

前言

历时半年，我们整理了这份市面上最全面的前端校招面试题精编解析大全，

包含了腾讯、字节跳动、百度、阿里、滴滴、美团、58、拼多多、360、新浪、搜狐等一线互联网公司面试被问到的题目。熟悉本文中列出的知识点会大大增加通过前两轮技术面试的几率。

特意推荐给前端初级开发的朋友们。

如何使用它？

可以通过目录索引直接翻看需要的知识点，查漏补缺。

目录

前言	1
第一章 HTML.....	13
1. 浏览器页面有哪三层构成，分别是什么，作用是什么?.....	13
2. HTML5 的优点与缺点?	13
3. Doctype 作用？严格模式与混杂模式如何区分？它们有何意义?.....	14
4. HTML5 有哪些新特性、移除了哪些元素?	15
5. 你做的网页在哪些浏览器测试过，这些浏览器的内核分别是什么?.....	17
6. 每个 HTML 文件里开头都有个很重要的东西，Doctype，知道这是干什么的吗?	17
7. 说说你对 HTML5 认识？（是什么，为什么）	17
8. 对 WEB 标准以及 W3C 的理解与认识?.....	18
9. HTML5 行内元素有哪些，块级元素有哪些，空元素有哪些?.....	18
10. 什么是 WebGL，它有什么优点?.....	21
11. 请你描述一下 cookies，sessionStorage 和 localStorage 的区别?.....	22
12. 说说你对 HTML 语义化的理解?.....	23
13. link 和@import 的区别?.....	25
14. 说说你对 SVG 理解?.....	26
15. HTML 全局属性(global attribute)有哪些?.....	27
16. 说说超链接 target 属性的取值和作用?	28

《前端校招面试题精编解析大全》

17. `data-` 属性的作用是什么？	28
18. 介绍一下你对浏览器内核的理解？	29
19. 常见的浏览器内核有哪些？	29
20. iframe 有那些缺点？	29
21. Label 的作用是什么，是怎么用的？	30
22. 如何实现浏览器内多个标签页之间的通信？	30
23. 如何在页面上实现一个圆形的可点击区域？	31
24. title 与 h3 的区别、b 与 strong 的区别、i 与 em 的区别？	31
25. 实现不使用 border 画出 1px 高的线，在不同浏览器的标准模式与怪异模式下都能保持一致的效果？	31
26. HTML5 标签的作用？(用途)	32
27. 简述一下 src 与 href 的区别？	32
28. 谈谈你对 canvas 的理解？	33
29. WebSocket 与消息推送？	33
30. mg 的 title 和 alt 有什么区别？	35
31. 表单的基本组成部分有哪些，表单的主要用途是什么？	35
32. 表单提交中 Get 和 Post 方式的区别？	35
33. HTML5 有哪些新增的表单元素？	36
34. HTML5 废弃了哪些 HTML4 标签？	36
35. HTML5 标准提供了哪些新的 API？	37
36. HTML5 存储类型有什么区别？	37
37. HTML5 应用程序缓存和浏览器缓存有什么区别？	37
38. HTML5 Canvas 元素有什么用？	38
39. 除了 audio 和 video，HTML5 还有哪些媒体标签？	38
40. HTML5 中如何嵌入视频？	39
41. HTML5 中如何嵌入音频？	40
42. 新的 HTML5 文档类型和字符集是？	40
第二章 CSS.....	40
1. 解释一下 CSS 的盒子模型？	40
2. 请你说说 CSS 选择器的类型有哪些，并举几个例子说明其用法？	41
3. 请你说说 CSS 有什么特殊性？（优先级、计算特殊值）	45
4. 要动态改变层中内容可以使用的方法？	47
5. 常见浏览器兼容性问题与解决方案？	47
6. 列出 display 的值并说明他们的作用？	50
7. 如何居中 div，如何居中一个浮动元素？	51
8. CSS 中 link 和@import 的区别是？	51
9. 请列举几种清除浮动的方法(至少两种)？	52
10. block, inline 和 inline-block 细节对比？	53
11. 什么叫优雅降级和渐进增强？	54
12. 说说浮动元素会引起的问题和你的解决办法	55

《前端校招面试题精编解析大全》

13. 你有哪些性能优化的方法?	56
14. 为什么要初始化 CSS 样式?	57
15. 解释下浮动和它的工作原理? 清除浮动的技巧?	58
16. CSS 样式表根据所在网页的位置, 可分为哪几种样式表?	58
17. 谈谈你对 CSS 中刻度的认识?	58
18. 请你说说 em 与 rem 的区别?	59
19. 请你说说 box-sizing 属性的用法?	60
20. 浏览器标准模式和怪异模式之间的区别是什么?	61
21. 怪异 Quirks 模式是什么, 它和标准 Standards 模式有什么区别?	61
22. 说说你对边距折叠的理解?	62
23. 内联与块级标签有何区别?	63
24. 说说隐藏元素的方式有哪些?	63
25. 为什么重置浏览器默认样式, 如何重置默认浏览器样式?	63
26. 谈谈你对 BFC 与 IFC 的理解? (是什么, 如何产生, 作用)	65
27. 说说你对页面中使用定位(position)的理解?	67
28. 如何解决多个元素重叠问题?	68
29. 页面布局的方式有哪些?	69
30. overflow :hidden 是否形成新的块级格式化上下文?	73
第三章 前端基础.....	73
1. 说一下 http 和 https.....	73
2. tcp 三次握手, 一句话概括.....	75
3. TCP 和 UDP 的区别.....	76
4. WebSocket 的实现和应用.....	76
5. HTTP 请求的方式, HEAD 方式.....	78
6. 一个图片 url 访问后直接下载怎样实现?	78
7. 说一下 web Quality (无障碍)	78
8. 几个很实用的 BOM 属性对象方法?	79
9. 说一下 HTML5 drag api.....	80
10. 说一下 http2.0.....	81
11. 补充 400 和 401、403 状态码.....	81
12. fetch 发送 2 次请求的原因.....	82
13. Cookie、sessionStorage、localStorage 的区别.....	82
14. 说一下 web worker.....	84
15. 对 HTML 语义化标签的理解.....	84
16. iframe 是什么? 有什么缺点?	84
17. Doctype 作用? 严格模式与混杂模式如何区分? 它们有何意义?	85
18. Cookie 如何防范 XSS 攻击.....	85
19. Cookie 和 session 的区别.....	85
20. 一句话概括 RESTFUL.....	85
21. 讲讲 viewport 和移动端布局.....	86

《前端校招面试题精编解析大全》

22. click 在 ios 上有 300ms 延迟, 原因及如何解决?	100
23. addEventListener 参数.....	100
24. cookie sessionStorage localStorage 区别.....	100
25. cookie session 区别.....	101
26. 介绍知道的 http 返回的状态码.....	101
27. http 常用请求头.....	104
28. 强, 协商缓存.....	107
29. HTTP 状态码说说你知道的.....	107
30. 讲讲 304.....	109
31. 强缓存、协商缓存什么时候用哪个.....	109
32. 前端优化.....	110
33. GET 和 POST 的区别.....	110
34. 301 和 302 的区别.....	111
35. 如何画一个三角形.....	111
36. 状态码 304 和 200.....	112
37. 说一下浏览器缓存.....	112
38. HTML5 新增的元素.....	113
39. 在地址栏里输入一个 URL, 到这个页面呈现出来, 中间会发生什么?	113
40. cookie 和 session 的区别, localStorage 和 sessionStorage 的区别.....	114
41. 常见的 HTTP 的头部.....	116
42. HTTP2.0 的特性.....	116
43. cache-control 的值有哪些.....	117
44. 浏览器在生成页面的时候, 会生成那两颗树?	117
45. csrf 和 xss 的网络攻击及防范.....	117
46. 怎么看网站的性能如何.....	118
47. 介绍 HTTP 协议(特征).....	118
48. 说一下对 Cookie 和 Session 的认知, Cookie 有哪些限制?	119
49. 描述一下 XSS 和 CRSF 攻击? 防御方法?	119
50. 知道 304 吗, 什么时候用 304?	120
51. 具体有哪些请求头是跟缓存相关的.....	120
52. cookie 和 session 的区别.....	121
53. cookie 有哪些字段可以设置.....	121
54. cookie 有哪些编码方式?	122
55. 既然你看过图解 http, 那你回答下 200 和 304 的区别.....	122
56. 除了 cookie, 还有什么存储方式. 说说 cookie 和 localStorage 的区别.....	123
57. 浏览器输入网址到页面渲染全过程.....	123
58. HTML5 和 CSS3 用的多吗? 你了解它们的新属性吗? 有在项目中用过吗?	124
59. http 常见的请求方法.....	124
60. get 和 post 的区别.....	125
61. 说说 302, 301, 304 的状态码.....	125

《前端校招面试题精编解析大全》

62. web 性能优化.....	126
63. 浏览器缓存机制.....	126
64. post 和 get 区别.....	126
65. 说一下 css 盒模型.....	127
66. 画一条 0.5px 的线.....	127
67. link 标签和 import 标签的区别.....	128
68. transition 和 animation 的区别.....	128
69. Flex 布局.....	128
70. BFC（块级格式化上下文，用于清楚浮动，防止 margin 重叠等）.....	130
71. 垂直居中的方法.....	130
72. 关于 js 动画和 css3 动画的差异性.....	133
73. 说一下块元素和行元素.....	134
74. 多行元素的文本省略号.....	134
75. visibility=hidden, opacity=0, display:none.....	134
76. 双边距重叠问题（外边距折叠）.....	134
77. position 属性.....	135
78. 浮动清除.....	136
79. css3 新特性.....	137
80. CSS 选择器有哪些，优先级呢.....	137
81. 怎么样让一个元素消失.....	138
82. 介绍一下盒模型.....	138
83. css 动画如何实现.....	139
84. 如何实现图片在某个容器中居中的？.....	139
85. 如何实现元素的垂直居中.....	139
86. CSS3 中对溢出的处理.....	140
87. float 的元素，display 是什么.....	140
88. 隐藏页面中某个元素的方法.....	140
89. 三栏布局的实现方式，尽可能多写，浮动布局时，三个 div 的生成顺序有没有影响.....	140
90. 什么是 BFC.....	142
91. calc 属性.....	142
92. 有一个 width300, height300, 怎么实现在屏幕上垂直水平居中.....	142
93. display: table 和本身的 table 有什么区别.....	143
94. position 属性的值有哪些及其区别.....	143
95. z-index 的定位方法.....	144
96. CSS 盒模型.....	144
97. 如果想要改变一个 DOM 元素的字体颜色，不在它本身上进行操作？.....	145
98. 对 CSS 的新属性有了解过的吗？.....	145
99. 用的最多的 css 属性是啥？.....	146
100. line-height 和 height 的区别.....	146
101. 设置一个元素的背景颜色，背景颜色会填充哪些区域？.....	146

《前端校招面试题精编解析大全》

102. 知道属性选择器和伪类选择器的优先级吗.....	146
103. inline-block、inline 和 block 的区别；为什么 img 是 inline 还可以设置宽高...	147
104. 用 css 实现一个硬币旋转的效果.....	147
105. 了解重绘和重排吗，知道怎么去减少重绘和重排吗，让文档脱离文档流有哪些方法	148
106. CSS 画正方体，三角形.....	149
107. overflow 的原理.....	152
108. 清除浮动的方法.....	152
109. box-sizing 的语法和基本用处.....	153
110. 使元素消失的方法有哪些？	153
111. 两个嵌套的 div，position 都是 absolute，子 div 设置 top 属性，那么这个 top 是相对于父元素的哪个位置定位的.....	153
112. 说说盒子模型.....	154
113. display.....	154
114. 怎么隐藏一个元素.....	154
115. display:none 和 visibilty:hidden 的区别.....	155
116. 相对布局和绝对布局，position:relative 和 absolute。	155
117. flex 布局.....	155
118. block、inline、inline-block 的区别.....	156
119. css 的常用选择器.....	157
120. css 布局.....	157
121. css 定位.....	158
122. relative 定位规则.....	160
123. 垂直居中.....	160
124. css 预处理器有什么.....	160
125. get 请求传参长度的误区.....	160
126. 补充 get 和 post 请求在缓存方面的区别.....	161
127. 说一下闭包.....	161
128. 说一下类的创建和继承.....	162
129. 如何解决异步回调地狱.....	165
130. 说说前端中的事件流.....	165
131. 如何让事件先冒泡后捕获.....	166
132. 说一下事件委托.....	166
133. 说一下图片的懒加载和预加载.....	167
134. mouseover 和 mouseenter 的区别.....	167
135. js 的 new 操作符做了哪些事情.....	167
136. 改变函数内部 this 指针的指向函数（bind，apply，call 的区别）	167
137. js 的各种位置，比如 clientHeight,scrollHeight,offsetHeight ,以及 scrollTop,offsetTop,clientTop 的区别？	168
138. js 拖拽功能的实现.....	168
139. 异步加载 js 的方法.....	169

《前端校招面试题精编解析大全》

140. Ajax 解决浏览器缓存问题.....	169
141. js 的节流和防抖.....	170
142. JS 中的垃圾回收机制.....	173
143. eval 是做什么的.....	175
144. 如何理解前端模块化.....	175
145. 说一下 Commonjs、AMD 和 CMD.....	176
146. 对象深度克隆的简单实现.....	177
147. 实现一个 once 函数，传入函数参数只执行一次.....	177
148. 将原生的 ajax 封装成 promise.....	178
149. js 监听对象属性的改变.....	178
150. 如何实现一个私有变量，用 getName 方法可以访问，不能直接访问.....	179
151. ==和===、以及 Object.is 的区别.....	180
152. setTimeout、setInterval 和 requestAnimationFrame 之间的区别.....	180
153. 实现一个两列等高布局，讲讲思路.....	181
154. 自己实现一个 bind 函数.....	181
155. 用 setTimeout 来实现 setInterval.....	182
156. 代码的执行顺序.....	184
157. 如何实现 sleep 的效果（es5 或者 es6）.....	184
158. 简单的实现一个 promise.....	186
159. Function.prototype.getPrototypeOf 是什么？.....	200
160. 实现 js 中所有对象的深度克隆（包装对象，Date 对象，正则对象）.....	201
161. 简单实现 Node 的 Events 模块.....	204
162. 箭头函数中 this 指向举例.....	206
163. js 判断类型.....	207
164. 数组常用方法.....	207
165. 数组去重.....	207
166. 闭包 有什么用.....	207
167. 事件代理在捕获阶段的实际应用.....	208
168. 去除字符串首尾空格.....	208
169. 性能优化.....	209
170. 能来讲讲 JS 的语言特性吗.....	209
171. 如何判断一个数组.....	209
172. 你说到 typeof，能不能加一个限制条件达到判断条件.....	210
173. JS 实现跨域.....	210
174. Js 基本数据类型.....	210
175. js 深度拷贝一个元素的具体实现.....	211
176. 之前说了 ES6set 可以数组去重，是否还有数组去重的.....	211
177. 跨域的原理.....	212
178. 不同数据类型的值的比较，是怎么转换的，有什么规则.....	212
179. null == undefined 为什么.....	213

《前端校招面试题精编解析大全》

180. this 的指向 哪几种.....	213
181. 暂停死区.....	214
182. AngularJS 双向绑定原理.....	214
183. 写一个深度拷贝.....	215
183. 简历中提到了 requestAnimationFrame, 请问是怎么使用的.....	216
184. 有一个游戏叫做 Flappy Bird, 就是一只小鸟在飞, 前面是无尽的沙漠, 上下不断有钢管生成, 你要躲避钢管。然后小明在玩这个游戏时候老是卡顿甚至崩溃, 说出原因 (3-5 个) 以及解决办法 (3-5 个)	217
185. 编写代码, 满足以下条件:	218
186. 什么是按需加载.....	219
187. 说一下什么是 virtual dom.....	219
188. webpack 用来干什么的.....	220
189. ant-design 优点和缺点.....	220
190. JS 中继承实现的几种方式.....	220
191. 写一个函数, 第一秒打印 1, 第二秒打印 2.....	221
192. vue 的生命周期.....	221
193. 简单介绍一下 symbol.....	223
194. 什么是事件监听.....	223
195. 介绍一下 promise, 及其底层如何实现.....	224
196. bootstrap 清除浮动的方法.....	226
197. 说说 C++, Java, JavaScript 这三种语言的区别.....	227
198. js 原型链, 原型链的顶端是什么? Object 的原型是什么? Object 的原型的原型是什么? 在数组原型链上实现删除数组重复数据的方法.....	229
199. 什么是 js 的闭包? 有什么作用, 用闭包写个单例模式.....	234
200. promise+Generator+Async 的使用.....	235
201. 事件委托以及冒泡原理.....	241
202. 写个函数, 可以转化下划线命名到驼峰命名.....	242
203. 深浅拷贝的区别和实现.....	242
204. JS 中 string 的 startwith 和 indexof 两种方法的区别.....	244
205. js 字符串转数字的方法.....	245
206. let const var 的区别, 什么是块级作用域, 如何用 ES5 的方法实现块级作用域 (立即执行函数), ES6 呢.....	245
207. ES6 箭头函数的特性.....	245
208. setTimeout 和 Promise 的执行顺序.....	246
209. 有了解过事件模型吗, DOM0 级和 DOM2 级有什么区别, DOM 的分级是什么.....	247
210. 平时是怎么调试 JS 的.....	248
211. JS 的基本数据类型有哪些, 基本数据类型和引用数据类型的区别, NaN 是什么的缩写, JS 的作用域类型, undefined==null 返回的结果是什么, undefined 与 null 的区别在哪, 写一个函数判断变量类型.....	249
212. setTimeout (fn, 100); 100 毫秒是如何权衡的.....	252

《前端校招面试题精编解析大全》

213. JS 的垃圾回收机制.....	252
214. 写一个 newBind 函数，完成 bind 的功能.....	252
215. 怎么获得对象上的属性：比如说通过 Object.key（）.....	253
216. 简单讲一讲 ES6 的一些新特性.....	254
217. call 和 apply 是用来做什么？.....	254
218. 了解事件代理吗，这样做有什么好处.....	255
219. 给出以下代码，输出的结果是什么？原因？.....	255
220. 给两个构造函数 A 和 B，如何实现 A 继承 B？.....	255
221. 如果已经有三个 promise，A、B 和 C，想串行执行，该怎么写？.....	256
222. 知道 private 和 public 吗.....	256
223. 基础的 js.....	257
224. async 和 await 具体该怎么用？.....	257
225. 知道哪些 ES6，ES7 的语法.....	257
226. promise 和 await/async 的关系.....	257
227. js 的数据类型.....	258
228. js 加载过程阻塞，解决方法.....	258
229. js 对象类型，基本对象类型以及引用对象类型的区别.....	258
230. JavaScript 中的轮播实现原理？假如一个页面上有两个轮播，你会怎么实现？.....	259
231. 怎么实现一个计算一年中有多少周？.....	259
232. 面向对象的继承方式.....	259
233. 引用类型常见的对象.....	262
234. es6 的常用.....	262
235. class.....	262
236. call 和 apply 的区别.....	262
237. es6 的常用特性.....	262
238. 箭头函数和 function 有什么区别.....	263
239. new 操作符原理.....	263
240. bind, apply, call.....	263
241. bind 和 apply 的区别.....	263
242. promise 实现.....	264
243. assign 的深拷贝.....	266
244. 说 promise，没有 promise 怎么办.....	267
245. arguments.....	267
246. 箭头函数获取 arguments.....	267
247. Promise.....	267
248. 事件代理.....	268
249. Eventloop.....	268
第四章 前端核心.....	269
1. JSONP 的缺点.....	269
2. 跨域（jsonp，ajax）.....	269

《前端校招面试题精编解析大全》

3. 如何实现跨域.....	269
4. dom 是什么，你的理解？	270
5. 关于 dom 的 api 有什么.....	271
6. ajax 返回的状态.....	271
7. 实现一个 Ajax.....	271
8. 如何实现 ajax 请求，假如我有多个请求，我需要让这些 ajax 请求按照某种顺序一次执行，有什么办法呢？ 如何处理 ajax 跨域.....	272
9. 如何实现一个 ajax 请求？ 如果我想发出两个有顺序的 ajax 需要怎么做？	274
10. Fetch 和 Ajax 比有什么优缺点？	275
11. 移动应用和 web 应用的关系.....	275
12. 知道 PWA 吗.....	275
13. 做过移动端吗.....	276
14. 知道 touch 事件吗.....	276
第五章 前端进阶.....	277
1. 前端测试.....	277
2. 接口文档.....	278
3. webpack 和 gulp 区别（模块化与流的区别）	280
4. redux 用处.....	280
6. redux 里常用方法.....	280
6. angularJs 和 react 区别.....	281
7. vue 双向绑定原理.....	281
8. 说说 vue react angularjs jquery 的区别.....	281
9. node 的事件方法讲讲看.....	282
10. node 的特性，适合处理什么场景.....	282
11. 你有用到 Express, 讲讲 Express.....	282
12. promise 的状态有哪些.....	282
13. 数组移除第一个元素的方法有哪些？	283
第六章 移动端开发.....	283
1. 介绍一下 react.....	283
2. React 单项数据流.....	284
3. react 生命周期函数和 react 组件的生命周期.....	284
4. react 和 Vue 的原理，区别，亮点，作用.....	286
5. reactJs 的组件交流.....	291
6. 有了解过 react 的虚拟 DOM 吗，虚拟 DOM 是怎么对比的呢.....	292
7. 项目里用到了 react，为什么要选择 react，react 有哪些好处.....	292
8. 怎么获取真正的 dom.....	293
9. 选择 react 的原因.....	293
10. react 的生命周期函数.....	293
11. setState 之后的流程.....	295
12. react 高阶组件知道吗？	295

《前端校招面试题精编解析大全》

13. React 的生命周期.....	295
14. 说说自己理解的 react.....	296
15. react 的组件是通过什么去判断是否刷新的.....	297
第七章 计算机基础.....	297
1. TCP 建立连接的三次握手过程.....	297
2. cdn 原理.....	298
3. HTTP 的头部包含哪些内容。常见的请求方法（我为什么要说后面的 options, head, connect）.....	298
4. 请求方法 head 特性.....	298
5. HTTP 状态码，301 和 302 有什么具体区别，200 和 304 的区别.....	299
6. OSI 七层模型.....	300
7. TCP 和 UDP 的区别，为什么三次握手四次挥手.....	300
8. HTTP 缓存机制.....	301
9. websocket 和 ajax 的区别是什么，websocket 的应用场景有哪些.....	302
10. TCP/IP 的网络模型.....	305
11. 知道什么跨域方式吗，jsonp 具体流程是什么，如何实现原生 Jsonp 封装，优化，对于 CORS，服务器怎么判断它该不该跨域呢.....	306
12. 怎么生成 token，怎么传递.....	307
13. 操作系统进程和线程的区别.....	311
14. 什么是进程 线程.....	311
15. 线程的那些资源共享，那些资源不共享.....	311
16. 操作系统里面进程和线程的区别.....	312
17. Linux 查询进程指令，查询端口，杀进程.....	312
18. 进程间的通信方式有哪些？.....	313
19. Redis 和 mysql.....	314
第八章 算法与数据结构.....	315
1. 二叉树层序遍历.....	315
2. B 树的特性，B 树和 B+树的区别.....	317
3. 尾递归.....	318
4. 如何写一个大数阶乘？递归的方法会出现什么问题？.....	318
5. 把多维数组变成一维数组的方法.....	318
6. 知道的排序算法 说一下冒泡快排的原理.....	320
7. Heap 排序方法的原理？复杂度？.....	320
8. 几种常见的排序算法，手写.....	320
9. 数组的去重，尽可能写出多个方法.....	324
10. 如果有一个大的数组，都是整型，怎么找出最大的前 10 个数.....	327
11. 知道数据结构里面的常见的数据结构.....	327
12. 找出数组中第 k 大的数组出现多少次，比如数组【1, 2, 4, 4, 3, 5】第二大的数字是 4，出现两次，所以返回 2.....	327
13. 合并两个有序数组.....	327

《前端校招面试题精编解析大全》

14. 给一个数，去一个已经排好序的数组中寻找这个数的位置（通过快速查找，二分查找）	328
第九章 设计模式	328
1. 设计模式：单例，工厂，发布订阅	328
2. 看过一些设计模式的书？你觉得设计模式怎么样？	329
第十章 项目	329
1. 介绍一个做过的项目	329
2. 遇到的难题，怎么解决	330
3. 简单的自我介绍	330
4. 项目的同源处理，跨越相关	331
5. 遇到过什么安全问题，怎么解决的	332
6. 让你带领一个小团队完成一个项目，我会怎么做？	338
7. 前端的项目如何进行优化，移动端呢	338
8. 项目中使用了 iframe，说说 iframe 的优缺点	339
第十一章 职业发展	340
1. 介绍一下前端的学习经历	340
2. 作为一个专业的前端，你觉得应该掌握哪些知识	340
3. 什么时候接触前端	341
4. 大学学过哪些编程的课	341
6. 对未来三年职业的规划	341
7. 你一般是通过什么方式学习前端的？	341
8. 你还有什么我没问到的优势吗	342
9. 看过什么书	342
10. 比较厉害的技术	342
11. 你学前端怎么坚持下来的	345
12. 学过哪些框架？	345
13. 你理解的框架	346
第十二章 Hr 面	346
1. 自我介绍	346
2. 为什么要学习前端	347
3. 到现在为止接触过几个项目，有在哪里实习过？	347
4. 让你收获最多的项目，你做了什么？	348
5. 个人的优缺点	348
6. 读不读研	349
7. 说说你最荣耀的事	349

第一章 HTML

1. 浏览器页面有哪三层构成，分别是什么，作用是什么？

参考答案：

构成：结构层、表示层、行为层

分别是：HTML、CSS、JavaScript

作用：HTML 实现页面结构，CSS 完成页面的表现与风格，JavaScript 实现一些客户端的功能与业务。

2. HTML5 的优点与缺点？

参考答案：

优点：

- a、网络标准统一、HTML5 本身是由 W3C 推荐出来的。
- b、多设备、跨平台
- c、即时更新。
- d、提高可用性和改进用户的友好体验；

- e、有几个新的标签，这将有助于开发人员定义重要的内容；
- f、可以给站点带来更多的多媒体元素(视频和音频)；
- g、可以很好的替代 Flash 和 Silverlight；
- h、涉及到网站的抓取和索引的时候，对于 SEO 很友好；
- i、被大量应用于移动应用程序和游戏。

缺点：

- a、安全：像之前 Firefox4 的 web socket 和透明代理的实现存在严重的安全问题，同时 web storage、web socket 这样的功能很容易被黑客利用，来盗取用户的信息和资料。
- b、完善性：许多特性各浏览器的支持程度也不一样。
- c、技术门槛：HTML5 简化开发者工作的同时代表了有许多新的属性和 API 需要开发者学习，像 web worker、web socket、web storage 等新特性，后台甚至浏览器原理的知识，机遇的同时也是巨大的挑战
- d、性能：某些平台上的引擎问题导致 HTML5 性能低下。
- e、浏览器兼容性：最大缺点，IE9 以下浏览器几乎全军覆没。

3. Doctype 作用？严格模式与混杂模式如何区分？它们有何意义？

参考答案：

回答 1：

- (1) 声明位于文档中的最前面，处于标签之前。告知浏览器的解析器，用什么文档类型 规范来解析这个文档。
- (2) 严格模式的排版和 JS 运作模式是以该浏览器支持的最高标准运行。
- (3) 在混杂模式中，页面以宽松的向后兼容的方式显示。模拟老式浏览器的行为以防止站点无法工作。

(4) DOCTYPE 不存在或格式不正确会导致文档以混杂模式呈现。

回答 2:

doctype 声明指出阅读程序应该用什么规则集来解释文档中的标记。在 Web 文档的情况下，“阅读程序”通常是浏览器或者校验器这样的程序，“规则”则是 W3C 所发布的一个文档类型定义（DTD）中包含的规则。

(1) 声明位于文档中的最前面的位置，处于标签之前。此标签可告知浏览器文档使用哪种 HTML 或 XHTML 规范。该标签可声明三种 DTD 类型，分别表示严格版本、过渡版本以及基于框架的 HTML 文档。

(2) 所谓的标准模式是指，浏览器按 W3C 标准解析执行代码；怪异模式则是使用浏览器自己的方式解析执行代码，因为不同浏览器解析执行的方式不一样，所以我们称之为怪异模式。严格模式是浏览器根据 web 标准去解析页面，是一种要求严格的 DTD，不允许使用任何表现层的语法，如。严格模式的排版和 JS 运作模式是以该浏览器支持的最高标准运行混杂模式则是一种向后兼容的解析方法，说的透明点就是可以实现 IE5.5 以下版本浏览器的渲染模式。

(3) 浏览器解析时到底使用标准模式还是怪异模式，与你网页中的 DTD 声明直接相关，DTD 声明定义了标准文档的类型（标准模式解析）文档类型，会使浏览器使用相应的方式加载网页并显示，忽略 DTD 声明，将使网页进入怪异模式。

4. HTML5 有哪些新特性、移除了哪些元素？

参考答案：

Html5 新增了 27 个元素，废弃了 16 个元素，根据现有的标准规范，把 HTML5 的元素按优先级定义为结构性属性、级块性元素、行内语义性元素和交互性元素 4 大类。

结构性元素主要负责 web 上下文结构的定义：

section: 在 web 页面应用中，该元素也可以用于区域的章节描述。

header: 页面主体上的头部， `header` 元素往往在一对 `body` 元素中。

footer: 页面的底部（页脚），通常会标出网站的相关信息。

nav: 专门用于菜单导航、链接导航的元素，是 `navigator` 的缩写。

article: 用于表现一篇文章的主体内容，一般为文字集中显示的区域。

级块性元素主要完成 web 页面区域的划分，确保内容的有效分割。

aside: 用于表达注记、贴士、侧栏、摘要、插入的引用等作为补充主体的内容。

figure: 是对多个元素进行组合并展示的元素，通常与 `figcaption` 联合使用。

code: 表示一段代码块。

dialog: 用于表达人与人之间的对话，该元素包含 `dt` 和 `dd` 这两个组合元素， `dt` 用于表示说话者，而 `dd` 用来表示说话内容。

行内语义性元素主要完成 web 页面具体内容的引用和描述，是丰富内容展示的基础。

meter: 表示特定范围内的数值，可用于工资、数量、百分比等。

time: 表示时间值。

progress: 用来表示进度条，可通过对其 `max` 、 `min` 、 `step` 等属性进行控制，完成对进度的表示和监事。

video: 视频元素，用于支持和实现视频文件的直接播放，支持缓冲预载和多种视频媒体格式。

audio: 音频元素，用于支持和实现音频文件的直接播放，支持缓冲预载和多种音频媒体格式。

交互性元素主要用于功能性的内容表达，会有一定的内容和数据的关联，是各种事件的基础。

details: 用来表示一段具体的内容，但是内容默认可能不显示，通过某种手段（如单击）与 `legend` 交互才会显示出来。

datagrid: 用来控制客户端数据与显示，可以由动态脚本及时更新。

menu: 主要用于交互菜单（曾被废弃又被重新启用的元素）。

command: 用来处理命令按钮。

5. 你做的网页在哪些浏览器测试过, 这些浏览器的内核分别是什么?

参考答案:

a、IE: trident 内核

b、Firefox: gecko 内核

c、Safari: webkit 内核

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

e、Chrome:Blink(基于 webkit, Google 与 Opera Software 共同开发)

6. 每个 HTML 文件里开头都有个很重要的东西, Doctype, 知道这是干什么的吗?

参考答案:

声明位于文档中的最前面的位置, 处于标签之前。此标签可告知浏览器文档使用哪种 HTML 或 XHTML 规范。（重点: 告诉浏览器按照何种规范解析页面）

7. 说说你对 HTML5 认识? (是什么, 为什么)

参考答案:

是什么:

HTML5 指的是包括 HTML、CSS 和 JavaScript 在内的一套技术组合。它希望能够减少网页浏览器对于需要插件的丰富性网络应用服务 (Plug-in-Based Rich Internet Application, RIA), 例如: AdobeFlash 、Microsoft Silverlight 与 Oracle JavaFX 的需求, 并且提供更多能有效加强网络应用的标准集。HTML5

是 HTML 最新版本，2014 年 10 月由万维网联盟（W3C）完成标准制定。目标是替换 1999 年所制定的 HTML4.01 和 XHTML 1.0 标准，以期能在互联网应用迅速发展的时候，使网络标准达到匹配当代的网络需求。

为什么：

HTML4 陈旧不能满足日益发展的互联网需要，特别是移动互联网。为了增强浏览器功能 Flash 被广泛使用，但安全与稳定堪忧，不适合在移动端使用（耗电、触摸、不开放）。

HTML5 增强了浏览器的原生功能，符合 HTML5 规范的浏览器功能将更加强大，减少了 Web 应用对插件的依赖，让用户体验更好，让开发更加方便，另外 W3C 从推出 HTML4.0 到 5.0 之间共经历了 17 年，HTML 的变化很小，这并不符合一个好产品的演进规则。

8. 对 WEB 标准以及 W3C 的理解与认识？

参考答案：

标签闭合、标签小写、不乱嵌套、提高搜索机器人搜索几率、使用外链 css 和 js 脚本、结构行为表现的分离、文件下载与页面速度更快、内容能被更多的用户所访问、内容能被更广泛的设备所访问、更少的代码和组件，容易维护、改版方便，不需要变动页面内容、提供打印版本而不需要复制内容、提高网站易用性。

9. HTML5 行内元素有哪些, 块级元素有哪些, 空元素有哪些？

参考答案：

(1) 行内元素

a - 锚点

abbr - 缩写

acronym - 首字

b - 粗体 （ 不推荐 ）

bdo - bidi override

big - 大字体

br - 换行

cite - 引用

code - 计算机代码（在引用源码的时候需要）

dfn - 定义字段

em - 强调

font - 字体设定（不推荐）

i - 斜体

img - 图片

input - 输入框

kbd - 定义键盘文本

label - 表格标签

q - 短引用

s - 中划线（不推荐）

samp - 定义范例计算机代码

select - 项目选择

small - 小字体文本

span - 常用内联容器，定义文本内区块

strike - 中划线

strong - 粗体强调

sub - 下标

sup - 上标

textarea - 多行文本输入框

tt - 电传文本

u - 下划线

var - 定义变量

(2)块元素 (block element)

- * address - 地址
- * blockquote - 块引用
- * center - 居中对齐块
- * dir - 目录列表
- * div - 常用块级容器，也是 css layout 的主要标签
- * dl - 定义列表
- * fieldset - form 控制组
- * form - 交互表单
- * h1 - 大标题
- * h2 - 副标题
- * h3 - 3 级标题
- * h4 - 4 级标题
- * h5 - 5 级标题
- * h6 - 6 级标题
- * hr - 水平分隔线
- * isindex - input prompt
- * menu - 菜单列表
- * noframes - frames 可选内容，（对于不支持 frame 的浏览器显示此区块内容）
- * noscript - 可选脚本内容（对于不支持 script 的浏览器显示此内容）
- * ol - 排序列表
- * p - 段落
- * pre - 格式化文本
- * table - 表格
- ul - 非排序列表

(3) 可变元素

可变元素为根据上下文语境决定该元素为块元素或者内联元素。

- * applet - java applet
- * button - 按钮
- * del - 删除文本
- * iframe - inline frame
- * ins - 插入的文本
- * map - 图片区块 (map)
- * object - object 对象
- * script - 客户端脚本

(4) 空元素 (在 HTML[1] 元素中, 没有内容的 HTML 元素被称为空元素)

 //换行

<hr> //分隔线

<input> //文本框等

 //图片

<link> <meta>

10. 什么是 WebGL, 它有什么优点?

参考答案:

WebGL (全写 Web Graphics Library) 是一种 3D 绘图标准, 这种绘图技术标准允许把 JavaScript 和 OpenGL ES 2.0 结合在一起, 通过增加 OpenGL ES 2.0 的一个 JavaScript 绑定, WebGL 可以为 HTML5 Canvas 提供硬件 3D 加速渲染, 这样 Web 开发人员就可以借助系统显卡来在浏览器里更流畅地展示 3D 场景和模型

了，还能创建复杂的导航和数据视觉化。显然，WebGL 技术标准免去了开发网页专用渲染插件的麻烦，可被用于创建具有复杂 3D 结构的网站页面，甚至可以用来设计 3D 网页游戏等等。

WebGL 完美地解决了现有的 Web 交互式三维动画的两个问题：

第一，它通过 HTML 脚本本身实现 Web 交互式三维动画的制作，无需任何浏览器插件支持；

第二，它利用底层的图形硬件加速功能进行的图形渲染，是通过统一的、标准的、跨平台的 OpenGL 接口实现的。

通俗说 WebGL 中 canvas 绘图中的 3D 版本。因为原生的 WebGL 很复杂，我们经常会使用一些三方的库，如 three.js 等，这些库多数用于 HTML5 游戏开发。

11. 请你描述一下 cookies, sessionStorage 和 localStorage 的区别？

参考答案：

sessionStorage 和 localStorage 是 HTML5 Web Storage API 提供的，可以方便的在 web 请求之间保存数据。有了本地数据，就可以避免数据在浏览器和服务端间不必要地来回传递。sessionStorage、localStorage、cookie 都是在浏览器端存储的数据，其中 sessionStorage 的概念很特别，引入了一个“浏览器窗口”的概念。sessionStorage 是在同源的同窗口（或 tab）中，始终存在的数据。也就是说只要这个浏览器窗口没有关闭，即使刷新页面或进入同源另一页面，数据仍然存在。关闭窗口后，sessionStorage 即被销毁。同时“独立”打开的不同窗口，即使是同一页面，sessionStorage 对象也是不同的 cookies 会发送到服务器端。其余两个不会。

Microsoft 指出 Internet Explorer 8 增加 cookie 限制为每个域名 50 个，但 IE7 似乎也允许每个域名 50 个 cookie。Firefox 每个域名 cookie 限制为 50 个。Opera

每个域名 cookie 限制为 30 个。Firefox 和 Safari 允许 cookie 多达 4097 个字节，包括名（name）、值（value）和等号。Opera 许 cookie 多达 4096 个字节，包括：名（name）、值（value）和等号。Internet Explorer 允许 cookie 多达 4095 个字节，包括：名（name）、值（value）和等号。

区别：

1) Cookie

每个域名存储量比较小（各浏览器不同，大致 4K ）

所有域名的存储量有限制（各浏览器不同，大致 4K ）

有个数限制（各浏览器不同）

会随请求发送到服务器

2) LocalStorage

永久存储

单个域名存储量比较大（推荐 5MB ，各浏览器不同）

总体数量无限制

3) SessionStorage

只在 Session 内有效

存储量更大（推荐没有限制，但是实际上各浏览器也不同）

12. 说说你对 HTML 语义化的理解？

参考答案：

(1)什么是 HTML 语义化？

＜基本上都是围绕着几个主要的标签，像标题（ H1~H6 ）、列表（ li ）、强调（ strong em ）等等 ＞

根据内容的结构化（内容语义化），选择合适的标签（代码语义化）便于开发者阅读和写出更优雅的代码的同时让浏览器的爬虫和机器很好地解析。

(2)为什么要语义化？

为了在没有 CSS 的情况下，页面也能呈现出很好地内容结构、代码结构：为了裸奔时好看；

用户体验：例如 title、alt 用于解释名词或解释图片信息、label 标签的活用；

有利于 SEO：和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息：爬虫依赖于标签来确定上下文和各个关键字的权重；

方便其他设备解析（如屏幕阅读器、盲人阅读器、移动设备）以意义的方式来渲染网页；

便于团队开发和维护，语义化更具可读性，是下一步网页的重要动向，遵循 W3C 标准的团队都遵循这个标准，可以减少差异化。

(3) 语义化标签

<header></header>

<footer></footer>

<nav></nav>

<section></section>

<article></article>

SM:用来在页面中表示一套结构完整且独立的内容部分

<aside></aside>

SM:主题的附属信息(用途很广，主要就是一个附属内容)，如果 article 里面为一篇文章的话，那么文章的作者以及信息内容就是这篇文章的附属内容了

<figure></figure>

SM:媒体元素，比如一些视频，图片啊等等

`<datalist></datalist>`

SM:选项列表, 与 `input` 元素配合使用, 来定义 `input` 可能的值

`<details></details>`

SM:用于描述文档或者文档某个部分的细节, 默认属性为 `open`

ps:配合 `summary` 一起使用

13. link 和@import 的区别?

参考答案:

XML/HTML 代码

```
<link rel='stylesheet' rev='stylesheet' href='CSS 文件 '
type='text/css' media='all' />
```

XML/HTML 代码

```
<style type='text/css' media='screen'>
```

```
@import url('CSS 文件 ');
```

```
</style>
```

两者都是外部引用 CSS 的方式, 但是存在一定的区别:

区别 1: `link` 是 XHTML 标签, 除了加载 CSS 外, 还可以定义 RSS 等其他事务; `@import` 属于 CSS 范畴, 只能加载 CSS。

区别 2: `link` 引用 CSS 时, 在页面载入时同时加载; `@import` 需要页面网页完全载入以后加载。

区别 3: `link` 是 XHTML 标签, 无兼容问题; `@import` 是在 CSS2.1 提出的, 低版本的浏览器不支持。

区别 4: `link` 支持使用 Javascript 控制 DOM 去改变样式; 而 `@import` 不支持。

14. 说说你对 SVG 理解?

