一个虚拟 DOM？说说你的思路（高薪常问）	163
68、Vue 中操作 data 中数组的方法中哪些可以触发视图更新，哪些不可以，不可以的话有什么解决办法？（高薪常问）	163
69、Vue 中怎么重置 data？（高薪常问）	163
70、如何对 Vue 首屏加载实现优化？（高薪常问）	164
71、Vue 的 nextTick 的原理是什么？（高薪常问）	164
72、在 Vue 实例中编写生命周期 hook 或其他 option/propertie 时，为什么不使用箭头函数？（高薪常问）	164
73、is 这个特性你有用过吗？主要用在哪些方面？（高薪常问）	165
74、scss 是什么？在 Vue.cli 中的安装使用步骤是？有哪几大特性？（高薪常问）	165
75、请详细介绍一些 package.json 中的配置的作用（了解）	165
人力资源中台项目	166

1、开发背景.....	166
1.1 项目介绍.....	166
1.2 Vue 简介.....	166
1.3 Vue 技术发展历程.....	166
1.4 SPA 单页应用与多页应用.....	166
2、系统架构.....	168
2.1 Vue-cli 脚手架工具.....	168
2.2 Element-ui 框架.....	168
2.3 Vue-cli 项目目录结构.....	169
2.4 Vue-cli 项目运行机制说明.....	170
2.5 开发规范.....	170
3、技术架构.....	171
4、开发环境与技术.....	171
4.1 nodejs 环境.....	171
4.2 git 版本控制.....	171
4.3 npm 淘宝镜像.....	171
4.4 VsCode 编辑器.....	171
4.4 关键技术.....	172
4.6 类似组件库.....	172
4.7 API 文档.....	172
4.8 人员配置.....	173
4.9 开发流程.....	174
5、项目功能架构.....	175
6、登录模块.....	175
6.1 业务实现思路.....	175
6.2 技术亮点.....	175
7、主页模块.....	176
7.1 业务实现思路.....	176
7.2 技术亮点.....	176
8、组织架构.....	176
8.1 业务实现思路.....	176
8.2 技术亮点.....	176
9、角色管理.....	176
9.1 业务实现思路.....	176
9.2 技术亮点.....	177
10、员工管理.....	177
10.1 业务实现思路.....	177
10.2 技术亮点.....	177
11、员工详情.....	177
11.1 业务实现思路.....	177
11.2 技术亮点.....	177
12、权限点管理.....	178
12.1 业务实现思路.....	178
12.2 技术亮点.....	178

13、RBAC 权限设计及应用.....	178
13.1 业务实现思路.....	178
13.2 技术亮点.....	178
14、首页功能实现.....	178
14.1 业务实现思路.....	178
14.2 技术亮点.....	179
15、全屏-多语言.....	179
15.1 业务实现思路.....	179
15.2 技术亮点.....	179
16、打包优化发布.....	179
16.1 业务实现思路.....	179
16.2 技术亮点.....	179
17、项目介绍话术.....	179
18、开发中遇到的问题.....	180
小程序.....	180
1、如何获得用户的授权信息？（必会）.....	180
2、数据绑定如何实现？（必会）.....	180
3、列表渲染如何实现？（必会）.....	180
4、事件及事件绑定是什么？（必会）.....	180
5、页面跳转的方式有哪些？（必会）.....	181
6、tabBar 配置参数有哪些？（必会）.....	181
7、页面生命周期包含那几个？（必会）.....	181
8、转发分享如何实现？（必会）.....	182
9、如何获取地理位置？（必会）.....	182
10、如何封装自定义组件？（必会）.....	182
11、webview 是什么？（必会）.....	183
12、简单描述下微信小程序的相关文件类型？（必会）.....	183
13、小程序有哪些参数传值的方法？（必会）.....	183
14、简述微信小程序原理？（必会）.....	183
15、小程序的双向绑定和 vue 哪里不一样？（必会）.....	184
16、分析下微信小程序的优劣势？（必会）.....	185
17、bindtap 和 catchtap 的区别是什么？（必会）.....	185
18、简述下 wx.navigateTo(), wx.redirectTo(), wx.switchTab(), wx.navigateBack(), wx.reLaunch() 的区别？（必会）.....	185
19、小程序尺寸单位 rpx？（必会）.....	186
20、小程序选择器有哪些？（必会）.....	187
21、小程序常用组件？（必会）.....	187
22、微信小程序长按识别二维码（必会）.....	187
23、如何封装微信小程序的数据请求(http-promise)？（高薪常问）.....	187
24、小程序申请微信支付？（了解）.....	188
25、客服电话？（了解）.....	188
26、小程序插槽的使用 slot？（了解）.....	188
28、如何分包加载？分包加载的优势在哪？（了解）.....	190
29、哪些方法可以用来提高微信小程序的应用速度？（了解）.....	190

30、webview 中的页面怎么跳回小程序中? (了解)	190
31、小程序如何实现下拉刷新? (了解)	190
32、小程序调用后台接口遇到哪些问题? (了解)	191
优购商城(小程序项目)	191
1、开发背景	191
1.1 项目介绍	191
1.2 小程序简介	191
1.3 小程序技术发展史	191
1.4 小程序与普通网页开发的区别	192
1.5 为什么要开发小程序	192
2、系统架构	193
2.1 传统原生 APP	193
2.2 微信运行环境	193
2.3 微信小程序运行环境	194
2.4 mina 框架	194
2.4.1 小程序文件结构和传统 web 对比	194
2.4.2 基本的目录结构	195
2.5 uni-app 框架	195
2.5.1 开发规范	195
2.5.2 目录结构	196
2.5.3 uni-app 组件的编译图解	196
2.6 其他框架	196
3、技术架构	197
4、开发环境与技术	197
4.1 关键技术	197
4.2 API 文档	198
4.3 人员配置	198
4.4 开发流程	198
4.4.1 注册账号	199
4.4.2 获取 APPID	199
4.4.3 开发工具	200
5、项目架构	201
6、首页展示	202
6.1 业务实现思路	202
6.2 技术亮点	202
7、商品分类	202
7.1 业务实现思路	202
7.2 技术亮点	202
8、商品列表	203
8.1 业务实现思路	203
8.2 技术亮点	203
9、商品详情	203
9.1 业务实现思路	203
9.2 技术亮点	203

10、搜索页面.....	203
10.1 业务实现思路.....	203
10.2 技术亮点.....	203
11、购物车.....	204
11.1 业务实现思路.....	204
11.2 技术亮点.....	204
12.1 业务实现思路.....	204
12.2 技术亮点.....	204
13.1 业务实现思路.....	204
13.2 技术亮点.....	204
17、项目介绍.....	205
18、开发中遇到的问题.....	205
18.1 域名必须是 HTTPS.....	205
18.2 tabbar 在切换时页面数据无法刷新.....	205
18.3 小程序 image 高度自适应及裁剪问题.....	205
18.4 小程序中 canvas 的图片不支持 base64 格式.....	205
18.5 wx.setStorageSync 和 wx.getStorageSync 报错问题.....	206
18.6 代码审核和发布.....	206
18.7 小程序微信认证.....	206
18.8 图片本地资源名称，尽量使用小写命名.....	206
React.....	206
1、谈谈你对 React 的了解（必会）.....	206
2、什么是 JSX？为什么浏览器无法读取 JSX？（必会）.....	207
3、shouldComponentUpdate 是做什么的？（必会）.....	207
4、React 性能优化是哪个周期函数？（必会）.....	207
5、React 中 keys 的作用是什么？（必会）.....	207
6、React 中 refs 的作用是什么？（必会）.....	207
7、请列举 React 中定义组件的方法？（必会）.....	208
8、调用 setState 之后发生了什么？（必会）.....	208
9、你怎么理解 redux 的 state 的？（必会）.....	208
10、除了在构造函数中绑定 this，还有其它方式吗？（必会）.....	208
11、（在构造函数中）调用 super(props) 的目的是什么？（必会）.....	208
12、简述 flux 思想？（必会）.....	209
13、事件在 React 中的处理方式？（必会）.....	209
14、列出 Redux 的核心方法？（必会）.....	209
15、（组件的）状态(state)和属性(props)之间有何不同？（必会）.....	209
16、何为受控组件(controlledcomponent)？（必会）.....	209
18、何为高阶组件(higherordercomponent)？（必会）.....	210
19、React 中组件如何进行数据传值？（必会）.....	210
20、解释 Reducer 的作用（必会）.....	210
21、redux 有什么缺点（必会）.....	210
22、了解 redux 么，说一下 redux（必会）.....	210
23、Redux 的工作流程.....	211
24、vue 和 React 的区别（必会）.....	211

25、React 生命周期函数有哪些？（必会）	211
26、运行阶段生命周期调用顺序？（必会）	212
27、React 中 component 和 pureComponent 区别是什么？（必会）	212
28、什么是无状态组件，与有状态组件的区别？（必会）	212
29、调用 render 时，DOM 一定会更新吗，为什么？（必会）	212
30、在哪些生命周期中可以修改组件的 state？（必会）	213
31、connect() 前两个参数是什么？（必会）	213
32、React-router 的原理（高薪常问）	213
33、React 的 diff 原理（高薪常问）	213
34、为什么建议传递给 setState 的参数是一个 callback 而不是一个对象（高薪常问）	214
35、redux 中间件原理（高薪常问）	214
36、React 性能优化的方案（高薪常问）	214
37、为什么虚拟 DOM 会提高性能？说下他的原理（高薪常问）	214
38、setState 何时同步何时异步？（高薪常问）	214
前端性能优化	215
1、如何进行前端性能优化？（必会）	215
2、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。（必会）	216
兼容问题（必会）	216
1、图片加 a 标签在 IE9 中会有边框（必会）	216
2、rgba 不支持 IE8（必会）	216
3、display:inline-block 在 IE6/7 不支持（必会）	217
4、不同浏览器的标签默认的外补丁和内补丁不同（必会）	217
5、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（必会）	217
6、设置较小高度标签（一般小于 10px），在 IE6，IE7 中高度超出自己设置高度（必会）	217
opacity: 0.5;filter: alpha(opacity = 50);filter: progid:DXImageTransform.Microsoft.Alpha(style = 0, opacity = 50);	217
8、IE6 背景闪烁的问题（必会）	217
9、event 事件问题：（必会）	218
10、DOM 节点相关的问题（必会）	218
11、设置监听事件：（必会）	218
12、阻止事件冒泡：（必会）	219
13、阻止默认事件：（必会）	219
14、IOS 移动端 click 事件 300ms 的延迟响应（必会）	219
15、一些情况下对非可点击元素如(label, span) 监听 click 事件，ios 下不会触发（必会）	220
16、三星手机遮罩层下的 input、select、a 等元素可以被点击和 focus(点击穿透)（必会）	220
17 Input 的 placeholder 会出现文本位置偏上的情况（必会）	220
计算机相关术语	220
1、关于计算机相关术语的介绍（高薪常问）	220

2、http 超文本传输协议（高薪常问）	221
3、计算机网络的分层体系结构（高薪常问）	223
4、计算机存储器相关知识（高薪常问）	223
5、浏览器（高薪常问）	224
6、服务器（高薪常问）	225
7、经典编程算法（高薪常问）	226
8、经典排序算法（高薪常问）	226
9、黑盒、白盒、灰盒测试（高薪常问）	226
10、二叉排序树（高薪常问）	227

H5 移动 web 开发

1、H5 的新特性有哪些？C3 的新特性有哪些？（必会）

H5 新特性

- 1、拖拽释放(Drap and drop) API ondrop
拖放是一种常见的特性，即抓取对象以后拖到另一个位置
在 HTML5 中，拖放是标准的一部分，任何元素都能够拖放
- 2、自定义属性 data-id
- 3、语义化更好的内容标签(header,nav,footer ,aside, article, section)
- 4、音频 ,视频(audio, video) 如果浏览器不支持自动播放怎么办？ 在属性中添加 autoplay(谷歌浏览器不支持音频自动播放，但是视频支持静音自动播放)
- 5、画布 Canvas
 - 5.1) getContext() 方法返回一个用于在画布上绘图的环境
Canvas.getContext(contextID) 参数 contextID 指定了您想要在画布上绘制的类型。当前唯一的合法值是 “2d”，它指定了二维绘图，并且导致这个方法返回一个环境对象，该对象导出一个二维绘图 API
 - 5.2) cxt.stroke() 绘制线条
 - 5.3) canvas 和 image 在处理图片的时候有什么区别？
image 是通过对象的形式描述图片的,canvas 通过专门的 API 将图片绘制在画布上.
- 6、 地理(Geolocation) API 其实 Geolocation 就是用来获取到当前设备的经纬度（位置）
- 7、 本地离线存储 localStorage 用于长久保存整个网站的数据，保存的数据没有过期时间，直到手动去删除
- 8、 sessionStorage 该数据对象临时保存同一窗口(或标签页)的数据，在关闭窗口或标签页之后将会删除这些数据。
- 9、 表单控件 calendar , date , time , email , url , search , tel , file , number
- 10、新的技术 webworker, websocket , Geolocation

CSS3 新特性

- 1、颜色：新增 RGBA , HSLA 模式
- 2、文字阴影(text-shadow)
- 3、边框：圆角(border-radius) 边框阴影 : box-shadow
- 4、盒子模型: box-sizing
- 5、背景:background-size background-origin background-clip
- 6、渐变: linear-gradient , radial-gradient
- 7、过渡 : transition 可实现属性的渐变

- 8、自定义动画 `animate @keyfrom`
- 9、媒体查询 多栏布局 `@media screen and (width:800px) {...}`
- 10、`border-image` 图片边框
- 11、2D 转换/3D 转换;`transform: translate(x,y) rotate(x,y) skew(x,y) scale(x,y)`
- 12、字体图标 `iconfont/iconmoon`
- 13、弹性布局 `flex`

2、如何使一个盒子水平垂直居中？（必会）

方法一：利用定位（常用方法, 推荐）

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .parent {
      width: 500px;
      height: 500px;
      border: 1px solid #000;
      position: relative;
    }

    .child {
      width: 100px;
      height: 100px;
      border: 1px solid #999;
      position: absolute;
      top: 50%;
      left: 50%;
      margin-top: -50px;
      margin-left: -50px;
    }
  </style>
</head>
<body>
  <div class="parent">
    <div class="child">我是子元素</div>
  </div>
</body>
</html>
```

方法二：利用 `margin:auto;`

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .parent {
      width: 500px;
```

```

        height: 500px;
        border: 1px solid #000;
        position: relative;
    }

    .child {
        width: 100px;
        height: 100px;
        border: 1px solid #999;
        position: absolute;
        margin: auto;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
    }
</style>
</head>
<body>
    <div class="parent">
        <div class="child">我是子元素</div>
    </div>
</body>
</html>

```

方法三：利用 display:table-cell

```

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .parent {
            width: 500px;
            height: 500px;
            border: 1px solid #000;
            display: table-cell;
            vertical-align: middle;
            text-align: center;
        }

        .child {
            width: 100px;
            height: 100px;
            border: 1px solid #999;
            display: inline-block;
        }
    </style>
</head>
<body>
    <div class="parent">
        <div class="child">我是子元素</div>
    </div>
</body>
</html>

```

方法四：利用 display: flex;设置垂直水平都居中

```

<!DOCTYPE html>

```

```

<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .parent {
      width: 500px;
      height: 500px;
      border: 1px solid #000;
      display: flex;
      justify-content: center;
      align-items: center;
    }

    .child {
      width: 100px;
      height: 100px;
      border: 1px solid #999;
    }
  </style>
</head>
<body>
  <div class="parent">
    <div class="child">我是子元素</div>
  </div>
</body>
</html>

```

方法五：计算父盒子与子盒子的空间距离(这跟方法一是一个道理)

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .parent {
      width: 500px;
      height: 500px;
      border: 1px solid #000;
    }

    .child {
      width: 100px;
      height: 100px;
      border: 1px solid #999;
      margin-top: 200px;
      margin-left: 200px;
    }
  </style>
</head>
<body>
  <div class="parent">
    <div class="child">我是子元素</div>
  </div>
</body>

```

```

    </html>
方法六：利用 transform
    <!DOCTYPE html>
    <html lang="en">

    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            .parent {
                width: 500px;
                height: 500px;
                border: 1px solid #000;
                position: relative;
            }

            .child {
                width: 100px;
                height: 100px;
                border: 1px solid #999;
                position: absolute;
                top: 50%;
                left: 50%;
                transform: translate(-50%, -50%);
            }
        </style>
    </head>
    <body>
        <div class="parent">
            <div class="child">我是子元素</div>
        </div>
    </body>
</html>

```

3、如何实现双飞翼（圣杯）布局？（必会）

1、利用定位实现两侧固定中间自适应

- 1.1) 父盒子设置左右 padding 值
- 1.2) 给左右盒子的 width 设置父盒子的 padding 值, 然后分别定位到 padding 处.
- 1.3) 中间盒子自适应

具体 CSS 代码:

```

<style>
    .father {
        height: 400px;
        background-color: pink;
        position: relative;
        padding: 0 200px;
    }
    .left,.right {
        width: 200px;
        height: 300px;
        background-color: yellow;
    }

```

```

        position: absolute;
        top: 0;
    }
    .left {
        left: 0;
    }
    .right {
        right: 0;
    }
    .center {
        background-color: blue;
        height: 350px;
    }

```

</style>

html 结构

```

<div class="father">
    <div class="left"></div>
    <div class="center"></div>
    <div class="right"></div>
</div>

```

2、利用 flex 布局实现两侧固定中间自适应

2.1) 父盒子设置 display:flex;

2.2) 左右盒子设置固定宽高

2.3) 中间盒子设置 flex:1 ;

<style>

```

    .father {
        height: 400px;
        background-color: pink;
        display: flex;
    }
    .left {
        width: 200px;
        height: 300px;
        background-color: yellow;
    }
    .right {
        width: 200px;
        height: 300px;
        background-color: yellow;
    }
    .center {
        flex: 1;
        background-color: blue;
    }

```

</style>

html 结构

```

<div class="father">
    <div class="left"></div>
    <div class="center"></div>
    <div class="right"></div>

```

</div>

3、利用 bfc 块级格式化上下文，实现两侧固定中间自适应

3.1) 左右固定宽高，进行浮动

3.2) 中间 overflow: hidden;

<style>

```
.father {
  height: 500px;
  background-color: pink;
}
.left {
  float: left;
  width: 200px;
  height: 400px;
  background-color: blue;
}
.right {
  float: right;
  width: 200px;
  height: 400px;
  background-color: blue;
}
.center {
  height: 450px;
  background-color: green;
  overflow: hidden;
}
```

</style>

html 结构

<!-- 注意:left 和 right 必须放在 center 前面 -->

<div class="father">

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

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

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

</div>

4、CSS 的盒模型？（必会）

盒子模型（Box Model）可以用来对元素进行布局，包括内边距，边框，外边距，和实际内容这几个部分

盒子模型分为两种：

第一种是 W3C 标准的盒子模型（标准盒模型）

第二种 IE 标准的盒子模型（怪异盒模型）

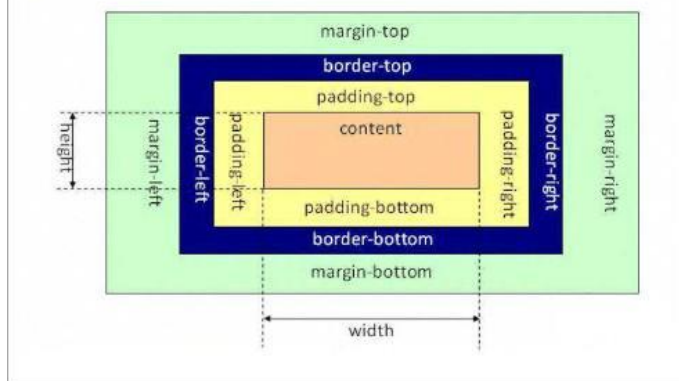
标准盒模型与怪异盒模型的表现效果的区别之处：

1、标准盒模型中 width 指的是内容区域 content 的宽度

height 指的是内容区域 content 的高度

标准盒模型下盒子的大小 = content + border + padding + margin

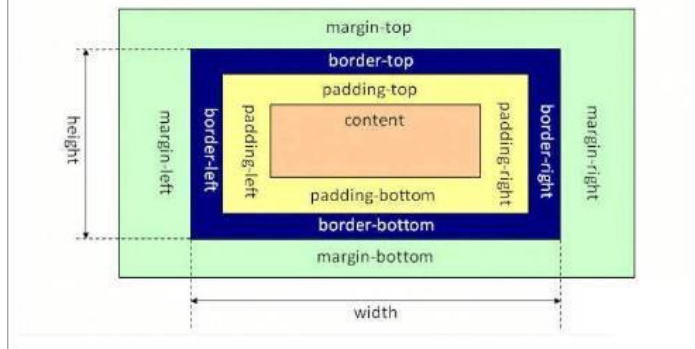
■ 标准盒子模型



2、怪异盒模型中的 width 指的是内容、边框、内边距总的宽度 (content + border + padding); height 指的是内容、边框、内边距总的高度

怪异盒模型下盒子的大小=width (content + border + padding) + margin

■ IE 盒子模型



除此之外，我们还可以通过属性 box-sizing 来设置盒子模型的解析模式可以为 box-sizing 赋两个值：

content-box: 默认值，border 和 padding 不算到 width 范围内，可以理解为是 W3c 的标准模型(default)。总宽=width+padding+border+margin

border-box: border 和 padding 划归到 width 范围内，可以理解为是 IE 的怪异盒模型，总宽=width+margin

5、CSS 中选择器的优先级以及 CSS 权重如何计算？（必会）

! Important>行内样式>ID 选择器>类选择器>标签>通配符>继承>浏览器默认属性

权重

CSS 权重选择器优先级计算表格：

选择器	选择器权重
继承 或者 *	0,0,0,0
元素选择器	0,0,0,1
类选择器, 伪类选择器	0,0,1,0
ID选择器	0,1,0,0
行内样式 style=""	1,0,0,0
!important 重要的	∞ 无穷大

优先级注意点：

1. 权重是有 4 组数字组成, 但是不会有进位。
2. 可以理解为类选择器永远大于元素选择器, id 选择器永远大于类选择器, 以此类推..
3. 等级判断从左向右, 如果某一位数值相同, 则判断下一位数值。
4. 可以简单记忆法: 通配符和继承权重为 0, 标签选择器为 1, 类(伪类)选择器为 10, id 选择器 100, 行内样式表为 1000, !important 无穷大.
5. 继承的权重是 0, 如果该元素没有直接选中, 不管父元素权重多高, 子元素得到的权重都是 0。

权重叠加: 如果是复合选择器, 则会有权重叠加, 需要计算权重。

```

- div ul li -----> 0,0,0,3
- .nav ul li -----> 0,0,1,2
- a:hover -----> 0,0,1,1
- .nav a -----> 0,0,1,1

```

6、列举 5 个以上的 H5input 元素 type 属性值? (必会)

值	描述
button	定义可点击的按钮 (大多与 JavaScript 使用来启动脚本)
checkbox	定义复选框。
color	定义拾色器。
date	定义日期字段 (带有 calendar 控件)
month	定义日期字段的月 (带有 calendar 控件)
time	定义日期字段的时、分、秒 (带有 time 控件)

email	定义用于 e-mail 地址的文本字段
file	定义输入字段和“浏览...”按钮，供文件上传
hidden	定义隐藏输入字段
image	定义图像作为提交按钮
number	定义带有 spinner 控件的数字字段
password	定义密码字段。字段中的字符会被遮蔽。
radio	定义单选按钮。
search	定义用于搜索的文本字段。
submit	定义提交按钮。提交按钮向服务器发送数据。
text	默认。定义单行输入字段，用户可在其中输入文本。默认是 20 个字符。
url	定义用于 URL 的文本字段。

7、CSS 中哪些属性可继承，哪些不可以？（必会）

能继承的属性

1. 字体系列属性: font、font-family、font-weight、font-size、font-style;
2. 文本系列属性:
 - 2.1) 内联元素: color、line-height、word-spacing、letter-spacing、text-transform;
 - 2.2) 块级元素: text-indent、text-align;
3. 元素可见性: visibility
4. 表格布局属性: caption-side、border-collapse、border-spacing、empty-cells、table-layout;
5. 列表布局属性: list-style

不能继承的属性

1. display: 规定元素应该生成的框的类型;
2. 文本属性: vertical-align、text-decoration;
3. 盒子模型的属性: width、height、margin、border、padding;
4. 背景属性: background、background-color、background-image;
5. 定位属性: float、clear、position、top、right、bottom、left、min-width、min-height、max-width、max-height、overflow、clip;

8、CSS 单位中 px、em 和 rem 的区别？（必会）

1. px 像素 (Pixel)。绝对单位。像素 px 是相对于显示器屏幕分辨率而言的，是一个虚拟长度单位，是计算机系统的数字化图像长度单位

2、em 是相对长度单位，相对于当前对象内文本的字体尺寸。如当前对行内文本的字体尺寸未被人为设置，则相对于浏览器的默认字体尺寸。它会继承父级元素的字体大小因此并不是一个固定的值

3、rem 是 CSS3 新增的一个相对单位 (root em, 根 em)，使用 rem 为元素设定字体大小时，仍然是相对大小，但相对的只是 HTML 根元素

4、区别：

IE 无法调整那些使用 px 作为单位的字体大小，而 em 和 rem 可以缩放，rem 相对的只是 HTML 根元素。这个单位可谓集相对大小和绝对大小的优点于一身，通过它既可以做到只修改根元素就成比例地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。目前，除了 IE8 及更早版本外，所有浏览器均已支持 rem

9、rem 适配方法如何计算 HTML 根字号及适配方案？（必会）

通用方案

1、设置根 font-size: 625% (或其它自定的值，但换算规则 1rem 不能小于 12px)

2、通过媒体查询分别设置每个屏幕的根 font-size

3、CSS 直接除以 2 再除以 100 即可换算为 rem

优：有一定适用性，换算也较为简单

劣：有兼容性的坑，对不同手机适配不是非常精准；需要设置多个媒体查询来适应不同手机，单某款手机尺寸不在设置范围之内，会导致无法适配

网易方案

1、拿到设计稿除以 100，得到宽度 rem 值

2、通过给 html 的 style 设置 font-size，把 1 里面得到的宽度 rem 值代入 x

```
document.documentElement.style.fontSize = document.documentElement.clientWidth / x + 'px';
```

3、设计稿 px/100 即可换算为 rem

优：通过动态根 font-size 来做适配，基本无兼容性问题，适配较为精准，换算简便

劣：无 viewport 缩放，且针对 iPhone 的 Retina 屏没有做适配，导致对一些手机的适配不是很到位

手淘方案

1、拿到设计稿除以 10，得到 font-size 基准值

2、引入 flexible

3、不要设置 meta 的 viewport 缩放值

4、设计稿 px/ font-size 基准值，即可换算为 rem

优：通过动态根 font-size、viewport、dpr 来做适配，无兼容性问题，适配精准。

劣：需要根据设计稿进行基准值换算，在不使用 sublime text 编辑器插件开发时，单位计算复杂

10、display: none 与 visibility: hidden 的区别？（必会）

元素隐藏和显示最常用的为 display:none 和 visibility:hidden

display:none 设置该属性后，该元素下的元素都会隐藏，占据的空间消失

visibility:hidden 设置该元素后，元素虽然不可见了，但是依然占据空间的位置

区别

1. `visibility` 具有继承性，其子元素也会继承此属性，若设置 `visibility:visible`，则子元素会显示
2. `visibility` 不会影响计数器的计算，虽然隐藏掉了，但是计数器依然继续运行着。
3. 在 CSS3 的 `transition` 中支持 `visibility` 属性，但是不支持 `display`，因为 `transition` 可以延迟执行，因此配合 `visibility` 使用纯 CSS 实现 `hover` 延时显示效果可以提高用户体验
4. `display:none` 会引起回流(重排)和重绘 `visibility:hidden` 会引起重绘

11、position 的值有哪些，分别有哪些作用？（必会）

静态定位: `static` 默认值不脱离文档流，`top`，`right`，`bottom`，`left` 等属性不生效

绝对定位: `absolute`

绝对定位的关键是找对参照物：找到最近的一级带有带定位的父级元素进行位置移动
如果找不到，那么相对于浏览器窗口进行定位

注：设置了 `position:absolute` 属性后，元素会脱离正常文档流，不在占据空间；左右 `margin` 为 `auto` 将会失效；我们通过 `left`、`top`、`bottom`、`right` 来决定元素位置

相对定位: `relative`

参照物：元素偏移前位置

注：设置了相对定位，左右 `margin` 为 `auto` 仍然有效、并且不会脱离文档流。

固定定位: `fixed`

参照物：浏览器窗口；

注：固定定位会脱离文档流；

当绝对定位和固定定位参照物都是浏览器窗口时的区别： 当出现滚动条时，固定定位的元素不会跟随滚动条滚动，绝对定位会跟随滚动条滚动

12、为什么会出现浮动？浮动元素会引起什么问题？如何清除浮；动？（必会）

浮动将元素排除在普通流之外，即元素将脱离文档流，不占据空间。浮动元素碰到包含它的边界或者浮动元素的边界停留

为什么需要清除浮动

- 1、子元素浮动后，不占位置，父元素的高度无法被撑开，影响与父元素同级的元素；
- 2、与浮动元素同级的非浮动元素（内联元素）会跟随其后；
- 3、若非第一个元素浮动，则该元素之前的元素也需要浮动，否则会影响页面显示的结构解决方法

清除浮动的方式

- 1、使用 CSS 中的 `clear:both`；（放一个空的标签，并设置上述 `css`，注意该标签必须是块元素），属性来清除元素的浮动 可解决 2、3 问题
- 2、对于问题 1，添加如下样式，给父元素添加 `clearfix` 样式：

```

.clearfix:after {
    content: ".";
    display: block;
    height: 0;
    clear: both;
    visibility: hidden;
}
/* for IE */
.clearfix{
    *zoom:1;
}

```

3、给父级元素设置双伪元素:

```

<div class="container clearfix">
    <div class="wrap">aaa</div>
</div>
.clearfix:after{
    content:"";                /*设置内容为空*/
    height:0;                  /*高度为 0*/
    line-height:0;             /*行高为 0*/
    display:block;             /*将文本转为块级元素*/
    visibility:hidden;         /*将元素隐藏*/
    clear:both;                /*清除浮动*/
}
.clearfix{
    zoom:1;                    /*为了兼容 IE*/
}

```

4、给父级元素设置 overflow: hidden; 或 overflow: auto; 本质是构建一个 BFC

13、简述弹性盒子 flex 布局及 rem 布局? (必会)

rem 是 CSS3 新增的一个相对单位, 相对于根节点(html)字体大小的值, r 就是 root

例如: html {font-size:10px} 则 2rem=20px

通过它就可以做到只修改根元素的大小, 就能成比例地调整所有的字体大小, 只依赖 html 字体的大小

适配方案步骤:

1、首先动态计算 html 的 font-size

2、将所有的 px 换算成 rem(计算过程请看下面代码和注释(注意: rem 的换算是根据设计图稿的像素计算的, 下面的计算只是动态计算 html 的 font-size 大小),

请看下面的注意事项

```

<meta name="viewport" content="width=device-width,user-scalable=no"/>
<style>
    body{
        margin: 0;
    }
    div{
        /*width: 80px;*/
        height: 100px;
        width: 4rem;
        height: 4rem;
        /*1rem=20;  nrem=80;    n=80/rem;    n=80/20;    n=4*/
    }

```

```

        background: green;
        float: left;
    }
</style>
<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
</body>
</script>
(function() {
    var html=document.documentElement;

    var width=html.getBoundingClientRect().width;    //获取屏幕宽度(设备独立像素)如
    iPhone6 为 414

    html.style.fontSize=width/16+'px'; // html 的 font-size = 20px

    //iphone5 下    1rem=20    这里之所以除以 16，是因为要把宽度分成 16 份，这个数并没有固定，一般选 15，16，以 ipone5 为准是 16，因为一除可以得到整数 20，好计算。
    })();
</script>

```

注意：

- 1、必需动态的去设置 html 的大小，才能适配
 - 2、根据页面的宽度除以一个系数，把算出的这个值赋给 html 的 font-size 属性，rem 换算值是根据 psd 设计图的宽度/系数的 rem 系数
- 以 640px 设计稿和 750px 的视觉稿，网易这样处理的：

```

var width = document.documentElement.clientWidth; // 屏幕的布局视口宽度
var rem = width / 7.5; // 750px 设计稿将布局视口分为 7.5 份
var rem = width / 6.4; // 640px 设计稿将布局视口分为 6.4 份

```

这样不管是 750px 设计稿还是 640px 设计稿，1rem 等于设计稿上的 100px。故 px 转换 rem 时：1rem = 1px * 0.01;

在 750px 设计稿上：

设计稿上 75px 对应 0.75rem，距离占设计稿的 10%；

在 ipone6 上：

```
width = document.documentElement.clientWidth = 375px;
```

```
1rem = 375px / 7.5 = 50px;
```

```
0.75rem = 37.5px; (37.5/375=10%; 占屏幕 10%)
```

在 ipone5 上：

```
width = document.documentElement.clientWidth = 320px;
```

```
rem = 320px / 7.5 = 42.667px;
```

```
0.75rem = 32px; (32/320=10%; 占屏幕 10%)
```

故对于设计稿上任何一个尺寸换成 rem 后，在任何屏下对应的尺寸占屏幕宽度的百分比相同。故这种布局可以百分比还原设计图

- 2.1) 为什么要除一个数字，原因是：一个页面里，不可能全都是整屏的元素，肯定有一行中放多个元素。所以就把一行分成 n 份
 - 2.2) 不除一个数字的话，那 1 个 rem 就是屏幕的宽度，这个值太大，如果一个元素的宽度比它小的话，就不方便计算
 - 2.3) 这个系数，自己定。多少都可以，但是建议给一个能整除的值（这个能整除的数，是还要根据设计稿能整除的数。）
- 3、对于切的图片，尺寸是根据设计图的尺寸宽度的，显示起来会很大，如果是 img 标签，可以设置宽度为切出的图片尺寸，换算成 rem，如果是 background-img，用 background-size 属性，设置设计图尺寸宽高，换算成 rem 进行图片的缩放适配。

对于上述的第二点，根据设计稿动态转换 rem，这里说一下，前面的计算是动态的设置 html 的 font-size 的大小，这是根据设备的独立像素计算的。而设计稿往往是根据物理像素，即设备像素设计的，往往很大，是 750px 及以上，所以在转换 rem 的时候，转换是根据 psd 设计稿的像素进行转换，即 $1\text{rem} = \text{设计稿像素宽度} / \text{系数}$ ，例如，如果是 1080px 的设计稿，那么，就用 $1\text{rem} = 1080 / 18 = 60\text{px}$ （这里用 18 做系数，是因为能整除），然后布局的时候就根据设计稿的元素尺寸转换，例如设计稿一个元素的高为 60px，那么就可以转化为 1rem 了

特点：

- 1、所有有单位的属性会根据屏幕的尺寸自动计算大小
- 2、同样一个元素，在不同的设备下的大小是不一样的。在尺寸小的设备下显示的小，在尺寸大的设备下显示的大

- 3、一般以 iphone6 为基准，以它的宽度 750 除上一个系数，再去算 rem

Tips: 上述步骤 2 中换算可以通过 Hbuilder 将 px 自动转 rem 以及通过 less 自动计算成 rem, sublime 也可以通过插件进行自动转换

- 3.1) 打开 Hbuilder，顶部栏的工具》选项》Hbuilder》代码助手》px 自动转 rem 设置

- 3.2) less 自动转换: Hbuilder 也可以将 less 文件自动转成 css 文件。less 文件的书写如下所示

比如想设置宽度为 187px，高度为 100px 的元素，可以通过下面方式计算适配

@rem: 25rem; /*这是 1rem = X px 的 X 的值，但是用了 rem 做单位而已*/

```
div{
    width: 187/@rem;
    height: 100/@rem;
}
```

弹性布局适配(会配合 rem 适配使用)

兼容情况

IE10 及以上、ios9 及以上、android4.4 及以上版本支持

特点

- 1、默认所有子元素都会在一行中显示，即使给予元素一个很大的宽度
 - 2、父级加了这条属性，子级的 float、vertical-align 就会失效
 - 3、如果兼容低版本的机型要加前缀-webkit-，包括后面讲的所有属性
- 容器属性（父元素样式） 具体看菜鸟教程或阮一峰的教程，这里说一下一些重点知识

- 3.1) flex-direction: 子元素排列方向（主轴的方向，如果设置了 column，则意味着主轴旋转了 90 度）

- 3.2) flex-wrap: 换行方式

- 3.3) flex-flow: 以上两种方式的简写

- 3.4) justify-content: 水平对齐方式（子元素在主轴上的对齐方式）

- 3.5) align-items: 垂直对齐方式（子元素在交叉轴上的对齐方式）

- 3.6) align-content: 多行垂直对齐方式（多根轴线的对齐方式）

项目属性（子元素样式）

- 1、order: 排列位置 //如果有两个的值是相等，按书写顺序排列

- 2、flex-grow: 扩展比例

flex-grow 当父级的宽度大于所有子元素宽度之和时，根据父级的剩余空间，设置子元素的扩展比例（设置后，元素给的固定宽度会被覆盖）它是一个系数默认为 0，即如果存在剩余空间也不扩展

剩余空间: 剩余空间=父级的宽度-所有子元素的宽度和

注意: 如果没有设置初始宽度，也没有内容，则默认为 0，否则为内容的宽度。例如设置了文字，会撑开有初始宽度

子元素宽度计算公式

子元素的宽度=(父级的宽度-所有子元素的宽度和)/所有子元素的 flex-grow 属性值之和*子元素的 flex-grow 属性值+子元素初始宽度

- 3、flex-shrink: 收缩比例

flex-shrink 当所有子元素宽度之和大于父级宽度的时候，根据超出的空间，设置子元素的

收缩比例（设置后，元素给的固定宽度会被覆盖）它是一个系数默认为 1，如果给个 0 的话，就不会收缩

超出空间：超出空间=所有子元素的宽度和-父级的宽度

子元素宽度计算公式

1、算出超出空间，所有子元素的宽度和-父级的宽度

2、子元素的初始宽度*子元素的 flex-shrink 值

3、算出第二步所有结果的和

4、每个子元素的第二步/第三步*第一步

5、子元素的初始宽度-第四步

flex-basis：元素的大小

flex：以上三个属性的简写

align-self：单独的垂直对齐方式（交叉轴方向上）

14、如何解决 margin “塌陷”？（必会）

外边距塌陷共有两种情况：

第一种情况：两个同级元素，垂直排列，上面的盒子给 margin-bottom 下面的盒子给 margin-top，那么他们两个的间距会重叠，以大的那个计算。解决这种情况

的方法为：两个外边距不同时出现

第二种情况：两个父子元素，内部的盒子给 margin-top，其父级也会受到影响，同时产生上边距，父子元素会进行粘连。

解决方案：

1、为父盒子设置 border，添加 border 后父子盒子就不是真正意义上的贴合（可以设置成透明：border: 1px solid transparent);

2、为父盒子添加 overflow: hidden;

3、为父盒子设定 padding 值;

4、为父盒子添加 position: fixed;

5、为父盒子添加 display: table;

6、利用伪元素给父元素的前面添加一个空元素

```
.father::before {  
    content:'';  
    display:table;  
}
```

15、::before 和 ::after 中双冒号和单冒号有什么区别、作用？（必会）

区别

在 CSS 中伪类一直用 : 表示，如 :hover, :active 等

伪元素在 CSS1 中已存在，当时语法是用 : 表示，如 :before 和 :after

后来在 CSS3 中修订，伪元素用 :: 表示，如 ::before 和 ::after，以此区分伪元素和伪类

由于低版本 IE 对双冒号不兼容，开发者为了兼容性各浏览器，继续使用 :after 这种老法表示伪元素

单冒号 (:) 用于 CSS3 的伪类

双冒号 (::) 用于 CSS3 的伪元素

想让插入的内容出现在其它内容前，使用 ::before，否则，使用 ::after;

在代码顺序上，::after 生成的内容也比 ::before 生成的内容靠后

作用：

`::before` 和 `::after` 的主要作用是在元素内容前后加上指定内容伪类与伪元素都是用于向选择器加特殊效果

伪类与伪元素的本质区别就是是否抽象创造了新元素

伪类只要不是互斥可以叠加使用

伪元素在一个选择器中只能出现一次，并且只能出现在开始和末尾

伪类与伪元素优先级分别与类、标签优先级相同

16、CSS3 新增伪类，以及伪元素？（必会）

CSS3 新增伪类

`p:first-of-type` 选择属于其父元素的首个<p>元素

`p:last-of-type` 选择属于其父元素的最后<p>元素

`p:nth-child(n)` 选择属于其父元素的第 n 个子元素并且必须是<p>元素

`p:nth-last-child(n)` 选择属于其父元素的倒数第 n 个子元素并且必须是<p>元素

`p:nth-of-type(n)` 选择属于其父元素第 n 个<p>元素

`p:nth-last-of-type(n)` 选择属于其父元素倒数第 n 个<p>元素

`p:last-child` 选择属于其父元素最后一个子元素的并且必须是<p>元素

`p:target` 和锚点链接一起使用

URL 带有后面跟有锚名称 #，指向文档内某个具体的元素。这个被链接的元素就是目标元素 (target element)。

`:target` 选择器可用于选取当前活动的目标元素。

`:not(p)` 选择非<p>元素

`:enabled` 选中不在禁用状态下的表单控件

`:disabled` 选中禁用状态下的表单控件

`:checked` 选中 单选框或复选框被选中 的元素

伪元素

`::first-letter` 将样式添加到文本的首字母

`::first-line` 将样式添加到文本的首行

`::before` 在某元素之前插入某些内容

`::after` 在某元素之后插入某些内容

17、Bootstrap 栅格系统的工作原理？（必会）

原理

1、行 (row) 必须包含在 `.container` (固定宽度) 或 `.container-fluid` (100% 宽度) 中，以便为其赋予合适的排列 (alignment) 和内补 (padding)

2、通过行 (row) 在水平方向创建一组列 (column)

3、自己内容应当放置于列 (column) 内，并且，只有列可以作为行 (row) 的直接子元素

4、类似 `.row` 和 `.col-xs-4` 这种预定义的类，可以用来快速创建栅格布局。Bootstrap 源码中定义的 `mixin` 也可以用来创建语义化布局

5、通过为列设置 `padding` 属性，从而创建列与列之间的间隔 (gutter)。通过为 `.row` 元素设置负值 `margin` 从而抵消为 `.container` 元素设置的 `padding`，也就间接为行 (row) 所包含的列 (column) 抵消掉了 `padding`

6、栅格系统的列是通过指定 1 到 12 的值来表示其跨越范围。例如三个等宽的列可以使用三个 `.col-xs-4` 来创建

7、如果一行 (row) 中包含了的列 (column) 大于 12，多余的列所在的元素将作为一个整体另起一行排列

8、栅格类适用于与屏幕宽度大于或等于分界点大小的设备，并且针对小屏幕覆盖栅格类

使用 Bootstrap 响应式布局

首先需要在 head 中引入 meta 标签, 添加 viewport 属性, content 中宽度等于设备宽度, initial-scale: 页面首次被显示可见区域的缩放级别, 取值 1 则页面按实际尺寸显示, 无任何缩放; maximum-scale: 允许用户缩放到的最小比例; user-scalable: 用户是否可以手动缩放。代码如下:

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

```
<link rel="stylesheet" type="text/css" href="/stylesheets/bootstrap.min.css">
```

下面为使用 Bootstrap 布局的页面 (登录表单界面), 针对的是手机超小屏幕 (iphone5s) 和 PC 屏幕 (>=1200px)。col-xs-12: 小屏幕占 12 列大小, col-lg-5: 大屏幕占 5 列大小, col-lg-offset-3: 大屏幕缩进 3 列大小。这是一个比较简单的实例, 想要适应其他屏幕如平板可添加 col-md-* 属性, 大屏手机可添加 col-sm-* 属性。具体的屏幕使用哪个属性, 可参考文档中的针对不同屏幕 Bootstrap 栅格系统的不同使用。

```
<div class="container-fluid login">
  <div class="row">
    <div class="col-xs-12 col-sm-12 col-md-8 col-lg-5 col-lg-offset-3">
      <form class="form-horizontal loginForm">
        <h3 class="form-signin-heading">用户登录</h3>
        <div class="form-group">
          <label for="email" class="col-sm-2 col-xs-3 control-label">邮箱</label>
          <div class="col-sm-8 col-xs-8">
            <input type="text" class="form-control" name="email" placeholder="请输入邮箱">
            <span class="glyphicon glyphicon-ok form-control-feedback" aria-hidden="true"></span>
          </div>
        </div>
        <div class="form-group">
          <label for="password" class="col-sm-2 col-xs-3 control-label">密码</label>
          <div class="col-sm-8 col-xs-8">
            <input type="password" class="form-control" name="password" placeholder="请输入密码">
            <span class="glyphicon glyphicon-ok form-control-feedback" aria-hidden="true"></span>
          </div>
        </div>
        <div class="form-group">
          <div class="col-sm-offset-2 col-sm-4 col-xs-4">
            <div class="checkbox">
              <label>
                <input type="checkbox"> 记住我 </label>
              </div>
            </div>
            <div class="col-sm-4 col-xs-4 control-label">
              <a href="resetPwd.html" id="forget">忘记密码? </a>
            </div>
          </div>
        </div>
        <div class="form-group">
          <div class="col-sm-12 col-lg-12">
            <button type="button" class="btn btn-primary btn-block" id="submit">登录</button>
          </div>
        </div>
      </form>
    </div>
  </div>
</div>
```

18、BFC 是什么？（高薪常问）

定义

BFC(Block formatting context)直译为“块级格式化上下文”。它是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

布局规则

- 1、内部的 Box 会在垂直方向，一个接一个地放置
- 2、Box 垂直方向的距离由 margin 决定。属于同一个 BFC 的两个相邻 Box 的 margin 会发生重叠
- 3、每个元素的 margin box 的左边，与包含块 border box 的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此
- 4、BFC 的区域不会与 float box 重叠
- 5、BFC 就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此
- 6、计算 BFC 的高度时，浮动元素也参与计算

哪些元素会生成 BFC：

- 1、根元素
- 2、float 属性不为 none
- 3、position 为 absolute 或 fixed
- 4、display 为 inline-block, table-cell, table-caption, flex, inline-flex
- 5、overflow 不为 visible

19、什么是渐进增强和优雅降级?它们有什么不同？（了解）

优雅降级和渐进增强印象中是随着 CSS3 流出来的一个概念。由于低级浏览器不支持 CSS3，但 CSS3 的效果又太优秀不忍放弃，所以在高级浏览中使用 CSS3 而低级浏览器只保证最基本的功能。关键的区别 是他们所侧重的内容，以及这种不同造成的工作流程的差异

举个例子：

```
a {
    display: block;
    width: 200px;
    height: 100px;
    background: aquamarine;
    /*我就是要用这个新 css 属性*/
    transition: all 1s ease 0s;
    /*可是发现了一些低版本浏览器不支持怎么办呢*/
    /*往下兼容*/
    -webkit-transition: all 1s ease 0s;
    -moz-transition: all 1s ease 0s;
    -o-transition: all 1s ease 0s;
    /*那么通常这样考虑的和这样的侧重点出发的 css 就是优雅降级*/
}
a:hover{
    height: 200px;
}
/*那如果我们的产品要求我们要从低版本的浏览器兼容开始*/
a {
    /*优先考虑低版本的*/
    -webkit-transition: all 1s ease 0s;
```

```

-moz-transition: all 1s ease 0s;
-o-transition: all 1s ease 0s;
/*高版本的就肯定是渐进渐强*/
transition: all 1s ease 0s;
}

```

“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
“渐进增强”观点则认为应关注于内容本身

20、iframe 有哪些优缺点？（了解）

iframe 的优点：

- 1、重载页面时不需要重载整个页面，只需要重载页面中的一个框架页(减少了数据的传输，加快了网页下载速度)
- 2、技术易于掌握，使用方便，使用者众多，可主要应用于不需搜索引擎来搜索的页面

iframe 的缺点：

- 3、iframe 会阻塞主页面的 Onload 事件；
- 4、会产生很多页面，不容易管理
- 5、不容易打印（目前只能实现分框架页面的打印，不能实现对 frameset 的打印）
- 6、浏览器的后退按钮无效（只能针对实现当前光标所在页面的前进与后退，无法实现 frameset 整个页面的前进与后退）
- 7、代码复杂,无法被一些搜索引擎索引到（有些搜索引擎对框架结构的页面不能正确处理，会影响到搜索结果的排列名次）
- 8、多数小型的移动设备（手机）无法完全显示框架
- 9、多框架的页面会增加服务器的 http 请求，影响页面的并行加载。
（并行加载：同一时间针对同一域名下的请求。一般情况，iframe 和所在页面在同一个域下面，而浏览器的并加载的数量是有限制的。

21、使用 CSS 怎么让 Chrome 支持小于 12px 的文字比如 10px？（了解）

作法

针对谷歌浏览器内核，加 webkit 前缀，用 transform:scale() 这个属性进行缩放！

```

<style>
    p span{font-size:10px;-webkit-transform:scale(0.8);display:block;}
</style>
<p>
    <span>豪豪豪 10px</span>
</p>

```

JavaScript 基础

1、 JavaScript 的基本类型有哪些？引用类型有哪些？ null 和 undefined 的区别？（必会）

数据类型

基本数据类型：Number、String、Boolean、null、undefined

引用数据类型：Function、Object、Array

区别

undefined: 表示变量声明但未初始化时的值

null: 表示准备用来保存对象，还没有真正保存对象的值。从逻辑角度看，null 值表示一个空对象指针

ECMA 标准要求 null 和 undefined 等值判断返回 true

```
null == undefined // true
```

```
null === undefined // false
```

2、 如何判断 JavaScript 的数据类型？（必会）

判断方法

typeof

typeof 可以用来区分除了 null 类型以外的原始数据类型，对象类型的可以从普通对象里面识别出函数：

```
typeof undefined // "undefined"
```

```
typeof null // "object"
```

```
typeof 1 // "number"
```

```
typeof "1" // "string"
```

```
typeof Symbol() // "symbol"
```

```
typeof function() {} // "function"
```

```
typeof {} // "object"
```

问题一：typeof 不能识别 null，如何识别 null？

答案：如果想要判断是否为 null，可以直接使用===全等运算符来判断（或者使用下面的 Object.prototype.toString 方法）：

```
let a = null
```

```
a === null // true
```

问题二：typeof 作用于未定义的变量，会报错吗？

答案：不会报错，返回"undefined"。

```
typeof randomVariable // "undefined"
```

问题三：typeof Number(1)的返回值是什么？

答案："number"。注意 Number 和 String 作为普通函数调用的时候，是把参数转化为相应的原始数据类型，也就是类似于做一个强制类型转换的操作，而不是默认当做构造函数调用。注意和 Array 区分，Array(...)等价于 new Array(...)

```
typeof Number(1) // "number"
```

```
typeof String("1") // "string"
```

```
Array(1, 2, 3)
```

```
// 等价于
```

```
new Array(1, 2, 3)
```

问题四：typeof new Number(1)的返回值是什么？

答案: "object".