参考答案:

SVG 可缩放矢量图形 (Scalable Vector Graphics) 是基于可扩展标记语言 (XML), 用于描述二维矢量图形的一种图形格式。SVG 是 W3C ('World Wide Web Consortium' 即 '国际互联网标准组织') 在 2000 年 8 月制定的一种新的二维矢量图形格式, 也是规范中的网络矢量图形标准。SVG 严格遵从 XML 语法, 并用文本格式的描述性语言来描述图像内容, 因此是一种和图像分辨率无关的矢量图形格式。SVG 于 2003 年 1 月 14 日成为 W3C 推荐标准。

特点:

(1) 任意放缩

用户可以任意缩放图像显示, 而不会破坏图像的清晰度、细节等。

(2) 文本独立

SVG 图像中的文字独立于图像, 文字保留可编辑和可搜寻的状态。也不会有字体的限制, 用户系统即使没有安装某一字体, 也会看到和他们制作时完全相同的画面。

(3) 较小文件

总体来讲, SVG 文件比那些 GIF 和 JPEG 格式的文件要小很多, 因而下载也很快。

(4) 超强显示效果

SVG 图像在屏幕上总是边缘清晰, 它的清晰度适合任何屏幕分辨率和打印分辨率。

(5) 超级颜色控制

SVG 图像提供一个 1600 万种颜色的调色板, 支持 ICC 颜色描述文件标准、RGB、线 X 填充、渐变和蒙版。

(6)交互 X 和智能化。SVG 面临的主要问题一个是如何和已经占有重要市场份额的矢量图形格式 Flash 竞争的问题,另一个问题就是 SVG 的本地运行环境下的厂家支持程度。

浏览器支持:

Internet Explorer9, 火狐, 谷歌 Chrome, Opera 和 Safari 都支持 SVG。

IE8 和早期版本都需要一个插件-如 Adobe SVG 浏览器, 这是免费提供的。

15. HTML 全局属性(global attribute)有哪些?

参考答案:

参考资料:

MDN: [html global attribute](#) 或者 W3C [HTML global-attributes](#)

accesskey: 设置快捷键, 提供快速访问元素如 aaa 在 windows 下的 firefox 中按 alt + shift + a 可激活元素;

class: 为元素设置类标识, 多个类名用空格分开, CSS 和 javascript 可通过 class 属性获取元素;

contenteditable: 指定元素内容是否可编辑;

contextmenu: 自定义鼠标右键弹出菜单内容 **data-***: 为元素增加自定义属性;

dir: 设置元素文本方向;

draggable: 设置元素是否可拖拽;

dropzone: 设置元素拖放类型: copy, move, link hidden: 表示一个元素是否与文档。样式上会导致元素不显示, 但是不能用这个属性实现样式效果 **id:** 元素 id, 文档内唯一;

lang: 元素内容的语言 **spellcheck:** 是否启动拼写和语法检查

style: 行内 css 样式

tabindex: 设置元素可以获得焦点, 通过 tab 可以导航;

title: 元素相关的建议信息;

translate: 元素和子孙节点内容是否需要本地化;

16. 说说超链接 target 属性的取值和作用?

参考答案:

target 这个属性指定所链接的页面在浏览器窗口中的打开方式。

它的参数值主要有:

- a、_blank: 在新浏览器窗口中打开链接文件;
- b、_parent : 将链接的文件载入含有该链接框架的父框架集或父窗口中。如果含有该链接的框架不是嵌套的, 则在浏览器全屏窗口中载入链接的文件, 就象 _self 参数一;
- c、_self: 在同一框架或窗口中打开所链接的文档。此参数为默认值, 通常不用指定。但是我不太理解;
- d、_top: 在当前的整个浏览器窗口中打开所链接的文档, 因而会删除所有框架;

17. `data-` 属性的作用是什么?

参考答案:

`data-` 为 H5 新增的为前端开发者提供自定义的属性, 这些属性集可以通过对象的 `dataset` 属性获取, 不支持该属性的浏览器可以通过 `getAttribute` 方法获取, 需要注意的是: `data-` 之后的以连字符分割的多个单词组成的属性, 获取的时候使用驼峰风格。所有主流浏览器都支持 data-* 属性。

即: 当没有合适的属性和元素时, 自定义的 data 属性是能够存储页面或 App 的私有的自定义数据。

18. 介绍一下你对浏览器内核的理解？

参考答案：

主要分成两部分：渲染引擎(layout engineer 或 Rendering Engine)和 JS 引擎。

渲染引擎：负责取得网页的内容（HTML、XML、图像等等）、整理讯息（例如加入 CSS 等），以及计算网页的显示方式，然后会输出至显示器或打印机。浏览器的内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同。所有网页浏览器、电子邮件客户端以及其它需要编辑、显示网络内容的应用程序都需要内核。

JS 引擎：解析和执行 javascript 来实现网页的动态效果。

最开始渲染引擎和 JS 引擎并没有区分的很明确，后来 JS 引擎越来越独立，内核就倾向于只指渲染引擎。

19. 常见的浏览器内核有哪些？

参考答案：

view sourceprint?

Trident 内核：IE, MaxThon, TT, The World, 360, 搜狗浏览器等。[又称 MSHTML]

Gecko 内核：Netscape6 及以上版本, FF, MozillaSuite/SeaMonkey 等

Presto 内核：Opera7 及以上。[Opera 内核原为：Presto，现为：Blink;]

Webkit 内核：Safari, Chrome 等。 [Chrome 的：Blink (WebKit 的分支)]

20. iframe 有那些缺点？

参考答案：

- (1) iframe 会阻塞主页面的 Onload 事件;
- (2) 搜索引擎的检索程序无法解读这种页面，不利于 SEO;

(3) iframe 和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载。

使用 iframe 之前需要考虑这两个缺点。如果需要使用 iframe，最好是通过 javascript

动态给 iframe 添加 src 属性值，这样可以绕开以上两个问题。

21. Label 的作用是什么，是怎么用的？

参考答案：

label 标签来定义表单控制间的关系，当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上。

```
<label for='Name'>Number:</label>
```

```
<input type="text" name='Name' id='Name' />
```

```
<label>Date:<input type='text' name='B' /></label>
```

注意:label 的 for 属性值要与后面对应的 input 标签 id 属性值相同

```
<label for='Name'>Number:</label>
```

```
<input type="text" name='Name' id='Name' />
```

22. 如何实现浏览器内多个标签页之间的通信？

参考答案：

WebSocket、SharedWorker；

也可以调用 localStorage、cookies 等本地存储方式；

localStorage 另一个浏览上下文里被添加、修改或删除时，它都会触发一个事件，我们通过监听事件，控制它的值来进行页面信息通信；

注意 quirks: Safari 在无痕模式下设置 localStorage 值时会抛出 QuotaExceededError 的异常；

23. 如何在页面上实现一个圆形的可点击区域？

参考答案：

a、map+area 或者 svg

b、border-radius

c、纯 js 实现 需要求一个点在不在圆上简单算法、获取鼠标坐标等等

24. title 与 h3 的区别、b 与 strong 的区别、i 与 em 的区别？

参考答案：

(1) title 属性没有明确意义只表示是个标题，H1 则表示层次明确的标题，对页面信息的抓取也有很大的影响；

(2) strong 是标明重点内容，有语气加强的含义，使用阅读设备阅读网时：

会重读，而是展示强调内容。

(3) i 内容展示为斜体，em 表示强调的文本；

Physical Style Elements -- 自然样式标签

b, i, u, s, pre

Semantic Style Elements -- 语义样式标签

strong, em, ins, del, code

应该准确使用语义样式标签,但不能滥用,如果不能确定时首选使用自然样式标签。

25. 实现不使用 border 画出 1px 高的线，在不同浏览器的标准模式与怪异模式下都能保持一致的效果？

参考答案：

```
<div style="width:100%;height:1px;background-color:black"></div>
```

26. HTML5 标签的作用?(用途)

参考答案:

- a、使 Web 页面的内容更加有序和规范;
- b、使搜索引擎更加容易按照 HTML5 规则识别出有效的内容;
- c、使 Web 页面更接近于一种数据字段和表;

27. 简述一下 src 与 href 的区别?

参考答案:

src 用于替换当前元素, href 用于在当前文档和引用资源之间确立联系。

src 是 source 的缩写, 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内, 例如 js 脚本, img 图片和 frame 等元素。

```
<script src = 'js.js'></script>
```

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, 图片和框架等元素也如此, 类似于将所指向资源嵌入当前标签内。这也是为什么将 js 脚本放在底部而不是头部。

href 是 Hypertext Reference 的缩写, 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。这也是为什么建议使用 link 方式来加载 css, 而不是使用@import 方式。

28. 谈谈你对 canvas 的理解？

参考答案：

canvas 是 HTML5 中新增一个 HTML5 标签与操作 canvas 的 javascript API，它可以实现在网页中完成动态的 2D 与 3D 图像技术。标记和 SVG 以及 VML 之间的一个重要不同是，有一个基于 JavaScript 的绘图 API，而 SVG 和 VML 使用一个 XML 文档来描述绘图。SVG 绘图很容易编辑与生成，但功能明显要弱一些。canvas 可以完成动画、游戏、图表、图像处理等原来需要 Flash 完成的一些功能。

29. WebSocket 与消息推送？

参考答案：

B/S 架构的系统多使用 HTTP 协议，

HTTP 协议的特点：

- 1) 无状态协议
- 2) 用于通过 Internet 发送请求消息和响应消息
- 3) 使用端口接收和发送消息，默认为 80 端口 底层通信还是使用 Socket 完成。

HTTP 协议决定了服务器与客户端之间的连接方式，无法直接实现消息推送（F5 已坏），一些变通的解决办法：

双向通信与消息推送

轮询：客户端定时向服务器发送 Ajax 请求，服务器接到请求后马上返回响应信息并关闭连接。

优点：后端程序编写比较容易。

缺点：请求中有大半是无用，浪费带宽和服务器资源。

实例：适于小型应用。

长轮询：客户端向服务器发送 Ajax 请求，服务器接到请求后 hold 住连接，直到有新消息才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。

优点：在无消息的情况下不会频繁的请求，耗费资小。

缺点：服务器 hold 连接会消耗资源，返回数据顺序无保证，难于管理维护。Comet 异步的 ashx，

实例：WebQQ、Hi 网页版、Facebook IM。

长连接：在页面里嵌入一个隐藏 iframe，将这个隐藏 iframe 的 src 属性设为对一个长连接的请求或是采用 xhr 请求，服务器端就能源源不断地往客户端输入数据。

优点：消息即时到达，不发无用请求；管理起来也相对便。

缺点：服务器维护一个长连接会增加开销。

实例：Gmail 聊天

Flash Socket：在页面中内嵌入一个使用了 Socket 类的 Flash 程序 JavaScript 通过调用此 Flash 程序提供的 Socket 接口与服务器端的 Socket 接口进行通信，JavaScript 在收到服务器端传送的信息后控制页面的显示。

优点：实现真正的即时通信，而不是伪即时。

缺点：客户端必须安装 Flash 插件；非 HTTP 协议，无法自动穿越防火墙。

实例：网络互动游戏。

Websocket：

WebSocket 是 HTML5 开始提供的一种浏览器与服务器间进行全双工通讯的网络技术。依靠这种技术可以实现客户端和服务器的长连接，双向实时通信。

特点：

- a、事件驱动
- b、异步
- c、使用 ws 或者 wss 协议的客户端 socket
- d、能够实现真正意义上的推送功能

缺点：少部分浏览器不支持，浏览器支持的程度与方式有区别。

30. `img` 的 `title` 和 `alt` 有什么区别？

参考答案：

`Alt` 用于图片无法加载时显示 `Title` 为该属性提供信息，通常当鼠标滑动到元素上的时候显示。

31. 表单的基本组成部分有哪些，表单的主要用途是什么？

参考答案：

组成：表单标签、表单域、表单按钮

a、表单标签：这里面包含了处理表单数据所用 CGI 程序的 URL, 以及数据提交到服务器的方法。

b、表单域：包含了文本框、密码框、隐藏域、多行文本框、复选框、单选框、下拉选择框、和文件上传框等。

c、表单按钮：包括提交按钮，复位按钮和一般按钮；用于将数据传送到服务器上的 CGI 脚本或者取消输入，还可以用表单按钮来控制其他定义了处理脚本的处理工作。

主要用途：表单在网页中主要负责数据采集的功能，和向服务器传送数据。

32. 表单提交中 `Get` 和 `Post` 方式的区别？

参考答案：

(1)、`get` 是从服务器上获取数据，`post` 是向服务器传送数据。

(2)、`get` 是把参数数据队列加到提交表单的 `ACTION` 属性所指的 URL 中，值和表单内各个字段一一对应，在 URL 中可以看到。`post` 是通过 HTTP `post` 机制，将

表单内各个字段与其内容放置在 HTML HEADER 内一起传送到 ACTION 属性所指的 URL 地址, 用户看不到这个过程。

(3)、对于 get 方式, 服务器端用 Request.QueryString 获取变量的值, 对于 post 方式, 服务器端用 Request.Form 获取提交的数据。

(4)、get 传送的数据量较小, 不能大于 2KB。post 传送的数据量较大, 一般被默认为不受限制。但理论上, IIS4 中最大量为 80KB, IIS5 中为 100KB 。

(5)、get 安全性低, post 安全性较高。

33. HTML5 有哪些新增的表单元素?

参考答案:

HTML5 新增了很多表单元素让开发者构建更优秀的 Web 应用程序, 主要有:

datalist

keygen

output

34. HTML5 废弃了哪些 HTML4 标签?

参考答案:

HTML5 废弃了一些过时的, 不合理的 HTML 标签:

- frame
- frameset
- noframe
- applet
- big
- center
- basefont

35. HTML5 标准提供了哪些新的 API？

参考答案：

HTML5 提供的应用程序 API 主要有：

- Media API
- Text Track API
- Application Cache API
- User Interaction
- Data Transfer API
- Command API
- Constraint Validation API
- History API

36. HTML5 存储类型有什么区别？

参考答案：

HTML5 能够本地存储数据，在之前都是使用 cookies 使用的。HTML5 提供了下面两种本地存储方案：

- localStorage 用于持久化的本地存储，数据永远不会过期，关闭浏览器也不会丢失。
- sessionStorage 同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此 sessionStorage 不是一种持久化的本地存储，仅仅是会话级别的存储

37. HTML5 应用程序缓存和浏览器缓存有什么区别？

参考答案：

应用程序缓存是 HTML5 的重要特性之一，提供了离线使用的功能，让应用程序可以获取本地的网站内容，例如 HTML、CSS、图片以及 JavaScript。这个特性可以提高网站性能，它的实现借助于 manifest 文件，如下：

```
<!doctype html>
<html manifest="example.appcache">
...
</html>
```

与传统浏览器缓存相比，它不强制用户访问的网站内容被缓存。

38. HTML5 Canvas 元素有什么用？

参考答案：

Canvas 元素用于在网页上绘制图形，该元素标签强大之处在于可以直接在 HTML 上进行图形操作，

```
<canvas id="canvas1" width="300" height="100">
</canvas>
```

39. 除了 audio 和 video，HTML5 还有哪些媒体标签？

参考答案：

HTML5 对于多媒体提供了强有力的支持，除了 audio 和 video 标签外，还支持以下标签：

<embed>标签定义嵌入的内容，比如插件。

```
<embed type="video/quicktime" src="Fishing.mov">
```

<source>对于定义多个数据源很有用。

```
<video width="450" height="340" controls>
```

```
<source src= "jamshed.mp4"    type= "video/mp4" >
<source src= "jamshed.ogg"    type=  "video/ogg" >
</video>
```

<track>标签为诸如 video 元素之类的媒介规定外部文本轨道。用于规定字幕文件或其他包含文本的文件，当媒介播放时，这些文件是可见的。

```
<video width= "450"    height= "340"    controls>
    <source src= "jamshed.mp4"    type= "video/mp4" >
    <source src= "jamshed.ogg"    type=  "video/ogg" >
    <track kind= "subtitles"    label= "English"    src=
"jamshed_en.vtt"    srclang= "en"    default></track>
    <track kind= "subtitles"    label= "Arabic"    src=
"jamshed_ar.vtt"    srclang= "ar" ></track>
</video>
```

据源很有用。标签为诸如 video 元素之类的媒介规定外部文本轨道。用于规定字幕文件或其他包含文本的文件，当媒介播放时，这些文件是可见的。

40. HTML5 中如何嵌入视频？

参考答案：

和音频类似，HTML5 支持 MP4、WebM 和 Ogg 格式的视频，

简单示例：

```
<video width= "450"    height= "340"    controls>
    <source src= "jamshed.mp4"    type= "video/mp4" >
    Your browser does' nt support video embedding feature.
</video>
```

41. HTML5 中如何嵌入音频？

参考答案：

HTML5 支持 MP3、Wav 和 Ogg 格式的音频，下面是在网页中嵌入音频的简单示例：

```
<audio controls>
  <source src= “jamshed.mp3”   type= “audio/mpeg” >
  Your browser does' nt support audio embedding feature.
</audio>
```

42. 新的 HTML5 文档类型和字符集是？

参考答案：

HTML5 文档类型很简单：

```
<!doctype html>
```

HTML5 使用 UTF-8 编码示例：

```
<meta charset= “UTF-8” >
```

第二章 CSS

1. 解释一下 CSS 的盒子模型？

参考回答：

回答一：

a、标准的 css 盒子模型：宽度=内容的宽度+边框的宽度+加上内边框的宽度

b、网页设计中常听的属性名：内容(content)、填充(padding)、边框(border)、边界(margin)，CSS 盒子模式都具备这些属性。

c、这些属性我们可以把它转移到我们日常生活中的盒子（箱子）上来理解，日常生活中所见的盒子也就是能装东西的一种箱子，也具有这些属性，所以叫它盒子模式。CSS 盒子模型就是在网页设计中经常用到的 CSS 技术所使用的一种思维模型。

回答二：

标准的盒模型： $\text{width} = \text{content}$

IE 盒模型： $\text{width} = \text{content} + \text{padding-left} + \text{padding-right} + \text{border-left} + \text{border-right}$

2. 请你说说 CSS 选择器的类型有哪些, 并举几个例子说明其用法?

参考答案：

类型：基础的选择器、组合选择器、属性选择器、伪类、伪元素

基础的选择器：

选择器	含义	示例
*	通用元素选择器，匹配任何元素	<code>* { margin:0; padding:0; }</code>
E	标签选择器，匹配所有使用E标签的元素	<code>p { font-size:2em; }</code>
.info和E.info	class选择器，匹配所有class属性中包含info的元素	<code>.info { background:#ff0; }</code> <code>p.info { background:#ff0; }</code> <code>p .info { background:#ff0; }</code>
#info和E#info	id选择器，匹配所有id属性等于info的元素	<code>#info { background:#ff0; }</code> <code>p#info { background:#ff0; }</code>

组合选择器：

选择器	含义	示例
E,F	多元素选择器，同时匹配所有E元素或F元素，E和F之间用逗号分隔	<code>div,p { color:#f00; }</code>
E F	后代元素选择器，匹配所有属于E元素后代的F元素，E和F之间用空格分隔	<code>#nav li { display:inline; }</code> <code>li a { font-weight:bold; }</code>
E > F	子元素选择器，匹配所有E元素的子元素F	<code>div > strong { color:#f00; }</code>
E + F	毗邻元素选择器，匹配所有紧随E元素之后的同级元素F	<code>p + p { color:#f00; }</code>

属性选择器：

选择器	含义	示例
E[att]	匹配所有具有att属性的E元素，不考虑它的值。（注意：E在此处可以省略，比如“[checked]”。以下同。）	p[title] { color:#f00; }
E[att=val]	匹配所有att属性等于“val”的E元素	div[class=" error"] { color:#f00; }
E[att~=val]	匹配所有att属性具有多个空格分隔的值、其中一个值等于“val”的E元素	td[class~=" name"] { color:#f00; }
E[att =val]	匹配所有att属性具有多个连字号分隔（hyphen-separated）的值、其中一个值以“val”开头的E元素，主要用于lang属性，比如“en”、“en-us”、“en-gb”等等	p[lang =en] { color:#f00; }

伪类：

选择器	含义	示例
E:first-child	匹配父元素的第一个子元素	p:first-child { font-style:italic; }
E:link	匹配所有未被点击的链接	
E:visited	匹配所有已被点击的链接	input[type=text]:focus { color:#000; background:#ffe; }
E:active	匹配鼠标已经其上按下、还没有释放的E元素	input[type=text]:focus:hover { background:#fff; }
E:hover	匹配鼠标悬停其上的E元素	
E:focus	匹配获得当前焦点的E元素	q:lang(sv) { quotes: "\201D" "\201D" "\2019" "\2019"; }
E:lang(c)	匹配lang属性等于c的E元素	

伪元素：

选择器	含义	示例
E:first-line	匹配E元素的第一行	p:first-line { font-weight:bold; color:#600; }
E:first-letter	匹配E元素的第一个字母	.preamble:first-letter { font-size:1.5em; font-weight:bold; }
E:before	在E元素之前插入生成的内容	.cbb:before { content:" "; display:block; height:17px; width:18px; background:url(top.png) no-repeat 0 0; margin:0 0 0 -18px; }
E:after	在E元素之后插入生成的内容	a:link:after { content: " (" attr(href) ") "; }

3. 请你说说 CSS 有什么特殊性？（优先级、计算特殊值）

参考答案：

优先级：

- (1)、同类型，同级别的样式后者先于前者
- (2))、ID > 类样式 > 标签 > *
- (3)、内联>ID 选择器>伪类>属性选择器>类选择器>标签选择器>通用选择器(*)>

继承的样式

- (4)、具体 > 泛化的，特殊性即 css 优先级
- (5)、近的 > 远的（内嵌样式 > 内部样式表 > 外联样式表）

内嵌样式：内嵌在元素中，span

内部样式表：在页面中的样式，写在<style></style>中的样式

外联样式表：单独存在一个 css 文件中，通过 link 引入或 import 导入的样式

(6)、!important 权重最高，比 inline style 还要高

计算特殊性值

important > 内嵌 > ID > 类 > 标签 | 伪类 | 属性选择 > 伪对象 > 继承 > 通配符

权重、特殊性算法：

CSS 样式选择器分为 4 个等级，a、b、c、d

(1)、如果样式是行内样式（通过 Style= “ ” 定义），那么 a=1, 1, 0, 0, 0

(2)、b 为 ID 选择器的总数 0, 1, 0, 0

(3)、c 为属性选择器，伪类选择器和 class 类选择器的数量。0, 0, 1, 0

(4)、d 为标签、伪元素选择器的数量 0, 0, 0, 1

(5)、!important 权重最高，比 inline style 还要高

比如结果为：1093 比 1100，按位比较，从左到右，只要一位高于则立即胜出，否则继续比较。

选择器	特殊性 (a,b,c,d)
Style= " "	1,0,0,0
#wrapper #content {}	0,2,0,0
#content .datePosted {}	0,1,1,0
div#content {}	0,1,0,1
#content p {}	0,1,0,1
#content {}	0,1,0,0
p.comment .dateposted {}	0,0,2,1
div.comment p {}	0,0,1,2
.comment p {}	0,0,1,1
p.comment {}	0,0,1,1
.comment {}	0,0,1,0
div p {}	0,0,0,2
p {}	0,0,0,1

从左到右1对1对比，当比出有大者时即可停止比较

4. 要动态改变层中内容可以使用的方法？

参考答案：

innerHTML, innerText

5. 常见浏览器兼容性问题与解决方案？

参考答案：

(1) 浏览器兼容问题一：不同浏览器的标签默认的外补丁和内补丁不同

问题症状：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。

碰到频率:100%

解决方案：CSS 里 *{margin:0;padding:0;}

备注：这个是最常见的也是最易解决的一个浏览器兼容性问题，几乎所有的 CSS 文件开头都会用通配符*来设置各个标签的内外补丁是 0。

(2) 浏览器兼容问题二：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

问题症状: 常见症状是 IE6 中后面的一块被顶到下一行

碰到频率: 90% (稍微复杂点的页面都会碰到, float 布局最常见的浏览器兼容问题)

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

备注: 我们最常用的就是 div+CSS 布局了, 而 div 就是一个典型的块属性标签, 横向布局的时候我们通常都是用 div float 实现的, 横向的间距设置如果用 margin 实现, 这就是一个必然会碰到的兼容性问题。

(3) 浏览器兼容问题三：设置较小高度标签（一般小于 10px），在 IE6, IE7, 遨游中高度超出自己设置高度

问题症状: IE6、7 和遨游里这个标签的高度不受控制, 超出自己设置的高度

碰到频率: 60%

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。

备注: 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。

(4) 浏览器兼容问题四：行内属性标签，设置 display:block 后采用 float 布局，又有横行的 margin 的情况，IE6 间距 bug

问题症状: IE6 里的间距比超过设置的间距

碰到几率: 20%

解决方案 : 在 `display:block`; 后面加入 `display:inline`; `display:table`;

备注: 行内属性标签, 为了设置宽高, 我们需要设置 `display:block`; (除了 input 标签比较特殊)。在用 float 布局并有横向的 margin 后, 在 IE6 下, 他就具有了块属性 float 后的横向 margin 的 bug。不过因为它本身就是行内属性标签, 所

以我们再加上 `display:inline` 的话，它的高宽就不可设了。这时候我们还需要在 `display:inline` 后面加入 `display: inline-block`。

(5) 浏览器兼容问题五：图片默认有间距

问题症状：几个 `img` 标签放在一起的时候，有些浏览器会有默认的间距，加了问题一中提到的通配符也不起作用。

碰到几率：20%

解决方案：使用 `float` 属性为 `img` 布局

备注：因为 `img` 标签是行内属性标签，所以只要不超出容器宽度，`img` 标签都会排在一行里，但是部分浏览器的 `img` 标签之间会有个间距。去掉这个间距使用 `float` 是正道。（我的一个学生使用负 `margin`，虽然能解决，但负 `margin` 本身就是容易引起浏览器兼容问题的用法，所以我禁止他们使用）

(6) 浏览器兼容问题六：标签最低高度设置 `min-height` 不兼容

问题症状：因为 `min-height` 本身就是一个不兼容的 CSS 属性，所以设置 `min-height` 时不能很好的被各个浏览器兼容

碰到几率：5%

解决方案：如果我们要设置一个标签的最小高度 200px，需要进行的设置为：

```
{min-height:200px; height:auto !important; height:200px;
overflow:visible;}
```

备注：在 B/S 系统前端开发时，有很多情况下我们又这种需求。当内容小于一个值（如 300px）时。容器的高度为 300px；当内容高度大于这个值时，容器高度被撑高，而不是出现滚动条。这时候我们就会面临这个兼容性问题。

(7) 浏览器兼容问题七：透明度的兼容 CSS 设置

一般在 ie 中用的是 `filter:alpha(opacity=0)`；这个属性来设置 `div` 或者是块级元素的透明度，而在 `firefox` 中，一般就是直接使用 `opacity:0`，对于兼容的，一般的做法就是在书写 `css` 样式的将 2 个都写上就行，就能实现兼容

6. 列出 display 的值并说明他们的作用？

display: none | inline | block | list-item
| inline-block | table | inline-table | table-caption | table-cell
| table-row | table-row-group | table-column | table-column-group |
table-footer-group | table-header-group | run-in | box | inline-box |
flexbox | inline-flexbox | flex | inline-flex

默认值: inline

none: 隐藏对象。与 visibility 属性的 hidden 值不同，其不为被隐藏的对象保留其物理空间

inline: 指定对象为内联元素。

block: 指定对象为块元素。

list-item: 指定对象为列表项目。

inline-block: 指定对象为内联块元素。(CSS2)

table: 指定对象作为块元素级的表格。类同于 html 标签<table> (CSS2)

inline-table: 指定对象作为内联元素级的表格。类同于 html 标签<table> (CSS2)

table-caption: 指定对象作为表格标题。类同于 html 标签<caption> (CSS2)

table-cell: 指定对象作为表格单元格。类同于 html 标签<td> (CSS2)

table-row: 指定对象作为表格行。类同于 html 标签<tr> (CSS2)

table-row-group: 指定对象作为表格行组。类同于 html 标签<tbody> (CSS2)

table-column: 指定对象作为表格列。类同于 html 标签<col> (CSS2)

table-column-group: 指定对象作为表格列组显示。类同于 html 标签<colgroup> (CSS2)

table-header-group: 指定对象作为表格标题组。类同于 html 标签<thead> (CSS2)

table-footer-group: 指定对象作为表格脚注组。类同于 html 标签<tfoot>

(CSS2)

run-in: 根据上下文决定对象是内联对象还是块级对象。(CSS3)

box: 将对象作为弹性伸缩盒显示。(伸缩盒最老版本)(CSS3)

inline-box: 将对象作为内联块级弹性伸缩盒显示。(伸缩盒最老版本)(CSS3)

flexbox: 将对象作为弹性伸缩盒显示。(伸缩盒过渡版本)(CSS3)

inline-flexbox: 将对象作为内联块级弹性伸缩盒显示。(伸缩盒过渡版本)(CSS3)

flex: 将对象作为弹性伸缩盒显示。(伸缩盒最新版本)(CSS3)

inline-flex: 将对象作为内联块级弹性伸缩盒显示。(伸缩盒最新版本)(CSS3)

7. 如何居中 div, 如何居中一个浮动元素?

参考答案:

(1) 非浮动元素居中: 可以设置 `margin:0 auto` 令其居中, 定位, 父级元素 `text-align:center` 等等

(2) 浮动元素居中:

方法一: 设置当前 div 的宽度, 然后设置 `margin-left:50%; position:relative; left:-250px;` 其中的 left 是宽度的一半。

方法二: 父元素和子元素同时左浮动, 然后父元素相对左移动 50%, 再然后子元素相对左移动-50%。

方法三: position 定位等等。

8. CSS 中 link 和@import 的区别是?

参考答案:

(1) link 属于 HTML 标签, 而@import 是 CSS 提供的;

(2) 页面被加载的时, link 会同时被加载, 而@import 引用的 CSS 会等到页面被加载完再加载;

(3) import 只在 IE5 以上才能识别，而 link 是 HTML 标签，无兼容问题；

(4) link 方式的样式的权重 高于@import 的权重.

9. 请列举几种清除浮动的方法(至少两种)?

参考答案:

(1) 父级 div 定义 height

原理：父级 div 手动定义 height，就解决了父级 div 无法自动获取到高度的问题。

优点：简单、代码少、容易掌握

缺点：只适合高度固定的布局，要给出精确的高度，如果高度和父级 div 不一样时，会产生问题

建议：不推荐使用，只建议高度固定的布局时使用

(2) 结尾处加空 div 标签 clear:both

原理：添加一个空 div，利用 css 提高的 clear:both 清除浮动，让父级 div 能自动获取到高度

优点：简单、代码少、浏览器支持好、不容易出现怪问题

缺点：不少初学者不理解原理；如果页面浮动布局多，就要增加很多空 div，让人感觉很不好

建议：不推荐使用，但此方法是以前主要使用的一种清除浮动方法

(3) 父级 div 定义 伪类:after 和 zoom

原理：IE8 以上和非 IE 浏览器才支持:after，原理和方法 2 有点类似，zoom(IE 独有属性)可解决 ie6, ie7 浮动问题

优点：浏览器支持好、不容易出现怪问题（目前：大型网站都有使用，如：腾讯，网易，新浪等等）

缺点：代码多、不少初学者不理解原理，要两句代码结合使用才能让主流浏览器都支持。

建议：推荐使用，建议定义公共类，以减少 CSS 代码。

(4) 父级 div 定义 `overflow:hidden`

原理：必须定义 width 或 zoom:1，同时不能定义 height，使用 `overflow:hidden` 时，浏览器会自动检查浮动区域的高度

优点：简单、代码少、浏览器支持好

缺点：不能和 position 配合使用，因为超出的尺寸的会被隐藏。

建议：只推荐没有使用 position 或对 `overflow:hidden` 理解比较深的朋友使用。

(5) 父级 div 定义 `overflow:auto`

原理：必须定义 width 或 zoom:1，同时不能定义 height，使用 `overflow:auto` 时，浏览器会自动检查浮动区域的高度

优点：简单、代码少、浏览器支持好

缺点：内部宽高超过父级 div 时，会出现滚动条。

建议：不推荐使用，如果你需要出现滚动条或者确保你的代码不会出现滚动条就使用吧。

10. block, inline 和 inline-block 细节对比？

参考答案：

• `display:block`

a、block 元素会独占一行，多个 block 元素会各自新起一行。默认情况下，block 元素宽度自动填满其父元素宽度。

b、block 元素可以设置 width,height 属性。块级元素即使设置了宽度，仍然是独占一行。

c、block 元素可以设置 margin 和 padding 属性。

• `display:inline`

a、inline 元素不会独占一行，多个相邻的行内元素会排列在同一行里，直到一行排列不下，才会新换一行，其宽度随元素的内容而变化。

b、inline 元素设置 width,height 属性无效。

c、inline 元素的 margin 和 padding 属性，水平方向的 padding-left, padding-right, margin-left, margin-right 都产生边距效果；但垂直方向的 padding-top, padding-bottom, margin-top, margin-bottom 不会产生边距效果。

- **display:inline-block**

a、简单来说就是将对象呈现为 inline 对象，但是对象的内容作为 block 对象呈现。之后的内联对象会被排列在同一行内。比如我们可以给一个 link (a 元素) inline-block 属性值，使其既具有 block 的宽度高度特性又具有 inline 的同行特性。

补充说明

a、一般我们会用 display:block, display:inline 或者 display:inline-block 来调整元素的布局级别，其实 display 的参数远远不止这三种，仅仅是比较常用而已。

b、IE（低版本 IE）本来是不支持 inline-block 的，所以在 IE 中对内联元素使用 display:inline-block, 理论上 IE 是不识别的，但使用 display:inline-block 在 IE 下会触发 layout，从而使内联元素拥有了 display:inline-block 属性的表象。

11. 什么叫优雅降级和渐进增强？

参考答案：

优雅降级：Web 站点在所有新式浏览器中都能正常工作，如果用户使用的是老式浏览器，则代码会检查以确认它们是否能正常工作。由于 IE 独特的盒模型布局问题，针对不同版本的 IE 的 hack 实践过优雅降级了，为那些无法支持功能的浏览器增加候选方案，使之在旧式浏览器上以某种形式降级体验却不至于完全失效。

渐进增强：从被所有浏览器支持的基本功能开始，逐步地添加那些只有新式浏览器才支持的功能，向页面增加无害于基础浏览器的额外样式和功能的。当浏览器支持时，它们会自动地呈现出来并发挥作用。

12. 说说浮动元素会引起的问题和你的解决办法

参考答案：

问题：

- (1) 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：

使用 CSS 中的 `clear:both;` 属性来清除元素的浮动可解决问题(2)、(3)，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;
}

.clearfix{
display: inline-block;
} /* for IE/Mac */
```

清除浮动的几种方法：

(1) 额外标签法, <div style="clear:both;"></div> (缺点: 不过这个办法会增加额外的标签使 HTML 结构看起来不够简洁。)

(2) 使用 after 伪类

```
#parent:after{
    content:" ";
    height:0;
    visibility:hidden;
    display:block;
    clear:both;
}
```

(3) 浮动外部元素

(4) 设置`overflow`为`hidden`或者 auto

13. 你有哪些性能优化的方法?

参考答案:

回答一:

(1) 减少 http 请求次数: CSS Sprites, JS、CSS 源码压缩、图片大小控制合适; 网页 Gzip, CDN 托管, data 缓存, 图片服务器。

(2) 前端模板 JS+数据, 减少由于 HTML 标签导致的带宽浪费, 前端用变量保存 AJAX 请求结果, 每次操作本地变量, 不用请求, 减少请求次数

(3) 用 innerHTML 代替 DOM 操作, 减少 DOM 操作次数, 优化 javascript 性能。

(4) 当需要设置的样式很多时设置 className 而不是直接操作 style。

(5) 少用全局变量、缓存 DOM 节点查找的结果。减少 IO 读取操作。

(6) 避免使用 CSS Expression (css 表达式) 又称 Dynamic properties (动态属性)。

(7) 图片预加载, 将样式表放在顶部, 将脚本放在底部 加上时间戳。

回答二：

- (1) 减少 HTTP 请求次数
- (2) 使用 CDN
- (3) 避免空的 src 和 href
- (4) 为文件头指定 Expires
- (5) 使用 gzip 压缩内容
- (6) 把 CSS 放到顶部
- (7) 把 JS 放到底部
- (8) 避免使用 CSS 表达式
- (9) 将 CSS 和 JS 放到外部文件中
- (10) 避免跳转
- (11) 可缓存的 AJAX
- (12) 使用 GET 来完成 AJAX 请求

14. 为什么要初始化 CSS 样式？

参考答案：

因为浏览器的兼容问题，不同浏览器对有些标签的默认值是不同的，如果没对 CSS 初始化往往会出现浏览器之间的页面显示差异。

当然，初始化样式会对 SEO 有一定的影响，但鱼和熊掌不可兼得，但力求影响最小的情况下初始化。

最简单的初始化方法就是：

```
* {  
padding: 0;  
margin: 0;  
} （不建议）
```

15. 解释下浮动和它的工作原理？清除浮动的技巧？

参考答案：

浮动元素脱离文档流，不占据空间。浮动元素碰到包含它的边框或者浮动元素的边框停留。

(1) 使用空标签清除浮动。

这种方法是在所有浮动标签后面添加一个空标签定义 `css clear:both`. 弊端就是增加了无意义标签。

(2) 使用 `overflow`。

给包含浮动元素的父标签添加 `css` 属性 `overflow:auto; zoom:1; zoom:1` 用于兼容 IE6。

(3) 使用 `after` 伪对象清除浮动。

该方法只适用于非 IE 浏览器。具体写法可参照以下示例。使用中需注意以下几点。该方法中必须为需要清除浮动元素的伪对象中设置 `height:0`，否则该元素会比实际高出若干像素；

16. CSS 样式表根据所在网页的位置，可分为哪几种样式表？

参考答案：

行内样式表，内嵌样式表，外部样式表

17. 谈谈你对 CSS 中刻度的认识？

参考答案：

在 CSS 中刻度是用于设置元素尺寸的单位。

a、特殊值 0 可以省略单位。例如：`margin:0px` 可以写成 `margin:0`

b、一些属性可能允许有负长度值，或者有一定的范围限制。如果不支持负长度值，那应该变换到能够被支持的最近的一个长度值。

c、长度单位包括：相对单位和绝对单位。

相对长度单位有：em, ex, ch, rem, vw, vh, vmax, vmin

绝对长度单位有：cm, mm, q, in, pt, pc, px

绝对长度单位：1in = 2.54cm = 25.4 mm = 72pt = 6pc = 96px

文本相对长度单位：em

相对长度单位是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。（相对父元素的字体大小倍数）

```
body { font-size: 14px; }
```

```
h1 { font-size: 16px; }
```

```
.size1 p { font-size: 1em; }
```

```
.size2 p { font-size: 2em; }
```

```
.size3 p { font-size: 3em; }
```

文本相对长度单位：rem

rem 是 CSS3 新增的一个相对单位（root em，根 em），相对于根元素（即 html 元素）font-size 计算值的倍数

只相对于根元素的大小

浏览器的默认字体大小为 16 像素，浏览器默认样式也称为 user agent

stylesheet，就是所有浏览器内置的默认样式，多数是可以被修改的，但 chrome 不能直接修改，可以被用户样式覆盖。

18. 请你说说 em 与 rem 的区别？

参考答案：

rem :

rem 是 CSS3 新增的一个相对单位（root em，根 em），相对于根元素（即 html 元素）font-size 计算值的倍数

只相对于根元素的大小

rem (font size of the root element) 是指相对于根元素的字体大小的单位。简单的说它就是一个相对单位。

作用：利用 rem 可以实现简单的响应式布局，可以利用 html 元素中字体的大小与屏幕间的比值设置 font-size 的值实现当屏幕分辨率变化时让元素也变化，以前的天猫 tmall 就使用这种办法。

em :

文本相对长度单位。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(默认 16px)。(相对父元素的字体大小倍数)

em (font size of the element) 是指相对于父元素的字体大小的单位。它与 rem 之间其实很相似，区别在。(相对是的 HTML 元素的字体大，默认 16px)

em 与 rem 的重要区别： 它们计算的规则一个是依赖父元素另一个是依赖根元素计算。

19. 请你说说 box-sizing 属性的用法？

参考答案：

设置或检索对象的盒模型组成模式

a、box-sizing:content-box: padding 和 border 不被包含在定义的 width 和 height 之内。对象的实际宽度等于设置的 width 值和 border、padding 之和，即 (Element width = width + border + padding，但占有页面位置还要加上 margin) 此属性表现为标准模式下的盒模型。

b、box-sizing:border-box: padding 和 border 被包含在定义的 width 和 height 之内。对象的实际宽度就等于设置的 width 值，即使定义有 border 和 padding 也不会改变对象的实际宽度，即 (Element width = width) 此属性表现为怪异模式下的盒模型。

20. 浏览器标准模式和怪异模式之间的区别是什么？

参考答案：

所谓的标准模式是指，浏览器按 W3C 标准解析执行代码；怪异模式则是使用浏览器自己的方式解析执行代码，因为不同浏览器解析执行的方式不一样，所以我们称之为怪异模式。浏览器解析时到底使用标准模式还是怪异模式，与你网页中的 DTD 声明直接相关，DTD 声明定义了标准文档的类型（标准模式解析）文档类型，会使浏览器使用相应的方式加载网页并显示，忽略 DTD 声明，将使网页进入怪异模式(quirks mode)。

21. 怪异 Quirks 模式是什么，它和标准 Standards 模式有什么区别？

参考答案：

从 IE6 开始，引入了 Standards 模式，标准模式中，浏览器尝试给符合标准的文档在规范上的正确处理达到在指定浏览器中的程度。

在 IE6 之前 CSS 还不够成熟，所以 IE5 等之前的浏览器对 CSS 的支持很差，IE6 将对 CSS 提供更好的支持，然而这时的问题就来了，因为有很多页面是基于旧的布局方式写的，而如果 IE6 支持 CSS 则将令这些页面显示不正常，如何在即保证不破坏现有页面，又提供新的渲染机制呢？

在写程序时我们也会经常遇到这样的问题，如何保证原来的接口不变，又提供更强大的功能，尤其是新功能不兼容旧功能时。遇到这种问题时的一个常见做法是增加参数和分支，即当某个参数为真时，我们就使用新功能，而如果这个参数不为真时，就使用旧功能，这样就能不破坏原有的程序，又提供新功能。IE6 也是类似这样做的，它将 DTD 当成了这个“参数”，因为以前的页面大家都不会去写

DTD, 所以 IE6 就假定 如果写了 DTD, 就意味着这个页面将采用对 CSS 支持更好的布局, 而如果没有, 则采用兼容之前的布局方式。这就是 Quirks 模式 (怪癖模式, 诡异模式, 怪异模式)。

区别: 总体会有布局、样式解析和脚本执行三个方面的区别。

盒模型: 在 W3C 标准中, 如果设置一个元素的宽度和高度, 指的是元素内容的宽度和高度, 而在 Quirks 模式下, IE 的宽度和高度还包含了 padding 和 border。

设置行内元素的高宽: 在 Standards 模式下, 给 等行内元素设置 width 和 height 都不会生效, 而在 quirks 模式下, 则会生效。

设置百分比的高度: 在 standards 模式下, 一个元素的高度是由其包含的内容来决定的, 如果父元素没有设置百分比的高度, 子元素设置一个百分比的高度是无效的。用 margin:0 auto 设置水平居中: 使用 margin:0 auto 在 standards 模式下可以使元素水平居中, 但在 quirks 模式下却会失效。

(还有很多, 答出什么不重要, 关键是看他答出的这些是不是自己经验遇到的, 还是说都是看文章看的, 甚至完全不知道。)

22. 说说你对边距折叠的理解?

参考答案:

外边距折叠: 相邻的两个或多个外边距 (margin) 在垂直方向会合并成一个外边距 (margin)

相邻: 没有被非空内容、padding、border 或 clear 分隔开的 margin 特性。非空内容就是说这元素之间要么是兄弟关系或者父子关系

垂直方向外边距合并计算:

a、参加折叠的 margin 都是正值: 取其中 margin 较大的值为最终 margin 值。

b、参与折叠的 margin 都是负值：取的是其中绝对值较大的，然后，从 0 位置，负向位移。

c、参与折叠的 margin 中有正值，有负值：先取出负 margin 中绝对值中最大的，然后，和正 margin 值中最大的 margin 相加。

23. 内联与块级标签有何区别？

参考答案：

Html 中的标签默认主要分为两大类型，一类为块级元素，另一类是行内元素，许多人也把行内称为内联，所以叫内联元素，其实就是一个意思。为了很好的布局，必须理解它们间的区别。

24. 说说隐藏元素的方式有哪些？

参考答案：

a、使用 CSS 的 display:none，不会占有原来的位置

b、使用 CSS 的 visibility:hidden，会占有原来的位置

c、使用 HTML5 中的新增属性 hidden="hidden"，不会占有原来的位置

25. 为什么重置浏览器默认样式，如何重置默认浏览器样式？

参考答案：

每种浏览器都有一套默认的样式表，即 user agent stylesheet，网页在没有指定的样式时，按浏览器内置的样式表来渲染。这是合理的，像 word 中也有一些预留样式，可以让我们的排版更美观整齐。不同浏览器甚至同一浏览器不同版本的默认样式是不同的。但这样会有很多兼容问题。

- a、最简单的办法：（不推荐使用）*{margin: 0;padding: 0;}。
- b、使用 CSSReset 可以将所有浏览器默认样式设置成一样。
- c、normalize：也许有些 cssreset 过于简单粗暴，有点伤及无辜，normalize 是另一个选择。bootstrap 已经引用该 css 来重置浏览器默认样式，比普通的 cssreset 要精细一些，保留浏览器有用的默认样式，支持包括手机浏览器在内的超多浏览器，同时对 HTML5 元素、排版、列表、嵌入的内容、表单和表格都进行了一般化。

天猫 使用的 css reset 重置浏览器默认样式：

```
@charset "gb2312";
body, h1, h2, h3, h4, h5, h6, hr, p, blockquote, dl, dt, dd, ul, ol, li,
pre, form, fieldset, legend, button, input, textarea, th, td {
margin: 0;
padding: 0
}
body, button, input, select, textarea {
font: 12px "microsoft yahei";
line-height: 1.5;
-ms-overflow-style: scrollbar
}
h1, h2, h3, h4, h5, h6 {
font-size: 100%
}
ul, ol {
list-style: none
}
a {
text-decoration: none;
cursor:pointer
```



```
}  
a:hover {  
text-decoration: underline  
}  
img {  
border: 0  
}  
button, input, select, textarea {  
font-size: 100%  
}  
table {  
border-collapse: collapse;  
border-spacing: 0  
}  
.clear {  
clear: both  
}  
.fr {  
float: right  
}  
.fl {  
float: left  
}  
.block {  
display: block;  
text-indent: -999em  
}
```

26. 谈谈你对 BFC 与 IFC 的理解？（是什么，如何产生，作用）

参考答案：

(1)、什么是 BFC 与 IFC

a、BFC (Block Formatting Context) 即“块级格式化上下文”，IFC (Inline Formatting Context) 即行内格式化上下文。常规流（也称标准流、普通流）是一个文档在被显示时最常见的布局形态。一个框在常规流中必须属于一个格式化上下文，你可以把 BFC 想象成一个大箱子，箱子外边的元素将不与箱子内的元素产生作用。

b、BFC 是 W3C CSS 2.1 规范中的一个概念，它决定了元素如何对其内容进行定位，以及与其他元素的关系和相互作用。当涉及到可视化布局的时候，Block Formatting Context 提供了一个环境，HTML 元素在这个环境中按照一定规则进行布局。一个环境中的元素不会影响到其它环境中的布局。比如浮动元素会形成 BFC，浮动元素内部子元素的主要受该浮动元素影响，两个浮动元素之间是互不影响的。也可以说 BFC 就是一个作用范围。

c、在普通流中的 Box(框) 属于一种 formatting context(格式化上下文)，类型可以是 block，或者是 inline，但不能同时属于这两者。并且，Block boxes(块框) 在 block formatting context(块格式化上下文) 里格式化，Inline boxes(块内框) 则在 Inline Formatting Context(行内格式化上下文) 里格式化。

(2)、如何产生 BFC

当一个 HTML 元素满足下面条件的任何一点，都可以产生 Block Formatting Context:

- a、float 的值不为 none
- b、overflow 的值不为 visible
- c、display 的值为 table-cell, table-caption, inline-block 中的任何一个
- d、position 的值不为 relative 和 static

CSS3 触发 BFC 方式则可以简单描述为：在元素定位非 static, relative 的情况下触发，float 也是一种定位方式。

(3)、BFC 的作用与特点

a、不和浮动元素重叠，清除外部浮动，阻止浮动元素覆盖；

如果一个浮动元素后面跟着一个非浮动的元素，那么就会产生一个重叠的现象。

常规流（也称标准流、普通流）是一个文档在被显示时最常见的布局形态，当 float 不为 none 时，position 为 absolute、fixed 时元素将脱离标准流。

27. 说说你对页面中使用定位(position)的理解？

参考答案：

使用 css 布局 position 非常重要，语法如下：

position: static | relative | absolute | fixed | center | page | sticky

默认值: static, center、page、sticky 是 CSS3 中新增加的值。

(1)、static

可以认为静态的，默认元素都是静态的定位，对象遵循常规流。此时 4 个定位偏移属性不会被应用，也就是使用 left, right, bottom, top 将不会生效。

(2)、relative

相对定位，对象遵循常规流，并且参照自身在常规流中的位置通过 top, right, bottom, left 这 4 个定位偏移属性进行偏移时不会影响常规流中的任何元素。

(3)、absolute

a、绝对定位，对象脱离常规流，此时偏移属性参照的是离自身最近的定位祖先元素，如果没有定位的祖先元素，则一直回溯到 body 元素。盒子的偏移位置不影响常规流中的任何元素，其 margin 不与其他任何 margin 折叠。

b、元素定位参考的是离自身最近的定位祖先元素，要满足两个条件，第一个是自己的祖先元素，可以是父元素也可以是父元素的父元素，一直找，如果没有则选择 body 为对照对象。第二个条件是要求祖先元素必须定位，通俗说就是 position 的属性值为非 static 都行。

(4)、fixed

固定定位，与 absolute 一致，但偏移定位是以窗口为参考。当出现滚动条时，对象不会随着滚动。

(5)、center

与 absolute 一致，但偏移定位是以定位祖先元素的中心点为参考。盒子在其包含容器垂直水平居中。（CSS3）

(6)、page

与 absolute 一致。元素在分页媒体或者区域块内，元素的包含块始终是初始包含块，否则取决于每个 absolute 模式。（CSS3）

(7)、sticky

对象在常态时遵循常规流。它就像是 relative 和 fixed 的合体，当在屏幕中时按常规流排版，当滚动到屏幕外时则表现如 fixed。该属性的表现是现实中你见到的吸附效果。（CSS3）

28. 如何解决多个元素重叠问题？

参考答案：

使用 z-index 属性可以设置元素的层叠顺序

z-index 属性

语法：z-index: auto | <integer>

默认值：auto

适用于：定位元素。即定义了 position 为非 static 的元素

取值：

auto：元素在当前层叠上下文中的层叠级别是 0。元素不会创建新的局部层叠上下文，除非它是根元素。

整数：用整数值来定义堆叠级别。可以为负值。说明：

检索或设置对象的层叠顺序。

z-index 用于确定元素在当前层叠上下文中的层叠级别，并确定该元素是否创建新的局部层叠上下文。

当多个元素层叠在一起时，数字大者将显示在上面。

29. 页面布局的方式有哪些？

参考答案：

方式：双飞翼、多栏、弹性、流式、瀑布流、响应式布局

(1)、双飞翼布局

经典三列布局，也叫做圣杯布局【Holy Grail of Layouts】是 Kevin Cornell 在 2006 年提出的一个布局模型概念，在国内最早是由淘宝 UED 的工程师传播开来，在中国也有叫法是双飞翼布局，它的布局要求有几点：

- a、三列布局，中间宽度自适应，两边定宽；
- b、中间栏要在浏览器中优先展示渲染；
- c、允许任意列的高度最高；
- d、要求只用一个额外的 DIV 标签；
- e、要求用最简单的 CSS、最少的 HACK 语句；

在不增加额外标签的情况下，圣杯布局已经非常完美，圣杯布局使用了相对定位，以后布局是有局限性的，而且宽度控制要改的地方也多。在淘宝 UED（User Experience Design）探讨下，增加多一个 div 就可以不用相对布局了，只用到了浮动和负边距，这就是我们所说的双飞翼布局。

(2)、多栏布局

- a、栏栅格系统：就是利用浮动实现的多栏布局，在 bootstrap 中用的非常多。
- b、多列布局：栅格系统并没有真正实现分栏效果（如 word 中的分栏），CSS3 为了满足这个要求增加了多列布局模块

(3)、弹性布局 (Flexbox)

CSS3 引入了一种新的布局模式——Flexbox 布局，即伸缩布局盒模型 (Flexible Box)，用来提供一个更加有效的方式制定、调整和分布一个容器里项目布局，即使它们的大小是未知或者动态的，这里简称为 Flex。

Flexbox 布局常用于设计比较复杂的页面，可以轻松的实现屏幕和浏览器窗口大小发生变化时保持元素的相对位置和大小不变，同时减少了依赖于浮动布局实现元素位置的定义以及重置元素的大小。

Flexbox 布局在定义伸缩项目大小时伸缩容器会预留一些可用空间，让你可以调节伸缩项目的相对大小和位置。例如，你可以确保伸缩容器中的多余空间平均分配多个伸缩项目，当然，如果你的伸缩容器没有足够大的空间放置伸缩项目时，浏览器会根据一定的比例减少伸缩项目的大小，使其不溢出伸缩容器。

综合而言，Flexbox 布局功能主要具有以下几点：

- a、屏幕和浏览器窗口大小发生改变也可以灵活调整布局；
- b、可以指定伸缩项目沿着主轴或侧轴按比例分配额外空间(伸缩容器额外空间)，从而调整伸缩项目的大小；
- c、可以指定伸缩项目沿着主轴或侧轴将伸缩容器额外空间，分配到伸缩项目之前、之后或之间；
- d、可以指定如何将垂直于元素布局轴的额外空间分布到该元素的周围；
- e、可以控制元素在页面上的布局方向；
- f、可以按照不同于文档对象模型 (DOM) 所指定排序方式对屏幕上的元素重新排序。也就是说可以在浏览器渲染中不按照文档流先后顺序重排伸缩项目顺序。

(4)、瀑布流布局

瀑布流布局是流式布局的一种。是当下比较流行的一种网站页面布局，视觉表现为参差不齐的多栏布局，随着页面滚动条向下滚动，这种布局还会不断加载数据

块并附加至当前尾部。最早采用此布局的网站是 Pinterest，逐渐在国内流行开来。

优点：

- a、有效的降低了界面复杂度，节省了空间：我们不再需要臃肿复杂的页码导航链接或按钮了。
- b、对触屏设备来说，交互方式更符合直觉：在移动应用的交互环境当中，通过向上滑动进行滚屏的操作已经成为最基本的用户习惯，而且所需要的操作精准程度远远低于点击链接或按钮。
- c、更高的参与度：以上两点所带来的交互便捷性可以使用户将注意力更多的集中在内容而不是操作上，从而让他们更乐于沉浸在探索与浏览当中。

缺点：

a、有限的用例：

无限滚动的方式只适用于某些特定类型产品当中一部分特定类型的内容。

例如，在电商网站当中，用户时常需要在商品列表与详情页面之间切换，这种情况下，传统的、带有页码导航的方式可以帮助用户更稳妥和准确的回到某个特定的列表页面当中。

b、额外的复杂度：

那些用来打造无限滚动的 JS 库虽然都自称很容易使用，但你总会需要自己的产品中进行不同程度的定制化处理，以满足你们自己的需求；另外这些 JS 库在浏览器和设备兼容性等方面的表现也参差不齐，你必须做好充分的测试与调整工作。

c、再见了，页脚：

如果使用了比较典型的无限滚动加载模式，这就意味着你可以和页脚说拜拜了。最好考虑一下页脚对于你的网站，特别是用户的重要性；如果其中确实有比较重要的内容或链接，那么最好换一种更传统和稳妥的方式。

千万不要耍弄你的用户，当他们一次次的浏览到页面底部，看到页脚，却因为自动加载的内容突然出现而无论如何都无法点击页脚中的链接时，他们会变的越发愤怒。

d、集中在一页当中动态加载数据，与一页一页的输出相比，究竟那种方式更利于 SEO，这是你必须考虑的问题。对于某些以类型网站来说，在这方面进行冒险是很不划算的。

e、关于页面数量的印象：

其实站在用户的角度来看，这一点并非负面；不过，如果对于你的网站来说，通过更多的内容页面展示更多的相关信息(包括广告)是很重要的策略，那么单页无限滚动的方式对你并不适用。

(5)、流式布局 (Fluid)

固定布局和流式布局在网页设计中最常用的两种布局方式。固定布局能呈现网页的原始设计效果，流式布局则不受窗口宽度影响，流式布局使用百分比宽度来限定布局元素，这样可以根据客户端分辨率的大小来进行合理的显示。

(6)、响应式布局

响应式布局是 Ethan Marcotte 在 2010 年 5 月份提出的一个概念，简而言之，就是一个网站能够兼容多个终端——而不是为每个终端做一个特定的版本。这个概念是为了解决移动互联网浏览而诞生的。

响应式布局可以为不同终端的用户提供更加舒适的界面和更好的用户体验，而且随着目前大屏幕移动设备的普及，用“大势所趋”来形容也不为过。随着越来越多的设计师采用这个技术，我们不仅看到很多的创新，还看到了一些成形的模式。

优点：

- a、面对不同分辨率设备灵活性强
- b、能够快捷解决多设备显示适应问题

缺点：

- a、兼容各种设备工作量大，效率低下
- b、代码累赘，会出现隐藏无用的元素，加载时间加长
- c、其实这是一种折中性质的设计解决方案，多方面因素影响而达不到最佳效果
- d、一定程度上改变了网站原有的布局结构，会出现用户混淆的情况

30. overflow :hidden 是否形成新的块级格式化上下文？

参考答案：

会形成，触发 BFC 的条件有：

- float 的值不为 none。
- overflow 的值不为 visible。
- display 的值为 table-cell, table-caption, inline-block 中的任何一个。
- position 的值不为 relative 和 static。

第三章 前端基础

1. 说一下 http 和 https

参考答案：

https 的 SSL 加密是在传输层实现的。

(1)http 和 https 的基本概念

http: 超文本传输协议，是互联网上应用最为广泛的一种网络协议，是一个客户端和服务端请求和应答的标准（TCP），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。

https: 是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版，即 HTTP 下加入 SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL。

https 协议的主要作用是：建立一个信息安全通道，来确保数据的传输，确保网站的真实性。

(2)http 和 https 的区别？

http 传输的数据都是未加密的，也就是明文的，网景公司设置了 SSL 协议来对 http 协议传输的数据进行加密处理，简单来说 https 协议是由 http 和 ssl 协议构建的可进行加密传输和身份认证的网络协议，比 http 协议的安全性更高。

主要的区别如下：

- a. Https 协议需要 ca 证书，费用较高。
- b. 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。
- c. 使用不同的链接方式，端口也不同，一般而言，http 协议的端口为 80，https 的端口为 443
- d. 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。

(3)https 协议的工作原理

客户端在使用 HTTPS 方式与 Web 服务器通信时有几个步骤。

- a. 客户使用 https url 访问服务器，则要求 web 服务器建立 ssl 链接。
- b. web 服务器接收到客户端的请求之后，会将网站的证书（证书中包含了公钥），返回或者说传输给客户端。
- c. 客户端和 web 服务器端开始协商 SSL 链接的安全等级，也就是加密等级。

- d. 客户端浏览器通过双方协商一致的安全等级，建立会话密钥，然后通过网站的公钥来加密会话密钥，并传送给网站。
- e. web 服务器通过自己的私钥解密出会话密钥。
- f. web 服务器通过会话密钥加密与客户端之间的通信。

(4)https 协议的优点

使用 HTTPS 协议可认证用户和服务器，确保数据发送到正确的客户机和服务器；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，要比 http 协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。

HTTPS 是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。

谷歌曾在 2014 年 8 月份调整搜索引擎算法，并称“比起同等 HTTP 网站，采用 HTTPS 加密的网站在搜索结果中的排名将会更高”。

(5)https 协议的缺点

https 握手阶段比较费时，会使页面加载时间延长 50%，增加 10%~20%的耗电。

https 缓存不如 http 高效，会增加数据开销。

SSL 证书也需要钱，功能越强大的证书费用越高。

SSL 证书需要绑定 IP，不能再同一个 ip 上绑定多个域名，ipv4 资源支持不了这种消耗。

2. tcp 三次握手，一句话概括

参考答案：

客户端和服务端都需要直到各自可收发，因此需要三次握手。

简化三次握手：

```

```

从图片可以得到三次握手可以简化为：

C 发起请求连接 S 确认，也发起连接 C 确认

每次握手的作用：

第一次握手：S 只可以确认自己可以接受 C 发送的报文段；

第二次握手：C 可以确认 S 收到了自己发送的报文段，并且可以确认自己可以接受 S 发送的报文段；

第三次握手：S 可以确认 C 收到了自己发送的报文段；

3. TCP 和 UDP 的区别

参考答案：

(1) TCP 是面向连接的，udp 是无连接的即发送数据前不需要先建立链接。

(2) TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达；UDP 尽最大努力交付，即不保证可靠交付。并且因为 tcp 可靠，面向连接，不会丢失数据因此适合大数据量的交换。

(3) TCP 是面向字节流，UDP 面向报文，并且网络出现拥塞不会使得发送速率降低（因此会出现丢包，对实时的应用比如 IP 电话和视频会议等）。

(4) TCP 只能是 1 对 1 的，UDP 支持 1 对 1, 1 对多。

(5) TCP 的首部较大为 20 字节，而 UDP 只有 8 字节。

TCP 是面向连接的可靠性传输，而 UDP 是不可靠的。

4. WebSocket 的实现和应用

参考答案：

(1)什么是 WebSocket?

WebSocket 是 HTML5 中的协议，支持持久连续，http 协议不支持持久性连接。

Http1.0 和 HTTP1.1 都不支持持久性的链接，HTTP1.1 中的 keep-alive，将多个 http 请求合并为 1 个；

(2)WebSocket 是什么样的协议，具体有什么优点？

HTTP 的生命周期通过 Request 来界定，也就是 Request 一个 Response，那么在 Http1.0 协议中，这次 Http 请求就结束了。

在 Http1.1 中进行了改进，是的有一个 connection: Keep-alive，也就是说，在一个 Http 连接中，可以发送多个 Request，接收多个 Response。但是必须记住，在 Http 中一个 Request 只能对应有一个 Response，而且这个 Response 是被动的，不能主动发起。

WebSocket 是基于 Http 协议的，或者说借用了 Http 协议来完成一部分握手，在握手阶段与 Http 是相同的。我们来看一个 websocket 握手协议的实现，基本是 2 个属性，upgrade，connection。

基本请求如下：

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

多了下面 2 个属性：

```
Upgrade:websocket
```

Connection:Upgrade

告诉服务器发送的是 websocket

Sec-WebSocket-Key: x3JJHmbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

5. HTTP 请求的方式，HEAD 方式

参考答案：

head: 类似于 get 请求，只不过返回的响应中没有具体的内容，用户获取报头

options: 允许客户端查看服务器的性能，比如说服务器支持的请求方式等等。

6. 一个图片 url 访问后直接下载怎样实现？

参考答案：

请求的返回头里面，用于浏览器解析的重要参数就是 OSS 的 API 文档里面的返回 http 头，决定用户下载行为的参数。

下载的情况下：

1) x-oss-object-type:

Normal

2) x-oss-request-id:

598D5ED34F29D01FE2925F41

3) x-oss-storage-class:

Standard

7. 说一下 web Quality（无障碍）

参考答案：

能够被残障人士使用的网站才能称得上一个易用的（易访问的）网站。

残障人士指的是那些带有残疾或者身体不健康的用户。

使用 alt 属性：

```

```

有时候浏览器会无法显示图像。

具体的原因有：

- a. 用户关闭了图像显示
- b. 浏览器是不支持图形显示的迷你浏览器
- c. 浏览器是语音浏览器（供盲人和弱视人群使用）

如果您使用了 alt 属性，那么浏览器至少可以显示或读出有关图像的描述。

8. 几个很实用的 BOM 属性对象方法？

参考答案：

什么是 Bom？Bom 是浏览器对象。有哪些常用的 Bom 属性呢？

(1) location 对象

location.href -- 返回或设置当前文档的 URL

location.search -- 返回 URL 中的查询字符串部分。

例如 `http://www.dreamdu.com/dreamdu.php?id=5&name=dreamdu` 返回包括(?)
后面的内容 `?id=5&name=dreamdu`

location.hash -- 返回 URL#后面的内容，如果没有#，返回空

location.host -- 返回 URL 中的域名部分，

例如 `www.dreamdu.com`

location.hostname -- 返回 URL 中的主域名部分，

例如 `dreamdu.com`

location.pathname -- 返回 URL 的域名后的部分。

例如 `http://www.dreamdu.com/xhtml/` 返回 `/xhtml/`

`location.port` -- 返回 URL 中的端口部分。

例如 `http://www.dreamdu.com:8080/xhtml/` 返回 8080

`location.protocol` -- 返回 URL 中的协议部分。

例如 `http://www.dreamdu.com:8080/xhtml/` 返回 `(//)` 前面的内容 `http:`

`location.assign` -- 设置当前文档的 URL

`location.replace()` -- 设置当前文档的 URL，并且在 `history` 对象的地址列表中移除这个 URL `location.replace(url);`

`location.reload()` -- 重载当前页面

(2)history 对象

`history.go()` -- 前进或后退指定的页面数 `history.go(num);`

`history.back()` -- 后退一页

`history.forward()` -- 前进一页

(3)Navigator 对象

`navigator.userAgent` -- 返回用户代理头的字符串表示(就是包括浏览器版本信息等的字符串)

`navigator.cookieEnabled` -- 返回浏览器是否支持(启用)cookie

9. 说一下 HTML5 drag api

参考答案:

`dragstart`: 事件主体是被拖放元素，在开始拖放被拖放元素时触发，。

`darg`: 事件主体是被拖放元素，在正在拖放被拖放元素时触发。

`dragenter`: 事件主体是目标元素，在被拖放元素进入某元素时触发。

`dragover`: 事件主体是目标元素，在被拖放在某元素内移动时触发。

dragleave: 事件主体是目标元素, 在被拖放元素移出目标元素是触发。

drop: 事件主体是目标元素, 在目标元素完全接受被拖放元素时触发。

dragend: 事件主体是被拖放元素, 在整个拖放操作结束时触发

10. 说一下 http2.0

参考答案:

首先补充一下, http 和 https 的区别, 相比于 http, https 是基于 ssl 加密的 http 协议

简要概括: http2.0 是基于 1999 年发布的 http1.0 之后的首次更新。

提升访问速度(可以对于, 请求资源所需时间更少, 访问速度更快, 相比 http1.0)

允许多路复用: 多路复用允许同时通过单一的 HTTP/2 连接发送多重请求-响应信息。

改善了: 在 http1.1 中, 浏览器客户端在同一时间, 针对同一域名下的请求有一定数量限制(连接数量), 超过限制会被阻塞。

二进制分帧: HTTP2.0 会将所有的传输信息分割为更小的信息或者帧, 并对他们进行二进制编码。

首部压缩

服务器端推送

11. 补充 400 和 401、403 状态码

参考答案:

(1) 400 状态码: 请求无效

产生原因:

前端提交数据的字段名称和字段类型与后台的实体没有保持一致

前端提交到后台的数据应该是 json 字符串类型，但是前端没有将对象 JSON.stringify 转化成字符串。

解决方法：

对照字段的名称，保持一致性

将 obj 对象通过 JSON.stringify 实现序列化

(2) 401 状态码：当前请求需要用户验证

(3) 403 状态码：服务器已经得到请求，但是拒绝执行

12. fetch 发送 2 次请求的原因

参考答案：

fetch 发送 post 请求的时候，总是发送 2 次，第一次状态码是 204，第二次才成功？

原因很简单，因为你用 fetch 的 post 请求的时候，导致 fetch 第一次发送了一个 Options 请求，询问服务器是否支持修改的请求头，如果服务器支持，则在第二次中发送真正的请求。

13. Cookie、sessionStorage、localStorage 的区别

参考答案：

共同点：都是保存在浏览器端，并且是同源的

Cookie：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 不会自动把数据发给服务器，仅在本地保存。cookie 数据还有路径（path）的概念，可以限制 cookie 只属于某个路径下，存储的大小很小只有 4K 左右。（key：可以在浏览器和服务器端来回传递，存储容量小，只有大约 4K 左右）

sessionStorage: 仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持，
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 过期时间之前一直有效，即使窗口或浏览器关闭。

（key: 本身就是一个回话过程，关闭浏览器后消失，session 为一个回话，当页面不同即使是同一页面打开两次，也被视为同一次回话）

localStorage: 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。（key: 同源窗口都会共享，并且不会失效，不管窗口或者浏览器关闭与否都会始终生效）

补充说明一下 cookie 的作用：

（1）保存用户登录状态

例如将用户 id 存储于一个 cookie 内，这样当用户下次访问该页面时就不需要重新登录了，现在很多论坛和社区都提供这样的功能。

cookie 还可以设置过期时间，当超过时间期限后，cookie 就会自动消失。因此，系统往往可以提示用户保持登录状态的时间：常见选项有一个月、三个月、一年等。

（2）跟踪用户行为

例如一个天气预报网站，能够根据用户选择的地区显示当地的天气情况。如果每次都需要选择所在地是烦琐的，当利用了 cookie 后就会显得很人性化了，系统能够记住上一次访问的地区，当下次再打开该页面时，它就会自动显示上次用户所在地区的天气情况。

因为一切都是在后台完成，所以这样的页面就像为某个用户所定制的一样，使用起来非常方便定制页面。如果网站提供了换肤或更换布局的功能，那么可以使用 cookie 来记录用户的选项，例如：背景色、分辨率等。当用户下次访问时，仍然可以保存上一次访问的界面风格。

14. 说一下 web worker

参考答案：

在 HTML 页面中，如果在执行脚本时，页面的状态是不可相应的，直到脚本执行完成后，页面才变成可相应。web worker 是运行在后台的 js，独立于其他脚本，不会影响页面你的性能。并且通过 postMessage 将结果回传到主线程。这样在进行复杂操作的时候，就不会阻塞主线程了。

如何创建 web worker：

检测浏览器对于 web worker 的支持性

创建 web worker 文件（js，回传函数等）

创建 web worker 对象

15. 对 HTML 语义化标签的理解

参考答案：

HTML5 语义化标签是指正确的标签包含了正确的内容，结构良好，便于阅读，比如 nav 表示导航条，类似的还有 article、header、footer 等等标签。

16. iframe 是什么？有什么缺点？

参考答案：

定义：iframe 元素会创建包含另一个文档的内联框架

提示：可以将提示文字放在<iframe></iframe>之间，来提示某些不支持 iframe 的浏览器

缺点：

- a. 会阻塞主页面的 onload 事件
- b. 搜索引擎无法解读这种页面，不利于 SEO
- c. iframe 和主页面共享连接池，而浏览器对相同区域有限制所以会影响性能。

17. Doctype 作用?严格模式与混杂模式如何区分? 它们有何意义?

参考答案:

Doctype 声明于文档最前面, 告诉浏览器以何种方式来渲染页面, 这里有两种模式, 严格模式和混杂模式。

严格模式的排版和 JS 运作模式是 以该浏览器支持的最高标准运行。

混杂模式, 向后兼容, 模拟老式浏览器, 防止浏览器无法兼容页面。

18. Cookie 如何防范 XSS 攻击

参考答案:

XSS (跨站脚本攻击) 是指攻击者在返回的 HTML 中嵌入 javascript 脚本, 为了减轻这些攻击, 需要在 HTTP 头部配上, set-cookie:

httponly-这个属性可以防止 XSS, 它会禁止 javascript 脚本来访问 cookie。

secure - 这个属性告诉浏览器仅在请求为 https 的时候发送 cookie。

结果应该是这样的: Set-Cookie=<cookie-value>.....

19. Cookie 和 session 的区别

参考答案:

HTTP 是一个无状态协议, 因此 Cookie 的最大的作用就是存储 sessionId 用来唯一标识用户。

20. 一句话概括 RESTFUL

参考答案:

就是用 URL 定位资源, 用 HTTP 描述操作

21. 讲讲 viewport 和移动端布局

参考答案：

前端开发中，静态网页通常需要适应不同分辨率的设备，常用的自适应解决方案包括媒体查询、百分比、rem 和 vw/vh 等。下面从 px 单位出发，分析 px 在移动端布局中的不足，接着介绍几种不同的自适应解决方案。

- px 和视口
- 媒体查询
- 百分比
- 自适应场景下的 rem 解决方案
- 通过 vw/vh 来实现自适应

一、px 和视口

1) 像素

像素是网页布局的基础，一个像素表示了计算机屏幕所能显示的最小区域

像素分为两种类型：css 像素和物理像素。

我们在 js 或者 css 代码中使用的 px 单位就是指的就是 css 像素，物理像素也称设备像素，只与设备或者说硬件有关，同样尺寸的屏幕，设备的密度越高，物理像素也就越多。

下表表示 css 像素和物理像素的具体区别：

css 像素 为 web 开发者提供，在 css 中使用的一个抽象单位

物理像素 只与设备的硬件密度有关，任何设备的物理像素都是固定的

那么 css 像素与物理像素的转换关系是怎么样的呢？

为了明确 css 像素和物理像素的转换关系，必须先了解视口是什么。

2) 视口

广义的视口，是指浏览器显示内容的屏幕区域，狭义的视口包括了布局视口、视觉视口和理想视口

(1) 布局视口 (*layout viewport*)

布局视口定义了 pc 网页在移动端的默认布局行为，因为通常 pc 的分辨率较大，布局视口默认为 980px。也就是说在不设置网页的 viewport 的情况下，pc 端的网页默认会以布局视口为基准，在移动端进行展示。因此我们可以明显看出来，默认为布局视口时，根植于 pc 端的网页在移动端展示很模糊。

(2) 视觉视口 (*visual viewport*)

视觉视口表示浏览器内看到的网站的显示区域，用户可以通过缩放来查看网页的显示内容，从而改变视觉视口。视觉视口的定义，就像拿着一个放大镜分别从不同距离观察同一个物体，视觉视口仅仅类似于放大镜中显示的内容，因此视觉视口不会影响布局视口的宽度和高度。

(3) 理想视口 (*ideal viewport*)

理想视口或者应该全称为“理想的布局视口”，在移动设备中就是指设备的分辨率。换句话说，理想视口或者说分辨率就是给定设备物理像素的情况下，最佳的“布局视口”。

上述视口中，最重要的是要明确理想视口的概念，在移动端中，理想视口或者说分辨率跟物理像素之间有什么关系呢？

为了理清分辨率和物理像素之间的联系，我们介绍一个用 DPR (Device pixel ratio) 设备像素比来表示，则可以写成：

$$\text{DPR} = \text{物理像素} / \text{分辨率}$$

在不缩放的情况下，一个 css 像素就对应一个 dpr，也就是说，在不缩放

CSS 像素 = 物理像素 / 分辨率

此外，在移动端的布局中，我们可以通过 viewport 元标签来控制布局，比如一般情况下，我们可以通过下述标签使得移动端在理想视口下布局：

```
<meta id="viewport" name="viewport" content="width=device-width;
initial-scale=1.0; maximum-scale=1; user-scalable=no;">
```

上述 meta 标签的每一个属性的详细介绍如下：

属性名	取值	描述
width	正整数	定义布局视口的宽度，单位为像素
height	正整数	定义布局视口的高度，单位为像素，很少使用
initial-scale	[0, 10]	初始缩放比例，1 表示不缩放
minimum-scale	[0, 10]	最小缩放比例
maximum-scale	[0, 10]	最大缩放比例
user-scalable	yes / no	是否允许手动缩放页面，默认值为 yes

其中我们来看 width 属性，在移动端布局时，在 meta 标签中我们会将 width 设置称为 device-width，device-width 一般是表示分辨率的宽，通过 width=device-width 的设置我们就将布局视口设置成了理想的视口。

3) px 与自适应

上述我们了解到了当通过 viewport 元标签，设置布局视口为理想视口时，1 个 css 像素可以表示成：

CSS 像素 = 物理像素 / 分辨率

我们直到，在 pc 端的布局视口通常情况下为 980px，移动端以 iphone6 为例，分辨率为 375* 667，也就是说布局视口在理想的情况下为 375px。

比如现在我们有一个 750px*1134px 的视觉稿，那么在 pc 端，一个 css 像素可以如下计算：

PC 端：CSS 像素 = 物理像素 / 分辨率 = 750 / 980 = 0.76 px

而在 iphone6 下：

iphone6：CSS 像素 = 物理像素 / 分辨率 = 750 / 375 = 2 px

也就是说在 PC 端，一个 CSS 像素可以用 0.76 个物理像素来表示，而 iphone6 中 一个 CSS 像素表示了 2 个物理像素。此外不同的移动设备分辨率不同，也就是 1 个 CSS 像素可以表示的物理像素是不同的，因此如果在 css 中仅仅通过 px 作为长度和宽度的单位，造成的结果就是无法通过一套样式，实现各端的自适应。

二、媒体查询

在前面我们说到，不同端的设备下，在 css 文件中，1px 所表示的物理像素的大小是不同的，因此通过一套样式，是无法实现各端的自适应。由此我们联想：如果一套样式不行，那么能否给每一种设备各一套不同的样式来实现自适应的效果？

答案是肯定的。

使用 @media 媒体查询可以针对不同的媒体类型定义不同的样式，特别是响应式页面，可以针对不同屏幕的大小，编写多套样式，从而达到自适应的效果。

举例来说：

```
@media screen and (max-width: 960px) {
```

```
body{
    background-color:#FF6699
}
}
@media screen and (max-width: 768px){
    body{
        background-color:#00FF66;
    }
}
@media screen and (max-width: 550px){
    body{
        background-color:#6633FF;
    }
}
@media screen and (max-width: 320px){
    body{
        background-color:#FFFF00;
    }
}
```

上述的代码通过媒体查询定义了几套样式,通过 max-width 设置样式生效时的最大分辨率,上述的代码分别对分辨率在 0~320px, 320px~550px, 550px~768px 以及 768px~960px 的屏幕设置了不同的背景颜色。

通过媒体查询,可以通过给不同分辨率的设备编写不同的样式来实现响应式的布局,比如我们为不同分辨率的屏幕,设置不同的背景图片。比如给小屏幕手机设置@2x 图,为大屏幕手机设置@3x 图,通过媒体查询就能很方便的实现。

但是媒体查询的缺点也很明显,如果在浏览器大小改变时,需要改变的样式太多,那么多套样式代码会很繁琐。

三、百分比

除了用 px 结合媒体查询实现响应式布局外，我们也可以通过百分比单位“%”来实现响应式的效果。

比如当浏览器的宽度或者高度发生变化时，通过百分比单位，通过百分比单位可以使得浏览器中的组件的宽和高随着浏览器的变化而变化，从而实现响应式的效果。

为了了解百分比布局，首先要了解的问题是：

css 中的子元素中的百分比（%）到底是谁的百分比？

直观的理解，我们可能会认为子元素的百分比完全相对于直接父元素，height 百分比相对于 height，width 百分比相对于 width。当然这种理解是正确的，但是根据 css 的盒式模型，除了 height、width 属性外，还具有 padding、border、margin 等等属性。那么这些属性设置成百分比，是根据父元素的那些属性呢？此外还有 border-radius 和 translate 等属性中的百分比，又是相对于什么呢？下面来具体分析。

1) 百分比的具体分析

(1) 子元素 height 和 width 的百分比

子元素的 height 或 width 中使用百分比，是相对于子元素的直接父元素，width 相对于父元素的 width，height 相对于父元素的 height。

比如：

```
<div class="parent">
  <div class="child"></div>
</div>
```

如果设置：

```
.father{
width:200px;
height:100px;
}
.child{
```

```
width:50%;  
height:50%;  
}
```

(2) *top 和 bottom 、 left 和 right*

子元素的 *top* 和 *bottom* 如果设置百分比，则相对于直接非 *static* 定位(默认定位)的父元素的高度，同样子元素的 *left* 和 *right* 如果设置百分比，则相对于直接非 *static* 定位(默认定位的)父元素的宽度。

(3) *padding*

子元素的 *padding* 如果设置百分比，不论是垂直方向或者是水平方向，都相对于直接父亲元素的 *width*，而与父元素的 *height* 无关。

举例来说：

```
.parent{  
  width:200px;  
  height:100px;  
  background:green;  
}  
.child{  
  width:0px;  
  height:0px;  
  background:blue;  
  color:white;  
  padding-top:50%;  
  padding-left:50%;  
}
```

子元素的初始宽高为 0，通过 *padding* 可以将父元素撑大，上图的蓝色部分是一个正方形，且边长为 100px，说明 *padding* 不论宽高，如果设置成百分比都相对于父元素的 *width*。

(4) *margin*

跟 padding 一样，margin 也是如此，子元素的 margin 如果设置成百分比，不论是垂直方向还是水平方向，都相对于直接父元素的 width。这里就不具体举例。

(5) *border-radius*

border-radius 不一样，如果设置 border-radius 为百分比，则是相对于自身的宽度，

举例来说：

```
<div class="trangle"></div>
```

设置 border-radius 为百分比：

```
.trangle{  
  width:100px;  
  height:100px;  
  border-radius:50%;  
  background:blue;  
  margin-top:10px;  
}
```

除了 border-radius 外，还有比如 translate、background-size 等都是相对于自身的，这里就不一一举例。

2) 百分比单位布局应用

百分比单位在布局上应用还是很广泛的，这里介绍一种应用。

比如我们要实现一个固定长宽比的长方形，比如要实现一个长宽比为 4:3 的长方形，我们可以根据 padding 属性来实现，因为 padding 不管是垂直方向还是水平方向，百分比单位都相对于父元素的宽度，因此我们可以设置 padding-top 为百分比来实现

长宽自适应的长方形：

```
<div class="triangle"></div>
```

设置样式让其自适应:

```
.triangle{  
  height:0;  
  width:100%;  
  padding-top:75%;  
}
```

通过设置 padding-top: 75%, 相对比宽度的 75%, 因此这样就设置了一个长宽高恒定比例的长方形, 具体效果展示如下:

3) 百分比单位缺点

从上述对于百分比单位的介绍我们很容易看出如果全部使用百分比单位来实现响应式的布局, 有明显的以下两个缺点:

(1) 计算困难, 如果我们要定义一个元素的宽度和高度, 按照设计稿, 必须换算成百分比单位。

(2) 从小节 1 可以看出, 各个属性中如果使用百分比, 相对父元素的属性并不是唯一的。

比如 width 和 height 相对于父元素的 width 和 height, 而 margin、padding 不管垂直还是水平方向都相对比父元素的宽度、border-radius 则是相对于元素自身等等, 造成我们使用百分比单位容易使布局问题变得复杂。

四、自适应场景下的 rem 解决方案

1) rem 单位

首先来看, 什么是 rem 单位。

rem 是一个灵活的、可扩展的单位, 由浏览器转化像素并显示。与 em 单位不同, rem 单位无论嵌套层级如何, 都只相对于浏览器的根元素 (HTML 元素) 的 font-size。默认情况下, html 元素的 font-size 为 16px,

所以: $\text{rem} = 16\text{px}$

为了计算方便, 通常可以将 html 的 font-size 设置成:

```
html {  
  font-size: 62.5%;  
}
```

这种情况下: $\text{rem} = 10\text{px}$

2) 通过 rem 来实现响应式布局

rem 单位都是相对于根元素 html 的 font-size 来决定大小的, 根元素的 font-size 相当于提供了一个基准, 当页面的 size 发生变化时, 只需要改变 font-size 的值, 那么以 rem 为固定单位的元素的大小也会发生响应的变化。因此, 如果通过 rem 来实现响应式的布局, 只需要根据视图容器的大小, 动态的改变 font-size 即可。