```
typeof new Number(1) // "object"
typeof new String(1) // "object"
instanceof
instanceof 不能用于判断原始数据类型的数据:
```

```
3 instanceof Number // false
'3' instanceof String // false
true instanceof Boolean // false
instanceof 可以用来判断对象的类型:
```

```
var date = new Date()
date instanceof Date // true
var number = new Number()
number instanceof Number // true
var string = new String()
string instanceof String // true
```

需要注意的是, instanceof 的结果并不一定是可靠的, 因为在 ECMAScript7 规范中可以通过自定义 Symbol.hasInstance 方法来覆盖默认行为。

```
Object.prototype.toString
Object.prototype.toString.call(undefined).slice(8, -1) // "Undefined"
Object.prototype.toString.call(null).slice(8, -1) // "Null"
Object.prototype.toString.call(3).slice(8, -1) // "Number"
Object.prototype.toString.call(new Number(3)).slice(8, -1) // "Number"
Object.prototype.toString.call(true).slice(8, -1) // "Boolean"
Object.prototype.toString.call('3').slice(8, -1) // "String"
Object.prototype.toString.call(Symbol()).slice(8, -1) // "Symbol"
```

由上面的示例可知, 该方法没有办法区分数字类型和数字对象类型, 同理还有字符串类型和字符串对象类型、布尔类型和布尔对象类型

另外, ECMAScript7 规范定义了符号 Symbol.toStringTag, 你可以通过这个符号自定义 Object.prototype.toString 方法的行为:

```
'use strict'
var number = new Number(3)
number[Symbol.toStringTag] = 'Custom'
Object.prototype.toString.call(number).slice(8, -1) // "Custom"
function a () {}
a[Symbol.toStringTag] = 'Custom'
Object.prototype.toString.call(a).slice(8, -1) // "Custom"
var array = []
array[Symbol.toStringTag] = 'Custom'
Object.prototype.toString.call(array).slice(8, -1) // "Custom"
```

因为 Object.prototype.toString 方法可以通过 Symbol.toStringTag 属性来覆盖默认行为, 所以使用这个方法来判断数据类型也不一定是可靠的

Array.isArray

Array.isArray(value) 可以用来判断 value 是否是数组:

```
Array.isArray([]) // true
Array.isArray({}) // false
(function () {console.log(Array.isArray(arguments))})() // false
```

3、简述创建函数的几种方式? (必会)

第一种 (函数声明):

```
function sum1(num1,num2){
```

```

        return num1+num2;
    }
}
第二种（函数表达式）
var sum2 = function(num1,num2){
    return num1+num2;
}
第三种（函数对象方式）
var sum3 = new Function("num1","num2","return num1+num2");

```

4、Javascript 创建对象的几种方式？（必会）

1、简单对象的创建 使用对象字面量的方式 {}

创建一个对象（最简单，好理解，推荐使用）

代码如下

```

var Cat = {}; //JSON
Cat.name="kity"; //添加属性并赋值
Cat.age=2;
Cat.sayHello=function(){
    alert("hello "+Cat.name+", 今年"+Cat["age"]+"岁了"); //可以使用 "." 的方式访问
    属性， 也 可以使用 HashMap 的方式访问
}
Cat.sayHello(); //调用对象的（方法）函数

```

2、用 function(函数)来模拟 class

2.1) 创建一个对象，相当于 new 一个类的实例(无参构造函数)

代码如下

```

function Person(){
}
var personOne=new Person(); //定义一个 function，如果有 new 关键字去“实例化”，那么该 function 可以看作是一个类
personOne.name="dylan";
personOne.hobby="coding";
personOne.work=function(){
    alert(personOne.name+" is coding now...");
}
personOne.work();

```

2.2) 可以使用有参构造函数来实现，这样定义更方便，扩展性更强（推荐使用）

代码如下

```

function Pet(name,age,hobby){
    this.name=name;//this 作用域：当前对象
    this.age=age;
    this.hobby=hobby;
    this.eat=function(){
        alert("我叫"+this.name+", 我喜欢"+this.hobby+", 也是个吃货");
    }
}
var maidou =new Pet("麦兜",5,"睡觉");//实例化/创建对象
maidou.eat();//调用 eat 方法(函数)

```

3、使用工厂方式来创建 (Object 关键字)

代码如下：

```

var wcDog = new Object();
wcDog.name="旺财";
wcDog.age=3;
wcDog.work=function(){
    alert("我是"+wcDog.name+", 汪汪汪.....");
}

```

```
wcDog.work();
```

4、使用原型对象的方式 prototype 关键字

代码如下：

```
function Dog() {  
}  
Dog.prototype.name = "旺财";  
Dog.prototype.eat = function() {  
    alert(this.name + "是个吃货");  
}  
var wangcai = new Dog();  
wangcai.eat();
```

5、混合模式(原型和构造函数)

代码如下：

```
function Car(name, price) {  
    this.name=name;  
    this.price=price;  
}  
Car.prototype.sell=function() {  
    alert("我是"+this.name+", 我现在卖"+this.price+"万元");  
}  
var camry = new Car("凯美瑞", 27);  
camry.sell();
```

6、动态原型的方式(可以看作是混合模式的一种特例)

代码如下：

```
function Car(name, price) {  
    this.name=name;  
    this.price=price;  
    if(typeof Car.sell=="undefined"){  
        Car.prototype.sell=function() {  
            alert("我是"+this.name+", 我现在卖"+this.price+"万元");  
        }  
        Car.sell=true;  
    }  
}  
var camry = new Car("凯美瑞", 27);  
camry.sell();
```

以上几种，是 javascript 中最常用的创建对象的方式

5、请指出 JavaScript 宿主对象和原生对象的区别？（必会）

原生对象

“独立于宿主环境的 ECMAScript 实现 提供的对象”

包含：Object、Function、Array、String、Boolean、Number、Date、RegExp、Error、EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URIError

内置对象

开发者不必明确实例化内置对象，它已被内部实例化了

同样是“独立于宿主环境”。而 ECMA-262 只定义了两个内置对象，即 Global 和 Math

宿主对象

BOM 和 DOM 都是宿主对象。因为其对于不同的“宿主”环境所展示的内容不同。其实说白了就是，ECMAScript 官方未定义的对象都属于宿主对象，因为其未定义的对象大多数是自己通过 ECMAScript 程序创建的对象

6、JavaScript 内置的常用对象有哪些？并列举该对象常用的方法？

(必会)

对象及方法

Arguments 函数参数集合

Arguments[] 函数参数的数组

Arguments 一个函数的参数和其他属性

Arguments.callee 当前正在运行的函数

Arguments.length 传递给函数的参数的个数

Array 数组

length 属性 动态获取数组长度

join() 将一个数组转成字符串。返回一个字符串。

reverse() 将数组中各元素颠倒顺序

delete 运算符 只能删除数组元素的值，而所占空间还在，总长度没变(arr.length)。

shift() 删除数组中第一个元素，返回删除的那个值，并将长度减 1。

pop() 删除数组中最后一个元素，返回删除的那个值，并将长度减 1。

unshift() 往数组前面添加一个或多个数组元素，长度要改变。arrObj.unshift(“a” , “b”, “c”)

push() 往数组结尾添加一个或多个数组元素，长度要改变。arrObj.push(“a” , “b”, “c”)

concat() 连接数组

slice() 返回数组的一部分

sort() 对数组元素进行排序

splice() 插入、删除或替换数组的元素

toLocaleString() 把数组转换成局部字符串

toString() 将数组转换成一个字符串

forEach 遍历所有元素

```
var arr = [1, 2, 3];
```

```
arr.forEach(function(item, index) {
```

```
    // 遍历数组的所有元素
```

```
    console.log(index, item);
```

```
});
```

every 判断所有元素是否都符合条件

```
var arr = [1, 2, 3];
```

```
var arr1 = arr.every(function(item, index) {
```

```
    if (item < 4) {
```

```
        return true;
```

```
    }
```

```
});
```

```
console.log(arr1); // true
```

sort 排序

```
var arr = [1, 5, 2, 7, 3, 4];
```

```
var arr2 = arr.sort(function(a, b) {
```

```
    // 从小到大
```

```
    return a-b;
```

```
    // 从大到小
```

```
    return b-a;
```

```
});
```

```
console.log(arr2); // 1, 2, 3, 4, 5, 7
```

map 对元素重新组装，生成新数组

```
var arr = [1, 5, 2, 7, 3, 4];
```



```

var arr2 = arr.map(function(item, index) {
    return '<b>' + item + '</b>';
})
console.log(arr2);
filter 过滤符合条件的元素
var arr = [1, 2, 3, 4];
var arr2 = arr.filter(function(item, index) {
    if (item>2) {
        return true;
    }
})
console.log(arr2); // [3, 4]

```

String 字符串对象

Length 获取字符串的长度。如: var len = strObj.length
 toLowerCase() 将字符串中的字母转成全小写。如: strObj.toLowerCase()
 toUpperCase() 将字符串中的字母转成全大写。如: strObj.toUpperCase()
 charAt(index) 返回指定下标位置的一个字符。如果没有找到, 则返回空字符串
 substr() 在原始字符串, 返回一个子字符串
 substring() 在原始字符串, 返回一个子字符串
 区别: ''
 "abcdefgh".substring(2, 3) = "c"
 "abcdefgh".substr(2, 3) = "cde"
 split() 将一个字符串转成数组
 charCodeAt() 返回字符串中的第 n 个字符的代码
 concat() 连接字符串
 fromCharCode() 从字符编码创建一个字符串
 indexOf() 返回一个子字符串在原始字符串中的索引值(查找顺序从左往右查找)。如果没有找到, 则返回-1
 lastIndexOf() 从后向前检索一个字符串
 localeCompare() 用本地特定的顺序来比较两个字符串
 match() 找到一个或多个正则表达式的匹配
 replace() 替换一个与正则表达式匹配的子串
 search() 检索与正则表达式相匹配的子串
 slice() 抽取一个子串
 toLocaleLowerCase() 把字符串转换小写
 toLocaleUpperCase() 将字符串转换成大写
 toLowerCase() 将字符串转换成小写
 toString() 返回字符串
 toUpperCase() 将字符串转换成大写
 valueOf()

Boolean 布尔对象

Boolean.toString() 将布尔值转换成字符串
 Boolean.valueOf() Boolean 对象的布尔值

Date 日期时间

创建 Date 对象的方法

(1) 创建当前(现在)日期对象的实例, 不带任何参数

```
var today = new Date();
```

(2) 创建指定时间戳的日期对象实例, 参数是时间戳。

时间戳: 是指某一个时间距离 1970 年 1 月 1 日 0 时 0 分 0 秒, 过去了多少毫秒值(1 秒

=1000 毫秒)

```
var timer = new Date(10000); //时间是 1970 年 1 月 1 日 0 时 0 分 10 秒
```

(3) 指定一个字符串的日期时间信息, 参数是一个日期时间字符串

```
var timer = new Date( "2015/5/25 10: 00: 00" );
```

(4) 指定多个数值参数

```
var timer = new Date(2015+100, 4, 25, 10, 20, 0); //顺序为: 年、月、日、  
时、分、秒, 年、月、日是必须的
```

方法:

Date.getDate() 返回一个月中的某一天

Date.getDay() 返回一周中的某一天

Date.getFullYear() 返回 Date 对象的年份字段

Date.getHours() 返回 Date 对象的小时字段

Date.getMilliseconds() 返回 Date 对象的毫秒字段

Date.getMinutes() 返回 Date 对象的分钟字段

Date.getMonth() 返回 Date 对象的月份字段

Date.getSeconds() 返回 Date 对象的秒字段

Date.getTime() 返回 Date 对象的毫秒表示

Error 异常对象

Error.message 可以读取的错误消息

Error.name 错误的类型

Error.toString() 把 Error 对象转换成字符串

EvalError 在不正确使用 eval()时抛出

SyntaxError 抛出该错误用来通知语法错误

RangeError 在数字超出合法范围时抛出

ReferenceError 在读取不存在的变量时抛出

TypeError 当一个值的类型错误时, 抛出该异常

URIError 由 URL 的编码和解码方法抛出

Function 函数构造器

Function.apply() 将函数作为一个对象的方法调用

Function.arguments[] 传递给函数的参数

Function.call() 将函数作为对象的方法调用

Function.caller 调用当前函数的函数

Function.length 已声明的参数的个数

Function.prototype 对象类的原型

Function.toString() 把函数转换成字符串

Math 数学对象

Math 对象是一个静态对象

Math.PI 圆周率

Math.abs() 绝对值

Math.ceil() 向上取整(整数加 1, 小数去掉)

Math.floor() 向下取整(直接去掉小数)

Math.round() 四舍五入

Math.pow(x, y) 求 x 的 y 次方

Math.sqrt() 求平方根

Number 数值对象

Number.MAX_VALUE 最大数值

Number.MIN_VALUE 最小数值

Number.NaN 特殊的非数字值

Number.NEGATIVE_INFINITY 负无穷大

Number.POSITIVE_INFINITY 正无穷大

Number.toExponential() 用指数计数法格式化数字

Number.toFixed() 采用定点计数法格式化数字

Number.toLocaleString() 把数字转换成本地格式的字符串

Number.toPrecision() 格式化数字的有效位

Number.toString() 将一个数字转换成字符串

Number.valueOf() 返回原始数值

Object 基础对象

Object 含有所有 JavaScript 对象的特性的超类

Object.constructor 对象的构造函数

Object.hasOwnProperty() 检查属性是否被继承

Object.isPrototypeOf() 一个对象是否是另一个对象的原型

Object.propertyIsEnumerable() 是否可以通过 for/in 循环看到属性

Object.toLocaleString() 返回对象的本地字符串表示

Object.toString() 定义一个对象的字符串表示

Object.valueOf() 指定对象的原始值

RegExp 正则表达式对象

RegExp.exec() 通用的匹配模式

RegExp.global 正则表达式是否全局匹配

RegExp.ignoreCase 正则表达式是否区分大小写

RegExp.lastIndex 下次匹配的起始位置

RegExp.source 正则表达式的文本

RegExp.test() 检测一个字符串是否匹配某个模式

RegExp.toString() 把正则表达式转换成字符串

7、=== 和 ==的区别? (必会)

区别

===: 三个等号称为等同符, 当等号两边的值为相同类型的时候, 直接比较等号两边的值, 值相同则返回 true, 若等号两边的值类型不同时直接返回 false。也就是说三个等号既要判断值也要判断类型是否相等

==: 两个等号称为等值符, 当等号两边的值为相同类型时比较值是否相同, 类型不同时会发生类型的自动转换, 转换为相同的类型后再作比较。也就是说两个等号只要值相等就可以

8、null, undefined 的区别 (必会)

区别

null 表示一个对象被定义了, 值为“空值”;

undefined 表示不存在这个值

typeof undefined //"undefined"

undefined : 是一个表示“无”的原始值或者说表示“缺少值”, 就是此处应该有一个值, 但还没有定义。当尝试读取时会返回 undefined;

例如变量被声明了, 但没有赋值时, 就等于 undefined

typeof null //"object"

null : 是一个对象(空对象, 没有任何属性和方法);

例如作为函数的参数, 表示该函数的参数不是对象;

注意:

在验证 null 时, 一定要使用 ===, 因为 == 无法分别 null 和 undefined

undefined 表示“缺少值”, 就是此处应该有一个值, 但是还没有定义。

典型用法是:

1、变量被声明了, 但没有赋值时, 就等于 undefined

2、调用函数时, 应该提供的参数没有提供, 该参数等于 undefined

- 3、对象没有赋值的属性，该属性的值为 undefined
 - 4、函数没有返回值时，默认返回 undefined
- null 表示“没有对象”，即该处不应该有值。典型用法是：
- 4.1) 作为函数的参数，表示该函数的参数不是对象
 - 4.2) 作为对象原型链的终点

9、JavaScript 中什么情况下会返回 undefined 值？（必会）

- 1、访问声明，但是没有初始化的变量

```
var aaa;
console.log(aaa); // undefined
```
- 2、访问不存在的属性

```
var aaa = {};
console.log(aaa.c);
```
- 3、访问函数的参数没有被显式的传递值

```
(function (b){
    console.log(b); // undefined
})()
```
- 4、访问任何被设置为 undefined 值的变量

```
var aaa = undefined; console.log(aaa); // undefined
```
- 5、没有定义 return 的函数隐式返回

```
function aaa() {} console.log(aaa()); // undefined
```
- 6、函数 return 没有显式的返回任何内容

```
function aaa() {
    return;
}
console.log(aaa()); // undefined
```

10、如何区分数组和对象？（必会）

- 方法一：通过 ES6 中的 Array.isArray 来识别
- ```
Array.isArray([]) //true
Array.isArray({}) //false
```
- 方法二：通过 instanceof 来识别
- ```
[] instanceof Array //true
{} instanceof Array //false
```
- 方法三：通过调用 constructor 来识别
- ```
{}.constructor //返回 object
[].constructor //返回 Array
```
- 方法四：通过 Object.prototype.toString.call 方法来识别
- ```
Object.prototype.toString.call([]) //["object Array"]
Object.prototype.toString.call({}) //["object Object"]
```

11、多维数组降维的几种方法（必会）

(1) 数组字符串化

```
let arr = [[222, 333, 444], [55, 66, 77] ]
arr += '';
arr = arr.split(',');
```

```
console.log(arr); // ["222", "333", "444", "55", "66", "77"]
```

(2) 递归

```
function reduceDimension(arr){
    let ret = [];
    let toArr = function(arr){
        arr.forEach(function(item){
            item instanceof Array ? toArr(item) : ret.push(item);
        });
    }
    toArr(arr);
    return ret;
}
```

3、Array.prototype.flat()

```
var arr1 = [1, 2, [3, 4]];
arr1.flat();
// [1, 2, 3, 4]

var arr2 = [1, 2, [3, 4, [5, 6]]];
arr2.flat();
// [1, 2, 3, 4, [5, 6]]

var arr3 = [1, 2, [3, 4, [5, 6]]];
arr3.flat(2);
// [1, 2, 3, 4, 5, 6]

//使用 Infinity 作为深度，展开任意深度的嵌套数组
arr3.flat(Infinity);
// [1, 2, 3, 4, 5, 6]
```

4、使用 stack 无限反嵌套多层嵌套数组

```
var arr1 = [1,2,3,[1,2,3,4, [2,3,4]]];
function flatten(input) {
    const stack = [...input];
    const res = [];
    while (stack.length) {
        // 使用 pop 从 stack 中取出并移除值
        const next = stack.pop();
        if (Array.isArray(next)) {
            // 使用 push 送回内层数组中的元素，不会改动原始输入 original input
            stack.push(...next);
        } else {
            res.push(next);
        }
    }
    // 使用 reverse 恢复原数组的顺序
    return res.reverse();
}
flatten(arr1);// [1, 2, 3, 1, 2, 3, 4, 2, 3, 4]
```

5、使用 reduce、concat 和递归无限反嵌套多层嵌套的数组

```
var arr1 = [1,2,3,[1,2,3,4, [2,3,4]]];
```

```

function flattenDeep(arr1) {
  return arr1.reduce((acc, val) => Array.isArray(val) ? acc.concat(flattenDeep(val)) : acc.concat(val), []);
}
flattenDeep(arr1);
// [1, 2, 3, 1, 2, 3, 4, 2, 3, 4]

```

12、怎么判断两个对象相等？（必会）

ES6 中有一个方法判断两个对象是否相等，这个方法判断是两个对象引用地址是否一致

```

let obj1= {
  a: 1
}
let obj2 = {
  a: 1
}
console.log(Object.is(obj1, obj2)) // false
let obj3 = obj1
console.log(Object.is(obj1, obj3)) // true
console.log(Object.is(obj2, obj3)) // false

```

当需求是比较两个对象内容是否一致时就没用了

想要比较两个对象内容是否一致，思路是要遍历对象的所有键名和键值是否都一致：

- 1、判断两个对象是否指向同一内存
- 2、使用 `Object.getOwnPropertyNames` 获取对象所有键名数组
- 3、判断两个对象的键名数组是否相等
- 4、遍历键名，判断键值是否都相等

```

let obj1 = {
  a: 1,
  b: {
    c: 2
  }
}
let obj2 = {
  b: {
    c: 3
  },
  a: 1
}

function isObjectValueEqual(a, b) {
  // 判断两个对象是否指向同一内存，指向同一内存返回 true
  if (a === b) return true
  // 获取两个对象键值数组
  let aProps = Object.getOwnPropertyNames(a)
  let bProps = Object.getOwnPropertyNames(b)
  // 判断两个对象键值数组长度是否一致，不一致返回 false
  if (aProps.length !== bProps.length) return false
  // 遍历对象的键值
  for (let prop in a) {
    // 判断 a 的键值，在 b 中是否存在，不存在，返回 false
    if (b.hasOwnProperty(prop)) {
      // 判断 a 的键值是否为对象，是则递归，不是对象直接判断键值是否相等，不相等返回 false
      if (typeof a[prop] === 'object') {
        if (!isObjectValueEqual(a[prop], b[prop])) return false
      } else if (a[prop] !== b[prop]) {

```

```

        return false
    }
    } else {
        return false
    }
}
return true
}
console.log(isObjectValueEqual(obj1, obj2)) // false

```

13、列举三种强制类型转换和两种隐式类型转换？（必会）

强制

转化成字符串 toString() String()

转换成数字 Number()、 parseInt()、 parseFloat()

转换成布尔类型 Boolean()

隐式

拼接字符串

例子 var str = "" + 18

- * / % ==

14、JavaScript 中怎么获取当前日期的月份？（必会）

方法

JavaScript 中获得当前日期是使用 new Date 这个内置对象的实例，其他一些进阶的操作也是基于这个内置对象的实例。

获取完整的日期（默认格式）：

```
var date = new Date(); // Sat Jul 06 2019 19:59:27 GMT+0800（中国标准时间）
```

获取当前年份：

```
var year = date.getFullYear(); // 2019
```

获取当前月份：

```
var month = date.getMonth() + 1; // 7
```

获取当前日：

```
var day = date.getDay(); // 6
```

获取当前日期（年-月-日）：

```
month = (month > 9) ? month : ("0" + month);
```

```
day = (day < 10) ? ("0" + day) : day;
```

```
var today = year + "-" + month + "-" + day; // 2019-07-06
```

另外的一些操作：

```
date.getFullYear(); // 获取当前年份(2位)
```

```
date.getFullYear(); // 获取完整的年份(4位, 1970-????)
```

```
date.getMonth(); // 获取当前月份(0-11, 0代表1月)
```

```
date.getDate(); // 获取当前日(1-31)
```

```
date.getDay(); // 获取当前星期X(0-6, 0代表星期天)
```

```
date.getTime(); // 获取当前时间(从 1970.1.1 开始的毫秒数)
```

```
date.getHours(); // 获取当前小时数(0-23)
```

```
date.getMinutes(); // 获取当前分钟数(0-59)
```

```
date.getSeconds(); // 获取当前秒数(0-59)
```

```
date.getMilliseconds(); // 获取当前毫秒数(0-999)
```

```
date.toLocaleDateString(); // 获取当前日期
```

```
date.toLocaleTimeString(); // 获取当前时间
date.toLocaleString(); // 获取日期与时间
```

15、什么是类数组（伪数组），如何将其转化为真实的数组？（必会）

伪数组

- 1、具有 length 属性
- 2、按索引方式存储数据
- 3、不具有数组的 push、pop 等方法

伪数组（类数组）：无法直接调用数组方法或期望 length 属性有什么特殊的行为，不具有数组的 push、pop 等方法，但仍可以对真正数据遍历方法来遍历它们。典型的是函数 document.childNodes 之类的，它们返回的 nodeList 对象都属于伪数组

伪数组→真实数组

1. 使用 Array.from() --ES6
2. [].slice.call(eleArr) 或则 Array.prototype.slice.call(eleArr)

示例：

```
let eleArr = document.querySelectorAll('li');
Array.from(eleArr).forEach(function(item) {
    alert(item);
});

let eleArr = document.querySelectorAll('li');
[].slice.call(eleArr).forEach(function(item) {
    alert(item);
});
```

16、如何遍历对象的属性？（必会）

- 1、遍历自身可枚举的属性（可枚举，非继承属性）Object.keys() 方法

该方法会返回一个由一个给定对象的自身可枚举属性组成的数组，数组中的属性名的排列顺序和使用 for...in 遍历该对象时返回的顺序一致（两者的区别是 for...in 还会枚举其原型链上的属性）

```
/**Array 对象**/
var arr = ['a','b','c'];
console.log(Object.keys(arr));
// ['0','1','2']
/**Object 对象**/
var obj = {foo:'bar',baz:42};
console.log(Object.keys(obj));
// ["foo","baz"]
/**类数组 对象 随机 key 排序**/
var anObj = {100:'a',2:'b',7:'c'};
console.log(Object.keys());
// ['2','7','100']
/**getFoo 是一个不可枚举的属性**/
var my_obj = Object.create(
    {}, { getFoo : { value : function () { return this.foo } } }
);
my_obj.foo = 1;
console.log(Object.keys(my_obj)); // ['foo']
```

- 2、遍历自身的所有属性（可枚举，不可枚举，非继承属性）Object.getOwnPropertyNames() 方法，该方法返回一个由指定对象的所有自身属性组成的数组（包括不可枚举属性但不包括

Symbol 值作为名称的属性)

```
var arr = ["a", "b", "c"];
console.log(Object.getOwnPropertyNames(arr).sort()); // ["0", "1", "2", "length"]
// 类数组对象
var obj = { 0: "a", 1: "b", 2: "c"};
console.log(Object.getOwnPropertyNames(obj).sort()); // ["0", "1", "2"]
// 使用 Array.forEach 输出属性名和属性值
Object.getOwnPropertyNames(obj).forEach(function(val, idx, array) {
    console.log(val + " -> " + obj[val]);
});
// 输出
// 0 -> a
// 1 -> b
// 2 -> c
//不可枚举属性
var my_obj = Object.create({}, {
    getFoo: {
        value: function() { return this.foo; },
        enumerable: false
    }
});
my_obj.foo = 1;
console.log(Object.getOwnPropertyNames(my_obj).sort()); // ["foo", "getFoo"]
```

3、遍历可枚举的自身属性和继承属性（可枚举，可继承的属性） for in
遍历对象的属性

```
var obj={
    name: '张三',
    age : '24',
    getAge:function() {
        console.log(this.age);
    }
}
var array ={};
for(var i in obj){
    if(obj.hasOwnProperty(i)&& typeof obj[i] != 'function'){
        array[i] = obj[i];
    }
}
console.log(array);
{name:'张三',age:24}
```

注：hasOwnProperty() 方法判断对象是有某个属性(本身的属性，不是继承的属性)

4、遍历所有的自身属性和继承属性

```
(function () {
    var getAllPropertyNames = function (obj) {
        var props = [];
        do {
            props = props.concat(Object.getOwnPropertyNames(obj));
        } while (obj = Object.getPrototypeOf(obj));
        return props;
    }
    var propertys = getAllPropertyNames(window);
    alert(propertys.length); //276
    alert(propertys.join("\n")); //toString 等
})();
```

17、如何使用原生 JavaScript 给一个按钮绑定两个 onclick 事件？

(必会)

```
var btn=document.getElementById( 'btn' );
//事件监听 绑定多个事件
var btn4 = document.getElementById("btn4");
btn4.addEventListener("click",hello1);
btn4.addEventListener("click",hello2);
function hello1(){
    alert("hello 1");
}
function hello2(){
    alert("hello 2");
}
```

18、JavaScript 中的作用域、预解析与变量声明提升？ (必会)

作用域

就是变量的有效范围。 在一定的空间里可以对数据进行读写操作，这个空间就是数据的作用域

- 1、全局作用域： 最外层函数定义的变量拥有全局作用域，即对任何内部函数来说，都是可以访问的；
- 2、局部作用域： 局部作用域一般只在固定的代码片段内可访问到，而对于函数外部是无法访问的，最常见的例如函数内部。在 ES6 之前，只有函数可以划分变量的作用域，所以在函数的外面无法访问函数内的变量
- 3、块级作用域： 凡是代码块就可以划分变量的作用域，这种作用域的规则就叫块级作用域

块级作用域 函数作用域 词法作用域之间的区别：

- 3.1) 块级作用域和函数作用域描述的是，什么东西可以划分变量的作用域
- 3.2) 词法作用域描述的是，变量的查找规则

之间的关系：

- 1、块级作用域 包含 函数作用域
- 2、词法作用域 与 块级作用域、函数作用域之间没有任何交集， 他们从两个角度描述了作用域的规则

ES6 之前 JavaScript 采用的是函数作用域+词法作用域，ES6 js 采用的是块级作用域+词法作用域

预解析

JavaScript 代码的执行是由浏览器中的 JavaScript 解析器来执行的。JavaScript 解析器执行 JavaScript 代码的时候，分为两个过程：预解析过程和代码执行过程

预解析过程：

1. 把变量的声明提升到当前作用域的最前面，只会提升声明，不会提升赋值
2. 把函数的声明提升到当前作用域的最前面，只会提升声明，不会提升调用
3. 先提升 function，在提升 var

JavaScript 的执行过程：

1	// 案例 1
2	var a = 25;
3	function abc() {
4	alert(a); // undefined
5	var a = 10;
6	}

7	abc();
8	
9	
10	// 案例 2
11	console.log(a); // 25
12	function a() {
13	console.log('aaaaa'); // 不会执行，没人调用 a()
14	}
15	var a = 1;
16	console.log(a); // 1

变量提升

变量提升：定义变量的时候，变量的声明会被提升到作用域的最上面，变量的赋值不会提升

函数提升：JavaScript 解析器首先会把当前作用域的函数声明提前到整个作用域的最前面

1	// 1、-----
2	var num = 10;
3	fun();
4	function fun() {
5	console.log(num);
6	var num = 20;
7	}
8	//2、-----
9	var a = 18;
10	f1(); // 调用执行 22 行，因为后者覆盖前者 f1 函数
11	function f1() {
12	var b = 9;
13	console.log(a);
14	console.log(b);
15	var a = '123';
16	}
17	// 3、-----
18	f1(); // 再次调用下面的 f1 执行
19	console.log(c); // 9
20	console.log(b); // 9
21	console.log(a); // 18
22	function f1() {
23	var a = b = c = 9; // 这里相当于 var a = 9; b = 9; c = 9，所以后 2 个隐
24	藏转换为全局变量了
25	console.log(a); // 9
26	console.log(b); // 9
27	console.log(c); // 9
	}

变量声明提升：

使用 var 关键字定义的变量，被称为变量声明；

函数声明提升的特点是，在函数声明的前面，可以调用这个函数

19、变量提升与函数提升的区别？（必会）

变量提升

简单说就是在 JavaScript 代码执行前引擎会先进行预编译，预编译期间会将变量声明与函数声明提升至其对应作用域的最顶端，函数内声明的变量只会提升至该函数作用域最顶层。当函数内部定义的一个变量与外部相同时，那么函数体内的这个变量就会被上升到最顶端。举例来说：

```
console.log(a); // undefined
var a = 3;
// 预编译后的代码结构可以看做如下运行顺序
var a; // 将变量 a 的声明提升至最顶端，赋值逻辑不提升。
console.log(a); // undefined
a = 3; // 代码执行到原位置即执行原赋值逻辑
```

函数提升

函数提升只会提升函数声明式写法，函数表达式的写法不存在函数提升
函数提升的优先级大于变量提升的优先级，即函数提升在变量提升之上

20、什么是作用域链？（必会）

作用域链

当代码在一个环境中执行时，会创建变量对象的一个作用域链
由子级作用域返回父级作用域中寻找变量，就叫做作用域链
作用域链中的下一个变量对象来自包含环境，也叫外部环境。而再下一个变量对象则来自下一个包含环境，一直延续到全局执行环境。全局执行环境的变量对象始终都是作用域链中的最后一个对象
作用域链前端始终都是当前执行的代码所在环境的变量对象，如果环境是函数，则将其活动对象作为变量对象

21、如何延长作用域链？（必会）

作用域链是可以延长的

延长作用域链：

执行环境的类型只有两种，全局和局部（函数）。但是有些语句可以在作用域链的前端临时增加一个变量对象，该变量对象会在代码执行后被移除

具体来说就是执行这两个语句时，作用域链都会得到加强

1、try - catch 语句的 catch 块；会创建一个新的变量对象，包含的是被抛出的错误对象的声明

2、with 语句。with 语句会将指定的对象添加到作用域链中

22、判断一个值是什么类型有哪些方法？（必会）

方法

1、typeof 运算符

2、instanceof 运算符

instanceof 严格来说是 Java 中的一个双目运算符，用来测试一个对象是否为一个类的实例，用法为：

// 判断 foo 是否是 Foo 类的实例

```
function Foo() {}
```

```
var foo = new Foo();
```

```
console.log(foo instanceof Foo) //true
```

3、Object.prototype.toString 方法

- 在 JavaScript 里使用 typeof 来判断数据类型，只能区分基本类型，即 “Number”，“String”，“undefined”，“Boolean”，“Object”，“Function”，“symbol”（ES6 新增）七种

- 对于数组、null、对象来说，其关系错综复杂，使用 typeof 都会统一返回 “object”

黑马程序员-武汉前端学科出品 www.itheima.com

字符串

- 要想区别对象、数组、函数单纯使用 `typeof` 是不行的, JavaScript 中, 通过 `Object.prototype.toString` 方法, 判断某个对象值属于哪种内置类型。
- 在介绍 `Object.prototype.toString` 方法之前, 我们先把 `toString()` 方法和 `Object.prototype.toString.call()` 方法进行对比
- `toString()` 方法和 `Object.prototype.toString.call()` 方法对比
- `var arr=[1,2];`
- `//直接对一个数组调用 toString()`
- `arr.toString();// "1,2"`
- `//通过 call 指定 arr 数组为 Object.prototype 对象中的 toString 方法的上下文`
- `Object.prototype.toString.call(arr); // "[object Array]"`

23、如何实现数组的随机排序? (必会)

方法一:

```
var arr = [1,2,3,4,5,6,7,8,9,10];
function randSort1(arr) {
    for(var i = 0, len = arr.length; i < len; i++) {
        var rand = parseInt(Math.random()*len);
        var temp = arr[rand];
        arr[rand] = arr[i];
        arr[i] = temp;
    }
    return arr;
}
console.log(randSort1(arr));
```

方法二:

```
var arr = [1,2,3,4,5,6,7,8,9,10];
function randSort2(arr) {
    var mixedArray = [];
    while(arr.length > 0) {
        var randomIndex = parseInt(Math.random()*arr.length);
        mixedArray.push(arr[randomIndex]);
        arr.splice(randomIndex, 1);
    }
    return mixedArray;
}
console.log(randSort2(arr));
```

方法三:

```
var arr = [1,2,3,4,5,6,7,8,9,10];
arr.sort(function() {
    return Math.random() - 0.5;
});
console.log(arr);
```

24、src 和 href 的区别是? (了解)

区别

`src (source)` 指向外部资源的位置, 指向的内容将会嵌入到文档中当前标签所在位置;

在请求 `src` 资源时会将其指向的资源下载并应用到文档中, 如 JavaScript 脚本, `img` 图片和 `iframe` 等元素

当浏览器解析到该元素时, 会暂停其他资源的下载和处理, 直到将该资源加载、编译、执

行完毕，类似于将所指向资源嵌入当前标签内

href(hypertext reference/超文本引用)指向网络资源所在位置，建立和当前元素(锚点)或当前文档(链接)之间的链接，如果我们在文档中添加<link href="common.css"rel="stylesheet"/>那么浏览器会识别该文档为 CSS 文件，就会并行下载资源并且不会停止对当前文档的处理

WebAPI

1、什么是 dom? (必会)

什么是 dom

- 1、DOM 是 W3C (万维网联盟) 的标准
- 2、DOM 定义了访问 HTML 和 XML 文档的标准

什么是 W3C

1、“W3C 文档对象模型 (DOM) 是中立于平台和语言的接口，它允许程序和脚本动态地访问和更新文档的内容、结构和样式。”

- 2、W3C DOM 标准被分为 3 个不同的部分
 - 2.1) 核心 DOM - 针对任何结构化文档的标准模型
 - 2.2) XML DOM - 针对 XML 文档的标准模型
 - 2.3) HTML DOM - 针对 HTML 文档的标准模型

备注：DOM 是 Document Object Model (文档对象模型) 的缩写

2、dom 节点的 Attribute 和 Property 有何区别? (必会)

1、什么是 Property

每个 DOM 节点都是一个 object 对象，有自己的 property 和 method

原则上 property 应该仅供 js 操作，不会出现在 html 中 (默认属性除外：id/src/href/className/dir/title/lang 等)，和其他 js object 一样，自定义的 property 也会出现在 object 的 for...in 遍历中

2、什么是 Attribute

attribute 出现在 dom 中，js 提供了 getAttribute/setAttribute 等方法来获取和改变它的值，最后作用于 html 中，可以影响 innerHTML 获取的值。可以通过访问 dom 节点的 attributes 属性来获取 改节点的所有的 attribute。(在 IE<9 中,attribute 获取和改变的实际上是 property。)

3、两者之间的区别是

- 3.1) 自定义的 Property 与 Attribute 不同步, 不相等
- 3.2) 非自定义的 DOM property 与 attributes 是有条件同步的
- 3.3) 非自定义的属性(id/src/href/name/value 等)，通过 setAttribute 修改其特性值可以同步作用到 property 上，而通过 .property 修改属性值有的(value)时候不会同步到 attribute 上，即不会反应到 html 上(除以下几种情况，非自定义属性 在二者之间是同步的)。

3、dom 结构操作怎样添加、移除、移动、复制、创建和查找节点? (必会)

1、创建新节点

```
createDocumentFragment() //创建一个 DOM 片段
createElement() //创建一个具体的元素
createTextNode() //创建一个文本节点
```

2、添加、移除、替换、插入

```
appendChild()
removeChild()
replaceChild()
insertBefore() //并没有 insertAfter()
```

3、查找

```
getElementsByTagName() //通过标签名称
getElementsByName() //通过元素的 Name 属性的值(IE 容错能力较强,
会得到一个数组, 其中包括 id 等于 name 值的)
getElementById() //通过元素 Id, 唯一性
```

4、dom 事件模型? (必会)

DOM 事件模型。

DOM 事件模型分为两种: 事件捕获和事件冒泡。

事件捕获以点击事件为例, 同类型事件会由 根—>目标的祖先素—>目标的父元素—>目标元素

事件冒泡和事件捕获截然相反。从内到外依次触发: 目标元素—>目标元素的父元素—>父元素的父元素—>根

事件传播

事件捕获和事件冒泡都有事件传播阶段, 传播阶段就是事件从触发开始到结束的过程。

优先级: 先捕获, 再冒泡。

两种传播方式的来源: W3C 推行 DOM2 级事件之前网景和 IE 在打架, 网景用的事件传播方式是捕获, IE 用的事件传播方式是冒泡

5、什么是事件冒泡, 它是如何工作的? 如何阻止事件冒泡、默认为? (必会)

1、什么是事件冒泡, 他是如何工作的

在一个对象上触发某类事件 (比如单击 onclick 事件), 这个事件会向这个对象的父级对象传播, 从里到外, 直至它被处理 (父级对象所有同类事件都将被激活), 或者它到达了对象层次的最顶层, 即 document 对象 (有些浏览器是 window)

2、阻止事件冒泡的方法

2.1) w3c 方法是: `event.stopPropagation()`; 事件处理过程中, 阻止冒泡事件, 但不会阻止默认行为 (跳转至超链接)

2.2) IE 则是使用 `event.cancelBubble = true` 阻止事件冒泡

2.3) `return false`; jq 里面事件处理过程中, 阻止冒泡事件, 也阻止默认行为 (不跳转超链接)

封装方法:

```
function bubbles(e) {
  var ev = e || window.event;
  if(ev && ev.stopPropagation) {
    //非 IE 浏览器
    ev.stopPropagation();
  } else {
```

```

//IE 浏览器(IE11 以下)
ev.cancelBubble = true;
}
console.log("最底层盒子被点击了")
}
}

```

阻止默认行为:

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

封装:

```

//假定有链接<a href="http://caibaojian.com/" id="testA" >caibaojian.com</a>
var a = document.getElementById("testA");
a.onclick =function(e){
    if(e.preventDefault){
        e.preventDefault();
    }else{
        window.event.returnValue == false;
    }
}
}

```

6、JavaScript 动画和 CSS3 动画有什么区别? (必会)

1、CSS 动画

优点:

1.1) 浏览器可以对动画进行优化。

1.1.1) 浏览器使用 `requestAnimationFrame` 比起 `setTimeout`, `setInterval` 设置动画的优势主要是:

- 1) `requestAnimationFrame` 会把每一帧中的所有 DOM 操作集中起来, 在一次重绘或回流中就完成, 并且重绘或回流的时间间隔紧紧跟随浏览器的刷新频率, 一般来说, 这个频率为每秒 60 帧。
- 2) 在隐藏或不可见的元素中 `requestAnimationFrame` 不会进行重绘或回流, 这当然就意味着更少的 `cpu`, `gpu` 和内存使用量。

1.1.2) 强制使用硬件加速 (通过 GPU 来提高动画性能)

1.2) 代码相对简单, 性能调优方向固定

1.3) 对于帧速表现不好的低版本浏览器, CSS3 可以做到自然降级, 而 JS 则需要撰写额外代码

缺点:

1.1) 运行过程控制较弱, 无法附加事件绑定回调函数。CSS 动画只能暂停, 不能在动画中寻找一个特定的时间点, 不能在半路反转动画, 不能变换时间尺度, 不能在特定的位置添加回调函数或是绑定回放事件, 无进度报告。

1.2) 代码冗长。想用 CSS 实现稍微复杂一点动画, 最后 CSS 代码都会变得非常笨重。

2、JS 动画

优点:

2.1) JavaScript 动画控制能力很强, 可以在动画播放过程中对动画进行控制: 开始、暂停、回放、终止、取消都是可以做到的。

2.2) 动画效果比 css3 动画丰富, 有些动画效果, 比如曲线运动, 冲击闪烁, 视差滚动效果, 只有 JavaScript 动画才能完成。

2.3) CSS3 有兼容性问题, 而 JS 大多时候没有兼容性问题。

缺点:

2.1) JavaScript 在浏览器的主线程中运行, 而主线程中还有其它需要运行的 JavaScript 脚本、样式计算、布局、绘制任务等, 对其干扰导致线程可能出现阻塞, 从而造成丢帧的情况。

2.2) 代码的复杂度高于 CSS 动画

3、css 动画和 js 动画的差异

- 3.1) 代码复杂度, js 动画代码相对复杂一些。
 - 3.2) 动画运行时, 对动画的控制程度上, js 能够让动画, 暂停, 取消, 终止, css 动画不能添加事件。
 - 3.3) 动画性能看, js 动画多了一个 js 解析的过程, 性能不如 css 动画好。
- 总结:
简单的状态切换, 不需要中间过程控制, css 动画是优选方案。
复杂的状态的 APP。应该使用 js 动画

7、event 对象的常见应用? (必会)

- 1、event.preventDefault(); // 阻止默认行为, 阻止 a 链接默认的跳转行为
- 2、event.stopPropagation(); // 阻止冒泡
- 3、event.stopImmediatePropagation(); // 按钮绑定了 2 个响应函数, 依次注册 a, b 两个事件, 点击按钮, a 事件中加 event.stopImmediatePropagation() 就能阻止 b 事件
- 4、event.currentTarget // 早期的 ie 不支持, 当前绑定的事件
- 5、event.target

8、通用事件绑定/ 编写一个通用的事件监听函数? (必会)

```
function bindEvent(elem, type, selector, fn) {
  if (fn == null) {
    fn = selector;
    selector = null;
  }
  elem.addEventListener(type, function(e) {
    var target;
    if (selector) {
      target = e.target;
      if (target.matches(selector)) {
        fn.call(target, e);
      }
    } else {
      fn(e);
    }
  })
}

// 使用代理
var div1 = document.getElementById('div1');
bindEvent(div1, 'click', 'a', function(e) {
  console.log(this.innerHTML);
});

// 不使用代理
var a = document.getElementById('a1');
bindEvent(div1, 'click', function(e) {
  console.log(a.innerHTML);
});
```

9、DOM 和 BOM 的区别 (必会)

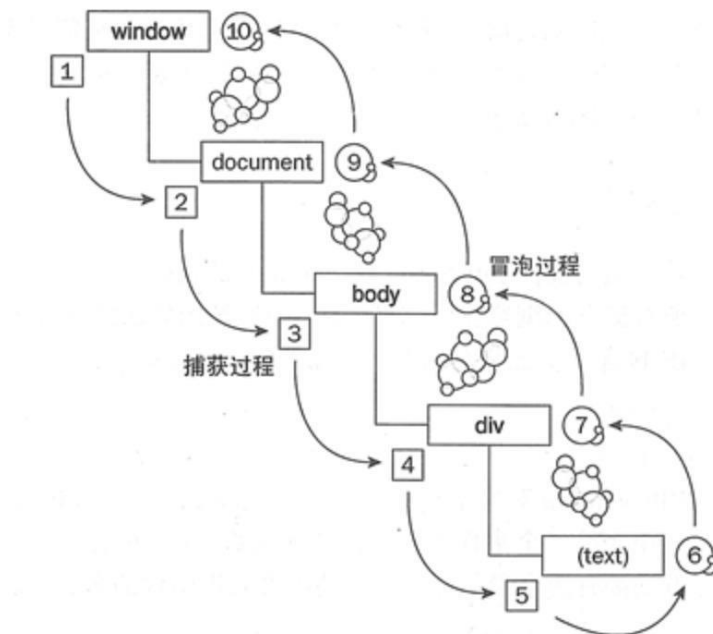
- 1、BOM

- 1.1) BOM 是 Browser Object Model 的缩写，即浏览器对象模型。
- 1.2) BOM 没有相关标准。
- 1.3) BOM 的最根本对象是 window
- 2、DOM
 - 2.1) DOM 是 Document Object Model 的缩写，即文档对象模型。
 - 2.2) DOM 是 W3C 的标准。
 - 2.3) DOM 最根本对象是 document（实际上是 window.document）

10、事件三要素（必会）

- 1、事件源、就是你点的那个 div，触发的对象
- 2、事件类型、表示动作，比如点击、滑过等
- 3、事件处理函数（事件处理程序）、表示结果，比如点开关跳转到另一个页面

11、事件执行过程（必会）



事件捕获过程：当我们点击 TEXT 时，首先是 window->document->body->div->text. 这个过程称为事件捕获，W3C 浏览器的标准执行流程。

事件冒泡过程：text->div->body->document->window. 这个过程称为事件冒泡。IE 浏览器只支持冒泡，不支持捕获。W3C 浏览器先执行捕获，后执行冒泡

12、获取元素位置（必会）

- 1、通过元素的 offsetLeft 和 offsetTop

dom 元素的 offsetLeft、offsetTop 指的是元素相对于其 offsetParent 指定的坐标来说的。

offsetParent：是指当前元素最近的经过定位的父级元素，如果没有则一直向上直至 body。注意当前元素为 fixed 时，其 offsetParent 的值为 null

拓展：

offsetWidth/offsetHeight: width+padding+border

clientLeft/clientTop: 表示内容区域的左上角相对于整个元素左上角的位置（包括边框）
 // border 值
 clientWidth/clientHeight: width+padding
 scrollWidth: 获取对象的滚动宽度
 scrollHeight: 获取对象的滚动高度。
 scrollLeft: 设置或获取位于对象左边界和窗口中目前可见内容的最左端之间的距离
 scrollTop: 设置或获取位于对象最顶端和窗口中可见内容的最顶端之间的距离
 window.screen.availHeight/window.screen.availWidth: 浏览器去除上方工具栏和下方菜单栏可用宽高
 window.screen.height/window.screen.width: 屏幕宽高
 2、event.clientX 和 event.clientY
 事件相对于浏览器窗口的水平和垂直距离
 3、getBoundingClientRect
 方法返回一个矩形对象，包含四个属性：left、top、right 和 bottom。分别表示元素各边与页面上边和左边的距离

13、封装运动函数（必会）

```

/*
obj 指的是 DOM 对象
- json 指的是 CSS 样式
例 startMove(oDiv, {width:100,height:100},function() {})
*/
function startMove(obj, json, fnEnd) {
  clearInterval(obj.timer); //先清除之前的定时器
  obj.timer = setInterval(function() {
    var bStop = true; // 假设所有的值都到了
    for( var attr in json ){ //遍历 json 属性
      var cur = (attr == 'opacity') ?
Math.round(parseFloat(getStyle(obj, attr))*100) : parseInt(getStyle(obj, attr)); //对 opacity
特殊处理
      var speed = (json[attr] - cur)/6;
      speed = speed > 0 ? Math.ceil(speed) : Math.floor(speed); //speed 数字
转化，防止不能到达目标的 bug
      if( cur != json[attr] ) bStop = false; //如果没有达到目标值，则 bStop 设为
false;
      if(attr == 'opacity'){
        obj.style.filter = 'alpha(opacity=' + (cur + speed) + ')';
        obj.style.opacity = (cur + speed)/100;
      }else{
        obj.style[attr] = cur + speed + 'px';
      }
    }
    if(bStop) {
      clearInterval(obj.timer);
      if(fnEnd) fnEnd(); //执行回调函数
    }
  }, 30);
}

function getStyle(obj, name) {
  return obj.currentStyle ? obj.currentStyle[name] :
window.getComputedStyle(obj, null)[name]; //浏览器兼容性处理，注意 getComputedStyle 为只读属性
}

function getByClass(oParent, sClass) {

```

```

var aEle = oParent.getElementsByTagName('*');
var aResult = [];
var re = new RegExp('\\b' + sClass + '\\b','i');
for(var i=0; i<aEle.length;i++){
    if(re.test(aEle[i].className)) aResult.push(aEle[i]);
}
return aResult;
}

```

14、绑定事件和解除事件的区别（必会）

1、事件绑定

定义：一个事件可以加多次，且不会覆盖

2、绑定方法

2.1) attachEvent ('on+事件名', 函数名) 这个只兼容 ie 6-8

2.2) addEventListener (事件名, 函数名, false) 支持 ie9+ chrom firefox

绑定事件的封装

```

function addEvent(obj, sEv, fn) {
    if(obj.addEventListener) {
        obj.addEventListener(sEv, fn, false);
    } else {
        obj.attachEvent('on'+sEv, fn);
    }
};

```

解除绑定事件的封装

```

function removeEvent(obj, sEv, fn) {
    if(obj.removeEventListener) {
        obj.removeEventListener(sEv, fn, false);
    } else {
        obj.detachEvent('on'+sEv, fn);
    }
}

```

15、谈谈事件委托的理解？（必会）

JavaScript 事件代理则是一种简单的技巧，把事件处理器添加到一个上级元素上，这样就避免了把事件处理器添加到多个子级元素上。这主要得益于浏览器的事件冒泡机制。

优点：

- 1、减少事件注册，节省内存。
- 2、在 table 上代理所有 td 的 click 事件。
- 3、在 ul 上代理所有 li 的 click 事件。
- 4、简化了 dom 节点更新时，相应事件的更新。
- 5、不用在新添加的 li 上绑定 click 事件。
- 6、当删除某个 li 时，不用移解绑上面的 click 事件。

缺点：

- 1、事件委托基于冒泡，对于不冒泡的事件不支持
- 2、层级过多，冒泡过程中，可能会被某层阻止掉。
- 3、理论上委托会导致浏览器频繁调用处理函数，虽然很可能不需要处理。所以建议就近委托，比如在 table 上代理 td，而不是在 document 上代理 td。
- 4、把所有事件都用代理就可能会出现事件误判。比如，在 document 中代理了所有 button 的 click 事件，另外的人在引用改 js 时，可能不知道，造成单击 button 触发了两个 click 事件

16、JavaScript 中的定时器有哪些？他们的区别及用法是什么？（必会）

1、JavaScript 中的定时器有以下几种

1) `setTimeout()` 方法用于在指定的毫秒数后调用函数或计算表达式。

2) `setInterval()` 方法可按照指定的周期（以毫秒计）来调用函数或计算表达式。

`setInterval()` 方法会不停地调用函数，直到 `clearInterval()` 被调用或窗口被关闭。由 `setInterval()` 返回的 ID 值可用作 `clearInterval()` 方法的参数。

`setTimeout` 也叫定时器

`setInterval` 也叫计时器

17、比较 `attachEvent` 和 `addEventListener`?（必会）

`attachEvent` 只能在 IE 浏览器给标签绑定事件，可以多次绑定

语法: `Element.attachEvent(Etype, EventName)`

参数 `Element`: 要为该元素动态添加一个事件

`Etype`: 指定事件类型。比如: `onclick, onkeyup, onmousemove` 等

`EventName`: 指定事件名称。也就是你写好的函数

`addEventListener` W3C 标准，除 IE 浏览器使用，它给标签绑定事件

语法: `Element.addEventListener(Etype, EventName, boole)`

`Etype`: 事件类型。比如: `click, keyup, mousemove`。注意使用 `addEventListener` 绑定事件时，设置参数事件类型时不必写 `on`。否则会出错

`EventName`: 要绑定事件的名称。也就是你写好的函数

`boole`: 该参数是一个布尔值：默认 `false`。 `false` 代表冒泡阶段执行， `true` 代表捕获阶段执行

18、`document.write` 和 `innerHTML` 的区别？（必会）

`document.write` 是直接写入到页面的内容流，如果在写之前没有调用 `document.open`，浏览器会自动调用 `open`。每次写完关闭之后重新调用该函数，会导致页面被重写

`innerHTML` 则是 DOM 页面元素的一个属性，代表该元素的 `html` 内容。

`innerHTML` 将内容写入某个 DOM 节点，不会导致页面全部重绘

`innerHTML` 很多情况下都优于 `document.write`，其原因在于其允许更精确的控制要刷新页面的那一个部分

19、什么是 `window` 对象？什么是 `document` 对象？（必会）

1、什么是 `window` 对象

简单来说，`document` 是 `window` 的一个对象属性。

`Window` 对象表示浏览器中打开的窗口。

如果文档包含框架（`frame` 或 `iframe` 标签），浏览器会为 `HTML` 文档创建一个 `window` 对象，并为每个框架创建一个额外的 `window` 对象。

所有的全局函数和对象都属于 `Window` 对象的属性和方法。

它是一个顶层对象，而不是另一个对象的属性，即浏览器的窗口。

属性

- defaultStatus 缺省的状态条消息
- document 当前显示的文档(该属性本身也是一个对象)
- frame 窗口里的一个框架(FRAME) (该属性本身也是一个对象)
- frames array 列举窗口的框架对象的数组, 按照这些对象在文档中出现的顺序列出(该属性本身也是一个对象)
- history 窗口的历史列表(该属性本身也是一个对象)
- length 窗口内的框架数
- location 窗口所显示文档的完整(绝对)URL(该属性本身也是一个对象) 不要把它与如 document.location 混淆, 后者是当前显示文档的 URL。用户可以改变 window.location(用另一个文档取代当前文档), 但却不能改变 document.location (因为这是当前显示文档的位置)
- name 窗口打开时, 赋予该窗口的名字
- opener 代表使用 window.open 打开当前窗口的脚本所在的窗口(这是 Netscape Navigator 3.0beta 3 所引入的一个新属性)
- parent 包含当前框架的窗口的同义词。frame 和 window 对象的一个属性
- self 当前窗口或框架的同义词
- status 状态条中的消息
- top 包含当前框架的最顶层浏览器窗口的同义词
- window 当前窗口或框架的同义词, 与 self 相同

方法

- alert() 打开一个 Alert 消息框
- clearTimeout() 用来终止 setTimeout 方法的工作
- close() 关闭窗口
- confirm() 打开一个 Confirm 消息框, 用户可以选择 OK 或 Cancel, 如果用户单击 OK, 该方法返回 true, 单击 Cancel 返回 false
- blur() 把焦点从指定窗口移开(这是 Netscape Navigator 3.0 beta 3 引入的新方法)
- focus() 把指定的窗口带到前台(另一个新方法)
- open() 打开一个新窗口
- prompt() 打开一个 Prompt 对话框, 用户可向该框键入文本, 并把键入的文本返回到脚本
- setTimeout() 等待一段指定的毫秒数时间, 然后运行指令事件处理程序事件处理程序
- onload() 页面载入时触发
- onunload() 页面关闭时触发

2、什么是 document 对象

[document 对象]

该对象是 window 和 frames 对象的一个属性, 是显示于窗口或框架内的一个文档。

属性

- alinkColor 活动链接的颜色(ALINK)
- anchor 一个 HTML 锚点, 使用标记创建(该属性本身也是一个对象)
- anchors array 列出文档锚点对象的数组()(该属性本身也是一个对象)
- bgColor 文档的背景颜色(BGCOLOR)
- cookie 存储于 cookie.txt 文件内的一段信息, 它是该文档对象的一个属性
- fgColor 文档的文本颜色(<BODY>标记里的 TEXT 特性)
- form 文档中的一个窗体(<FORM>)(该属性本身也是一个对象)
- forms array 按照其出现在文档中的顺序列出窗体对象的一个数组(该属性本身也是一个对象)
- lastModified 文档最后的修改日期
- linkColor 文档的链接的颜色, 即<BODY>标记中的 LINK 特性(链接到用户没有观察到的文档)
- link 文档中的一个标记(该属性本身也是一个对象)

links array 文档中 link 对象的一个数组,按照它们出现在文档中的顺序排列(该属性本身也是一个对象)

location 当前显示文档的 URL。用户不能改变 document.location(因为这是当前显示文档的位置)。但是,可以改变 window.location (用其它文档取代当前文档)window.location 本身也是一个对象,而 document.location 不是对象

referrer 包含链接的文档的 URL,用户单击该链接可到达当前文档

title 文档的标题((TITLE>))

vlinkColor 指向用户已观察过的文档的链接文本颜色,即<BODY>标记的 VLINK 特性方法

clear 清除指定文档的内容

close 关闭文档流

open 打开文档流

write 把文本写入文档

writeln 把文本写入文档,并以换行符结尾

区别:

- 1、window 指窗体。document 指页面。document 是 window 的一个子对象。
- 2、用户不能改变 document.location(因为这是当前显示文档的位置)。但是,可以改变 window.location (用其它文档取代当前文档)window.location 本身也是一个对象,而 document.location 不是对象

20、Js 拖动的原理? (必会)

js 的拖拽效果主要用到以下三个事件:

mousedown 鼠标按下事件

mousemove 鼠标移动事件

mouseup 鼠标抬起事件

当点击 dom 的时候,记录当前鼠标的坐标值,也就是 x、y 值,以及被拖拽的 dom 的 top、left 值,而且还要在鼠标按下的回调函数里添加鼠标移动的事件:

```
document.addEventListener("mousemove", moving, false)
```

和添加鼠标抬起的事件

```
document.addEventListener("mouseup",function() {
```

```
document.removeEventListener("mousemove", moving, false);}, false);
```

这个抬起的事件是为了解除鼠标移动的监听,因为只有在鼠标按下才可以拖拽,抬起就停止不会移动了。

当鼠标按下鼠标移动的时候,记录移动中的 x、y 值,那么这个被拖拽的 dom 的 top 和 left 值就是:

top=鼠标按下时记录的 dom 的 top 值+ (移动中的 y 值 - 鼠标按下时的 y 值)

left=鼠标按下时记录的 dom 的 left 值+ (移动中的 x 值 - 鼠标按下时的 x 值);

```

</div>
<div id="drabble"></div>
<script>
  window.onload = function() {
    var dom = document.getElementById("drabble");
    dom.addEventListener(
      "mousedown",
      function(event) {
        var x = event.clientX;
        var y = event.clientY;
        var marginLeft = parseInt(dom.offsetLeft);
        var marginTop = parseInt(dom.offsetTop);
        function moving(e) {
          var movedX = e.clientX - x;
          var movedY = e.clientY - y;
          dom.style.marginLeft = marginLeft + movedX + "px";
          dom.style.marginTop = marginTop + movedY + "px";
        }
        document.addEventListener("mousemove", moving, false);
        document.addEventListener("mouseup", function() {
          document.removeEventListener("mousemove", moving, false);
        }, false);
      },
      false
    );
  };
</script>

```

21、描述浏览器的渲染过程，DOM 树和渲染树的区别（必会）

1、浏览器的渲染过程：

解析 HTML 构建 DOM (DOM 树)，并行请求 css/image/js

CSS 文件下载完成，开始构建 CSSOM (CSS 树)

CSSOM 构建结束后，和 DOM 一起生成 Render Tree (渲染树)

布局 (Layout)：计算出每个节点在屏幕中的位置

显示 (Painting)：通过显卡把页面画到屏幕上

2、DOM 树 和 渲染树 的区别

DOM 树与 HTML 标签一一对应，包括 head 和隐藏元素

渲染树不包括 head 和隐藏元素，大段文本的每一个行都是独立节点，每一个节点都有对应的 css 属性

22、如何最小化重绘 (repaint) 和回流 (reflow)（必会）

什么是重绘 Repaint 和重排（回流 reflow）

重绘：当元素的一部分属性发生改变，如外观、背景、颜色等不会引起布局变化，只需要浏览器根据元素的新属性重新绘制，使元素呈现新的外观叫做重绘。

重排（回流）：当 render 树中的一部分或者全部因为大小边距等问题发生改变而需要 DOM 树重新计算的过程

重绘不一定需要重排（比如颜色的改变），重排必然导致重绘（比如改变网页位置）

方法：

- 1、需要要对元素进行复杂的操作时，可以先隐藏 (display: "none")，操作完成后再显示
- 2、需要创建多个 DOM 节点时，使用 DocumentFragment 创建完后一次性的加入 document 缓存 Layout 属性值，如：var left = elem.offsetLeft; 这样，多次使用 left 只产生一次回流
- 3、尽量避免用 table 布局（table 元素一旦触发回流就会导致 table 里所有的其它元素回流）
- 4、避免使用 css 表达式 (expression)，因为每次调用都会重新计算值（包括加载页面）
- 5、尽量使用 css 属性简写，如：用 border 代替 border-width, border-style, border-color
- 6、批量修改元素样式：elem.className 和 elem.style.cssText 代替 elem.style.xxx

23、Js 延迟加载的方式有哪些？（了解）

js 的延迟加载有助与提高页面的加载速度

JS 延迟加载，也就是等页面加载完成之后再加载 JavaScript 文件

一般有以下几种方式：[使用 setTimeout 延迟方法](#)

[让 JS 最后加载](#)

1、defer 属性

用途：表明脚本在执行时不会影响页面的构造。也就是说，脚本会被延迟到整个页面都解析完毕之后再执行

在<script>元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <script src="test1.js" defer="defer"></script>
5     <script src="test2.js" defer="defer"></script>
6 </head>
7 <body>
8 <!-- 这里放内容 -->
9 </body>
10 </html>
```

说明：虽然<script>元素放在了<head>元素中，但包含的脚本将延迟浏览器遇到</html>标签后再执行

HTML5 规范要求脚本按照它们出现的先后顺序执行。在现实当中，延迟脚本并不一定会按照顺序执行

defer 属性只适用于外部脚本文件。支持 HTML5 的实现会忽略嵌入脚本设置的 defer 属性

2、async 属性

HTML5 为<script>标签定义了 async 属性。与 defer 属性类似，都用于改变处理脚本的行为。同样，只适用于外部脚本文件。标签定义了 async 属性。与 defer 属性类似，都用于改变处理脚本的行为。同样，只适用于外部脚本文件

目的：不让页面等待脚本下载和执行，从而异步加载页面其他内容。异步脚本一定会在页面 load 事件前执行。不能保证脚本会按顺序执行

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <script src="test1.js" async></script>
5     <script src="test2.js" async></script>
6 </head>
7 <body>
8 <!-- 这里放内容 -->
9 </body>
10 </html>
```

async 和 defer 一样，都不会阻塞其他资源下载，所以不会影响页面的加载。

缺点：不能控制加载的顺序

3、动态创建 DOM 方式

```

1 //这些代码应被放置在</body>标签前(接近HTML文件底部)
2 <script type="text/javascript">
3     function downloadJSAtOnload() {
4         varelement = document.createElement("script");
5         element.src = "defer.js";
6         document.body.appendChild(element);
7     }
8     if (window.addEventListener)
9         window.addEventListener("load",downloadJSAtOnload, false);
10    else if (window.attachEvent)
11        window.attachEvent("onload",downloadJSAtOnload);
12    else
13        window.onload =downloadJSAtOnload;
14 </script>

```

4、使用 jQuery 的 getScript() 方法

```

1 $.getScript("outer.js",function(){//回调函数，成功获取文件后执行的函数
2     console.log("脚本加载完成")
3 });

```

5、使用 setTimeout 延迟方法的加载时间

延迟加载 js 代码，给网页加载留出更多时间

```

1 <script type="text/javascript" >
2     function A(){
3         $.post("/lord/login",{name:username,pwd:password},function(){
4             alert("Hello");
5         });
6     }
7     $(function (){
8         setTimeout('A()', 1000); //延迟1秒
9     })
10 </script>

```

6、让 JS 最后加载

把 js 外部引入的文件放到页面底部，来让 js 最后引入，从而加快页面加载速度

例如引入外部 js 脚本文件时，如果放入 html 的 head 中，则页面加载前该 js 脚本就会被加载入页面，而放入 body 中，则会按照页面从上倒下的加载顺序来运行 JavaScript 的代码

所以我们可以把 js 外部引入的文件放到页面底部，来让 js 最后引入，从而加快页面加载速度

```

1 //这些代码应被放置在</body>标签前(接近HTML文件底部)
2 <script type="text/javascript">
3     function downloadJSAtOnload() {
4         var element = document.createElement("script");
5         element.src = "defer.js";
6         document.body.appendChild(element);
7     }
8     if (window.addEventListener)
9         window.addEventListener("load", downloadJSAtOnload, false);
10    else if (window.attachEvent)
11        window.attachEvent("onload", downloadJSAtOnload);
12    else window.onload = downloadJSAtOnload;
13 </script>

```

这段代码意思等到整个文档加载完后，再加载外部文件“defer.js”。

使用此段代码的步骤：

6.1) 复制上面代码

6.2) 粘贴代码到 HTML 的标签前（靠近 HTML 文件底部）

6.3) 修改“defer.js”为你的外部 JS 文件名

6.4) 确保你文件路径是正确的。例如：如果你仅输入“defer.js”，那么“defer.js”文件一定与 HTML 文件在同一文件夹下。

注意：

这段代码直到文档加载完才会加载指定的外部 js 文件。因此，不应该把那些页面正常加载需要依赖的 javascript 代码放在这里。而应该将 JavaScript 代码分成两组。一组是

因页面需要而立即加载的 javascript 代码，另外一组是在页面加载后进行操作的 javascript 代码(例如添加 click 事件或其他东西)。这些需等到页面加载后再执行的 JavaScript 代码，应放在一个外部文件，然后再引进来

在元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行

24、IE 与标准事件模型有哪些差别？（了解）

1. 添加事件

DOM 事件模型 - addEventListener

addEventListener(eventType, handler, useCapture)

eventType 不带有 on 字符串；

handler 参数是一个事件句柄，这个函数或方法带有一个事件对象参数；

useCapture 参数决定了事件句柄触发在何种事件传播阶段，如果 useCapture 为 true 则为捕获阶段，反之则为冒泡阶段。

IE 事件模型 - attachEvent

attachEvent(eventType, handler)

eventType 带 on 字符串；

handler 参数是一个事件句柄，这个函数或方法带有一个事件对象参数；

2. 事件过程

DOM 事件模型包含捕获阶段和冒泡阶段，IE 事件模型只包含冒泡阶段；

DOM 事件模型可使用 e.stopPropagation() 来阻止事件流

JavaScript 高级

1、typeof 和 instanceof 区别（必会）

在 javascript 中，判断一个变量的类型可以用 typeof

1、数字类型，typeof 返回的值是 number。比如说：typeof(1)，返回值是 number

2、字符串类型，typeof 返回的值是 string。比如 typeof(“123”) 返回值是 string

3、布尔类型，typeof 返回的值是 boolean。比如 typeof(true) 返回值是 boolean

4、对象、数组、null 返回的值是 object。比如 typeof(window)，typeof(document)，typeof(null) 返回的值都是 object

5、函数类型，返回的值是 function。比如：typeof(eval)，typeof(Date) 返回的值都是 function。

6、不存在的变量、函数或者 undefined，将返回 undefined。比如：typeof(abc)、typeof(undefined) 都返回 undefined

使用 typeof 运算符无论引用的是什么类型的对象，它都返回” object ”

运算符 instanceof 来解决这个问题。用于判断某个对象是否被另一个函数构造

2、js 使用 typeof 能得到的哪些类型？（必会）

typeof 只能区分值类型

typeof undefined // undefined

typeof null // object

```
typeof console.log // function
typeof NaN // number
```

3、解释一下什么是回调函数，并提供一个简单的例子？（必会）

回调函数就是一个通过调用的函数。如果你把函数的（地址）作为给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的

案例：

```
#include<stdio.h>
//callbackTest.c
//1. 定义函数 onHeight（回调函数）
//@onHeight 函数名
//@height 参数
//@context 上下文
void onHeight(double height, void *context)
{
    printf("current height is %lf", height);
}
//2. 定义 onHeight 函数的原型
//@CallbackFun 指向函数的指针类型
//@height 回调参数，当有多个参数时，可以定义一个结构体
//@context 回调上下文，在 C 中一般传入 nullptr，在 C++ 中可传入对象指针
typedef void (*CallbackFun)(double height, void *context);
//定义全局指针变量
CallbackFun m_pCallback;
//定义注册回调函数
void registHeightCallback(CallbackFun callback, void *context)
{
    m_pCallback = callback;
}
//定义调用函数
void printHeightFun(double height)
{
    m_pCallback(height, NULL);
}
//main 函数
int main()
{
    //注册回调函数 onHeight
    registHeightCallback(onHeight, NULL);
    //打印 height
    double h = 99;
    printHeightFun(99);
}
```

4、什么是闭包？（必会）

定义：

一个作用域可以访问另外一个函数内部的局部变量，或者说一个函数（子函数）访问另一个函数（父函数）中的变量。此时就会有闭包产生，那么这个变量所在的函数我们就称之为闭包函数。

```
function aaa() {
```

```

        var a = 0;
        return function () {
            alert(a++);
        };
    }
    var fun = aaa();
    fun();    //1

```

优缺点:

闭包的主要作用：延伸了变量的作用范围，因为闭包函数中的局部变量不会等着闭包函数执行完就销毁，因为还有别的函数要调用它，只有等着所有的函数都调用完了他才会销毁。闭包会造成内存泄漏，如何解决：用完之后手动释放

详解:

闭包不仅仅可以实现函数内部的作用域访问这个函数中的局部变量，还可以实现全局作用域或者是别的地方的作用域也可以访问到函数内部的局部变量，实现方法就是 return 了一个函数。所以 return 函数也是我们实现闭包的一个主要原理，因为返回的这个函数本身就是我们函数内部的一个子函数，所以子函数是可以访问父函数里面的局部变量的，所以返回完毕之后，外面的函数一调用，就会回头调用返回的这个函数，所以就可以拿到这个子函数对应的父函数里面的局部变量。

注意:

- 1、由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在 IE 中可能导致内存泄露。解决方法是，在退出函数之前，将不使用的局部变量全部删除。
- 2、闭包会在父函数外部，改变父函数内部变量的值。所以，如果你把父函数当作对象（object）使用，把闭包当作它的公用方法（Public Method），把内部变量当作它的私有属性（private value），这时一定要小心，不要随便改变父函数内部变量的值。

5、什么是内存泄漏（必会）

内存泄露是指：内存泄漏也称作“存储渗漏”，用动态存储分配函数动态开辟的空间，在使用完毕后未释放，结果导致一直占据该内存单元。直到程序结束。（其实说白了就是该内存空间使用完毕之后未回收）即所谓内存泄漏。

6、哪些操作会造成内存泄漏？（必会）

- 1、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），或对该对象的唯一引用是循环的，那么该对象的内存即可回收
- 2、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
- 3、闭包、控制台日志、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）

7、JS 内存泄漏的解决方式（必会）

1、global variables: 对未声明的变量的引用在全局对象内创建一个新变量。在浏览器中，全局对象就是 window。

```

function foo(arg) {
    bar = 'some text'; // 等同于 window.bar = 'some text';
}

```

1.1) 解决:

1.1.1) 创建意外的全局变量

```
function foo() {  
    this.var1 = 'potential accident'  
}
```

1.1.2) 可以在 JavaScript 文件开头添加 “use strict”，使用严格模式。这样在严格模式下解析 JavaScript 可以防止意外的全局变量

1.1.3) 在使用完之后，对其赋值为 null 或者重新分配

1.2) 被忘记的 Timers 或者 callbacks

在 JavaScript 中使用 setInterval 非常常见

大多数库都会提供观察者或者其它工具来处理回调函数，在他们自己的实例变为不可达时，会让回调函数也变为不可达的。对于 setInterval，下面这样的代码是非常常见的：

```
var serverData = loadData();  
setInterval(function() {  
    var renderer = document.getElementById('renderer');  
    if(renderer) {  
        renderer.innerHTML = JSON.stringify(serverData);  
    }  
}, 5000); //This will be executed every ~5 seconds.
```

这个例子阐述着 timers 可能发生的情况：计时器会引用不再需要的节点或数据

1.3) 闭包：一个可以访问外部（封闭）函数变量的内部函数

JavaScript 开发的一个关键方面就是闭包：一个可以访问外部（封闭）函数变量的内部函数。由于 JavaScript 运行时的实现细节，可以通过以下方式泄漏内存：

```
var theThing = null;  
var replaceThing = function () {  
    var originalThing = theThing;  
    var unused = function () {  
        if (originalThing) // a reference to 'originalThing'  
            console.log("hi");  
    };  
    theThing = {  
        longStr: new Array(1000000).join('*'),  
        someMethod: function () {  
            console.log("message");  
        }  
    };  
};  
setInterval(replaceThing, 1000);
```

1.4) DOM 引用

有时候，在数据结构中存储 DOM 结构是有用的。假设要快速更新表中的几行内容。将每行 DOM 的引用存储在字典或数组中可能是有意义的。当这种情况发生时，就会保留同一 DOM 元素的两份引用：一个在 DOM 树种，另一个在字典中。如果将来某个时候你决定要删除这些行，则需要让两个引用都不可达。

```
var elements = {  
    button: document.getElementById('button'),  
    image: document.getElementById('image')  
};  
function doStuff() {  
    elements.image.src = 'http://example.com/image_name.png';  
}  
function removeImage() {  
    // The image is a direct child of the body element.  
    document.body.removeChild(document.getElementById('image'));  
    // At this point, we still have a reference to #button in the  
    // global elements object. In other words, the button element is  
    // still in memory and cannot be collected by the GC.  
}
```

8、说说你对原型（prototype）理解（必会）

JavaScript 中所有都是对象，在 JavaScript 中，原型也是一个对象，通过原型可以实现对象的属性继承，JavaScript 的函数对象中都包含了一个” prototype” 内部属性，这个属性所对应的就是该函数对象的原型

“prototype” 作为函数对象的内部属性，是不能被直接访问的。所以为了方便查看一个对象的原型，Firefox 和 Chrome 内核的 JavaScript 引擎中提供了” proto “这个非标准的访问器

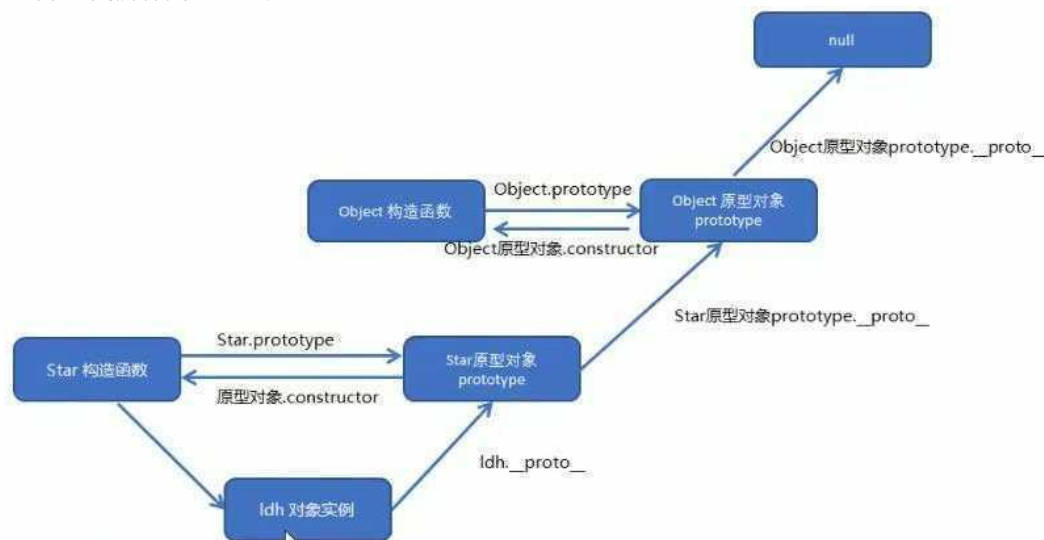
原型的主要作用就是为了实现继承与扩展对象

9、介绍下原型链（解决的是继承问题吗）（必会）

JavaScript 原型： 每个对象都会在其内部初始化一个属性，就是 prototype(原型)

原型链：

当访问一个对象的某个属性时，会先在这个对象本身属性上查找，如果没有找到，则会去它的__proto__隐式原型上查找，即它的构造函数的 prototype，如果还没有找到就会再在构造函数的 prototype 的__proto__中查找，这样一层一层向上查找就会形成一个链式结构，我们称为原型链。



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

10、常见的 js 中的继承方法有哪些（必会）

ES5 继承有以下六种方法：

- 1、原型链继承 JavaScript 实现继承的基本思想：通过原型将一个引用类型继承另一个引用类型的属性和方法
- 2、借用构造函数继承(伪造对象或经典继承) JavaScript 实现继承的基本思想：在子类构造函数内部调用超类型构造函数。 通过使用 apply() 和 call() 方法可以在新创建的子类对象上执行构造函数
- 3、组合继承(原型+借用构造)(伪经典继承) JavaScript 实现继承的基本思想：将原型链和借用构造函数的技术组合在一块，从而发挥两者之长的一种继承模式，将原型链和借用构造函数的技术组合到一起，从而取长补短发挥两者长处的一种继承模式

黑马程序员-武汉前端学科出品 www.itheima.com

4、型式继承 JavaScript 实现继承的基本思想：借助原型可以基于已有的对象创建新对象，同时还必须因此创建自定义的类型

5、寄生式继承 JavaScript 实现继承的基本思想：创建一个仅用于封装继承过程的函数，该函数在内部以某种方式来增强对象，最后再像真正是它做了所有工作一样返回对象。

寄生式继承是原型式继承的加强版

6、寄生组合式继承 JavaScript 实现继承的基本思想：通过借用函数来继承属性，通过原型链的混成形式来继承方法

ES6 的继承：

1、使用 class 构造一个父类

```
class Parent {
  constructor(name, age) {
    this.name = name
    this.age = age
  }
  sayName() {
    console.log(this.name);
  }
}
```

2、使用 class 构造一个子类，并使用 extends 实现继承，super 指向父类的原型对象

```
class Child extends Parent {
  constructor(name, age, gender) {
    super(name, age)
    this.gender = gender
  }
  sayGender() {
    console.log(this.gender);
  }
}
```

3、实例化对象

```
const ming = new Child('ming', 18, '男')
ming.sayGender()
ming.sayName()
console.log(ming.name);
console.log(ming.age);
```

11、介绍 this 各种情况（必会）

this 的情况：

1、以函数形式调用时，this 永远都是 window

2、以方法的形式调用时，this 是调用方法的对象

3、以构造函数的形式调用时，this 是新创建的那个对象

4、使用 call 和 apply 调用时，this 是指定的那个对象

5、箭头函数：箭头函数的 this 看外层是否有函数

如果有，外层函数的 this 就是内部箭头函数的 this

如果没有，就是 window

6、特殊情况：通常意义上 this 指针指向为最后调用它的对象。这里需要注意的一点就是如果返回值是一个对象，那么 this 指向的就是那个返回的对象，如果返回值不是一个对象那么 this 还是指向函数的实例

12、数组中的 forEach 和 map 的区别？（必会）

forEach 和 map 的相同点

相同点 都是循环遍历数组中的每一项

forEach 和 map 方法里每次执行匿名函数都支持 3 个参数，参数分别是 item（当前每一项），index（索引值），arr（原数组）

匿名函数中的 this 都是指向 window 只能遍历数组 都不会改变原数组 区别 map 方法

- 1.map 方法返回一个新的数组，数组中的元素为原始数组调用函数处理后的值
- 2.map 方法不会对空数组进行检测，map 方法不会改变原始数组。
- 3.浏览器支持：chrome、Safari1.5+、opera 都支持，IE9+，若 arr 为空数组，则 map 方法返回的也是一个空数组。

forEach 方法

- 1.forEach 方法用来调用数组的每个元素，将元素传给回调函数
- 2.forEach 对于空数组是不会调用回调函数的。无论 arr 是不是空数组，forEach 返回的都是 undefined。这个方法只是将数组中的每一项作为 callback 的参数执行一次

13、for in 和 for of 的区别（必会）

- 1、推荐在循环对象属性时使用 for...in，在遍历数组的时候使用 for...of
- 2、for...in 循环出的是 key，for...of 循环出的是 value
- 3、注意，for...of 是 ES6 新引入的特性。修复了 ES5 引入的 for...in 的不足
- 4、for...of 不能循环普通的对象，需要通过和 Object.keys() 搭配使用

14、call 和 apply，bind 的区别（必会）

共同点：

- 1、都是用来改变函数的 this 对象的指向的。
- 2、第一个参数都是 this 要指向的对象。
- 3、都可以利用后续参数传参。

call 方法调用一个函数，其具有一个指定的 this 值和分别地提供的参数(参数的列表)。

注意：

该方法的作用和 apply() 方法类似，只有一个区别，就是 call() 方法接受的是若干个参数的列表，而 apply() 方法接受的是一个包含多个参数的数组
方法调用一个具有给定 this 值的函数，以及作为一个数组（或类似数组对象）提供的参数。

注意：

call() 方法的作用和 apply() 方法类似，区别就是 call() 方法接受的是参数列表，而 apply() 方法接受的是一个参数数组

bind() 方法创建一个新的函数，当这个新的函数被调用时，其 this 值为提供的值，其参数列表前几项，置为创建时指定的参数序列

15、EventLoop 事件循环机制（必会）

什么是 Event Loop

JavaScript 的事件分两种，宏任务(macro-task)和微任务(micro-task)

宏任务：包括整体代码 script, setTimeout, setInterval

微任务：Promise.then(非 new Promise), process.nextTick(node 中)

事件的执行顺序——先执行宏任务，然后执行微任务，任务有同步的任务和异步的任务，同步的进入主线程，异步的进入 Event Table 并注册函数，异步事件完成后，会将回调函数放在队列中，如果还有异步的宏任务，那么就会进行循环执行上述的操作。

```
setTimeout(() => {  
    console.log('延时 1 秒');  
}, 1000)  
console.log("开始")
```

```
//开始  
//延时 1 秒
```

上述代码，setTimeout 函数是宏任务，且是异步任务，因此会将函数放入 Event Table 并注册函数，经过指定时间后，把要执行的任务加入到 Event Queue 中，等待同步任务 console.log(“开始”)执行结束后，读取 Event Queue 中 setTimeout 的回调函数执行。

上述代码不包含微任务，接下来看包含微任务的代码：

```
setTimeout(function() {  
    console.log('setTimeout');  
}, 1000)  
  
new Promise(function(resolve) {  
    console.log('promise');  
}).then(function() {  
    console.log('then');  
})
```

```
console.log('console');
```

- 1、首先 setTimeout，放入 Event Table 中，1 秒后将回调函数放入宏任务的 Event Queue 中
- 2、new Promise 同步代码，立即执行 console.log(‘promise’), 然后看到微任务 then，因此将其放入微任务的 Event Queue 中
- 3、接下来执行同步代码 console.log(‘console’)
- 4、主线程的宏任务，已经执行完毕，接下来要执行微任务，因此会执行 Promise.then，到此，第一轮事件循环执行完毕
- 5、第二轮事件循环开始，先执行宏任务，即 setTimeout 的回调函数，然后查找是否有微任务，没有，事件循环结束

总结：

事件循环先执行宏任务，其中同步任务立即执行，异步任务加载到对应的 Event Queue 中，微任务也加载到对应的微任务的 Event Queue 中，所有的同步微任务执行完之后，如果发现微任务的 Event Queue 中有未执行完的任务，先执行他们这样算是完成了一轮事件循环。接下来查看宏任务的队列中是否有异步代码，有的话执行第二轮的事件循环，以此类推。

再来看一个复杂点的例子：

```
console.log('1');  
setTimeout(function() {  
    console.log('2');  
    process.nextTick(function() {  
        console.log('3');  
    })  
})  
new Promise(function(resolve) {  
    console.log('4');  
    resolve();  
})
```

```

    }).then(function() {
        console.log('5')
    })
})

```

//1、2、4、3、5

- 1、宏任务同步代码 console.log('1')
- 2、setTimeout，加入宏任务 Event Queue，没有发现微任务，第一轮事件循环走完
- 3、第二轮事件循环开始，先执行宏任务，从宏任务 Event Queue 中取出 setTimeout 的回调函数
- 4、同步代码 console.log('2')，发现 process.nextTick，加入微任务 Event Queue
- 5、new Promise，同步执行 console.log('4')，发现 then，加入微任务 Event Queue
- 6、宏任务执行完毕，接下来执行微任务，先执行 process.nextTick，然后执行 Promise.then
- 7、微任务执行完毕，第二轮事件循环走完，没有发现宏任务，事件循环结束

16、js 防抖和节流（必会）

在进行窗口的 resize、scroll，输入框内容校验等操作时，如果事件处理函数调用的频率无限制，会加重浏览器的负担，导致用户体验非常糟糕
此时我们可以采用 debounce（防抖）和 throttle（节流）的方式来减少调用频率，同时又不影响实际效果

函数防抖：

函数防抖（debounce）：当持续触发事件时，一定时间段内没有再触发事件，事件处理函数才会执行一次，如果设定的时间到来之前，又一次触发了事件，就重新开始延时

如下，持续触发 scroll 事件时，并不执行 handle 函数，当 1000 毫秒内没有触发 scroll 事件时，才会延时触发 scroll 事件

```

function debounce(fn, wait) {
    var timeout = null;
    return function() {
        if(timeout !== null) clearTimeout(timeout);
        timeout = setTimeout(fn, wait);
    }
} // 处理函数
function handle() {
    console.log(Math.random());
}
// 滚动事件 window.addEventListener('scroll', debounce(handle, 1000));

```

函数节流（throttle）：

当持续触发事件时，保证一定时间段内只调用一次事件处理函数

节流通俗解释就比如我们水龙头放水，阀门一打开，水哗哗的往下流，秉着勤俭节约的优良传统美德，我们要把水龙头关小点，最好是如我们心意按照一定规律在某个时间间隔内一滴一滴的往下滴

如下，持续触发 scroll 事件时，并不立即执行 handle 函数，每隔 1000 毫秒才会执行一次 handle 函数

```

var throttle =function(func, delay) {
    var prev = Date.now();
    return function() {
        var context = this;
        var args = arguments;
        var now = Date.now();
        if (now - prev >= delay) {
            func.apply(context, args);
        }
    }
}

```

```

        prev = Date.now();
    }
}
}
function handle() {console.log(Math.random());}
window.addEventListener('scroll', throttle(handle, 1000));

```

总结:

函数防抖:

将几次操作合并为一次操作进行。原理是维护一个计时器，规定在延迟时间后触发函数，但是在延迟时间内再次触发的话，就会取消之前的计时器而重新设置。只有最后一次操作能被触发

函数节流:

使得一定时间内只触发一次函数。原理是通过判断是否到达一定时间来触发函数

区别:

函数节流不管事件触发有多频繁，都会保证在规定时间内一定会执行一次真正的事件处理函数，而函数防抖只是在最后一次事件后才触发一次函数。

结合应用场景

防抖(debounce)

search 搜索联想，用户在不断输入值时，用防抖来节约请求资源。

window 触发 resize 的时候，不断的调整浏览器窗口大小会不断的触发这个事件，用防抖来让其只触发一次

节流(throttle)

鼠标不断点击触发，mousedown(单位时间内只触发一次)

监听滚动事件，比如是否滑到底部自动加载更多，用 throttle 来判断

17、new 操作符具体干了什么呢？（必会）

- 1、创建一个空对象：并且 this 变量引入该对象,同时还继承了函数的原型
- 2、设置原型链 空对象指向构造函数的原型对象
- 3、执行函数体 修改构造函数 this 指针指向空对象,并执行函数体
- 4、判断返回值 返回对象就用该对象,没有的话就创建一个对象

18、用 JavaScript 实现冒泡排序。数据为 23、45、18、37、92、

13、24 （必会）

```

//升序算法
function sort(arr){
    for (var i = 0; i < arr.length; i++) {
        for (var j = 0; j < arr.length-i; j++) {
            if(arr[j]>arr[j+1]) {
                var c=arr[j];//交换两个变量的位置
                arr[j]=arr[j+1];
                arr[j+1]=c;
            }
        }
    }
};

```

```

    };
    return arr.toString();
}
console.log(sort([23, 45, 18, 37, 92, 13, 24]));

```

19、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（必会）

```

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);

```

20、已知数组 var stringArray = [“This”, “is”, “Baidu”, “Campus”], Alert 出” This is Baidu Campus”（必会）

```

var stringArray = ["This", "is", "Baidu", "Campus"]
alert(stringArray.join(""))

```

21、已知有字符串 foo=” get-element-by-id”, 写一个 function 将其转化成驼峰表示法” getElementById”（必会）

```

function combo(msg){
    var arr=msg.split("-");
    for(var i=1;i<arr.length;i++){
        arr[i]=arr[i].charAt(0).toUpperCase()+arr[i].substr(1,arr[i].length-1);
    }
    return arr.join("");
}

```

黑马程序员-武汉前端学科出品 www.itheima.com

```

}
msg=arr.join("");
return msg;
}

```

22 、 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} (必会)

```

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;
}

```

23、输出今天的日期, 以 YYYY-MM-DD 的方式, 比如今天是 2014 年 9 月 26 日, 则输出 2014-09-26 (必会)

```

var d = new Date();
// 获取年, getFullYear() 返回 4 位的数字
var year = d.getFullYear();
// 获取月, 月份比较特殊, 0 是 1 月, 11 是 12 月
var month = d.getMonth() + 1;
// 变成两位
month = month < 10 ? '0' + month : month;
// 获取日
var day = d.getDate();
day = day < 10 ? '0' + day : day;
alert(year + '-' + month + '-' + day);
}

```

24、把两个数组合并, 并删除第二个元素。(必会)

```

var array1 = ['a', 'b', 'c'];

```

```
var bArray = ['d','e','f'];
var cArray = array1.concat(bArray);
cArray.splice(1,1);
```

25、写一个 function，清除字符串前后的空格。（兼容所有浏览器）

（必会）

```
//使用自带接口 trim(), 考虑兼容性:
if (!String.prototype.trim) {
String.prototype.trim = function() {
    return this.replace(/^\s+/, "").replace(/\s+$/, "");
}
}
// test the function
var str = " \t\n test string ".trim();
alert(str == "test string"); // alerts "true"
```

26、截取字符串 abcdefg 的 efg （必会）

```
alert('abcdefg'.substring(4));
```

27、判断一个字符串中出现次数最多的字符，统计这个次数（必会）

```
var str = 'asdfsaaasasasasaa';
var json = {};
for (var i = 0; i < str.length; i++) {
    if(!json[str.charAt(i)]){
        json[str.charAt(i)] = 1;
    }else{
        json[str.charAt(i)]++;
    }
};
var iMax = 0;
var iIndex = '';
for(var i in json){
    if(json[i]>iMax){
        iMax = json[i];
        iIndex = i;
    }
}
alert('出现次数最多的是:' + iIndex + '出现' + iMax + '次');
```

28、将数字 12345678 转化成 RMB 形式 如： 12,345,678 （必会）

//思路：先将数字转为字符， str= str + '' ;
//利用反转函数，每三位字符加一个 ',' 最后一位不加； re() 是自定义的反转函数，最后再反转回去！

```
for(var i = 1; i <= re(str).length; i++){
    tmp += re(str)[i - 1];
```

```

        if(i % 3 == 0 && i != re(str).length){
            tmp += ',';
        }
    }
}

```

29、split () 和 join () 的区别？（必会）

split()是把一串字符（根据某个分隔符）分成若干个元素存放在一个数组里
即切割成数组的形式；

join() 是把数组中的字符串连成一个长串，可以大体上认为是 split()的逆操作

30、JavaScript 中如何对一个对象进行深度 clone？（必会）

```

<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>深克隆</title>
<script>
function clone(obj)
{
    if(typeof obj== 'object '){
        if(obj instanceof Array)
        {
            var result=[];
            for(var i=0;i<obj.length;i++)
            {
                result[i]=clone(obj[i]);
            }
            return result;
        }
        else
        {
            var result={};
            for(var i in obj)
            {
                result[i]=clone(obj[i]);
            }
            return result;
        }
    }
    else
    {
        return obj;
    }
}

var obj1=[12, {a: 11, b: 22}, 5];
var obj2=clone(obj1);
obj2[1].a+=5;
console.log(obj1, obj2);
</script>
</head>
<body>

```



```
</body>
</html>
```

31、js 数组去重，能用几种方法实现（必会）

1、使用 es6 set 方法 [...new Set(arr)]

```
let arr = [1, 2, 3, 4, 3, 2, 3, 4, 6, 7, 6];
let unique = (arr) => [...new Set(arr)];
unique(arr); // [1, 2, 3, 4, 6, 7]
```

2、利用新数组 indexOf 查找 indexOf() 方法可返回某个指定的元素在数组中首次出现的位置。如果没有就返回-1。

3、for 双重循环 通过判断第二层循环，去重的数组中是否含有该元素，如果有就退出第二层循环，如果没有 j==result.length 就相等，然后把对应的元素添加到最后的数组里面。

```
let arr = [1, 2, 3, 4, 3, 2, 3, 4, 6, 7, 6];
let result = [];
for (var i = 0 ; i < arr.length; i++) {
  for (var j = 0 ; j < result.length ; j++) {
    if (arr[i] === result[j]) {
      break;
    }
  };
  if (j == result.length) {
    result.push(arr[i]);
  };
};
console.log(result);
```

4、利用 for 嵌套 for，然后 splice 去重

```
function unique(arr) {
  for (var i = 0; i < arr.length; i++) {
    for (var j = i + 1; j < arr.length; j++) {
      if (arr[i] === arr[j]) {
        // 第一个等同于第二个，splice 方法删除第二个
        arr.splice(j, 1); j--;
      }
    }
  }
  return arr;
}
```

5、利用 filter

```
let arr = [1, 2, 3, 4, 3, 2, 3, 4, 6, 7, 6];
let unique = (arr) => {
  return arr.filter((item, index) => {
    return arr.indexOf(item) === index;
  });
};
unique(arr);
```

6、利用 Map 数据结构去重

```
let arr = [1, 2, 3, 4, 3, 2, 3, 4, 6, 7, 6];
```

```

let unique = (arr)=> {
let seen = new Map();
  return arr.filter((item) => {
    return !seen.has(item) && seen.set(item,1);
  });
};
unique(arr);

```

32、谈谈你对 Javascript 垃圾回收机制的理解？（高薪常问）

1、标记清除（mark and sweep）

这是 JavaScript 最常见的垃圾回收方式，当变量进入执行环境的时候，比如函数中声明一个变量，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”

垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

2、引用计数(reference counting)

在低版本 IE 中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加 1，如果该变量的值变成了另外一个，则这个值得引用次数减 1，当这个值的引用次数变为 0 的时候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为 0 的值占用的空间

在 IE 中虽然 JavaScript 对象通过标记清除的方式进行垃圾回收，但 BOM 与 DOM 对象却是通过引用计数回收垃圾的，也就是说只要涉及 BOM 及 DOM 就会出现循环引用问题

33、class 和普通构造函数有何区别？（高薪常问）

Js 构造函数：

```

function MathHandle(x,y){
  this.x=x
  this.y=y
}
MathHandle.prototype.add=function(){
  return this.x+this.y
}
var m =new MathHandle(1,2)
console.log(m.add())

```

class 基本语法：

```

class MathHandle{
  constructor(x,y){
    this.x = x
    this.y = y
  }
  add(){
    return this.x + this.y
  }
}

```

```
const m = new MathHandle(1,2)
console.log(m.add())
```

语法糖：

在上述两段代码中分别加入如下代码，运行

```
console.log(typeof MathHandle) // 'function'
console.log(MathHandle.prototype.constructor === MathHandle) //true
console.log(m.__proto__ === MathHandle.prototype) //true
```

运行结果一致。我认为，class 是构造函数的语法糖

综上所述：

class 在语法上更加贴合面向对象的写法

class 实现继承更加易读、易理解

更易于写 java 等后端语言的使用

本质还是语法糖，使用 prototype

34、JS 里垃圾回收机制是什么，常用的是哪种，怎么处理的？（高薪常问）

JS 的垃圾回收机制是为了以防内存泄漏，内存泄漏的含义就是当已经不需要某块内存时这块内存还存在着，垃圾回收机制就是间歇的不定期的寻找到不再使用的变量，并释放掉它们所指向的内存

JS 中最常见的垃圾回收方式是标记清除

工作原理：是当变量进入环境时，将这个变量标记为“进入环境”。当变量离开环境时，则将其标记为“离开环境”。标记“离开环境”的就回收内存

工作流程：

垃圾回收器，在运行的时候会给存储在内存中的所有变量都加上标记

去掉环境中的变量以及被环境中的变量引用的变量的标记

再被加上标记的会被视为准备删除的变量

垃圾回收器完成内存清除工作，销毁那些带标记的值并回收他们所占用的内存空间

35、什么是进程、什么是线程、它们之间是什么关系（了解）

1、进程：

1.1) 程序执行时的一个实例

1.2) 每个进程都有独立的内存地址空间

1.3) 系统进行资源分配和调度的基本单位

1.4) 进程里的堆，是一个进程中最大的一块内存，被进程中的所有线程共享的，进程创建时分配，主要存放 new 创建的对象实例

1.5) 进程里的方法区，是用来存放进程中的代码片段的，是线程共享的

1.6) 在多线程 OS 中，进程不是一个可执行的实体，即一个进程至少创建一个线程去执行代码

2、线程：

2.1) 进程中的一个实体

- 2.2) 进程的一个执行路径
- 2.3) CPU 调度和分派的基本单位
- 2.4) 线程本身是不会独立存在
- 2.5) 当前线程 CPU 时间片用完后, 会让出 CPU 等下次轮到自己时候在执行
- 2.6) 系统不会为线程分配内存, 线程组之间只能共享所属进程的资源
- 2.7) 线程只拥有在运行中必不可少的资源(如程序计数器、栈)
- 2.8) 线程里的程序计数器就是为了记录该线程让出 CPU 时候的执行地址, 待再次分配到时间片时候就可以从自己私有的计数器指定地址继续执行
- 2.9) 每个线程有自己的栈资源, 用于存储该线程的局部变量和调用栈帧, 其它线程 无权访问

3、关系:

- 3.1) 一个程序至少一个进程, 一个进程至少一个线程, 进程中的多个线程是共享进程的资源
- 3.2) Java 中当我们启动 main 函数时候就启动了一个 JVM 的进程, 而 main 函数所在线程就是这个进程中的一个线程, 也叫做主线程
- 3.3) 一个进程中有多个线程, 多个线程共享进程的堆和方法区资源, 但是每个线程 有自己的程序计数器, 栈区域

36、什么是任务队列? (了解)

任务队列 (task queue) 主要分两种:

- 1、宏任务 (macrotask): 在新标准中叫 task
 - 1.1) 主要包括: script(整体代码), setTimeout, setInterval, setImmediate, I/O, ui rendering
- 2、微任务 (microtask): 在新标准中叫 jobs
 - 2.1) 主要包括: process.nextTick, Promise, MutationObserver (html5 新特性)
- 3、扩展:
 - 3.1) 同步任务: 在主线程上, 排队执行的任务, 只有前一个任务执行完毕, 才能执行后一个任务
 - 3.2) 异步任务: 不进入主线程, 而进入“任务队列”的任务, 只有“任务队列”通知主线程, 某个异步任务可以执行了, 该任务才会进入主线程执行

37、栈和队列的区别? (了解)

- 1、栈的插入和删除操作都是在一端进行的, 而队列的操作却是在两端进行的
- 2、队列先进先出, 栈先进后出
- 3、栈只允许在一端进行插入和删除, 而队列允许在一端进行插入, 在另一端进行删除

38、栈和堆的区别? (了解)

- 1、栈区 (stack) — 由编译器自动分配释放, 存放函数的参数值, 局部变量的值等。堆区 (heap) — 一般由程序员分配释放, 若程序员不释放, 程序结束时可能由 OS 回收
- 2、堆 (数据结构): 堆可以被看成是一棵树, 如: 堆排序; 栈 (数据结构): 一种先进后出的数据结构

jQuery

1、jQuery 的\$(document).ready(function () {}), \$(function () {}))与原生 JS 的 window.onload 有什么不同？（必会）

1. 执行时间

window.onload 必须等到页面内包括图片、音频、视频在内的所有元素加载完毕后才能执行 \$(document).ready() 是 DOM 结构绘制完毕后就执行，而无需对图像或外部资源加载的等待，从而执行起来更快

2. 编写个数不同

window.onload 不能同时编写多个，如果有多个 window.onload 方法，只会执行一个 \$(document).ready() 可以同时编写多个，并且都可以得到执行

3. 简化写法

window.onload 没有简化写法

\$(document).ready(function() {})) 可以简写成 \$(function() {})

2、jQuery 和 Zepto 的区别？各自的使用场景？（必会）

1、同：

1) Zepto 最初是为移动端开发的库，是 jQuery 的轻量级替代品，因为它的 API 和 jQuery 相似，而文件更小

2) Zepto 最大的优势是它的文件大小，只有 8k 多，是目前功能完备的库中最小的一个，尽管不大，Zepto 所提供的工具足以满足开发程序的需要

3) 大多数在 jQuery 中常用的 API 和方法 Zepto 都有

4) 因为 Zepto 的 API 大部分都能和 jQuery 兼容，所以用起来极其容易，如果熟悉 jQuery，就能很容易掌握 Zepto。

2、异：

1) Zepto 更轻量级

2) Zepto 是 jQuery 的精简，针对移动端去除了大量 jQuery 的兼容代码

3) 针对移动端程序，Zepto 有一些基本的触摸事件可以用来做触摸屏交互（tap 事件、swipe 事件），Zepto 是不支持 IE 浏览器的。

4) DOM 操作的区别：添加 id 时 jQuery 不会生效而 Zepto 会生效

5) 事件触发的区别：使用 jQuery 时 load 事件的处理函数不会执行；使用 zepto 时 load 事件的处理函数会执行

6) 事件委托的区别：zepto 中，选择器上所有的委托事件都依次放入到一个队列中，而在 jQuery 中则委托成独立的多个事件

7) width() 与 height() 的区别：zepto 由盒模型（box-sizing）决定，用.width() 返回赋值的 width，用.css(‘width’) 返回 border 等的结果；jQuery 会忽略盒模型，始终返回内容区域的宽/高（不包含 padding、border）

8) offset() 的区别：zepto 返回 {top, left, width, height}；jQuery 返回 {width, height}。zepto 无法获取隐藏元素宽高，jQuery 可以

9) zepto 中没有为原型定义 extend 方法而 jQuery 有

10) zepto 的 each 方法只能遍历数组，不能遍历 JSON 对象

3、你是如何使用 jQuery 中的 ajax 的？（必会）

1、\$.ajax，这个是 jQuery 对 ajax 封装的最基础函数，通过使用这个函数可以完成异步通讯的所有功能。也就是说什么情况下我们都可以通过此方法进行异步刷新的操作。但是它的参数较多，有的时候可能会麻烦一些。看一下常用的参数：

```
var configObj = {  
    method    //数据的提交方式：get 和 post  
    url       //数据的提交路劲  
    async     //是否支持异步刷新，默认是 true  
    data      //需要提交的数据  
    dataType  //服务器返回数据的类型，例如 xml,String,Json 等  
    success   //请求成功后的回调函数  
    error     //请求失败后的回调函数  
}
```

\$.ajax(configObj); //通过\$.ajax 函数进行调用。

2、\$.post，这个函数其实就是对\$.ajax 进行了更进一步的封装，减少了参数，简化了操作，但是运用的范围更小了。\$.post 简化了数据提交方式，只能采用 POST 方式提交。只能是异步访问服务器，不能同步访问，不能进行错误处理。在满足这些情况下，我们可以使用这个函数来方便我们的编程，它的主要几个参数，像 method，async 等进行了默认设置，我们不可以改变的。

url: 发送请求地址。

data: 待发送 Key/value 参数。

callback: 发送成功时回调函数。

type: 返回内容格式，xml, html, script, json, text, _default。

3、\$.get，和\$.post 一样，这个函数是对 get 方法的提交数据进行封装，只能使用在 get 提交数据解决异步刷新的方式上，使用方式和\$.post 差不多。

4、\$.getJSON，这个是进一步的封装，也就是对返回数据类型为 Json 进行操作。里边就三个参数，需要我们设置，非常简单：url, [data], [callback]。

4、jQuery 的常用的方法增、删、复制、改、查（必会）

1、插入

append(content)：将 content 内容插入到匹配元素内容的最后

prepend(content)：将 content 内容插入到匹配元素内容的最前

2、删除

empty() 将内容清空标签还在

remove() 指定的标签和内容都移除

3、复制

clone([true])

参数说明：有 true：克隆元素和元素绑定的事件，没有 true：只克隆元素

4、替换

replaceWith()

5、查找

`eq(index)`: 查找指定下标的元素下标从 0 开始

`filter(expr)`: 过滤匹配的 class 选择器，其实就是缩小范围查找

`not(expr)`: 排除匹配指定选择器之外的元素

`next([expr])`: 查找指定元素下一个元素

`prev([expr])`: 查找指定元素的上一个元素

`parent([expr])`: 查找当前元素的父元素

5、jQuery 中\$.get()提交和\$.post()提交的区别？（必会）

相同点：都是异步请求的方式来获取服务端的数据；

异同点：1、请求方式不同：`$.get()` 方法使用 GET 方法来进行异步请求的。`$.post()` 方法使用 POST 方法来进行异步请求的。

2、参数传递方式不同：`get` 请求会将参数跟在 URL 后进行传递，而 POST 请求则是作为 HTTP 消息的实体内容发送给 Web 服务器的，这种传递是对用户不可见的。

3、数据传输大小不同：`get` 方式传输的数据大小不能超过 2KB 而 POST 要大的多

4、安全问题：`GET` 方式请求的数据会被浏览器缓存起来，因此有安全问题。

6、简单的讲叙一下 jQuery 是怎么处理事件的，你用过哪些事件？

（必会）

首先去加载文档，在页面加载完毕后，浏览器会通过 javascript 为 DOM 元素添加事件
jQuery 中的常用事件

`.click()` 鼠标单击触发 du 事件

`.dblclick()` 双击触发

`.mousedown()/up()` 鼠标按下/弹起触发事件

`.mousemove()`，鼠标移动事件；`.mouseover()/out()`，鼠标移入/移出触发事件

`.mouseenter()/leave()` 鼠标进入/离开触发事件

`.hover(func1, func2)`，鼠标移入调用 func1 函数，移出调用 func2 函数

`.focusin()`，鼠标聚焦到该元素时触发事件

`.focusout()`，鼠标失去焦点时触发事件

`.focus()/blur()` 鼠标聚焦/失去焦点触发事件（不支持冒泡）

`.change()`，表单元素发生改变时触发事件

`.select()`，文本元素被选中时触发事件

`.submit()`，表单提交动作触发

`.keydown()/up()`，键盘按键按下/弹起触发

`.on()`，多事件的绑定

7、你使用过 jQuery 中的动画吗，是怎样用的？（必会）

使用过。

- 1) hide() 和 show() 同时修改多个样式属性，像高度，宽度，不透明度；
- 2) fadeIn() 和 fadeOut() fadeTo() 只改变不透明度
- 3) slideUp() 和 slideDown() slideToggle() 只改变高度；
- 4) animate() 属于自定义动画的方法。

8、你在 jQuery 中使用过哪些插入节点的方法，它们的区别是什么？

（必会）

append(),
appendTo(),
prepend(),
prependTo(),
after(),
insertAfter(),
before(),
insertBefore() 大致可以分为内部追加和外部追加
append() 表示向每个元素内部追加内容
appendTo() 将所有匹配的元素追加到指定的元素中
prepend(): 向每个匹配的元素内部前置添加内容
prependTo(): 将所有匹配的元素前置到指定的元素中
after(): 在每个匹配元素之后插入内容
insertAfter(): 将所有匹配的元素插入到指定元素的后面

9、jQuery 中如何来获取或设置属性？（必会）

jQuery 中可以用 attr() 方法来获取和设置元素属性，removeAttr() 方法来删除元素属性

10、jQuery 如何设置和获取 HTML、文本和值？（必会）

- 1、html() 方法：如果想更改或者是设置 HTML 的内容，我们可以使用 html() 方法，首先我们先使用这个方法获取元素里面的内容 var html=\$("#p").html()。如果需要设置某元素的 HTML 代码，那么我们就可以使用此方法加上一个参数。此方法只能应用于 XHTML 中，不能用于 xml。
- 2、text() 方法，去设置某个元素中的文本内容，代码是 var text=\$("#p").text(); 如果想设置文本同样需要给它传一个参数。
- 3、val() 方法，可以用来设置和获取元素的值，它不仅仅可以设置元素，同时也能获取元素，另外，它能是下拉列表框，多选框，和单选框相应的选项被选中，在表单操作中会经常用到。

11、有哪些查询节点的选择器？（必会）

:first 查询第一个

:last 查询最后一个
:odd 查询奇数但是索引从 0 开始
:even 查询偶数
:eq(index) 查询相等的
:gt(index) 查询大于 index 的
:lt 查询小于 index:header 选取所有的标题等

12、jQuery 中的 hover() 和 toggle() 有什么区别？（必会）

1、hover() 和 toggle() 都是 jQuery 中两个合成事件

hover(fn1, fn2): 一个模仿悬停事件的方法。当鼠标移动到一个匹配的元素上面时，会触发指定的第一个函数。当鼠标移出这个元素时，会触发指定的第二个函数

2、toggle(evenFn, oddFn): 每次点击时切换要调用的函数。如果点击了一个匹配的元素，则触发指定的第一个函数，当再次点击同一元素时，则触发指定的第二个函数。随后的每次点击都重复对这两个函数的轮番调用

13、jQuery 中 detach() 和 remove() 方法的区别是什么？（必会）

detach() 和 remove() 作用相同，即移除被选元素，包括所有文本和子节点

不同之处在于 detach(): 移除被选元素，包括所有文本和子节点。会保留所有绑定的事件、附加的数据

remove(): 移除被选元素，包括所有文本和子节点。绑定的事件、附加的数据等都会被移除

14、\$(this) 和 this 关键字在 jQuery 中有何不同？（必会）

\$(this) 返回一个 jQuery 对象，你可以对它调用多个 jQuery 方法，比如用 text() 获取文本，用 val() 获取值等等。

而 this 代表当前元素，它是 JavaScript 关键词中的一个，表示上下文中的当前 DOM 元素。你不能对它调用 jQuery 方法，直到它被 \$() 函数包裹，例如 \$(this)。

15、jQuery 中 attr() 和 prop() 的区别（必会）

1、对于 HTML 元素本身就带有的固有属性，或者说 W3C 标准里就包含有这些属性，更直观的说法就是，编辑器里面可以智能提示出来的一些属性，如: src、href、value、class、name、id 等。在处理时，使用 prop() 方法。

2、对于 HTML 元素我们自定义的 DOM 属性，即元素本身是没有这个属性的，如: data-*。在处理时，使用 attr() 方法。