```
function refreshRem() {  
  var docEl = doc.documentElement;  
  var width = docEl.getBoundingClientRect().width;  
  var rem = width / 10;  
  docEl.style.fontSize = rem + 'px';  
  flexible.rem = win.rem = rem;  
}  
  
win.addEventListener('resize', refreshRem);
```

上述代码中将视图容器分为 10 份, font-size 用十分之一的宽度来表示, 最后在 header 标签中执行这段代码, 就可以动态定义 font-size 的大小, 从而 1rem 在不同的视觉容器中表示不同的大小, 用 rem 固定单位可以实现不同容器内布局的自适应。

3) rem2px 和 px2rem

如果在响应式布局中使用 rem 单位，那么存在一个单位换算的问题，rem2px 表示从 rem 换算成 px，这个就不说了，只要 rem 乘以相应的 font-size 中的大小，就能换算成 px。更多的应用是 px2rem，表示的是从 px 转化为 rem。

比如给定的视觉稿为 750px（物理像素），如果我们要将所有的布局单位都用 rem 来表示，一种比较笨的办法就是对所有的 height 和 width 等元素，乘以相应的比例，现将视觉稿换算成 rem 单位，然后一个个的用 rem 来表示。另一种比较方便的解决方法就是，在 css 中我们还是用 px 来表示元素的大小，最后编写完 css 代码之后，将 css 文件中的所有 px 单位，转化成 rem 单位。

px2rem 的原理也很简单，重点在于预处理以 px 为单位的 css 文件，处理后将所有的 px 变成 rem 单位。

可以通过两种方式来实现：

（1）webpack loader 的形式：

```
npm install px2rem-loader
```

在 webpack 的配置文件中：

```
module.exports = {  
  // ...  
  module: {  
    rules: [{  
      test: /\.css$/,  
      use: [{  
        loader: 'style-loader'  
      }, {  
        loader: 'css-loader'  
      }, {  
        loader: 'px2rem-loader',  
        // options here  
        options: {  
          remUni: 75,  
          remPrecision: 8  
        }  
      }  
    ]  
  }  
}
```



```
    }  
  }]  
}]  
}  
}
```

(2) webpack 中使用 postcss plugin

npm install postcss-loader

在 webpack 的 plugin 中:

```
var px2rem = require('postcss-px2rem');  
module.exports = {  
  module: {  
    loaders: [  
      {  
        test: /\.css$/,  
        loader: "style-loader!css-loader!postcss-loader"  
      }  
    ],  
    postcss: function() {  
      return [px2rem({remUnit: 75})];  
    }  
  }  
}
```

4) rem 布局应用举例

网易新闻的移动端页面使用了 rem 布局，具体例子如下：

5. rem 布局的缺点

通过 rem 单位，可以实现响应式的布局，特别是引入相应的 postcss 相关插件，免去了设计稿中的 px 到 rem 的计算。rem 单位在国外的一些网站也有使用，这里所说的 rem 来实现布局的缺点，或者说是小缺陷是：

在响应式布局中，必须通过 js 来动态控制根元素 font-size 的大小。

也就是说 css 样式和 js 代码有一定的耦合性。且必须将改变 font-size 的代码放在 css 样式之前。

五. 通过 vw/vh 来实现自适应

1) 什么是 vw/vh ?

css3 中引入了一个新的单位 vw/vh, 与视图窗口有关, vw 表示相对于视图窗口的宽度, vh 表示相对于视图窗口高度, 除了 vw 和 vh 外, 还有 vmin 和 vmax 两个相关的单位。

各个单位具体的含义如下:

单位	含义
----	----

vw	相对于视窗的宽度, 视窗宽度是 100vw
----	-----------------------

vh	相对于视窗的高度, 视窗高度是 100vh
----	-----------------------

vmin	vw 和 vh 中的较小值
------	---------------

vmax	vw 和 vh 中的较大值
------	---------------

这里我们发现视窗宽高都是 100vw / 100vh, 那么 vw 或者 vh, 下简称 vw, 很类似百分比单位。

vw 和%的区别为:

单位	含义
----	----

%	大部分相对于祖先元素, 也有相对于自身的情况比如 (border-radius、translate 等)
---	--

vw/vh	相对于视窗的尺寸
-------	----------

从对比中我们可以发现，vw 单位与百分比类似，单确有区别，前面我们介绍了百分比单位的换算困难，这里的 vw 更像“理想的百分比单位”。任意层级元素，在使用 vw 单位的情况下，1vw 都等于视图宽度的百分之一。

2) vw 单位换算

同样的，如果要将 px 换算成 vw 单位，很简单，只要确定视图的窗口大小（布局视口），如果我们将布局视口设置成分辨率大小，比如对于 iphone6/7 375*667 的分辨率，那么 px 可以通过如下方式换算成 vw：

$$1\text{px} = (1/375) * 100 \text{vw}$$

此外，也可以通过 postcss 的相应插件，预处理 css 做一个自动的转换，

postcss-px-to-viewport 可以自动将 px 转化成 vw。

postcss-px-to-viewport 的默认参数为：

```
var defaults = {  
  viewportWidth: 320,  
  viewportHeight: 568,  
  unitPrecision: 5,  
  viewportUnit: 'vw',  
  selectorBlackList: [],  
  minPixelValue: 1,  
  mediaQuery: false  
};
```

通过指定视窗的宽度和高度，以及换算精度，就能将 px 转化成 vw。

3) vw/vh 单位的兼容性

可以在 <https://caniuse.com/> 查看各个版本的浏览器对 vw 单位的支持性。

绝大多数的浏览器支持 vw 单位，但是 ie9-11 不支持 vmin 和 vmax，考虑到 vmin 和 vmax 单位不常用，vw 单位在绝大部分高版本浏览器内的支持性很好，但是 opera 浏览器整体不支持 vw 单位，如果需要兼容 opera 浏览器的布局，不推荐使用 vw。

22. click 在 ios 上有 300ms 延迟，原因及如何解决？

参考答案：

(1)粗暴型，禁用缩放

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

(2)利用 FastClick，其原理是：

检测到 touchend 事件后，立刻出发模拟 click 事件，并且把浏览器 300 毫秒之后真正出发的事件给阻断掉

23. addEventListener 参数

参考答案：

```
addEventListener(event, function, useCapture)
```

其中，event 指定事件名；function 指定要事件触发时执行的函数；useCapture 指定事件是否在捕获或冒泡阶段执行。

24. cookie sessionStorage localStorage 区别

参考答案：

cookie 数据始终在同源的 http 请求中携带(即使不需要)，即 cookie 在浏览器和服务端间来回传递。

cookie 数据还有路径(path)的概念，可以限制。cookie 只属于某个路径下。

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, 如回话标识;

webStorage 虽然也有存储大小的限制, 但是比 cookie 大得多, 可以达到 5M 或更大;

数据的有效期不同

sessionStorage: 仅在当前的浏览器窗口关闭有效;

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 过期时间之前一直有效, 即使窗口和浏览器关闭;

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: 在所有同源窗口都是共享的;

cookie: 也是在所有同源窗口中共享的

25. cookie session 区别

参考答案:

- (1) cookie 数据存放在客户的浏览器上, session 数据放在服务器上。
- (2) cookie 不是很安全, 别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗
- (3) 考虑到安全应当使用 session。
- (4) session 会在一定时间内保存在服务器上。当访问增多, 会比较占用你服务器的性能, 考虑到减轻服务器性能方面, 应当使用 COOKIE。
- (5) 单个 cookie 保存的数据不能超过 4K, 很多浏览器都限制一个站点最多保存 20 个 cookie。

26. 介绍知道的 http 返回的状态码

参考答案:

- 100 Continue 继续。客户端应继续其请求
- 101 Switching Protocols 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到 HTTP 的新版本协议
- 200 OK 请求成功。一般用于 GET 与 POST 请求
- 201 Created 已创建。成功请求并创建了新的资源
- 202 Accepted 已接受。已经接受请求，但未处理完成
- 203 Non-Authoritative Information 非授权信息。请求成功。但返回的 meta 信息不在原始的服务器，而是一个副本
- 204 No Content 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
- 205 Reset Content 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
- 206 Partial Content 部分内容。服务器成功处理了部分 GET 请求
- 300 Multiple Choices 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
- 301 Moved Permanently 永久移动。请求的资源已被永久的移动到新 URI，返回信息会包括新的 URI，浏览器会自动定向到新 URI。今后任何新的请求都应使用新的 URI 代替
- 302 Found 临时移动。与 301 类似。但资源只是临时被移动。客户端应继续使用原有 URI
- 303 See Other 查看其它地址。与 301 类似。使用 GET 和 POST 请求查看
- 304 Not Modified 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
- 305 Use Proxy 使用代理。所请求的资源必须通过代理访问
- 306 Unused 已经被废弃的 HTTP 状态码

- 307 Temporary Redirect 临时重定向。与 302 类似。使用 GET 请求重定向
- 400 Bad Request 客户端请求的语法错误，服务器无法理解
- 401 Unauthorized 请求要求用户的身份认证
- 402 Payment Required 保留，将来使用
- 403 Forbidden 服务器理解请求客户端的请求，但是拒绝执行此请求
- 404 Not Found 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置“您所请求的资源无法找到”的个性页面
- 405 Method Not Allowed 客户端请求中的方法被禁止
- 406 Not Acceptable 服务器无法根据客户端请求的内容特性完成请求
- 407 Proxy Authentication Required 请求要求代理的身份认证，与 401 类似，但请求者应当使用代理进行授权
- 408 Request Time-out 服务器等待客户端发送的请求时间过长，超时
- 409 Conflict 服务器完成客户端的 PUT 请求是可能返回此代码，服务器处理请求时发生了冲突
- 410 Gone 客户端请求的资源已经不存在。410 不同于 404，如果资源以前有现在被永久删除了可使用 410 代码，网站设计人员可通过 301 代码指定资源的新位置
- 411 Length Required 服务器无法处理客户端发送的不带 Content-Length 的请求信息
- 412 Precondition Failed 客户端请求信息的先决条件错误
- 413 Request Entity Too Large 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个 Retry-After 的响应信息
- 414 Request-URI Too Large 请求的 URI 过长（URI 通常为网址），服务器无法处理
- 415 Unsupported Media Type 服务器无法处理请求附带的媒体格式

- 416 Requested range not satisfiable 客户端请求的范围无效
- 417 Expectation Failed 服务器无法满足 Expect 的请求头信息
- 500 Internal Server Error 服务器内部错误，无法完成请求
- 501 Not Implemented 服务器不支持请求的功能，无法完成请求
- 502 Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应
- 503 Service Unavailable 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的 Retry-After 头信息中
- 504 Gateway Time-out 充当网关或代理的服务器，未及时从远端服务器获取请求
- 505 HTTP Version not supported 服务器不支持请求的 HTTP 协议的版本，无法完成处理

27. http 常用请求头

参考答案：

协议头	说明
Accept	可接受的响应内容类型 (Content-Types)
Accept-Charset	可接受的字符集
Accept-Encoding	可接受的响应内容的编码方式
Accept-Language	可接受的响应内容语言列表
Accept-Datetime	可接受的按照时间来表示的响应内容 版本
Authorization	用于表示 HTTP 协议中需要认证资源的 认证信息
Cache-Control	用来指定当前的请求/回复中的，是否 使用缓存机制
Connection	客户端（浏览器）想要优先使用的连接

	类型
Cookie	由之前服务器通过 Set-Cookie（见下文）设置的一个 HTTP 协议 Cookie
Content-Length	以 8 进制表示的请求体的长度
Content-MD5	请求体的内容的二进制 MD5 散列值（数字签名），以 Base64 编码的结果
Content-Type	请求体的 MIME 类型（用于 POST 和 PUT 请求中）
Date	发送该消息的日期和时间（以 RFC 7231 中定义的“HTTP 日期”格式来发送）
Expect	表示客户端要求服务器做出特定的行为
From	发起此请求的用户的邮件地址
Host	表示服务器的域名以及服务器所监听的端口号。如果所请求的端口是对应的服务的标准端口（80），则端口号可以省略
If-Match	仅当客户端提供的实体与服务器上对应的实体相匹配时，才进行对应的操作。主要用于像 PUT 这样的方法中，仅当从用户上次更新某个资源后，该资源未被修改的情况下，才更新该资源
If-Modified-Since	允许在对应的资源未被修改的情况下返回 304 未修改
If-None-Match	允许在对应的内容未被修改的情况下返回 304 未修改（ 304 Not Modified ），参考 超文本传输协议 的实体标记
If-Range	如果该实体未被修改过，则向返回所缺少的那一个或多个部分。否则，返回整个新的实体
If-Unmodified-Since	仅当该实体自某个特定时间以来未被

	修改的情况下，才发送回应
Max-Forwards	限制该消息可被代理及网关转发的次数
Origin	发起一个针对跨域资源共享的请求（该请求要求服务器在响应中加入一个 Access-Control-Allow-Origin 的消息头，表示访问控制所允许的来源）
Pragma	与具体的实现相关，这些字段可能在请求/回应链中的任何时候产生
Proxy-Authorization	用于向代理进行认证的认证信息
Range	表示请求某个实体的一部分，字节偏移以 0 开始
Referer	表示浏览器所访问的前一个页面，可以认为是之前访问页面的链接将浏览器带到了当前页面。Referer 其实是 Referrer 这个单词，但 RFC 制作标准时给拼错了，后来也就将错就错使用 Referer 了
TE	浏览器预期接受的传输时的编码方式：可使用回应协议头 Transfer-Encoding 中的值（还可以使用“trailers”表示数据传输时的分块方式）用来表示浏览器希望在最后一个大小为 0 的块之后还接收到一些额外的字段
User-Agent	浏览器的身份标识字符串
Upgrade	要求服务器升级到一个高版本协议
Via	告诉服务器，这个请求是由哪些代理发出的
Warning	一个一般性的警告，表示在实体内容体

	中可能存在错误
--	---------

28. 强，协商缓存

参考答案：

缓存分为两种：强缓存和协商缓存，根据响应的 header 内容来决定。

	获取资源形式	状态码	发送请求到服务器
强缓存	从缓存取	200 (from cache)	否，直接从缓存取
协商缓存	从缓存取	304 (not modified)	是，通过服务器来告知缓存是否可用

强缓存相关字段有 expires, cache-control。如果 cache-control 与 expires 同时存在的话，cache-control 的优先级高于 expires。

协商缓存相关字段有 Last-Modified/If-Modified-Since, Etag/If-None-Match

29. HTTP 状态码说说你知道的

参考答案：

200 OK 请求成功。一般用于 GET 与 POST 请求

201 Created 已创建。成功请求并创建了新的资源

202 Accepted 已接受。已经接受请求，但未处理完成

203 Non-Authoritative Information 非授权信息。请求成功。但返回的 meta 信息不在原始的服务器，而是一个副本

204 No Content 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档

- 205 Reset Content 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
- 206 Partial Content 部分内容。服务器成功处理了部分 GET 请求
- 300 Multiple Choices 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
- 301 Moved Permanently 永久移动。请求的资源已被永久的移动到新 URI，返回信息会包括新的 URI，浏览器会自动定向到新 URI。今后任何新的请求都应使用新的 URI 代替
- 302 Found 临时移动。与 301 类似。但资源只是临时被移动。客户端应继续使用原有 URI
- 303 See Other 查看其它地址。与 301 类似。使用 GET 和 POST 请求查看
- 304 Not Modified 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
- 305 Use Proxy 使用代理。所请求的资源必须通过代理访问
- 306 Unused 已经被废弃的 HTTP 状态码
- 307 Temporary Redirect 临时重定向。与 302 类似。使用 GET 请求重定向
- 400 Bad Request 客户端请求的语法错误，服务器无法理解
- 401 Unauthorized 请求要求用户的身份认证
- 402 Payment Required 保留，将来使用
- 403 Forbidden 服务器理解请求客户端的请求，但是拒绝执行此请求
- 404 Not Found 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置“您所请求的资源无法找到”的个性页面
- 500 Internal Server Error 服务器内部错误，无法完成请求
- 501 Not Implemented 服务器不支持请求的功能，无法完成请求

502 Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应

503 Service Unavailable 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的 Retry-After 头信息中

504 Gateway Time-out 充当网关或代理的服务器，未及时从远端服务器获取请求

505 HTTP Version not supported 服务器不支持请求的 HTTP 协议的版本，无法完成处理

30. 讲讲 304

参考答案：

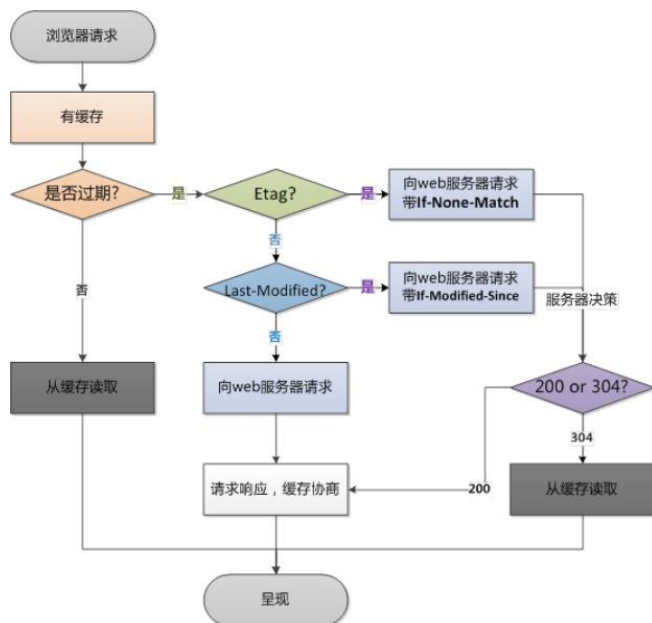
304：如果客户端发送了一个带条件的 GET 请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个 304 状态码。

31.强缓存、协商缓存什么时候用哪个

参考答案：

因为服务器上的资源不是一直固定不变的，大多数情况下它会更新，这个时候如果我们还访问本地缓存，那么对用户来说，那就相当于资源没有更新，用户看到的还是旧的资源；所以我们希望服务器上的资源更新了浏览器就请求新的资源，

没有更新就使用本地的缓存，以最大程度的减少因网络请求而产生的资源浪费。



32. 前端优化

参考答案：

降低请求量：合并资源，减少 HTTP 请求数，minify / gzip 压缩，webP, lazyLoad。

加快请求速度：预解析 DNS，减少域名数，并行加载，CDN 分发。

缓存：HTTP 协议缓存请求，离线缓存 manifest，离线数据缓存 localStorage。

渲染：JS/CSS 优化，加载顺序，服务端渲染，pipeline。

33. GET 和 POST 的区别

参考答案：

get 参数通过 url 传递，post 放在 request body 中。

get 请求在 url 中传递的参数是有长度限制的，而 post 没有。

get 比 post 更不安全，因为参数直接暴露在 url 中，所以不能用来传递敏感信息。

get 请求只能进行 url 编码，而 post 支持多种编码方式

get 请求会浏览器主动 cache，而 post 支持多种编码方式。

get 请求参数会被完整保留在浏览历史记录里，而 post 中的参数不会被保留。

GET 和 POST 本质上就是 TCP 链接，并无差别。但是由于 HTTP 的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

GET 产生一个 TCP 数据包；POST 产生两个 TCP 数据包。

34. 301 和 302 的区别

参考答案：

301 Moved Permanently 被请求的资源已永久移动到新位置，并且将来任何对此资源的引用都应该使用本响应返回的若干个 URI 之一。如果可能，拥有链接编辑功能的客户端应当自动把请求的地址修改为从服务器反馈回来的地址。除非额外指定，否则这个响应也是可缓存的。

302 Found 请求的资源现在临时从不同的 URI 响应请求。由于这样的重定向是临时的，客户端应当继续向原有地址发送以后的请求。只有在 Cache-Control 或 Expires 中进行了指定的情况下，这个响应才是可缓存的。字面上的区别就是 301 是永久重定向，而 302 是临时重定向。

301 比较常用的场景是使用域名跳转。302 用来做临时跳转 比如未登陆的用户访问用户中心重定向到登录页面。

35. 如何画一个三角形

参考答案：

三角形原理：边框的均分原理

```
div {  
width:0px;  
height:0px;  
border-top:10px solid red;  
border-right:10px solid transparent;  
border-bottom:10px solid transparent;  
border-left:10px solid transparent;  
}
```

36. 状态码 304 和 200

参考答案:

状态码 200: 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。即返回的数据为全量的数据, 如果文件不通过 GZIP 压缩的话, 文件是多大, 则要有多大传输量。

状态码 304: 如果客户端发送了一个带条件的 GET 请求且该请求已被允许, 而文档的内容(自上次访问以来或者根据请求的条件)并没有改变, 则服务器应当返回这个状态码。即客户端和服务端只需要传输很少的数据量来做文件的校验, 如果文件没有修改过, 则不需要返回全量的数据。

37. 说一下浏览器缓存

参考答案:

缓存分为两种: 强缓存和协商缓存, 根据响应的 header 内容来决定。

强缓存相关字段有 expires, cache-control。如果 cache-control 与 expires 同时存在的话, cache-control 的优先级高于 expires。

协商缓存相关字段有 Last-Modified/If-Modified-Since, Etag/If-None-Match

38. HTML5 新增的元素

参考答案:

首先 html5 为了更好的实践 web 语义化, 增加了 header, footer, nav, aside, section 等语义化标签;

在表单方面, 为了增强表单, 为 input 增加了 color, email, data , range 等类型;

在存储方面, 提供了 sessionStorage, localStorage, 和离线存储, 通过这些存储方式方便数据在客户端的存储和获取;

在多媒体方面规定了音频和视频元素 audio 和 video, 另外还有地理定位, canvas 画布, 拖放, 多线程编程的 web worker 和 websocket 协议;

39. 在地址栏里输入一个 URL, 到这个页面呈现出来, 中间会发生什么?

参考答案:

输入 url 后, 首先需要找到这个 url 域名的服务器 ip, 为了寻找这个 ip, 浏览器首先会寻找缓存, 查看缓存中是否有记录, 缓存的查找记录为: 浏览器缓存-》系统缓存-》路由器缓存, 缓存中没有则查找系统的 hosts 文件中是否有记录, 如果没有则查询 DNS 服务器, 得到服务器的 ip 地址后, 浏览器根据这个 ip 以及相应的端口号, 构造一个 http 请求, 这个请求报文会包括这次请求的信息, 主要是请求方法, 请求说明和请求附带的数据, 并将这个 http 请求封装在一个 tcp 包中, 这个 tcp 包会依次经过传输层, 网络层, 数据链路层, 物理层到达服务器, 服务器解析这个请求来作出响应, 返回相应的 html 给浏览器, 因为 html 是一个树形结构, 浏览器根据这个 html 来构建 DOM 树, 在 dom 树的构建过程中如果遇到 JS 脚本和外部 JS 连接, 则会停止构建 DOM 树来执行和下载相应的代码, 这会

造成阻塞，这就是为什么推荐 JS 代码应该放在 html 代码的后面，之后根据外部
央视，内部央视，内联样式构建一个 CSS 对象模型树 CSSOM 树，构建完成后和
DOM 树合并为渲染树，这里主要做的是排除非视觉节点，比如 script，meta 标
签和排除 display 为 none 的节点，之后进行布局，布局主要是确定各个元素的
位置和尺寸，之后是渲染页面，因为 html 文件中会含有图片，视频，音频等资
源，在解析 DOM 的过程中，遇到这些都会进行并行下载，浏览器对每个域的并行
下载数量有一定的限制，一般是 4-6 个，当然在这些所有的请求中我们还需要关
注的就是缓存，缓存一般通过 Cache-Control、Last-Modify、Expires 等首部字
段控制。Cache-Control 和 Expires 的区别在于 Cache-Control 使用相对时间，
Expires 使用的是基于服务器 端的绝对时间，因为存在时差问题，一般采用
Cache-Control，在请求这些有设置了缓存的数据时，会先 查看是否过期，如果
没有过期则直接使用本地缓存，过期则请求并在服务器校验文件是否修改，如果
上一次 响应设置了 ETag 值会在这次请求的时候作为 If-None-Match 的值交给服
务器校验，如果一致，继续校验 Last-Modified，没有设置 ETag 则直接验证
Last-Modified，再决定是否返回 304

40. cookie 和 session 的区别，localStorage 和 sessionstorage 的区别

参考答案：

Cookie 和 session 都可用来存储用户信息，cookie 存放于客户端，session 存
放于服务器端，因为 cookie 存放于客户端有可能被窃取，所以 cookie 一般用来
存放不敏感的信息，比如用户设置的网站主题，敏感的信息用 session 存储，比
如用户的登陆信息，session 可以存放于文件，数据库，内存中都可以，cookie
可以服务器端响应的时候设置，也可以客户端通过 JS 设置 cookie 会在请求时在
http 首部发送给客户端，cookie 一般在客户端有大小限制，一般为 4K。

下面从几个方向区分一下 cookie, localStorage, sessionStorage 的区别:

1) 生命周期:

Cookie: 可设置失效时间, 否则默认为关闭浏览器后失效

LocalStorage: 除非被手动清除, 否则永久保存

SessionStorage: 仅在当前网页会话下有效, 关闭页面或浏览器后就会被清除

2) 存放数据:

Cookie: 4k 左右

LocalStorage 和 sessionStorage: 可以保存 5M 的信息

3) http 请求:

Cookie: 每次都会携带在 http 头中, 如果使用 cookie 保存过多数据会带来性能问题

其他两个: 仅在客户端即浏览器中保存, 不参与和服务器的通信

4) 易用性:

Cookie: 需要程序员自己封装, 原生的 cookie 接口不友好

其他两个: 即可采用原生接口, 亦可再次封装

5) 应用场景:

从安全性来说, 因为每次 http 请求都回携带 cookie 信息, 这样子浪费了带宽, 所以 cookie 应该尽可能的少用, 此外 cookie 还需要指定作用域, 不可以跨域调用, 限制很多, 但是用户识别用户登陆来说, cookie 还是比 storage 好用, 其他情况下可以用 storage, localStorage 可以用来在页面传递参数, sessionStorage 可以用来保存一些临时的数据, 防止用户刷新页面后丢失了一些参数。

41. 常见的 HTTP 的头部

参考答案：

可以将 http 首部分为通用首部，请求首部，响应首部，实体首部

通用首部表示一些通用信息

date 表示报文创建时间，请求首部就是请求报文中独有的；

cookie，和缓存相关的；

if-Modified-Since 响应首部就是响应报文中独有的；

set-cookie，和重定向相关的 location，

实体首部用来描述实体部分

allow 用来描述可执行的请求方法

content-type 描述主题类型

content-Encoding 描述主体的编码方式

42. HTTP2.0 的特性

参考答案：

http2.0 的特性如下：

1) 内容安全，应为 http2.0 是基于 https 的，天然具有安全特性，通过 http2.0 的特性可以避免单纯使用 https 的性能下降；

2) 二进制格式，http1.X 的解析是基于文本的，http2.0 将所有的传输信息分割为更小的消息和帧，并对他们采用二进制格式编码，基于二进制可以让协议有更多的扩展性，比如引入了帧来传输数据和指令；

3) 多路复用, 这个功能相当于是长连接的增强, 每个 request 请求可以随机的混杂在一起, 接收方可以根据 request 的 id 将 request 再归属到各自不同的服务端请求里面, 另外多路复用中也支持了流的优先级, 允许客户端告诉服务器那些内容是更优先级的资源, 可以优先传输;

43. cache-control 的值有哪些

参考答案:

cache-control 是一个通用消息头字段被用于 HTTP 请求和响应中, 通过指定指令来实现缓存机制, 这个缓存指令是单向的, 常见的取值有 private、no-cache、max-age、must-revalidate 等, 默认为 private。

44. 浏览器在生成页面的时候, 会生成那两颗树?

参考答案:

构造两棵树, DOM 树和 CSSOM 规则树

当浏览器接收到服务器相应来的 HTML 文档后, 会遍历文档节点, 生成 DOM 树, CSSOM 规则树由浏览器解析 CSS 文件生成。

45. csrf 和 xss 的网络攻击及防范

参考答案:

CSRF: 跨站请求伪造, 可以理解为攻击者盗用了用户的身份, 以用户的名义发送了恶意请求, 比如用户登录了一个网站后, 立刻在另一个 t a b 页面访问量攻击者用来制造攻击的网站, 这个网站要求访问刚刚登陆的网站, 并发送了一个恶意请求, 这时候 CSRF 就产生了, 比如这个制造攻击的网站使用一张图片, 但是这

种图片的链接却是可以修改数据库的,这时候攻击者就可以以用户的名义操作这个数据库;

防御方式: 使用验证码, 检查 https 头部的 refer, 使用 token

XSS: 跨站脚本攻击, 是说攻击者通过注入恶意的脚本, 在用户浏览网页的时候进行攻击, 比如获取 cookie, 或者其他用户身份信息, 可以分为存储型和反射型, 存储型是攻击者输入一些数据并且存储到了数据库中, 其他浏览者看到的时候进行攻击, 反射型的话不存储在数据库中, 往往表现为将攻击代码放在 url 地址的请求参数中;

防御方式: 为 cookie 设置 httpOnly 属性, 对用户的输入进行检查, 进行特殊字符过滤;

46. 怎么看网站的性能如何

参考答案:

(1) 检测页面加载时间一般有两种方式

被动去测: 就是在被检测的页面置入脚本或探针, 当用户访问网页时, 探针自动采集数据并传回数据库进行分析;

(2) 主动监测的方式, 即主动的搭建分布式受控环境, 模拟用户发起页面访问请求, 主动采集性能数据并分析, 在检测的精准度上, 专业的第三方工具效果更佳, 比如说性能极客;

47. 介绍 HTTP 协议(特征)

参考答案:

HTTP 是一个基于 TCP/IP 通信协议来传递数据 (HTML 文件, 图片文件, 查询结果等) HTTP 是一个属于应用层的面向对象的协议, 由于其简捷、快速的方式, 适用于分布式超媒体信息系统。

它于 1990 年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在 WWW 中使用的是 HTTP/1.0 的第六版，HTTP/1.1 的规范化工作正在进行之中，而且 HTTP-NG (Next Generation of HTTP) 的建议已经提出。HTTP 协议工作于客户端-服务端架构为上。浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端即 WEB 服务器发送所有请求。Web 服务器根据接收到的请求后，向客户端发送响应信息。

48. 说一下对 Cookie 和 Session 的认知，Cookie 有哪些限制？

参考答案：

- 1) cookie 数据存放在客户的浏览器上，session 数据放在服务器上。
- 2) cookie 不是很安全，别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗考虑到安全应当使用 session。
- 3) session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用 COOKIE。
- 4) 单个 cookie 保存的数据不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie。

49. 描述一下 XSS 和 CSRF 攻击？防御方法？

参考答案：

XSS，即为 (Cross Site Scripting)，中文名为跨站脚本，是发生在目标用户的浏览器层面上的，当渲染 DOM 树的过程成发生了不在预期内执行的 JS 代码时，就发生了 XSS 攻击。大多数 XSS 攻击的主要方式是嵌入一段远程或者第三方域上的 JS 代码。实际上是在目标网站的作用域下执行了这段 js 代码。

CSRF (Cross Site Request Forgery，跨站请求伪造)，字面理解意思就是在别的站点伪造了一个请求。专业术语来说就是在受害者访问一个网站时，其 Cookie

还没有过期的情况下，攻击者伪造一个链接地址发送受害者并欺骗让其点击，从而形成 CSRF 攻击。

XSS 防御的总体思路是：对输入(和 URL 参数)进行过滤，对输出进行编码。也就是对提交的所有内容进行过滤，对 url 中的参数进行过滤，过滤掉会导致脚本执行的相关内容；然后对动态输出到页面的内容进行 html 编码，使脚本无法在浏览器中执行。虽然对输入过滤可以被绕过，但是也还是会拦截很大一部分的 XSS 攻击。

防御 CSRF 攻击主要有三种策略：

- (1) 验证 HTTP Referer 字段；
- (2) 在请求地址中添加 token 并验证；
- (3) 在 HTTP 头中自定义属性并验证。

50. 知道 304 吗，什么时候用 304？

参考答案：

304：如果客户端发送了一个带条件的 GET 请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个 304 状态码。

51. 具体有哪些请求头是跟缓存相关的

参考答案：

缓存分为两种：强缓存和协商缓存，根据响应的 header 内容来决定。

强缓存相关字段有 expires, cache-control。如果 cache-control 与 expires 同时存在的话，cache-control 的优先级高于 expires。

协商缓存相关字段有 Last-Modified/If-Modified-Since, Etag/If-None-Match

52. cookie 和 session 的区别

参考答案:

- 1) cookie 数据存放在客户的浏览器上, session 数据放在服务器上。
- 2) cookie 不是很安全, 别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗
考虑到安全应当使用 session。
- 3) session 会在一定时间内保存在服务器上。当访问增多, 会比较占用你服务器的性能
考虑到减轻服务器性能方面, 应当使用 COOKIE。
- 4) 单个 cookie 保存的数据不能超过 4K, 很多浏览器都限制一个站点最多保存 20 个 cookie。

53. cookie 有哪些字段可以设置

参考答案:

name 字段为一个 cookie 的名称。

value 字段为一个 cookie 的值。

domain 字段为可以访问此 cookie 的域名。

非顶级域名, 如二级域名或者三级域名, 设置的 cookie 的 domain 只能为顶级域名或者二级域名或者三级域名本身, 不能设置其他二级域名的 cookie, 否则 cookie 无法生成。

顶级域名只能设置 domain 为顶级域名, 不能设置为二级域名或者三级域名, 否则 cookie 无法生成。

二级域名能读取设置了 domain 为顶级域名或者自身的 cookie，不能读取其他二级域名 domain 的 cookie。所以要想 cookie 在多个二级域名中共享，需要设置 domain 为顶级域名，这样就可以在所有二级域名里面或者到这个 cookie 的值了。顶级域名只能获取到 domain 设置为顶级域名的 cookie，其他 domain 设置为二级域名的无法获取。

path 字段为可以访问此 cookie 的页面路径。

比如 domain 是 abc.com, path 是 /test，那么只有 /test 路径下的页面可以读取此 cookie。

expires/Max-Age 字段为此 cookie 超时时间。

若设置其值为一个时间，那么当到达此时间后，此 cookie 失效。不设置的话默认值是 Session，意思是 cookie 会和 session 一起失效。当浏览器关闭（不是浏览器标签页，而是整个浏览器）后，此 cookie 失效。

Size 字段 此 cookie 大小。

http 字段 cookie 的 httponly 属性。若此属性为 true，则只有在 http 请求头中会带有此 cookie 的信息，而不能通过 document.cookie 来访问此 cookie。

secure 字段 设置是否只能通过 https 来传递此条 cookie

54. cookie 有哪些编码方式？

参考答案：

encodeURIComponent（）

55. 既然你看过图解 http，那你回答下 200 和 304 的区别

参考答案：

200 OK 请求成功。一般用于 GET 与 POST 请求；

304 Not Modified 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源；

56. 除了 cookie，还有什么存储方式。说说 cookie 和 localStorage 的区别

参考答案：

还有 localStorage, sessionStorage, indexedDB 等

cookie 和 localStorage 的区别：

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

cookie 数据还有路径(path)的概念，可以限制。cookie 只属于某个路径下
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。

localStorage 虽然也有存储大小的限制，但是比 cookie 大得多，可以达到 5M 或更大

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 过期时间之前一直有效，即使窗口和浏览器关闭。

57. 浏览器输入网址到页面渲染全过程

参考答案：

DNS 解析

TCP 连接

发送 HTTP 请求

服务器处理请求并返回 HTTP 报文

浏览器解析渲染页面

连接结束

58. HTML5 和 CSS3 用的多吗？你了解它们的新属性吗？有在项目中用过吗？

参考答案：

html5:

1) 标签增删

(1) 8 个语义元素 header section footer aside nav main article figure

(2) 内容元素 mark 高亮 progress 进度

(3) 新的表单控件 calander date time email url search

(4) 新的 input 类型 color date datetime datetime-local email

(5) 移除过时标签 big font frame frameset

2) canvas 绘图，支持内联 SVG。支持 MathML

3) 多媒体 audio video source embed track

4) 本地离线存储，把需要离线存储在本地文件列在一个 manifest 配置文件

5) web 存储。localStorage、SessionStorage

css3:

CSS3 边框如 border-radius, box-shadow 等；

CSS3 背景如 background-size, background-origin 等；

CSS3 2D, 3D 转换如 transform 等；

CSS3 动画如 animation 等。

59. http 常见的请求方法

参考答案：

get、post，这两个用的是最多的，还有很多比如 patch、delete、put、options 等等

60. get 和 post 的区别

参考答案：

GET - 从指定的资源请求数据。

POST - 向指定的资源提交要被处理的数据。

GET：不同的浏览器和服务器不同，一般限制在 2~8K 之间，更加常见的是 1k 以内。

GET 和 POST 的底层也是 TCP/IP，GET/POST 都是 TCP 链接。

GET 产生一个 TCP 数据包；POST 产生两个 TCP 数据包。

对于 GET 方式的请求，浏览器会把 http header 和 data 一并发送出去，服务器响应 200（返回数据）；

而对于 POST，浏览器先发送 header，服务器响应 100 continue，浏览器再发送 data，服务器响应 200 ok（返回数据）。

61. 说说 302，301，304 的状态码

参考答案：

301 Moved Permanently 永久移动。请求的资源已被永久的移动到新 URI，返回信息会包括新的 URI，浏览器会自动定向到新 URI。今后任何新的请求都应使用新的 URI 代替

302 Found 临时移动。与 301 类似。但资源只是临时被移动。客户端应继续使用原有 URI

304 Not Modified 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源

62. web 性能优化

参考答案：

降低请求量：合并资源，减少 HTTP 请求数，minify / gzip 压缩，webP, lazyLoad。

加快请求速度：预解析 DNS，减少域名数，并行加载，CDN 分发。

缓存：HTTP 协议缓存请求，离线缓存 manifest，离线数据缓存 localStorage。

渲染：JS/CSS 优化，加载顺序，服务端渲染，pipeline。

63. 浏览器缓存机制

参考答案：

缓存分为两种：强缓存和协商缓存，根据响应的 header 内容来决定。

强缓存相关字段有 expires, cache-control。如果 cache-control 与 expires 同时存在的话，cache-control 的优先级高于 expires。

协商缓存相关字段有 Last-Modified/If-Modified-Since, Etag/If-None-Match

64. post 和 get 区别

参考答案：

GET - 从指定的资源请求数据。

POST - 向指定的资源提交要被处理的数据。

GET：不同的浏览器和服务端不同，一般限制在 2~8K 之间，更加常见的是 1k 以内。

GET 和 POST 的底层都是 TCP/IP，GET/POST 都是 TCP 链接。

GET 产生一个 TCP 数据包；POST 产生两个 TCP 数据包。

对于 GET 方式的请求，浏览器会把 http header 和 data 一并发送出去，服务器响应 200（返回数据）；

而对于 POST，浏览器先发送 header，服务器响应 100 continue，浏览器再发送 data，服务器响应 200 ok（返回数据）。

65. 说一下 css 盒子模型

参考答案：

简介：就是用来装页面上的元素的矩形区域。CSS 中的盒子模型包括 IE 盒子模型和标准的 W3C 盒子模型。

box-sizing(有 3 个值哦)：border-box, padding-box, content-box.

区别：这两种盒子模型最主要的区别就是 width 的包含范围，在标准的盒子模型中，width 指 content 部分的宽度，在 IE 盒子模型中，width 表示 content+padding+border 这三个部分的宽度，故这使得在计算整个盒子的宽度时存在着差异：

标准盒子模型的盒子宽度：左右 border+左右 padding+width

IE 盒子模型的盒子宽度：width

在 CSS3 中引入了 box-sizing 属性，box-sizing:content-box;表示标准的盒子模型，box-sizing:border-box 表示的是 IE 盒子模型

最后，前面我们还提到了，box-sizing:padding-box, 这个属性值的宽度包含了左右 padding+width

也很好理解性记忆，包含什么，width 就从什么开始算起。

66. 画一条 0.5px 的线

参考答案:

采用 meta viewport 的方式

```
<meta name="viewport" content="initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
```

采用 border-image 的方式

采用 transform: scale() 的方式

67. link 标签和 import 标签的区别

参考答案:

link 属于 html 标签, 而@import 是 css 提供的

页面被加载时, link 会同时被加载, 而@import 引用的 css 会等到页面加载结束后加载。

link 是 html 标签, 因此没有兼容性, 而@import 只有 IE5 以上才能识别。

link 方式样式的权重高于@import 的。

68. transition 和 animation 的区别

参考答案:

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

69. Flex 布局

参考答案:

Flex 是 Flexible Box 的缩写，意为“弹性布局”，用来为盒状模型提供最大的灵活性。

布局的传统解决方案，基于盒状模型，依赖 display 属性 + position 属性 + float 属性。它对于那些特殊布局非常不方便，比如，垂直居中就不容易实现。

简单的分为容器属性和元素属性

容器的属性：

flex-direction: 决定主轴的方向（即子 item 的排列方法）

```
.box {  
flex-direction: row | row-reverse | column | column-reverse;  
}
```

flex-wrap: 决定换行规则

```
.box {  
flex-wrap: nowrap | wrap | wrap-reverse;  
}
```

flex-flow:

```
.box {  
flex-flow: <flex-direction> || <flex-wrap>;  
}
```

justify-content: 对其方式，水平主轴对齐方式

align-items: 对齐方式，垂直轴线方向

项目的属性（元素的属性）：

order 属性: 定义项目的排列顺序，顺序越小，排列越靠前，默认为 0

flex-grow 属性: 定义项目的放大比例，即使存在空间，也不会放大

flex-shrink 属性: 定义了项目的缩小比例，当空间不足的情况下会等比例的缩小，如果定义个 item 的 flex-shrink 为 0，则为不缩小

flex-basis 属性：定义了分配多余的空间，项目占据的空间。

flex：是 flex-grow 和 flex-shrink、flex-basis 的简写，默认值为 0 1 auto。

align-self：允许单个项目与其他项目不一样的对齐方式，可以覆盖

align-items，默认属性为 auto，表示继承父元素的 align-items

比如说，用 flex 实现圣杯布局

70. BFC（块级格式化上下文，用于清楚浮动，防止 margin 重叠等）

参考答案：

直译成：块级格式化上下文，是一个独立的渲染区域，并且有一定的布局规则。

BFC 区域不会与 float box 重叠

BFC 是页面上的一个独立容器，子元素不会影响到外面

计算 BFC 的高度时，浮动元素也会参与计算

那些元素会生成 BFC：

- (1) 根元素
- (2) float 不为 none 的元素
- (3) position 为 fixed 和 absolute 的元素
- (4) display 为 inline-block、table-cell、table-caption, flex, inline-flex 的元素
- (5) overflow 不为 visible 的元素

71. 垂直居中的方法

参考答案：

- (1) margin:auto 法

css:

```
div{
width: 400px;
height: 400px;
position: relative;
border: 1px solid #465468;
}
img{
position: absolute;
margin: auto;
top: 0;
left: 0;
right: 0;
bottom: 0;
}
```

html:

```
<div>

</div>
```

定位为上下左右为 0, margin: 0 可以实现脱离文档流的居中.

(2)margin 负值法

```
.container{
width: 500px;
height: 400px;
border: 2px solid #379;
position: relative;
}
.inner{
width: 480px;
height: 380px;
```

```
background-color: #746;
position: absolute;
top: 50%;
left: 50%;
margin-top: -190px; /*height 的一半*/
margin-left: -240px; /*width 的一半*/
}
```

补充：其实这里也可以将 margin-top 和 margin-left 负值替换成，
transform: translateX(-50%) 和 transform: translateY(-50%)

(3) table-cell（未脱离文档流的）

设置父元素的 display:table-cell, 并且 vertical-align:middle, 这样子元素可以实现垂直居中。

css:

```
div{
width: 300px;
height: 300px;
border: 3px solid #555;
display: table-cell;
vertical-align: middle;
text-align: center;
}
img{
vertical-align: middle;
}
```

(4) 利用 flex

将父元素设置为 display:flex, 并且设置

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

css:

```
.container{
```

```
width: 300px;
height: 200px;
border: 3px solid #546461;
display: -webkit-flex;
display: flex;
-webkit-align-items: center;
align-items: center;
-webkit-justify-content: center;
justify-content: center;
}
.inner{
border: 3px solid #458761;
padding: 20px;
}
```

72. 关于 js 动画和 css3 动画的差异性

参考答案:

渲染线程分为 main thread 和 compositor thread, 如果 css 动画只改变 transform 和 opacity, 这时整个 CSS 动画得以在 compositor thread 完成 (而 js 动画则会在 main thread 执行, 然后出发 compositor thread 进行下一步操作), 特别注意的是如果改变 transform 和 opacity 是不会 layout 或者 paint 的。

区别:

- (1) 功能涵盖面, js 比 css 大
- (2) 实现/重构难度不一, CSS3 比 js 更加简单, 性能跳优方向固定
- (3) 对帧速表现不好的低版本浏览器, css3 可以做到自然降级
- (4) css 动画有天然事件支持
- (5) css3 有兼容性问题

73. 说一下块元素和行元素

参考答案：

块元素：独占一行，并且有自动填满父元素，可以设置 margin 和 padding 以及高度和宽度

行元素：不会独占一行，width 和 height 会失效，并且在垂直方向的 padding 和 margin 会失效。

74. 多行元素的文本省略号

参考答案：

```
display: -webkit-box
-webkit-box-orient: vertical
-webkit-line-clamp: 3
overflow: hidden
```

75. visibility=hidden, opacity=0, display:none

参考答案：

opacity=0，该元素隐藏起来了，但不会改变页面布局，并且，如果该元素已经绑定一些事件，如 click 事件，那么点击该区域，也能触发点击事件的

visibility=hidden，该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已经绑定的事件 display=none，把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素删除掉一样。

76. 双边距重叠问题（外边距折叠）

参考答案：

多个相邻（兄弟或者父子关系）普通流的块元素垂直方向 margin 会重叠
折叠的结果为：

两个相邻的外边距都是正数时，折叠结果是它们两者之间较大的值。

两个相邻的外边距都是负数时，折叠结果是两者绝对值的较大值。

两个外边距一正一负时，折叠结果是两者的相加的和。

77. position 属性

参考答案：

固定定位 fixed:

元素的位置相对于浏览器窗口是固定位置，即使窗口是滚动的它也不会移动。

Fixed 定位使元素的位置与文档流无关，因此不占据空间。Fixed 定位的元素和其他元素重叠。

相对定位 relative:

如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。

绝对定位 absolute:

绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>。absolute 定位使元素的位置与文档流无关，因此不占据空间。absolute 定位的元素和其他元素重叠。

粘性定位 sticky:

元素先按照普通文档流定位，然后相对于该元素在流中的 flow root (BFC) 和 containing block (最近的块级祖先元素) 定位。而后，元素定位表现为在跨越特定阈值前为相对定位，之后为固定定位。

默认定位 Static:

默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。

inherit:

规定应该从父元素继承 position 属性的值。

78. 浮动清除

参考答案:

方法一：使用带 clear 属性的空元素

在浮动元素后使用一个空元素如<div class="clear"></div>，并在 CSS 中赋予.clear{clear:both;}属性即可清理浮动。亦可使用<br class="clear" />或<hr class="clear" />来进行清理。

方法二：使用 CSS 的 overflow 属性

给浮动元素的容器添加 overflow:hidden;或 overflow:auto;可以清除浮动，另外在 IE6 中还需要触发 hasLayout ，例如为父元素设置容器宽高或设置 zoom:1。

在添加 overflow 属性后，浮动元素又回到了容器层，把容器高度撑起，达到了清理浮动的效果。

方法三：给浮动的元素的容器添加浮动

给浮动元素的容器也添加上浮动属性即可清除内部浮动，但是这样会使其整体浮动，影响布局，不推荐使用。

方法四：使用邻接元素处理

什么都不做，给浮动元素后面的元素添加 clear 属性。

方法五：使用 CSS 的 :after 伪元素

结合 :after 伪元素（注意这不是伪类，而是伪元素，代表一个元素之后最近的元素）和 IEhack，可以完美兼容当前主流的各大浏览器，这里的 IEhack 指的是触发 hasLayout。

给浮动元素的容器添加一个 clearfix 的 class，然后给这个 class 添加一个 :after 伪元素实现元素末尾添加一个看不见的块元素（Block element）清理浮动。

79. css3 新特性

参考答案：

开放题。

CSS3 边框如 border-radius, box-shadow 等；

CSS3 背景如 background-size, background-origin 等；

CSS3 2D, 3D 转换如 transform 等；

CSS3 动画如 animation 等。

80. CSS 选择器有哪些，优先级呢

参考答案：

id 选择器，class 选择器，标签选择器，伪元素选择器，伪类选择器等

同一元素引用了多个样式时，排在后面的样式属性的优先级高；

样式选择器的类型不同时，优先级顺序为：id 选择器 > class 选择器 > 标签选择器；

标签之间存在层级包含关系时，后代元素会继承祖先元素的样式。如果后代元素定义了与祖先元素相同的样式，则祖先元素的相同的样式属性会被覆盖。继承的样式的优先级比较低，至少比标签选择器的优先级低；

带有!important 标记的样式属性的优先级最高；

样式表的来源不同时，优先级顺序为：内联样式> 内部样式 > 外部样式 > 浏览器用户自定义样式 > 浏览器默认样式

81. 怎么样让一个元素消失

参考答案：

display:none; visibility:hidden; opacity: 0; 等等

82. 介绍一下盒模型

参考答案：

CSS 盒模型本质上是一个盒子，封装周围的 HTML 元素，它包括：边距，边框，填充，和实际内容。

标准盒模型：一个块的总宽度=width+margin(左右)+padding(左右)+border(左右)

怪异盒模型：一个块的总宽度=width+margin（左右）（既 width 已经包含了 padding 和 border 值）

设置盒模型: `box-sizing: border-box`

83. css 动画如何实现

参考答案:

创建动画序列, 需要使用 `animation` 属性或其子属性, 该属性允许配置动画时间、时长以及其他动画细节, 但该属性不能配置动画的实际表现, 动画的实际表现是由 `@keyframes` 规则实现, 具体情况参见使用 `keyframes` 定义动画序列小节部分。`transition` 也可实现动画。`transition` 强调过渡, 是元素的一个或多个属性发生变化时产生的过渡效果, 同一个元素通过两个不同的途径获取样式, 而第二个途径当某种改变发生 (例如 `hover`) 时才能获取样式, 这样就会产生过渡动画。

84. 如何实现图片在某个容器中居中的?

参考答案:

父元素固定宽高, 利用定位及设置子元素 `margin` 值为自身的一半。

父元素固定宽高, 子元素设置 `position: absolute`, `margin: auto` 平均分配 `margin`

css3 属性 `transform`。子元素设置 `position: absolute`; `left: 50%`; `top: 50%`; `transform: translate(-50%, -50%)`; 即可。

将父元素设置成 `display: table`, 子元素设置为单元格 `display: table-cell`。

弹性布局 `display: flex`。设置 `align-items: center`; `justify-content: center`

85. 如何实现元素的垂直居中

参考答案:

法一: 父元素 `display: flex`, `align-items: center`;

法二: 元素绝对定位, `top: 50%`, `margin-top: -(高度/2)`

法三：高度不确定用 `transform: translateY(-50%)`

法四：父元素 table 布局，子元素设置 `vertical-align:center`;

86. CSS3 中对溢出的处理

参考答案：

`cnk0hu`

`text-overflow` 属性，值为 `clip` 是修剪文本；

`ellipsis` 为显示省略符号来表被修剪的文本；

`string` 为使用给定的字符串来代表被修剪的文本。

87. float 的元素，display 是什么

参考答案：

`display` 为 `block`

88. 隐藏页面中某个元素的方法

参考答案：

`display:none`;

`visibility:hidden`;

`opacity: 0`;

`position` 移到外部，`z-index` 涂层遮盖等等

89. 三栏布局的实现方式，尽可能多写，浮动布局时，三个 div 的生成顺序有没有影响

参考答案：

三列布局又分为两种，两列定宽一列自适应，以及两侧定宽中间自适应

两列定宽一列自适应：

1、使用 float+margin:

给 div 设置 float: left, left 的 div 添加属性 margin-right: left 和 center 的间隔 px, right 的 div 添加属性 margin-left: left 和 center 的宽度之和加上间隔

2、使用 float+overflow:

给 div 设置 float: left, 再给 right 的 div 设置 overflow:hidden。这样子两个盒子浮动，另一个盒子触发 bfc 达到自适应

3、使用 position:

父级 div 设置 position: relative, 三个子级 div 设置 position: absolute, 这个要计算好盒子的宽度和间隔去设置位置，兼容性比较好，

4、使用 table 实现:

父级 div 设置 display: table, 设置 border-spacing: 10px//设置间距，取值随意, 子级 div 设置 display:table-cell, 这种方法兼容性好，适用于高度宽度未知的情况，但是 margin 失效，设计间隔比较麻烦，

5、flex 实现:

parent 的 div 设置 display: flex; left 和 center 的 div 设置 margin-right; 然后 right 的 div 设置 flex: 1; 这样子 right 自适应，但是 flex 的兼容性不好

6、grid 实现:

parent 的 div 设置 display: grid, 设置 grid-template-columns 属性，固定第一列第二列宽度，第三列 auto，

对于两侧定宽中间自适应的布局，对于这种布局需要把 center 放在前面，可以采用双飞翼布局：圣杯布局，来实现，也可以使用上述方法中的 grid, table, flex, position 实现

90. 什么是 BFC

参考答案：

BFC 也就是常说的块格式化上下文，这是一个独立的渲染区域，规定了内部如何布局，并且这个区域的子元素不会影响到外面的元素，其中比较重要的布局规则有内部 box 垂直放置，计算 BFC 的高度的时候，浮动元素也参与计算，触发 BFC 的规则有根元素，浮动元素，position 为 absolute 或 fixed 的元素，display 为 inline-block, table-cell, table-caption, flex, inline-flex, overflow 不为 visible 的元素

91. calc 属性

参考答案：

Calc 用户动态计算长度值，任何长度值都可以使用 calc() 函数计算，需要注意的是，运算符前后都需要保留一个空格，例如：width: calc(100% - 10px);

92. 有一个 width300, height300, 怎么实现在屏幕上垂直水平居中

参考答案：

对于行内块级元素

1) 父级元素设置 text-align: center, 然后设置 line-height 和 vertical-align 使其垂直居中，最后设置 font-size: 0 消除近似居中的 bug

2) 父级元素设置 `display: table-cell, vertical-align: middle` 达到水平垂直居中

3、采用绝对定位，原理是子绝父相，父元素设置 `position: relative`，子元素设置 `position: absolute`，然后通过 `transform` 或 `margin` 组合使用达到垂直居中效果，设置 `top: 50%, left: 50%, transform: translate(-50%, -50%)`

4、绝对居中，原理是当 `top, bottom` 为 0 时，`margin-top&bottom` 设置 `auto` 的话会无限延伸沾满空间并平分，当 `left, right` 为 0 时，`margin-left&right` 设置 `auto` 会无限延伸占满空间并平分，

5、采用 `flex`，父元素设置 `display: flex`，子元素设置 `margin: auto`

视窗居中，`vh` 为视口单位，`50vh` 即是视口高度的 50/100，设置 `margin: 50vh auto 0, transform: translate(-50%)`

93. `display: table` 和本身的 `table` 有什么区别

参考答案：

`Display:table` 和本身 `table` 是相对应的，区别在于，`display: table` 的 `css` 声明能够让一个 `html` 元素和它的子节点像 `table` 元素一样，使用基于表格的 `css` 布局，是我们能够轻松定义一个单元格的边界，背景等样式，而不会产生因为使用了 `table` 那样的制表标签导致的语义化问题。

之所以现在逐渐淘汰了 `table` 系表格元素，是因为用 `div+css` 编写出来的文件比用 `table` 边写出来的文件小，而且 `table` 必须在页面完全加载后才显示，`div` 则是逐行显示，`table` 的嵌套性太多，没有 `div` 简洁

94. `position` 属性的值有哪些及其区别

参考答案：

`Position` 属性把元素放置在一个静态的，相对的，绝对的，固定的位置中，

Static: 位置设置为 static 的元素，他始终处于页面流给予的位置，static 元素会忽略任何 top, bottom, left, right 声明

Relative: 位置设置为 relative 的元素，可将其移至相对于其正常位置的地方，因此 left: 20 会将元素移至元素正常位置左边 20 个像素的位置

Absolute: 此元素可定位于相对包含他的元素的指定坐标，此元素可通过 left, top 等属性规定

Fixed: 位置被设为 fixed 的元素，可定为与相对浏览器窗口的指定坐标，可以通过 left, top, right 属性来定位

95. z-index 的定位方法

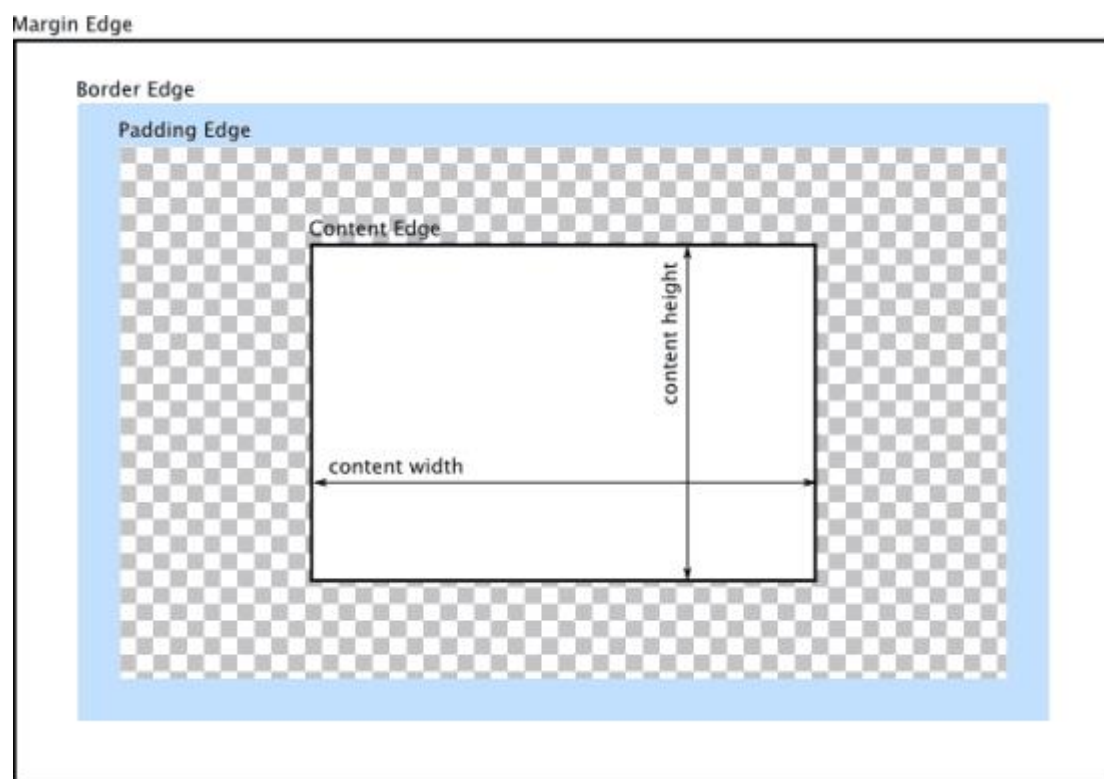
参考答案:

z-index 属性设置元素的堆叠顺序，拥有更好堆叠顺序的元素会处于较低顺序元素之前，z-index 可以为负，且 z-index 只能在定位元素上奏效，该属性设置一个定位元素沿 z 轴的位置，如果为正数，离用户越近，为负数，离用户越远，它的属性值有 auto，默认，堆叠顺序与父元素相等，number，inherit，从父元素继承 z-index 属性的值

96. CSS 盒模型

参考答案:

当对一个文档进行布局时候，浏览器渲染引擎会根据 CSS-box 模型，将所有元素表示为一个矩形盒子，CSS 决定这些盒子的大小，位置，属性，如图：



content 包含元素真实内容的区域，由 width, height, 控制内容大小，内边距 padding, 边框区域 border, 外边距 margin, 用空白区域扩展边框区域，已分开相邻的元素。

97. 如果想要改变一个 DOM 元素的字体颜色，不在它本身上进行操作？

参考答案：

可以更改父元素的 color

98. 对 CSS 的新属性有了解过的吗？

参考答案：

CSS3 的新特性中，在布局方面新增了 flex 布局，在选择器方面新增了例如 first-of-type, nth-child 等选择器，在盒模型方面添加了 box-sizing 来改变盒模型，在动画方面增加了 animation, 2d 变换, 3d 变换等，在颜色方面添加透明, rgba 等，在字体方面允许嵌入字体和设置字体阴影，最后还有媒体查询等

99. 用的最多的 css 属性是啥？

参考答案：

用的目前来说最多的是 flex 属性，灵活但是兼容性方面不强

100. line-height 和 height 的区别

参考答案：

line-height 一般是指布局里面一段文字上下行之间的高度，是针对字体来设置的，height 一般是指容器的整体高度。

101. 设置一个元素的背景颜色，背景颜色会填充哪些区域？

参考答案：

background-color 设置的背景颜色会填充元素的 content、padding、border 区域。

102. 知道属性选择器和伪类选择器的优先级吗

参考答案：

属性选择器和伪类选择器优先级相同

103. inline-block、inline 和 block 的区别；为什么 img 是 inline 还可以设置宽高

参考答案：

Block 是块级元素，其前后都会有换行符，能设置宽度，高度，margin/padding 水平垂直方向都有效。

Inline：设置 width 和 height 无效，margin 在竖直方向上无效，padding 在水平方向垂直方向都有效，前后无换行符

Inline-block：能设置宽度高度，margin/padding 水平垂直方向 都有效，前后无换行符

104. 用 css 实现一个硬币旋转的效果

参考答案：

虽然不认为很多人能在面试中写出来

```
#euro {
width: 150px;
height: 150px;
margin-left: -75px;
margin-top: -75px;
position: absolute;
top: 50%;
left: 50%;
transform-style: preserve-3d;
animation: spin 2.5s linear infinite;
}

.back {
background-image: url("/uploads/160101/backeuro.png");
width: 150px;
```

```
height: 150px;
}
.middle {
background-image: url("/uploads/160101/faceeuro.png");
width: 150px;
height: 150px;
transform: translateZ(1px);
position: absolute;
top: 0;
}
.front {
background-image: url("/uploads/160101/faceeuro.png");
height: 150px;
position: absolute;
top: 0;
transform: translateZ(10px);
width: 150px;
}
@keyframes spin {
0% {
transform: rotateY(0deg);
}
100% {
transform: rotateY(360deg);
}
}
```

105. 了解重绘和重排吗，知道怎么去减少重绘和重排吗，让文档脱离文档流有哪些方法

参考答案：

DOM 的变化影响到了预算内宿的几何属性比如宽高，浏览器重新计算元素的几何属性，其他元素的几何属性也会受到影响，浏览器需要重新构造渲染书，这个过程称之为重排，浏览器将受到影响的部分重新绘制在屏幕上 的过程称为重绘，引起重排重绘的原因有：

- (1) 添加或者删除可见的 DOM 元素，
- (2) 元素尺寸位置的改变
- (3) 浏览器页面初始化，
- (4) 浏览器窗口大小发生改变，重排一定导致重绘，重绘不一定导致重排.

减少重绘重排的方法有：

- (1) 不在布局信息改变时做 DOM 查询，
- (2) 使用 `csstext, className` 一次性改变属性
- (3) 使用 `fragment`
- (4) 对于多次重排的元素，比如说动画。使用绝对定位脱离文档流，使其不影响其他元素

106. CSS 画正方形，三角形

参考答案：

画三角形：

```
#triangle02{
width: 0;
height: 0;
border-top: 50px solid blue;
border-right: 50px solid red;
border-bottom: 50px solid green;
border-left: 50px solid yellow;
}
```

画正方体:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>perspective</title>
<style>
.wrapper{
width: 50%;
float: left;
}
.cube{
font-size: 4em;
width: 2em;
margin: 1.5em auto;
transform-style:preserve-3d;
transform:rotateX(-35deg) rotateY(30deg);
}
.side{
position: absolute;
width: 2em;
height: 2em;
background: rgba(255,99,71,0.6);
border: 1px solid rgba(0,0,0,0.5);
color: white;
text-align: center;
line-height: 2em;
}
.front{
transform:translateZ(1em);
}
.bottom{
```

```
transform:rotateX(-90deg) translateZ(1em);
}
.top{
transform:rotateX(90deg) translateZ(1em);
}
.left{
transform:rotateY(-90deg) translateZ(1em);
}
.right{
transform:rotateY(90deg) translateZ(1em);
}
.back{
transform:translateZ(-1em);
}
</style>
</head>
<body>
<div class="wrapper w1">
<div class="cube">
<div class="side front">1</div>
<div class="side back">6</div>
<div class="side right">4</div>
<div class="side left">3</div>
<div class="side top">5</div>
<div class="side bottom">2</div>
</div>
</div>
<div class="wrapper w2">
<div class="cube">
<div class="side front">1</div>
<div class="side back">6</div>
<div class="side right">4</div>
```

```
<div class="side left">3</div>
<div class="side top">5</div>
<div class="side bottom">2</div>
</div>
</div>
</body>
</html>
```

107. overflow 的原理

参考答案：

要讲清楚这个解决方案的原理，首先需要了解块格式化上下文，A block formatting context is a part of a visual CSS rendering of a Web page. It is the region in which the layout of block boxes occurs and in which floats interact with each other. 翻译过来就是块格式化上下文是 CSS 可视化渲染的一部分，它是一块区域，规定了内部块盒 的渲染方式，以及浮动相互之间的影响关系；

当元素设置了 overflow 样式且值部位 visible 时，该元素就构建了一个 BFC，BFC 在计算高度时，内部浮动元素的高度也要计算在内，也就是说技术 BFC 区域内只有一个浮动元素，BFC 的高度也不会发生塌缩，所以达到了清除浮动的目的

108. 清除浮动的方法

参考答案：

给要清除浮动的元素添加样式 clear，

父元素结束标签钱插入清除浮动的块级元素，给该元素添加样式 clear

添加伪元素，在父级元素的最后，添加一个伪元素，通过清除伪元素的浮动，注意该伪元素的 display 为 block，

父元素添加样式 overflow 清除浮动，overflow 设置除 visible 以外的任何位置

109. box-sizing 的语法和基本用处

参考答案:

box-sizing 规定两个并排的带边框的框，语法为 box-sizing:

content-box/border-box/inherit

content-box: 宽度和高度分别应用到元素的内容框，在宽度和高度之外绘制元素的内边距和边框

border-box: 为元素设定的宽度和高度决定了元素的边框盒，

inherit: 继承父元素的 box-sizing

110. 使元素消失的方法有哪些?

参考答案:

1) opacity: 0, 该元素隐藏起来了，但不会改变页面布局，并且，如果该元素已经绑定一些事件，如 click 事件，那么点击该区域，也能触发点击事件的

2) visibility: hidden, 该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已经绑定的事件

3) display: none, 把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素删除掉。

111. 两个嵌套的 div, position 都是 absolute, 子 div 设置 top 属性, 那么这个 top 是相对于父元素的哪个位置定位的

参考答案:

margin 的外边缘

112. 说说盒子模型

参考答案：

CSS 盒模型本质上是一个盒子，封装周围的 HTML 元素，它包括：边距，边框，填充，和实际内容。

标准盒模型：一个块的总宽度=width+margin(左右)+padding(左右)+border(左右)

怪异盒模型：一个块的总宽度=width+margin(左右)（既 width 已经包含了 padding 和 border 值）

如何设置：box-sizing:border-box

113. display

参考答案：

主要取值有 none, block, inline-block, inline, flex 等。

114. 怎么隐藏一个元素

参考答案：

- 1) opacity: 0, 该元素隐藏起来了，但不会改变页面布局，并且，如果该元素已经绑定一些事件，如 click 事件，那么点击该区域，也能触发点击事件
- 2) visibility: hidden, 该元素隐藏起来了，但不会改变页面布局，但是不会触发该元素已经绑定的事件
- 3) display: none, 把元素隐藏起来，并且会改变页面布局，可以理解成在页面中把该元素删除掉。

115. display:none 和 visibility:hidden 的区别

参考答案:

- 1) visibility: hidden, 该元素隐藏起来了, 但不会改变页面布局, 但是不会触发该元素已经绑定的事件
- 2) display: none, 把元素隐藏起来, 并且会改变页面布局, 可以理解成在页面中把该元素删除掉。

116. 相对布局和绝对布局, position:relative 和 absolute。

参考答案:

相对定位 relative:

如果对一个元素进行相对定位, 它将出现在它所在的位置上。然后, 可以通过设置垂直或水平位置, 让这个元素“相对于”它的起点进行移动。 在使用相对定位时, 无论是否进行移动, 元素仍然占据原来的空间。因此, 移动元素会导致它覆盖其它框。

绝对定位 absolute:

绝对定位的元素的位置相对于最近的已定位父元素, 如果元素没有已定位的父元素, 那么它的位置相对于<html>。 absolute 定位使元素的位置与文档流无关, 因此不占据空间。 absolute 定位的元素和其他元素重叠。

117. flex 布局

参考答案:

flex 是 Flexible Box 的缩写, 意为“弹性布局”。指定容器 display: flex 即可。

容器有以下属性: flex-direction, flex-wrap, flex-flow, justify-content, align-items, align-content。

`flex-direction` 属性决定主轴的方向；

`flex-wrap` 属性定义，如果一条轴线排不下，如何换行；

`flex-flow` 属性是 `flex-direction` 属性和 `flex-wrap` 属性的简写形式，默认值为 `row nowrap`；

`justify-content` 属性定义了项目在主轴上的对齐方式。

`align-items` 属性定义项目在交叉轴上如何对齐。

`align-content` 属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

项目（子元素）也有一些属性：`order`, `flex-grow`, `flex-shrink`, `flex-basis`, `flex`, `align-self`。

`order` 属性定义项目的排列顺序。数值越小，排列越靠前，默认为 0。

`flex-grow` 属性定义项目的放大比例，默认为 0，即如果存在剩余空间，也不放大。

`flex-shrink` 属性定义了项目的缩小比例，默认为 1，即如果空间不足，该项目将缩小。

`flex-basis` 属性定义了分配多余空间之前，项目占据的主轴空间(main size)。

`flex` 属性是 `flex-grow`, `flex-shrink` 和 `flex-basis` 的简写，默认值为 `0 1 auto`。后两个属性可选。

`align-self` 属性允许单个项目有与其他项目不一样的对齐方式，可覆盖

`align-items` 属性。默认值为 `auto`，表示继承父元素的 `align-items` 属性，如果没有父元素，则等同于 `stretch`。

118. `block`、`inline`、`inline-block` 的区别

参考答案：

block 元素会独占一行，多个 block 元素会各自新起一行。默认情况下，block 元素宽度自动填满其父元素宽度。

block 元素可以设置 width,height 属性。块级元素即使设置了宽度, 仍然是独占一行。

block 元素可以设置 margin 和 padding 属性。

inline 元素不会独占一行，多个相邻的行内元素会排列在同一行里，直到一行排列不下，才会新换一行，其宽度随元素的内容而变化。

inline 元素设置 width,height 属性无效。

inline 元素的 margin 和 padding 属性，水平方向的 padding-left, padding-right, margin-left, margin-right 都产生边距效果；但竖直方向的 padding-top, padding-bottom, margin-top, margin-bottom 不会产生边距效果。

inline-block：简单来说就是将对象呈现为 inline 对象，但是对象的内容作为 block 对象呈现。之后的内联对象会被排列在同一行内。比如我们可以给一个 link (a 元素) inline-block 属性值，使其既具有 block 的宽度高度特性又具有 inline 的同行特性。

119. css 的常用选择器

参考答案：

id 选择器，类选择器，伪类选择器等

120. css 布局

参考答案：

六种布局方式总结：圣杯布局、双飞翼布局、Flex 布局、绝对定位布局、表格布局、网格布局。

圣杯布局是指布局从上到下分为 header、container、footer，然后 container 部分定为三栏布局。这种布局方式同样分为 header、container、footer。圣杯布局的缺陷在于 center 是在 container 的 padding 中的，因此宽度小的时候会出现混乱。

双飞翼布局给 center 部分包裹了一个 main 通过设置 margin 主动地把页面撑开。

Flex 布局是由 CSS3 提供的一种方便的布局方式。

绝对定位布局是给 container 设置 position: relative 和 overflow: hidden，因为绝对定位的元素的参照物为第一个 position 不为 static 的祖先元素。left 向左浮动，right 向右浮动。center 使用绝对定位，通过设置 left 和 right 并把两边撑开。center 设置 top: 0 和 bottom: 0 使其高度撑开。

表格布局的好处是能使三栏的高度统一。

网格布局可能是最强大的布局方式了，使用起来极其方便，但目前而言，兼容性并不好。网格布局，可以将页面分割成多个区域，或者用来定义内部元素的大小，位置，图层关系。

121. css 定位

参考答案：

固定定位 fixed:

元素的位置相对于浏览器窗口是固定位置，即使窗口是滚动的它也不会移动。

Fixed 定位使元素的位置与文档流无关，因此不占据空间。Fixed 定位的元素和其他元素重叠。

相对定位 relative:

如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。

绝对定位 absolute:

绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>。absolute 定位使元素的位置与文档流无关，因此不占据空间。absolute 定位的元素和其他元素重叠。

粘性定位 sticky:

元素先按照普通文档流定位，然后相对于该元素在流中的 flow root (BFC) 和 containing block (最近的块级祖先元素) 定位。而后，元素定位表现为在跨越特定阈值前为相对定位，之后为固定定位。

默认定位 Static:

默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。

inherit:

规定应该从父元素继承 position 属性的值。

122. relative 定位规则

参考答案:

如果对一个元素进行相对定位,它将出现在它所在的位置上。然后,可以通过设置垂直或水平位置,让这个元素“相对于”它的起点进行移动。在使用相对定位时,无论是否进行移动,元素仍然占据原来的空间。因此,移动元素会导致它覆盖其它框。

123. 垂直居中

参考答案:

父元素固定宽高,利用定位及设置子元素 margin 值为自身的一半。

父元素固定宽高,子元素设置 position: absolute, margin: auto 平均分配 margin

css3 属性 transform。子元素设置 position: absolute; left: 50%; top: 50%; transform: translate(-50%, -50%); 即可。

将父元素设置成 display: table, 子元素设置为单元格 display: table-cell。

弹性布局 display: flex。设置 align-items: center; justify-content: center;

124. css 预处理器有什么

参考答案:

less, sass 等

125. get 请求传参长度的误区

参考答案:

误区：我们经常说 get 请求参数的大小存在限制，而 post 请求的参数大小是无限制的。

实际上 HTTP 协议从未规定 GET/POST 的请求长度限制是多少。对 get 请求参数的限制是来源与浏览器或 web 服务器，浏览器或 web 服务器限制了 url 的长度。为了明确这个概念，我们必须再次强调下面几点：

HTTP 协议 未规定 GET 和 POST 的长度限制

GET 的最大长度显示是因为 浏览器和 web 服务器限制了 URI 的长度

不同的浏览器和 WEB 服务器，限制的最大长度不一样

要支持 IE，则最大长度为 2083byte，若只支持 Chrome，则最大长度 8182byte

126. 补充 get 和 post 请求在缓存方面的区别

参考答案：

post/get 的请求区别，具体不再赘述。

补充补充一个 get 和 post 在缓存方面的区别：

get 请求类似于查找的过程，用户获取数据，可以不用每次都与数据库连接，所以可以使用缓存。

post 不同，post 做的一般是修改和删除的工作，所以必须与数据库交互，所以不能使用缓存。因此 get 请求适合于请求缓存。

127. 说一下闭包

参考答案：

一句话可以概括：闭包就是能够读取其他函数内部变量的函数，或者子函数在外调用，子函数所在的父函数的作用域不会被释放。

128. 说一下类的创建和继承

参考答案：

(1) 类的创建 (es5)：

new 一个 function，在这个 function 的 prototype 里面增加属性和方法。

下面来创建一个 Animal 类：

```
// 定义一个动物类
function Animal (name) {
  // 属性
  this.name = name || 'Animal';
  // 实例方法
  this.sleep = function() {
    console.log(this.name + '正在睡觉!');
  }
}

// 原型方法
Animal.prototype.eat = function(food) {
  console.log(this.name + '正在吃:' + food);
};
```

这样就生成了一个 Animal 类，实例化生成对象后，有方法和属性。

(2) 类的继承——原型链继承

```
function Cat() { }
Cat.prototype = new Animal();
Cat.prototype.name = 'cat';
// Test Code
var cat = new Cat();
```

```
console.log(cat.name);  
console.log(cat.eat('fish'));  
console.log(cat.sleep());  
console.log(cat instanceof Animal); //true  
console.log(cat instanceof Cat); //true
```

介绍：在这里我们可以看到 new 了一个空对象，这个空对象指向 Animal 并且 Cat.prototype 指向了这个空对象，这种就是基于原型链的继承。

特点：基于原型链，既是父类的实例，也是子类的实例

缺点：无法实现多继承

（3）构造继承：

使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）

```
function Cat(name){  
  Animal.call(this);  
  this.name = name || 'Tom';  
}  
  
// Test Code  
var cat = new Cat();  
console.log(cat.name);  
console.log(cat.sleep());  
console.log(cat instanceof Animal); // false  
console.log(cat instanceof Cat); // true
```

特点：可以实现多继承

缺点：只能继承父类实例的属性和方法，不能继承原型上的属性和方法。

（4）实例继承和拷贝继承

实例继承：为父类实例添加新特性，作为子类实例返回

拷贝继承：拷贝父类元素上的属性和方法

上述两个实用性不强，不一一举例。

(5) 组合继承：

相当于构造继承和原型链继承的组合物。通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用

```
function Cat(name) {  
  Animal.call(this);  
  this.name = name || 'Tom';  
}  
Cat.prototype = new Animal();  
Cat.prototype.constructor = Cat;  
// Test Code  
var cat = new Cat();  
console.log(cat.name);  
console.log(cat.sleep());  
console.log(cat instanceof Animal); // true  
console.log(cat instanceof Cat); // true
```

特点：可以继承实例属性/方法，也可以继承原型属性/方法

缺点：调用了两次父类构造函数，生成了两份实例

(6) 寄生组合继承：

通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性

```
function Cat(name) {  
  Animal.call(this);  
  this.name = name || 'Tom';  
}  
  
(function() {  
  // 创建一个没有实例方法的类  
  var Super = function() {};
```

```
Super.prototype = Animal.prototype;
//将实例作为子类的原型
Cat.prototype = new Super();
})();
// Test Code
var cat = new Cat();
console.log(cat.name);
console.log(cat.sleep());
console.log(cat instanceof Animal); // true
console.log(cat instanceof Cat); //true
```

129. 如何解决异步回调地狱

参考答案:

promise、generator、async/await

130. 说说前端中的事件流

参考答案:

HTML 中与 javascript 交互是通过事件驱动来实现的，例如鼠标点击事件 onclick、页面的滚动事件 onscroll 等等，可以向文档或者文档中的元素添加事件侦听器来预订事件。要知道这些事件是在什么时候进行调用的，就需要了解一下“事件流”的概念。

什么是事件流：事件流描述的是从页面中接收事件的顺序, DOM2 级事件流包括下面几个阶段。

事件捕获阶段

处于目标阶段

事件冒泡阶段

addEventListener: `addEventListener` 是 DOM2 级事件新增的指定事件处理程序的操作，这个方法接收 3 个参数：要处理的事件名、作为事件处理程序的函数和一个布尔值。最后这个布尔值参数如果是 `true`，表示在捕获阶段调用事件处理程序；如果是 `false`，表示在冒泡阶段调用事件处理程序。

IE 只支持事件冒泡。

131. 如何让事件先冒泡后捕获

参考答案：

在 DOM 标准事件模型中，是先捕获后冒泡。但是如果要实现先冒泡后捕获的效果，对于同一个事件，监听捕获和冒泡，分别对应相应的处理函数，监听到捕获事件，先暂缓执行，直到冒泡事件被捕获后再执行捕获之间。

132. 说一下事件委托

参考答案：

简介：事件委托指的是，不在事件的发生地（直接 dom）上设置监听函数，而是在其父元素上设置监听函数，通过事件冒泡，父元素可以监听到子元素上事件的触发，通过判断事件发生元素 DOM 的类型，来做出不同的响应。

举例：最经典的就是 `ul` 和 `li` 标签的事件监听，比如我们在添加事件时候，采用事件委托机制，不会在 `li` 标签上直接添加，而是在 `ul` 父元素上添加。

好处：比较合适动态元素的绑定，新添加的子元素也会有监听函数，也可以有事件触发机制。

133. 说一下图片的懒加载和预加载

参考答案：

预加载：提前加载图片，当用户需要查看时可直接从本地缓存中渲染。

懒加载：懒加载的主要目的是作为服务器前端的优化，减少请求数或延迟请求数。

两种技术的本质：两者的行为是相反的，一个是提前加载，一个是迟缓甚至不加载。

懒加载对服务器前端有一定的缓解压力作用，预加载则会增加服务器前端压力。

134. mouseover 和 mouseenter 的区别

参考答案：

mouseover：当鼠标移入元素或其子元素都会触发事件，所以有一个重复触发，冒泡的过程。对应的移除事件是 mouseout

mouseenter：当鼠标移除元素本身（不包含元素的子元素）会触发事件，也就是不会冒泡，对应的移除事件是 mouseleave

135. js 的 new 操作符做了哪些事情

参考答案：

new 操作符新建了一个空对象，这个对象原型指向构造函数的 prototype，执行构造函数后返回这个对象。

136. 改变函数内部 this 指针的指向函数(bind, apply, call 的区别)

参考答案：

通过 `apply` 和 `call` 改变函数的 `this` 指向, 他们两个函数的第一个参数都是一样的表示要改变指向的那个对象, 第二个参数, `apply` 是数组, 而 `call` 则是 `arg1, arg2...` 这种形式。通过 `bind` 改变 `this` 作用域会返回一个新的函数, 这个函数不会马上执行。

137. js 的各种位置, 比如

`clientHeight`, `scrollHeight`, `offsetHeight` , 以及 `scrollTop`, `offsetTop`, `clientTop` 的区别?

参考答案:

`clientHeight`: 表示的是可视区域的高度, 不包含 `border` 和滚动条

`offsetHeight`: 表示可视区域的高度, 包含了 `border` 和滚动条

`scrollHeight`: 表示了所有区域的高度, 包含了因为滚动被隐藏的部分。

`clientTop`: 表示边框 `border` 的厚度, 在未指定的情况下一般为 0

`scrollTop`: 滚动后被隐藏的高度, 获取对象相对于由 `offsetParent` 属性指定的父坐标 (css 定位的元素或 `body` 元素) 距离顶端的高度。

138. js 拖拽功能的实现

参考答案:

首先是三个事件, 分别是 `mousedown`, `mousemove`, `mouseup`

当鼠标点击按下的时候, 需要一个 `tag` 标识此时已经按下, 可以执行 `mousemove` 里面的具体方法。

`clientX`, `clientY` 标识的是鼠标的坐标, 分别标识横坐标和纵坐标, 并且我们用 `offsetX` 和 `offsetY` 来表示元素的元素的初始坐标, 移动的举例应该是:

鼠标移动时候的坐标-鼠标按下去时候的坐标。

也就是说定位信息为：

鼠标移动时候的坐标-鼠标按下去时候的坐标+元素初始情况下的 offsetLeft.

还有一点也是原理性的东西，也就是拖拽的同时是绝对定位，我们改变的是绝对定位条件下的 left

以及 top 等等值。

补充：也可以通过 html5 的拖放（Drag 和 drop）来实现

139. 异步加载 js 的方法

参考答案：

defer：只支持 IE 如果您的脚本不会改变文档的内容，可将 defer 属性加入到 <script> 标签中，以便加快处理文档的速度。因为浏览器知道它将能够安全地读取文档的剩余部分而不用执行脚本，它将推迟对脚本的解释，直到文档已经显示给用户为止。

async，HTML5 属性仅适用于外部脚本，并且如果在 IE 中，同时存在 defer 和 async，那么 defer 的优先级比较高，脚本将在页面完成时执行。

创建 script 标签，插入到 DOM 中。

140. Ajax 解决浏览器缓存问题

参考答案：

在 ajax 发送请求前加上

anyAjaxObj.setRequestHeader("If-Modified-Since", "0")。

在 ajax 发送请求前加上

anyAjaxObj.setRequestHeader("Cache-Control", "no-cache")。

在 URL 后面加上一个随机数： "fresh=" + Math.random()。

在 URL 后面加上时间戳： "nowtime=" + new Date().getTime()。

如果是使用 jQuery，直接这样就可以了 `$.ajaxSetup({cache:false})`。这样页面的所有 ajax 都会执行这条语句就是不需要保存缓存记录。

141. js 的节流和防抖

参考答案：

scroll 事件本身会触发页面的重新渲染，同时 scroll 事件的 handler 又会被高频度的触发，因此事件的 handler 内部不应该有复杂操作，例如 DOM 操作就不应该放在事件处理中。

针对此类高频度触发事件问题（例如页面 scroll，屏幕 resize，监听用户输入等），下面介绍两种常用的解决方法，防抖和节流。

防抖（Debouncing）：

防抖技术即是可以把多个顺序地调用合并成一次，也就是在一定时间内，规定事件被触发的次数。

通俗一点来说，看看下面这个简化的例子：

// 简单的防抖动函数

```
function debounce(func, wait, immediate) {  
    // 定时器变量  
    var timeout;  
    return function() {  
        // 每次触发 scroll handler 时先清除定时器  
        clearTimeout(timeout);  
        // 指定 xx ms 后触发真正想进行的操作 handler  
        timeout = setTimeout(func, wait);  
    };  
};  
  
// 实际想绑定在 scroll 事件上的 handler  
function realFunc() {  
    console.log("Success");  
};
```

```
}  
// 采用了防抖动  
window.addEventListener('scroll', debounce(realFunc, 500));  
// 没采用防抖动  
window.addEventListener('scroll', realFunc);
```

上面简单的防抖的例子可以拿到浏览器下试一下，大概功能就是如果 500ms 内没有连续触发两次 scroll 事件，那么才会触发我们真正想在 scroll 事件中触发的函数。

上面的示例可以更好的封装一下：

```
// 防抖动函数  
function debounce(func, wait, immediate) {  
    var timeout;  
    return function() {  
        var context = this, args = arguments;  
        var later = function() {  
            timeout = null;  
            if (!immediate) func.apply(context, args);  
        };  
        var callNow = immediate && !timeout;  
        clearTimeout(timeout);  
        timeout = setTimeout(later, wait);  
        if (callNow) func.apply(context, args);  
    };  
};  
  
var myEfficientFn = debounce(function() {  
    // 滚动中的真正的操作  
}, 250);  
// 绑定监听  
window.addEventListener('resize', myEfficientFn);
```

节流（Throttling）：

防抖函数确实不错，但是也有问题，譬如图片的懒加载，我希望在下滑过程中图片不断的被加载出来，而不是只有当我停止下滑时候，图片才被加载出来。又或者下滑时候的数据的 ajax 请求加载也是同理。

这个时候，我们希望即使页面在不断被滚动，但是滚动 handler 也可以以一定的频率被触发（譬如 250ms 触发一次），这类场景，就要用到另一种技巧，称为节流函数（throttling）。

节流函数，只允许一个函数在 X 毫秒内执行一次。

与防抖相比，节流函数最主要的不同在于它保证在 X 毫秒内至少执行一次我们希望触发的事件 handler。

与防抖相比，节流函数多了一个 mustRun 属性，代表 mustRun 毫秒内，必然会触发一次 handler，同样是利用定时器，看看简单的示例：

// 简单的节流函数

```
function throttle(func, wait, mustRun) {  
    var timeout,  
        startTime = new Date();  
    return function() {  
        var context = this,  
            args = arguments,  
            curTime = new Date();  
        clearTimeout(timeout);  
        // 如果达到了规定的触发时间间隔，触发 handler  
        if(curTime - startTime >= mustRun){  
            func.apply(context, args);  
            startTime = curTime;  
        }  
        // 没达到触发间隔，重新设定定时器  
        else{  
            timeout = setTimeout(func, wait);  
        }  
    }  
}
```

```
    };  
};  
// 实际想绑定在 scroll 事件上的 handler  
function realFunc() {  
    console.log("Success");  
}  
// 采用了节流函数  
window.addEventListener('scroll', throttle(realFunc, 500, 1000));
```

上面简单的节流函数的例子可以拿到浏览器下试一下，大概功能就是如果在一段时间内 scroll 触发的间隔一直短于 500ms，那么能保证事件我们希望调用的 handler 至少在 1000ms 内会触发一次。

142. JS 中的垃圾回收机制

参考答案：

必要性：由于字符串、对象和数组没有固定大小，所有当它们的大小已知时，才能对它们进行动态的存储分配。JavaScript 程序每次创建字符串、数组或对象时，解释器都必须分配内存来存储那个实体。只要像这样动态地分配了内存，最终都要释放这些内存以便它们能够被再用，否则，JavaScript 的解释器将会消耗完系统中所有可用的内存，造成系统崩溃。

这段话解释了为什么需要系统需要垃圾回收，JS 不像 C/C++，他有自己的一套垃圾回收机制（Garbage Collection）。JavaScript 的解释器可以检测到何时程序不再使用一个对象了，当他确定了一个对象是无用的时候，他就知道不再需要这个对象，可以把它所占用的内存释放掉了。

例如：

```
var a="hello world";  
var b="world";  
var a=b;
```

//这时，会释放掉“hello world”，释放内存以便再引用

垃圾回收的方法：标记清除、计数引用。

标记清除：

这是最常见的垃圾回收方式，当变量进入环境时，就标记这个变量为“进入环境”，从逻辑上讲，永远不能释放进入环境的变量所占的内存，永远不能释放进入环境变量所占用的内存，只要执行流程进入相应的环境，就可能用到他们。当离开环境时，就标记为离开环境。

垃圾回收器在运行的时候会给存储在内存中的变量都加上标记（所有都加），然后去掉环境变量中的变量，以及被环境变量中的变量所引用的变量（条件性去除标记），删除所有被标记的变量，删除的变量无法在环境变量中被访问所以会被删除，最后垃圾回收器，完成了内存的清除工作，并回收他们所占用的内存。

引用计数法：

另一种不太常见的方法就是引用计数法，引用计数法的意思就是每个值没引用的次数，当声明了一个变量，并用一个引用类型的值赋值给改变量，则这个值的引用次数为 1；相反的，如果包含了对这个值引用的变量又取得了另外一个值，则原先的引用值引用次数就减 1，当这个值的引用次数为 0 的时候，说明没有办法再访问这个值了，因此就把所占的内存给回收进来，这样垃圾收集器再次运行的时候，就会释放引用次数为 0 的这些值。

用引用计数法会存在内存泄露，下面来看原因：

```
function problem() {  
  var objA = new Object();  
  var objB = new Object();  
  objA.someOtherObject = objB;
```

```
objB.anotherObject = objA;  
}
```

在这个例子里面，objA 和 objB 通过各自的属性相互引用，这样的话，两个对象的引用次数都为 2，在采用引用计数的策略中，由于函数执行之后，这两个对象都离开了作用域，函数执行完成之后，因为计数不为 0，这样的相互引用如果大量存在就会导致内存泄露。

特别是在 DOM 对象中，也容易存在这种问题：

```
var element=document.getElementById ( ' ' );  
var myObj=new Object();  
myObj.element=element;  
element.someObject=myObj;  
这样就不会有垃圾回收的过程。
```

143. eval 是做什么的

参考答案：

它的功能是将对应的字符串解析成 js 并执行，应该避免使用 js，因为非常消耗性能（2 次，一次解析成 js，一次执行）

144. 如何理解前端模块化

参考答案：

前端模块化就是复杂的文件编程一个一个独立的模块，比如 js 文件等等，分成独立的模块有利于重用（复用性）和维护（版本迭代），这样会引来模块之间相互依赖的问题，所以有了 commonJS 规范，AMD，CMD 规范等等，以及用于 js 打包（编译等处理）的工具 webpack。

145. 说一下 Commonjs、AMD 和 CMD

参考答案:

一个模块是能实现特定功能的文件，有了模块就可以方便的使用别人的代码，想要什么功能就能加载什么模块。

Commonjs: 开始于服务器端的模块化，同步定义的模块化，每个模块都是一个单独的作用域，模块输出，`module.exports`，模块加载 `require()` 引入模块。

AMD: 中文名异步模块定义的意思。

requireJS 实现了 AMD 规范，主要用于解决下述两个问题。

- 1) 多个文件有依赖关系，被依赖的文件需要早于依赖它的文件加载到浏览器
- 2) 加载的时候浏览器会停止页面渲染，加载文件越多，页面失去响应的时间越长。

语法: requireJS 定义了一个函数 `define`，它是全局变量，用来定义模块。

requireJS 的例子:

//定义模块

```
define(['dependency'], function() {  
  var name = 'Byron';  
  function printName() {  
    console.log(name);  
  }  
  return {  
    printName: printName  
  };  
});
```

//加载模块

```
require(['myModule'], function (my) {  
  my.printName();  
})
```

requirejs 定义了一个函数 `define`，它是全局变量，用来定义模块:


```
define(id?dependencies?, factory)
```

在页面上使用模块加载函数：

```
require([dependencies], factory);
```

总结 AMD 规范：require（）函数在加载依赖函数的时候是异步加载的，这样浏览器不会失去响应，它指定的回调函数，只有前面的模块加载成功，才会去执行。因为网页在加载 js 的时候会停止渲染，因此我们可以通过异步的方式去加载 js，而如果需要依赖某些，也是异步去依赖，依赖后再执行某些方法。

146. 对象深度克隆的简单实现

参考答案：

```
function deepClone(obj) {  
  var newObj= obj instanceof Array ? []: {};  
  for(var item in obj) {  
    var temple= typeof obj[item] == 'object' ?  
    deepClone(obj[item]):obj[item];  
    newObj[item] = temple;  
  }  
  return newObj;  
}
```

ES5 的常用的对象克隆的一种方式。注意数组是对象，但是跟对象又有一定区别，所以我们一开始判断了一些类型，决定 newObj 是对象还是数组

147. 实现一个 once 函数，传入函数参数只执行一次

参考答案：

```
function ones(func) {  
  var tag=true;  
  return function() {
```

```
if(tag==true){  
func.apply(null,arguments);  
tag=false;  
}  
return undefined  
}  
}
```

148. 将原生的 ajax 封装成 promise

参考答案:

```
var myNewAjax=function(url){  
return new Promise(function(resolve,reject){  
var xhr = new XMLHttpRequest();  
xhr.open('get',url);  
xhr.send(data);  
xhr.onreadystatechange=function(){  
if(xhr.status==200&&readyState==4){  
var json=JSON.parse(xhr.responseText);  
resolve(json)  
}else if(xhr.readyState==4&&xhr.status!=200){  
reject('error');  
}  
}  
})  
}
```

149. js 监听对象属性的改变

参考答案:

我们假设这里有一个 user 对象,

(1)在 ES5 中可以通过 Object.defineProperty 来实现已有属性的监听

```
Object.defineProperty(user, 'name', {  
  set: function(key, value) {  
  }  
})
```

缺点：如果 id 不在 user 对象中，则不能监听 id 的变化

(2) 在 ES6 中可以通过 Proxy 来实现

```
var user = new Proxy({}, {  
  set: function(target, key, value, receiver) {  
  }  
})
```

这样即使有属性在 user 中不存在，通过 user.id 来定义也同样可以这样监听这个属性的变化哦~

150. 如何实现一个私有变量，用 getName 方法可以访问，不能直接访问

参考答案：

(1) 通过 defineProperty 来实现

```
obj={  
  name:yuxiaoliang,  
  getName:function() {  
    return this.name  
  }  
}  
  
Object.defineProperty(obj, "name", {  
  //不可枚举不可配置  
});
```

(2) 通过函数的创建形式

```
function product() {  
  var name='yuxiaoliang';
```

```
this.getName=function() {  
  return name;  
}  
}  
var obj=new product();
```

151. ==和===、以及 Object.is 的区别

参考答案:

(1) ==

主要存在: 强制转换成 number, null==undefined

" "==0 //true

"0 "==0 //true

" " != "0" //true

123=="123" //true

null==undefined //true

(2)Object.is

主要的区别就是+0!==-0 而 NaN==NaN

(相对比===和==的改进)

152. setTimeout、setInterval 和 requestAnimationFrame 之间的区别

参考答案:

requestAnimationFrame 与 setTimeout 和 setInterval 不同,

requestAnimationFrame 不需要设置时间间隔,

大多数电脑显示器的刷新频率是 60Hz, 大概相当于每秒钟重绘 60 次。大多数浏览器都会对重绘操作加以限制, 不超过显示器的重绘频率, 因为即使超过那个频

率用户体验也不会有提升。因此，最平滑动画的最佳循环间隔是 1000ms/60，约等于 16.6ms。

RAF 采用的是系统时间间隔，不会因为前面的任务，不会影响 RAF，但是如果前面的任务多的话，

会响应 setTimeout 和 setInterval 真正运行时的时间间隔。

特点：

(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。

(2) 在隐藏或不可见的元素中，requestAnimationFrame 将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。

requestAnimationFrame 是由浏览器专门为动画提供的 API，在运行时浏览器会自动优化方法的调用，并且如果页面不是激活状态下的话，动画会自动暂停，有效节省了 CPU 开销。

153. 实现一个两列等高布局，讲讲思路

参考答案：

为了实现两列等高，可以给每列加上 padding-bottom:9999px;

margin-bottom:-9999px;同时父元素设置 overflow:hidden;

154. 自己实现一个 bind 函数

参考答案：

原理：通过 apply 或者 call 方法来实现。

(1) 初始版本

```
Function.prototype.bind=function(obj,arg){  
var arg=Array.prototype.slice.call(arguments,1);
```

```
var context=this;
return function(newArg) {
  arg=arg.concat(Array.prototype.slice.call(newArg));
  return context.apply(obj, arg);
}
}
```

(2) 考虑到原型链

为什么要考虑？因为在 new 一个 bind 过生成的新函数的时候，必须的条件是要继承原函数的原型

```
Function.prototype.bind=function(obj, arg) {
  var arg=Array.prototype.slice.call(arguments, 1);
  var context=this;
  var bound=function(newArg) {
    arg=arg.concat(Array.prototype.slice.call(newArg));
    return context.apply(obj, arg);
  }
  var F=function() {}
  //这里需要一个寄生组合继承
```

```
F.prototype=context.prototype;
bound.prototype=new F();
return bound;
}
```

155. 用 setTimeout 来实现 setInterval

参考答案：

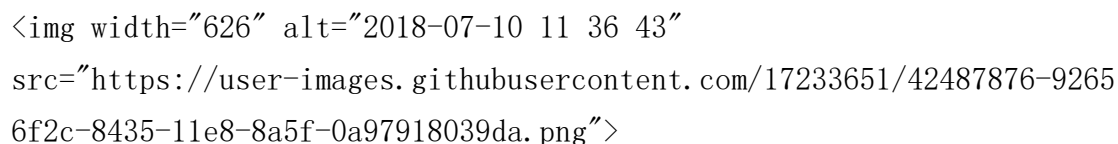
(1) 用 setTimeout() 方法来模拟 setInterval() 与 setInterval() 之间的什么区别？

首先来看 `setInterval` 的缺陷，使用 `setInterval()` 创建的定时器确保了定时器代码规则地插入队列中。这个问题在于：如果定时器代码在代码再次添加到队列之前还没完成执行，结果就会导致定时器代码连续运行好几次。而之间没有间隔。不过幸运的是：javascript 引擎足够聪明，能够避免这个问题。当且仅当没有该定时器的如何代码实例时，才会将定时器代码添加到队列中。这确保了定时器代码加入队列中最小的时间间隔为指定时间。

这种重复定时器的规则有两个问题：1. 某些间隔会被跳过 2. 多个定时器的代码执行时间可能会比预期小。

下面举例子说明：

假设，某个 `onclick` 事件处理程序使用啦 `setInterval()` 来设置了一个 200ms 的重复定时器。如果事件处理程序花了 300ms 多一点的时间完成。



这个例子中的第一个定时器是在 205ms 处添加到队列中，但是要过 300ms 才能执行。在 405ms 又添加了一个副本。在一个间隔，605ms 处，第一个定时器代码还在执行中，而且队列中已经有了一个定时器实例，结果是 605ms 的定时器代码不会添加到队列中。结果是在 5ms 处添加的定时器代码执行结束后，405 处的代码立即执行。

```
function say() {  
  //something  
  setTimeout(say, 200);  
}
```

```
setTimeout(say, 200)
```

或者

```
setTimeout(function() {  
  //do something  
  setTimeout(arguments.callee, 200);  
}, 200);
```

156. 代码的执行顺序

参考答案:

```
setTimeout(function() {console.log(1)}, 0);  
new Promise(function(resolve, reject) {  
  console.log(2);  
  resolve();  
}).then(function() {console.log(3)})  
  .then(function() {console.log(4)});  
process.nextTick(function() {console.log(5)});  
console.log(6);  
//输出 2, 6, 5, 3, 4, 1
```

157. 如何实现 sleep 的效果 (es5 或者 es6)

参考答案:

(1)while 循环的方式

```
function sleep(ms) {  
  var start=Date.now(), expire=start+ms;  
  while(Date.now()<expire);  
  console.log('1111');  
  return;  
}
```

执行 sleep(1000) 之后，休眠了 1000ms 之后输出了 1111。上述循环的方式缺点很明显，容易造成死循环。

(2)通过 promise 来实现

```
function sleep(ms) {
```



```
var temple=new Promise(  
  (resolve)=>{  
    console.log(111);setTimeout(resolve,ms)  
  });  
return temple  
}  
sleep(500).then(function() {  
  //console.log(222)  
})  
//先输出了 111， 延迟 500ms 后输出 222
```

(3)通过 async 封装

```
function sleep(ms) {  
  return new Promise((resolve)=>setTimeout(resolve,ms));  
}  
async function test() {  
  var temple=await sleep(1000);  
  console.log(1111)  
  return temple  
}  
test();  
//延迟 1000ms 输出了 1111
```

(4). 通过 generate 来实现

```
function* sleep(ms) {  
  yield new Promise(function(resolve, reject) {  
    console.log(111);  
    setTimeout(resolve,ms);  
  })  
}  
sleep(500).next().value.then(function() {console.log(2222)})
```

158. 简单的实现一个 promise

参考答案:

Promise 允许我们通过链式调用的方式来解决“回调地狱”的问题，特别是在异步过程中，通过 Promise 可以保证代码的整洁性和可读性。下面主要解读 Promise/A+规范，并在此规范的基础上，自己实现一个 Promise。

一、Promise 的使用

在了解 Promise 规范之前，我们知道主流的高版本浏览器已经支持 ECMA 中的 Promise。

创建一个 promise 实例：

```
var p=new Promise(function(resolve,reject){
    setTimeout(function(){
        resolve("success")
    },1000);
    console.log("创建一个新的 promise");
})
p.then(function(x){
    console.log(x)
})
```

//输出：

创建一个新的 promise

Success

上述是一个 promise 的实例，输出内容为，“创建一个 promise”，延迟 1000ms 后，输出“success”。

从上述的例子可以看出，promise 方便处理异步操作。此外 promise 还可以链式的调用：

```
var p=new Promise(function(resolve,reject){resolve()});  
p.then(...).then(...).then(...)
```

此外 Promise 除了 then 方法外，还提供了 Promise.resolve、Promise.all、Promise.race 等等方法。

二、Promise/A+规范

Promise/A+规范扩展了早期的 Promise/A proposal 提案，我们来解读一下 Promise/A+规范。

1) 术语

- (1) “promise”是一个对象或者函数，该对象或者函数有一个 then 方法
- (2) “thenable”是一个对象或者函数，用来定义 then 方法
- (3) “value”是 promise 状态成功时的值
- (4) “reason”是 promise 状态失败时的值

我们明确术语的目的，是为了在自己实现 promise 时，保持代码的规范性（也可以跳过此小节）

2) 要求

- (1) 一个 promise 必须有 3 个状态，pending, fulfilled(resolved), rejected
当处于 pending 状态的时候，可以转移到 fulfilled(resolved)或者 rejected 状态。当处于 fulfilled(resolved)状态或者 rejected 状态的时候，就不可变。

promise 英文译为承诺，也就是说 promise 的状态一旦发生改变，就永远是不可逆的。

(2) 一个 promise 必须有一个 then 方法，then 方法接受两个参数：

```
promise.then(onFulfilled, onRejected)
```

其中 onFulfilled 方法表示状态从 pending——>fulfilled(resolved)时所执行的方法，而 onRejected 表示状态从 pending——>rejected 所执行的方法。

(3) 为了实现链式调用，then 方法必须返回一个 promise

```
promise2=promise1.then(onFulfilled, onRejected)
```

三、实现一个符合 Promise/A+规范的 Promise

解读了 Promise/A+规范之后，下面我们来看如何实现一个 Promise，首先构造一个 myPromise 函数，关于所有变量和函数名，应该与规范中保持相同。

1.v1.0 初始版本 myPromise

```
function myPromise(constructor) {  
  let self=this;  
  self.status="pending" //定义状态改变前的初始状态  
  self.value=undefined;//定义状态为 resolved 的时候的状态  
  self.reason=undefined;//定义状态为 rejected 的时候的状态  
  function resolve(value) {  
    //两个==="pending"，保证了状态的改变是不可逆的  
    if(self.status==="pending") {  
      self.value=value;  
      self.status="resolved";  
    }  
  }  
  function reject(reason) {  
    //两个==="pending"，保证了状态的改变是不可逆的  
    if(self.status==="pending") {  
      self.reason=reason;  
    }  
  }  
}
```

```
        self.status="rejected";
    }
}
//捕获构造异常
try{
    constructor(resolve, reject);
} catch(e) {
    reject(e);
}
}
```

同时，需要在 myPromise 的原型上定义链式调用的 then 方法：

```
myPromise.prototype.then=function(onFullfilled,onRejected){
    let self=this;
    switch(self.status){
        case "resolved":
            onFullfilled(self.value);
            break;
        case "rejected":
            onRejected(self.reason);
            break;
        default:
    }
}
```

上述就是一个初始版本的 myPromise，在 myPromise 里发生状态改变，然后在相应的 then 方法里面根据不同的状态可以执行不同的操作。

```
var p=new myPromise(function(resolve, reject){resolve(1)});
p.then(function(x){console.log(x)})
//输出 1
```

但是这里 myPromise 无法处理异步的 resolve. 比如：

```
var p=new
myPromise(function(resolve,reject){setTimeout(function(){resolve(1)},
1000)});
p.then(function(x){console.log(x)})

//无输出
```

2. v2.0 基于观察模式实现

为了处理异步 resolve，我们修改 myPromise 的定义，用 2 个数组 onFullfilledArray 和 onRejectedArray 来保存异步的方法。在状态发生改变时，一次遍历执行数组中的方法。

```
function myPromise(constructor){
  let self=this;
  self.status="pending" //定义状态改变前的初始状态
  self.value=undefined;//定义状态为 resolved 的时候的状态
  self.reason=undefined;//定义状态为 rejected 的时候的状态
  self.onFullfilledArray=[];
  self.onRejectedArray=[];
  function resolve(value){
    if(self.status==="pending"){
      self.value=value;
      self.status="resolved";
      self.onFullfilledArray.forEach(function(f){
        f(self.value);
        //如果状态从 pending 变为 resolved,
        //那么就遍历执行里面的异步方法
      });
    }
  }
  function reject(reason){
```

```
    if(self.status==="pending"){
        self.reason=reason;
        self.status="rejected";
        self.onRejectedArray.forEach(function(f){
            f(self.reason);
            //如果状态从 pending 变为 rejected,
            //那么就遍历执行里面的异步方法
        })
    }
}
//捕获构造异常
try{
    constructor(resolve, reject);
} catch(e) {
    reject(e);
}
}
```

对于 then 方法，状态为 pending 时，往数组里面添加方法：

```
myPromise.prototype.then=function(onFullfilled,onRejected){
    let self=this;
    switch(self.status){
        case "pending":
            self.onFullfilledArray.push(function(){
                onFullfilled(self.value)
            });
            self.onRejectedArray.push(function(){
                onRejected(self.reason)
            });
        case "resolved":
            onFullfilled(self.value);
            break;
        case "rejected":
```

```
        onRejected(self.reason);
        break;
      default:
    }
  }
}
```

这样，通过两个数组，在状态发生改变之后再开始执行，这样可以处理异步 resolve 无法调用的问题。这个版本的 myPromise 就能处理所有的异步，那么这样做就完整了吗？

没有，我们做 Promise/A+ 规范的最大的特点就是链式调用，也就是说 then 方法返回的应该是一个 promise。

3. v3.0 then 方法实现链式调用

要通过 then 方法实现链式调用，那么也就是说 then 方法每次调用需要返回一个 promise，同时在返回 promise 的构造体里面，增加错误处理部分，我们来改造 then 方法

```
myPromise.prototype.then=function(onFullfilled,onRejected){
  let self=this;
  let promise2;
  switch(self.status){
    case "pending":
      promise2=new myPromise(function(resolve,reject){
        self.onFullfilledArray.push(function(){
          try{
            let temple=onFullfilled(self.value);
            resolve(temple)
          }catch(e){
            reject(e) //error catch
          }
        });
      });
  }
}
```



```
self.onRejectedArray.push(function() {
    try{
        let temple=onRejected(self.reason);
        reject(temple)
    }catch(e) {
        reject(e)// error catch
    }
});
})
case "resolved":
    promise2=new myPromise(function(resolve, reject) {
        try{
            let temple=onFullfilled(self.value);
            //将上次一 then 里面的方法传递进下一个 Promise 的状态
            resolve(temple);
        }catch(e) {
            reject(e);//error catch
        }
    })
    break;
case "rejected":
    promise2=new myPromise(function(resolve, reject) {
        try{
            let temple=onRejected(self.reason);
            //将 then 里面的方法传递到下一个 Promise 的状态里
            resolve(temple);
        }catch(e) {
            reject(e);
        }
    })
    break;
default:
```

```
    }  
    return promise2;  
  }  
}
```

这样通过 then 方法返回一个 promise 就可以实现链式的调用：

```
p.then(function(x){console.log(x)}).then(function(){console.log("链式  
调用 1")}).then(function(){console.log("链式调用 2")})  
//输出  
1  
链式调用 1  
链式调用 2
```

这样我们虽然实现了 then 函数的链式调用，但是还有一个问题，就是在 Promise/A+ 规范中 then 函数里面的 onFulfilled 方法和 onRejected 方法的返回值可以是对象，函数，甚至是另一个 promise。

4. v4.0 then 函数中的 onFulfilled 和 onRejected 方法的返回值问题

特别的为了解决 onFulfilled 和 onRejected 方法的返回值可能是一个 promise 的问题。

(1) 首先来看 promise 中对于 onFulfilled 函数的返回值的要求

I) 如果 onFulfilled 函数返回的是该 promise 本身，那么会抛出类型错误

II) 如果 onFulfilled 函数返回的是一个不同的 promise，那么执行该 promise 的 then 函数，在 then 函数里将这个 promise 的状态转移给新的 promise

III) 如果返回的是一个嵌套类型的 promise，那么需要递归。

IV) 如果返回的是非 promise 的对象或者函数，那么会选择直接将该对象或者函数，给新的 promise。

根据上述返回值的要求，我们要重新的定义 resolve 函数，这里 Promise/A+ 规范里面称为：resolvePromise 函数，该函数接受当前的 promise、onFulfilled 函数或者 onRejected 函数的返回值、resolve 和 reject 作为参数。

下面我们来看 resolvePromise 函数的定义：

```
function resolvePromise(promise, x, resolve, reject) {
  if(promise===x) {
    throw new TypeError("type error")
  }
  let isUsed;
  if(x!==null&&(typeof x==="object"||typeof x==="function")) {
    try{
      let then=x.then;
      if(typeof then==="function") {
        //是一个 promise 的情况
        then.call(x, function(y) {
          if(isUsed)return;
          isUsed=true;
          resolvePromise(promise, y, resolve, reject);
        }, function(e) {
          if(isUsed)return;
          isUsed=true;
          reject(e);
        })
      }else{
        //仅仅是一个函数或者是对象
        resolve(x)
      }
    }
  }
```

```
    }catch(e) {
        if(isUsed)return;
        isUsed=true;
        reject(e);
    }
}else{
    //返回的基本类型，直接 resolve
    resolve(x)
}
}
```

改变了 resolvePromise 函数之后，我们在 then 方法里面的调用也变成了 resolvePromise 而不是 promise。

```
myPromise.prototype.then=function(onFullfilled,onRejected) {
    let self=this;
    let promise2;
    switch(self.status) {
        case "pending":
            promise2=new myPromise(function(resolve,reject) {
                self.onFullfilledArray.push(function() {
                    setTimeout(function() {
                        try{
                            let temple=onFullfilled(self.value);
                            resolvePromise(temple)
                        }catch(e) {
                            reject(e) //error catch
                        }
                    })
                });
            self.onRejectedArray.push(function() {
                setTimeout(function() {
                    try{
```

```
        let temple=onRejected(self.reason);
        resolvePromise(temple)
    }catch(e) {
        reject(e)// error catch
    }
    })
    });
    })
case "resolved":
    promise2=new myPromise(function(resolve,reject) {
        setTimeout(function() {
            try{
                let temple=onFullfilled(self.value);
                //将上次一 then 里面的方法传递进下一个 Promise 状态
                resolvePromise(temple);
            }catch(e) {
                reject(e);//error catch
            }
        })
    })
    break;
case "rejected":
    promise2=new myPromise(function(resolve,reject) {
        setTimeout(function() {
            try{
                let temple=onRejected(self.reason);
                //将 then 里面的方法传递到下一个 Promise 的状态里
                resolvePromise(temple);
            }catch(e) {
                reject(e);
            }
        })
    })
```

```
    })  
    break;  
    default:  
  }  
  return promise2;  
}
```

这样就能处理 onFullfilled 各种返回值的情况。

```
var p=new Promise(function(resolve,reject){resolve("初始化 promise")})  
p.then(function(){return new  
Promise(function(resolve,reject){resolve("then 里面的 promise 返回值  
")}})).then(function(x){console.log(x)})
```

//输出

then 里面 promise 的返回值

到这里可能有点乱，我们再理一理，首先返回值有两个：

then 函数的返回值——>返回一个新 promise，从而实现链式调用

then 函数中的 onFullfilled 和 onRejected 方法——>返回基本值或者新的 promise

这两者其实是有关联的，onFullfilled 方法的返回值可以决定 then 函数的返回值。

四、检测是否完全符合 Promise/A+规范

```
npm install -g promises-aplus-tests
```

具体用法请看 promise test 然后

promises-aplus-tests myPromise.js

说明我们的实现完全符合 Promise/A+规范。

五、最后补充 Typescript 实现的 Promise/A+规范（可以忽略此节）

```
interface IConstructor{
  (resolve:IResolve,reject:IReject):void
}
interface IResolve {
  (x:any):void
}
interface IReject {
  (x:any):void
}
function myPromise(constructor:IConstructor):void{
  let self:object=this;
  self.status="pending";
  self.value=undefined;//if pending->resolved
  self.reason=undefined;//if pending->rejected
  self.onFullfilledArray=[];//to deal with async(resolved)
  self.onRejectedArray=[];//to deal with async(rejected)
  let resolve:IResolve;
  resolve=function(value:any):void{
    //pending->resolved
    if(self.status==="pending"){
      self.status="resolved";
      self.value=value;
      self.onFullfilledArray.forEach(function(f){
        f(self.value);
      })
    }
  }
}
```

```
let reject:IReject;
reject=function(reason:any):void{
  if(self.status==="pending"){
    self.status="rejected";
    self.reason=reason;
    self.onRejectedArray.forEach(function(f){
      f(self.reason);
    })
  }
}
//According to the definition that the function "constructor" accept
two parameters
//error catch
try {
  constructor(resolve,reject);
} catch (e) {
  reject(e);
}
}
```

159. Function.__proto__(getPrototypeOf)是什么？

参考答案：

获取一个对象的原型，在 chrome 中可以通过__proto__的形式，或者在 ES6 中可以通过 Object.getPrototypeOf 的形式。

那么 Function.proto 是什么么？也就是说 Function 由什么对象继承而来，我们来做如下判别。

```
Function.__proto__==Object.prototype //false
Function.__proto__==Function.prototype//true
```


我们发现 Function 的原型也是 Function。

160. 实现 js 中所有对象的深度克隆（包装对象，Date 对象，正则对象）

参考答案：

通过递归可以简单实现对象的深度克隆，但是这种方法不管是 ES6 还是 ES5 实现，都有同样的缺陷，就是只能实现特定的 object 的深度复制（比如数组和函数），不能实现包装对象 Number，String，Boolean，以及 Date 对象，RegExp 对象的复制。

(1) 前文的方法

```
function deepClone(obj) {  
  var newObj= obj instanceof Array?[]: {};  
  for(var i in obj){  
    newObj[i]=typeof obj[i]=='object'?  
    deepClone(obj[i]):obj[i];  
  }  
  return newObj;  
}
```

这种方法可以实现一般对象和数组对象的克隆，比如：

```
var arr=[1,2,3];  
var newArr=deepClone(arr);  
// newArr->[1,2,3]  
var obj={  
  x:1,  
  y:2  
}
```

```
var newObj=deepClone(obj);  
// newObj={x:1,y:2}
```

但是不能实现例如包装对象 Number, String, Boolean, 以及正则对象 RegExp 和 Date 对象的克隆, 比如:

```
//Number 包装对象  
var num=new Number(1);  
typeof num // "object"  
var newNum=deepClone(num);  
//newNum -> {} 空对象
```

```
//String 包装对象  
var str=new String("hello");  
typeof str //"object"  
var newStr=deepClone(str);  
//newStr-> {0:'h',1:'e',2:'l',3:'l',4:'o'};
```

```
//Boolean 包装对象
```

```
var bol=new Boolean(true);  
typeof bol //"object"  
var newBol=deepClone(bol);  
// newBol -> {} 空对象
```

....

(2)valueOf() 函数

所有对象都有 valueOf 方法, valueOf 方法对于: 如果存在任意原始值, 它就默认将对象转换为表示它的原始值。对象是复合值, 而且大多数对象无法真正表示为一个原始值, 因此默认的 valueOf() 方法简单地返回对象本身, 而不是返回一

个原始值。数组、函数和正则表达式简单地继承了这个默认方法，调用这些类型的实例的 `valueOf()` 方法只是简单返回这个对象本身。

对于原始值或者包装类：

```
function baseClone(base) {  
  return base.valueOf();  
}  
  
//Number  
var num=new Number(1);  
var newNum=baseClone(num);  
//newNum->1  
  
//String  
var str=new String('hello');  
var newStr=baseClone(str);  
// newStr->"hello"  
  
//Boolean  
var bol=new Boolean(true);  
var newBol=baseClone(bol);  
//newBol-> true
```

其实对于包装类，完全可以用 `=` 号来进行克隆，其实没有深度克隆一说，这里用 `valueOf` 实现，语法上比较符合规范。

对于 `Date` 类型：

因为 `valueOf` 方法，日期类定义的 `valueOf()` 方法会返回它的一个内部表示：1970 年 1 月 1 日以来的毫秒数。因此我们可以在 `Date` 的原型上定义克隆的方法：

```
Date.prototype.clone=function() {  
  return new Date(this.valueOf());  
}  
  
var date=new Date('2010');
```

```
var newDate=date.clone();
// newDate-> Fri Jan 01 2010 08:00:00 GMT+0800
对于正则对象 RegExp:
RegExp.prototype.clone = function() {
var pattern = this.valueOf();
var flags = '';
flags += pattern.global ? 'g' : '';
flags += pattern.ignoreCase ? 'i' : '';
flags += pattern.multiline ? 'm' : '';
return new RegExp(pattern.source, flags);
};
var reg=new RegExp('/111/');
var newReg=reg.clone();
//newReg-> /\111\//
```

161. 简单实现 Node 的 Events 模块

参考答案:

简介：观察者模式或者说订阅模式，它定义了对象间的一种一对多的关系，让多个观察者对象同时监听某一个主题对象，当一个对象发生改变时，所有依赖于它的对象都将得到通知。

node 中的 Events 模块就是通过观察者模式来实现的:

```
var events=require('events');
var eventEmitter=new events.EventEmitter();
eventEmitter.on('say',function(name){
console.log('Hello',name);
})
eventEmitter.emit('say','Jony_yu');
```

这样，eventEmitter 发出 say 事件，通过 On 接收，并且输出结果，这就是一个

订阅模式的实现，下面我们来简单的实现一个 Events 模块的 EventEmitter。

(1)实现简单的 Event 模块的 emit 和 on 方法

```
function Events() {
  this.on=function(eventName, callBack) {
    if(!this.handles) {
      this.handles={};
    }
    if(!this.handles[eventName]) {
      this.handles[eventName]=[];
    }
    this.handles[eventName].push(callBack);
  }
  this.emit=function(eventName, obj) {
    if(this.handles[eventName]) {
      for(var i=0;i<this.handles[eventName].length;i++) {
        this.handles[eventName][i](obj);
      }
    }
  }
  return this;
}
```

这样我们就定义了 Events，现在我们可以开始来调用：

```
var events=new Events();
events.on('say',function(name) {
  console.log('Hello', name);
});
events.emit('say', 'Jony yu');
//结果就是通过 emit 调用之后，输出了 Jony yu
```

(2) 每个对象是独立的

因为是通过 new 的方式，每次生成的对象都是不相同的，因此：

```
var event1=new Events();
var event2=new Events();
event1.on('say',function(){
  console.log('Jony event1');
});
event2.on('say',function(){
  console.log('Jony event2');
})
event1.emit('say');
event2.emit('say');
//event1、event2 之间的事件监听互相不影响
//输出结果为'Jony event1' 'Jony event2'
```

162. 箭头函数中 this 指向举例

参考答案：

```
var a=11;
function test2(){
  this.a=22;
  let b()=>{console.log(this.a)}
  b();
}
var x=new test2();
//输出 22
定义时绑定。
```

163. js 判断类型

参考答案:

判断方法: `typeof()`, `instanceof`, `Object.prototype.toString.call()` 等 a

164. 数组常用方法

参考答案:

`push()`, `pop()`, `shift()`, `unshift()`, `splice()`, `sort()`, `reverse()`, `map()` 等

165. 数组去重

参考答案:

法一: `indexOf` 循环去重

法二: ES6 Set 去重; `Array.from(new Set(array))`

法三: Object 键值对去重; 把数组的值存成 Object 的 key 值, 比如 `Object[value1] = true`, 在判断另一个值的时候, 如果 `Object[value2]` 存在的话, 就说明该值是重复的。

166. 闭包 有什么用

参考答案:

(1) 什么是闭包:

闭包是指有权访问另外一个函数作用域中的变量的函数。

闭包就是函数的局部变量集合, 只是这些局部变量在函数返回后会继续存在。闭包就是就是函数的“堆栈”在函数返回后并不释放, 我们也可以理解为这些函数堆栈并不在栈上分配而是在堆上分配。当在一个函数内定义另外一个函数就会产

生闭包。

(2) 为什么要用：

匿名自执行函数：我们知道所有的变量，如果不加上 var 关键字，则默认会添加到全局对象的属性上去，这样的临时变量加入全局对象有很多坏处，比如：别的函数可能误用这些变量；造成全局对象过于庞大，影响访问速度(因为变量的取值是需要从原型链上遍历的)。除了每次使用变量都是用 var 关键字外，我们在实际情况下经常遇到这样一种情况，即有的函数只需要执行一次，其内部变量无需维护，可以用闭包。

结果缓存：我们开发中会碰到很多情况，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算，然后更新缓存并返回值，如果找到了，直接返回查找到的值即可。闭包正是可以做到这一点，因为它不会释放外部的引用，从而函数内部的值可以得以保留。

封装：实现类和继承等。

167. 事件代理在捕获阶段的实际应用

参考答案：

可以在父元素层面阻止事件向子元素传播，也可代替子元素执行某些操作。

168. 去除字符串首尾空格

参考答案：

使用正则 `(^\s*)|(\s*$)` 即可

169. 性能优化

参考答案：

减少 HTTP 请求

使用内容发布网络（CDN）

添加本地缓存

压缩资源文件

将 CSS 样式表放在顶部，把 javascript 放在底部（浏览器的运行机制决定）

避免使用 CSS 表达式

减少 DNS 查询

使用外部 javascript 和 CSS

避免重定向

图片 lazyLoad

170. 能来讲讲 JS 的语言特性吗

参考答案：

运行在客户端浏览器上；

不用预编译，直接解析执行代码；

是弱类型语言，较为灵活；

与操作系统无关，跨平台的语言；

脚本语言、解释性语言

171. 如何判断一个数组

参考答案：

`Object.prototype.call.toString()`

`Instanceof`

172. 你说到 typeof，能不能加一个限制条件达到判断条件

参考答案：

typeof 只能判断是 object, 可以判断一下是否拥有数组的方法

173. JS 实现跨域

参考答案：

JSONP：通过动态创建 script，再请求一个带参网址实现跨域通信。

document.domain + iframe 跨域：两个页面都通过 js 强制设置 document.domain 为基础主域，就实现了同域。

location.hash + iframe 跨域：a 欲与 b 跨域相互通信，通过中间页 c 来实现。三个页面，不同域之间利用 iframe 的 location.hash 传值，相同域之间直接 js 访问来通信。

window.name + iframe 跨域：通过 iframe 的 src 属性由外域转向本地域，跨域数据即由 iframe 的 window.name 从外域传递到本地域。

postMessage 跨域：可以跨域操作的 window 属性之一。

CORS：服务端设置 Access-Control-Allow-Origin 即可，前端无须设置，若要带 cookie 请求，前后端都需要设置。

代理跨域：启一个代理服务器，实现数据的转发

174. Js 基本数据类型

参考答案：

基本数据类型：undefined、null、number、boolean、string、symbol

175. js 深度拷贝一个元素的具体实现

参考答案：

```
var deepCopy = function(obj) {  
  if (typeof obj !== 'object') return;  
  var newObj = obj instanceof Array ? [] : {};  
  for (var key in obj) {  
    if (obj.hasOwnProperty(key)) {  
      newObj[key] = typeof obj[key] === 'object' ? deepCopy(obj[key]) :  
        obj[key];  
    }  
  }  
  return newObj;  
}
```