```
<a href="#" id="link1" class="btn" action="delete">删除</a>
```

这个例子里的<a>元素的 dom 属性值有" id、href、class 和 action"，很明显，前三个是固有属性，而后面一个 action 属性是我们自己定义上去的

<a>元素本身是没有属性的。这种就是自定义的 dom 属性。处理这些属性时，建议使用 attr 方法，使用 prop 方法对自定义属性取值和设置属性值时，都会返回 undefined 值。

像 checkbox, radio 和 select 这样的元素，选中属性对应 "checked" 和 "selected"，这些也属于固有属性，因此需要使用 prop 方法去操作才能获取正确答案

16、jQuery 库中的\$()是什么？（必会）

\$() 函数是 jQuery() 函数的别称，\$() 函数用于将任何对象包裹成 jQuery 对象，然后被允许调用定义在 jQuery 对象上的多个不同方法。甚至可以将一个选择器字符串传入\$() 函数，它会返回一个包含所有匹配的 DOM 元素数组的 jQuery 对象。

17、jQuery 的属性拷贝(extend)的实现原理是什么，如何实现深浅拷贝？（高薪常问）

jQuery.extend() 函数用于将一个或多个对象的内容合并到目标对象。

语法

```
$.extend( target [, object1 ] [, objectN ] )
```

指示是否深度合并

```
$.extend( [deep ], target, object1 [, objectN ] )
```

注意:不支持第一个参数传递 false 。

参数	描述
deep	可选。 Boolean 类型 指示是否深度合并对象，默认为 false。 如果该值为 true，且多个对象的某个同名属性也都是对象， 则该“属性对象”的属性也将进行合并。
target	Object 类型 目标对象，其他对象的成员属性将被附加到该对象上。
object1	可选。 Object 类型 第一个被合并的对象。
objectN	可选。 Object 类型 第 N 个被合并的对象。

深拷贝，深拷贝代码把 extend 函数的第一个参数设置为 true：（对原始对象属性所引用的对象进行递归拷贝）

```
var newObject = $.extend(true, {}, oldObject);
```

浅拷贝，浅拷贝代码 extend 函数里不传入第一个参数，默认为 false（只复制一份原始对象的引用）

```
var newObject = $.extend({}, oldObject);
```

18、jQuery 的实现原理？（高薪常问）

- 1、为了防止全局变量污染，把 jQuery 的代码写在一个自调用函数中
- 2、咱们平常使用的\$实际上 jQuery 对外暴露的一个工厂函数
- 3、而构造函数在 jQuery 的内部叫 init，并且这个构造函数还被添加到了 jQuery 的原型中。当我们调用工厂函数的时候返回的其实是一个构造函数的实例
- 4、jQuery 为了让第三方能够对其功能进行扩展，所以把工厂函数的原型与构造函数的原型保持

了一致。这样子对外暴露工厂函数，即可对原型进行扩展

数据可视化

1、echarts 的基本用法（必会）

1、初始化类

Html 里面创建一个 id 为 box1 的 div，并初始化 echarts 绘图实例 `var myChart = echarts.init(document.getElementById('box1'))`

2、样式配置

- title：标题
- tooltip：鼠标悬停气泡
- xAxis：配置横轴类别，type 类型为 category 类别
- series：销量数据，data 参数与横轴一一对应，如果想调样式，也可以简单调整，比如每个条形图的颜色可以通过函数进行数组返回渲染

3、渲染图展示表

`myChart.setOption(option);`

2、如何使用 echarts（必会）

- ①获取 echarts：在官网下载 echarts 版本 或 npm 下载
- ②引入 echarts：script 引入 或者 vue 在入口文件里引用
- ③创建一个 dom 元素 用来放置图表
- ④配置 echarts 属性

3、echarts 如何画图？（必会）

1、echarts 是通过 canvas 来实现的，由于 canvas 的限制，所以 echarts 在实现的时候多是绘制一些规则的，可预期的，易于实现的东西

2、echarts 的核心就是 options 配置的对象。一般使用最多的是直角坐标图，极点图，饼状图，地图。

3、对于直角坐标，必须配置 xAsix 和 yAxis，对于直角坐标必须配置 radiusAxis 和 angleAxis。

4、就是 series 系列的认识，它是一个数组，数组的每一项都代表着一个单独的系列，可以配置各种图形等等功能。然后 data 设置数据源

一般是一个每一项都是数组的数组，也就是嵌套数组。里层数组一般代表坐标位置

4、echarts 绘制条形图（必会）

1、初始化类

Html 里面创建一个 id 为 box1 的 div，并初始化 echarts 绘图实例

黑马程序员-武汉前端学科出品 www.itheima.com

```
var myChart = echarts.init(document.getElementById('box1'))
```

2、样式配置

title : 标题

tooltip : 鼠标悬停气泡

xAxis : 配置横轴类别, type 类型为 category 类别

series: 销量数据, data 参数与横轴一一对应, 如果想调样式, 也可以简单调整, 比如每个条形图的颜色可以通过函数进行数组返回渲染

3、渲染图展示表

```
myChart.setOption(option);
```

5、切换其他组件统计图时, 出现卡顿问题如何解决 (必会)

1、原因: 每一个图例在没有数据的时候它会创建一个定时器去渲染气泡, 页面切换后, echarts 图例是销毁了, 但是这个 echarts 的实例还在内存当中, 同时它的气泡渲染定时器还在运行。这就导致 echarts 占用 CPU 高, 导致浏览器卡顿, 当数据量比较大时甚至浏览器崩溃

2、解决方法: 在 mounted() 方法和 destroy() 方法之间加一个 beforeDestroy() 方法释放该页面的 chart 资源, clear() 方法则是清空图例数据, 不影响图例的 resize, 而且能够释放内存, 切换的时候就很顺畅了

```
beforeDestroy () {  
  this.chart.clear()  
}
```

6、echarts 图表自适应 div resize 问题 (必会)

echarts 官网的实例都具有响应式功能

echarts 图表本身是提供了一个 resize 的函数的。

用于当 div 发生 resize 事件的时候, 让其触发 echarts 的 resize 事件, 重绘 canvas。

```
<div class="chart">  
  <div class="col-md-3" style="width:73%;height:270px" id="chartx"></div>  
</div>  
<script src="/static/assets/scripts/jquery.ba-resize.js"></script>  
js 代码:  
var myChartx = echarts.init(document.getElementById('chartx'));  
$('.chart').resize(function() {  
  myChartx.resize();  
})
```

7、echarts 在 vue 中怎么引用? (必会)

首先我们初始化一个 vue 项目, 执行 vue create echart

接着我们进入初始化的项目下。安装 echarts

```
npm install echarts -S //或
cnpm install echarts -S
```

安装完成之后，我们就可以开始引入我们需要的 echarts 了，接下来介绍几种使用 echarts 的方式。

全局引用：

首先在 main.js 中引入 echarts，将其绑定到 vue 原型上：

```
import echarts from 'echarts'
Vue.prototype.$echarts = echarts;
```

接着，我们就可以在任何一个组件中使用 echarts 了。

局部使用：

当然，很多时候没必要在全局引入 echarts，那么我们只在单个组件内使用即可，代码更加简单：

```
import echarts from 'echarts'
```

可以看到，我们直接在组件内引入 echarts，接下来跟全局引入的使用一样。区别在于，这种方式如果你在其他组件内用 echarts，则必须重新引入了。

8、echarts 支持哪些图标？（了解）

折线图（区域图）、柱状图（条状图）、散点图（气泡图）、K 线图、饼图（环形图）

雷达图（填充雷达图）、和弦图、力导向布局图、地图、仪表盘、漏斗图、事件流程图等 12 类图表

Ajax/计算机网络相关

1、什么是 Ajax，Ajax 的原理，Ajax 都有哪些优点和缺点？（必会）

什么是 Ajax

Ajax 是 “Asynchronous JavaScript and XML” 的缩写。他是指一种创建交互式网页应用的网页开发技术。沟通客户端与服务器，可以在不必刷新整个浏览器的情况下，与服务器进行异步通讯的技术

Ajax 的原理

通过 XMLHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后用 javascript 来操作 DOM 而更新页面。这其中最关键的一步就是从服务器获得请求数据。

XMLHttpRequest 是 Ajax 的核心机制，它是在 IE5 中首先引入的，是一种支持异步请求的技术。简单的说，也就是 javascript 可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

Ajax 的优点

- 1、最大的一点是页面无刷新，用户的体验非常好。
- 2、使用异步方式与服务器通信，具有更加迅速的响应能力。
- 3、可以把以前一些服务器负担的工作转嫁到客户端，利用客户端闲置的能力来处理，减轻服务器和带宽的负担，节约空间和宽带租用成本。并且减轻服务器的负担，Ajax 的原则是“按需取数据”，可以最大程度的减少冗余请求，和响应对服务器造成的负担。
- 4、基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。

黑马程序员-武汉前端学科出品 www.itheima.com

Ajax 的缺点

- 1、Ajax 不支持浏览器 back 按钮。
- 2、安全问题 Ajax 暴露了与服务器交互的细节。
- 3、对搜索引擎的支持比较弱。
- 4、破坏了程序的异常机制。

2、常见的 HTTP 状态码以及代表的意义（必会）

5 种常见的 HTTP 状态码以及代表的意义

200 (OK): 请求已成功, 请求所希望的响应头或数据体将随此响应返回。

400 (Bad Request): 请求格式错误。

- 1) 语义有误, 当前请求无法被服务器理解。除非进行 修改, 否则客户端不应该重复提交这个请求;
- 2) 请求参数有误。

404 (Not Found): 请求失败, 请求所希望得到的资源未被在服务器上发现。

500 (Internal Server Error): 服务器遇到了一个未曾预料的状态, 导致了它无法完成对请求的处理。

更多状态码

100 => 正在初始化 (一般是看不到的)

101 => 正在切换协议 (websocket 浏览器提供的)

202 => 表示接受

301 => 永久重定向/永久转移

302 => 临时重定向/临时转移 (一般用来做服务器负载均衡)

304 => 本次获取的内容是读取缓存中的数据, 会每次去服务器校验

401 => 未认证, 没有登录网站

403 => 禁止访问, 没有权限

502 => 充当网关或代理的服务器, 从远端服务器接收到了一个无效的请求

503 => 服务器超负荷 (假设一台服务器只能承受 10000 人, 当第 10001 人访问的时候, 如果服务器没有做负载均衡, 那么这个人的网络状态码就是 503)

505 => 服务器不支持请求的 HTTP 协议的版本, 无法完成处理。

3、请介绍一下 XMLHttpRequest 对象及常用方法和属性（必会）

XMLHttpRequest 对象

Ajax 的核心是 XMLHttpRequest。它是一种支持异步请求的技术。XMLHttpRequest 使您可以使用 JavaScript 向服务器提出请求并处理响应, 而不阻塞用户。可以在页面加载以后进行页面的局部更新

方法

open(String method,String url,boolean asynch,String username,String password)

send(content)

setRequestHeader(String header,String value)

getAllResponseHeaders()

getResponseHeader(String header)

abort()

常用详细解析

open(): 该方法创建 HTTP 请求

第一个参数是指定提交方式(post、get)

第二个参数是指定要提交的地址是哪

黑马程序员-武汉前端学科出品 www.itheima.com

第三个参数是指定是异步还是同步(true 表示异步, false 表示同步)

第四和第五参数在 HTTP 认证的时候会用到。是可选的

setRequestHeader(String header,String value): 设置消息头 (使用 post 方式才会使用到, get 方法并不需要调用该方法)

```
xmlHTTP.setRequestHeader("Content-type","application/x-www-form-urlencoded");
```

send(content): 发送请求给服务器

如果是 get 方式, 并不需要填写参数, 或填写 null

如果是 post 方式, 把要提交的参数写上去

常用属性

onreadystatechange: 请求状态改变的事件触发器 (readyState 变化时会调用此方法), 一般用于指定回调函数

readyState: 请求状态 readyState 一改变, 回调函数被调用, 它有 5 个状态

0: 未初始化

1: open 方法成功调用以后

2: 服务器已经应答客户端的请求

3: 交互中。HTTP 头信息已经接收, 响应数据尚未接收。

4: 完成。数据接收完成

responseText: 服务器返回的文本内容

responseXML: 服务器返回的兼容 DOM 的 XML 内容

status: 服务器返回的状态码

statusText: 服务器返回状态码的文本信息

回调函数是什么

回调函数就是接收服务器返回的内容!

4、Ajax 的实现流程是怎样的? (必会)

1、创建 XMLHttpRequest 对象, 也就是创建一个异步调用对象.

2、创建一个新的 HTTP 请求, 并指定该 HTTP 请求的方法、URL 及验证信息.

3、设置响应 HTTP 请求状态变化的函数.

4、发送 HTTP 请求.

5、获取异步调用返回的数据.

6、使用 JavaScript 和 DOM 实现局部刷新.

```
<script type="text/javascript">
```

```
var HTTPRequest;
```

```
function checkUsername() {
```

```
    //创建 XMLHttpRequest 对象
```

```
    if(window.XMLHttpRequest) {
```

```
        //在 IE6 以上的版本以及其他内核的浏览器(Mozilla)等
```

```
        HTTPRequest = new XMLHttpRequest();
```

```
    }else if(window.ActiveXObject) {
```

```
        //在 IE6 以下的版本
```

```
        HTTPRequest = new ActiveXObject();
```

```
    }
```

```
    //创建 HTTP 请求
```

```
    HTTPRequest.open("POST", "Servlet1", true);
```

```
    //因为我使用的是 post 方式, 所以需要设置消息头
```

```
        HTTPRequest.setRequestHeader("Content-type",
```

```
        "application/x-www-form-urlencoded");
```

```
    //指定回调函数
```

```

        XMLHttpRequest.onreadystatechange = response22;
        //得到文本框的数据
        var name = document.getElementById("username").value;
        //发送 HTTP 请求，把要检测的用户名传递进去
        XMLHttpRequest.send("username=" + name);
    }
    //接收服务器响应数据
    function response22() {
        //判断请求状态码是否是 4【数据接收完成】
        if(XMLHttpRequest.readyState==4) {
            //再判断状态码是否为 200【200 是成功的】
            if(XMLHttpRequest.status==200) {
                //得到服务端返回的文本数据
                var text = XMLHttpRequest.responseText;
                //把服务端返回的数据写在 div 上
                var div = document.getElementById("result");
                div.innerText = text;
            }
        }
    }
}
</script>

```

5、Ajax 接收到的数据类型有哪些，数据如何处理？（必会）

接收到的数据类型

String / JSON 字符串 / 二进制数据流
 JSON 字符串反序列化后，转成引用类型使用
 String 直接使用
 前端用 Blob 转换

如何处理数据

1、字符串转对象

第一种方式：eval（）；

```

var data='{ "student" : [ { "name": "张三", "age": "11" } , { "name": "李四", "age": "11" }, { "name": "王五", "age": "11" } ] }';
eval (' (" +data+ ") ');

```

第二种方式：JSON.parse（）；

```

var data='{ "student" : [ { "name": "张三", "age": "11" } , { "name": "李四", "age": "11" }, { "name": "王五", "age": "11" } ] }';
JSON.parse (data);

```

parse（）与 eval（）区别

eval（）方法不会去检查给的字符串时候符合 json 的格式~同时如果给的字符串中存在 js 代码 eval（）也会一并执行~比如：

```

var data='{ "student" : [ { "name": "张三", "age": "11" } , { "name": "李四", "age": "alert(11)" }, { "name": "王五", "age": "11" } ] }';

```

此时执行 eval 方法后会先弹出一个提示框输出 11 的字符串；

这时候使用 JSON.parse() 就会报错，显示错误信息为当前字符串不符合 json 格式；即 JSON.parse() 方法会检查需要转换的字符串是否符合 json 格式

相比而言 eval（）方法是很不安全，特别是当涉及到第三方时我们需要确保传给 eval（）

的参数是我们可以控制的，不然里面插入比如 `window.location~` 指向一个恶意的连接总的来说，还是推荐使用 `JSON.parse()` 来实现 json 格式字符串的解析

6、请解释一下 JavaScript 的同源策略（必会）

同源策略是客户端脚本的重要的安全度量标准。其目的是防止某个文档或脚本从多个不同源装载。所谓同源指的是：协议，域名，端口相同，同源策略是一种安全协议，指一段脚本只能读取来自同一来源的窗口和文档的属性。

7、为什么会有跨域的问题出现，如何解决跨域问题（必会）

什么是跨域

指的是浏览器不能执行其他网站的脚本，它是由浏览器的同源策略造成的，是浏览器对 javascript 施加的安全限制，防止他人恶意攻击网站

比如一个黑客，他利用 `iframe` 把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名和密码登录时，如果没有同源限制，他的页面就可以通过 JavaScript 读取到你的表单中输入的内容，这样用户名和密码就轻松到手了。

解决方式

1、jsonp

原理：动态创建一个 script 标签。利用 script 标签的 `src` 属性不受同源策略限制。因为所有的 `src` 属性和 `href` 属性都不受同源策略限制。可以请求第三方服务器数据内容。

步骤

- 1.1) 去创建一个 script 标签
- 1.2) script 的 `src` 属性设置接口地址
- 1.3) 接口参数, 必须要带一个自定义函数名 要不然后台无法返回数据。
- 1.4) 通过定义函数名去接收后台返回数据

//去创建一个 script 标签

```
var script = document.createElement("script");
```

//script 的 `src` 属性设置接口地址 并带一个 callback 回调函数名称

```
script.src = "HTTP://127.0.0.1:8888/index.php?callback=jsonpCallback";
```

//插入到页面

```
document.head.appendChild(script);
```

//通过定义函数名去接收后台返回数据 `function jsonpCallback(data) {`

 //注意 jsonp 返回的数据是 json 对象可以直接使用

 //Ajax 取得数据是 json 字符串需要转换成 json 对象才可以使用。

```
}
```

2、CORS：跨域资源共享

原理：服务器设置 `Access-Control-Allow-Origin` HTTP 响应头之后，浏览器将会允许跨域请求

限制：浏览器需要支持 HTML5，可以支持 POST，PUT 等方法兼容 ie9 以上

需要后台设置

`Access-Control-Allow-Origin: *` //允许所有域名访问，或者

`Access-Control-Allow-Origin: HTTP://a.com` //只允许所有域名访问

3、反向代理

4、window+iframe

8、Get 和 Post 的区别以及使用场景（必会）

区别

- 1、Get 使用 URL 或 Cookie 传参。而 Post 将数据放在 body 中
- 2、Get 的 URL 会有长度上的限制，则 Post 的数据则可以非常大
- 3、Post 比 Get 安全，因为数据在地址栏上不可见

最本质的区别

基于 http 协议进行请求，其实 GET 和 POST 无区别，只是请求时的方式不同，都可以携带请求体，也可以在 URL 带参数

区别来自于浏览器对 URL 长度的限制，请求体大小来源于服务器的限制

还有语义的区别：

GET 是获取，POST 是提交

Get 是用来从服务器上获得数据，而 post 是用来向服务器上传递数据

Get/Post 使用场景

若符合下列任一情况，则 Post 方法：

- 1、请求的结果有持续性的作用，例如：数据库内添加新的数据行
- 2、若使用 Get 方法，则表单上收集的数据可能让 URL 过长
- 3、要传送的数据不是采用 ASCII 编码

若符合下列任一情况，则用 Get 方法：

- 1、请求是为了查找资源，html 表单数据仅用来搜索
- 2、请求结果无持续性的副作用
- 3、收集的数据及 html 表单内的输入字段名称的总长不超过 1024 个字符

9、解释 jsonp 的原理（必会）

什么是 jsonp，jsonp 的作用

jsonp 并不是一种数据格式，而 json 是一种数据格式，jsonp 是用来解决跨域获取数据的一种解决方案

具体原理

是通过动态创建 script 标签，然后通过标签的 src 属性获取 js 文件中的 js 脚本，该脚本的内容是一个函数调用，参数就是服务器返回的数据，为了处理这些返回的数据，需要事先在页面定义好回调函数，本质上使用的并不是 Ajax 技术，Ajax 请求受同源策略的影响，不允许进行跨域请求，而 script 标签的 src 属性中的链接却可以访问跨域的 js 脚本，利用这个特性，服务端不在返回 json 格式的数据，而是返回调用某个函数的 js 代码，在 src 中进行了调用，这样就实现了跨域

10、封装好的 Ajax 里的常见参数及其代表的含义（必会）

url： 发送请求的地址。

type： 请求方式（post 或 get）默认为 get。

async： 同步异步请求，默认 true 所有请求均为异步请求。

timeout： 超时时间设置，单位毫秒

data： 要求为 Object 或 String 类型的参数，发送到服务器的数据

cache： 默认为 true（当 dataType 为 script 时，默认为 false），设置为 false 将不会从浏览器缓存中加载请求信息。

dataType： 预期服务器返回的数据类型。

可用的类型如下：

xml： 返回 XML 文档，可用 JQuery 处理。

html: 返回纯文本 HTML 信息; 包含的 script 标签会在插入 DOM 时执行。
script: 返回纯文本 JavaScript 代码。不会自动缓存结果。
json: 返回 JSON 数据。
jsonp: JSONP 格式。使用 JSONP 形式调用函数时, 例如 myurl?callback=?, JQuery 将自动替换后一个 “?” 为正确的函数名, 以执行回调函数。
text: 返回纯文本字符串。
success: 请求成功后调用的回调函数, 有两个参数。
 1、由服务器返回, 并根据 dataType 参数进行处理后的数据。
 2、描述状态的字符串。
error: 要求为 Function 类型的参数, 请求失败时被调用的函数。该函数有 3 个参数
 1、XMLHttpRequest 对象
 2、错误信息
 3、捕获的错误对象(可选)
complete :function(XMLHttpRequest, status){ //请求完成后最终执行参数}

11、jQuery 中 \$.ajax 与 fetch 、 axios 有什么区别? (必会)

1、jQuery \$.ajax

```
$.ajax({
  type: 'POST',
  url: url,
  data: data,
  dataType: dataType,
  success: function () {},
  error: function () {}
});
```

jQuery 本身是针对 MVC 的编程, 不符合现在前端 MVVM 的开发模式
jQuery 整个项目很大, 单纯使用 Ajax 却要引入整个 jQuery 非常的不合理

2、axios

```
axios({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'Flintstone'
  }
}).then(function (response) {
  console.log(response);
})
```

客户端支持防止 CSRF/XSRF 自动转换 JSON 数据 取消请求 转换请求和响应数据 拦截请求
和响应支持 Promise API 从 node.js 发出 HTTP 请求 从浏览器中创建 XMLHttpRequest
axios 是一个基于 Promise 用于浏览器和 nodejs 的 HTTP 客户端

2、fetch

```
let data = response.json();
let response = await fetch(url);
try {}
catch(e) {
  console.log(data);
  console.log("Oops, error", e);
}
```

为什么要用 axios

- 3.1) fetch 没有办法原生监测请求的进度，而 XHR 可以
- 3.2) fetch 不支持 abort，不支持超时控制，使用 setTimeout 及 Promise.reject 的实现
的 超时控制并不能阻止请求过程继续在后台运行，造成了量的浪费
- 3.3) fetch 默认不会带 cookie，需要添加配置项
- 3.4) fetch 只对网络请求报错，对 400，500 都当做成功的请求，需要封装去处理

脱离了 XHR，是 ES 规范里新的实现方式 更加底层，提供的 API 丰富 (request, response) 更好更方便的写法符合关注分离，没有将输入、输出和用事件来跟踪的状态混杂在一个对象里

12、Ajax 注意事项及适用和不适用场景（必会）

Ajax 开发时，网络延迟——即用户发出请求到服务器发出响应之间的间隔——需要慎重考虑。不给予用户明确的回应，没有恰当的预读数据，或者对 XMLHttpRequest 的不恰当处理，都会使用户感到延迟，这是用户不希望看到的，也是他们无法理解的。通常的解决方案是，使用一个可视化的组件来告诉用户系统正在进行后台操作并且正在读取数据和内容。

Ajax 适用场景

- 1、表单驱动的交互
- 2、深层次的树的导航
- 3、快速的用户与用户间的交流响应
- 4、类似投票、yes/no 等无关痛痒的场景
- 5、对数据进行过滤和操纵相关数据的场景
- 6、普通的文本输入提示和自动完成的场景

Ajax 不适用场景

- 1、部分简单的表单
- 2、搜索
- 3、基本的导航
- 4、替换大量的文本
- 5、对呈现的操纵

13、HTTP 与 HTTPS 的区别（必会）

- 1、HTTPS 协议需要到 CA (Certificate Authority, 证书颁发机构) 申请证书，一般免费证书较少，因而需要一定费用。(以前网易官网是 HTTP，而网易邮箱是 HTTPS。)
- 2、HTTP 是超文本传输协议，信息是明文传输，HTTPS 则是具有安全性的 SSL 加密传输协议
- 3、HTTP 和 HTTPS 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443
- 4、HTTP 的连接很简单，是无状态的。HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 HTTP 协议安全。(无状态的意思是其数据包的发送、传输和接收都是相互独立的。无连接的意思是指通信双方都不长久的维持对方的任何信息。)

14、localStorage、sessionStorage、cookie 的区别（必会）

共同点：都是保存在浏览器端、且同源的

区别：

- 1、cookie 数据始终在同源的 http 请求中携带（即使不需要），即 cookie 在浏览器和服务器间

- 来回传递，而 sessionStorage 和 localStorage 不会自动把数据发送给服务器，仅在本机保存。cookie 数据还有路径（path）的概念，可以限制 cookie 只属于某个路径下
- 2、存储大小限制也不同，cookie 数据不能超过 4K，同时因为每次 http 请求都会携带 cookie、所以 cookie 只适合保存很小的数据，如会话标识。sessionStorage 和 localStorage 虽然也有存储大小的限制，但比 cookie 大得多，可以达到 5M 或更大
- 3、数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭之前有效；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie：只在设置的 cookie 过期时间之前有效，即使窗口关闭或浏览器关闭
- 4、作用域不同，sessionStorage 不在不同的浏览器窗口中共享，即使是同一个页面；localStorage 在所有同源窗口中都是共享的；cookie 也是在所有同源窗口中都是共享的
- 5、web Storage 支持事件通知机制，可以将数据更新的通知发送给监听者
- 6、web Storage 的 api 接口使用更方便

15、简述 web 前端 Cookie 机制，并结合该机制说明会话保持原理？

（必会）

Cookie 是进行网站用户身份，实现服务端 Session 会话持久化的一种非常好方式。

1、为什么需要 Cookie

HTTP 是一种无状态的协议，客户端与服务器建立连接并传输数据，数据传输完成后，连接就会关闭。再次交互数据需要建立新的连接，因此，服务器无法从连接上跟踪会话，也无法知道用户上一次做了什么。

例如：在网络有时候需要用户登录才进一步操作，用户输入用户名密码登录后，浏览了几个页面，由于 HTTP 的无状态性，服务器并不知道用户有没有登录

Cookie 是解决 HTTP 无状态性的有效手段，服务器可以设置或读取 Cookie 中所包含的信息。当用户登录后，服务器会发送包含登录凭据的 Cookie 到用户浏览器客户端，而浏览器对该 Cookie 进行某种形式的存储（内存或硬盘）。用户再次访问该网站时，浏览器会发送该 Cookie（Cookie 未到期时）到服务器，服务器对该凭据进行验证，合法时使用户不必输入用户名和密码就可以直接登录

本质上讲，Cookie 是一段文本信息。客户端请求服务器时，如果服务器需要记录用户状态，就在响应用户请求时发送一段 Cookie 信息。客户端浏览器保存该 Cookie 信息，当用户再次访问该网站时，浏览器会把 Cookie 做为请求信息的一部分提交给服务器。服务器检查 Cookie 内容，以此来判断用户状态，服务器还会对 Cookie 信息进行维护，必要时会对 Cookie 内容进行修改

2、Cookie 的类型

Cookie 总时由用户客户端进行保存的（一般是浏览器），按其存储位置可分为：内存式 Cookie 和硬盘式 Cookie。

内存式 Cookie 存储在内存中，浏览器关闭后就会消失，由于其存储时间较短，因此也被称为非持久 Cookie 或会话 Cookie。

硬盘式 Cookie 保存在硬盘中，其不会随浏览器的关闭而消失，除非用户手工清理或到了过期时间。由于硬盘式 Cookie 存储时间是长期的，因此也被称为持久 Cookie。

3、Cookie 的实现原理

Cookie 定义了一些 HTTP 请求头和 HTTP 响应头，通过这些 HTTP 头信息使服务器可以与客户进行状态交互。

客户端请求服务器后，如果服务器需要记录用户状态，服务器会在响应信息中包含一个

Set-Cookie 的响应头，客户端会根据这个响应头存储 Cookie 信息。再次请求服务器时，客户端会在请求信息中包含一个 Cookie 请求头，而服务器会根据这个请求头进行用户身份、状态等校验。

下面是一个实现 Cookie 机制的，简单的 HTTP 请求过程：

- 3.1) 客户端请求服务器
客户端请求 IT 笔录网站首页，请求头如下：
GET / HTTP/1.0
HOST: itbilu.com
- 3.2) 服务器响应请求
Cookie 是一种 key=value 形式的字符串，服务器需要记录这个客户端请求的状态，因此在响应头中包一个 Set-Cookie 字段。响应头如下：
HTTP/1.0 200 OK
Set-Cookie: UserID=itbilu; Max-Age=3600; Version=1
Content-type: text/html
.....
- 3.3) 再次请求时，客户端请求中会包含一个 Cookie 请求头
客户端会对服务器响应的 Set-Cookie 头信息进行存储。再次请求时，将会在请求头中包含服务器响应的 Cookie 信息。请求头如下
GET / HTTP/1.0
HOST: itbilu.com
Cookie: UserID=itbilu

16、一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么（高薪常问）

- 1、浏览器查找域名对应的 IP 地址 (DNS 查询：浏览器缓存->系统缓存->路由器缓存->ISP DNS 缓存->根域名服务器)
- 2、浏览器向 Web 服务器发送一个 HTTP 请求 (TCP 三次握手)
- 3 服务器 301 重定向 (从 [HTTP://example.com](http://example.com) 重定向到 [HTTP://www.example.com](http://www.example.com))
- 4、浏览器跟踪重定向地址，请求另一个带 www 的网址
- 5、服务器处理请求 (通过路由读取资源)
- 6、服务器返回一个 HTTP 响应 (报头中把 Content-type 设置为 'text/html')
- 7、浏览器进 DOM 树构建
- 8、浏览器发送请求获取嵌在 HTML 中的资源 (如图片、音频、视频、CSS、JS 等)
- 9、浏览器显示完成页面
- 10、浏览器发送异步请求

17、你知道的 HTTP 请求方式有几种（高薪常问）

HttpRequestMethod 共计 17 种

- 1、GET 请求指定的页面信息，并返回实体主体。
- 2、HEAD 类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头
- 3、POST 向指定资源提交数据进行处理请求 (例如提交表单或者上传文件)。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。
- 4、PUT 从客户端向服务器传送的数据取代指定的文档的内容。
- 5、DELETE 请求服务器删除指定的页面。
- 6、CONNECT HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

- 7、OPTIONS 允许客户端查看服务器的性能。
- 8、TRACE 回显服务器收到的请求，主要用于测试或诊断。
- 9、PATCH 实体中包含一个表，表中说明与该 URI 所表示的原内容的区别。
- 10、MOVE 请求服务器将指定的页面移至另一个网络地址。
- 11、COPY 请求服务器将指定的页面拷贝至另一个网络地址。
- 12、LINK 请求服务器建立链接关系。
- 13、UNLINK 断开链接关系。
- 14、WRAPPED 允许客户端发送经过封装的请求。
- 15、LOCK 允许用户锁定资源，比如可以再编辑某个资源时将其锁定，以防别人同时对
其进行编辑。
- 16、MKCOL 允许用户创建资源
- 17、Extension-mothed 在不改动协议的前提下，可增加另外的方法。

18、什么是 TCP 连接的三次握手（高薪常问）

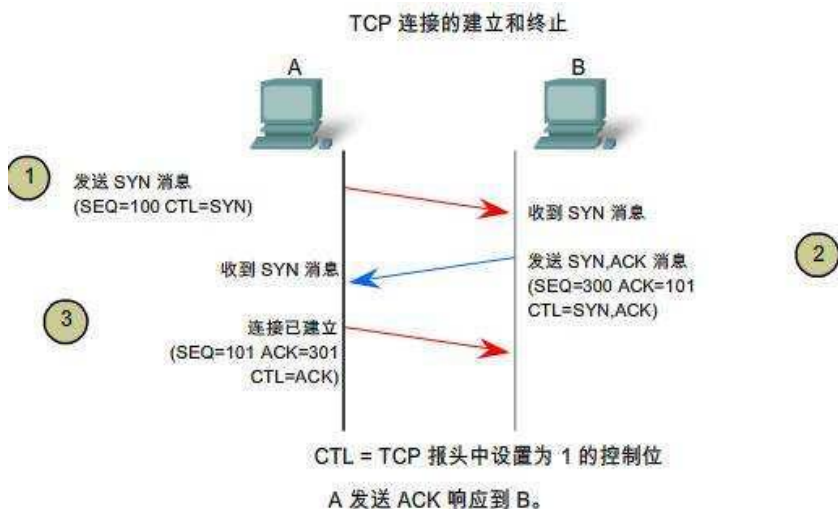
TCP 是因特网中的传输层协议，使用建立连接，完成三次握手，与服务器开始传送。

第一次握手：建立连接时，发送包（ $\text{syn}=j$ ）到服务器，并进入等待状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：[服务器](#)

第二次握手：收到 SYN 包，必须确认客户的 SYN（ $\text{syn}=j+1$ ），同时自己也发送一个 SYN 包（ $\text{syn}=k$ ），即 SYN+ACK 包，此时服务器进入等待状态；

第三次握手：客户端收到服务器的 SYN+ACK 包，向服务器发送确认包 ACK（[ack](#)= $k+1$ ），此包发送完毕，客户端和服务器进入（TCP 连接成功）状态，完成三次握手。



TCP 连接建立图

TCP 协议优点

TCP 发送的包有序号，对方收到包后要给一个反馈，如果超过一定时间还没收到反馈就自动执行超时重发，因此 TCP 最大的优点是可靠。

TCP 协议缺点

很简单，就是麻烦，如果数据量比较小的话建立连接的过程反而占了大头，不断地重发也会造成网络延迟，因此比如视频聊天通常就使用 UDP，因为丢失一些包也没关系，速度流

畅才是重要的。

19、为什么 TCP 连接需要三次握手四次挥手（高薪常问）

为什么是三次握手

为了防止已失效的连接请求报文段突然有送到了服务器，因而产生错误，假设两次握手时，客户发出的第一个请求连接报文段在某一网络节点长时间滞留，以致延误到连接释放后才到达服务器。服务器收到失效的连接请求报文段后，认为是客户又发出一次新的连接请求。于是向客户发送确认报文段，同意建立连接，此时在假定两次握手的前提下，连接建立成功。这样会导致服务器的资源白白浪费

为什么是四次挥手

TCP 协议是全双工通信，这意味着客户端和服务端都可以向彼此发送数据，所以关闭连接是双方都需要确认的共同行为，假设是三次挥手时，首先释放了客户到服务器方向的连接，此时 TCP 连接处于半关闭状态，这时客户不能向服务器发送数据，而服务器还是可以向客户发送数据。如果此时客户收到了服务器的确认报文段后，就立即发送一个确认报文段，这会导致服务器向客户还在发送数据时连接就被关闭。这样会导致客户没有完整收到服务器所发的报文段

20、TCP 与 UDP 的区别有哪些（高薪常问）

什么是 TCP

TCP (Transmission Control Protocol 传输控制协议) 是一种面向连接的、可靠的、基于字节流的传输层通信协议

什么是 UDP

UDP (User Datagram Protocol 用户数据报协议) 是 OSI (Open System Interconnection, 开放式系统互联) 参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务

区别

TCP 是面向连接的传输控制协议，而 UDP 提供了无链接的数据报服务//类似电话与短信

TCP 面向连接，提供可靠的数据服务

TCP 首部开销 20 字节, UDP 首部开销 8 字节

TCP 逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道

UDP 没有拥塞机制，因此网络出现拥堵不会使源主机的发送效率降低（有利于实时会议视频等）

TCP 的连接只能是点到点的, UDP 支持一对一，多对一，多对多的交互通信

21、介绍一下 websocket（高薪常问）

什么是 websocket

websocket 是一种网络通信协议，是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通信的协议，这个对比着 HTTP 协议来说，HTTP 协议是一种无状态的、无连接的、单向的应用层协议，通信请求只能由客户端发起，服务端对请求做出应答处理。HTTP 协议无法实现服务器主动向客户端发起消息，websocket 连接允许客户端和服务端之间进行全双工通信，以便任何一方都可以通过建立的连接将数据推送到另一端。websocket 只需要建立一次连接，就可以一直保持连接状态


```

<script>
// 初始化一个 WebSocket 对象
var ws = new WebSocket("ws://localhost:9998/echo");
// 建立 web socket 连接成功触发事件
ws.onopen = function () {
// 使用 send() 方法发送数据
ws.send("发送数据");
alert("数据发送中..."); };
// 接收服务端数据时触发事件
ws.onmessage = function (evt) { var received_msg = evt.data; alert("数据已接收..."); };
// 断开 web socket 连接成功触发事件
ws.onclose = function () { alert("连接已关闭..."); };

</script>

```

22、拆解一下 URL 的各个部分，分别是什么意思（高薪常问）

例如：scheme://host:port/path?query#fragment

-
- 1、. scheme:通信协议，常用的 HTTP, ftp, mailto 等
 - 2、. host:主机，服务器(计算机)域名系统（DNS）主机名或 IP 地址
 - 3、. port:端口号，整数，可选，省略时使用方案的默认端口，如 HTTP 的默认端口为 80
 - 4、. path:路径，由零或多个“/”符号隔开的字符串，一般用来表示主机上的一个目录或文件地址
 - 5、. query:查询，可选，用于给动态网页传递参数，可有多参数，用“&”符号隔开，每个参数的名和值用“=”符号隔开
 - 6、. fragment:信息片断，字符串，用于指定网络资源中的片断。例如一个网页中有多个名词解释，可使用 fragment 直接定位到某一名词解释。（也称为锚点）

23、HTTP 缓存机制（高薪常问）

浏览器缓存也包含很多内容：

HTTP 缓存、indexDB、cookie、localStorage 等等。这里我们只讨论 HTTP 缓存相关内容。

浏览器缓存分为强缓存和协商缓存

强缓存

是利用 http 的返回头中的 Expires 或者 Cache-Control 两个字段来控制的，用来表示资源的缓存时间。

Expires

缓存过期时间，用来指定资源到期的时间，是服务器端的具体时间点。也就是说，Expires=max-age + 请求时间，需要和 Last-modified 结合使用。但在上面我们提到过，cache-control 的优先级更高。Expires 是 Web 服务器响应消息头字段，在响应 http 请求时告诉浏览器在过期时间前浏览器可以直接从浏览器缓存取数据，而无需再次请求。

Cache-Control

Cache-Control 是一个相对时间，例如 Cache-Control:3600，代表着资源的有效期是 3600 秒。由于是相对时间，并且都是与客户端时间比较，所以服务器与客户端时间偏差也不会导致问题。

Cache-Control 与 Expires 可以在服务端配置同时启用或者启用任意一个，同时启用的时候 Cache-Control 优先级高。

协商缓存：

304 在第一次请求时候，返回状态码 200，和响应头中返回 cache-Control，控制缓存使用的时间/方式(私有缓存和共享缓存)，在第二次发起请求时，先查看 max-age 如果过期了，在请求头设置 If-None-Match 等于刚刚 Etag 的值，去后台对比，如果 etag 值相同证明后端没更新，所以返回 304 状态，前端提取本地的缓存继续使用。（也是协商缓存）

ES6

1、 ES5 和 ES6 的区别，说几个 ES6 的新增方法（必会）

ES5 和 ES6 的区别

ECMAScript5，即 ES5，是 ECMAScript 的第五次修订，于 2009 年完成标准化

ECMAScript6，即 ES6，是 ECMAScript 的第六次修订，于 2015 年完成，也称 ES2015

ES6 是继 ES5 之后的一次改进，相对于 ES5 更加简洁，提高了开发效率

ES6 的新增方法

1、新增声明命令 let 和 const

在 ES6 中通常用 let 和 const 来声明，let 表示变量、const 表示常量

1.1) 特点

let 和 const 都是块级作用域。以 {} 代码块作为作用域范围 只能在代码块里面使用

不存在变量提升，只能先声明再使用，否则会报错。语法上，称为“暂时性死区”

在同一个代码块内，不允许重复声明

const 声明的是一个只读常量，在声明时就需要赋值。（如果 const 的是一个对象，对象所包含的值是可以被修改的。抽象一点儿说，就是对象所指向的地址不能改变，而变量成员是可以修改的。）

2、模板字符串（Template String）

用一对反引号(`)标识，它可以当作普通字符串使用，也可以用来定义多行字符串，也可以在字符串中嵌入变量，js 表达式或函数，变量、js 表达式或函数需要写在`\${}`中。

3、函数的扩展

3.1) 函数的默认参数

ES6 为参数提供了默认值。在定义函数时便初始化了这个参数，以便在参数没有被传递

进去时使用。

3.2) 箭头函数

在 ES6 中，提供了一种简洁的函数写法，我们称作“箭头函数”。

3.2.1) 写法

函数名=(形参)=>{……} 当函数体中只有一个表达式时，{} 和 return 可以省略 当函数体中形参只有一个时，() 可以省略。

3.2.2) 特点

箭头函数中的 this 始终指向箭头函数定义时的离 this 最近的一个函数，如果没有最近的函数就指向 window。

4、对象的扩展

4.1) 属性的简写

ES6 允许在对象之中，直接写变量。这时，属性名为变量名，属性值为变量 的值。

```
var foo = 'bar';
var baz = {foo}; //等同于 var baz = {foo: foo};
```

方法的简写。省略冒号与 function 关键字。

```
var o = {
  method() {
    return "Hello!";
  }
};
// 等同于
var o = {
  method: function() {
    return "Hello!";
  }
};
```

4.2) Object.keys() 方法

获取对象的所有属性名或方法名（不包括原形的内容），返回一个数组。

```
var obj={name: "john", age: "21", getName: function () { alert(this.name)}};
console.log(Object.keys(obj)); // ["name", "age", "getName"]
console.log(Object.keys(obj).length); //3
console.log(Object.keys(["aa", "bb", "cc"])); //["0", "1", "2"]
console.log(Object.keys("abcdef")); //["0", "1", "2", "3", "4", "5"]
```

4.3) Object.assign ()

assign 方法将多个原对象的属性和方法都合并到了目标对象上面。可以接收多个参数，第一个参数是目标对象，后面的都是源对象

```
var target = {}; //目标对象
var source1 = {name : 'ming', age: '19'}; //源对象 1
var source2 = {sex : '女'}; //源对象 2
var source3 = {sex : '男'}; //源对象 3, 和 source2 中的对象有同名属性 sex
Object.assign(target, source1, source2, source3);
console.log(target); // {name : 'ming', age: '19', sex: '男'}
```

5、for...of 循环

```
var arr=["小林","小吴","小佳"];
for(var v of arr){
  console.log(v);
}
```

//小林 //小吴 //小佳

6、import 和 export

ES6 标准中，JavaScript 原生支持模块(module)。这种将 JS 代码分割成不同功能的小块进行模块化，将不同功能的代码分别写在不同文件中，各模块只需导出公共接口部分，然后通过模块的导入的方式可以在其他地方使用

export 用于对外输出本模块（一个文件可以理解为一个模块）变量的接口

import 用于在一个模块中加载另一个含有 export 接口的模块

import 和 export 命令只能在模块的顶部，不能在代码块之中

7、Promise 对象

Promise 是异步编程的一种解决方案，将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数，要是为了解决异步处理回调地狱（也就是循环嵌套的问题）而产生的

Promise 构造函数包含一个参数和一个带有 resolve（解析）和 reject（拒绝）两个参数的回调。在回调中执行一些操作（例如异步），如果一切都正常，则调用 resolve，否则调用 reject。对于已经实例化过的 Promise 对象可以调用 Promise.then() 方法，传递 resolve 和 reject 方法作为回调。then() 方法接收两个参数：onResolve 和 onReject，分别代表当前 Promise 对象在成功或失败时

Promise 的 3 种状态

Fulfilled 为成功的状态，Rejected 为失败的状态，Pending 既不是 Fulfilled 也不是 Rejected 的状态，可以理解为 Promise 对象实例创建时候的初始状态

7、 解构赋值

8.1) 数组的解构赋值

解构赋值是对赋值运算符的扩展。

是一种针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。

在代码书写上简洁且易读，语义更加清晰明了；也方便了复杂对象中数据字段获取。

数组中的值会自动被解析到对应接收该值的变量中，数组的解构赋值要一一对应如果有对应不上就是 undefined

```
let [a, b, c] = [1, 2, 3];  
// a = 1 // b = 2 // c = 3
```

8.2) 对象的解构赋值

对象的解构赋值和数组的解构赋值其实类似，但是数组的数组成员是有序的

而对象的属性则是无序的，所以对象的解构赋值简单理解是等号的左边和右边的结构相同

```
let { foo, bar } = { foo: 'aaa', bar: 'bbb' }; // foo = 'aaa' // bar = 'bbb'  
let { baz : foo } = { baz : 'ddd' }; // foo = 'ddd'
```

9、Set 数据结构

Set 数据结构，类似数组。所有的数据都是唯一的，没有重复的值。它本身是一个构造函数。

9.1) Set 属性和方法

Size () 数据的长度

Add () 添加某个值，返回 Set 结构本身。

Delete () 删除某个值，返回一个布尔值，表示删除是否成功。

Has () 查找某条数据，返回一个布尔值。

Clear () 清除所有成员，没有返回值。

9.2) 主要应用场景：数组去重

10、class

class 类的继承 ES6 中不再像 ES5 一样使用原型链实现继承，而是引入 Class 这个概念

ES6 所写的类相比于 ES5 的优点：

区别于函数，更加专业化（类似于 JAVA 中的类）

写法更加简便，更加容易实现类的继承

11、...

展开运算符可以将数组或对象里面的值展开；还可以将多个值收集为一个变量

12、async、await

使用 async/await，搭配 Promise，可以通过编写形似同步的代码来处理异步流程，提高代码的简洁性和可读性 async 用于申明一个 function 是异步的，而 await 用于等待一个异步方法执行完成

13、修饰器

@decorator 是一个函数，用来修改类甚至是方法的行为。修饰器本质就是编译时执行的函数

14、Symbol

Symbol 是一种基本类型。Symbol 通过调用 symbol 函数产生，它接收一个可选的名字参数，该函数返回的 symbol 是唯一的

15、Proxy

Proxy 代理使用代理（Proxy）监听对象的操作，然后可以做一些相应事情

2、ES6 的继承和 ES5 的继承有什么区别（必会）

ES6 的继承和 ES5 的继承的区别

ES5 的继承是通过原型或者是构造函数机制来实现

ES6 用过 class 关键字定义类，里面有构造方法，类之间通过 extends 关键字实现，子类必须在 constructor 方法中调用 super 方法

3、var、let、const 之间的区别（必会）

区别

var 声明变量可以重复声明，而 let 不可以重复声明

var 是不受限于块级的，而 let 是受限于块级

var 会与 window 相映射（会挂一个属性），而 let 不与 window 相映射

var 可以在声明的上面访问变量，而 let 有暂存死区，在声明的上面访问变量会报错

const 声明之后必须赋值，否则会报错

const 定义不可变的量，改变了就会报错

const 和 let 一样不会与 window 相映射、支持块级作用域、在声明的上面访问变量会报错

4、class、extends 是什么，有什么作用（必会）

什么是 class，class 的作用

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。

什么是 extends，extends 的作用

extends 是 ES6 引入的关键字，其本质仍然是构造函数+原型链的组合式继承。

class 类可以通过 extends 实现继承。

class 和 ES5 构造函数的不同点

- 1、类的内部定义的所有方法，都是不可枚举的。
- 2、ES6 的 class 类必须用 new 命令操作，而 ES5 的构造函数不用 new 也可以执行。
- 3、ES6 的 class 类不存在变量提升，必须先定义 class 之后才能实例化，不像 ES5 中可以将构造函数写在实例化之后。
- 4、ES5 的继承，实质是先创造子类的实例对象 this，然后再将父类的方法添加到 this 上面。ES6 的继承机制完全不同，实质是先将父类实例对象的属性和方法，加到 this 上面（所以必须先调用 super 方法），然后再用子类的构造函数修改 this。

5、module、export、import 有什么作用（必会）

module、export、import 是 ES6 用来统一前端模块化方案的设计思路和实现方案。

export、import 的出现统一了前端模块化的实现方案，整合规范了浏览器/服务端的模块化方法，用来取代传统的 AMD/CMD、requireJS、seaJS、commonJS 等等一系列前端模块不同的

实现方案，使前端模块化更加统一规范，JS 也能更加能实现大型的应用程序开发。
import 引入的模块是静态加载（编译阶段加载）而不是动态加载（运行时加载）。
import 引入 export 导出的接口值是动态绑定关系，即通过该接口，可以取到模块内部实时的值。

6、使用箭头函数应注意什么/箭头函数和普通函数的区别（必会）

区别

用了箭头函数，this 就不是指向 window，而是父级（指向是可变的）
不能够使用 arguments 对象
不能用作构造函数，这就是说不能够使用 new 命令，否则会抛出一个错误
不可以使用 yield 命令，因此箭头函数不能用作 Generator 函数

7、ES6 的模板字符串有哪些新特性？并实现一个类模板字符串的功能（必会）

模板字符串新特性

基本的字符串格式化。将表达式嵌入字符串中进行拼接。用 \${} 来界定
在 ES5 时我们通过反斜杠 (/) 来做多行字符串或者字符串一行行拼接。ES6 反引号 (`) 就能解决

类模板字符串的功能

实现一个类模板字符串的功能