176. 之前说了 ES6set 可以数组去重，是否还有数组去重的

参考答案：

```
function permutate(str) {  
  var result = [];  
  if(str.length > 1) {  
    var left = str[0];  
    var rest = str.slice(1, str.length);  
    var preResult = permutate(rest);  
    for(var i=0; i<preResult.length; i++) {  
      for(var j=0; j<preResult[i].length; j++) {  
        var tmp = preResult[i].slice(0, j) + left + preResult[i].slice(j,  
          preResult[i].length);  
        result.push(tmp);  
      }  
    }  
  }  
}
```

```
}  
}  
} else if (str.length == 1) {  
  return [str];  
}  
return result;  
}
```

177. 跨域的原理

参考答案：

跨域，是指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 实施的安全限制，那么只要协议、域名、端口有任何一个不同，都被当作是不同的域。跨域原理，即是通过各种方式，避开浏览器的安全限制。

178. 不同数据类型的值的比较，是怎么转换的，有什么规则

参考答案：

比较运算 $x==y$, 其中 x 和 y 是值, 产生 $true$ 或者 $false$ 。这样的比较按如下方式进行:

1. 若 $Type(x)$ 与 $Type(y)$ 相同, 则
 - a. 若 $Type(x)$ 为 $Undefined$, 返回 $true$ 。
 - b. 若 $Type(x)$ 为 $Null$, 返回 $true$ 。
 - c. 若 $Type(x)$ 为 $Number$, 则
 - i. 若 x 为 NaN , 返回 $false$ 。
 - ii. 若 y 为 NaN , 返回 $false$ 。
 - iii. 若 x 与 y 为相等数值, 返回 $true$ 。
 - iv. 若 x 为 $+0$ 且 y 为 -0 , 返回 $true$ 。
 - v. 若 x 为 -0 且 y 为 $+0$, 返回 $true$ 。
 - vi. 返回 $false$ 。
 - d. 若 $Type(x)$ 为 $String$, 则当 x 和 y 为完全相同的字符序列(长度相等且相同字符在相同位置)时返回 $true$ 。否则, 返回 $false$ 。
 - e. 若 $Type(x)$ 为 $Boolean$, 当 x 和 y 同为 $true$ 或者同为 $false$ 时返回 $true$ 。否则, 返回 $false$ 。
 - f. 当 x 和 y 为引用同一对象时返回 $true$ 。否则, 返回 $false$ 。
 2. 若 x 为 $null$ 且 y 为 $undefined$, 返回 $true$ 。
 3. 若 x 为 $undefined$ 且 y 为 $null$, 返回 $true$ 。
 4. 若 $Type(x)$ 为 $Number$ 且 $Type(y)$ 为 $String$, 返回 $comparison\ x == ToNumber(y)$ 的结果。
 5. 若 $Type(x)$ 为 $String$ 且 $Type(y)$ 为 $Number$,
 6. 返回比较 $ToNumber(x) == y$ 的结果。
 7. 若 $Type(x)$ 为 $Boolean$, 返回比较 $ToNumber(x) == y$ 的结果。
 8. 若 $Type(y)$ 为 $Boolean$, 返回比较 $x == ToNumber(y)$ 的结果。
 9. 若 $Type(x)$ 为 $String$ 或 $Number$, 且 $Type(y)$ 为 $Object$, 返回比较 $x == ToPrimitive(y)$ 的结果。
 10. 若 $Type(x)$ 为 $Object$ 且 $Type(y)$ 为 $String$ 或 $Number$, 返回比较 $ToPrimitive(x) == y$ 的结果。
 11. 返回 $false$ 。
-

179. $null == undefined$ 为什么

参考答案:

要比较相等性之前, 不能将 $null$ 和 $undefined$ 转换成其他任何值, 但 $null == undefined$ 会返回 $true$ 。ECMAScript规范中是这样定义的。

180. $this$ 的指向 哪几种

参考答案:

默认绑定: 全局环境中, $this$ 默认绑定到 $window$ 。

隐式绑定: 一般地, 被直接对象所包含的函数调用时, 也称为方法调用, $this$ 隐式绑定到该直接对象。

隐式丢失：隐式丢失是指被隐式绑定的函数丢失绑定对象，从而默认绑定到 window。显式绑定：通过 `call()`、`apply()`、`bind()` 方法把对象绑定到 `this` 上，叫做显式绑定。

new 绑定：如果函数或者方法调用之前带有关键字 `new`，它就构成构造函数调用。对于 `this` 绑定来说，称为 new 绑定。

【1】构造函数通常不使用 `return` 关键字，它们通常初始化新对象，当构造函数的函数体执行完毕时，它会显式返回。在这种情况下，构造函数调用表达式的计算结果就是这个新对象的值。

【2】如果构造函数使用 `return` 语句但没有指定返回值，或者返回一个原始值，那么这时将忽略返回值，同时使用这个新对象作为调用结果。

【3】如果构造函数显式地使用 `return` 语句返回一个对象，那么调用表达式的值就是这个对象。

181. 暂停死区

参考答案：

在代码块内，使用 `let`、`const` 命令声明变量之前，该变量都是不可用的。这在语法上，称为“暂时性死区”。

182. AngularJS 双向绑定原理

参考答案：

Angular 将双向绑定转换为一堆 `watch` 表达式，然后递归这些表达式检查是否发生过变化，如果变了则执行相应的 `watcher` 函数（指 `view` 上的指令，如 `ng-bind`，

ng-show 等或是{{}})。等到 model 中的值不再发生变化，也就不会再有 watcher 被触发，一个完整的 digest 循环就完成了。

Angular 中在 view 上声明的事件指令，如：ng-click、ng-change 等，会将浏览器的事件转发给\$scope 上相应的 model 的响应函数。等待相应函数改变 model，紧接着触发脏检查机制刷新 view。

watch 表达式：可以是一个函数、可以是\$scope 上的一个属性名，也可以是一个字符串形式的表达式。\$watch 函数所监听的对象叫做 watch 表达式。watcher 函数：指在 view 上的指令（ngBind, ngShow、ngHide 等）以及{{}}表达式，他们所注册的函数。每一个 watcher 对象都包括：监听函数，上次变化的值，获取监听表达式的方法以及监听表达式，最后还包括是否需要使用深度对比（angular.equals()）

183. 写一个深度拷贝

参考答案：

```
function clone( obj ) {  
  var copy;  
  switch( typeof obj ) {  
    case "undefined":  
      break;  
    case "number":  
      copy = obj - 0;  
      break;  
    case "string":  
      copy = obj + "";  
      break;  
    case "boolean":  
      copy = obj;  
  }
```

```
break;
case "object": //object 分为两种情况 对象 (Object) 和数组 (Array)

if(obj === null) {
copy = null;
} else {
if( Object.prototype.toString.call(obj).slice(8, -1) === "Array") {
copy = [];
for( var i = 0 ; i < obj.length ; i++ ) {
copy.push(clone(obj[i]));
}
} else {
copy = {};
for( var j in obj) {
copy[j] = clone(obj[j]);
}
}
}
break;
default:
copy = obj;
break;
}
return copy;
}
```

183. 简历中提到了 requestAnimationFrame，请问是怎么使用的

参考答案：

requestAnimationFrame() 方法告诉浏览器您希望执行动画并请求浏览器在下

一次重绘之前调用指定的函数来更新动画。该方法使用一个回调函数作为参数，这个回调函数会在浏览器重绘之前调用。

184. 有一个游戏叫做 Flappy Bird，就是一只小鸟在飞，前面是无尽的沙漠，上下不断有钢管生成，你要躲避钢管。然后小明在玩这个游戏时候老是卡顿甚至崩溃，说出原因（3-5 个）以及解决办法（3-5 个）

参考答案：

原因可能是：

- 1) 内存溢出问题。
- 2) 资源过大问题。
- 3) 资源加载问题。
- 4) canvas 绘制频率问题

解决办法：

- 1) 针对内存溢出问题，我们应该在钢管离开可视区域后，销毁钢管，让垃圾收集器回收钢管，因为不断生成的钢管不及时清理容易导致内存溢出游戏崩溃。
- 2) 针对资源过大问题，我们应该选择图片文件大小更小的图片格式，比如使用 webp、png 格式的图片，因为绘制图片需要较大计算量。
- 3) 针对资源加载问题，我们应该在可视区域之前就预加载好资源，如果在可视区域生成钢管的话，用户的体验就认为钢管是卡顿后才生成的，不流畅。
- 4) 针对 canvas 绘制频率问题，我们应该需要知道大部分显示器刷新频率为 60 次/s, 因此游戏的每一帧绘制间隔时间需要小于 $1000/60=16.7\text{ms}$ ，才能让用户觉得不卡顿。

（注意因为这是单机游戏，所以回答与网络无关）

185. 编写代码，满足以下条件：

(1) `Hero("37er");`执行结果为

Hi! This is 37er

(2) `Hero("37er").kill(1).recover(30);`执行结果为

Hi! This is 37er

Kill 1 bug

Recover 30 bloods

(3) `Hero("37er").sleep(10).kill(2)`执行结果为

Hi! This is 37er

//等待 10s 后

Kill 2 bugs //注意为 bugs

(双斜线后的为提示信息，不需要打印)

参考答案：

```
function Hero(name) {
  let o=new Object();
  o.name=name;
  o.time=0;
  console.log("Hi! This is "+o.name);
  o.kill=function(bugs) {
    if(bugs==1){
      console.log("Kill "+(bugs)+" bug");
    }else {
      setTimeout(function () {
        console.log("Kill " + (bugs) + " bugs");
      }, 1000 * this.time);
    }
  }
  return o;
}
```

```
};  
o.recover=function (bloods) {  
  console.log("Recover "+(bloods)+" bloods");  
  return o;  
}  
o.sleep=function (sleepTime) {  
  o.time=sleepTime;  
  return o;  
}  
return o;  
}
```

186. 什么是按需加载

参考答案:

当用户触发了动作时才加载对应的功能。触发的动作，是要看具体的业务场景而言，包括但不限于以下几个情况：鼠标点击、输入文字、拉动滚动条，鼠标移动、窗口大小更改等。加载的文件，可以是 JS、图片、CSS、HTML 等。

187. 说一下什么是 virtual dom

参考答案:

用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中 当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异 把所记录的差异应用到所构建的真正的 DOM 树上，视图就更新了。Virtual DOM 本质上就是在 JS 和 DOM 之间做了一个缓存。

188. webpack 用来干什么的

参考答案:

webpack 是一个现代 JavaScript 应用程序的静态模块打包器 (module bundler)。当 webpack 处理应用程序时，它会递归地构建一个依赖关系图 (dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle。

189. ant-design 优点和缺点

参考答案:

优点：组件非常全面，样式效果也都不错。

缺点：框架自定义程度低，默认 UI 风格修改困难。

190. JS 中继承实现的几种方式

参考答案:

1) 原型链继承，将父类的实例作为子类的原型，他的特点是实例是子类的实例也是父类的实例，父类新增的原型方法/属性，子类都能够访问，并且原型链继承简单易于实现，缺点是来自原型对象的所有属性被所有实例共享，无法实现多继承，无法向父类构造函数传参。

2) 构造继承，使用父类的构造函数来增强子类实例，即复制父类的实例属性给子类，构造继承可以向父类传递参数，可以实现多继承，通过 call 多个父类对象。但是构造继承只能继承父类的实例属性和方法，不能继承原型属性和方法，无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

3) 实例继承，为父类实例添加新特性，作为子类实例返回，实例继承的特点是不限制调用方法，不管是 new 子类（）还是子类（）返回的对象具有相同的效果，缺点是实例是父类的实例，不是子类的实例，不支持多继承

4) 拷贝继承：特点：支持多继承，缺点：效率较低，内存占用高（因为要拷贝父类的属性）无法获取父类不可枚举的方法（不可枚举方法，不能使用 for in 访问到）

5) 组合继承：通过调用父类构造，继承父类的属性并保留传参的优点，然后将父类实例作为子类原型，实现函数复用

6) 寄生组合继承：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性，避免的组合继承的缺点

191. 写一个函数，第一秒打印 1，第二秒打印 2

参考答案：

两个方法，第一个是用 let 块级作用域

```
for(let i=0;i<5;i++){
  setTimeout(function(){
    console.log(i)
  },1000*i)
}
```

第二个方法闭包

```
for(var i=0;i<5;i++){
  (function(i){
    setTimeout(function(){
      console.log(i)
    },1000*i)
  })(i)
}
```

192. vue 的生命周期

参考答案：

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

每一个组件或者实例都会经历一个完整的生命周期，总共分为三个阶段：初始化、运行中、销毁。

实例、组件通过 `new Vue()` 创建出来之后会初始化事件和生命周期，然后就会执行 `beforeCreate` 钩子函数，这个时候，数据还没有挂载呢，只是一个空壳，无法访问到数据和真实的 dom，一般不做操作

挂载数据，绑定事件等等，然后执行 `created` 函数，这个时候已经可以使用到数据，也可以更改数据，在这里更改数据不会触发 `updated` 函数，在这里可以在渲染前倒数第二次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取

接下来开始找实例或者组件对应的模板，编译模板为虚拟 dom 放入到 `render` 函数中准备渲染，然后执行 `beforeMount` 钩子函数，在这个函数中虚拟 dom 已经创建完成，马上就要渲染，在这里也可以更改数据，不会触发 `updated`，在这里可以在渲染前最后一次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取

接下来开始 `render`，渲染出真实 dom，然后执行 `mounted` 钩子函数，此时，组件已经出现在页面中，数据、真实 dom 都已经处理好了，事件都已经挂载好了，可以在这里操作真实 dom 等事情...

当组件或实例的数据更改之后，会立即执行 `beforeUpdate`，然后 vue 的虚拟 dom 机制会重新构建虚拟 dom 与上一次的虚拟 dom 树利用 `diff` 算法进行对比之后重

新渲染，一般不做什么事儿

当更新完成后，执行 `updated`，数据已经更改完成，`dom` 也重新 `render` 完成，可以操作更新后的虚拟 `dom`

当经过某种途径调用 `$destroy` 方法后，立即执行 `beforeDestroy`，一般在这里做一些善后工作，例如清除计时器、清除非指令绑定的事件等等

组件的数据绑定、监听... 去掉后只剩下 `dom` 空壳，这个时候，执行 `destroyed`，在这里做善后工作也可以

193. 简单介绍一下 `symbol`

参考答案：

`Symbol` 是 ES6 的新增属性，代表用给定名称作为唯一标识，这种类型的值可以这样创建，`let id=symbol(“id”)`

`Symbol` 确保唯一，即使采用相同的名称，也会产生不同的值，我们创建一个字段，仅为知道对应 `symbol` 的人能访问，使用 `symbol` 很有用，`symbol` 并不是 100% 隐藏，有内置方法 `Object.getOwnPropertySymbols(obj)` 可以获得所有的 `symbol`。也有一个方法 `Reflect.ownKeys(obj)` 返回对象所有的键，包括 `symbol`。

所以并不是真正隐藏。但大多数库内置方法和语法结构遵循通用约定他们是隐藏的。

194. 什么是事件监听

参考答案：

`addEventListener()` 方法，用于向指定元素添加事件句柄，它可以更简单的控制

事件，语法为

```
element.addEventListener(event, function, useCapture);
```

第一个参数是事件的类型(如 "click" 或 "mousedown").

第二个参数是事件触发后调用的函数。

第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。

事件传递有两种方式，冒泡和捕获

事件传递定义了元素事件触发的顺序，如果你将 P 元素插入到 div 元素中，用户点击 P 元素，

在冒泡中，内部元素先被触发，然后再触发外部元素，

捕获中，外部元素先被触发，在触发内部元素，

195. 介绍一下 promise，及其底层如何实现

参考答案：

Promise 是一个对象，保存着未来将要结束的事件，她有两个特征：

1) 对象的状态不受外部影响，Promise 对象代表一个异步操作，有三种状态，pending 进行中，fulfilled 已成功，rejected 已失败，只有异步操作的结果，才可以决定当前是哪一种状态，任何其他操作都无法改变这个状态，这也就是 promise 名字的由来。

2) 一旦状态改变，就不会再变，promise 对象状态改变只有两种可能，从 pending 改到 fulfilled 或者从 pending 改到 rejected，只要这两种情况发生，状态就凝固了，不会再改变，这个时候就称为定型 resolved，

Promise 的基本用法：

```
let promise1 = new Promise(function(resolve, reject) {  
  setTimeout(function() {  
    resolve('ok')  
  }, 1000);  
});
```



```
    }, 1000)
  })
  promise1.then(function success(val) {
    console.log(val)
  })
}
```

最简单代码实现 promise

```
class PromiseM {
  constructor (process) {
    this.status = 'pending'
    this.msg = ''
    process(this.resolve.bind(this), this.reject.bind(this))
    return this
  }
  resolve (val) {
    this.status = 'fulfilled'
    this.msg = val
  }
  reject (err) {
    this.status = 'rejected'
    this.msg = err
  }
  then (fulfilled, reject) {
    if(this.status === 'fulfilled') {
      fulfilled(this.msg)
    }
    if(this.status === 'rejected') {
      reject(this.msg)
    }
  }
}

//测试代码
```

```
var mm=new PromiseM(function(resolve,reject){
  resolve('123');
});
mm.then(function(success){
  console.log(success);
},function(){
  console.log('fail!');
});
```

196. bootstrap 清除浮动的方法

参考答案:

```
.clearfix:before,
.clearfix:after {
  content: " ";
  display: table;
}
.clearfix:after {
  clear: both;
}
/**
 * For IE 6/7 only
 */
.clearfix {
  *zoom: 1;
}
```

:after 伪类在元素末尾插入了一个包含空格的字符，并设置 display 为 table

display:table 会创建一个匿名的 table-cell，从而触发块级上下文（BFC），因为容器内 float 的元素也是 BFC，由于 BFC 有不能互相重叠的特性，并且设置

了 `clear: both`, `:after` 插入的元素会被挤到容器底部, 从而将容器撑高。并且设置 `display: table` 后, `content` 中的空格字符会被渲染为 `0*0` 的空白元素, 不会占用页面空间。

`content` 包含一个空格, 是为了解决 Opera 浏览器的 BUG。当 HTML 中包含 `contenteditable` 属性时, 如果 `content` 没有包含空格, 会造成清除浮动元素的顶部、底部有一个空白 (设置 `font-size: 0` 也可以解决这个问题)。

`:after` 伪类的设置已经达到了清除浮动的目的, 但还要设置 `:before` 伪类, 原因如下

`:before` 的设置也触发了一个 BFC, 由于 BFC 有内部布局不受外部影响的特性, 因此 `:before` 的设置可以阻止 `margin-top` 的合并。

这样做, 其一是为了和其他清除浮动的方式的效果保持一致; 其二, 是为了与 ie6/7 下设置 `zoom: 1` 后的效果一致 (即阻止 `margin-top` 合并的效果)。

`zoom: 1` 用于在 ie6/7 下触发 `haslayout` 和 `contain floats`

197. 说说 C++, Java, JavaScript 这三种语言的区别

参考答案:

从静态类型还是动态类型来看:

静态类型, 编译的时候就能够知道每个变量的类型, 编程的时候也需要给定类型, 如 Java 中的整型 `int`, 浮点型 `float` 等。C、C++、Java 都属于静态类型语言。

动态类型，运行的时候才知道每个变量的类型，编程的时候无需显示指定类型，如 JavaScript 中的 var、PHP 中的\$。JavaScript、Ruby、Python 都属于动态类型语言。

静态类型还是动态类型对语言的性能有很大影响。

对于静态类型，在编译后会大量利用已知类型的优势，如 int 类型，占用 4 个字节，编译后的代码就可以用内存地址加偏移量的方法存取变量，而地址加偏移量的算法汇编很容易实现。

对于动态类型，会当做字符串通通存下来，之后存取就用字符串匹配。

从编译型还是解释型来看：

编译型语言，像 C、C++，需要编译器编译成本地可执行程序后才能运行，由开发人员在编写完成后手动实施。用户只使用这些编译好的本地代码，这些本地代码由系统加载器执行，由操作系统的 CPU 直接执行，无需其他额外的虚拟机等。

源代码=》抽象语法树=》中间表示=》本地代码

解释性语言，像 JavaScript、Python，开发语言写好后直接将代码交给用户，用户使用脚本解释器将脚本文件解释执行。对于脚本语言，没有开发人员的编译过程，当然，也不绝对。

源代码=》抽象语法树=》解释器解释执行。

对于 JavaScript，随着 Java 虚拟机 JIT 技术的引入，工作方式也发生了改变。

可以将抽象语法树转成中间表示（字节码），再转成本地代码，如 JavaScriptCore，这样可以大大提高执行效率。也可以从抽象语法树直接转成本地代码，如 V8

Java 语言，分为两个阶段。首先像 C++ 语言一样，经过编译器编译。和 C++ 的不同，C++ 编译生成本地代码，Java 编译后，生成字节码，字节码与平台无关。第二阶段，由 Java 的运行环境也就是 Java 虚拟机运行字节码，使用解释器执行这些代码。一般情况下，Java 虚拟机都引入了 JIT 技术，将字节码转换成本地代码来提高执行效率。

注意，在上述情况中，编译器的编译过程没有时间要求，所以编译器可以做大量的代码优化措施。

对于 JavaScript 与 Java 它们还有的不同：

对于 Java，Java 语言将源代码编译成字节码，这个同执行阶段是分开的。也就是从源代码到抽象语法树到字节码这段时间的长短是无所谓的。

对于 JavaScript，这些都是在网页和 JavaScript 文件下载后同执行阶段一起在网页的加载和渲染过程中实施的，所以对于它们的处理时间有严格要求。

198. js 原型链，原型链的顶端是什么？Object 的原型是什么？Object 的原型的原型是什么？在数组原型链上实现删除数组重复数据的方法

参考答案：

能够把这个讲清楚弄明白是一件很困难的事，

首先明白原型是什么，在 ES6 之前，JS 没有类和继承的概念，JS 是通过原型来实现继承的，在 JS 中一个构造函数默认带有一个 prototype 属性，这个的属性值是一个对象，同时这个 prototype 对象自带有一个 constructor 属性，这个属性指向这个构造函数，同时每一个实例都会有一个 _proto_ 属性指向这个 prototype 对象，我们可以把这个叫做隐式原型，我们在使用一个实例的方法的时候，会先检查这个实例中是否有这个方法，没有的话就会检查这个 prototype 对象是否有这个方法，

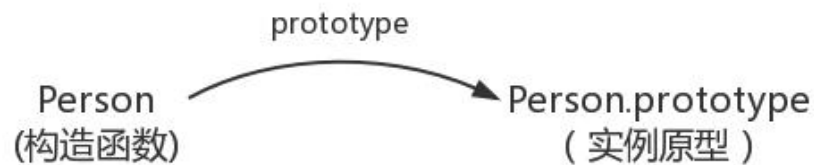
基于这个规则，如果让原型对象指向另一个类型的实例，即 `constructor1.prototype=instance2`，这时候如果试图引用 `constructor1` 构造的实例 `instance1` 的某个属性 `p1`，

首先会在 `instance1` 内部属性中找一遍，接着会在 `instance1._proto_` (`constructor1.prototype`) 即是 `instance2` 中寻找 `p1`

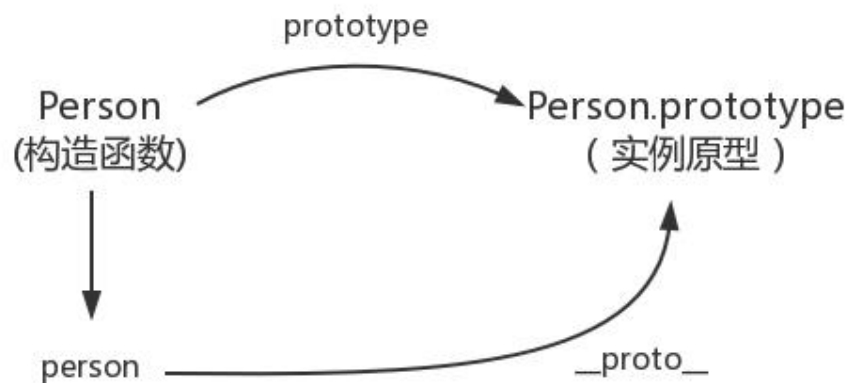
搜寻轨迹：`instance1->instance2->constructor2.prototype->Object.prototype`；这即是原型链，原型链顶端是 `Object.prototype`

补充学习：

每个函数都有一个 prototype 属性，这个属性指向了一个对象，这个对象正是调用该函数而创建的实例的原型，那么什么是原型呢，可以这样理解，每一个 JavaScript 对象在创建的时候就会预制管理另一个对象，这个对象就是我们所说的原型，每一个对象都会从原型继承属性，如图：



那么怎么表示实例与实例原型的关系呢，这时候就要用到第二个属性`_proto_`这是每一个 JS 对象都会有一个属性，指向这个对象的原型，如图：

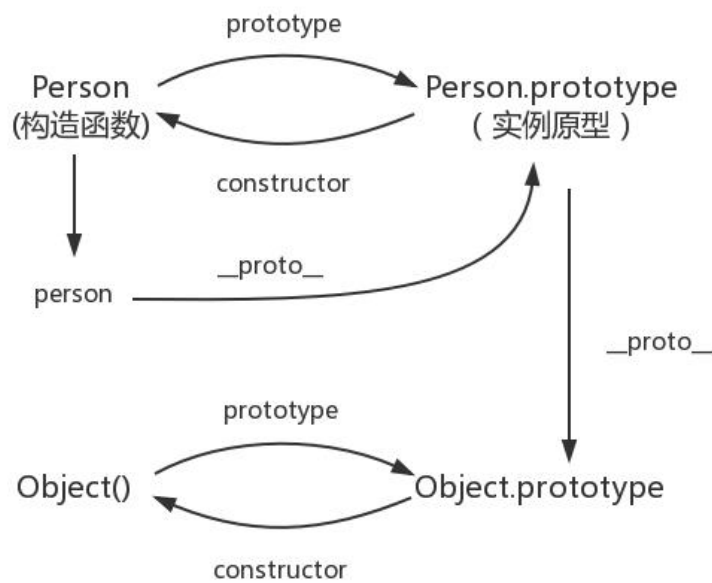


既然实例对象和构造函数都可以指向原型，那么原型是否有属性指向构造函数或者实例呢，指向实例是没有的，因为一个构造函数可以生成多个实例，但是原型有属性可以直接指向构造函数，通过 `constructor` 即可

接下来讲解实例和原型的关系：

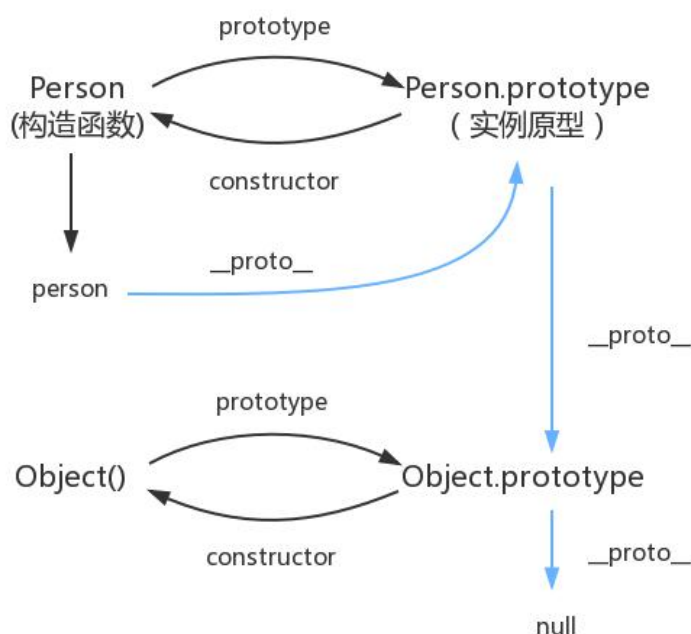
当读取实例的属性时，如果找不到，就会查找与对象相关的原型中的属性，如果还查不到，就去找原型的原型，一直找到最顶层，那么原型的原型是什么呢，首

先，原型也是一个对象，既然是对象，我们就可以通过构造函数的方式创建它，所以原型对象就是通过 Object 构造函数生成的，如图：



那么 `Object.prototype` 的原型呢，我们可以打印 `console.log(Object.prototype.__proto__ === null)`，返回 `true`。
`null` 表示没有对象，即该处不应有值，所以 `Object.prototype` 没有原型，如图：

图中这条蓝色的线即是原型链，



最后补充三点：

constructor：

```
function Person() {  
}
```

```
var person = new Person();
```

```
console.log(Person === person.constructor);
```

原本 person 中没有 constructor 属性，当不能读取到 constructor 属性时，会从 person 的原型中读取，所以指向构造函数 Person

__proto__：

绝大部分浏览器支持这个非标准的方法访问原型，然而它并不存在与 Person.prototype 中，实际上它来自 Object.prototype，当使用 obj.__proto__ 时，可以理解为返回来 Object.getPrototypeOf(obj)

继承：

前面说到，每个对象都会从原型继承属性，但是引用《你不知道的 JS》中的话，继承意味着复制操作，然而 JS 默认不会复制对象的属性，相反，JS 只是在两个对象之间创建一个关联，这样子一个对象就可以通过委托访问另一个对象的属性和函数，所以与其叫继承，叫委托更合适，

199. 什么是 js 的闭包？有什么作用，用闭包写个单例模式

参考答案：

MDN 对闭包的定义是：闭包是指那些能够访问自由变量的函数，自由变量是指在函数中使用的，但既不是函数参数又不是函数的局部变量的变量，由此可以看出，闭包=函数+函数能够访问的自由变量，所以从技术的角度讲，所有 JS 函数都是闭包，但是这是理论上的闭包，还有一个实践角度上的闭包，从实践角度上来说，只有满足 1) 即使创建它的上下文已经销毁，它仍然存在，2) 在代码中引入了自由变量，才称为闭包。

闭包的应用：

- 1) 模仿块级作用域
- 2) 保存外部函数的变量
- 3) 封装私有变量

单例模式：

```
var Singleton = (function() {  
  var instance;  
  var CreateSingleton = function (name) {  
    this.name = name;  
  }  
})
```

```
if(instance) {  
  return instance;  
}  
// 打印实例名字  
  
this.getName();  
// instance = this;  
// return instance;  
return instance = this;  
}  
// 获取实例的名字  
CreateSingleton.prototype.getName = function() {  
  console.log(this.name)  
}  
return CreateSingleton;  
})();  
// 创建实例对象 1  
var a = new Singleton('a');  
// 创建实例对象 2  
var b = new Singleton('b');  
console.log(a===b);
```

200. promise+Generator+Async 的使用

参考答案:

Promise

解决的问题:回调地狱

Promise 规范:

promise 有三种状态, 等待 (pending)、已完成 (fulfilled/resolved)、已拒绝 (rejected). Promise 的状态只能从“等待”转到“完成”或者“拒绝”, 不能逆向转换, 同时“完成”和“拒绝”也不能相互转换.

promise 必须提供一个 then 方法以访问其当前值、终值和据因。
promise.then(resolve, reject), resolve 和 reject 都是可选参数。如果 resolve 或 reject 不是函数, 其必须被忽略。
then 方法必须返回一个 promise 对象。

使用:

实例化 promise 对象需要传入函数(包含两个参数), resolve 和 reject, 内部确定状态。resolve 和 reject 函数可以传入参数在回调函数中使用。
resolve 和 reject 都是函数, 传入的参数在 then 的回调函数中接收。

```
var promise = new Promise(function(resolve, reject) {  
  setTimeout(function() {  
    resolve('好哈哈哈哈哈');  
  });  
});  
promise.then(function(val) {  
  console.log(val)  
})
```

then 接收两个函数, 分别对应 resolve 和 reject 状态的回调, 函数中接收实例化时传入的参数。

```
promise.then(val=>{  
  //resolved  
}, reason=>{  
  //rejected  
})
```

catch 相当于.then(null, rejection)

当 then 中没有传入 rejection 时, 错误会冒泡进入 catch 函数中, 若传入了

rejection, 则错误会被 rejection 捕获, 而且不会进入 catch. 此外, then 中的回调函数中发生的错误只会在下一级的 then 中被捕获, 不会影响该 promise 的状态.

```
new Promise((resolve, reject)=>{
  throw new Error(' 错误')
}).then(null, (err)=>{
  console.log(err, 1); //此处捕获
}).catch((err)=>{
  console.log(err, 2);
});
// 对比
new Promise((resolve, reject)=>{
  throw new Error(' 错误')
}).then(null, null).catch((err)=>{
  console.log(err, 2); //此处捕获
});

// 错误示例
new Promise((resolve, reject)=>{
  resolve(' 正常');
}).then((val)=>{
  throw new Error(' 回调函数中错误')
}, (err)=>{
  console.log(err, 1);
}).then(null, (err)=>{
  console.log(err, 2); //此处捕获, 也可用 catch
});
```

两者不等价的情况:

此时, catch 捕获的并不是 p1 的错误, 而是 p2 的错误,

```
p1().then(res=>{  
  return p2()//p2 返回一个 promise 对象  
}).catch(err=> console.log(err))
```

一个错误捕获的错误用例:

该函数调用中即使发生了错误依然会进入 then 中的 resolve 的回调函数, 因为函数 p1 中实例化 promise 对象时已经调用了 catch, 若发生错误会进入 catch 中, 此时会返回一个新的 promise, 因此即使发生错误依然会进入 p1 函数的 then 链中的 resolve 回调函数.

```
function p1(val) {  
  return new Promise((resolve, reject)=>{  
    if(val) {  
      var len = val.length;//传入 null 会发生错误, 进入 catch 捕获错误  
  
      resolve(len);  
    }else{  
      reject();  
    }  
  }).catch((err)=>{  
    console.log(err)  
  })  
};  
  
p1(null).then((len)=>{  
  console.log(len, 'resolved');  
}, ()=>{  
  console.log('rejected');  
}).catch((err)=>{  
  console.log(err, 'catch');  
})
```

Promise 回调链:

promise 能够在回调函数里面使用 return 和 throw, 所以在 then 中可以 return 出一个 promise 对象或其他值, 也可以 throw 出一个错误对象, 但如果没有 return, 将默认返回 undefined, 那么后面的 then 中的回调参数接收到的将是 undefined.

```
function p1(val) {
  return new Promise((resolve, reject) => {
    val == 1 ? resolve(1) : reject()
  })
};

function p2(val) {
  return new Promise((resolve, reject) => {
    val == 2 ? resolve(2) : reject();
  })
};

let promimse = new Promise(function(resolve, reject) {
  resolve(1)
})

. then(function(data1) {
  return p1(data1) // 如果去掉 return, 则返回 undefined 而不是 p1 的返回值, 会导致报错
})

. then(function(data2) {
  return p2(data2+1)
})

. then(res => console.log(res))
```

Generator 函数:

generator 函数使用:

1) 分段执行, 可以暂停

2) 可以控制阶段和每个阶段的返回值

3) 可以知道是否执行到结尾

```
function* g() {  
  var o = 1;  
  yield o++;  
  yield o++;  
}  
  
var gen = g();  
console.log(gen.next()); // Object {value: 1, done: false}  
var xxx = g();  
console.log(gen.next()); // Object {value: 2, done: false}  
console.log(xxx.next()); // Object {value: 1, done: false}  
console.log(gen.next()); // Object {value: undefined, done: true}
```

generator 和异步控制:

利用 Generator 函数的暂停执行的效果, 可以把异步操作写在 yield 语句里面, 等到调用 next 方法时再往后执行。这实际上等同于不需要写回调函数了, 因为异步操作的后续操作可以放在 yield 语句下面, 反正要等到调用 next 方法时再执行。所以, Generator 函数的一个重要实际意义就是用来处理异步操作, 改写回调函数。

async 和异步:

用法:

async 表示这是一个 async 函数, await 只能用在这个函数里面。

await 表示在这里等待异步操作返回结果, 再继续执行。

await 后一般是一个 promise 对象

示例:async 用于定义一个异步函数, 该函数返回一个 Promise。

如果 async 函数返回的是一个同步的值，这个值将被包装成一个理解 resolve 的 Promise，等同于 `return Promise.resolve(value)`。

`await` 用于一个异步操作之前，表示要“等待”这个异步操作的返回值。`await` 也可以用于一个同步的值。

```
let timer = async function timer() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('500');
    }, 500);
  });
}

timer().then(result => {
  console.log(result); //500
}).catch(err => {
  console.log(err.message);
});

//返回一个同步的值

let sayHi = async function sayHi() {
  let hi = await 'hello world';
  return hi; //等同于 return Promise.resolve(hi);
}

sayHi().then(result => {
  console.log(result);
});
```

201. 事件委托以及冒泡原理

参考答案：

事件委托是利用冒泡阶段的运行机制来实现的，就是把一个元素响应事件的函数委托到另一个元素，一般是把一组元素的事件委托到他的父元素上，委托的优点

是：

- 1) 减少内存消耗，节约效率
- 2) 动态绑定事件

事件冒泡，就是元素自身的事件被触发后，如果父元素有相同的事件，如 onclick 事件，那么元素本身的触发状态就会传递，也就是冒到父元素，父元素的相同事件也会一级一级根据嵌套关系向外触发，直到 document/window，冒泡过程结束。

202. 写个函数，可以转化下划线命名到驼峰命名

参考答案：

```
public static String UnderlineToHump(String para) {
    StringBuilder result=new StringBuilder();
    String a[]=para.split("_");
    for(String s:a) {
        if(result.length()==0) {
            result.append(s.toLowerCase());
        }else{
            result.append(s.substring(0, 1).toUpperCase());
            result.append(s.substring(1).toLowerCase());
        }
    }
    return result.toString();
}
```

203. 深浅拷贝的区别和实现

参考答案：

数组的浅拷贝：

如果是数组，我们可以利用数组的一些方法，比如 slice，concat 方法返回一个新数组的特性来实现拷贝，但假如数组嵌套了对象或者数组的话，使用 concat

方法克隆并不完整，如果数组元素是基本类型，就会拷贝一份，互不影响，而如果是对象或数组，就会只拷贝对象和数组的引用，这样我们无论在新旧数组进行了修改，两者都会发生变化，我们把这种复制引用的拷贝方法称为浅拷贝，

深拷贝就是指完全的拷贝一个对象，即使嵌套了对象，两者也互相分离，修改一个对象的属性，不会影响另一个

如何深拷贝一个数组：

1) 这里介绍一个技巧，不仅适用于数组还适用于对象！那就是：

```
var arr = ['old', 1, true, ['old1', 'old2'], {old: 1}]
var new_arr = JSON.parse( JSON.stringify(arr) );
console.log(new_arr);
```

原理是 JSON 对象中的 stringify 可以把一个 js 对象序列化为一个 JSON 字符串，parse 可以把 JSON 字符串反序列化为一个 js 对象，通过这两个方法，也可以实现对象的深复制。

但是这个方法不能够拷贝函数

浅拷贝的实现：

以上三个方法 concat, slice, JSON.stringify 都是技巧类，根据实际项目情况选择使用，我们可以思考下如何实现一个对象或数组的浅拷贝，遍历对象，然后把属性和属性值都放在一个新的对象里即可。

```
var shallowCopy = function(obj) {
// 只拷贝对象
if (typeof obj !== 'object') return;
// 根据 obj 的类型判断是新建一个数组还是对象
var newObj = obj instanceof Array ? [] : {};
// 遍历 obj，并且判断是 obj 的属性才拷贝
```

```
for (var key in obj) {  
  if (obj.hasOwnProperty(key)) {  
    newObj[key] = obj[key];  
  }  
}  
return newObj;  
}
```

深拷贝的实现：

那如何实现一个深拷贝呢？说起来也好简单，我们在拷贝的时候判断一下属性值的类型，如果是对象，我们递归调用深拷贝函数不就好了~

```
var deepCopy = function(obj) {  
  if (typeof obj !== 'object') return;  
  var newObj = obj instanceof Array ? [] : {};  
  for (var key in obj) {  
    if (obj.hasOwnProperty(key)) {  
      newObj[key] = typeof obj[key] === 'object' ? deepCopy(obj[key]) :  
        obj[key];  
    }  
  }  
  return newObj;  
}
```

204. JS 中 string 的 startwith 和 indexof 两种方法的区别

参考答案：

JS 中 startwith 函数，其参数有 3 个，stringObj, 要搜索的字符串对象，str, 搜索的字符串，position, 可选，从哪个位置开始搜索，如果以 position 开始的字符串以搜索字符串开头，则返回 true，否则返回 false

IndexOf 函数，indexOf 函数可返回某个指定字符串在字符串中首次出现的位置

205. js 字符串转数字的方法

参考答案：

通过函数 parseInt ()，可解析一个字符串，并返回一个整数，语法为 parseInt (string ,radix)

string: 被解析的字符串

radix: 表示要解析的数字的基数，默认是十进制，如果 radix<2 或>36, 则返回 NaN

206. let const var 的区别 ， 什么是块级作用域， 如何用 ES5 的方法实现块级作用域（立即执行函数）， ES6 呢

参考答案：

提起这三个最明显的区别是 var 声明的变量是全局或者整个函数块的，而 let, const 声明的变量是块级的变量，var 声明的变量存在变量提升，let, const 不存在，let 声明的变量允许重新赋值，const 不允许