```
let name = 'sunny';  
let age = 21;  
let str = '你好，${name} 已经 ${age} 岁了'  
str = str.replace(/\${([^\}]*\})/g, function() {  
    return eval(arguments[1]);  
})  
console.log(str); // 你好，sunny 已经 21 岁了
```

8、介绍下 Set、Map 的区别（必会）

区别

应用场景 Set 用于数据重组，Map 用于数据储存

Set:

成员不能重复

只有键值没有键名，类似数组

可以遍历，方法有 add, delete, has

Map:

本质上是键值对的集合，类似集合

可以遍历，可以跟各种数据格式转换

9、setTimeout、Promise、Async/Await 的区别（必会）

事件循环中分为宏任务队列和微任务队列

宏任务（macrotask）：在新标准中叫 task

主要包括：script(整体代码), setTimeout, setInterval, setImmediate, I/O, ui rendering

微任务（microtask）：在新标准中叫 jobs

主要包括：process.nextTick, Promise, MutationObserver (html5 新特性)

setTimeout、Promise、Async/Await 的区别

setTimeout 的回调函数放到宏任务队列里，等到执行栈清空以后执行

Promise.then 里的回调函数会放到相应宏任务的微任务队列里，等宏任务里面的同步代码执行完再执行

async 函数表示函数里面可能会有异步方法，await 后面跟一个表达式

async 方法执行时，遇到 await 会立即执行表达式，然后把表达式后面的代码放到微任务队列里，让出执行栈让同步代码先执行

10、Promise 有几种状态，什么时候会进入 catch？（必会）

Promise 有几种状态

三个状态：pending、fulfilled、reject

两个过程：padding -> fulfilled、padding -> rejected

Promise 什么时候会进入 catch

当 pending 为 rejected 时，会进入 catch

11、ES6 怎么写 class，为何会出现 class（必会）

什么是 class，class 的作用

ES6 的 class 可以看作是一个语法糖，它的绝大部分功能 ES5 都可以做到，新的 class 写法只是让对象原型的写法更加清晰、更像面向对象编程的语法

ES6 怎么写 class

```
//定义类
class Point {
  constructor(x, y) {
    //构造方法
    this.x = x; //this 关键字代表实例对象
    this.y = y;
  } toString() {
    return '(' + this.x + ',' + this.y + ')';
  }
}
```

12、如何获取多个 Promise 最后整体结果？（必会）

使用 Promise.all()

Promise.all() 用于将多个 Promise 实例，包装成一个新的 Promise 实例

Promise.all() 接受一个数组作为参数，数组里的元素都是 Promise 对象的实例，如果不是，就会先调用下面讲到的 Promise.resolve()，将参数转为 Promise 实例，再进一步处理。

(Promise.all() 方法的参数可以不是数组, 但必须具有 Iterator 接口, 且返回的每个成员都是 Promise 实例。)

示例: `var p = Promise.all([p1, p2, p3])`

p 的状态由 p1、p2、p3 决定, 分为两种情况。

当该数组里的所有 Promise 实例都进入 Fulfilled 状态: Promise.all**返回的实例才会变成 Fulfilled 状态。并将 Promise 实例数组的所有返回值组成一个数组, 传递给 Promise.all 返回实例的回调函数**。

当该数组里的某个 Promise 实例都进入 Rejected 状态: Promise.all 返回的实例会立即变成 Rejected 状态。并将第一个 rejected 的实例返回值传递给 Promise.all 返回实例的回调函数

13、ES6 如何转化为 ES5, 为什么要转化 (必会)

ES6 语法为什么要转化 ES5 语法

ECMAScript2015, 更新语法、规则、功能, 浏览器对 ES6 的支持程度并不是很好, 如果写了 ES6 的代码, 需要运行在浏览器上的时候, 需要将 ES6 的代码转成 ES5 的代码去浏览器上运行。

Babel 是什么

babel 是一个 ES6 转码器, 可以将 ES6 代码转为 ES5 代码, 以便兼容那些还没支持 ES6 的平台

ES6 如何转化为 ES5

1、使用npm安装转换插件babel-cli, 在js文件中引入插件

2、创建babel-cli配置文件.babelrc, 输入以下内容

```
1 {
2   "presets": [
3     "es2015" // 定义转换规则
4   ],
5   "plugins": [
6   ]
7 }
```

3、在终端输入**babel src/index.js -o dist/index.js** (src为开发路径, 即ES6所在目录, dist为转换后ES5路径)

babel-polyfill
Babel默认只转换新的javascript语法, 但并不转换新的API, 比如 Generator、Set、Symbol、promise等全局对象, 以及一些定义在全局对象上的方法都不会转码。如果让这些方法运行则必须使用babel-polyfill。
安装

```
$ npm install --save babel-polyfill
```

在js中使用

```
require("babel-polyfill");
```

将Babel集成到webpack中
Babel配置
1) 安装babel-loader与babel-core

```
$ npm install babel-core babel-loader --save-dev
```


2)安装预设

```
$ npm install babel-preset-latest --save-dev
```

3)配置.babelrc

```
{"presets":["latest"]}
```

webpack配置

1)安装webpack

```
$ npm install webpack webpack-cli --save-dev
```

2)添加配置文件 webpack.config.js

```
const path=require('path');
module.exports={
  entry: './index.js',
  output:{
    filename:'bundle.js',
    path:path.resolve(__dirname,'dist')
  },
  module:{
    rules:[{
      test:/\.js$/,
      use:'babel-loader'
    }]
  }
}
```

3) 修改package.json

```
"scripts":{
  "build":"webpack"
}
```

4)打包

```
$ npm run build
```

可能会报如下错误:

cdCannot find module '@babel/core' babel-loader@8 requires Babel 7.x.如果按我上面步骤我们装的babel-loader是8.0.4版本, 因为我们只需要重新装7版本。

```
npm install babel-loader@7 --save-dev
```

14、日常前端代码开发中，有哪些值得用 ES6 去改进的编程优化或者规范（必会）

- 1、常用箭头函数来取代 `var self = this;` 的做法。
- 2、常用 `let` 取代 `var` 命令。
- 3、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。
- 4、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。
- 5、用 `class` 类取代传统的构造函数，来生成实例化对象。
- 6、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。

15、ES6 和 node 的 commonjs 模块化规范的区别（高薪常问）

ES6 是 js 的增强版，是 js 的语法规则，commonjs 都只是为了解决 js 文件之间的依赖和引用问题，所以是一种 js 的包管理规范，其中的代表是 node 遵循 commonjs 规范

16、Promise 中 reject 和 catch 处理上有什么区别（高薪常问）

`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

17、理解 async/await 以及相对 Generator 的优势

理解 async await

`async await` 是用来解决异步的，`async` 函数是 `Generator` 函数的语法糖
使用关键字 `async` 来表示，在函数内部使用 `await` 来表示异步

`async` 函数返回一个 `Promise` 对象，可以使用 `then` 方法添加回调函数

当函数执行的时候，一旦遇到 `await` 就会先返回，等到异步操作完成，再接着执行函数体内后面的语句

async 较 Generator 的优势

1、内置执行器

`Generator` 函数的执行必须依靠执行器，而 `Async` 函数自带执行器，调用方式跟普通函数的调用一样

2、更好的语义

`async` 和 `await` 相较于 `*` 和 `yield` 更加语义化

3、更广的适用性

`yield` 命令后面只能是 `Thunk` 函数或 `Promise` 对象，`async` 函数的 `await` 后面可以是 `Promise` 也可以是原始类型的值

4、返回值是 Promise

`async` 函数返回的是 `Promise` 对象，比 `Generator` 函数返回的 `Iterator` 对象方便，可

以直接使用 `then()` 方法进行调用

`generator` 函数就是一个封装的异步任务，也就是异步任务的容器，执行 `Generator` 函数会返回一个遍历器对象，`async` 函数的实现，就是将 `Generator` 函数和自动执行器，包装在一个函数里

18、手写一个 Promise（高薪常问）

```
var Promise = new Promise((resolve, reject) => {
  if (操作成功) {
    resolve(value)
  } else {
    reject(error)
  }
})
Promise.then(function (value) {
  // success
}, function (value) {
  // failure
})
```

如果是要自己模拟一个

```
function MyPromise(fn) {
  this.callBackFnArr = []; // 2. 用来装 then 里的回调函数
  const resolve = (value) => {
    setTimeout(() => { // 6. 在这里改装，确保在 then 调用后，再执行这里
      this.callBackFnArr.map(v => v(value)); // 5. 遍历数组里 then 里的回调函数执行(注意代码执行顺序，这时候 pro.then() 还没执行呢)
    });
  }
  fn(resolve);
}

MyPromise.prototype.then = function(thenFn) { // 1. 定义 then 方法，把 then 里要执行的函数加入到数组中
  this.callBackFnArr.push(thenFn);
}

let pro = new MyPromise(resolve => { // 3. 这里的回调函数马上执行
  resolve(123); // 4. 调用 4 行内部 resolve 触发
});

pro.then(result => { // ? 此时还没有添加 then 函数，上面就走完了
  console.log(result);
})
```

19、Promise 如何封装一个 Ajax（高薪常问）

```
function ajax(optionsOverride) {  
  // 将传入的参数与默认设置合并  
  var options = {};  
  for (var k in ajaxOptions) {  
    options[k] = optionsOverride[k] || ajaxOptions[k];  
  }  
  options.async = options.async === false ? false : true;  
  var xhr = options.xhr = options.xhr || new XMLHttpRequest();  
  
  return new Promise(function (resolve, reject) {  
    xhr.open(options.method, options.url, options.async);  
    xhr.timeout = options.timeout;  
  
    //设置请求头  
    for (var k in options.headers) {  
      xhr.setRequestHeader(k, options.headers[k]);  
    }  
    // 注册xhr对象事件  
    xhr.onprogress = options.onprogress;  
    xhr.upload.onprogress = options.onuploadprogress;  
    xhr.responseType = options.dataType;  
  
    xhr.onabort = function () {  
      reject(new Error({  
        errorType: 'abort_error',  
        xhr: xhr  
      }));  
    };  
    xhr.ontimeout = function () {  
      reject({  
        errorType: 'timeout_error',  
        xhr: xhr  
      });  
    };  
    xhr.onerror = function () {  
      reject({  
        errorType: 'onerror',  
        xhr: xhr  
      });  
    };  
    xhr.onloadend = function () {  
      if ((xhr.status >= 200 && xhr.status < 300) || xhr.status === 304)  
        resolve(xhr);  
      else  
        reject({  
          errorType: 'status_error',  
          xhr: xhr  
        });  
    };  
    try {  
      xhr.send(options.data);  
    }  
    catch (e) {  
      reject({  
        errorType: 'send_error',  
        error: e  
      });  
    }  
  })  
}
```

20、下面的输出结果是多少（高薪常问）

```
const Promise = new Promise((resolve, reject) => {  
  console.log(2);  
  resolve();  
  console.log(333);  
})  
Promise.then(() => {  
  console.log(666);  
})  
console.log(888);
```

解析：Promise 新建后立即执行，所以会先输出 2，333，而 Promise.then() 内部的代码在当次 事件循环的 结尾 立刻执行，所以会继续输出 888，最后输出 666

21、以下代码依次输出的内容是（高薪常问）

```
setTimeout(function () {  
  console.log(1)  
}, 0);  
  
new Promise(function executor(resolve) {  
  console.log(2);  
  for (var i = 0; i < 10000; i++) {  
    i == 9999 && resolve();  
  }  
  console.log(3);  
}).then(function () {  
  console.log(4);  
});  
  
console.log(5);
```

解析：首先碰到一个 `setTimeout`，于是会先设置一个定时，在定时结束后将传递这个函数放到任务队列里面，因此开始肯定不会输出 1。
然后是一个 `Promise`，里面的函数是直接执行的，因此应该直接输出 2 3。
然后，`Promise` 的 `then` 应当会放到当前 `tick` 的最后，但是还是在当前 `tick` 中。
因此，应当先输出 5，然后再输出 4，最后在到下一个 `tick`，就是 1。

22、分析下列程序代码，得出运行结果，解释其原因（高薪常问）

```
const Promise = new Promise((resolve, reject) => {  
  console.log(1)  
  resolve()  
  console.log(2)  
})  
Promise.then(() => {  
  console.log(3)  
})  
console.log(4)  
运行结果：1 2 4 3
```

解析：`Promise` 构造函数是同步执行的，`Promise.then` 中的函数是异步执行的。

23、分析下列程序代码，得出运行结果，解释其原因（高薪常问）

```
const Promise = new Promise((resolve, reject) => {  
  resolve('success1')  
  reject('error')  
  resolve('success2')  
})  
Promise  
  .then((res) => {  
    console.log('then: ', res)  
  })
```

```
.catch((err) => {  
  console.log('catch: ', err)  
})
```

运行结果: then: success1

解析: 构造函数中的 resolve 或 reject 只有第一次执行有效, 多次调用没有任何作用,
呼 应代码二结论: Promise 状态一旦改变则不能再变。

24、使用结构赋值, 实现两个变量的值的交换 (高薪常问)

```
let a = 1;  
let b = 2;  
[a, b] = [b, a];
```

25、说一下 ES6 的导入导出模块 (高薪常问)

导入模块

通过 import 关键字

// 只导入一个

```
import {sum} from './example.js'
```

// 导入多个

```
import {sum, multiply, time} from './exportExample.js'
```

// 导入一整个模块

```
import * as example from './exportExample.js'
```

导出模块

导出通过 export 关键字

// 可以将 export 放在任何变量, 函数或类声明的前面

```
export var firstName = 'Chen';
```

```
export var lastName = 'Sunny';
```

```
export var year = 1998;
```

// 也可以使用大括号指定所要输出的一组变量

```
var firstName = 'Chen';
```

```
var lastName = 'Sunny';
```

```
var year = 1998;
```

```
export {firstName, lastName, year};
```

// 使用 export default 时, 对应的 import 语句不需要使用大括号

```
let bosh = function crs() {}
```

```
export default bosh;
```

```
import crc from 'crc';
```

// 不使用 export default 时, 对应的 import 语句需要使用大括号

```
let bosh = function crs() {}
```

```
export bosh;
```

```
import {crc} from 'crc';
```

git

1、git 的基本使用方法（必会）

第一步：window 本机电脑安装 git 软件（只需要一次）

第二步：配置环境变量（只需要一次）

安装到 D:\software\git\目录,把 bin 目录路径完整加入 Path 变量。D:\software\git\bin

第三步：配置 git 的 config（只需要一次）

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

查看你的配置是 `git config --list`

第四步：使用 git 开始工作（每次）

1、在本地建立一个文件夹，作为本地代码仓库，并初始化 cmd 中 cd 到该文件夹，执行 `git init` 命令，让该文件夹成为受 git 管理的仓库目录。

2、把某个文件添加到本地仓库（前提项目文件夹有这个文件）

执行 `git add HelloWorld.html` 命令

如果暂存所有（`git add .`）

3、提交文件到仓库

```
git commit -m "第一次使用 git 提交文件" m 后面的 " " 可以写上备注信息的
```

2、git 工作流程（必会）

git 的作用

1、在工作目录中修改某些文件

2、对修改后的文件进行快照，然后保存到暂存区域

3、提交更新，将保存在暂存区域的文件快照永久转储到 git 目录中

git 的工作中使用场景：

两个分支 master 和 dev

项目开始执行流程

`git branch -a`（查看所有分支）

0、克隆代码 `git clone 地址`

1、拉取线上 master 最新代码：`git pull origin master`

2、切换到开发分支：`git checkout dev`

3、合并 master 本地分支（master）：`git merge master`

4、开始开发

5、开发结束

6、查看当前文件更改状态：`git status`

7、把所有更改代码放到缓存区：`git add -A`

8、查看当前文件更改状态：`git status`

9、缓存区内容添加到仓库中：`git commit -m '本次更改注释'`

10、把代码传到 gitLab 上：`git push origin dev`

11、若代码到达上线标准则合并代码到 master，切换分支到 master：`git checkout master`

12、拉取 master 最新分支：`git pull origin master`

13、合并分支代码到 master（若有冲突则解决冲突）：`git merge dev`

14、把当前代码上传到 gitLab：`git push origin master`

15、代码上线后，用 tag 标签标记发布结点（命名规则：prod_+版本_+上线日期）

```
git tag -a prod_V2.1.8_20200701
```

16、tag 标签推到 gitLab

```
git push origin prod_V2.1.8_20200701
```

缓存区的应用

1、需要合并别人代码进来

1.1) 把自己的代码放入暂存: `git stash`

1.2) 如果需要释放出来用: `git stash pop`#恢复最近一次的暂存

1.3) 查看你有哪些队列: `git stash list`

1.4) 删除第一个队列, 以此可以类推: `git stash drop stash@{0}`

2、需要切换分支

2.1) `git add -A`

2.2) `git stash save 'demo'`

2.3) `git stash list`

2.4) `git stash apply stash@{0}`

补充指令

`git reflog` 查看提交记录命令:

`git show #` 显示某次提交的内容 `git show $id`

`git rm <file> #` 从版本库中删除文件

`git reset <file> #` 从暂存区恢复到工作文件

`git reset HEAD^ #` 恢复最近一次提交过的状态, 即放弃上次提交后的所有本次修改

`git diff <file> #` 比较当前文件和暂存区文件差异 `git diff`

`git log -p <file> #` 查看每次详细修改内容的 diff

`git branch -r #` 查看远程分支

`git merge <branch> #` 将 branch 分支合并到当前分支

`git stash pop git pull #` 抓取远程仓库所有分支更新并合并到本地

`git push origin master #` 将本地主分支推到远程主分支

`git branch 分支名#`创建分支

`git checkout 分支名#`切换分支

`git checkout -b 分支名#`创建并切换分支

`git branch --merge / git branch --no-merge#`查看已经合并的分支/未合并的分支

`git branch -d 分支名 / git branch -D 分支名#`删除的已合并的分支/未合并的分支

3、我们如何使用 git 和开源的码云或 github 上面的远端仓库的项目进行工作呢（必会）

客户端本地 git 如何和远程仓库码云, github 连接上次文件

git 仓库如 github 都是通过使用 SSH 与客户端连接的!

我们通过本地 git 生成生成密钥对后, 将公钥保存至 github, 每次连接时 SSH 客户端发送本地私钥（默认 ~/.ssh/id_rsa）到服务端验证。单用户情况下, 连接的服务器上保存的公钥和发送的私钥自然是配对的

命令如下: `ssh-keygen -t rsa -C 'XXX@qq.com' -f id_rsa_second`

或 `ssh-keygen -t rsa -C "XXX@qq.com"`

邮箱可以换成你的


```

$ ssh-keygen -t rsa -C "946846782@qq.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c:/Users/Admin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c:/Users/Admin/.ssh/id_rsa.
Your public key has been saved in /c:/Users/Admin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:wRl0MEfk3Rw46lZEyJ6SzPwshfCWGc7TBR+j+chVqDwM 946846782@qq.com
The key's randomart image is:
+--[RSA 2048]-----+
|
|oO=oo+
|o+B=B .
|=o^O++
|.B+=E.+ .
|+.oS= o
|oo+. =
|o+ . +
|. . .
+-----[SHA256]-----+

```

此电脑 > Win 10 Home x64 (C:) > 用户 > Admin > .ssh

名称	修改日期	类型	大小
id_rsa	2018/5/26 10:46	文件	2 KB
id_rsa.pub	2018/5/26 10:46	PUB 文件	1 KB
known_hosts	2018/5/26 10:38	文件	1 KB

添加公钥（id_rsa_second.pub）到你的远程仓库（github）

登陆你的 github 帐户。点击你的头像，然后 Settings -> 左栏点击 SSH and GPG keys

-> 点击 New SSH key

然后你复制上面的公钥内容，粘贴进“Key”文本域内。 title 域，自己随便起个名字。

点击 Add key。

完成以后，验证下这个 key 是不是正常工作：

```
$ ssh -T git@github.com
```

Attempts to ssh to github

如果，看到：

Hi xxx! You've successfully authenticated, but github does not # provide shell access.

表示设置已经成功

码云：

进入码云的设置页面



在终端(Terminal)中输入 `ssh -T git@gitee.com` 若返回 `Welcome to gitee.com, yourname!`

```
Admin@PV-X00237052 MINGW64 /ssh_key/.ssh
$ ssh -T git@gitee.com
Welcome to Gitee.com, xiaomai!

Admin@PV-X00237052 MINGW64 /ssh_key/.ssh
$
```

代表成功!

通常步骤:

本地新建仓库, 输入 `git init` 初始化, 让 `git` 接管

关联一个远程仓库: `git remote add origin git@github.com:XXXXXXXXX.git`

把文件添加到本地版本库

`git add 文件名`

把文件修改提交到本地仓库

`git commit -m"注释"`

`git pull origin master` 先将 `github` 上的代码 `pull` 下来

然后在 `git push origin master` 将最新的修改推送到远程仓库

`git -` 查看远程仓库信息

可以通过命令 `git remote show [remote-name]` 查看某个远程仓库的详细信息

4、git, github, gitlab 三者之间的联系以及区别（必会）

1、git

git 是一个版本控制系统。

版本控制是一种用于记录一个或多个文件内容变化，方便我们查阅特定版本修订情况的系统。

早期出现的版本控制系统有：svn、cvs 等，它们是集中式版本控制系统，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同合作的开发人员都通过客户端连接到这台服务器，取出最新的文件或者提交更新。

而我们的主角 git 是分布式版本控制系统。git 已经成为越来越多开发者的青睐，因为分布式的优势是很显著的。

2、集中式和分布式版本控制系统的区别：

2.1) 分布式版本控制系统下的本地仓库包含代码库还有历史库，在本地就可以查看版本历史

2.2) 而集中式版本控制系统下的历史仓库是存在于中央仓库，每次对比与提交代码都必须连接到中央仓库

2.3) 多人开发时，如果充当中央仓库的 git 仓库挂掉了，任何一个开发者都可以随时创建一个新的中央仓库然后同步就可以恢复中央仓库

2、github 和 gitlab

github 和 gitlab 都是基于 web 的 git 仓库，使用起来二者差不多，它们都提供了分享开源项目的平台，为开发团队提供了存储、分享、发布和合作开发项目的中心化云存储的场所。

github 作为开源代码库，拥有超过 900 万的开发者用户，目前仍然是最火的开源项目托管平台，github 同时提供公共仓库和私有仓库，但如果使用私有仓库，是需要付费的。

gitlab 解决了这个问题，你可以在上面创建私人的免费仓库。

gitlab 让开发团队对他们的代码仓库拥有更多的控制，相比较 github，

gitlab 特色

3.1) 允许免费设置仓库权限；

3.2) 允许用户选择分享一个 project 的部分代码；

3.3) 允许用户设置 project 的获取权限，进一步提升安全性；

3.4) 可以设置获取到团队整体的改进进度；

3.5) 通过 innersourcing 让不在权限范围内的人访问不到该资源；

所以，从代码的私有性上来看，gitlab 是一个更好的选择。但是对于开源项目而言，github 依然是代码托管的首选。

5、github 和码云的区别（必会）

github

全英文、用户基数多，知名库多、国内访问的话，偶尔会有不稳定，出现上不去的情况、私有项目需要付费

码云

全中文、用户量没有 github 多，知名库相对较少、服务器在国内，相对稳定、每个用户有 1000 个免费的私有项目、访问速度很快，支持 svn，git 两种方式、每个仓库有 1G 的容量限制

6、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（必会）

冲突是如何产生

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。

如何解决冲突

1、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。

2、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；

分支提交冲突：当分支对某文件某句话进行修改后，切换到主分支也对该文件该句话进行修改，使用 `git merge` 进行合并，需要将两个修改进行合并。此时合并产生冲突

3、另外一种解决方法

3.1) `git status` 查看冲突文件

3.2) 编辑器打开冲突文件，查看内容。Git 用<<<<<<, =====, >>>>>> 标记出不同分支的内容

3.3) 修改文件内容

3.4) 提交 `git add file ; git commit -m ""`

查看分支合并图 `git log -graph`

7、如果本次提交误操作，如何撤销（必会）

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。

8、git 修改提交的历史信息（必会）

git 修改提交的历史信息详细操作

`git rebase -i HEAD~3`

输出如下

pick 1 commit 1

pick 2 commit 2

pick 3 commit 3

要修改哪个，就把那行的 pick 改为 edit，然后退出。例如想修改 commit 1 的 author，光标移到第一个 pick，按 i 键进入 INSERT 模式，把 pick 改为 edit：

edit 1 commit 1

pick 2 commit 2

```
pick 3 commit 3
...
```

- INSERT -

然后按 `esc` 键，退出 INSERT 模式，输入 `:wq` 退出，这时可以看到提示，可以修改 `commit 1` 的信息了

输入 `amd` 命令重置用户信息： `$ git commit --amend --reset-author`

会出现 `commit 1` 的提交记录及注释内容，可进入 INSERT 模式修改注释，`:wq` 退出

这时再查看提交历史，发现 `commit 1` 的 `author` 已经变成 `b (b@email.com)` 了， 且是最新一次的记录

通过 `continue` 命令回到正常状态： `$ git rebase --continue`

9、如何删除 github 和 gitlab 上的文件夹（必会）

解决办法

重点在于 `git push -u`

方法一：

这里以删除 `.setting` 文件夹为案例

`git rm -r --cached .setting` `--cached` 不会把本地的 `.setting` 删除

`git commit -m 'delete .setting dir'`

`git push -u origin master`

方法二：

如果误提交的文件夹比较多，方法一也较繁琐

直接修改 `.gitignore` 文件, 将不需要的文件过滤掉，然后执行命令

`git rm -r --cached .`

`git add .`

`git commit`

`git push -u origin master`

10、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（必会）

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

11、git 跟 svn 有什么区别（必会）

`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

git 和 svn 的区别

git 是分布式版本控制，svn 是集中式版本控制（核心区别）

git 相对于 svn 的优势就是不需要网络即可版本控制

git 把内容按数据方式存储，而 svn 是按文件

git 可以是公用的，可以分享，svn 基本是公司内部才能访问，网外不方便访问

git 不依赖中央服务器，即使服务器有问题也不受影响，svn 依赖服务器，一旦服务器有问题就会受影响

git 没有一个全局的版本号，svn 有

12、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（必会）

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

```
$ vim .gitignore
```

```
$ cat .gitignore
```

```
/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、git fetch 和 git merge 和 git pull 的区别（必会）

区别如下

git pull 相当于 git fetch 和 git merge，即更新远程仓库的代码到本地仓库，然后将内容合并到当前分支。

git merge： 将内容合并到当前分支

git pull 相当于是从远程获取最新版本并 merge 到本地

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 相当于是从远程获取最新版本到本地，不会自动 merge

方便记忆：

git pull = git fetch + git merge

14、如何把本地仓库的内容推向一个空的远程仓库（高薪常问）

首先确保本地仓库与远程之间是连通的。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
```

大事件项目 PC 端

1、开发背景

1.1 项目介绍

大事件项目是一款文章信息类的项目，主要包含登陆功能，文章类别管理模块，文章列表模块，以及游客模块

2、系统架构

2.1 关键技术

2.1.1 ajax

AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML），AJAX 不是新的编程语言，而是一种使用现有标准的新方法，AJAX 最大的优点是在不重新加载整个页面的情况下，可以与服务器交换数据并更新部分网页内容，并且不需要任何浏览器插件，但需要用户允许 JavaScript 在浏览器上执行。

2.1.2 art-template

art-template 是一个简约、超快的模板引擎。它采用作用域预声明的技术来优化模板渲染速度，从而获得接近 JavaScript 极限的运行性能，并且同时支持 nodeJS 和浏览器。使用 art-template 也便于维护代码，以前我们进行数据渲染的时候是通过字符串拼接然后再通过 append 的方式追加到数据源 id 上，而用了模板引擎以后，我们只需要 html 文件中修改 html 内容。还有使用了模板引擎以后 DOM 操作的效率也会更高一点。

2.1.3 bootstrap

Bootstrap，来自 Twitter，是目前最受欢迎的前端框架。Bootstrap 是基于 HTML、CSS、JavaScript 的，它简洁灵活，使得 Web 开发更加快捷。

2.1.4 Git

Git 是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目，Git 是 Linux Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件，Git 与常用的版本控制工具 CVS，Subversion 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。

2.2 API 文档

用户认证（登录）

基本信息

Path: /mp/v1_0/authorizations

Method: POST

接口描述:

1. 线上地址

```
1 | http://ttapi.research.itcast.cn/mp/v1_0/authorizations
```

2. 返回HTTP状态码

1. 201 OK

2. 400 请求参数错误

包括：参数缺失、手机号格式不正确、验证码失效等

3. 403 用户非实名认证用户，无权限登录

4. 507 服务器数据库异常

3. token说明

1. **token** 用于访问需要身份认证的普通接口，有效期2小时

2. **refresh_token** 用于在token过期后，获取新的用户token，有效期14天

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
mobile	string	必须		手机号	
code	string	必须		验证码	

2.3 人员配置

产品经理：1 人，确定需求以及给出产品原型图。

项目经理：1 人，项目管理。

前端团队：2 人，根据产品经理给出的原型图制作静态页面。

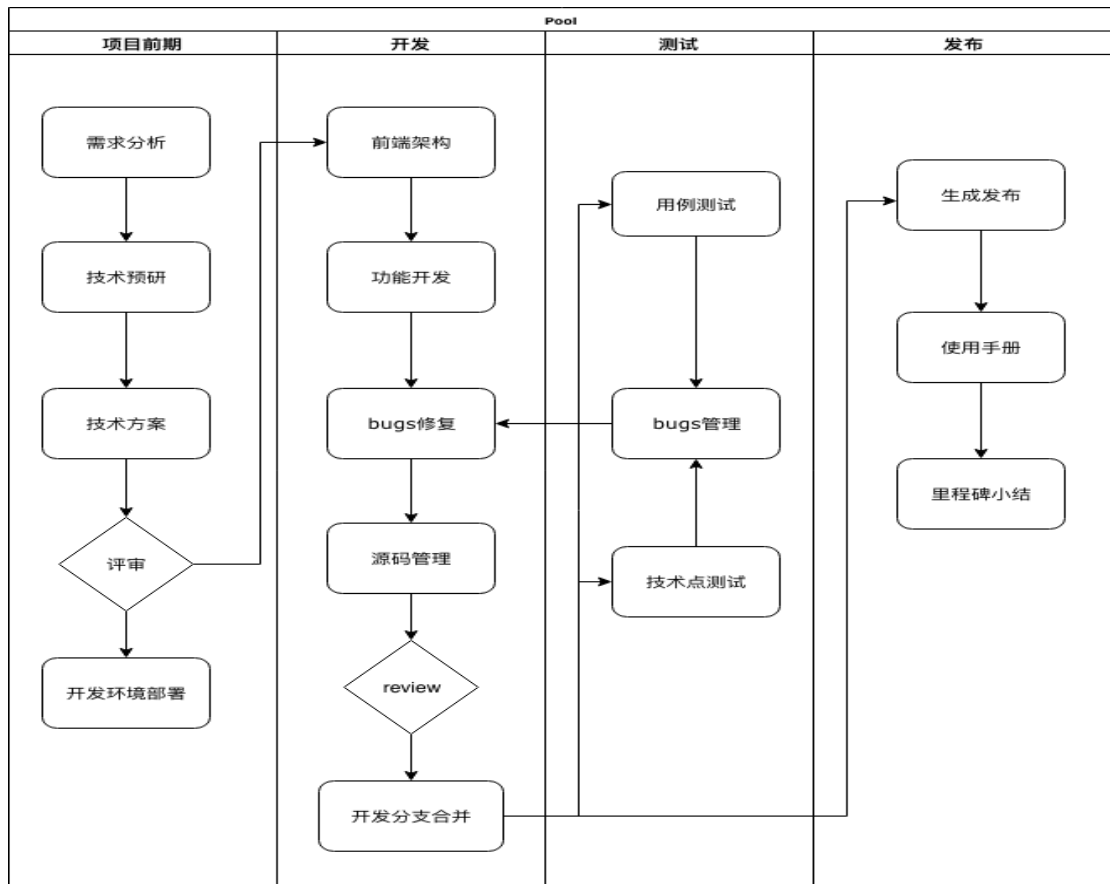
后端团队：2 人，根据项目经理分配的任务完成产品功能。

测试团队：0 人，前后端开发人员联调解决。

运维团队：1 人，项目的发布以及维护，其中小程序的上线和发布由前端团队完成。

2.4 开发流程

产品提出需求 -- 画出原型图 -- 需求评审会议 -- 安排工期(各部门制定) --- UI 设计
图 -- 前端开发 -- 后端开发(顺序不一定) -- 测试阶段 -- 上线 --- 回测碑小结 --->维护
项 目



3、登录模块

3.1 业务实现思路

- 1、统一封装单独接口文件，实现对业务分层
- 2、注册点击事件，在点击按钮时，触发接下来的操作
- 3、简单检验，匹配用户名和密码是否符合项目规则
- 4、通过校验后，点击按钮并且发送 ajax 请求，完成登录

3.2 技术亮点

api 封装，点击事件，登录校验，\$ajax

4、首页模块

4.1 业务实现思路

- 1、登录成功后，展示登录进来的用户信息，实现退出登录
- 2、调用接口数据，完成首页文章信息展示
- 3、使用 art-template 实现文章查询功能

4.2 技术亮点

Art-template 基本使用，接口封装

5、文章类别管理

5.1 业务实现思路

- 1、arttemplate 生成的动态页面，使用 js 动态绑定函数，并且传入删除的实参
- 2、获取用户参数，使用 \$.ajax 调用接口数据，实现添加功能，添加完毕后，关闭弹出框结构
- 3、添加 editTr 函数，获取参数并且展示，获取用户填入的数据，完成编辑功能

5.2 技术亮点

函数传参，文章删除，文章编辑，文章添加，函数封装

6、文章列表管理

6.1 业务实现思路

- 1、利用 twbs-pagination，配合 ajax 完成动态数据的渲染，以及分页效果
- 2、根据用户填入的分类，实现对文章的筛选
- 3、自定义属性，获取当前文章 ID 值，实现对当前文章的删除
- 4、根据 URL 地址栏中的 ID，实现编辑保存文章

6.2 技术亮点

分页功能，文章筛选，根据 ID 删除文章、插件的基本使用方式

7、前台页面文章获取

7.1 业务实现思路

- 1、通过封装好的单独 js 文件，可以是直接导入使用，实现业务逻辑
- 2、使用 ajax 结合 art-template 实现对文章的渲染
- 3、在每次完成业务功能后，使用 git 上传代码，实现对项目代码的管理

7.2 技术亮点

抽离 API，\$.ajax、art-template、git

8、项目介绍话术

首先描述自己所做项目有哪些功能模块，然后描述其中单个模块有哪些功能，再对其中的一个单独功能进行详细描述，中间可以穿插一下遇到的技术问题，循环往复，和面试官保持平等对话。

举例：

我在上家单位最近做的项目是一个基于 ajax 和 template 完成了一款文章咨询类的项目，其中包含登录功能，一文章列表渲染等功能，通过封装单独的 API 文件，为了保证业务逻辑之前更加清晰，配合 ajax 完成了对文章分类功能的数据渲染，以及实现删除文章，以及修改，添加文章等操作

Nodejs

1、对 Node. js 有没有了解，它有什么特点，适合做什么业务（必会）

什么是 Node

Node. js... 它既是开发平台，也是运行环境，也是个新的语言... 它本身是基于 google 的 javascript v8 引擎开发的，因此在编写基于它的代码的时候使用 javascript 语言。但是又不同于传统概念的 javascript... 它的服务端功能以及部分客户端功能必须在服务端运行，所以它实际上是一种在服务端的开发+运行的 javascript 语言。它本身可以作为 HTTP Server，也可以当作 TCP Server 用

特点

他是一个 javascript 运行环境，依赖于 Chrome V8 引擎进行代码解释

特征：单线程、事件驱动、非阻塞 I/O，轻量，可伸缩，适于实时数据交互应用

单进程，单线程（一个应用程序对应一个进程，一个进程下面会有多个线程，每个线程用于处理任务..）

Node 无法直接渲染静态页面，提供静态服务

Node 没有根目录的概念

Node 必须通过路由程序指定文件才能渲染文件

Node 比其他服务端性能更好，速度更快

适用业务

Node. js 是单线程，非阻塞 I/O，事件驱动，它的特点决定了它适合做一些大量 I/O 的东西，比如，聊天室，表单提交等不需要大量计算的功能。做一些微信后端开发，或者做消息系统等。可以整个项目用，也可以根据它的特点在某个模块使用，比如 socketio，打造一个消息系统等

2、Node 和前端项目怎么解决跨域的（必会）

设置 CORS 或者 使用使用 CORS 模块

通过在node服务器端设置

```
1 // 解决跨域问题
2 app.use(async(ctx, next) => {
3
4     // 指定服务器端允许进行跨域资源访问的来源域。可以用通配符*表示允许任何域的JavaScript访问资源，但是在响应一个请求
5     ctx.set("Access-Control-Allow-Origin", "*");
6
7     // 可选。它的值是一个布尔值，表示是否允许客户端跨域请求时携带身份信息(Cookie或者HTTP认证信息)。默认情况下，Coo
8     ctx.set("Access-Control-Allow-Credentials", true);
9
10    // 指定服务器端允许进行跨域资源访问的请求方法列表，一般用在响应预检请求上
11    ctx.set("Access-Control-Allow-Methods", "OPTIONS, GET, PUT, POST, DELETE");
12
13    // 必需。指定服务器端允许进行跨域资源访问的请求头列表，一般用在响应预检请求上
14    ctx.set("Access-Control-Allow-Headers", "x-requested-with, accept, origin, content-type");
15    // ctx.set("X-Powered-By", ' 3.2.1');
16
17    // 告诉客户端返回数据的MIME的类型，这只是一个标识信息，并不是真正的数据文件的一部分
18    ctx.set("Content-Type", "application/json;charset=utf-8");
19
20    // 如果不设置mode，直接设置content-type为application/json，则fetch会默认这是跨域模式（mode:'cors'），在
21    if (ctx.request.method === "OPTIONS") {
22        ctx.response.status = 200
23    }
24    await next();
25 }}
```

3、Node 的优点是什么？缺点是什么（必会）

优点

- 1、高并发（最主要的一个优点）
- 2、适合 I/O 密集型应用

缺点

1、不适合 CPU 密集型应用；CPU 密集型应用给 node 带来的挑战主要是：由于 JavaScript 单线程的原因，如果有长时间运行的计算（比如大循环），将会导致 CPU 时间片不能释放，使得后续 I/O 无法发起；

解决方案：分解大型运算任务为多个小任务，使得运算能够适时释放，不阻塞 I/O 调用的发起；

2、只支持单核 CPU，不能充分利用 CPU

3、可靠性低，一旦代码某个环节崩溃，整个系统都崩溃

原因：单进程，单线程

解决方案

- 3.1) Nnigx 反向代理，负载均衡，开多个进程，绑定多个端口
- 3.2) 开多个进程监听同一个端口，使用 cluster 模块
- 4、开源组件库质量参差不齐，更新快，向下不兼容
- 5、Debug 不方便，错误没有 stack trace

4、commonJS 中的 require/exports 和 ES6 中 import/export 的区别是什么（必会）

commonJS 中的 require/exports

commonJS 模块的重要特性是加载时执行，及脚本代码在 require 的时候，就会全部执行。一旦出现某个模块被“循环加载”就只输出已经执行的部分，还没有执行的部分是不输出的

ES6 中 import/export

ES6 模块是动态引用，如果使用 import 从一个模块加载变量，那些变量不会缓存，而是成为一个指向被加载模块的引用，import/export 最终都是编译为 require/exports 来执行的

5、简述同步和异步的区别，如何避免回调地狱，Node 的异步问题是如何解决的（必会）

同步

方法调用一旦开始，调用者必须等到方法调用返回后，才能继续后续的行为

异步

方法调用一旦开始，方法调用就会立即返回，调用者就可以继续后续的操作。而异步方法通常会在另外一个线程中，整个过程，不会阻碍调用者的工作

避免回调地狱

- 1、Promise
- 2、async/await
- 3、generator
- 4、事件发布/监听模式

Node 的异步问题是如何解决

模块化：将回调函数转换为独立的函数

使用流程控制库，例如 aync

使用 Promise

使用 aync/await(参考 aync/await 替代 Promise 的 6 个理由)

6、[dependencies 和 devDependencies 两者区别](#)（必会）

区别

在 npm 生成的 package.json 文件中，有 devDependencies 和 dependencies 两个环境

```
"devDependencies": {
  "css-loader": "^0.28.8",
  "extract-text-webpack-plugin": "^3.0.2",
  "html-webpack-plugin": "^2.30.1",
  "less": "^2.7.3",
  "less-loader": "^4.0.5",
  "node-notifier": "^5.1.2",
  "optimize-css-assets-webpack-plugin": "^3.2.0",
  "ora": "^1.3.0",
  "postcss-loader": "^2.0.10",
  "rimraf": "^2.6.2",
  "url-loader": "^0.6.2",
  "vue-loader": "^13.7.0",
  "vue-style-loader": "^3.0.3",
},
"dependencies": {
  "axios": "^0.17.1",
  "babel-polyfill": "^6.26.0",
  "qs": "^6.5.1",
  "vue": "^2.5.13",
  "vue-awesome-swiper": "^3.1.2",
  "vue-router": "^3.0.1",
  "vuex": "^3.0.1"
```

devDependencies 用于开发环境（本地）

dependencies 用于生产环境（发布）

-save //会把依赖包名称添加到 package.json 文件 dependencies 下

-save-dev //则添加到 package.json 文件 devDependencies 下

devDependencies 下列出的模块，是我们开发时用的依赖项，像一些进行单元测试之类的包 //webpack, gulp 等打包工具，这些都是我们开发阶段使用的，代码提交线上时，不需要这些工具，所以我们将它放入 devDependencies 即可

dependencies 下的模块，则是我们生产环境中需要的依赖，即正常运行该包时所需要的依赖项

//像 jQuery 库文件以及 vue 插件 vue-awesome-swiper, vue-router 路由等是在打包之后继续用到的，所以放到 dependencies 里面

"dependencies": 应用程序在生产中需要这些包,即项目上线后所依赖的环境。

"devDependencies": 这些包仅用于开发和测试,即开发中所需要的产品中就不需要。

7、什么是前后端分离的项目?什么是 JS 渲染的项目，前端渲染和后端渲染的区别（高薪常问）

前后端分离的项目

前端 HTML 页面通过 Ajax 调用后端的 RESTFUL API 接口并使用 JSON 数据进行交互

JS 渲染的项目

黑马程序员-武汉前端学科出品 www.itheima.com

第 133 页 共 227 页

通过 Ajax 请求数据以后，通过 JS 代码动态创建 html 的标签和数据等（一般右键查看网页源代码是看不到渲染后的 HTML 标签的）

前端渲染

指的是后端返回 JSON 数据，前端利用预先写的 html 模板，循环读取 JSON 数据，拼接字符串（ES6 的模板字符串特性大大减少了拼接字符串的成本），并插入页面。

好处：网络传输数据量小。不占用服务端运算资源（解析模板），模板在前端（很有可能仅部分在前端），改结构变交互都前端自己来了，改完自己调就行。

坏处：前端耗时较多，对前端工作人员水平要求相对较高。前端代码较多，因为部分以前在后台处理的交互逻辑交给了前端处理。占用少部分客户端运算资源用于解析模板。

后端渲染：

前端请求，后端用后台模板引擎直接生成 html，前端接收到数据之后，直接插入页面。

好处：前端耗时少，即减少了首屏时间，模板统一在后端。前端（相对）省事，不占用客户端运算资源（解析模板）

坏处：占用服务器资源。

前端渲染与后端渲染对比

1、后端渲染

页面呈现速度：快，受限于用户的带宽

流量消耗：少一点点（可以省去前端框架部分的代码）

可维护性：差（前后端东西放一起，掐架多年，早就在闹分手啦）

seo 友好度：好

编码效率：低（这个跟不同的团队不同，可能不对）

2、前端渲染

页面呈现速度：主要受限于带宽和客户端机器的好坏，优化的好，可以逐步动态展开内容，感觉上会更快一点

流量消耗：多一点点（一个前端框架大概 50KB）当然，有的用后端渲染的项目前端部分也有在用框架

可维护性：好，前后端分离，各施其职，代码一目了然

SEO 友好度：差，大量使用 Ajax，多数浏览器不能抓取 Ajax 数据

编码效率：高，前后端各自只做自己擅长的东西，后端最后只输出接口，不用管页面呈现，只要前后端人员能力不错，效率不会低

8、mysql 和 mongoDB 有什么区别（高薪常问）

MySQL

1、关系型数据库

2、在不同的引擎上有不同的存储方式

3、查询语句是使用传统的 sql 语句，拥有较为成熟的体系，成熟度很高

4、开源数据库的份额在不断增加，mysql 的份额在持续增长

5、缺点就是在海量数据处理的时候效率会显著变慢

MongoDB

非关系型数据库(Nosql)，属于文档型数据库。先解释一下文档的数据库，即可以存放 xml、json、bson（即 Binary-JSON）类型系那个的数据。这些数据具备自述性（self-describing），呈现分层的树状数据结构。数据结构由键值(key=>value)对组成

MongoDB 是由 C++语言编写的，主要是在为 WEB 应用提供可扩展的高性能数据存储解决方案

存储方式：虚拟内存+持久化

查询语句：是独特的 Mongoddb 的查询方式

适合场景：事件的记录，内容管理或者博客平台等等

架构特点：可以通过副本集，以及分片来实现高可用

数据处理：数据是存储在硬盘上的，只不过需要经常读取的数据会被加载到内存中，将数据存储在物理内存中，从而达到高速读写

成熟度与广泛度：新兴数据库，成熟度较低，Nosql 数据库中最为接近关系型数据库，比较

完善的 DB 之一，适用人群不断在增长

MongoDB 的优势

- 1、快速！在适量级的内存的 Mongodb 的性能是非常迅速的，它将热数据存储在物理内存中，使得热数据的读写变得十分快
- 2、高扩展。
- 3、自身的 Failover 机制。
- 4、json 的存储格式。
- 5、内置 GridFS，支持大容量的存储。
- 6、内置 Sharding，分片简单。
- 7、海量数据下，性能优越。
- 8、支持自动故障恢复（复制集）。

MongoDB 的缺陷

- 1、不支持事务操作
- 2、占用空间过大。
- 3、MongoDB 没有如 MySQL 那样成熟的维护工具。
- 4、无法进行关联表查询，不适用于关系多的数据。
- 5、复杂聚合操作通过 mapreduce 创建，速度慢
- 6、模式自由，自由灵活的文件存储格式带来的数据错误
- 7、MongoDB 没有如 MySQL 那样成熟的维护工具，这对于开发和 IT 运营都是个值得注意的地方

9、什么是中间件（高薪常问）

中间件是什么

其实就是一个个的函数，当调用 next 时，才会执行下一个中间件函数 express 是一个自身功能极简，完全是路由和中间件

构成一个 web 开发框架：从本质上来说，一个 express 应用就是在调用各种中间件函数。封装了一些或许复杂但肯定是通用的功能，

非内置的中间件需要通过安装后，require 到文件就可以运行

10、为什么要进行模块化（高薪常问）

目前前端的开发形势就是模块化和组件化；从软件工程学分析来说就是有了更好的可维护性、可复用性等好处；但是前端的主要语言 js 在 ES6

之前却没有模块化功能，之前有使用 require.js 和 sea.js 但是推出 ES6 的模块化之后，ES6 的模块化使用形式基本统一

11、谈谈你对 AMD 和 CMD 的理解（高薪常问）

AMD

AMD 推崇依赖前置，在定义模块的时候就要声明其依赖的模块

同样都是异步加载模块，AMD 在加载模块完成后就会执行该模块，所有模块都加载执行完后会进入 require 的回调函数，执行主逻辑，这样的效果就是依赖模块的执行顺序和书写顺序不一定一致，看网络速度，哪个先下载下来，哪个先执行，但是主逻辑一定在所有依赖，加载完成后才执行

CMD

CMD 推崇就近依赖，只有在用到某个模块的时候再去 require

CMD 加载完某个依赖模块后并不执行，只是下载而已，在所有依赖模块加载完成后进入主

逻辑，遇到 `require` 语句的时候才执行对应的模块，这样模块的执行顺序和书写顺序是完全一致的。

特点对比

AMD 用户体验好，因为没有延迟，依赖模块提前执行了；CMD 性能好，因为只有用户需要的时候才执行。

12、Node 怎么跟 MongoDB 建立连接（高薪常问）

- 1、引入 `mongoose`
- 2、使用 `mongoose.connect()` 方法连接到 MongoDB 数据库
- 3、监听连接是否成功
- 3、然后通过 Node，书写接口，对数据库进行增删改查

13、请介绍一下 `require` 的模块加载机制（高薪常问）

这道题基本上就可以了解到面试者对 node 模块机制的了解程度

- 1、先计算模块路径
- 2、如果模块在缓存里面，取出缓存
- 3、加载模块
- 4、输出模块的 `exports` 属性即可

14、express 中如何获取路由的参数（高薪常问）

`/users/:name` 使用 `req.params.name` 来获取；
`req.body.username` 则是获得表单传入参数 `username`
`express` 路由支持常用通配符 `?`, `+`, `*`, and `()`

Webpack

1、什么是 Webpack（必会）

1、基本定义

Webpack 是一个打包模块化 javascript 的工具，在 Webpack 里一切文件皆模块，通过 loader 转换文件，通过 plugin 注入钩子，最后输出由多个模块组合成的文件，Webpack 专注构建模块化项目，Webpack 可以看做是模块打包机：它做的事情是，分析你的项目结构，找到 JavaScript 模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript 等），并将其打包为合适的格式以供浏览器使用

2、Webpack 的优点是什么？（必会）

- 1、专注于处理模块化的项目，能做到开箱即用，一步到位
- 2、通过 plugin 扩展，完整好用又不失灵活
- 3、使用场景不局限于 web 开发
- 4、社区庞大活跃，经常引入紧跟时代发展的新特性，能为大多数场景找到已有的开源扩展

黑马程序员-武汉前端学科出品 www.itheima.com

5、提供了更好的开发体验

3、Webpack 的构建流程是什么?从读取配置到输出文件这个过程尽量说全（必会）

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

4、说一下 Webpack 的热更新原理(必会)

1、基本定义

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。

2、核心定义

2,1) HMR 的核心就是客户端从服务端拉去更新后的文件，准确的说是 chunk diff (chunk 需要更新的部分)，实际上 WDS 与浏览器之间维护了一个 websocket，当本地资源发生变化时，WDS 会向浏览器推送更新，并带上构建时的 hash，让客户端与上一次资源进行对比

2,2) 客户端对比出差异后会向 WDS 发起 Ajax 请求来获取更改内容（文件列表、hash），这样客户端就可以再借助这些信息继续向 WDS 发起 jsonp 请求获取该 chunk 的增量更新

2,3) 后续的部分（拿到增量更新之后如何处理？哪些状态该保留？哪些又需要更新？）由 HotModulePlugin 来完成，提供了相关 API 以供开发者针对自身场景进行处理，像 react-hot-loader 和 vue-loader 都是借助这些 API 实现 HMR

5、Webpack 与 grunt、gulp 的不同？（必会）

1、三者之间的区别

三者都是前端构建工具，grunt 和 gulp 在早期比较流行，现在 Webpack 相对来说比较主流，不过一些轻量化的任务还是会用 gulp 来处理，比如单独打包 CSS 文件等

1,1) grunt 和 gulp 是基于任务和流（Task、Stream）的。类似 jQuery，找到一个（或一类）文件，对其做一系列链式操作，更新流上的数据，整条链式操作构成了一个任务，多个任务就构成了整个 web 的构建流程。

1,2) Webpack 是基于入口的。Webpack 会自动地递归解析入口所需要加载的所有资源文件，然后用不同的 Loader 来处理不同的文件，用 Plugin 来扩展 Webpack 功能。

2、构建思路的区别

2,1) gulp 和 grunt 需要开发者将整个前端构建过程拆分成多个`Task`，并合理控制所有`Task`的调用关系

2,2) Webpack 需要开发者找到入口，并需要清楚对于不同的资源应该使用什么 Loader 做何种解析和加工

3、从知识背景区别

3,1) gulp 更像后端开发者的思路，需要对于整个流程了如指掌

3,2) Webpack 更倾向于前端开发者的思路

6、有哪些常见的 Loader？他们是解决什么问题的？（必会）

1、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

2、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

3、source-map-loader：加载额外的 Source Map 文件，以方便断点调试

4、image-loader：加载并且压缩图片文件

5、babel-loader：把 ES6 转换成 ES5

6、css-loader：加载 CSS，支持模块化、压缩、文件导入等特性

7、style-loader：把 CSS 代码注入到 JavaScript 中，通过 DOM 操作去加载 CSS

8、eslint-loader：通过 ESLint 检查 JavaScript 代码

7、Loader 和 Plugin 的不同？（必会）

1、不同的作用

1,1) 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。

1,2) 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

2、不同的用法

2,1) 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)

2,2) 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

8、如何利用 Webpack 来优化前端性能（高薪常问）

1、压缩代码。uglifyJsPlugin 压缩 js 代码，mini-css-extract-plugin 压缩 css 代码

2、利用 CDN 加速，将引用的静态资源修改为 CDN 上对应的路径，可以利用 Webpack 对于 output 参数和 loader 的 publicPath 参数来修改资源路径

3、删除死代码(tree shaking)，css 需要使用 Purify-CSS

4、提取公共代码。Webpack4 移除了 CommonsChunkPlugin（提取公共代码），用 optimization.splitChunks 和 optimization.runtimeChunk 来代替

9、是否写过 Loader 和 Plugin? 描述一下编写 loader 或 plugin 的思路? (高薪常问)

1、基本定义

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。

1、编写思路

2.1) 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

2.2) 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

2、编写注意事项

3.1) Loader 支持链式调用,所以开发上需要严格遵循“单一职责”,每个 Loader 只负责自己需要负责的事情

3.2) Loader 运行在 node.js 中,我们可以调用任意 node.js 自带的 API 或者安装第三方模块进行调用

3.3) Webpack 传给 Loader 的原内容都是 UTF-8 格式编码的字符串,当某些场景下 Loader 处理二进制文件时,需要通过 `exports.raw = true` 告诉 Webpack 该 Loader 是否需要二进制数据

3.4) 尽可能的异步化 Loader,如果计算量很小,同步也可以

3.5) Loader 是无状态的,我们不应该在 Loader 中保留状态

3.6) 使用 loader-utils 和 schema-utils 为我们提供的实用工具

3.7) 加载本地 Loader 方法

10、使用 Webpack 开发时,你用过哪些可以提高效率的插件? (高薪常问)

1、Webpack-dashboard: 可以更友好的展示相关打包信息。

2、Webpack-merge: 提取公共配置,减少重复配置代码

3、speed-measure-Webpack-plugin: 简称 SMP,分析出 Webpack 打包过程中 Loader 和 Plugin 的耗时,有助于找到构建过程中的性能瓶颈

4、size-plugin: 监控资源体积变化,尽早发现问题

5、HotModuleReplacementPlugin: 模块热替换

11、什么是长缓存? 在 Webpack 中如何做到长缓存优化? (高薪常问)

1、什么是长缓存

浏览器在用户访问页面的时候,为了加快加载速度,会对用户访问的静态资源进行存储,但是每一次代码升级或者更新,都需要浏览器去下载新的代码,最方便和最简单的更新方式

就是引入新的文件名称

2、具体实现

在 Webpack 中，可以在 output 给出输出的文件制定 chunkhash，并且分离经常更新的代码和框架代码，通过 NameModulesPlugin 或者 HashedModulesPlugin 使再次打包文件名不变

12、如何提高 Webpack 的构建速度？（高薪常问）

在多入口情况下，使用 CommonsChunkPlugin 来提取公共代码

1、通过 externals 配置来提取常用库

2、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。

3、使用 Happypack 实现多线程加速编译

4、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

5、使用 Tree-shaking 和 Scope Hoisting 来剔除多余代码

13、怎么实现 Webpack 的按需加载？什么是神奇注释？(高薪常问)

1、按需加载

在 Webpack 中，import 不仅仅是 ES6 module 的模块导入方式，还是一个类似 require 的函数，我们可以通过 import('module') 的方式引入一个模块，import() 返回的是一个 Promise 对象；使用 import() 方式就可以实现 Webpack 的按需加载

2、神奇注释

在 import() 里可以添加一些注释，如定义该 chunk 的名称，要过滤的文件，指定引入的文件等等，这类带有特殊功能的注释被称为神器注释

Vue 面试题

1、Vue 的最大的优势是什么？（必会）

Vue 作为一款轻量级框架、简单易学、数据绑定、组件化、数据和结构的分离、虚拟 DOM、运行速度快，并且作者是中国人尤雨溪，对应的 API 文档对国内开发者优化，作为前端开发人员的首选入门框架，Vue 有很多优势：

Vue.js 可以进行组件化开发，使代码编写量大大减少，读者更加易于理解。

Vue.js 最突出的优势在于可以对数据进行双向绑定。

相比传统的页面通过超链接实现页面的切换和跳转，Vue 使用路由不会刷新页面。

Vue 是单页面应用，使页面局部刷新，不用每次跳转页面都要请求所有数据和 dom，这样大大加快了访问速度和提升用户体验。

而且他的第三方 UI 组件库使用起来节省很多开发时间，从而提升开发效率。

2、Vue 和 jQuery 两者之间的区别是什么？

1、jQuery 介绍：曾经也是现在依然最流行的 web 前端 js 库，可是现在无论是国内还是国外他的使用率正在渐渐被其他的 js 库所代替，随着浏览器厂商对 HTML5 规范统一遵循以及 ECMA6 在浏览器端的实现，jQuery 的使用率将会越来越低

2、Vue 介绍：Vue 是一个兴起的前端 js 库，是一个精简的 MVVM。从技术角度讲，Vue.js 专注于 MVVM 模型的 ViewModel 层。它通过双向数据绑定把 View 层和 Model 层连接了起来，通过对数据的操作就可以完成对页面视图的渲染。当然还有很多其他的 mvmm 框架如 Angular, React 都是大同小异，本质上都是基于 MVVM 的理念。然而 Vue 以他独特的优势简单，快速，组合，紧凑，强大而迅速崛起

3、Vue 和 jQuery 对比 jQuery 是使用选择器（`$(...)`）选取 DOM 对象，对其进行赋值、取值、事件绑定等操作，其实和原生的 HTML 的区别只在于可以更方便的选取和操作 DOM 对象，而数据和界面是在一起的。比如需要获取 label 标签的内容：`$(“label”).val()`；它还是依赖 DOM 元素的值。Vue 则是通过 Vue 对象将数据和 View 完全分离开来了。对数据进行操作不再需要引用相应的 DOM 对象，可以说数据和 View 是分离的，他们通过 Vue 对象这个 vm 实现相互的绑定。这就是传说中的 MVVM。

3、mvvm 和 mvc 区别是什么？哪些场景适合？（必会）

1) MVVM 基本定义

MVVM 即 Model-View-ViewModel 的简写。即模型-视图-视图模型。模型（Model）指的是后端传递的数据。视图（View）指的是所看到的页面。视图模型（ViewModel）是 mvvm 模式的核心，它是连接 view 和 model 的桥梁。它有两个方向：一是将模型（Model）转化成视图（View），即将后端传递的数据转化成所看到的页面。实现的方式是：数据绑定。二是将视图（View）转化成模型（Model），即将所看到的页面转化成后端的数据。实现的方式是：DOM 事件监听。这两个方向都实现的，我们称之为数据的双向绑定。

2) MVC 基本定义

MVC 是 Model-View- Controller 的简写。即模型-视图-控制器。M 和 V 指的意思和 MVVM 中的 M 和 V 意思一样。C 即 Controller 指的是页面业务逻辑。使用 MVC 的目的就是将 M 和 V 的代码分离。MVC 是单向通信。也就是 View 跟 Model，必须通过 Controller 来承上启下。MVC 和 MVVM 的区别并不是 VM 完全取代了 C，只是在 MVC 的基础上增加了一层 VM，只不过是弱化了 C 的概念，ViewModel 存在目的在于抽离 Controller 中展示的业务逻辑，而不是替代 Controller，其它视图操作业务等还是应该放在 Controller 中实现。也就是说 MVVM 实现的是业务逻辑组件的重用，使开发更高效，结构更清晰，增加代码的复用性。

3) 使用场景

MVC 和 MVVM 其实区别并不大。都是一种设计思想。主要就是 MVC 中

Controller 演变成 MVVM 中的 viewModel, 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。

区别: Vue 数据驱动, 通过数据来显示视图层而不是节点操作。 场景: 数据操作比较多的场景, 需要大量操作 DOM 元素时, 采用 MVVM 的开发方式, 会更加便捷, 让开发者更多的精力放在数据的变化上, 解放繁琐的操作 DOM 元素。

4、Vue 数据双向绑定的原理是什么? (必会)

Vue.js 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。

第一步: 需要 `observe` 的数据对象进行递归遍历, 包括子属性对象的属性, 都加上 `setter` 和 `getter`
这样的话, 给这个对象的某个值赋值, 就会触发 `setter`, 那么就能监听到了数据变化

第二步: `compile` 解析模板指令, 将模板中的变量替换成数据, 然后初始化渲染页面视图, 并将每个指令对应的节点绑定更新函数, 添加监听数据的订阅者, 一旦数据有变动, 收到通知, 更新视图

第三步: `Watcher` 订阅者是 `Observer` 和 `Compile` 之间通信的桥梁, 主要做的事情是:

- 1、在自身实例化时往属性订阅器(`dep`)里面添加自己
- 2、自身必须有一个 `update()` 方法
- 3、待属性变动 `dep.notice()` 通知时, 能调用自身的 `update()` 方法, 并触发 `Compile` 中绑定的回调, 则功成身退。

第四步: MVVM 作为数据绑定的入口, 整合 `Observer`、`Compile` 和 `Watcher` 三者, 通过 `Observer` 来监听自己的 `model` 数据变化, 通过 `Compile` 来解析编译模板指令, 最终利用 `Watcher` 搭起 `Observer` 和 `Compile` 之间的通信桥梁, 达到数据变化 -> 视图更新; 视图交互变化(`input`) -> 数据 `model` 变更的双向绑定效果。

5、Object.defineProperty 和 Proxy 的区别 (必会)

1) Proxy 的优势如下:

Proxy 可以直接监听对象而非属性;

Proxy 可以直接监听数组的变化;

Proxy 有多达 13 种拦截方法, 不限于 `apply`、`ownKeys`、`deleteProperty`、`has` 等等是 `Object.defineProperty` 不具备的;

Proxy 返回的是一个新对象, 我们可以只操作新的对象达到目的, 而 `Object.defineProperty` 只能遍历对象属性直接修改;

Proxy 作为新标准将受到浏览器厂商重点持续的性能优化, 也就是传说中的新标准的性能红利;

2) Object.defineProperty 的优势如下:

兼容性好, 支持 IE9, 而 Proxy 的存在浏览器兼容性问题, 而且无法用

6、Vue 生命周期总共分为几个阶段？（必会）

Vue 实例从创建到销毁的过程，就是生命周期。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、卸载等一系列过程，我们称这是 Vue 的生命周期。

1) beforeCreate

在实例初始化之后，数据观测（data observer）和 event/watcher 事件配置之前被调用。

2) created

在实例创建完成后被立即调用。在这一步，实例已完成以下的配置：数据观测（data observer），属性和方法的运算，watch/event 事件回调。然而，挂载阶段还没开始，\$el 属性目前不可见。

3) beforeMount

在挂载开始之前被调用：相关的 render 函数首次被调用。

4) mounted

el 被新创建的 vm.\$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.\$el 也在文档内。

5) beforeUpdate

数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。该钩子在服务器端渲染期间不被调用，因为只有初次渲染会在服务端进行。

6) updated

由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

7) activated

keep-alive 组件激活时调用。该钩子在服务器端渲染期间不被调用。

8) deactivated

keep-alive 组件停用时调用。该钩子在服务器端渲染期间不被调用。

9) beforeDestroy

实例销毁之前调用。在这一步，实例仍然完全可用。该钩子在服务器端渲染期间不被调用。

10) destroyed

Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。

11) errorCaptured (2.5.0+ 新增)

当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止该错误继续向上传播。

7、第一次加载页面会触发哪几个钩子函数？（必会）

当页面第一次加载时会触发 `beforeCreate`, `created`, `beforeMount`, `mounted` 这几个钩子函数

8、请说下封装 Vue 组件的过程？（必会）

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

1) **分析需求：**确定业务需求，把页面中可以服用的结构，样式以及功能，单独抽离成一个文件，实现复用

2) **具体步骤：**

1. 使用 `Vue.component` 方法注册组件，子组件需要数据，可以在 `props` 中接受定义，而子组件修改好数据后，想把数据传递给父组件。可以采用 `$emit` 方法向外抛数据

2. 如果需要给组件传入模板，则定义为插槽 `slot`

3. 如果需要 父组件主动调用子组件的方法 可以在 `methods` 选项中开放方法

9、Vue 组件如何进行传值的？（必会）

1) **父组件向子组件传递数据**

父组件内设置要传的数据，在父组件中引用的子组件上绑定一个自定义属性并把数据绑定在自定义属性上，在子组件添加参数 `props` 接收即可

2) **子组件向父组件传递数据**

子组件通过 `Vue` 实例方法 `$emit` 进行触发并且可以携带参数，父组件监听使用 `@(v-on)` 进行监听，然后进行方法处理

3) **非父子组件之间传递数据**

1、引入第三方 `new Vue` 定义为 `eventBus`

2、在组件中 `created` 中订阅方法 `eventBus.$on("自定义事件名", methods 中的方法名)`

3、在另一个兄弟组件中的 `methods` 中写函数，在函数中发布 `eventBus` 订阅的方法 `eventBus.$emit("自定义事件名")`

4、在组件的 `template` 中绑定事件(比如 `click`)

10、组件中写 `name` 选项有什么作用？（必会）

1, 项目使用 `keep-alive` 时，可搭配组件 `name` 进行缓存过滤

2, `DOM` 做递归组件时需要调用自身 `name`

3, `Vue-devtools` 调试工具里显示的组件名称是由 `Vue` 中组件 `name` 决定的

11、Vue 组件 `data` 为什么必须是函数（必会）

1、每个组件都是 `Vue` 的实例。

- 2、组件共享 data 属性，当 data 的值是同一个引用类型的值时，改变其中一个会影响其他
- 3、组件中的 data 写成一个函数，数据以函数返回值形式定义，这样每复用一次组件，就会返回一份新的 data，类似于给每个组件实例创建一个私有的数据空间，让各个组件实例维护各自的数据。而单纯的写成对象形式，就使得所有组件实例共用了一份 data，就会造成一个变了全都会变的结果。

12、讲一下组件的命名规范（必会）

给组件命名有两种方式，一种是使用链式命名 my-component，一种是使用大驼峰命名 MyComponent，
在字符串模板中 `<my-component></my-component>` 和 `<MyComponent></MyComponent>` 都可以使用，
在非字符串模板中最好使用 `<MyComponent></MyComponent>`，因为要遵循 W3C 规范中的自定义组件名（字母全小写且必须包含一个连字符），避免和当前以及未来的 HTML 元素相冲突

13、怎么在组件中监听路由参数的变化？（必会）

有两种方法可以监听路由参数的变化，但是只能用在包含 `<router-view />` 的组件内。

第一种

```
watch: {
  '$route' (to, from) {
    // 在此处监听
  },
},
```

第二种

```
beforeRouteUpdate (to, from, next) {
  //这里监听
},
```

14、怎么捕获 Vue 组件的错误信息？（必会）

errorCaptured 是组件内部钩子，当捕获一个来自子孙组件的错误时被调用，接收 error、vm、info 三个参数，return false 后可以阻止错误继续向上抛出。

errorHandler 为全局钩子，使用 Vue.config.errorHandler 配置，接收参数与 errorCaptured 一致，2.6 后可捕捉 v-on 与 promise 链的错误，可用于统一错误处理与错误兜底。

15、Vue 组件里的定时器要怎么销毁？（必会）

如果页面上有很多定时器，可以在 data 选项中创建一个对象 timer，给

每个定时器取个名字一一映射在对象 timer 中， 在 beforeDestroy 构造函数中

```
for(let k in this.timer){clearInterval(k)};
    如果页面只有单个定时器，可以这么做。
const timer = setInterval(() => {}, 500);
this.$once('hook:beforeDestroy', () => {
  clearInterval(timer);
})
```

16、Vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？（必会）

第一步：在 components 目录新建你的组件文件（indexPage.vue），script 一定要 export default {}

第二步：在需要用的页面（组件）中导入：import indexPage from '@/components/indexPage.vue'

第三步：注入到 Vue 的子组件的 components 属性上面，components: {indexPage}

第四步：在 template 视图 view 中使用，

例如有 indexPage 命名，使用的时候则 index-page

17、Vue 中 slot 的使用方式，以及 slot 作用域插槽的用法（必会）

使用方式：当组件当做标签进行使用的时候，用 slot 可以用来接受组件标签包裹的内容，当给 slot 标签添加 name 属性的时候，可以调换响应的位置

插槽作用域：作用域插槽其实就是带数据的插槽，父组件接收来自子组件的 slot 标签上通过 v-bind 绑定进而传递过来的数据，父组件通过 scope 来进行接受子组件传递过来的数据

18、Vue 该如何实现组件缓存？（必会）

在面向组件化开发中，我们会把整个项目拆分为很多业务组件，然后按照合理的方式组织起来，那么自然会存在组件之前切换的问题，Vue 中有个动态组件的概念，它能够帮助开发者更好的实现组件之间的切换，但是在面对需求频繁的变化，去要切换组件时，动态组件在切换的过程中，组件的实例都是重新创建的，而我们需要保留组件的状态，为了解决这个问题，需要使用到 Vue 中内置组件 <keep-alive>

<keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例，主要用于保留组件状态或避免重新渲染，

简答的说：比如有一个列表和一个详情，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<keep-alive></keep-alive>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

19、跟 keep-alive 有关的生命周期是哪些？（必会）

1) **前言：**在开发 Vue 项目的时候，大部分组件是没必要多次渲染的，所以 Vue 提供了一个内置组件 keep-alive 来缓存组件内部状态，避免重新渲染，在开发 Vue 项目的时候，大部分组件是没必要多次渲染的，所以 Vue 提供了一个内置组件 keep-alive 来缓存组件内部状态，避免重新渲染

2) **生命周期函数：**在被 keep-alive 包含的组件/路由中，会多出两个生命周期的钩子:activated 与 deactivated。

1、**activated 钩子：**在在组件第一次渲染时会被调用，之后在每次缓存组件被激活时调用。

2、**Activated 钩子调用时机：**第一次进入缓存路由/组件，在 mounted 后面，beforeRouteEnter 守卫传给 next 的回调函数之前调用，并且给因为组件被缓存了，再次进入缓存路由、组件时，不会触发这些钩子函数，beforeCreate created beforeMount mounted 都不会触发

1、**deactivated 钩子：**组件被停用（离开路由）时调用。

2、**deactivated 钩子调用时机：**使用 keep-alive 就不会调用 beforeDestroy(组件销毁前钩子)和 destroyed(组件销毁)，因为组件没被销毁，被缓存起来了，这个钩子可以看作 beforeDestroy 的替代，如果你缓存了组件，要在组件销毁的时候做一些事情，可以放在这个钩子里，组件内的离开当前路由钩子 beforeRouteLeave => 路由前置守卫 beforeEach =>全局后置钩子 afterEach => deactivated 离开缓存组件 => activated 进入缓存组件(如果你进入的也是缓存路由)

20、Vue 常用的修饰符都有哪些？（必会）

.prevent: 提交事件不再重载页面; .stop: 阻止单击事件冒泡; .self: 当事件发生在该元素本身而不是子元素的时候会触发; .capture: 事件侦听, 事件发生的时候会调用

21、Vue 常用的指令都有哪些？并且说明其作用（必会）

- 1、v-model 多用于表单元素实现双向数据绑定（同 angular 中的 ng-model）
- 2、v-for 格式: v-for="字段名 in(of) 数组 json" 循环数组或 json(同 angular 中的 ng-repeat), 需要注意从 Vue2 开始取消了 \$index
- 3、v-show 显示内容（同 angular 中的 ng-show）
- 4、v-hide 隐藏内容（同 angular 中的 ng-hide）
- 5、v-if 显示与隐藏（dom 元素的删除添加 同 angular 中的 ng-if 默认值为 false）
- 6、v-else-if 必须和 v-if 连用
- 7、v-else 必须和 v-if 连用 不能单独使用 否则报错 模板编译错误
- 8、v-bind 动态绑定 作用: 及时对页面的数据进行更改
- 9、v-on:click 给标签绑定函数, 可以缩写为 @, 例如绑定一个点击函数 函数必须写在 methods 里面
- 10、v-text 解析文本
- 11、v-html 解析 html 标签
- 12、v-bind:class 三种绑定方法 1、对象型 '{red:isred}' 2、三元型 'isred?"red":"blue"' 3、数组型 ' [{red:"isred"}, {blue:"isblue"}]'

- 13、v-once 进入页面时 只渲染一次 不在进行渲染
- 14、v-cloak 防止闪烁
- 15、v-pre 把标签内部的元素原位输出

22、自定义指令(v-check、v-focus)的方法有哪些?它有哪些钩子函数?还有哪些钩子函数参数?(必会)

全局定义指令: 在 Vue 对象的 directive 方法里面有两个参数, 一个是指令名称, 另外一个函数。组件内定义指令: directives

钩子函数: bind(绑定事件触发)、inserted(节点插入的时候触发)、update(组件内相关更新)

钩子函数参数: el、binding

24、v-show 和 v-if 指令的共同点和不同点?(必会)

1) 相同点:

v-show 和 v-if 都能控制元素的显示和隐藏。

2) 不同点:

实现本质方法不同

v-show 本质就是通过设置 css 中的 display 设置为 none, 控制隐藏

v-if 是动态的向 DOM 树内添加或者删除 DOM 元素

3) 编译的区别

v-show 其实就是在控制 css

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

4) 编译的条件

v-show 都会编译, 初始值为 false, 只是将 display 设为 none, 但它也编译了

v-if 初始值为 false, 就不会编译了

5) 性能比较

v-show 只编译一次, 后面其实就是在控制 css, 而 v-if 不停的销毁和创建, 故 v-show 性能更好一点。

注意点: 因为 v-show 实际是操作 display: "" 或者 none, 当 css 本身有 display: none 时, v-show 无法让显示

总结(适用场景): 如果要频繁切换某节点时, 使用 v-show(无论 true 或者 false 初始都会进行渲染, 此后通过 css 来控制显示隐藏, 因此切换开销比较小, 初始开销较大), 如果不需要频繁切换某节点时, 使用 v-if(因为懒加载, 初始为 false 时, 不会渲染, 但是因为它通过添加和删除 dom 元素来控制显示和隐藏的, 因此初始渲染开销较小, 切换开销比较大)

25、为什么避免 v-if 和 v-for 用在一起(必会)

当 Vue 处理指令时, v-for 比 v-if 具有更高的优先级, 通过 v-if 移动

到容器元素，不会再重复遍历列表中的每个值。取而代之的是，我们只检查它一次，且不会在 `v-if` 为否的时候运算 `v-for`。

26、watch、methods 和 computed 的区别？（必会）

1) 基本说明

1. computed:

计算属性将被混入到 Vue 实例中。所有 getter 和 setter 的 this 上下文自动地绑定为 Vue 实例。

2. methods:

methods 将被混入到 Vue 实例中。可以直接通过 VM 实例访问这些方法，或者在指令表达式中使用。方法中的 this 自动绑定为 Vue 实例。

3. watch:

观察和响应 Vue 实例上的数据变动，一个对象，键是需要观察的表达式，值是对应回调函数。值也可以是方法名，或者包含选项的对象，Vue 实例将会在实例化时调用 `$watch()`，遍历 watch 对象的每一个属性。

2) 三者的加载顺序

1. computed 是在 HTML DOM 加载后马上执行的，如赋值；（属性将被混入到 Vue 实例）

2. methods 则必须要有一定的触发条件才能执行，如点击事件

3. watch 呢？它用于观察 Vue 实例上的数据变动。

3) 默认加载的时候

先 computed 再 watch，不执行 methods；

4) 触发某一事件后

先 computed 再 methods 再到 watch

computed 属性 vs method 方法

computed 计算属性是基于它们的依赖进行缓存的。

5) 总结

计算属性 computed 只有在它的相关依赖发生改变时才会重新求值，当有一个性能开销比较大的计算属性 A，它需要遍历一个极大的数组和做大量的计算，然后我们可能有其他的计算属性依赖于 A，这时候，我们就需要缓存，每次确实需要重新加载，不需要缓存时用 methods

27、怎么在 watch 监听开始之后立即被调用？（必会）

在选项参数中指定 `immediate: true` 将立即以表达式的当前值触发回调。

27、watch 怎么深度监听对象变化？（必会）

```
<input type="text" v-model="cityName"/>
new Vue({
  el: '#root',
  data: {
    cityName: 'shanghai'
  }
})
```

```

    },
    watch: {
      cityName(newName, oldName) { // ...
    }
  })

```

直接写一个监听处理函数，当每次监听到 `cityName` 值发生改变时，执行函数。也可以在所监听的数据后面直接加字符串形式的方法名：

```

watch: {
  cityName: 'nameChange'
}

```

29、computed 中的属性名和 data 中的属性名可以相同吗？

（必会）

不能同名，因为不管是 `computed` 属性名还是 `data` 数据名还是 `props` 数据名都会被挂载在 `vm` 实例上，因此这三个都不能同名。

```

if (key in vm.$data) {
  warn(`The computed property "${key}" is already defined in data.` , vm)
} else if (vm.$options.props && key in vm.$options.props) {
  warn(`The computed property "${key}" is already defined as a prop.` , vm)
}

```

30、什么是 Vue 的计算属性（必会）

在模板中放入太多的逻辑会让模板过重且难以维护，在需要对数据进行复杂处理，且可能多次使用的情况下，尽量采取计算属性的方式，好处：使得数据处理结构清晰；

- 1、依赖于数据，数据更新，处理结果自动更新；
- 2、计算属性内部 `this` 指向 `vm` 实例；
- 3、在 `template` 调用时，直接写计算属性名即可；
- 4、常用的是 `getter` 方法，获取数据，也可以使用 `set` 方法改变数据；
- 5、相较于 `methods`，不管依赖的数据变不变，`methods` 都会重新计算，但是依赖数据不变的时候 `computed` 从缓存中获取，不会重新计算。

31、Vue 中 key 值的作用是什么？（必会）

当 `Vue.js` 用 `v-for` 正在更新已渲染过的元素列表时，它默认用“就地复用”策略。如果数据项的顺序被改变，`Vue` 将不会移动 `DOM` 元素来匹配数据项

的顺序，而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。key 的作用主要是为了高效的更新虚拟 DOM。

32、Vue-loader 是什么？使用它的用途有哪些？（必会）

Vue-loader 会解析文件，提取出每个语言块，如果有必要会通过其他 loader 处理，最后将他们组装成一个 commonjs 模块；module.exports 出一个 Vue.js 组件对象；

1) < template>语言块

- (1) 默认语言：html
- (2) 每个.Vue 文件最多包含一个< template>块
- (3) 内容将被提取为字符串，将编译用作 Vue 组件的 template 选项；

2) < script>

- (1) 默认语言：JS（在监测到 babel-loader 或者 buble-loader 配置时，自动支持 ES2015）
- (2) 每个.Vue 文件最多包含一个< script>块
- (3) 该脚本在类 CommonJS 环境中执行（就像通过 webpack 打包的正常 JS 模块）。所以你可以 require() 其他依赖。在 ES2015 支持下，也可以使用 import 跟 export 语法
- (4) 脚本必须导出 Vue.js 组件对象，也可以导出由 Vue.extend() 创建的扩展对象；但是普通对象是更好的选择；

3) < style>

默认语言：css

- 1、一个.Vue 文件可以包含多个< style>标签
- 2、这个标签可以有 scoped 或者 module 属性来帮助你讲样式封装到当前组件；具有不同封装模式的多个< style>标签可以在同一个组件中混合使用
- 3、默认情况下，可以使用 style-loader 提取内容，并且通过< style>标签动态假如文档的< head>中，也可以配置 webpack 将所有的 styles 提取到单个 CSS 文件中；

4) 自定义块

可以在.Vue 文件中添加额外的自定义块来实现项目的特殊需求；例如< docs>块；Vue-loader 将会使用标签名来查找对应的 webpack loaders 来应用到对应的模块上；webpack 需要在 Vue-loader 的选项 loaders 中指定；Vue-loader 支持使用非默认语言，比如 CSS 预处理器，预编译的 HTML 模板语言，通过设置语言块的 lang 属性：

```
<style lang='sass'>
  /*sass*/
</style>
```

33、Vue 中怎么自定义过滤器（必会）

Vue.js 允许自定义过滤器，可被用于一些常见的文本格式化。过滤器可以在两个地方：双花括号插值和 v-bind 表达式。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符号指示

可以用全局方法 `Vue.filter()` 注册一个自定义过滤器，它接收两个参数：过滤器 ID 和过滤器函数。过滤器函数以值为参数，返回转换后的值。

```
Vue.filter('reverse', function (value) { return value.split('').reverse().join('') })
```

```
<span v-text = "message | reverse"></span>
```

过滤器也同样接受全局注册和局部注册。

34、你是怎么认识 Vuex 的? (必会)

Vuex 可以理解为一种开发模式或框架。比如 PHP 有 thinkphp, java 有 spring 等, 通过状态 (数据源) 集中管理驱动组件的变化 (好比 spring 的 IOC 容器对 bean 进行集中管理)。

- 1、应用级的状态集中放在 store 中;
- 2、改变状态的方式是提交 mutations, 这里必须是同步的;
- 3、异步逻辑应该封装在 action 中。

35、Vuex 的 5 个核心属性是什么? (必会)

分别是 State、Getter、Mutation、Action、Module

1) state

state 为单一状态树，在 state 中需要定义我们所需要管理的数组、对象、字符串等等，只有在这里定义了，在 Vue.js 的组件中才能获取你定义的这个对象的状态。

2) getter

getter 有点类似 Vue.js 的计算属性，当我们需要从 store 的 state 中派生出一些状态，那么我们就需要使用 getter，getter 会接收 state 作为第一个参数，而且 getter 的返回值会根据它的依赖被缓存起来，只有 getter 中的依赖值 (state 中的某个需要派生状态的值) 发生改变的时候才会被重新计算。

3) mutation

更改 store 中 state 状态的唯一方法就是提交 mutation，就很类似事件。每个 mutation 都有一个字符串类型的事件类型和一个回调函数，我们需要改变 state 的值就要在回调函数中改变。我们要执行这个回调函数，那么我们需要执行一个相应的调用方法：store.commit。

4) action

action 可以提交 mutation，在 action 中可以执行 store.commit，而且 action 中可以有任何的异步操作。在页面中如果我们要调用这个 action，则需要执行 store.dispatch

5) module

module 其实只是解决了当 state 中很复杂臃肿的时候，module 可以将 store 分割成模块，每个模块中拥有自己的 state、mutation、action 和 getter

36、Vuex 的出现解决了什么问题？（必会）

主要解决了以下两个问题：

1，多个组件依赖于同一状态时，对于多层嵌套的组件的传参将会非常繁琐，并且对于兄弟组件间的状态传递无能为力。

2，来自不同组件的行为需要变更同一状态。以往采用父子组件直接引用或者通过事件来变更和同步状态的多份拷贝。以上的这些模式非常脆弱，通常会导致无法维护的代码。

37、简述 Vuex 的数据传递流程（必会）

当组件进行数据修改的时候我们需要调用 `dispatch` 来触发 `actions` 里面的方法。`actions` 里面的每个方法中都会有一个 `commit` 方法，当方法执行的时候会通过 `commit` 来触发 `mutations` 里面的方法进行数据的修改。`mutations` 里面的每个函数都会有一个 `state` 参数，这样就可以在 `mutations` 里面进行 `state` 的数据修改，当数据修改完毕后，会传导给页面。页面的数据也会发生改变

38、Vuex 的 Mutation 和 Action 之间的区别是什么？（必会）

1) 流程顺序

“相应视图→修改 State”拆分成两部分，视图触发 Action，Action 再触发 Mutation。

2) 角色定位

基于流程顺序，二者扮演不同的角色。

Mutation：专注于修改 State，理论上是修改 State 的唯一途径。

Action：业务代码、异步请求。

3) 限制

角色不同，二者有不同的限制。

Mutation：必须同步执行。

Action：可以异步，但不能直接操作 State。

39、Vue-Router 是干什么的，原理是什么？（必会）

Vue-Router 是 Vue.js 官方的路由插件，它和 Vue.js 是深度集成的，适合用于构建单页面应用。Vue 的单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来。传统的页面应用，是用一些超链接来实现页面切换和跳转的。在 Vue-Router 单页面应用中，则是路径之间的切换，也就是组件的切换。路由模块的本质就是建立起 url 和页面之间的映射关系。

“更新视图但不重新请求页面”是前端路由原理的核心之一，目前在浏览器环境中这一功能的实现主要有两种方式：

利用 URL 中的 hash (“#”)

利用 History interface 在 HTML5 中新增的方法

40、路由之间是怎么跳转的？有哪些方式？（必会）

- 1、<router-link to="需要跳转到页面的路径">
- 2、this.\$router.push() 跳转到指定的 url，并在 history 中添加记录，点击后退返回到上一个页面
- 3、this.\$router.replace() 跳转到指定的 url，但是 history 中不会添加记录，点击后退到上上个页面
- 4、this.\$router.go(n) 向前或者后跳转 n 个页面，n 可以是正数也可以是负数

41、Vue-Router 怎么配置路由（必会）

在 Vue 中配置路由分为 5 个步骤，分别是：

- 1, 安装

```
npm install --save Vue-Router
```
- 2, 引用

```
import VueRouter from 'Vue-Router'
```
- 3, 配置路由文件

```
var router = new VueRouter({  
  routes:[  
    {  
      path:"/hello",  
      component:HelloWorld  
    },  
    {  
      path:"/wen",  
      component:HelloWen  
    }  
  ],  
  new Vue({  
    el: '#app',  
    components: { App },  
    router,  
    template: '<App/>'  
  })  
})
```
- 4, 视图加载的位置
默认 App.Vue 文件中加<router-view></router-view>
- 5, 跳转导航
<router-link to="/hello">helloworld</router-link>（渲染出来的是 a 标签）

42、Vue-Router 有哪几种路由守卫？（必会）

路由守卫为：

全局守卫：beforeEach

后置守卫：afterEach

全局解析守卫: beforeResolve

路由独享守卫: beforeEnter

43、Vue-Router 的钩子函数都有哪些? (必会)

关于 Vue-Router 中的钩子函数主要分为 3 类

1) 全局钩子函数要包含 beforeEach

beforeEach 函数有三个参数, 分别是:

to:router 即将进入的路由对象

from:当前导航即将离开的路由

next:function, 进行管道中的一个钩子, 如果执行完了, 则导航的状态就是 confirmed (确认的) 否则为 false, 终止导航。

2) 单独路由独享组件

beforeEnter,

3) 组件内钩子

beforeRouterEnter,

beforeRouterUpdate,

beforeRouterLeave

44、路由传值的方式有哪几种(必会)

Vue-Router 传参可以分为两大类, 分别是编程式的导航 router.push 和声明式的导航

1) router.push

字符串: 直接传递路由地址, 但是不能传递参数

this.\$router.push("home")

对象:

命名路由

这种方式传递参数, 目标页面刷新会报错

this.\$router.push({name:"

news", params:{userId:123}})

查询参数

和 name 配对的式 params, 和 path 配对的是

query

this.\$router.push({path:"

/news', query:{uersId:123}})

接收参数

this.\$route.query

2) 声明式导航

字符串 <router-link to:"news"></router-link>

命名路由

<router-

link :to="{name:'news',params:{userid:1111}}"></route-link>

查询参数

<router-

link :to="{path:'/news',query:{userId:1111}}"></router-link>

45、怎么定义 Vue-Router 的动态路由?怎么获取传过来的动态参数?

我们经常需要把某种模式匹配到的所有路由，全都映射到同个组件。例如，我们有一个 User 组件，对于所有 ID 各不相同的用户，都要使用这个组件来渲染。那么，我们可以在 Vue-Router 的路由路径中使用“动态路径参数”(dynamic segment)来达到这个效果。

动态路径参数，使用“冒号”开头，一个路径参数，使用冒号标记，当匹配到一个路由时，参数会被设置到 `this.$router.params` 中，并且可以在每个组件中使用。

现在我们知道了可以通过动态路由传参，在路由中设置了，多段路径参数后，对应的值分别都会设置到 `$router.query` 和 `$router.params` 中

46、query 和 params 之间的区别是什么？（必会）

- 1、query 要用 path 来引入，params 要用 name 来引入
- 2、接收参数时，分别是 `this.$route.query.name` 和 `this.$route.params.name`（注意：是 `$route` 而不是 `$router`）

47、\$route 和 \$router 的区别是什么？（必会）

`$route` 是“路由信息对象”，包括 `path`, `params`, `hash`, `query`, `fullPath`, `matched`, `name` 等路由信息参数。

`$router` 为 `VueRouter` 的实例，相当于一个全局的路由器对象，里面含有很多属性和子对象，例如 `history` 对象，经常用的跳转链接就可以用 `this.router.push` 会往 `history` 栈中添加一个新的记录。返回上一个 `history` 也是使用 `$router.go` 方法

48、active-class 属于哪个组件中的属性？该如何使用？

首先 `active-class` 是 `Vue-Router` 模块中 `router-link` 组件中的属性，主要作用是用来实现选中样式的切换，在 `Vue-Router` 中要使用 `active-class` 有两种方式：

1) 在 router-link 中写入 active-class

active-class 选择样式时根据路由中的路径 (to= "/home") 去匹配, 然后显示

```
<router-link to="/home" class="menu-home" active-class="active">首页</router-link>
```

2) 直接在路由 js 文件中配置 linkActiveClass

```
export default new Router({
  linkActiveClass: 'active',
})
```

```
<div class="menu-btn">
  <router-link to="/" class="menu-home" active-class="active">
    首页
  </router-link>
</div>
<div class="menu-btn">
  <router-link to="/my" class="menu-my" active-class="active">
    我的
  </router-link>
</div>
```

3) 引起的问题

因为 to="/" 引起的, active-class 选择样式时根据路由中的路径去匹配, 然后显示,

例如在 my 页面中, 路由为 localhost:8081/#/my, 那么 to="/" 和 to="/my" 都可以匹配到, 所有都会激活选中样式

4) 解决方法

1、在 router-link 中写入 exact

```
<router-link to="/" class="menu-home" active-class="active" exact>首页</router-link>
```

2、在路由中加入重定向

```
<router-link to="/" class="menu-home" active-class="active" exact>首页</router-link>
{
  path: '/',
  redirect: '/home'
}
```

49、Vue 的路由实现模式: hash 模式和 history 模式(必会)

1) **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。

2) **history 模式**: history 采用 HTML5 的新特性; 且提供了两个新方法

黑马程序员-武汉前端学科出品 www.itheima.com

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

50、请说出路由配置项常用的属性及作用 (必会)

路由配置参数:

- `path`: 跳转路径
- `component`: 路径相对于的组件
- `name`: 命名路由
- `children`: 子路由的配置参数 (路由嵌套)
- `props`: 路由解耦
- `redirect`: 重定向路由

51、程式导航使用的方法以及常用的方法 (必会)

- 路由跳转: `this.$router.push()`
- 路由替换: `this.$router.replace()`
- 后退: `this.$router.back()`
- 前进: `this.$router.forward()`

52、Vue 怎么实现跨域 (必会)

1) 什么是跨域

跨域指浏览器不允许当前页面的所在的源去请求另一个源的数据。源指协议, 端口, 域名。只要这个 3 个中有一个不同就是跨域

2) 使用 Vue-cli 脚手架搭建项目时 proxyTable 解决跨域问题

打开 `config/index.js`, 在 `proxyTable` 中添写如下代码:

```
proxyTable: {  
  '/api': { //使用"/api"来代替"http://f.apipus.cn"  
    target: 'http://f.apipus.cn', //源地址  
    changeOrigin: true, //改变源  
    pathRewrite: {  
      '^/api': 'http://f.apipus.cn' //路径重写  
    }  
  }  
}
```

3) 使用 CORS (跨域资源共享)

1、前端设置:

前端 Vue 设置 axios 允许跨域携带 cookie (默认是不带 cookie)
`axios.defaults.withCredentials = true;`

2、后端设置:

- 1、跨域请求后的响应头中需要设置:
- 2、`Access-Control-Allow-Origin` 为发起请求的主机地址。
- 3、`Access-Control-Allow-Credentials`, 当它被设置为 `true` 时, 允许跨域带 cookie, 但此时 `Access-Control-Allow-Origin` 不能为通配符*。
- 4、`Access-Control-Allow-Headers`, 设置跨域请求允许的请求头。
- 5、`Access-Control-Allow-Methods`, 设置跨域请求允许的请求方式。

53、Vue 中动画如何实现 (必会)

哪个元素需要动画就给那个元素加 transition 标签
进入时 class 的类型分为以下几种 <name>-enter <name>-enter-active <name>-enter-to
离开时 class 的类型分为以下几种
 <name>-leave <name>-leave-active <name>-leave-to

如果需要一组元素发生动画需要用标签<transition-group><transition-group>

54、你对 Vue.js 的 template 编译的理解？（必会）

简而言之，就是先转化成 AST 树，再得到的 render 函数返回 VNode（Vue 的虚拟 DOM 节点）

首先，通过 compile 编译器把 template 编译成 AST 语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile 是 createCompiler 的返回值，createCompiler 是用以创建编译器的。另外 compile 还负责合并 option。

然后，AST 会经过 generate（将 AST 语法树转化成 render function 字符串的过程）得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有（标签名、子节点、文本等等）

55、Vue 渲染模板时怎么保留模板中的 HTML 注释呢？（必会）

在组件中将 comments 选项设置为 true

<template comments> ... <template>

56、Vue2.0 兼容 IE 哪个版本以上吗？（必会）

不支持 ie8 及以下，部分兼容 ie9，完全兼容 10 以上，因为 Vue 的响应式原理是基于 es5 的 Object.defineProperty(), 而这个方法不支持 ie8 及以下。

57、Vue 如何去除 URL 中的#（必会）

Vue-Router 默认使用 hash 模式，所以在路由加载的时候，项目中的 URL 会自带 “#”。如果不想使用 “#”，可以使用 Vue-Router 的另一种模式 history: new Router ({ mode : 'history', routes: []})

需要注意的是，当我们启用 history 模式的时候，由于我们的项目是一个单页面应用，所以在路由跳转的时候，就会出现访问不到静态资源而出现“404”的情况，这时候就需要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。

58、说一下你在 Vue 中踩过的坑（必会）

1、第一个是给对象添加属性的时候，直接通过给 data 里面的对象添加属性然后赋值，新添加的属性不是响应式的

【解决办法】通过 `Vue.set(对象, 属性, 值)` 这种方式就可以达到，对象新添加的属性是响应式的

1、在 created 操作 dom 的时候，是报错的，获取不到 dom，这个时候实例 Vue 实例没有挂载

【解决办法】通过：`Vue.nextTick(回调函数进行获取)`

59、在 Vue 中使用插件的步骤（必会）

采用 ES6 的 `import ... from ...` 语法或 CommonJS 的 `require()` 方法引入插件

使用全局方法 `Vue.use( plugin )` 使用插件，可以传入一个选项对象 `Vue.use(MyPlugin, { someOption: true })`

60、Vue 项目优化的解决方案都有哪些？（必会）

- 1、使用 `mini-css-extract-plugin` 插件抽离 css
- 2、配置 `optimization` 把公共的 js 代码抽离出来
- 3、通过 `webpack` 处理文件压缩
- 4、不打包框架、库文件，通过 `cdn` 的方式引入
- 5、小图片使用 `base64`
- 6、配置项目文件懒加载
- 7、UI 库配置按需加载
- 8、开启 `Gzip` 压缩

61、使用 Vue 的时候加载造成页面卡顿，该如何解决？（必会）

`Vue-Router` 解决首次加载缓慢的问题。懒加载简单来说就是按需加载。

1、像 Vue 这种单页面应用，如果没有应用懒加载，运用 `webpack` 打包后的文件将会异常的大，造成进入首页时，需要加载的内容过多，时间过长，会出现长时间的白屏，即使做了 `loading` 也是不利于用户体验，而运用懒加载 则可以将页面进行划分，需要的时候加载页面，可以有效的分担首页所承担的加载压力，减少首页加载用时

62、请说出 Vue-cli 项目中 src 目录每个文件夹和文件的用法？（必会）

- 1、`assets` 文件夹是放静态资源
- 2、`components` 是放组件
- 3、`router` 是定义路由相关的配置

- 4、view 视图
- 5、app.Vue 是一个应用主组件
- 6、main.js 是入口文件

63、你知道 style 上加 scoped 属性的原理吗？（必会）

1) 什么是 scoped

在 Vue 组件中，为了使样式私有化（模块化），不对全局造成污染，可以在 style 标签上添加 scoped 属性以表示它的只属于当下的模块，局部有效。如果一个项目中的所有 Vue 组件 style 标签全部加上了 scoped，相当于实现了样式的私有化。如果引用了第三方组件，需要在当前组件中局部修改第三方组件的样式，而又不想去除 scoped 属性造成组件之间的样式污染。此时只能通过穿透 scoped 的方式来解决，选择器

2) scoped 的实现原理：

Vue 中的 scoped 属性的效果主要通过 PostCSS 转译实现，如下是转译前的 Vue 代码：

```
<template><div>Vue.js scoped</div></template>
```

```
<style scoped>.scoped {font-size:14px;}</style>
```

浏览器渲染后的代码：

```
<div data-v-fed36922>Vue.js  
scoped</div>.scoped[data-v-fed36922] {font-size:14px;}
```

即：PostCSS 给所有 dom 添加了一个唯一不重复的动态属性，然后，给 CSS 选择器额外添加一个对应的属性选择器来选择该组件中 dom，这种做法使得样式私有化

64、说说你对 SPA 单页面的理解，它的优缺点分别是什么？

（必会）

单页 Web 应用（single-page application 简称为 SPA）是一种特殊的 Web 应用。它将所有的活动局限于一个 Web 页面中，仅在该 Web 页面初始化时加载相应的 HTML、JavaScript 和 CSS。一旦页面加载完成了，SPA 不会因为用户的操作而进行页面的重新加载或跳转。取而代之的是利用 JavaScript 动态的变换 HTML 的内容，从而实现 UI 与用户的交互。由于避免了页面的重新加载，SPA 可以提供较为流畅的用户体验。得益于 ajax，我们可以实现无跳转刷新，又多亏了浏览器的 histroy 机制，我们用 hash 的变化从而可以实现推动界面变化。从而模拟元素客户端的单页面切换效果：

SPA 被人追捧是有道理的，但是它也有不足之处。当然任何东西都有两面性，以下是卤煮总结的一些目前 SPA 的优缺点：

1) 优点：

- 1、无刷新界面，给用户体验原生的应用感觉
- 2、节省原生（android 和 ios）app 开发成本
- 3、提高发布效率，无需每次安装更新包。这个对于 ios 开发人员来说印象尤其深吧。
- 4、容易借助其他知名平台更有利于营销和推广

5、符合 web2.0 的趋势

2) 缺点:

- 1、效果和性能确实和原生的有较大差距
- 2、各个浏览器的版本兼容性不一样
- 3、业务随着代码量增加而增加，不利于首屏优化
- 4、某些平台对 hash 有偏见，有些甚至不支持 pushstate
- 5、不利于搜索引擎抓取

65、怎样理解 Vue 的单向数据流？（必会）

数据从父级组件传递给子组件，只能单向绑定。

子组件内部不能直接修改从父级传递过来的数据。

所有的 prop 都使得其父子 prop 之间形成了一个单向下行绑定：父级 prop 的更新会向下流动到子组件中，但是反过来则不行。

这样会防止从子组件意外改变父级组件的状态，从而导致你的应用的数据流向难以理解。

额外的，每次父级组件发生更新时，子组件中所有的 prop 都将会刷新为最新的值。

这意味着你不应该在一个子组件内部改变 prop。如果你这样做了，Vue 会在浏览器的控制台中发出警告。

子组件想修改时，只能通过 \$emit 派发一个自定义事件，父组件接收到后，由父组件修改。

66、VNode 是什么？什么是虚拟 DOM？（高薪常问）

1) VNode 是什么

VNode 是 JavaScript 对象，VNode 表示 Virtual DOM，用 JavaScript 对象来描述真实的 DOM 把 DOM 标签，属性，内容都变成对象的属性。就像使用 JavaScript 对象对一种动物进行说明一样 {name: 'Hello Kitty', age: 1, children: null}。

2) VNode 的作用

通过 render 将 template 模版描述成 VNode，然后进行一系列操作之后形成真实的 DOM 进行挂载。

3) VNode 的优点

1、兼容性强，不受执行环境的影响。VNode 因为是 JS 对象，不管 Node 还是浏览器，都可以统一操作，从而获得了服务端渲染、原生渲染、手写渲染函数等能力。

2、减少操作 DOM，任何页面的变化，都只使用 VNode 进行操作对比，只需要在最后一步挂载更新 DOM，不需要频繁操作 DOM，从而提高页面性能。

4) 什么是虚拟 DOM?

文档对象模型或 DOM 定义了一个接口，该接口允许 JavaScript 之类的语言访问和操作 HTML 文档。元素由树中的节点表示，并且接口允许我们操纵它们。但是此接口需要付出代价，大量非常频繁的 DOM 操作会使页面速度变。

Vue 通过在内存中实现文档结构的虚拟表示来解决此问题，其中虚拟节点（VNode）表示 DOM 树中的节点。当需要操纵时，可以在虚拟 DOM 的内存中执行计算和操作，而不是在真实 DOM 上进行操纵。这自然会更快，并且允许虚拟 DOM 算法计算出最优化的方式来更新实际 DOM 结构，一旦计算出，就将其应用于实际的 DOM 树，这就提高了性能，这就是为什么基于虚拟 DOM 的框架（例如 Vue 和 React）如此突出的原因。

67、Vue 中如何实现一个虚拟 DOM？说说你的思路（高薪常问）

首先要构建一个 VNode 的类，DOM 元素上的所有属性在 VNode 类实例化出来的对象上都存在对应的属性。例如 tag 表示一个元素节点的名称，text 表示一个文本节点的文本，children 表示子节点等。将 VNode 类实例化出来的对象进行分类，例如注释节点、文本节点、元素节点、组件节点、函数式节点、克隆节点。

然后通过编译将模板转成渲染函数 render，执行渲染函数 render，在其中创建不同类型的 VNode 类，最后整合就可以得到一个虚拟 DOM（vnode）。

最后通过 patch 将 vnode 和 oldVnode 进行比较后，生成真实 DOM。

68、Vue 中操作 data 中数组的方法中哪些可以触发视图更新，哪些不可以，不可以的话有什么解决办法？（高薪常问）

push()、pop()、shift()、unshift()、splice()、sort()、reverse() 这些方法会改变被操作的数组；filter()、concat()、slice() 这些方法不会改变被操作的数组，返回一个新的数组；以上方法都可以触发视图更新。

利用索引直接设置一个数组项，例：this.array[index] = newValue

直接修改数组的长度，例：this.array.length = newLength

以上两种方法不可以触发视图更新；

解决方法 1：可以用 this.\$set(this.array, index, newValue) 或 this.array.splice(index, 1, newValue)

解决方法 2：可以用 this.array.splice(newLength)

69、Vue 中怎么重置 data？（高薪常问）

要初始化 data 中的数据，可以使用 Object.assign() 方法，实现重置 data 中的数据，以下就是对该方法的详细介绍，以及如何使用该方法，重置 data 中的数据

1) Object.assign() 方法基本定义

Object.assign() 方法用于将所有可枚举属性的值从一个或多个源对象复制到目标对象。它将返回目标对象。

用法: `Object.assign(target, ...sources)`, 第一个参数是目标对象, 第二个参数是源对象, 就是将源对象属性复制到目标对象, 返回目标对象

2) 具体使用方式

使用 `Object.assign()`, `vm.$data` 可以获取当前状态下的 `data`, `vm.$options.data(this)` 可以获取到组件初始化状态下的 `data`, 复制 `Object.assign(this.$data, this.$options.data(this))` // 注意加 `this`, 不然取不到 `data() { a: this.methodA }` 中的 `this.methodA`。

70、如何对 Vue 首屏加载实现优化? (高薪常问)

- 1、把不常改变的库放到 `index.html` 中, 通过 `cdn` 引入
- 2、Vue 路由的懒加载
- 3、不生成 `map` 文件
- 4、Vue 组件尽量不要全局引入
- 5、使用更轻量级的工具库
- 6、开启 `gzip` 压缩
- 7、首页单独做服务端渲染

71、Vue 的 nextTick 的原理是什么? (高薪常问)

1. 为什么需要 `nextTick`, Vue 是异步修改 DOM 的并且不鼓励开发者直接接触 DOM, 但有时候业务需要必须对数据更改一刷新后的 DOM 做相应的处理, 这时候就可以使用 `Vue.nextTick(callback)` 这个 api 了。

2. 理解原理前的准备 首先需要知道事件循环中宏任务和微任务这两个概念, 常见的宏任务有 `script`, `setTimeout`, `setInterval`, `setImmediate`, I/O, UI rendering 常见的微任务有 `process.nextTick(Nodejs)`, `Promise.then()`, `MutationObserver`;

3. 理解 `nextTick` 的原理正是 Vue 通过异步队列控制 DOM 更新和 `nextTick` 回调函数先后执行的方式。如果大家看过这部分的源码, 会发现其中做了很多 `isNative()` 的判断, 因为这里还存在兼容性优雅降级的问题。可见 Vue 开发团队的深思熟虑, 对性能的良苦用心。

72、在 Vue 实例中编写生命周期 hook 或其他 option/property 时, 为什么不使用箭头函数? (高薪常问)

箭头函数自己没有定义 `this` 上下文, 而是绑定到其父函数的上下文中。当你在 Vue 程序中使用箭头函数 (`=>`) 时, `this` 关键字不会绑定到 Vue 实例, 因此会引发错误。所以强烈建议改用标准函数声明。

73、is 这个特性你有用过吗？主要用在哪些方面？（高薪常问）

1) 动态组件

`<component :is="componentName"></component>`， `componentName` 可以在本页面已经注册的局部组件名和全局组件名，也可以是一个组件的选项对象。当控制 `componentName` 改变时就可以动态切换选择组件。

2) is 的用法

有些 HTML 元素，诸如 `<ul>`、`<ol>`、`<table>`和`<select>`，对于哪些元素可以出现在其内部是有严格限制的。

而有些 HTML 元素，诸如 `<li>`、`<tr>` 和 `<option>`，只能出现在其它某些特定的元素内部。

```
<ul>
  <card-list></card-list>
</ul>
```

所以上面`<card-list></card-list>`会被作为无效的内容提升到外部，并导致最终渲染结果出错。应该这么写：

```
<ul>
  <li is="cardList"></li>
</ul>
```

74、scss 是什么？在 Vue.cli 中的安装使用步骤是？有哪几大特性？（高薪常问）

答：css 的预编译。

使用步骤：

第一步：先装 `css-loader`、`node-loader`、`sass-loader` 等加载器模块

第二步：在 `build` 目录找到 `webpack.base.config.js`，在那个 `extends` 属性中加一个拓展 `scss`

第三步：在同一个文件，配置一个 `module` 属性

第四步：然后在组件的 `style` 标签加上 `lang` 属性，例如：`lang="scss"`

特性：

可以用变量，例如（`$变量名称=值`）；

可以用混合器，例如（`~`）

可以嵌套

75、请详细介绍一些 package.json 中的配置的作用（了解）

- 1、Name: 项目名称
- 2、Version: 项目版本
- 3、Description: 项目描述

- 4、 Author: 作者
- 5、 Prinate: 项目是否私有
- 6、 Scripts:npm run *** 命令用于调用 node 执行的. js 文件

人力资源中台项目

1、开发背景

1.1 项目介绍

HR 人力资源项目采用 RBAC 权限设计思想，研发的一款便于现代企业 HR 管理资源的系统，为用户提供了登录、主页查看、公司组织架构、员工、权限等核心模块，并集成了工资、社保、审批、考勤等常见业务功能、可用于企业多角色根据权限进行登录使用，可以对公司人员组织架构，实现查看、采取单页面应用实现页面的局部刷新效果

1.2 Vue 简介

Vue 构建用户界面的渐进式框架，与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用，Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合，另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动

1.3 Vue 技术发展历程

1. 2013 年，在 Google 工作的尤雨溪，受到 Angular 的启发，从中提取自己所喜欢的部分，开发出了一款轻量框架，最初命名为 Seed。
2. 同年 12 月，这粒种子发芽了，更名为 Vue，版本号是 0.6.0。
3. 2014.01.24，Vue 正式对外发布，版本号是 0.8.0。
4. 发布于 2014.02.25 的 0.9.0，有了自己的代号：Animatrix，这个名字来自动画版的《骇客帝国》，此后，重要的版本都会有自己的代号。
5. 0.12.0 发布于 2015.06.13，代号 Dragon Ball（龙珠），这一年，Vue 迎来了大爆发，Laravel 社区（一款流行的 PHP 框架的社区）首次使用 Vue（我也是在这个论坛上认识 Vue 的），Vue 在 JS 社区也打响了知名度。Evangelion（新世纪福音战士）是 Vue 历史上的第一个里程碑。
6. 同年，Vue-Router（2015-08-18）、Vuex（2015-11-28）、Vue-cli（2015-12-27）相继发布，标志着 Vue 从一个视图层库发展为一个渐进式框架。很多前端同学也是从这个版本开始成为 Vue 的用户。
7. 2.0.0 Ghost in the Shell（攻壳机动队）是第二个重要的里程碑，它吸收了 React 的 Virtual Dom 方案，还支持服务端渲染。
8. 就在不久前，Vue 发布了 2.6.0 Macross（超时空要塞），这是一个承前启后的版本，因为它之后，3.0.0 也呼之欲出了。

1.4 SPA 单页应用与多页应用

1. 多页面每次页面跳转，后台都会返回一个新的 HTML 文档，就是多页面应用

多页应用



2. 单页面：使用 Vue 写的项目就是单页面应用，刷新页面会请求一个 Html 文件，切换页面的时候，并不会发起新的请求一个新的 html 文件，只是页面的内容发生了变化

单页应用



2、系统架构

2.1 Vue-cli 脚手架工具

Vue 脚手架指的是 Vue-cli，它是一个专门为单页面应用快速搭建繁杂的脚手架，它可以轻松的创建新的应用程序而且可用于自动生成 Vue 和 webpack 的项目模板，Vue-cli 是有 Vue 提供的一个官方 cli，专门为单页面应用快速搭建繁杂的脚手架。它是用于自动生成 Vue.js+webpack 的项目模板，是为现代前端工作流提供了 batteries-included

2.2 Element-ui 框架

Element UI 是一套为开发者、设计师和产品经理准备的基于 Vue 2.0 的桌面端元组件库，由饿了么前端团队推出。它并不依赖于 Vue，却是一个十分适合 Vue 项目的框架。可使用 Element UI 轻松制作出网页，为前端开发人员大大减轻了代码负担

2.3 Vue-cli 项目目录结构

```
|--- build                # 构建相关
|--- mock                # 项目mock 模拟数据
|--- public              # 静态资源
|   |--- favicon.ico    # favicon图标
|   |--- index.html      # html模板
|--- src                 # 源代码
|   |--- api             # 所有请求
|   |--- assets          # 主题 字体等静态资源 不会参与打包 直接直出
|   |--- components     # 全局公用组件 和业务不相关 上传组件
|   |--- icons           # 项目所有 svg icons
|   |--- layout          # 全局 layout 负责搭建项目的整体架子结构 html结构
|   |--- router          # 路由
|   |--- store           # 全局 store管理 vuex管理数据的位置 模块化开发 全局getters
|   |--- styles          # 全局样式
|   |--- utils           # 全局公用方法 request.js
|   |--- vendor          # 公用vendor
|   |--- views           # views 所有页面 路由级别的组件
|   |--- App.vue         # 入口页面 根组件
|   |--- main.js         # 入口文件 加载组件 初始化等
|   |--- permission.js   # 权限管理
|   |--- settings.js     # 配置文件
|--- tests               # 测试
|--- .env.xxx            # 环境变量配置
|--- .eslintrc.js        # eslint 配置项
|--- .babelrc            # babel-loader 配置
|--- .travis.yml         # 自动化CI配置
|--- vue.config.js       # vue-cli 配置
|--- postcss.config.js   # postcss 配置
|--- package.json        # package.json
```

2.4 Vue-cli 项目运行机制说明

```
├── src                                # 源代码
│   ├── api                          # 所有请求
│   ├── assets                       # 主题 字体等静态资源
│   ├── components                   # 全局公用组件
│   ├── icons                       # 项目所有 svg icons
│   ├── layout                       # 全局 layout
│   ├── router                       # 路由
│   ├── store                       # 全局 store管理
│   ├── styles                       # 全局样式
│   ├── utils                       # 全局公用方法
│   ├── vendor                      # 公用vendor
│   ├── views                       # views 所有页面
│   ├── App.vue                     # 入口页面
│   ├── main.js                     # 入口文件 加载组件 初始化等
│   ├── permission.js               # 权限管理
│   └── settings.js                 # 配置文件
```

2.5 开发规范

为了实现符合标准规范的开发方式,Vue-cli 对以下几点做出了几点约束规范:

- 文件夹

- 文件夹名称应统一格式,小写开头,见名思意,page 页面下的文件夹名称统一以 page 结尾,例如:homePage,loginPage。其余文件夹名称统一按照项目结构目录命名规范统一命名。

- 组件

组件名以单词大写开头,当多个单词拼写成的组件时,采用驼峰式命名规则。一般是多个单词全拼,减少简写的情况,这样增加可读性。

组件应该都放到 components 文件夹下,单个页面独立一个文件夹,用来放相对应的 Vue 文件以及页面相关的样式文件,样式少可直接写到页面组件里边,这样更符合组件化的思想。

- 公用组件应该统一放到 public 文件下。数据绑定及事件处理同 `Vue.js` 规范,同时补充了 App 及页面的生命周期

- 基础组件

当项目中需要自定义比较多的基础组件的时候,比如一些 button, input, icon, 建议以一个统一的单词 Base 开头,或者放到 base 文件夹统一管理,这样做的目的是为了更方便查找。

页面级组件应该放到相对应页面文件夹下,比如一些组件只有这个页面用到,其他地方没有用到的,可以直接放到页面文件夹,然后以父组件开头命名,例如:

HomeHeader.Vue, HomeNav.Vue。

- 项目级组件一般放到公共文件夹 public 下给所有的页面使用。

- 组件结构

- 组件结构遵循从上往下 templat,script,style 的结构

3、技术架构

4、开发环境与技术

4.1 nodejs 环境

nodejs 是当下前端工程化必不可少的环境，使用 nodejs 的 npm 功能来管理项目中的依赖包
查看 nodejs 和 npm 版本号

```
node -V #查看 node 版本
npm -V # 查看 npm 版本
```

4.2 git 版本控制

git 版本控制工具，是目前最为主流的分布式版本管理工具，代码的提交，检出，日志都需要通过 git 完成

```
$ git --version # 查看 git 安装版本
```

4.3 npm 淘宝镜像

npm 是非常重要的 npm 管理工具，由于 npm 的服务器位于国外，所以一般建议将 npm 设置成国内的淘宝镜像

```
$ npm config set registry https://registry.npm.taobao.org/ #设置
淘宝镜像地址
$ npm config get registry #查看镜像地址
```

4.4 VsCode 编辑器

vscode 编辑器是目前前端开发的编码利器，以及丰富的插件系统，非常适合开发前端项目
(vetur + eslint)
除此之外，eslint 需要在 vscode 中进行一些参数的配置

```
{
  "eslint.enable": true,
  "eslint.run": "onType",
}
```

```
    "eslint.options": {
      "extensions": [
        ".js",
        ".Vue",
        ".jsx",
        ".tsx"
      ]
    },
    "editor.codeActionsOnSave": {
      "source.fixAll.eslint": true
    }
  }
}
```

4.4 关键技术

- 1、基于 Vue2.0 语法开发
- 2、使用 Vue-cli 创建项目基本目录
- 3、结合 Vue-element-admin 组件库搭建项目页面
- 4、使用 Vue-Router 约定路由规则
- 5、配置 axios 完成前后端数据之间的交互
- 6、在 Vue 项目中使用 SCSS 完成对样式修改
- 7、父子组件以及非父子组件之间传值
- 8、结合 webpack 实现对项目后期的优化, 以及打包
- 9、使用 echarts 封装雷达图组件
- 10、screenfull 插件, 实现全屏多语言切换
- 11、使用 git 完成对项目的管理工作

4.6 类似组件库

- 1、**element-ui**
饿了么团队研发开源的一套 pc 端的组件库
- 2、**Vue-manage-system**
基于 Vue + Element UI 的后台管理系统解决方案
- 3、**Vuetify-material-dashboard**
一个基于 Vuetify 设计风格的管理系统

4.7 API 文档

登录

基本信息

Path: /api/sys/login

Method: POST

接口描述:

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
mobile	string	必须		手机号	
password	string	必须		密码	

返回数据

名称	类型	是否必须	默认值	备注	其他信息
success	boolean	必须		成功状态	
code	number	必须		状态码	
message	string	必须		消息	
data	string	必须		用户token	

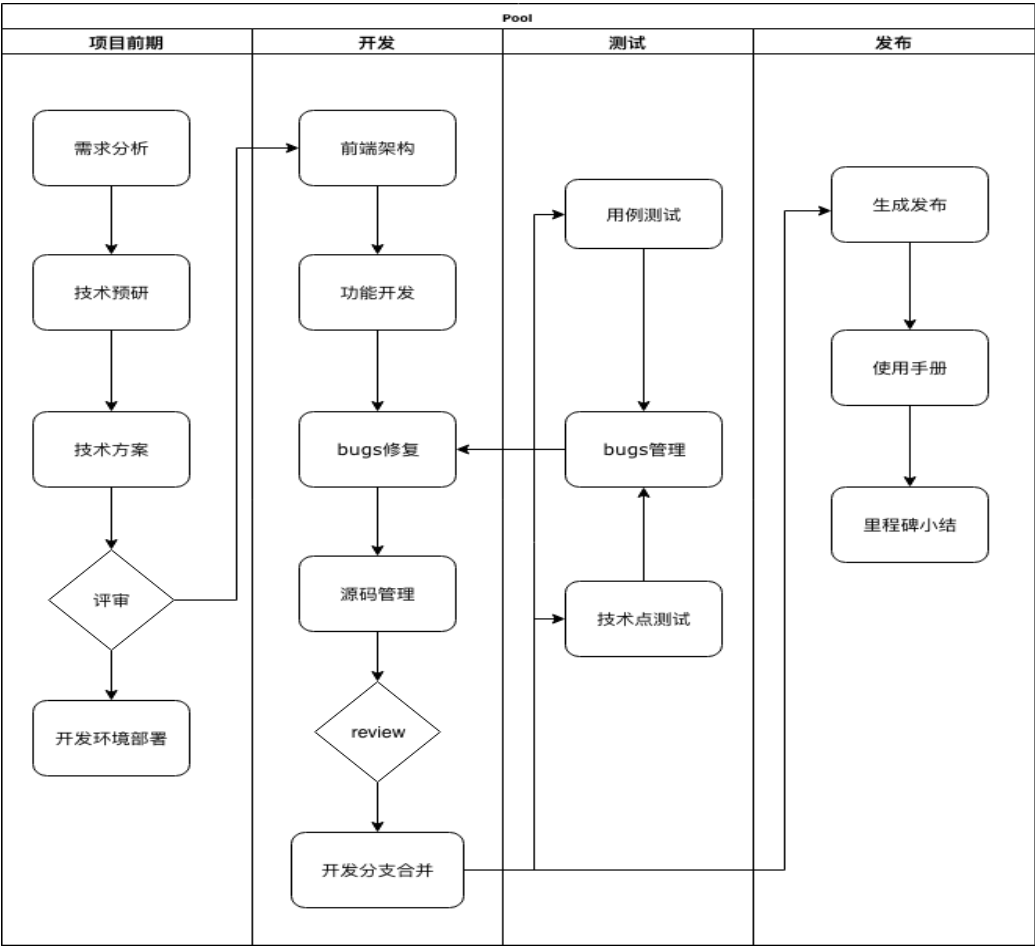
4.8 人员配置

- 产品经理：1，确定需求以及给出产品原型图
- 项目经理：1 人，项目管理
- 前端团队：2 人，根据产品经理给出的原型图制作静态页面
- 后端团队：2 人，根据项目经理分配的任务完成产品功能
- 测试团队：0 人，前后端开发人员联调解决

运维团队：1 人，项目的发布以及维护，其中小程序的上线和发布由前端团队完成

4.9 开发流程

产品提出需求-- 画出原型图-- 需求评审会议-- 安排工期(各部门制定)--- UI 设计图-- 前端开发-- 后端开发(顺序不一定)--- 测试阶段-- 上线--- 回测里程碑小结--->维护项目



5、项目功能架构



6、登录模块

6.1 业务实现思路

- 1、结合业务需要，实现样式修改，设置 Rules 校验函数，对手机号和密码实现校验
- 2、使用 token 信息作为用户登陆的唯一标识，并且存储在 LocalStorage 中，通过 Vuex 统一管理 token，并且实现 token 持久化
- 3、利用 axios 中设置请求拦截器，在每次请求的请求头中，注入 token 信息，作为登录的标识
- 4、配合 Vue-Router 中的 beforeEach 前置导航守卫函数，实现对 token 信息的统一监测，和拦截登录

6.2 技术亮点

Token, LocalStorage, Vue-Router, axios

7、主页模块

7.1 业务实现思路

- 1、登录成功后,根据业务需求,配合 scss 实现对样式的二次修改
- 2、初始化 Vuex 中的 mutations 信息,更新登录后用户的信息收集,封装 action 获取用户资料
- 3、利用 Vuex 中的 getters 属性,完成用户登录后的视图层渲染
- 4、封装 action,实现用户退出登录,调用 commit 方法,清除 Vuex 中保存的 token 信息
- 5、根据后端检测 token 返回的状态码,设置拦截器,对失效 token 信息实现拦截登录,并提示用户 token 失效

7.2 技术亮点

Vuex, token, scss, 请求拦截器

8、组织架构

8.1 业务实现思路

- 1、利用 el-tree 组件,实现结构层渲染,结合 slot 插槽,渲染数据内容
- 2、采用对象键值对的特点,封装方法,实现对 tree 真实数据的替换和渲染
- 3、利用 es6 中的模块化特点,配合 async 与 await 封装对部门数据的增加,删除,新增,编辑功能
- 4、利用 Vue 组件中的子父组件传值的方式,实现对 Dialog 弹框优化

8.2 技术亮点

Tree 树形控件基本使用及动态数据处理、es6 模块化、async/await 使用方式

9、角色管理

9.1 业务实现思路

- 1、借助 element-ui 中的 el-tabs、el-table、el-form 实现基础的结构搭建
- 2、通过属性绑定的方式,绑定角色管理动态数据
- 3、通过作用域插槽 slot-scope 实现删除按钮的注册事件
- 4、通过设置 rules 兜底校验,完成对新增业务的校验
- 5、封装独立编辑业务 api 接口函数,通过 \$refs 操作表单属性,实现按需获取编辑接口

数据

9.2 技术亮点

表单基本绘制，作用域插槽，rules 兜底检验，api 接口封装

10、员工管理

10.1 业务实现思路

- 1、按照组件封装逻辑，实现工具栏通用组件的业务封装，并注册到全局使用
- 2、定义根据字段进行排序，结合 slot-scope，及 prop 属性，实现升序和降序排列
- 3、封装分页组件逻辑代码，实现员工管理表格数据中的分页展示效果
- 4、通过 el-table-column 的 formatter 属性绑定聘用格式化函数，安装 day.js 实现日期格式化处理
- 5、利用 xlsx 插件，通过组件注册的方式拓展为全局插件，实现对表格数据的批量导出

10.2 技术亮点

组件封装逻辑，分页组件，slot-scope，数据格式化，day.js 插件使用，xlsx 插件，excel 组件导出数据

11、员工详情

11.1 业务实现思路

- 1、腾讯云 cos 申请，配置，完成对图片云托管的基本搭建和配置
- 2、封装公共上传组件，结合业务需求，实现对上传图片格式校验
- 3、配置 cos 实例对象，完成基本的 ID 和密钥的配置，实现上传功能
- 4、安 qrcode 生成二维码插件，完成店家员工时，弹出二维码功能

11.2 技术亮点

Cos 腾讯云图床配置使用，组件封装，qrcode 插件配置使用

12、权限点管理

12.1 业务实现思路

- 1、通过树形操作方法，将列表数据转化成层级数据展示，并设置 row-key 属性，实现访问权限
- 2、采用 el-dialog 弹窗组件，结合 type 属性，实现新增权限点的操作
- 3、通过接口封装，完成权限点的删除，和编辑，以及重置等后续功能

12.2 技术亮点

Tree 转换树形数据，接口封装调用，表单绑定，删除逻辑，编辑逻辑

13、RBAC 权限设计及应用

13.1 业务实现思路

- 1、新建弹窗组件，根据对应 api 接口，完成用户数据展示，使用 \$emit 父子组件传值方式，实现关闭弹窗
- 2、根据属性，使用 tree 组件展示权限点数据，结合 async/await 完成数据回显
- 3、采用 addRouters 方法，更改成动态添加式路由，从新改写 Vuex 中的动态路由逻辑，实现菜单生成
- 4、通过提供的标识信息，结合 filter 方法，完成对权限数据的过滤筛选，并渲染视图
- 5、通过 commit() 方法，操作 Vuex 中的状态数据，完成对 404 页面优化，以及退出登录重置路由操作

13.2 技术亮点

组件传值，tree 树形数据，Vuex 操作，ES6-filter 方法使用，addRouters 动态路由配置

14、首页功能实现

14.1 业务实现思路

- 1、使用全局 Vuex 中的 getters 属性，访问模块中的数据，映射 userName 属性，显示用户资料
- 2、抽离日历组件，配合作用域插槽渲染日历数据
- 3、安装 antv/data-set 和 antv/g2 插件，实现雷达图的初始化，以及可视化数据展示

14.2 技术亮点

Vuex 操作使用，日历组件抽离，antv 数据可视化插件配置和使用

15、全屏-多语言

15.1 业务实现思路

- 1、配置 ScreenFull 全屏插件，通过 document 操作属性，实现开启和关闭全屏，监听事件，完成 esc 退出全屏功能
- 2、安装 Vue-i18n 插件，导入 element-ui/locale 插件，配置基本信息，实现动态切换中英文效果

15.2 技术亮点

ScreenFull 插件安装使用，Vue-i18n 插件基本使用，以及动态切换逻辑

16、打包优化发布

16.1 业务实现思路

- 1、preview 插件实现项目文件体积分析
- 2、配置 webpack 排除打包
- 3、CDN 文件配置，实现对项目的资源托管
- 4、修改 Router 模式，防止服务器解析资源报错问题
- 5、安装 koa 中间件，配置 koa2-proxy-middleware 信息实现代理跨域访问
- 6、修改生产环境 baseUrl，执行 run build 命令，实现对项目整体的打包和资源整合

16.2 技术亮点

Preview 插件使用，webpack 打包项目、CDN 资源托管，koa 中间件使用配置，koa-proxy 跨域代理方案，Router 路由模式切换

17、项目介绍话术

首先描述自己所做项目有哪些功能模块，然后描述其中单个模块有哪些功能，再对其中的一个单独功能进行详细描述，中间可以穿插一下遇到的技术问题，循环往复，和面试官保持平等对话。

举例：

我在上家单位最近做的项目是一个基于 Vue 开发的什么类型项目，我负责的模块有登录模块，组织架构模块，主页模块，等核心模块，其中在登录模块中，采取 Vuex 保存登录时生成 token 信息，同时借助 axios 中提供的请求拦截器和相应拦截，由于后端接口设计原因，在发送请求时产生了跨域影响，解决方案是在 vue.config.js 中配置代理，解决接口设计产生跨域的问题。

18、开发中遇到的问题

1. Vue 中使用样式无法生效
2. 页面预渲染问题（seo 问题）
3. 在 Vue 中使用 axios 下载 excel 文档，出现乱码
4. 需要把 Vue 项目兼容 IE 浏览器
5. cookie 和 token 都存在在 header 中，为什么不会劫持 token
6. 使用 webpack 打包项目，导致图片无法显示
7. 如何解决在使用 webpack 打包过程中很慢的问题
8. Vue-Router 路由切换 组件重用挖下的坑

小程序

1、如何获得用户的授权信息？（必会）

通过 wx.getUserProfile() 方法 和小程序 wx.getSetting() 方法获取

2、数据绑定如何实现？（必会）

使用 mustache 语法实现数据和属性的绑定

3、列表渲染如何实现？（必会）

在 wxml 标签添加 wx:for 属性并赋值循环数据即可渲染

4、事件及事件绑定是什么？（必会）

事件是视图层到逻辑层的通讯方式

事件可以将用户的行为反馈到逻辑层进行处理

事件可以绑定在组件上，当达到触发事件，就会执行逻辑层中对应的事件处理函数

事件对象可以携带额外信息，如 id, dataset, touches

事件分为冒泡事件和非冒泡事件：

冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递

非冒泡事件：当一个组件上的事件被触发后，该事件不会向父节点传递。事件绑定的写法类似于组件的属性，如

```
<view bindtap="handleTap">
  Click here!
</view>
```

5、页面跳转的方式有哪些？（必会）

1. 声明式导航：使用 `navigator` 组件实现页面的跳转
2. 编程式导航：使用小程序提供的 API 实现页面的跳转
 - `wx.switchTab()`
 - `wx.navigateTo()`
 - `wx.navigateBack()` 等
 -

6、tabBar 配置参数有哪些？（必会）

属性	类型	必填	默认值	描述	最低版本
color	HexColor	是		tab 上的文字默认颜色，仅支持十六进制颜色	
selectedColor	HexColor	是		tab 上的文字选中时的颜色，仅支持十六进制颜色	
backgroundColor	HexColor	是		tab 的背景色，仅支持十六进制颜色	
borderStyle	string	否	black	tabbar 上边框的颜色，仅支持 black / white	
list	Array	是		tab 的列表，详见 list 属性说明，最少 2 个、最多 5 个 tab	
position	string	否	bottom	tabBar 的位置，仅支持 bottom / top	
custom	boolean	否	false	自定义 tabBar，见详情	2.5.0

7、页面生命周期包含那几个？（必会）

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

2. `onShow()` 页面显示/切入前台时触发
3. `onReady()` 页面初次渲染完成时触发。一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互
4. `onHide()` 页面隐藏/切入后台时触发。如 `navigateTo` 或底部 `tab` 切换到其他页面，小程序切入后台等
5. `onUnload()` 页面卸载时触发。如 `redirectTo` 或 `navigateBack` 到其他页面时

8、转发分享如何实现？（必会）

获取更多转发信息

通常开发者希望转发出去的小程序被二次打开的时候能够获取到一些信息，例如群的标识。现在通过调用 `wx.showShareMenu` 并且设置 `withShareTicket` 为 `true`，当用户将小程序转发到任一群聊之后，此转发卡片在群聊中被其他用户打开时，可以在 `App.onLaunch` 或 `App.onShow` 获取到一个 `shareTicket`。通过调用 `wx.getShareInfo` 接口传入此 `shareTicket` 可以获取到转发信息。

页面内发起转发

基础库 1.2.0 开始支持，低版本需做兼容处理。

通过给 `button` 组件设置属性 `open-type="share"`，可以在用户点击按钮后触发 `Page.onShareAppMessage` 事件，相关组件：`button`。

9、如何获取地理位置？（必会）

首先要通过 `wx.openSetting` 接口拿到用户的授权，在拿到用户授权以后，使用微信的 `wx.getLocation` 接口获取当前位置的经纬度，然后结合第三方地图接口查询区域信息

10、如何封装自定义组件？（必会）

创建自定义组件

类似于页面，一个自定义组件由 `json` `wxml` `wxss` `js` 4个文件组成。要编写一个自定义组件，首先需要要在 `json` 文件中进行自定义组件声明（将 `component` 字段设为 `true` 可将这一组文件设为自定义组件）：

```
{
  "component": true
}
```

同时，还要在 `wxml` 文件中编写组件模板，在 `wxss` 文件中加入组件样式，它们的写法与页面的写法类似。具体细节和注意事项参见 [组件模板和样式](#)。

11、webview 是什么？（必会）

承载网页的容器。会自动铺满整个小程序页面，个人类型的小程序暂不支持使用

12、简单描述下微信小程序的相关文件类型？（必会）

1. WXML 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
2. WXSS 是一套样式语言，用于描述 WXML 的组件样式
3. js 逻辑处理，网络请求
4. json 小程序设置，如页面注册，页面标题及 tabBar
5. 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
6. 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

13、小程序有哪些参数传值的方法？（必会）

1. 给 view 组件添加 data-* 属性来传递我们需要的值，然后通过 e.currentTarget.dataset 或 onload 的 param 参数获取。但 data-名称不能有大写字母和不可以存放对象
2. 设置 id 的方法标识来传值通过 e.currentTarget.id 获取设置的 id 的值，然后通过设置全局对象的方式来传递数值
3. 在 navigator 中添加参数传值

14、简述微信小程序原理？（必会）

1. 微信小程序采用 JavaScript、WXML、WXSS 三种技术进行开发，从技术讲和现有的前端开发差不多，但深入挖掘的话却又有所不同

2. JavaScript: 首先 JavaScript 的代码是运行在微信 App 中的, 并不是运行在浏览器中, 因此一些 H5 技术的应用, 需要微信 App 提供对应的 API 支持, 而这限制住了 H5 技术的应用, 且其不能称为严格的 H5, 可以称其为伪 H5, 同理, 微信提供的独有的某些 API, H5 也不支持或支持的不是特别好
3. WXML: WXML 微信自己基于 XML 语法开发的, 因此开发时, 只能使用微信提供的现有标签, HTML 的标签是无法使用的
4. WXSS: WXSS 具有 CSS 的大部分特性, 但并不是所有的都支持, 而且支持哪些, 不支持哪些并没有详细的文档
5. 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
6. 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

15、小程序的双向绑定和 vue 哪里不一样? (必会)

小程序直接 this、data 的属性是不可以同步到视图的, 必须使用 this.setData() 方法

小程序:

```
Page({
  data: {
    items: []
  },

  onLoad: function(options) {
    this.setData({
      items: [1, 2, 3]
    })
  }
})
```

```
Vue:
new Vue({
  data: {
    items: []
  },
  mounted () {
```



```
    this.items = [1, 2, 3]
  }
})
```

16、分析下微信小程序的优劣势？（必会）

优势：

1. 无需下载，通过搜索和扫一扫就可以打开
2. 良好的用户体验：打开速度快
3. 开发成本要比 App 要低
4. 安卓上可以添加到桌面，与原生 App 差不多
5. 为用户提供良好的安全保障。小程序的发布，微信拥有一套严格的审查流程，不能通过审查的小程序是无法发布到线上的

劣势：

1. 限制较多。页面大小不能超过 1M。不能打开超过 5 个层级的页面
2. 样式单一。小程序的部分组件已经是成型了的，样式很难修改。例如：幻灯片、导航
3. 推广面窄，不能分享朋友圈，只能通过分享给朋友，附近小程序推广。其中附近小程序也受到微信的限制

17、bindtap 和 catchtap 的区别是什么？（必会）

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 是阻止冒泡的

18、简述下 wx.navigateTo(), wx.redirectTo(), wx.switchTab(), wx.navigateBack(), wx.reLaunch() 的区别？（必会）

1. 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
2. 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
3. wx.switchTab(): 跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面
4. 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
5. wx.reLaunch(): 关闭所有页面，打开到应用内的某个页面

19、小程序尺寸单位 rpx？（必会）

WXSS 是一套样式语言，用于描述 WXML 的组件样式，用来决定 WXML 的组件应该怎么显示。为了适应广大的前端开发者，WXSS 具有 CSS 大部分特性。同时为了更适合开发微信小程序，WXSS 对 CSS 进行了扩充以及修改。

与 CSS 相比，WXSS 扩展的特性有：

- 尺寸单位
 - 样式导入
- rpx (responsive pixel)：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 $750\text{rpx} = 375\text{px} = 750\text{物理像素}$ ， $1\text{rpx} = 0.5\text{px} = 1\text{物理像素}$ 。

设备	rpx换算px (屏幕宽度/750)	px换算rpx (750/屏幕宽度)
iPhone5	$1\text{rpx} = 0.42\text{px}$	$1\text{px} = 2.34\text{rpx}$
iPhone6	$1\text{rpx} = 0.5\text{px}$	$1\text{px} = 2\text{rpx}$
iPhone6 Plus	$1\text{rpx} = 0.552\text{px}$	$1\text{px} = 1.81\text{rpx}$

建议： 开发微信小程序时设计师可以用 iPhone6 作为视觉稿的标准。

注意： 在较小的屏幕上不可避免的会有一些毛刺，请在开发时尽量避免这种情况。

20、小程序选择器有哪些？（必会）

目前支持的选择器有：

选择器	样例	样例描述
.class	<code>.intro</code>	选择所有拥有 class="intro" 的组件
#id	<code>#firstname</code>	选择拥有 id="firstname" 的组件
element	<code>view</code>	选择所有 view 组件
element, element	<code>view, checkbox</code>	选择所有文档的 view 组件和所有的 checkbox 组件
::after	<code>view::after</code>	在 view 组件后边插入内容
::before	<code>view::before</code>	在 view 组件前边插入内容

21、小程序常用组件？（必会）

view、swiper、scroll-view、text、button、input、image 等

22、微信小程序长按识别二维码（必会）

image 组件中二维码/小程序码图片不支持长按识别。仅在 wx.previewImage 中支持长按识别示例代码

23、如何封装微信小程序的数据请求(http-promise)？（高薪常问）

- 1、将所有的接口放在统一的 js 文件中并导出
- 2、在 app.js 中创建封装请求数据的方法
- 3、在子页面中调用封装的方法请求数据

24、小程序申请微信支付？（了解）

微信支付支持在注册并完成微信认证的小程序接入支付功能。小程序接入支付后，可以通过小程序支付产品来完成在小程序内销售商品或内容时的收款需求。具体申请流程，可以直接根据注册流程提供相关信息即可

25、客服电话？（了解）

```
wx.makePhoneCall({
  phoneNumber: '1340000xxxx' //仅为示例，并非真实的电话号码
})
```

26、小程序插槽的使用 slot？（了解）

小程序中的插槽分为两类：默认插槽和多个插槽

组件模板

组件模板的写法与页面模板相同。组件模板与组件数据结合后生成的节点树，将被插入到组件的引用位置上。

在组件模板中可以提供一个 `<slot>` 节点，用于承载组件引用时提供的子节点。

代码示例：

在开发者工具中预览效果

```
<!-- 组件模板 -->
<view class="wrapper">
  <view>这里是组件的内部节点</view>
  <slot></slot>
</view>
```

```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot> 的位置上 -->
    <view>这里是插入到组件slot中的内容</view>
  </component-tag-name>
</view>
```

注意，在模板中引用到的自定义组件及其对应的节点名需要在 `json` 文件中显式定义，否则会被当作一个无意义的节点。除此以外，节点名也可以被声明为抽象节点。

组件 wxml 的 slot

在组件的 wxml 中可以包含 `slot` 节点，用于承载组件使用者提供的 wxml 结构。

默认情况下，一个组件的 wxml 中只能有一个 slot。需要使用多 slot 时，可以在组件 js 中声明启用。

```
Component({
  options: {
    multipleSlots: true // 在组件定义时的选项中启用多slot支持
  },
  properties: { /* ... */ },
  methods: { /* ... */ }
})
```

此时，可以在这个组件的 wxml 中使用多个 slot，以不同的 `name` 来区分。

```
<!-- 组件模板 -->
<view class="wrapper">
  <slot name="before"></slot>
  <view>这里是组件的内部细节</view>
  <slot name="after"></slot>
</view>
```

使用时，用 `slot` 属性来将节点插入到不同的 slot 上。

```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot name="before"> 的位置上 -->
    <view slot="before">这里是插入到组件slot name="before"中的内容</view>
    <!-- 这部分内容将被放置在组件 <slot name="after"> 的位置上 -->
    <view slot="after">这里是插入到组件slot name="after"中的内容</view>
  </component-tag-name>
</view>
```

28、如何分包加载？分包加载的优势在哪？（了解）

某些情况下，开发者需要将小程序划分成不同的子包，在构建时打包成不同的分包，用户在使用时按需进行加载。

在构建小程序分包项目时，构建会输出一个或多个分包。每个使用分包小程序必定含有一个**主包**。所谓的主包，即放置默认启动页面/TabBar 页面，以及一些所有分包都需用到公共资源/JS 脚本；而**分包**则是根据开发者的配置进行划分。

在小程序启动时，默认会下载主包并启动主包内页面，当用户进入分包内某个页面时，客户端会把对应分包下载下来，下载完成后再进行展示。

目前小程序分包大小有以下限制：

- 整个小程序所有分包大小不超过 20M
- 单个分包/主包大小不能超过 2M

对小程序进行分包，可以优化小程序首次启动的下载时间，以及在多团队共同开发时可以更好的解耦协作。

具体使用方法请参考：

- [使用分包](#)
- [独立分包](#)
- [分包预下载](#)
- [分包异步化](#)

29、哪些方法可以用来提高微信小程序的应用速度？（了解）

1. 提高页面加载速度
2. 用户行为预测
3. 减少默认 data 的大小
4. 组件化方案

30、webview 中的页面怎么跳回小程序中？（了解）

<web-view/>网页中可使用 JSSDK 提供的接口返回小程序页面

```
wx.miniProgram.navigateTo({  
  url: '/pages/login/login' + '$params'  
})
```

31、小程序如何实现下拉刷新？（了解）

1. 在 json 文件中配置 enablePullDownRefresh 为 true（app.json 中在 window 中设置 enablePullDownRefresh）

2. 在 js 文件中实现 onPullDownRefresh 方法，在网络请求完成后调用 wx.stopPullDownRefresh() 来结束下拉刷新

32、小程序调用后台接口遇到哪些问题？（了解）

- 1、数据的大小有限制，超过范围会直接导致整个小程序崩溃，除非重启小程序；
- 2、小程序不可以直接渲染文章内容页这类型的 html 文本内容，若需显示要借助插件，但插件渲染会导致页面加载变慢，所以最好在后台对文章内容的 html 进行过滤，后台直接处理批量替换 p 标签、div 标签为 view 标签，然后其它的标签让插件来做，减轻前端的时间

优购商城（小程序项目）

1、开发背景

1.1 项目介绍

优购商城是一个小程序电商业务项目，本项目主要功能有首页推荐频道展示、分类筛选、搜索商品、商品详情、分页加载数据及长列表展示优化、购物车、下单、支付、用户个人中心等模块。

1.2 小程序简介

小程序是一种全新的连接用户与服务的方式，它可以在微信内被便捷地获取和传播，同时具有出色的使用体验。

1.3 小程序技术发展史

小程序并非凭空冒出来的一个概念。当微信中的 WebView 逐渐成为移动 Web 的一个重要入口时，微信就有相关的 JS API 了。此类 API 最初是提供给腾讯内部一些业务使用，很多外部开发者发现了之后，依葫芦画瓢地使用了，逐渐成为微信中网页的事实标准。2015 年初，微信发布了一整套网页开发工具包，称之为 JS-SDK，开放了拍摄、录音、语音识别、二维码、地图、支付、分享、卡券等几十个 API。给所有的 Web 开发者打开了一扇全新的窗户，让所有开发者都可以使用到微信的原生能力，去完成一些之前做不到或者难以做到的事情。

JS-SDK 解决了移动网页能力不足的问题，通过暴露微信的接口使得 Web 开发者能够拥有更多的能力，然而在更多的能力之外，JS-SDK 的模式并没有解决使用移动网页遇到的体验不良的问题。用户在访问网页的时候，在浏览器开始显示之前都会有一个的白屏过程，在移动端，受限于设备性能和网络速度，白屏会更加明显。微信团队把很多技术精力放置在如何帮助平台上的 Web 开发者解决这个问题。因此微信设计了一个 JS-SDK 的增强版本，其中有一个重要的功能，称之为“微信 Web 资源离线存储”。

以下文字引用自内部的文档（没有最终对外开放）：

黑马程序员-武汉前端学科出品 www.itheima.com

微信 Web 资源离线存储是面向 Web 开发者提供的基于微信内的 Web 加速方案。

通过使用微信离线存储，Web 开发者可借助微信提供的资源存储能力，直接从微信本地加载 Web 资源而不需要再从服务端拉取，从而减少网页加载时间，为微信用户提供更优质的网页浏览体验。每个公众号下所有 Web App 累计最多可缓存 5M 的资源。

这个设计有点类似 HTML5 的 Application Cache，但在设计上规避了一些 Application Cache 的不足。在内部测试中，微信团队发现离线存储能够解决一些问题，但对于一些复杂的页面依然会有白屏问题，例如页面加载了大量的 CSS 或者是 JavaScript 文件。除了白屏，影响 Web 体验的问题还有缺少操作的反馈，主要表现在两个方面：页面切换的生硬和点击的迟滞感。

微信面临的问题是如何设计一个比较好的系统，使得所有开发者在微信中都能获得比较好的体验。这个问题是之前的 JS-SDK 所处理不了的，需要一个全新的系统来完成，它需要使得所有的开发者都能做到：

- 更强大的能力
- 原生的体验
- 易用且安全的微信数据开放
- 高效和简单的开发
- 快速的加载
- 这就是小程序的由来。

1.4 小程序与普通网页开发的区别

小程序的主要开发语言是 JavaScript，小程序的开发同普通的网页开发相比有很大的相似性。对于前端开发者而言，从网页开发迁移到小程序的开发成本并不高，但是二者还是有些许区别的。

网页开发渲染线程和脚本线程是互斥的，这也是为什么长时间的脚本运行可能会导致页面失去响应，而在小程序中，二者是分开的，分别运行在不同的线程中。网页开发者可以使用到各种浏览器暴露出来的 DOM API，进行 DOM 选中和操作。而小程序的逻辑层和渲染层是分开的，逻辑层运行在 JSCore 中，并没有一个完整浏览器对象，因而缺少相关的 DOM API 和 BOM API。这一区别导致了前端开发非常熟悉的一些库，例如 jQuery、Zepto 等，在小程序中是无法运行的。同时 JSCore 的环境同 nodeJS 环境也是不尽相同，所以一些 NPM 的包在小程序中也是无法运行的。

1.5 为什么要开发小程序

微信有海量用户，而且粘性很高，在微信里开发产品更容易触达用户，触手可及，用完即走

推广 app 或公众号的成本太高

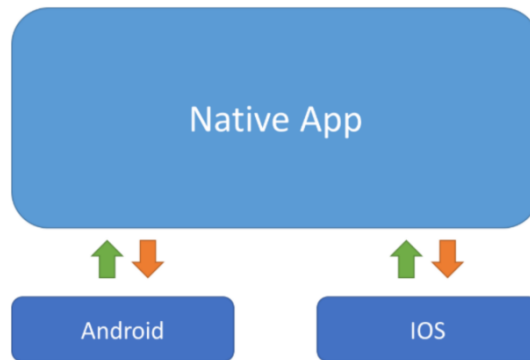
开发适配成本低，拥有和原生 APP 的体验

容易小规模试错，然后快速迭代

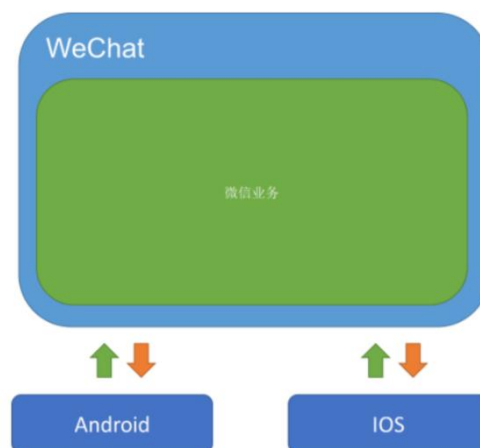
跨平台，面向所有用户开放（企业，组织，个人均可以发布自己的小程序）

2、系统架构

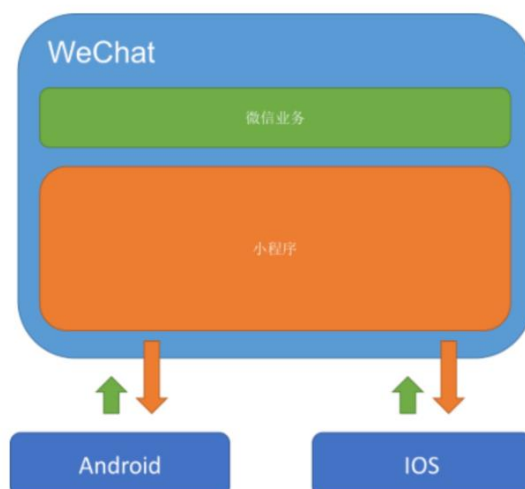
2.1 传统原生 APP



2.2 微信运行环境



2.3 微信小程序运行环境



2.4 mina 框架

mina 框架，又称小程序框架，小程序开发框架的目标是通过尽可能简单、高效的方式让开发者可以在微信中开发具有原生 APP 体验的服务。mina 框架提供了自己的视图层描述语言 WXML 和 WXSS，以及基于 JavaScript 的逻辑层框架，并在视图层与逻辑层间提供了数据传输和事件系统，让开发者能够专注于数据与逻辑。

2.4.1 小程序文件结构和传统 web 对比

结构	传统web	微信小程序
结构	HTML	WXML
样式	CSS	WXSS
逻辑	Javascript	Javascript
配置	无	JSON

通过以上对比得出，传统 web 是三层结构。而微信小程序是四层结构，多了一层 配置. json

2.4.2 基本的目录结构



2.5 uni-app 框架

uni-app 是一个使用 Vue.js 开发所有前端应用的框架，开发者编写一套代码，可发布到 iOS、Android、H5、以及各种小程序（微信/支付宝/百度/头条/QQ/钉钉/淘宝）、快应用等多个平台。

2.5.1 开发规范

为了实现多端兼容，综合考虑编译速度、运行性能等因素，uni-app 约定了如下开发规范：

- 1、页面文件遵循 [Vue 单文件组件 (SFC) 规范]
- 2、组件标签靠近小程序规范，详见[uni-app 组件规范]
- 3、接口能力 (JS API) 靠近微信小程序规范，但需将前缀 `wx` 替换为 `uni`
- 4、数据绑定及事件处理同 `Vue.js` 规范，同时补充了 App 及页面的生命周期
- 5、为兼容多端运行，建议使用 flex 布局进行开发

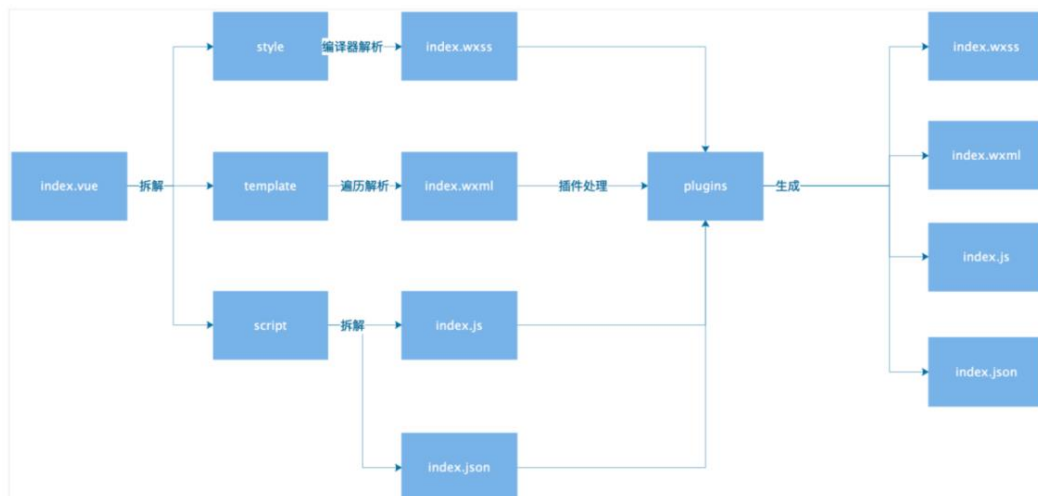
2.5.2 目录结构

一个uni-app工程，默认包含如下目录及文件：

1	components	uni-app组件目录
2	comp-a.vue	可复用的a组件
3	hybrid	存放本地网页的目录，详见
4	platforms	存放各平台专用页面的目录，详见
5	pages	业务页面文件存放的目录
6	index	
7	index.vue	index页面
8	list	
9	list.vue	list页面
10	static	存放应用引用静态资源（如图片、视频等）的目录，注意：静态资源只能存放于此
11	wxcomponents	存放小程序组件的目录，详见
12	main.js	Vue初始化入口文件
13	App.vue	应用配置，用来配置App全局样式以及监听 应用生命周期
14	manifest.json	配置应用名称、appid、logo、版本等打包信息，详见
15	pages.json	配置页面路由、导航条、选项卡等页面类信息，详见
16		

2.5.3 uni-app 组件的编译图解

uni-app 编译的流程如图所示，我们在使用 uniapp 框架之前，最好提前了解一下。本次项目使用的框架就是 uni-app



2.6 其他框架

1. wepy：腾讯团队推出的类 vue 小程序组件化开发框架
2. mpvue：美团类 vue 的小程序前端框架
3. taro：京东 React 语法规范的多端统一开发框架

3、技术架构



优购项目后端开发语言以 node 技术栈为主。另外实际工作开发本项目并不局限于 node，请根据公司后端开发团队掌握的技术语言来回答即可，比如公司后端主要开发语言是 java，那么可以回答本项目是通过 java 语言来编写服务端接口的。另外后端开发语言还有 php、python、node、java、c#等可以作为选择。

4、开发环境与技术

开发小程序之前，必须要准备好相关的环境，并安排好开发人员的工作。

4.1 关键技术

1. 基于 uniapp 框架开发
2. 基于 uni-ui 二次封装 搜索、商品列表、收货地址 等组件
3. 用 Vuex 中的 modules 方法将 store 分割成 模块 (module)
4. 接入微信登录、微信支付等解决方案
5. 使用分包加载的技术，将项目分包，提高首屏加载速度
6. 使用 git 来管理项目

4.2 API 文档

输入关键字后按回车以搜索

首页

轮播图

导航

楼层

分类

商品分类

商品

商品列表搜索

商品详情

商品搜索

用户

获取用户token

支付

获取支付参数

订单

创建订单

查看订单支付状态

历史订单查询

轮播图

简要描述:

• 轮播图

请求URL:

• `https://api-hmugo-web.itheima.net/api/public/v1/home/swiperdata`

请求方式:

• GET

参数: 无

返回示例

```
{
  "message": [
    {
      "image_src": "https://api-hmugo-web.itheima.net/pyg/banner1.png",
      "open_type": "navigate",
      "goods_id": 129,
      "navigator_url": "/pages/goods_detail/index?goods_id=129"
    }
  ],
  "meta": {
    "msg": "获取成功",
    "status": 200
  }
}
```

返回参数说明

参数名	类型	说明
image_src	string	图片路径
open_type	string	打开方式
goods_id	number	商品id
navigator_url	string	导航链接

备注

• 更多返回错误代码请看首页的错误代码描述

4.3 人员配置

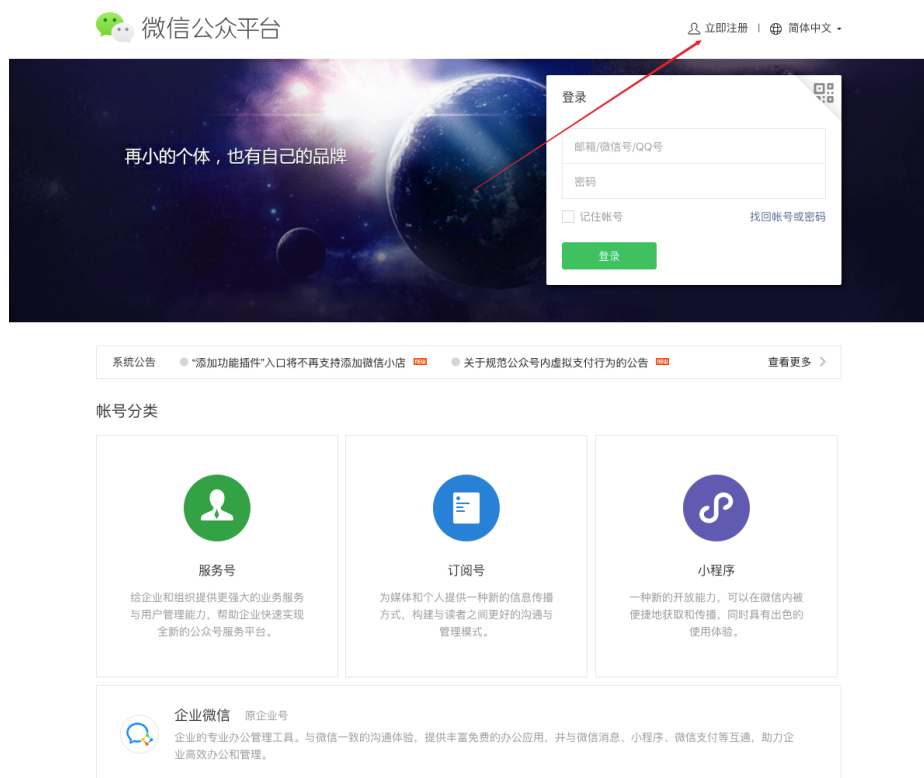
产品经理：1，确定需求以及给出产品原型图。
项目经理：1人，项目管理。
前端团队：2人，根据产品经理给出的原型图制作静态页面。
后端团队：2人，根据项目经理分配的任务完成产品功能。
测试团队：0人，前后端开发人员联调解决。
运维团队：1人，项目的发布以及维护，其中小程序的上线和发布由前端团队完成。

4.4 开发流程

产品提出需求----> 画出原型图---->开会评审---->安排工期(各部门商量)---->ui设计图---->注册小程序账号---->前端开发(后端开发 顺序不一定)---->边开发边自测-->上线---->回测---->维护项目

4.4.1 注册账号

登录微信公众平台，点击立即注册，根据步骤耐心操作即可。



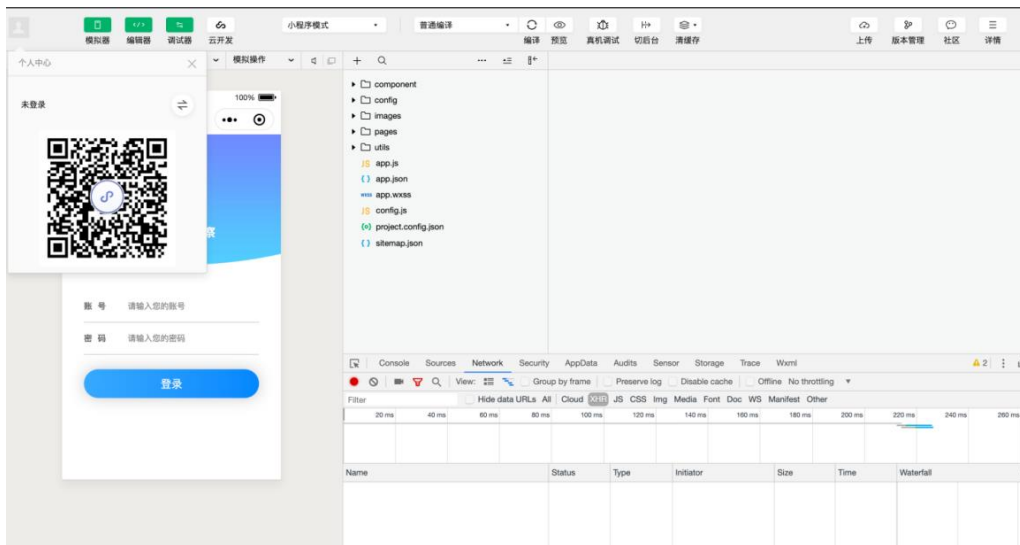
4.4.2 获取 APPID

由于后期调用微信小程序的接口等功能，需要索要开发者的小程序中的 APPID, 所以在注册成功之后，登录公众平台并获取 APPID



4.4.3 开发工具

微信小程序自带开发者工具，集开发，预览，调试，发布于一身的完整环境。下载地址：
<https://developers.weixin.qq.com/miniprogram/dev/devtools/stable.html>



HBuilderX 是通用的前端开发工具，但为 uni-app 做了特别强化。因此开发此项目还需要安装 HBuilderX 编辑器，下载地址：<https://www.dcloud.io/hbuilderx.html>



5、项目架构



6、首页展示

6.1 业务实现思路

1. 使用 tabbar 实现底部的导航功能
2. 利用配置 json 文件配置页面信息
3. 利用自定义组件实现头部搜索框
4. 使用 scss 技术和 flex 技术进行页面布局
5. 利用 swiper 实现轮播图效果
6. 发送网络请求并加载轮播图、导航区域、楼层模块的数据，并组件渲染数据

6.2 技术亮点

1. 入门到进阶：由易到难，掌握使用 Vue 全家桶 + uni-app 进行小程序项目开发
2. 造轮子：基于 uni-ui 二次封装 搜索、商品列表、收货地址 等组件
2. 状态管理模块化：利用 Vuex 中的 modules 方法将 store 分割成 模块 (module)
3. 分包加载：将项目分包，提高首屏加载速度
4. 微信登录 / 微信支付 完整流程，

7、商品分类

7.1 业务实现思路

1. 使用 scroll-view 实现左侧导航功能
2. 利用 css3 伪类实现右侧竖行导航背景
3. 利用双向绑定数据更改，索引位置变化，定位导航位置
4. 初始化页数据，并在 onload 生命周期钩子函数里时请求加载页面分类数据并赋值给页面，右侧数据动态更新

7.2 技术亮点

es7 的 async 和 await 技术，scroll-view 组件

8、商品列表

8.1 业务实现思路

- 1、分析页面结构，并加载商品列表数据
- 2、启用上拉加载事件、下拉页面功能
- 3、编写选项卡效果
- 4、数组拼接并加载下一页

8.2 技术亮点

自定义组件、上拉加载、下拉刷新、条件编译

9、商品详情

9.1 业务实现思路

- 1、分析页面结构，并渲染商品详情数据
- 2、调用 api 实现图片点击预览效果
- 3、绑定事件方法实现收藏和客服功能
- 4、利用导航组件实现页面跳转，并传递相关参数
- 5、打开页面分享函数并配置分享参数
- 6、利用内置 api 实现购物车数据的本地缓存

9.2 技术亮点

事件处理、swiper 组件、本地缓存、富文本兼容性、图片预览接口、Vuex 、购物车

10、搜索页面

10.1 业务实现思路

- 1、分析页面结构，并渲染搜索列表
- 2、输入框事件绑定
- 3、绑定状态控制组件的显示和隐藏

10.2 技术亮点

输入框组件、防抖技术、列表渲染、路由导航、Set 数组去重

11、购物车

11.1 业务实现思路

- 1、分析页面结构并书写页面和布局
- 2、通过内置 api 获取微信用户收货地址
- 3、渲染购物车数据
- 4、根据相关数据填充条件，实现全选和购物车数量的修改

11.2 技术亮点

收货地址、radio 组件、二次封装组件、E6 数组方法、本地存储、网络请求。

12、支付管理

12.1 业务实现思路

- 1、分析页面结构并书写页面和布局
- 2、获取商品数据并渲染订单页面数据
- 3、根据接口参数要求，利用框架内置接口创建微信支付订单，并向用户发起支付
- 4、支付成功之后的页面跳转逻辑的实现

12.2 技术亮点

用户信息、小程序支付。

13、用户登录

13.1 业务实现思路

- 1、分析页面结构并书写页面和布局
- 2、获取小程序 API 获取用户的信息和 登录 code
- 3、根据接口参数要求，发起登录请求，获取 Token
- 4、支付成功之后的页面跳转逻辑的实现

13.2 技术亮点

用户登录、登录 API 使用

17、项目介绍

首先描述自己所做项目有哪些功能模块，然后描述其中单个模块有哪些功能，再对其中的一个单独功能进行详细描述，中间可以穿插一下遇到的技术问题，循环往复，和面试官保持平等对话。

举例：

我在上家公司最近做的项目是一个小程序商城，我负责的模块包括首页模块、分类模块、个人中心模块。其中首页模块主要包括轮播图、导航区域、楼层等功能，轮播图主要是通过 swiper 组件和小程序数据绑定实现的，导航区域利用了小程序的路由导航和 css3 的 flex 弹性布局进行编写实现的页面展示，楼层区域这里判断条件较多，实现起来花了一些时间，因为这里要不仅要保证图片的布局合理，还要保证图片渲染之后，不产生变形，这里用到了小程序内置的图片组件，并通过组件的等比例缩放功能，从而保证了楼层的正确显示。

18、开发中遇到的问题

18.1 域名必须是 HTTPS

在测试阶段小程序请求网络数据都没出现问题，等上线发布时，结果审核失败，小程序页面内无法渲染数据，最后发现是因为腾讯的限制，因为在上架阶段，每个微信小程序必须事先设置一个通讯域名，并通过 HTTPS 请求进行网络通信，不满足条件的域名和协议无法请求。也就是说，请求 request 地址必须是合法域名，需要有 SSL 证书认证过。

18.2 tabbar 在切换时页面数据无法刷新

之前一直是在 onLoad 钩子函数之中调用页面的初始化方法，但是会出现页面数据无法刷新的情况，后来官方文档提供了另一个钩子函数：onShow；这个方法会在页面展示的时候重新执行，这样就可以解决这个问题。

18.3 小程序 image 高度自适应及裁剪问题

在做微信小程序的商品详情页，商品的详情是图片集合，渲染完成后发现图片加载的很不自然，后来我把样式设置宽度 100%，并对 image 组件添加属性 mode="widthFix" 解决了。

18.4 小程序中 canvas 的图片不支持 base64 格式

首先使用 wx.base64ToArrayBuffer 将 base64 数据转换为 ArrayBuffer 数据，使用 FileSystemManager.writeFile 将 ArrayBuffer 数据写为本地用户路径的二进制图片文件，此时的图片文件路径在 wx.env.USER_DATA_PATH 中，wx.getImageInfo 接口能正确

获取到这个图片资源并 drawImage 至 canvas 上。

18.5 wx.setStorageSync 和 wx.getStorageSync 报错问题

因为自身编写的代码逻辑问题，导致没有正确获取数据。修改业务逻辑即可。

18.6 代码审核和发布

首先开发好的小程序代码需要先通过开发者工具提交到微信公众平台进行审核，然后点击提交审核并填写小程序相关资料完成提交，一般会在 7 个工作日之内，小程序完成审核，最后在点击发布，才能让小程序上线。

18.7 小程序微信认证

因为不仔细阅读注册流程说明文档导致认证不成功，因此认证之前，一定要仔细阅读微信认证指南和注意事项。

18.8 图片本地资源名称，尽量使用小写命名

在解决 iPhone X 适配时，底部多余部分使用图片时

```
<image class='iphonexImg' src="/imgs/iphoneBGT.png" mode="aspectFill"></image>
```

路径是 `src='imgs/iphoneBGT.png'`，发现在 pc IDE 上面可以显示出来，但是真机调试时，图片找不到，然后将图片名称改为 `iphonex.png` 真机调试就可以了

```
<image class='iphonexImg' src="/imgs/iphonex.png" mode="aspectFill"></image>
```

写在最后：

代码总是有各种 bug，像上面列举的问题还是在开发中就可以发现。而代码上线以后呢，测试也不能保证 100%没有问题。

React

1、谈谈你对 React 的了解（必会）

React 是用于构建前端页面的 JavaScript 库。React 主要用于构建 UI，由 Facebook 于 2013 年开源。

React 特点:

1. 声明式设计 - React 采用声明范式, 可以轻松描述应用。
2. 高效 - React 通过对 DOM 的模拟, 最大限度地减少与 DOM 的交互。
3. 灵活 - React 可以与已知的库或框架很好地配合。
4. JSX - JSX 是 JavaScript 扩展语法。React 开发不一定使用 JSX, 但我们建议使用它。
5. 组件 - 通过 React 构建组件, 代码更容易复用, 能够很好的应用在大项目的开发中。
6. 单向数据流 - React 实现了单向响应的数据流, 从而减少了重复代码, 这也是它为什么比传统数据绑定更简单。

2、什么是 JSX? 为什么浏览器无法读取 JSX? (必会)

JSX 是 JavaScript XML 的简写, 它利用 JavaScript 的表现力和类似 HTML 的模板语法, 得 HTML 文件非常容易理解。此文件能使应用非常可靠, 并能够提高其性能。浏览器只能处理 JavaScript 对象, 而不能读取常规 JavaScript 对象中的 JSX, 所以为了使浏览器能够读取 JSX, 首先, 需要用 Babel 转换器将 JSX 文件转换为 JavaScript 对象, 然后再将其传给浏览器

3、shouldComponentUpdate 是做什么的? (必会)

shouldComponentUpdate 这个方法用来判断是否需要调用 render 方法重新绘制 dom, 因为 DOM 的描绘非常消耗性能, 如果我们能在 shouldComponentUpdate 方法中能够写出更优化的 dom diff 算法, 可以极大的提高性能。

4、React 性能优化是哪个周期函数? (必会)

shouldComponentUpdate

5、React 中 keys 的作用是什么? (必会)

Keys 是 React 用于追踪哪些列表中元素被修改、被添加或者被移除的辅助标识。在开发过程中, 我们需要保证某个元素的 key 在其同级元素中具有唯一性。

在 React Diff 算法中 React 会借助元素的 Key 值来判断该元素是新近创建的还是被移动而来的元素, 从而减少不必要的元素重渲染;

此外, React 还需要借助 Key 值来判断元素与本地状态的关联关系, 因此我们绝不可忽视转换函数中 Key 的重要性。

6、React 中 refs 的作用是什么? (必会)

Refs 是 React 提供给我们安全访问 DOM 元素或者某个组件实例的句柄, 我们可以为

元素添加 `ref` 属性然后在回调函数中接受该元素在 DOM 树中的句柄，该值会作为回调函数的第一个参数返回。

7、请列举 React 中定义组件的方法？（必会）

1. 函数式定义的无状态组件
2. es6 中 `extends React.Component` 定义的组件

8、调用 `setState` 之后发生了什么？（必会）

1. 代码中调用 `setState` 函数之后，React 会将传入的参数对象与组件当前的状态合并，然后触发所谓的调和过程（Reconciliation）。
2. 经过调和过程，React 会以相对高效的方式根据新的状态构建 React 元素树并且着手重新渲染整个 UI 界面；
3. 在 React 得到元素树之后，React 会自动计算出新的树与老树的节点差异，然后根据差异对界面进行最小化重渲染；
4. 在差异计算算法中，React 能够相对精确地知道哪些位置发生了改变以及应该如何改变，这就保证了按需更新，而不是全部重新渲染。

9、你怎么理解 `redux` 的 `state` 的？（必会）

Store 对象包含所有数据。如果想得到某个时点的数据，就要对 Store 生成快照。这种时点的数据集合，就叫做 State。当前时刻的 State，可以通过 `store.getState()` 拿到。唯一改变 state 的方法就是触发 action

10、除了在构造函数中绑定 `this`，还有其它方式吗？（必会）

你可以使用属性初始值设定项 (property initializers) 来正确绑定回调，`create-React-app` 也是默认支持的。在回调中你可以使用箭头函数，但问题是每次组件渲染时都会创建一个新的回调

11、(在构造函数中)调用 `super(props)` 的目的是什么？（必会）

在 `super()` 被调用之前，子类是不能使用 `this` 的，在 ES2015 中，子类必须在 `constructor` 中调用 `super()`。传递 `props` 给 `super()` 的原因则是便于(在子类中)能在 `constructor` 访问 `this.props`

12、简述 flux 思想？（必会）

1. 用户访问 View
2. View 发出用户的 Action
3. Dispatcher 收到 Action, 要求 Store 进行相应的更新
4. Store 更新后, 发出一个"change"事件
5. View 收到"change"事件后, 更新页面

13、事件在 React 中的处理方式？（必会）

React 元素的事件处理和 DOM 元素类似。但是有一点语法上的不同

1. React 事件绑定属性的命名采用驼峰式写法, 而不是小写。
2. 如果采用 JSX 的语法你需要传入一个函数作为事件处理函数, 而不是一个字符串 (DOM 元素的写法)

14、列出 Redux 的核心方法？（必会）

1. Action - 这是一个用来描述发生了什么事情的对象
2. Reducer - 这是一个确定状态将如何变化的地方
3. Store - 整个程序的状态/对象树保存在 Store 中
4. View - 只显示 Store 提供的数据

15、(组件的)状态(state)和属性(props)之间有何不同？（必会）

State 是一种数据结构, 用于组件挂载时所需数据的默认值。State 可能会随着时间的推移而发生突变, 但多数时候是作为用户事件行为的结果。Props (properties 的简写) 则是组件的配置。props 由父组件传递给子组件, 并且就子组件而言, props 是不可变的, 组件不能改变自身的 props, 但是可以把其子组件的 props 放在一起(统一管理)

16、何为受控组件(controlled component)？（必会）

在 HTML 中, 类似 `<input>`, `<textarea>` 和 `<select>` 这样的表单元素会维护自身的状态, 并基于用户的输入来更新, 当用户提交表单时, 前面提到的元素的值将随表单一起被发送, 但在 React 中会有些不同, 包含表单元素的组件将会在 state 中追踪输入的值, 并且每次调用回调函数时, 如 `onChange` 会更新 state, 重新渲染组件, 一个输入表单元素, 它的值通过 React 的这种方式来控制, 这样的元素就被称为"受控元素"

18、何为高阶组件(higherordercomponent)？（必会）

高阶组件是一个以组件为参数并返回一个新组件的函数。HOC 运行你重用代码、逻辑和引导抽象，最常见的可能是 Redux 的 connect 函数，除了简单分享工具库和简单的组合，HOC 最好的方式是共享 React 组件之间的行为，如果你发现你在不同的地方写了大量代码来做同一件事时，就应该考虑将代码重构为可重用的 HOC

19、React 中组件如何进行数据传值？（必会）

1. 父级传递子级：把数据挂载子组件的属性上，子组件通过 `this.props` 来接收父组件的数据。
2. 子级传递父级：父级需要定义一个修改数据的方法，把修改数据的方法传给子组件，当子组件需要修改父级数据时，调用父级传过来的修改方法
3. 兄弟组件传递：属于同一个父级，父组件分别和这两个组件传递。比如子组件 A 操作执行父组件方法，父组件进行修改，然后把信息传给子组件 B
4. Context 跨组件传递数据：顶级组件向最里面组件进行传值

20、解释 Reducer 的作用（必会）

Reducers 是纯函数，它规定应用程序的状态怎样因响应 ACTION 而改变。Reducers 通过接受先前的状态和 action 来工作，然后它返回一个新的状态。它根据操作的类型确定需要执行哪种更新，然后返回新的值。如果不需要完成任务，它会返回原来的状态

21、redux 有什么缺点（必会）

一个组件所需要的数据，必须由父组件传过来，而不能像 flux 中直接从 store 取；当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新 render，可能会有效率影响，或者需要写复杂的 `shouldComponentUpdate` 进行判断

22、了解 redux 么，说一下 redux（必会）

redux 是一个应用数据流框架，主要是解决了组件间状态共享的问题，原理是集中式管理，主要有三个核心方法，action, store, reducer

三大原则：

1. 唯一数据源(整个应用的 state 被储存在一棵 object tree 中, 并且这个 object tree 只存在于唯一一个 store 中)
2. reducer 必须是纯函数(输入必须对应着唯一的输出)
3. State 是只读的, 想要更改必须经过派发 action

23、Redux 的工作流程

使用通过 reducer 创建出来的 Store 发起一个 Action, reducer 会执行相应的更新 state 的方法, 当 state 更新之后, view 会根据 state 做出相应的变化

1. 提供 getState() 获取到 state
2. 通过 dispatch(action) 发起 action 更新 state
3. 通过 subscribe() 注册监听器

24、vue 和 React 的区别（必会）

监听数据变化的实现原理不同; 数据流的不同; React 组合不同功能的方式是通过 HoC(高阶组件), Vue 组合不同功能的方式是通过 mixin; 组件通信的不同; 模板渲染方式的不同; 渲染过程不同等

25、React 生命周期函数有哪些？（必会）

1. Mounting 挂载阶段

constructor()、componentWillMount() 组件挂载到页面之前
render() 创建虚拟 DOM, 进行 diff 运算, 更新 DOM 树。
不可进行 setState()、componentDidMount() 组件挂载到页面之后, 可以在此请求数据

2. Updateing 更新阶段

componentWillReceiveProps() 父级数据发生变化
shouldComponentUpdate()、性能优化

3. Unmounting 卸载阶段

componentWillUnmount 组件卸载和销毁之前立刻停用、可以在此销毁定时器, 取消网络请求, 消除创建的相关 DOM 节点等

26、运行阶段生命周期调用顺序？（必会）

运行中阶段可以使用的函数：

- **componentWillReceiveProps**：父组件修改属性触发，**可以**修改新属性、修改状态
- **shouldComponentUpdate**：返回false会**阻止**render调用
- **componentWillUpdate**：**不能**修改属性和状态
- **render**：只能访问this.props和this.state，只有一个顶层组件，**不允许**修改状态和DOM输出
- **componentDidUpdate**：**可以**修改DOM

27、React 中 component 和 PureComponent 区别是什么？（必会）

1. Component 是 ReactApp 的基本构建的单位，也是 React 中的基本代码复用单位
2. PureComponent 与 Component 在除了其 shouldComponentUpdate 方法的实现之外几乎完全相同。
3. PureComponent 已经替我们实现了 shouldComponentUpdate 方法。对于 PureComponent 而言，当其 props 或者 state 改变之时，新旧 props 与 state 将进行浅对比(shallow comparison)。另一方面，Component 默认的情况下其 shouldComponentUpdate 方法并不进行新旧 props 与 state 的对比

28、什么是无状态组件，与有状态组件的区别？（必会）

无状态组件主要用来定义模板，接收来自父组件 props 传递过来的数据，使用 {props.xxx} 的表达式把 props 塞到模板里面

有状态组件主要用来定义交互逻辑和业务数据，使用 {this.state.xxx} 的表达式把业务数据挂载到容器组件的实例上（有状态组件也可以叫做容器组件，无状态组件也可以叫做展示组件），然后传递 props 到展示组件，展示组件接收到 props，把 props 塞到模板里面

29、调用 render 时，DOM 一定会更新吗，为什么？（必会）

不一定更新，React 组件中存在两类 DOM，render 函数被调用后，React 会根据 props 或者 state 重新创建一棵 virtual DOM 树，虽然每一次调用都重新创建，但因为创建是发生在内存中，所以很快不影响性能。而 virtual dom 的更新并不意味着真实 DOM 的更新，React 采用 diff 算法将 virtual DOM 和真实 DOM 进行比较，找出需要更新的最小的部分，这时 Real DOM 才可能发

生修改,所以每次 state 的更改都会使得 render 函数被调用,但是页面 DOM 不一定发生修改

30、在哪些生命周期中可以修改组件的 state? (必会)

`componentDidMount` 和 `componentDidUpdate`

`constructor`、`componentWillMount` 中 `setState` 会发生错误: `setState` 只能在 `mounted` 或 `mounting` 组件中执行。`componentWillUpdate` 中 `setState` 会导致死循环

31、`connect()` 前两个参数是什么? (必会)

1. `mapStateToProps(state, ownProps)`

允许我们将 store 中的数据作为 props 绑定到组件中,只要 store 更新了就会调用 `mapStateToProps` 方法, `mapStateToProps` 返回的结果必须是 object 对象,该对象中的值将会更新到组件中

2. `mapDispatchToProps(dispatch, [ownProps])`

允许我们将 action 作为 props 绑定到组件中,如果不传这个参数 `redux` 会把 `dispatch` 作为属性注入给组件,可以手动当做 `store.dispatch` 使用 `mapDispatchToProps` 希望你返回包含对应 action 的 object 对象

32、React-router 的原理 (高薪常问)

1. `BrowserRouter` 或 `hashRouter` 用来渲染 Router 所代表的组件
2. `Route` 用来匹配组件路径并且筛选需要渲染的组件
3. `Switch` 用来筛选需要渲染的唯一组件
4. `Link` 直接渲染某个页面组件
5. `Redirect` 类似于 `Link`, 在没有 `Route` 匹配成功时触发

33、React 的 diff 原理 (高薪常问)

diff (翻译差异): 计算一棵树形结构转换成另一棵树形结构的最少操作

1. 把树形结构按照层级分解, 只比较同级元素
2. 给列表结构的每个单元添加唯一的 key 属性, 方便比较
3. React 只会匹配相同 class 的 component (这里面的 class 指的是组件的名字)
4. 合并操作, 调用 component 的 `setState` 方法的时候, React 将其标记为 `dirty`. 到每一个事件循环结束, React 检查所有标记 `dirty` 的 component 重新绘制
5. 选择性子树渲染。开发人员可以重写 `shouldComponentUpdate` 提高 diff 的性能

34、为什么建议传递给 setState 的参数是一个 callback 而不是一个对象（高薪常问）

因为 this.props 和 this.state 的更新可能是异步的，不能依赖它们的值去计算下一个 state。

35、redux 中间件原理（高薪常问）

redux 的中间件就是对 store 的 dispatch 做了个升级，升级之后 dispatch 就可以对象和函数都可以接收，这个时候当调用 dispatch 方法给 dispatch 方法传递的参数是一个对象的话，那么 dispatch 就会把这个对象直接传递给 store，跟之前我们写 dispatch 传递给它一个对象没什么区别，但是如果传递给 dispatch 方法是一个函数的话，这个时候 dispatch 已经升级了，它就不会把这个函数直接传递给 store，它会先让这个函数执行，执行完了之后需要调用 store 的时候再去调用 store。所以 dispatch 在这里会根据参数的不同执行不同的事情

36、React 性能优化的方案（高薪常问）

减少 render 方法的调用；避免使用状态提升来共享 state, 此时应该使用 redux 解决方案；保持稳定的 dom 结构，尽量避免 dom 节点跨层级移动操作；使用 css 来隐藏节点，而不是真的移除或添加 DOM 节点等

37、为什么虚拟 DOM 会提高性能?说下他的原理（高薪常问）

虚拟 dom 相当于在 js 和真实 dom 中间加了一个缓存，利用 dom 的 diff 算法避免了没有必要的 dom 操作，从而提高性能

VirtualDOM 工作过程有三个简单的步骤：

- 1、每当底层数据发生改变时，整个 UI 都将在 VirtualDOM 描述中重新渲染
- 2、然后计算之前 DOM 表示与新表示的之间的差异
- 3、完成计算后，将只用实际更改的内容更新 realDOM

38、setState 何时同步何时异步？（高薪常问）

1. setState 只在合成事件（React 为了解决跨平台，兼容性问题，自己封装了一套事件机制，代理了原生的事件，像在 jsx 中常见的 onClick、onChange 这些都是合成事件）和钩子函数（生命周期）中是“异步”的，在原生事件和 setTimeout 中都是同步的
2. setState 的“异步”并不是说内部由异步代码实现，其实本身执行的过程和代码都是同步

的，只是合成事件和钩子函数的调用顺序在更新之前，导致在合成事件和钩子函数中没法立马拿到更新后的值，形式了所谓的“异步”，当然可以通过第二个参数 `setState(partialState, callback)` 中的 `callback` 拿到更新后的结果

3. `setState` 的批量更新优化也是建立在“异步”（合成事件、钩子函数）之上的，在原生事件和 `setTimeout` 中不会批量更新，在“异步”中如果对同一个值进行多次 `setState`，`setState` 的批量更新策略会对其进行覆盖，取最后一次的执行，如果是同时 `setState` 多个不同的值，在更新时会对其进行合并批量更新

前端性能优化

1、如何进行前端性能优化？（必会）

1、减少 http 请求

减少 http 请求的方案主要有：合并 JavaScript 和 CSS 文件、合并图片 CSS Sprites、图像映射（Image Map）和使用 Data URI 来编码图片，图片较多的页面也可以使用 lazyLoad 等技术进行优化。

2、减少对 DOM 的操作

修改和访问 DOM 元素会造成页面的 Repaint（重绘）和 Reflow（重排），循环对 DOM 操作更是不推荐的行为。所以合理的使用 JavaScript 变量储存内容，考虑大量 DOM 元素中循环的性能开销，在循环结束时一次性写入。

减少对 DOM 元素的查询和修改，查询时可将其赋值给局部变量。

注：在 IE 中: hover 会降低响应速度。

3、使用 JSON 格式来进行数据交换

JSON 是一种轻量级的数据交换格式，采用完全独立于语言的文本格式，是理想的数据交换格式。同时，JSON 是 JavaScript 原生格式，这意味着在 JavaScript 中处理 JSON 数据不需要任何特殊的 API 或工具包。与 XML 序列化相比，JSON 序列化后产生的数据一般要比 XML 序列化后数据体积小。

4、高效使用 HTML 标签和 CSS 样式

HTML 是一门标记语言，使用合理的 HTML 标签前你必须了解其属性，比如 Flow Elements, Metadata Elements, Phrasing Elements。比较基础的就是要知道块级元素和内联元素、盒模型、SEO 方面的知识。

CSS 是用来渲染页面的，也是存在渲染效率的问题。CSS 选择符是从右向左进行匹配的，当页面被触发引起回流（reflow）的时候，低效的选择符依然会引发更高的开销，所以要避免低效。

5、使用 CDN 加速（内容分发网络）

其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。通过在网络各处放置节点服务器所构成的在现有的互联网基础之上的一层智能虚拟网络，CDN 系统能够实时的根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上

6、将 CSS 和 JS 放到外部文件中引用，CSS 放头，JS 放尾

- 7、精简 CSS 和 JS 文件
- 8、压缩图片和使用图片 Sprite 技术
- 9、注意控制 Cookie 大小和污染

因为 Cookie 是本地的磁盘文件，每次浏览器都会去读取相应的 Cookie，所以建议去除不必要的 Cookie，使 Cookie 体积尽量小以减少对用户响应的影响；

使用 Cookie 跨域操作时注意在适应级别的域名上设置 Cookie 以便使子域名不受其影响

Cookie 是有生命周期的，所以请注意设置合理的过期时间，合理地 Expire 时间和不要过早去清除 Cookie，都会改善用户的响应时间。

2、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。（必会）

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。

兼容问题（必会）

注意：兼容问题不局限于以下，但是面试时候要能结合实际开发或者项目说出几个 pc 端以及移动端方面的兼容问题

1、图片加 a 标签在 IE9 中会有边框（必会）

解决方案：`img{border:none;}`

2、rgba 不支持 IE8（必会）

解决方案:可以用 opacity

eg:opacity:0.7;/*FF chrome safari opera*/

filter:alpha(opacity:70);/*用了 ie 滤镜,可以兼容 ie*/

但是,需要注意的是,opacity 会影响里面元素的透明度

3、display:inline-blockie6/7 不支持（必会）

解决方案:Display:inline-block;Display:inline;

4、不同浏览器的标签默认的外补丁和内补丁不同（必会）

问题：随便写几个标签，不加样式控制的情况下，各自的 margin 和 padding 差异较大。

解决方案：CSS 里*{margin:0;padding:0;}

5、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（必会）

问题:IE6 中后面的一块被顶到下一行

解决方案：在 float 的标签样式控制中加入 display:inline;将其转化为行内属性

6、设置较小高度标签（一般小于 10px），在 IE6，IE7 中高度超出自己设置高度（必会）

问题：IE6、7 里这个标签的高度不受控制，超出自己设置的高度

解决方案：给超出高度的标签设置 overflow:hidden;或者设置行高 line-height 小于你设置的高度。

注：这种情况一般出现在我们设置小圆角背景的标签里。出现这个问题的原因是 IE8 之前的浏览器都会给标签一个最小默认的行高的高度。即使你的标签是空的，这个标签的高度还是会达到默认的行高。

7、IE9 一下浏览器不能使用 opacity（必会）

解决方案：

```
opacity: 0.5;filter: alpha(opacity = 50);filter: progid:DXImageTransform.Microsoft.Alpha(style = 0, opacity = 50);
```

8、IE6 背景闪烁的问题（必会）

问题：链接、按钮用 CSSsprites 作为背景，在 ie6 下会有背景图闪烁的现象。原因是 IE6 没有将背景图缓存，每次触发 hover 的时候都会重新加载

解决：可以用 JavaScript 设置 ie6 缓存这些图片：

```
document.execCommand("BackgroundImageCache", false, true);
```

9、event 事件问题：（必会）

```
//event 事件问题
document.onclick=function(ev) { //谷歌火狐的写法，IE9 以上支持，往下不支持；
    var e=ev;
    console.log(e);
}
document.onclick=function() { //谷歌和 IE 支持，火狐不支持；
    var e=event;
    console.log(e);
}
document.onclick=function(ev) { //兼容写法；
    var e=ev||window.event;
    var mouseX=e.clientX; //鼠标 X 轴的坐标
    var mouseY=e.clientY; //鼠标 Y 轴的坐标
}
```

10、DOM 节点相关的问题（必会）

```
//DOM 节点相关，主要兼容 IE 6 7 8
function nextnode(obj) { //获取下一个兄弟节点
    if (obj.nextElementSibling) {
        return obj.nextElementSibling;
    } else {
        return obj.nextSibling;
    }
};
function prenode(obj) { //获取上一个兄弟节点
    if (obj.previousElementSibling) {
        return obj.previousElementSibling;
    } else {
        return obj.previousSibling;
    }
};
function firstnode(obj) { //获取第一个子节点
    if (obj.firstElementChild) {
        return obj.firstElementChild; //非 IE678 支持
    } else {
        return obj.firstChild; //IE678 支持
    }
};
function lastnode(obj) { //获取最后一个子节点
    if (obj.lastElementChild) {
        return obj.lastElementChild; //非 IE678 支持
    } else {
        return obj.lastChild; //IE678 支持
    }
};
```

11、设置监听事件：（必会）

```
//设置监听事件
function addEvent(obj, type, fn) { //添加事件监听，三个参数分别为 对象、事件类型、事件处理
```

函数，默认为 false

```
if (obj.addEventListener) {
    obj.addEventListener(type, fn, false); // 非 IE
} else {
    obj.attachEvent('on'+type, fn); // ie, 这里已经加上 on, 传参的时候注意不要重复加了
};
}
function removeEvent(obj, type, fn) { // 删除事件监听
    if (obj.removeEventListener) {
        obj.removeEventListener(type, fn, false); // 非 IE
    } else {
        obj.detachEvent('on'+type, fn); // ie, 这里已经加上 on, 传参的时候注意不要重复加了
    };
}
```

12、阻止事件冒泡：（必会）

```
// js 阻止事件冒泡，这里使用 click 事件为例
document.onclick=function(e){
    var e=e||window.event;
    if (e.stopPropagation) {
        e.stopPropagation(); // W3C 标准
    } else {
        e.cancelBubble=true; // IE...
    }
}
```

13、阻止默认事件：（必会）

```
// js 阻止默认事件
document.onclick=function(e){
    var e=e||window.event;
    if (e.preventDefault) {
        e.preventDefault(); // W3C 标准
    } else {
        e.returnValue=false; // IE..
    }
}
```

14、IOS 移动端 click 事件 300ms 的延迟响应（必会）

解决方案：

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

触摸事件的响应顺序为 touchstart --> touchmove --> touchend --> click, 也可以通过绑定 ontouchstart 事件，加快对事件的响应，解决 300ms 延迟问题

15、一些情况下对非可点击元素如(label, span)监听 click 事件，ios 下不会触发（必会）

解决方案：css 增加 cursor:pointer。

16、三星手机遮罩层下的 input、select、a 等元素可以被点击和 focus(点击穿透)（必会）

有两种解决方案，

1. 是通过层显示以后加入对应的 class 名控制，截断显示层下方可获取焦点元素的事件获取
2. 是通过将可获取焦点元素加入的 disabled 属性，也可以利用属性加 dom 锁定的方式(disabled 的一种变换方式)

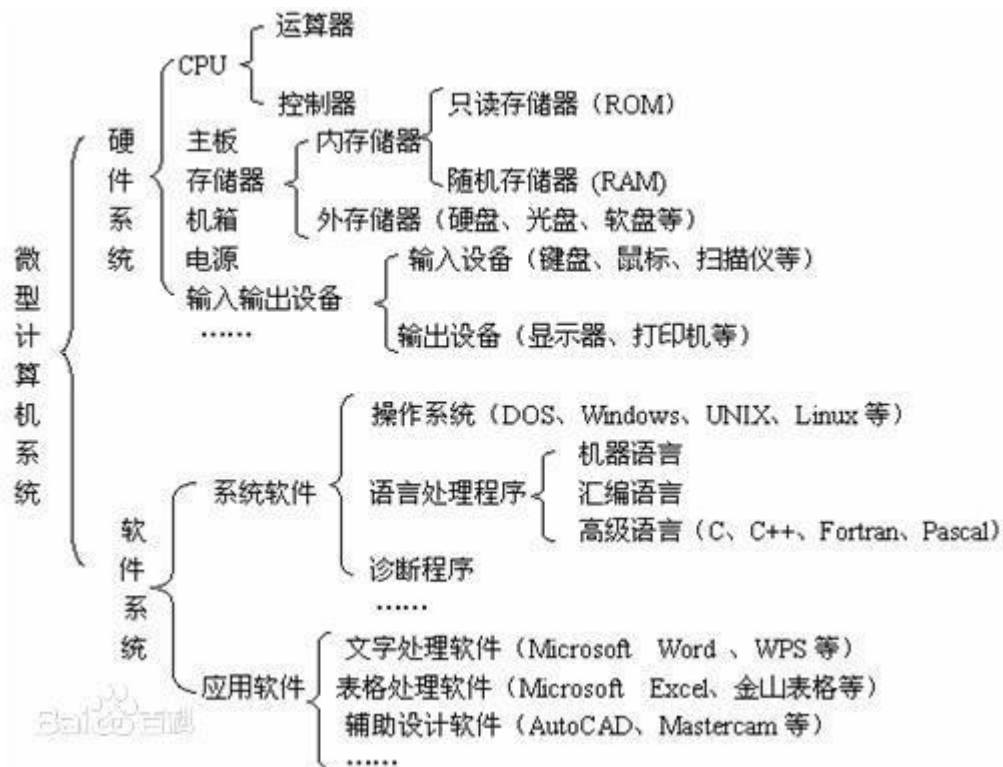
17 Input 的 placeholder 会出现文本位置偏上的情况（必会）

input 的 placeholder 会出现文本位置偏上的情况：PC 端设置 line-height 等于 height 能够对齐，而移动端仍然是偏上，解决是设置 line-height: normal

计算机相关术语

1、关于计算机相关术语的介绍（高薪常问）

了解计算机相关术语的目的：作为一个计算机相关专业的学生来说，大学开设的课程有：计算机基础、网页设计、计算机组成原理、数据结构、C 语言、C++、java、.net、计算机网络、高等数学、线性代数、离散数学、概率论、操作系统、软件测试、linux、汇编语言等。了解一定的计算机相关术语有助于更好的体现我们的专业性、技术性。



计算机组成图

2、http 超文本传输协议（高薪常问）

[超文本传输协议](#)（HTTP，HyperText Transfer Protocol）是[互联网](#)上应用最为广泛的一种[网络协议](#)。所有的[WWW](#)文件（即超文本文件（Hypertext），是指具有超链接功能的文件，它可以将文件中已经定义好的关键字（Keyword），经过鼠标的点取（Click），便可以得到该关键字的相关解释，这种方法使用户使用起来更感舒适。类似于早期使用的 WIN32 下的 HELP 文件。）都必须遵守这个标准。设计 HTTP 最初的目的是为了提供一种发布和接收[HTML](#)页面的方法。

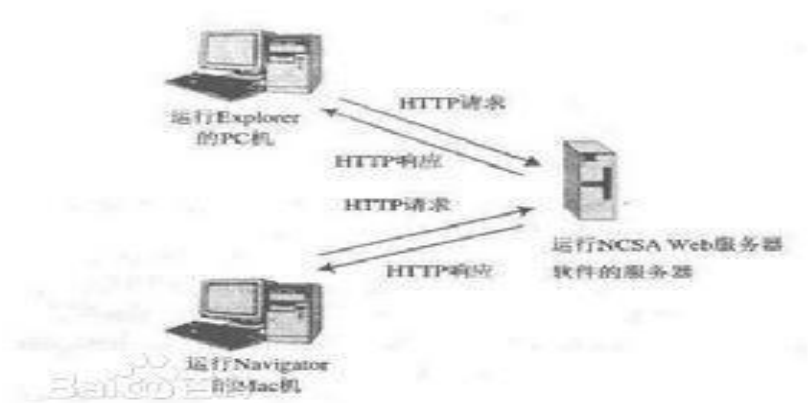
工作原理：

一次 HTTP 操作称为一个事务，其工作过程可分为四步：

- 1、 首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP 的工作就开始了。 2、 建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议[版本号](#)，后边是 MIME 信息包括请求修饰符、客户机信息和可能的内容。
- 3、 服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务器信息、实体信息和可能的内容。
- 4、 客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。

注意：如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，由显示屏输

出。对于用户来说，这些过程是由 HTTP 自己完成的，用户只要用鼠标点击，等待信息显示就可以了。



http 工作流程图

报文格式：

请求报文格式如下：

请求行 — 通用信息头 — 请求头 — 实体头 — 报文主体

应答报文格式如下：

状态行 — 通用信息头 — 响应头 — 实体头 — 报文主体

协议功能：

HTTP 协议（HyperText Transfer Protocol，超文本传输协议）是用于从 WWW 服务器传输超文本到本地浏览器的传输协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示（如文本先于图形）等。

HTTP 是客户端浏览器或其他程序与 [Web 服务器](#) 之间的应用层通信协议。在 Internet 上的 Web 服务器上存放的都是超文本信息，客户机需要通过 HTTP 协议传输所要访问的超文本信息。HTTP 包含命令和传输信息，不仅可用于 Web 访问，也可以用于其他因特网/内联网应用系统之间的通信，从而实现各类应用资源超媒体访问的集成。

我们在浏览器的地址栏里输入的网站地址叫做 URL（Uniform Resource Locator，[统一资源定位符](#)）。就像每家每户都有一个门牌地址一样，每个网页也都有一个 Internet 地址。当你在浏览器的地址框中输入一个 URL 或是单击一个[超级链接](#)时，URL 就确定了要浏览的地址。浏览器

通过超文本传输协议(HTTP)，将 Web 服务器上站点的网页代码提取出来，并翻译成漂亮的网页。

3、计算机网络的分层体系结构（高薪常问）

物理层：物理接口规范, 传输比特流, 网卡是工作在物理层的。

数据链路层：成帧, 保证帧的无误传输, MAC 地址, 形成 EHTHERNET 帧 数据链路层在不可靠的物理介质上提供可靠的传输。该层的作用包括：物理地址寻址、数据的成帧、[流量控制](#)、数据的检错、重发等。

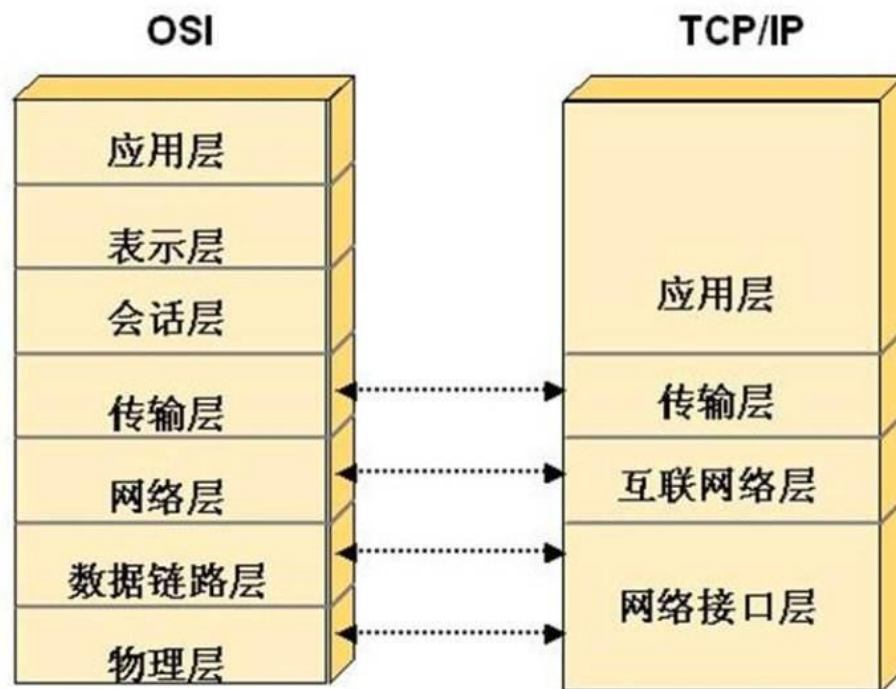
网络层：路由选择, 流量控制, IP 地址, 形成 IP 包

传输层：端口地址, 如 HTTP 对应 80 端口. TCP 和 UDP 工作于该层, 还有差错校验和流量控制。

会话层：组织两个会话进程之间的通信, 并管理数据的交换使用 ETBIOS 和 WINSOCK 协议. QQ 等软件进行通讯因该是工作在会话层的。

表示层：使得不同操作系统之间通信成为可能。

应用层：对应于各个应用软件



计算机网络的分层体系结构图

4、计算机存储器相关知识（高薪常问）

存储器：是计算机的重要组成部分。它可分为：

计算机内部的存储器（简称内存）

计算机外部的存储器（简称外存）

[内存存储器](#)从功能上可以分为：读写存储器 RAM、只读存储器 ROM 两大类

RAM 和 ROM 的区别：

黑马程序员-武汉前端学科出品 www.itheima.com

RAM: (Random Access Memory) 易挥发性[随机存取存储器](#)，高速存取，读写时间相等，且与地址无关，如计算机内存等。RAM 表示的是读写存储器，可以与任一存储单元进行读或写操作，计算机关闭电源后其内的信息将不在保存，再次开机需要重新装入，通常用来存放操作系统，各种正在运行的软件、输入和输出数据、中间结果及与外存交换信息等，我们常说的内存主要是指 RAM。

ROM: (Read Only Memory) [只读存储器](#)。断电后信息不丢失，如计算机启动用的 [BIOS 芯片](#)。存取速度很低，(较 RAM 而言) 且不能改写。由于不能改写信息，不能升级，现已很少使用。ROM 表示的是[只读存储器](#)，即：它只能读出信息，不能写入信息，计算机关闭电源后其内的信息仍旧保存，一般用它存储固定的系统软件和字库等。

5、浏览器（高薪常问）

1、浏览器相关知识介绍：

浏览器是指可以显示网页[服务器](#)或者[文件系统](#)的 [HTML 文件](#) ([标准通用标记语言](#) 的一个应用) 内容，并让用户与这些文件交互的一种[软件](#)。

它用来显示在[万维网](#)或局域网等内的文字、图像及其他信息。这些文字或图像，可以是连接其他网址的超链接，用户可迅速及轻易地浏览各种信息。大部分网页为 [HTML](#) 格式。

国内网民计算机上常见的网页浏览器有，[QQ 浏览器](#)、[Internet Explorer](#)、[Firefox](#)、[Safari](#)，[Opera](#)、[Google Chrome](#)、[百度浏览器](#)、[搜狗浏览器](#)、[猎豹浏览器](#)、[360 浏览器](#)、[UC 浏览器](#)、[傲游浏览器](#)、[世界之窗浏览器](#)等，浏览器是最经常使用到的客户端程序。

移动端产品有（移动端的浏览器）：百度、搜狗、UC、腾讯

内核：

IE 内核。包括 [360 安全浏览器](#)、IE、Greenbrowser、Maxthon2、世界之窗、刚开始的搜狗浏览器。

Chrome 内核，如 Chrome 浏览器。

双核(IE 和 chrome/webkit 内核)。双核的意思是一般网页用 chrome 内核(即 webkit 或高速模式)打开，网银等指定的网页用 IE 内核打开。如 360 高速浏览器, 搜狗高速浏览器，并不是 1 个网页同时用 2 个内核处理。

Firefox。

2、浏览器的主要构成 (High Level Structure)

浏览器的主要组件包括：

- 1) 用户界面 — 包括地址栏、后退/前进按钮、书签目录等，也就是你所看到的除了用来显示你所请求页面的主窗口之外的其他部分。
- 2) 浏览器引擎 — 用来查询及操作渲染引擎的接口。
- 3) 渲染引擎 — 用来显示请求的内容，例如，如果请求内容为 html，它负责解析 html 及 css，并将解析后的结果显示出来。

- 4) 网络 — 用来完成网络调用, 例如 http 请求, 它具有平台无关的接口, 可以在不同平台上工作。
- 5) UI 后端 — 用来绘制类似组合选择框及对话框等基本组件, 具有不特定于某个平台的通用接口, 底层使用操作系统的用户接口。
- 6) JS 解释器 — 用来解释执行 JS 代码。
6. 数据存储 — 属于持久层, 浏览器需要在硬盘中保存类似 cookie 的各种数据, HTML5 定义了 web database 技术, 这是一种轻量级完整的客户端存储技术

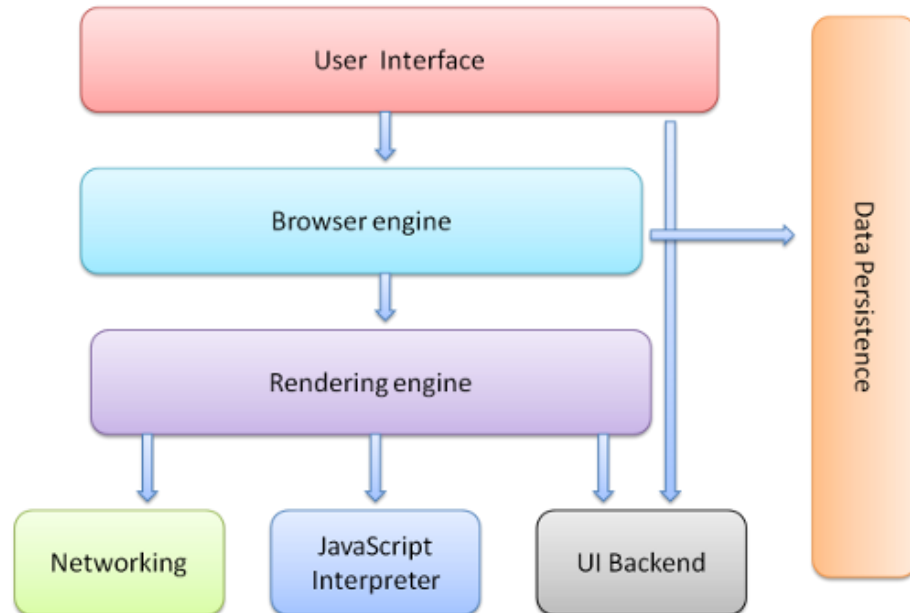


图 1: 浏览器主要组件

需要注意的是, 不同于大部分浏览器, Chrome 为每个 Tab 分配了各自的渲染引擎实例, 每个 Tab 就是一个独立的进程。

对于构成浏览器的这些组件, 后面会逐一详细讨论。

3、渲染引擎 (The rendering engine)

渲染引擎的职责就是渲染, 即在浏览器窗口中显示所请求的内容。

默认情况下, 渲染引擎可以显示 html、xml 文档及图片, 它也可以借助插件 (一种浏览器扩展) 显示其他类型数据, 例如使用 PDF 阅读器插件, 可以显示 PDF 格式, 将由专门一章讲解插件及扩展, 这里只讨论渲染引擎最主要的用途——显示应用了 CSS 之后的 html 及图片

详情参照: http://blog.csdn.net/finish_dream/article/details/52304276

6、服务器 (高薪常问)

1、介绍

服务器, 也称伺服器, 是提供计算服务的设备。由于服务器需要响应服务请求, 并进行处理, 因此一般来说服务器应具备承担服务并且保障服务的能力。

服务器的构成包括处理器、硬盘、内存、系统总线等, 和通用的计算机架构类似, 但是由于需要提供高可靠的服务, 因此在处理能力、稳定性、可靠性、安全性、可扩展性、可管理性等方面要求较高。

在网络环境下, 根据服务器提供的服务类型不同, 分为文件服务器, [数据库服务器](#), 应用程

序服务器，WEB 服务器等。

2、分类：

按照体系架构来区分，服务器主要分为两类：非 x86 服务器和 x86 服务器

按应用层次划分：1、入门级服务器 2、工作组服务器 3、部门级服务器 4、企业级服务器 5、典型服务器应用（办公 [OA](#) 服务器、ERP 服务器、[WEB 服务器](#)、[数据库](#) 服务器、财务服务器、打印服务器、集群服务器、视频监控服务器、[游戏](#) 服务器、论坛服务器）

3、特性：1、可扩展性 2、易使用性 3、可用性 4、易管理性

4、外形：1、机架式 2、刀片 3、塔式 4、机柜式

5、操作系统：服务器平台的操作系统。Unix [操作系统](#)，由于是 Unix 的后代，大多都有较好的作服务器平台的功能。

7、经典编程算法（高薪常问）

- 1、快速排序算法
- 2、堆排序算法
- 3、归并排序
- 4、二分查找算法
- 5、BFPRT（线性查找算法）
- 6、DFS（深度优先搜索）
- 7、BFS（广度优先搜索）
- 8、Floyd-Warshall all-pairs 最短路径算法

8、经典排序算法（高薪常问）

- 1、插入排序—直接插入排序 (Straight Insertion Sort)
- 2、插入排序—希尔排序 (Shell's Sort)
- 3、选择排序—简单选择排序 (Simple Selection Sort)
- 4、选择排序—堆排序 (Heap Sort)
- 5、交换排序—冒泡排序 (Bubble Sort)
- 6、快速排序 (Quick Sort)
- 7、归并排序 (Merge Sort)
- 8、桶排序/基数排序 (Radix Sort)
- 9、堆排序
- 10、计数排序

9、黑盒、白盒、灰盒测试（高薪常问）

白盒测试：

白盒测试也称为结构测试或逻辑驱动测试，是针对被测单元内部是如何进行工作的测试。它

根据程序的控制结构设计测试用例，主要用于软件或程序验证。白盒测试法检查程序内部逻辑结构，对所有的逻辑路径进行测试，是一种穷举路径的测试方法，但即使每条路径都测试过了，但仍然有可能存在错误。因为：穷举路径测试无法检查出程序本身是否违反了设计规范，即程序是否是一个错误的程序；穷举路径测试不可能检查出程序因为遗漏路径而出错；穷举路径测试发现不了与数据相关的错误。

白盒测试需要遵循的原则有：1. 保证一个模块中的所有独立路径至少被测试一次；2. 所有逻辑值均需要测试真（true）和假（false）；两种情况；3. 检查程序的内部数据结构，保证其结构的有效性；4. 在上下边界及可操作范围内运行所有循环。

白盒测试方法有：

静态测试、动态测试、单元测试、代码检查、同行评审、技术评审

黑盒测试：

黑盒测试又称为功能测试、数据驱动测试或基于规格说明书的测试，是一种从用户观点出发的测试。测试人员一般把被测程序当作一个黑盒子。

黑盒测试主要测到的错误类型有：不正确或遗漏的功能；接口、界面错误；性能错误；数据结构或外部数据访问错误；初始化或终止条件错误等等。

常用的黑盒测试方法有：等价类划分法；边界值分析法；因果图法；场景法；正交实验设计法；判定表驱动分析法；错误推测法；功能图分析法。

两者之间的区别：

黑盒测试着重测试软件功能。

黑盒测试并不能取代白盒测试，它是与白盒测试互补的测试方法，它很可能发现白盒测试不易发现的其他类型错误。

灰盒测试（Gray-Box Testing）

灰盒测试更像是白盒测试和黑盒测试的混合测试，现阶段对灰盒测试没有更明确的定义，但更多的时候，我们的测试做的就是灰盒测试，即既会做黑盒测试又会做白盒测试。

10、二叉排序树（高薪常问）

定义：二叉排序树或者是一棵空树，或者是具有下列性质的二叉树：

- （1）若左子树不空，则左子树上所有结点的值均小于或等于它的根结点的值；
- （2）若右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值；
- （3）左、右子树也分别为二叉排序树；

详见：http://m.blog.csdn.net/yxb_yingu/article/details/51336197