207. ES6 箭头函数的特性

参考答案：

ES6 增加了箭头函数，基本语法为

```
let func = value => value;
```

相当于

```
let func = function (value) {  
  return value;  
}
```

```
};
```

箭头函数与普通函数的区别在于：

1) 箭头函数没有 this，所以需要通过查找作用域链来确定 this 的值，这就意味着如果箭头函数被非箭头函数包含，this 绑定的就是最近一层非箭头函数的 this，

2) 箭头函数没有自己的 arguments 对象，但是可以访问外围函数的 arguments 对象

3) 不能通过 new 关键字调用，同样也没有 new.target 值和原型

208. setTimeout 和 Promise 的执行顺序

参考答案：

首先我们来看这样一道题：

```
setTimeout(function() {  
  console.log(1)  
}, 0);  
  
new Promise(function(resolve, reject) {  
  console.log(2)  
  for (var i = 0; i < 10000; i++) {  
    if(i === 10) {console.log(10)}  
    i == 9999 && resolve();  
  }  
  console.log(3)  
}).then(function() {  
  console.log(4)  
})  
console.log(5)
```

输出答案为 2 10 3 5 4 1

要先弄清楚 `setTimeout (fun,0)` 何时执行，`promise` 何时执行，`then` 何时执行
`setTimeout` 这种异步操作的回调，只有主线程中没有执行任何同步代码的前提下，才会执行异步回调，而 `setTimeout (fun,0)` 表示立刻执行，也就是用来改变任务的执行顺序，要求浏览器尽可能快的进行回调；

`promise` 何时执行，由上图可知 `promise` 新建后立即执行，所以 `promise` 构造函数里代码同步执行的，

`then` 方法指向的回调将在当前脚本所有同步任务执行完成后执行，

那么 `then` 为什么比 `setTimeout` 执行的早呢，因为 `setTimeout (fun,0)` 不是真的立即执行，

经过测试得出结论：执行顺序为：同步执行的代码-》`promise.then->setTimeout`

209. 有了解过事件模型吗，DOM0 级和 DOM2 级有什么区别，DOM 的分级是什么

参考答案：

JSDOM 事件流存在如下三个阶段：

事件捕获阶段

处于目标阶段

事件冒泡阶段

JSDOM 标准事件流的触发的先后顺序为：先捕获再冒泡，点击 DOM 节点时，事件传播顺序：事件捕获阶段，从上往下传播，然后到达事件目标节点，最后是冒泡阶段，从下往上传播

DOM 节点添加事件监听方法 `addEventListener`，中参数 `capture` 可以指定该监听

是添加在事件捕获阶段还是事件冒泡阶段，为 false 是事件冒泡，为 true 是事件捕获，并非所有的事件都支持冒泡，比如 focus, blur 等等，我们可以通过 event.bubbles 来判断

事件模型有三个常用方法：

event.stopPropagation: 阻止捕获和冒泡阶段中，当前事件的进一步传播，

event.stopImmediatePropagation, 阻止调用相同事件的其他侦听器，

event.preventDefault, 取消该事件（假如事件是可取消的）而不停止事件的进一步传播

event.target: 指向触发事件的元素，在事件冒泡过程中这个值不变

event.currentTarget = this, 时间帮顶的当前元素，只有被点击时目标元素的 target 才会等于 currentTarget,

最后，对于执行顺序的问题，如果 DOM 节点同时绑定了两个事件监听函数，一个用于捕获，一个用于冒泡，那么两个事件的执行顺序真的是先捕获在冒泡吗，答案是否定的，绑定在被点击元素的事件是按照代码添加顺序执行的，其他函数是先捕获再冒泡

210. 平时是怎么调试 JS 的

参考答案：

一般用 Chrome 自带的控制台

211. JS 的基本数据类型有哪些，基本数据类型和引用数据类型的区别，NaN 是什么的缩写，JS 的作用域类型，
undefined==null 返回的结果是什么，undefined 与 null 的区别在哪，写一个函数判断变量类型

参考答案：

JS 的基本数据类型有字符串，数字，布尔，对象，Null，Undefined, 基本数据类型是按值访问的，也就是说我们可以操作保存在变量中的实际的值，
基本数据类型和引用数据类型的区别如下：

基本数据类型的值是不可变的，任何方法都无法改变一个基本类型的值，当这个变量重新赋值后看起来变量的值是改变了，但是这里变量名只是指向变量的一个指针，所以改变的是指针的指向改变，该变量是不变的，但是引用类型可以改变

基本数据类型不可以添加属性和方法，但是引用类型可以

基本数据类型的赋值是简单赋值，如果从一个变量向另一个变量赋值基本类型的值，会在变量对象上创建一个新值，然后把该值复制到为新变量分配的位置上，引用数据类型的赋值是对象引用，

基本数据类型的比较是值的比较，引用类型的比较是引用的比较，比较对象的内存地址是否相同

基本数据类型是存放在栈区的，引用数据类型是保存在栈区和堆区

NaN 是 JS 中的特殊值，表示非数字，NaN 不是数字，但是他的数据类型是数字，

它不等于任何值，包括自身，在布尔运算时被当做 false，NaN 与任何数运算得到的结果都是 NaN，党员算失败或者运算无法返回正确的数值的就会返回 NaN，一些数学函数的运算结果也会出现 NaN

JS 的作用域类型：

一般认为的作用域是词法作用域，此外 JS 还提供了一些动态改变作用域的方法

常见的作用域类型有：

- 1) 函数作用域，如果在函数内部我们给未定义的一个变量赋值，这个变量会变成成为一个全局变量，
- 2) 块作用域：块作用域吧标识符限制在 {} 中，

改变函数作用域的方法：

eval ()，这个方法接受一个字符串作为参数，并将其中的内容视为好像在书写时就存在于程序中这个位置的代码，

with 关键字：通常被当做重复引用同一个对象的多个属性的快捷方式

undefined 与 null：目前 null 和 undefined 基本是同义的，只有一些细微的差别，null 表示没有对象，undefined 表示缺少值，就是此处应该有一个值但是没有定义

```
undefined == null; //返回值是 true
```

```
undefined === null; //返回值是 false
```

此外了解== 和===的区别：

在做==比较时。不同类型的数据会先转换成一致后在做比较，===中如果类型不一致就直接返回 false，一致的才会比较

类型判断函数，使用 `typeof` 即可，首先判断是否为 `null`，之后用 `typeof` 哦按段，如果是 `object` 的话，再用 `array.isArray` 判断是否为数组，如果是数字的话用 `isNaN` 判断是否是 `NaN` 即可

扩展学习：

JS 采用的是词法作用域，也就是静态作用域，所以函数的作用域在函数定义的时候就决定了，

看如下例子：

```
var value = 1;
function foo() {
  console.log(value);
}
function bar() {
  var value = 2;
  foo();
}
bar();
```

假设 JavaScript 采用静态作用域，让我们分析下执行过程：

执行 `foo` 函数，先从 `foo` 函数内部查找是否有局部变量 `value`，如果没有，就根据书写的位置，查找上面一层的代码，也就是 `value` 等于 1，所以结果会打印 1。

假设 JavaScript 采用动态作用域，让我们分析下执行过程：

执行 `foo` 函数，依然是从 `foo` 函数内部查找是否有局部变量 `value`。如果没有，就从调用函数的作用域，也就是 `bar` 函数内部查找 `value` 变量，所以结果会打印 2。

前面我们已经说了，JavaScript 采用的是静态作用域，所以这个例子的结果是 1。

212. `setTimeout(fn, 100);` 100 毫秒是如何权衡的

参考答案：

`setTimeout()` 函数只是将事件插入了任务列表，必须等到当前代码执行完，主线程才会去执行它指定的回调函数，有可能要等很久，所以没有办法保证回调函数一定会在 `setTimeout` 指定的时间内执行，100 毫秒是插入队列的时间+等待的时间

213. JS 的垃圾回收机制

参考答案：

GC (garbage collection)，GC 执行时，中断代码，停止其他操作，遍历所有对象，对于不可访问的对象进行回收，在 V8 引擎中使用两种优化方法，分代回收，2、增量 GC，目的是通过对对象的使用频率，存在时长来区分新生代和老生代对象，多回收新生代区，少回收老生代区，减少每次遍历的时间，从而减少 GC 的耗时。

回收方法：

引用计次，当对象被引用的次数为零时进行回收，但是循环引用时，两个对象都至少被引用了一次，因此导致内存泄漏。

标记清除

214. 写一个 `newBind` 函数，完成 `bind` 的功能

参考答案：

bind() 方法，创建一个新函数，当这个新函数被调用时，bind() 的第一个参数将作为它运行时的 this，之后的一序列参数将会在传递的实参前传入作为它的参数

```
Function.prototype.bind2 = function (context) {  
  if (typeof this !== "function") {  
    throw new Error("Function.prototype.bind - what is trying to be bound is not callable");  
  }  
  var self = this;  
  var args = Array.prototype.slice.call(arguments, 1);  
  var fNOP = function () {};  
  var fbound = function () {  
    self.apply(this instanceof self ? this : context, args.concat(Array.prototype.slice.call(arguments)));  
  }  
  fNOP.prototype = this.prototype;  
  fbound.prototype = new fNOP();  
  return fbound;  
}
```

215. 怎么获得对象上的属性：比如说通过 Object.key ()

参考答案：

从 ES5 开始，有三种方法可以列出对象的属性

for (let I in obj) 该方法依次访问一个对象及其原型链中所有可枚举的类型

object.keys: 返回一个数组，包括所有可枚举的属性名称

object.getOwnPropertyNames: 返回一个数组包含不可枚举的属性。

216. 简单讲一讲 ES6 的一些新特性

参考答案：

ES6 在变量的声明和定义方面增加了 `let`、`const` 声明变量，有局部变量的概念，赋值中有比较吸引人的结构赋值，同时 ES6 对字符串、数组、正则、对象、函数等拓展了一些方法，如字符串方面的模板字符串、函数方面的默认参数、对象方面属性的简洁表达方式，ES6 也引入了新的数据类型 `symbol`，新的数据结构 `set` 和 `map`，`symbol` 可以通过 `typeof` 检测出来，为解决异步回调问题，引入了 `promise` 和 `generator`，还有最为吸引人了实现 `Class` 和模块，通过 `Class` 可以更好的面向对象编程，使用模块加载方便模块化编程，当然考虑到浏览器兼容性，我们在实际开发中需要使用 `babel` 进行编译

重要的特性：

块级作用域：ES5 只有全局作用域和函数作用域，块级作用域的好处是不再需要立即执行的函数表达式，循环体中的闭包不再有问题

`rest` 参数：用于获取函数的多余参数，这样就不需要使用 `arguments` 对象了，

`promise`：一种异步编程的解决方案，比传统的解决方案回调函数和事件更合理强大

模块化：其模块功能主要有两个命令构成，`export` 和 `import`，`export` 命令用于规定模块的对外接口，`import` 命令用于输入其他模块提供的功能

217. `call` 和 `apply` 是用来做什么？

参考答案：

`Call` 和 `apply` 的作用是一模一样的，只是传参的形式有区别而已

1) 改变 `this` 的指向

2) 借用别的对象的方法，

调用函数，因为 `apply`，`call` 方法会使函数立即执行

218. 了解事件代理吗，这样做有什么好处

参考答案：

事件代理/事件委托：利用了事件冒泡，只指定一个事件处理程序，就可以管理某一类型的事件，

简而言之：事件代理就是说我们将事件添加到本来要添加的事件的父节点，将事件委托给父节点来触发处理函数，这通常会使用在大量的同级元素需要添加同一类事件的时候，比如一个动态的非常多的列表，需要为每个列表项都添加点击事件，这时就可以使用事件代理，通过判断 `e.target.nodeName` 来判断发生的具体元素，这样做的好处是减少事件绑定，同事动态的 DOM 结构任然可以监听，事件代理发生在冒泡阶段

219. 给出以下代码，输出的结果是什么？原因？

```
for(var i=0;i<5;i++) {  
  setTimeout(function() {  
    console.log(i);  
  },1000);  
}  
console.log(i)
```

参考答案：

在一秒后输出 5 个 5

每次 for 循环的时候 `setTimeout` 都会执行，但是里面的 `function` 则不会执行被放入任务队列，因此放了 5 次；for 循环的 5 次执行完之后不到 1000 毫秒；1000 毫秒后全部执行任务队列中的函数，所以就是输出 5 个 5。

220. 给两个构造函数 A 和 B，如何实现 A 继承 B？

参考答案：

```
function A(...) {} A.prototype...
function B(...) {} B.prototype...
A.prototype = Object.create(B.prototype);
// 再在 A 的构造函数里 new B(props);
```

```
for(var i = 0; i < lis.length; i++) {
  lis[i].addEventListener('click', function(e) {
    alert(i);
  }, false)
}
```

221. 如果已经有三个 promise，A、B 和 C，想串行执行，该怎么写？

参考答案：

```
// promise
A.then(B).then(C).catch(...)
// async/await
(async ()=>{
  await a();
  await b();
  await c();
})();
```

222. 知道 private 和 public 吗

参考答案：

public: public 表明该数据成员、成员函数是对所有用户开放的，所有用户都可以直接进行调用

private: private 表示私有，私有的意思就是除了 class 自己之外，任何人都不可以直接使用

223. 基础的 js

参考答案:

```
Function.prototype.a = 1;
Object.prototype.b = 2;
function A() {}
var a = new A();
console.log(a.a, a.b); // undefined, 2
console.log(A.a, A.b); // 1, 2
```

224. async 和 await 具体该怎么用?

参考答案:

```
(async () => {
  await new Promise();
})();
```

225. 知道哪些 ES6, ES7 的语法

参考答案:

promise, await/async, let、const、块级作用域、箭头函数

226. promise 和 await/async 的关系

参考答案:

都是异步编程的解决方案

227. js 的数据类型

参考答案:

字符串, 数字, 布尔, 数组, null, Undefined, symbol, 对象。

228. js 加载过程阻塞, 解决方法

参考答案:

指定 script 标签的 async 属性。

如果 async="async", 脚本相对于页面的其余部分异步地执行 (当页面继续进行解析时, 脚本将被执行)

如果不使用 async 且 defer="defer": 脚本将在页面完成解析时执行

229. js 对象类型, 基本对象类型以及引用对象类型的区别

参考答案:

分为基本对象类型和引用对象类型

基本数据类型: 按值访问, 可操作保存在变量中的实际的值。基本类型值指的是简单的数据段。基本数据类型有这六种: undefined、null、string、number、boolean、symbol。

引用类型: 当复制保存着对象的某个变量时, 操作的是对象的引用, 但在为对象添加属性时, 操作的是实际的对象。引用类型值指那些可能为多个值构成的对象。引用类型有这几种: Object、Array、RegExp、Date、Function、特殊的基本包装类型 (String、Number、Boolean) 以及单体内置对象 (Global、Math)。

230. JavaScript 中的轮播实现原理？假如一个页面上有两个轮播，你会怎么实现？

参考答案：

图片轮播的原理就是图片排成一行，然后准备一个只有一张图片大小的容器，对这个容器设置超出部分隐藏，在控制定时器来让这些图片整体左移或右移，这样呈现出来的效果就是图片在轮播了。

如果有两个轮播，可封装一个轮播组件，供两处调用

231. 怎么实现一个计算一年中有多少周？

参考答案：

首先你得知道是不是闰年，也就是一年是 365 还是 366.

其次你得知道当年 1 月 1 号是周几。假如是周五，一年 365 天把 1 号 2 号 3 号减去，也就是把第一个不到一周的天数减去等于 362

还得知道最后一天是周几，加入是周五，需要把周一到周五减去，也就是 $362 - 5 = 357$. 正常情况 357 这个数计算出来是 7 的倍数。 $357 / 7 = 51$ 。即为周数。

232. 面向对象的继承方式

参考答案：

原型链继承：

核心： 将父类的实例作为子类的原型

特点：

- (1) 非常纯粹的继承关系，实例是子类的实例，也是父类的实例
- (2) 父类新增原型方法/原型属性，子类都能访问到

(3) 简单，易于实现

缺点：

- (1) 要想为子类新增属性和方法，不能放到构造器中
- (2) 无法实现多继承
- (3) 来自原型对象的所有属性被所有实例共享
- (4) 创建子类实例时，无法向父类构造函数传参

构造继承：

核心：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）

特点：

- (1) 解决了子类实例共享父类引用属性的问题
- (2) 创建子类实例时，可以向父类传递参数
- (3) 可以实现多继承（call 多个父类对象）

缺点：

- (1) 实例并不是父类的实例，只是子类的实例
- (2) 只能继承父类的实例属性和方法，不能继承原型属性/方法
- (3) 无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

实例继承：

核心：为父类实例添加新特性，作为子类实例返回

特点：

不限制调用方式，不管是 new 子类() 还是子类(), 返回的对象具有相同的效果

缺点：

- (1) 实例是父类的实例，不是子类的实例
- (2) 不支持多继承

拷贝继承

特点：

支持多继承

缺点：

效率较低，内存占用高（因为要拷贝父类的属性）

组合继承

核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后将父类实例作为子类原型，实现函数复用

特点：

- (1) 可以继承实例属性/方法，也可以继承原型属性/方法
- (2) 既是子类的实例，也是父类的实例
- (3) 不存在引用属性共享问题
- (4) 可传参
- (5) 函数可复用

寄生组合继承：

核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后将父类实例作为子类原型，实现函数复用

233. 引用类型常见的对象

参考答案:

Object、Array、RegExp、Date、Function、特殊的基本包装类型(String、Number、Boolean)以及单体内置对象(Global、Math)等

234. es6 的常用

参考答案:

promise, await/async, let、const、块级作用域、箭头函数

235. class

参考答案:

ES6 提供了更接近传统语言的写法, 引入了 Class (类) 这个概念, 作为对象的模板。通过 class 关键字, 可以定义类。

236. call 和 apply 的区别

参考答案:

apply: 调用一个对象的一个方法, 用另一个对象替换当前对象。例如: B.apply(A, arguments); 即 A 对象应用 B 对象的方法。

call: 调用一个对象的一个方法, 用另一个对象替换当前对象。例如: B.call(A, args1, args2); 即 A 对象调用 B 对象的方法。

237. es6 的常用特性

参考答案:

promise, await/async, let、const、块级作用域、箭头函数

238. 箭头函数和 function 有什么区别

参考答案:

箭头函数根本就没有绑定自己的 this，在箭头函数中调用 this 时，仅仅是简单的沿着作用域链向上寻找，找到最近的一个 this 拿来使用

239. new 操作符原理

参考答案:

- 1) 创建一个类的实例：创建一个空对象 obj，然后把这个空对象的__proto__设置为构造函数的 prototype。
- 2) 初始化实例：构造函数被传入参数并调用，关键字 this 被设定指向该实例 obj。
- 3) 返回实例 obj。

240. bind, apply, call

参考答案:

apply: 调用一个对象的一个方法，用另一个对象替换当前对象。例如: B.apply(A, arguments); 即 A 对象应用 B 对象的方法。

call: 调用一个对象的一个方法，用另一个对象替换当前对象。例如: B.call(A, args1, args2); 即 A 对象调用 B 对象的方法。

bind 除了返回是函数以外，它的参数和 call 一样。

241. bind 和 apply 的区别

参考答案:

返回不同: bind 返回是函数

参数不同: `apply(A, arguments)`, `bind(A, args1, args2)`

242. promise 实现

参考答案:

Promise 实现如下

```
function Promise(fn) {
  var state = 'pending',
      value = null,
      callbacks = [];
  this.then = function (onFulfilled, onRejected) {
    return new Promise(function (resolve, reject) {
      handle({
        onFulfilled: onFulfilled || null,
        onRejected: onRejected || null,
        resolve: resolve,
        reject: reject
      });
    });
  };
  function handle(callback) {
    if (state === 'pending') {
      callbacks.push(callback);
      return;
    }
    var cb = state === 'fulfilled' ? callback.onFulfilled :
      callback.onRejected,
      ret;
    if (cb === null) {
      cb = state === 'fulfilled' ? callback.resolve : callback.reject;
      cb(value);
    }
    return;
  }
}
```



```
}
ret = cb(value);
callback.resolve(ret);
}
function resolve(newValue) {
  if (newValue && (typeof newValue === 'object' || typeof newValue ===
'function')) {
    var then = newValue.then;
    if (typeof then === 'function') {
      then.call(newValue, resolve, reject);
    }
    return;
  }
  state = 'fulfilled';
  value = newValue;
  execute();
}
function reject(reason) {
  state = 'rejected';
  value = reason;
  execute();
}
function execute() {
  setTimeout(function () {
    callbacks.forEach(function (callback) {
      handle(callback);
    });
  }, 0);
}
fn(resolve, reject);
}
```

243. assign 的深拷贝

参考答案:

```
function clone( obj ) {  
  var copy;  
  switch( typeof obj ) {  
    case "undefined":  
      break;  
    case "number":  
      copy = obj - 0;  
      break;  
    case "string":  
      copy = obj + "";  
      break;  
    case "boolean":  
      copy = obj;  
      break;  
    case "object": //object 分为两种情况 对象 (Object) 和数组 (Array)  
  
      if(obj === null) {  
        copy = null;  
      } else {  
        if( Object.prototype.toString.call(obj).slice(8, -1) === "Array") {  
          copy = [];  
          for( var i = 0 ; i < obj.length ; i++ ) {  
            copy.push(clone(obj[i]));  
          }  
        } else {  
          copy = {};  
          for( var j in obj ) {  
            copy[j] = clone(obj[j]);  
          }  
        }  
      }  
      break;  
    default:  
      copy = obj;  
      break;  
  }  
}
```

```
}  
return copy;  
}
```

244. 说 promise, 没有 promise 怎么办

参考答案:

没有 promise, 可以用回调函数代替

245. arguments

参考答案:

arguments 是类数组对象, 有 length 属性, 不能调用数组方法

可用 Array.from() 转换

246. 箭头函数获取 arguments

参考答案:

可用...rest 参数获取

247. Promise

参考答案:

Promise 对象是 CommonJS 工作组提出的一种规范, 目的是为异步编程提供统一接口。每一个异步任务返回一个 Promise 对象, 该对象有一个 then 方法, 允许指定回调函数。

```
f1().then(f2);
```

一个 promise 可能有三种状态: 等待 (pending)、已完成 (resolved, 又称 fulfilled)、已拒绝 (rejected)。

promise 必须实现 then 方法（可以说，then 就是 promise 的核心），而且 then 必须返回一个 promise，同一个 promise 的 then 可以调用多次，并且回调的执行顺序跟它们被定义时的顺序一致。

then 方法接受两个参数，第一个参数是成功时的回调，在 promise 由“等待”态转换到“完成”态时调用，另一个是失败时的回调，在 promise 由“等待”态转换到“拒绝”态时调用。同时，then 可以接受另一个 promise 传入，也接受一个“类 then”的对象或方法，即 thenable 对象。

248. 事件代理

参考答案：

事件代理是利用事件的冒泡原理来实现的，何为事件冒泡呢？就是事件从最深的节点开始，然后逐步向上传播事件，举个例子：页面上有这么一个节点树，`div>ul>li>a`；比如给最里面的 a 加一个 click 点击事件，那么这个事件就会一层一层的往外执行，执行顺序 `a>li>ul>div`，有这样一个机制，那么我们给最外面的 div 加点击事件，那么里面的 ul, li, a 做点击事件的时候，都会冒泡到最外层的 div 上，所以都会触发，这就是事件代理，代理它们父级代为执行事件

249. Eventloop

参考答案：

任务队列中，在每一次事件循环中，macrotask 只会提取一个执行，而 microtask 会一直提取，直到 microtask 队列为空为止。

也就是说如果某个 microtask 任务被推入到执行中，那么当主线程任务执行完成后，会循环调用该队列任务中的下一个任务来执行，直到该任务队列到最后一个任务为止。而事件循环每次只会入栈一个 macrotask，主线程执行完成该任务后

又会检查 microtasks 队列并完成里面的所有任务后再执行 macrotask 的任务。

macrotasks: setTimeout, setInterval, setImmediate, I/O, UI rendering

microtasks: process.nextTick, Promise, MutationObserver

265.

第四章 前端核心

1. JSONP 的缺点

参考答案:

JSON 只支持 get, 因为 script 标签只能使用 get 请求;

JSONP 需要后端配合返回指定格式的数据。

2. 跨域 (jsonp, ajax)

参考答案:

JSONP: ajax 请求受同源策略影响, 不允许进行跨域请求, 而 script 标签 src 属性中的链接却可以访问跨域的 js 脚本, 利用这个特性, 服务端不再返回 JSON 格式的数据, 而是返回一段调用某个函数的 js 代码, 在 src 中进行了调用, 这样实现了跨域。

3. 如何实现跨域

参考答案:

JSONP: 通过动态创建 script, 再请求一个带参网址实现跨域通信。

document.domain + iframe 跨域：两个页面都通过 js 强制设置 document.domain 为基础主域，就实现了同域。

location.hash + iframe 跨域：a 欲与 b 跨域相互通信，通过中间页 c 来实现。三个页面，不同域之间利用 iframe 的 location.hash 传值，相同域之间直接 js 访问来通信。

window.name + iframe 跨域：通过 iframe 的 src 属性由外域转向本地域，跨域数据即由 iframe 的 window.name 从外域传递到本地域。

postMessage 跨域：可以跨域操作的 window 属性之一。

CORS：服务端设置 Access-Control-Allow-Origin 即可，前端无须设置，若要带 cookie 请求，前后端都需要设置。

代理跨域：起一个代理服务器，实现数据的转发

4. dom 是什么，你的理解？

参考答案：

文档对象模型（Document Object Model，简称 DOM），是 W3C 组织推荐的处理可扩展标志语言的标准编程接口。在网页上，组织页面（或文档）的对象被组织在一个树形结构中，用来表示文档中对象的标准模型就称为 DOM。

5. 关于 dom 的 api 有什么

参考答案:

节点创建型 api, 页面修改型 API, 节点查询型 API, 节点关系型 api, 元素属性型 api, 元素样式型 api 等

6. ajax 返回的状态

参考答案:

- 0 — (未初始化) 还没有调用 send() 方法
- 1 — (载入) 已调用 send() 方法, 正在发送请求
- 2 — (载入完成) send() 方法执行完成, 已经接收到全部响应内容
- 3 — (交互) 正在解析响应内容
- 4 — (完成) 响应内容解析完成, 可以在客户端调用了

7. 实现一个 Ajax

参考答案:

AJAX 创建异步对象 XMLHttpRequest

操作 XMLHttpRequest 对象

- (1) 设置请求参数 (请求方式, 请求页面的相对路径, 是否异步)
- (2) 设置回调函数, 一个处理服务器响应的函数, 使用 onreadystatechange , 类似函数指针
- (3) 获取异步对象的 readyState 属性: 该属性存有服务器响应的状态信息。每当 readyState 改变时, onreadystatechange 函数就会被执行。

(4) 判断响应报文的状态，若为 200 说明服务器正常运行并返回响应数据。

(5) 读取响应数据，可以通过 `responseText` 属性来取回由服务器返回的数据。

8. 如何实现 ajax 请求，假如我有多个请求，我需要通过这些 ajax 请求按照某种顺序一次执行，有什么办法呢？如何处理 ajax 跨域

参考答案：

通过实例化一个 `XMLHttpRequest` 对象得到一个实例，调用实例的 `open` 方法为这次 ajax 请求设定相应的 http 方法，相应的地址和是否异步，以异步为例，调用 `send` 方法，这个方法可以设定需要发送的报文主体，然后通过监听 `readystatechange` 事件，通过这个实例的 `readyState` 属性来判断这个 ajax 请求状态，其中分为 0, 1, 2, 3, 4 这四种状态（0 未初始化，1 载入/正在发送请求 2 载入完成/数据接收，3 交互/解析数据，4 接收数据完成），当状态为 4 的时候也就是接受数据完成的时候，这时候可以通过实例的 `status` 属性判断这个请求是否成功

```
var xhr = new XMLHttpRequest();
xhr.open('get', 'aabb.php', true);
xhr.send(null);
xhr.onreadystatechange = function() {
  if(xhr.readyState==4) {
    if(xhr.status==200) {
      console.log(xhr.responseText);
    }
  }
}
```

使 ajax 请求按照队列顺序执行，通过调用递归函数：

//按顺序执行多个 ajax 命令，因为数量不定，所以采用递归


```
function send(action, arg2) {
//将多个命令按顺序封装成数组对象，递归执行
//利用了 deferred 对象控制回调函数的特点
$.when(send_action(action[0], arg2))
.done(function () {
//前一个 ajax 回调函数完毕之后判断队列长度
if (action.length > 1) {
//队列长度大于 1，则弹出第一个，继续递归执行该队列
action.shift();
send(action, arg2);
}
}).fail(function () {
//队列中元素请求失败后的逻辑
//
//重试发送
//send(action, arg2);
//
//忽略错误进行下个
//if (action.length > 1) {
//队列长度大于 1，则弹出第一个，继续递归执行该队列
//    action.shift();
//    send(action, arg2);
//}
});
}
//处理每个命令的 ajax 请求以及回调函数
function send_action(command, arg2) {
var dtd = $.Deferred();//定义 deferred 对象

$.post(
"url",
{
```

```
command: command,
arg2: arg2
}
).done(function (json) {
  json = $.parseJSON(json);
  //每次请求回调函数的处理逻辑

  //
  //
  //

  //逻辑结束
  dtd.resolve();
}).fail(function () {
  //ajax 请求失败的逻辑
  dtd.reject();
});
return dtd.promise(); //返回 Deferred 对象的 promise，防止在外部修改状态
}
```

9. 如何实现一个 ajax 请求？如果我想发出两个有顺序的 ajax 需要怎么做？

参考答案：

AJAX 创建异步对象 XMLHttpRequest

操作 XMLHttpRequest 对象

(1) 设置请求参数（请求方式，请求页面的相对路径，是否异步）

(2) 设置回调函数，一个处理服务器响应的函数，使用 onreadystatechange，类似函数指针

- (3) 获取异步对象的 `readyState` 属性：该属性存有服务器响应的状态信息。每当 `readyState` 改变时，`onreadystatechange` 函数就会被执行。
 - (4) 判断响应报文的状态，若为 200 说明服务器正常运行并返回响应数据。
 - (5) 读取响应数据，可以通过 `responseText` 属性来取回由服务器返回的数据。
- 发出两个有顺序的 ajax，可以用回调函数，也可以使用 `Promise.then` 或者 `async` 等。

10. Fetch 和 Ajax 比有什么优缺点？

参考答案：

promise 方便异步，在不想用 jQuery 的情况下，相比原生的 ajax，也比较好写。

11. 移动应用和 web 应用的关系

参考答案：

原生 app 是基于手机操作系统，所以兼容性和接口方面特别丰富。而 webapp 是基于浏览器所以接口并不是很多，但是开发速率快，版本控制简单。成本低。

12. 知道 PWA 吗

参考答案：

PWA 全称 Progressive Web App，即渐进式 WEB 应用。一个 PWA 应用首先是一个网页，可以通过 Web 技术编写出一个网页应用。随后添加上 App Manifest 和 Service Worker 来实现 PWA 的安装和离线等功能

13. 做过移动端吗

参考答案：

根据自身实际情况回答

14. 知道 touch 事件吗

参考答案：

触摸事件，有 `touchstart` `touchmove` `touchend` `touchcancel` 四种之分，常用的有：

- (1) `touchstart`：当有新手指触控到绑定的元素，会触发一次事件。
- (2) `touchmove`：当有手指放绑定的元素上会一直触发，从触发条件准确的说只有手指移动时才触发。但是经过测试，这一项检测十分灵敏，人为手指保持不动，系统也会侦测到细小的移动。所以会一直触发。
- (3) `touchend`：当有手指从绑定元素上抬起，会触发一次。
- (4) `touchcancel`：可由系统进行的触发（不常用事件），比如手指触摸屏幕的时候，突然 `alert` 了一下，或者系统中其他打断了 `touch` 的行为，则可以触发该事件。

事件列表：

在移动端中上面的三个触摸事件每个事件都有以下列表：

- (1) `changedTouches`：保存了所有引发事件的手指信息
- (2) `targetTouches`：保存了当前对象上所有触摸点的列表；
- (3) `touches`：保存了当前所有触碰屏幕的手指信息

第五章 前端进阶

1. 前端测试

参考答案：

为什么要进行测试？

- (1) 保证代码正确性
- (2) 放心进行重构
- (3) 驱动开发 TDD
- (4) 实现自动化测试

测试驱动开发

它是一种测试先于编写代码的思想用于指导软件开发

在 TDD 中侧重点偏向开发，通过测试用例来规范约束开发者编写出质量更高、bug 更少的代码

行为驱动开发

行为驱动开发是一种敏捷软件开发的技术，它鼓励软件项目中的开发者、QA 和非技术人员或商业参与者之间的协作

BDD 更加侧重设计，其要求在设计测试用例的时候对系统进行定义，倡导使用通用的语言将系统的行为描述出来，将系统设计和测试用例结合起来，从而以此为驱动进行开发工作。

断言库

所谓“断言”，就是判断源码的实际执行结果与预期结果是否一致，如果不一致就抛出一个错误。

它是编写测试用例的关键。断言功能由断言库来实现，Mocha 本身不带断言库，所以必须先引入断言库。

前端测试流

1) 测试脚本

通常，测试脚本与所要测试的源码脚本同名，但是后缀名为 `.test.js`（表示测试）或者 `.spec.js`（表示规格）。

2) mocha

Mocha 测试用例主要包含下面几部分：

- (1) `describe` 定义的测试套件 (test suite)，表示一组相关的测试。
- (2) `it` 定义的测试用例 (test case)，表示一个单独的测试，是测试的最小单位。
- (3) 测试代码
- (4) 断言部分

3) Karma

可以监控一套文件的变换，并立即开始测试已保存的文件，用户无需离开文本编辑器。

如果要使用 `karma` 和 `mocha` 最好通过 `npm install karma-cli -g` 全局安装 `karma-cli`。

4) Travis.CI

提供的是持续集成服务 (Continuous Integration，简称 CI)。它绑定 Github 上面的项目，只要有新的代码，就会自动抓取。然后，提供一个运行环境，执行测试，完成构建，还能部署到服务器。

2. 接口文档

参考答案：

一、什么是接口文档？

在项目开发中，web 项目的前后端分离开发，APP 开发，需要由前后端工程师共

同定义接口，编写接口文档，之后大家都根据这个接口文档进行开发，到项目结束前都要一直维护。

二、为什么要写接口文档？

- 1) 项目开发过程中前后端工程师有一个统一的文件进行沟通交流开发
- 2) 项目维护中或者项目人员更迭，方便后期人员查看、维护

三、接口规范是什么？

首先接口分为四部分：方法、uri、请求参数、返回参数

- 1) **方法**: 新增(post) 修改(put) 删除(delete) 获取(get)
- 2) **uri**: 以/a 开头，如果需要登录才能调用的接口(如新增、修改；前台的用户个人信息，资金信息等)后面需要加/u，即：/a/u；中间一般放表名或者能表达这个接口的单词；get 方法，如果是后台通过搜索查询列表，那么以/search 结尾，如果是前台的查询列表，以/list 结尾；url 参数就不说了。
- 3) **请求参数和返回参数**，都分为 5 列：字段、说明、类型、备注、是否必填
字段是类的属性；说明是中文释义；类型是属性类型，只有 String、Number、Object、Array 四种类型；备注是一些解释，或者可以写一下例子，比如负责 json 结构的情况，最好写上例子，好让前端能更好理解；是否必填是字段的是否必填。
- 4) **返回参数结构有几种情况**：

(1) 如果只返回接口调用成功还是失败（如新增、删除、修改等），则只有一个结构体：code 和 message 两个参数；

(2) 如果要返回某些参数，则有两个结构体：1 是 code/message/data，2 是 data 里写返回的参数, data 是 object 类型；

(3) 如果要返回列表，那么有三个结构体，1 是 code/message/data, data 是 object，里面放置 page/size/total/totalPage/list 5 个参数，其中 list 是 Array 类型，list 里放 object，object 里是具体的参数。

3. webpack 和 gulp 区别（模块化与流的区别）

参考答案：

gulp 强调的是前端开发的工作流程，我们可以通过配置一系列的 task，定义 task 处理的事务（例如文件压缩合并、雪碧图、启动 server、版本控制等），然后定义执行顺序，来让 gulp 执行这些 task，从而构建项目的整个前端开发流程。webpack 是一个前端模块化方案，更侧重模块打包，我们可以把开发中的所有资源（图片、js 文件、css 文件等）都看成模块，通过 loader（加载器）和 plugins（插件）对资源进行处理，打包成符合生产环境部署的前端资源。

4. redux 用处

参考答案：

在组件化的应用中，会有着大量的组件层级关系，深嵌套的组件与浅层父组件进行数据交互，变得十分繁琐困难。而 redux，站在一个服务级别的角度，可以毫无阻碍地将应用的状态传递到每一个层级的组件中。redux 就相当于整个应用的管家。

6. redux 里常用方法

参考答案：

提供 `getState()` 方法获取 state；
提供 `dispatch(action)` 方法更新 state；
通过 `subscribe(listener)` 注册监听器；
等等

6. angularJs 和 react 区别

参考答案:

React 对比 Angular 是思想上的转变，它也并不是一个库，是一种开发理念，组件化，分治的管理，数据与 view 的一体化。它只有一个中心，发出状态，渲染 view，对于虚拟 dom 它并没有提高渲染页面的性能，它提供更多的是利用 jsx 便捷生成 dom 元素，利用组件概念进行分治管理页面每个部分(例如 header section footer slider)

7. vue 双向绑定原理

参考答案:

vue 数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的。利用了 `Object.defineProperty()` 这个方法重新定义了对象获取属性值(get)和设置属性值(set)。

8. 说说 vue react angularjs jquery 的区别

参考答案:

JQuery 与另外几者最大的区别是，JQuery 是事件驱动，其他两者是数据驱动。JQuery 业务逻辑和 UI 更改该混在一起，UI 里面还参杂这交互逻辑，让本来混乱的逻辑更加混乱。

Angular, vue 是双向绑定，而 React 不是
其他还有设计理念上的区别等

9. node 的事件方法讲讲看

参考答案:

```
emitter.addListener(eventName, listener) ,
emitter.emit(eventName[, ...args]), emitter.on(eventName, listener),
emitter.removeListener(eventName, listener)等
```

10. node 的特性，适合处理什么场景

参考答案:

Node.js 借助事件驱动，非阻塞 I/O 模型变得轻量和高效，非常适合运行在分布式设备的数据密集型实时应用。

11. 你有用到 Express, 讲讲 Express

参考答案:

Express 是一个简洁而灵活的 node.js Web 应用框架，提供了一系列强大特性帮助你创建各种 Web 应用，和丰富的 HTTP 工具。

12. promise 的状态有那些

参考答案:

等待(pending)、已完成(fulfilled)、已拒绝(rejected)

13. 数组移除第一个元素的方法有哪些？

参考答案：

splice 和 shift 等

第六章 移动端开发

1. 介绍一下 react

参考答案：

React 是一个用于构建用户界面的 JAVASCRIPT 库。React 主要用于构建 UI，很多人认为 React 是 MVC 中的 V（视图）

React 特点有：

声明式设计 - React 采用声明范式，可以轻松描述应用。

高效 - React 通过对 DOM 的模拟，最大限度地减少与 DOM 的交互。

灵活 - React 可以与已知的库或框架很好地配合。

JSX - JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX，但我们建议使用它。

组件 - 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。

单向响应的数据流 - 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. React 单项数据流

参考答案:

在 React 中，数据是单向流动的，是从上向下的方向，即从父组件到子组件的方向。

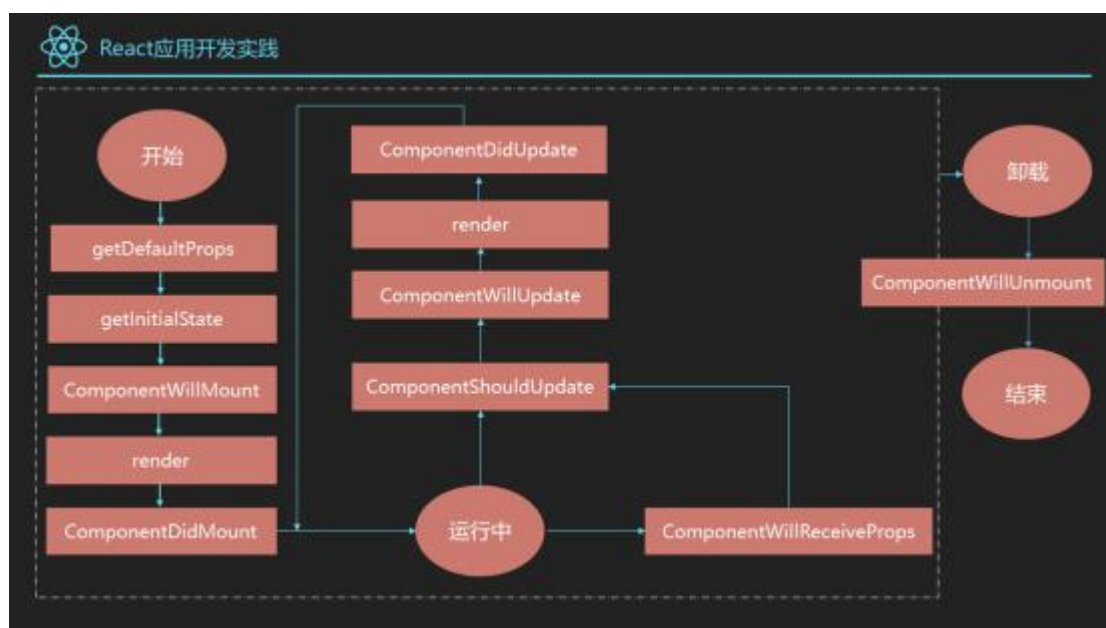
state 和 props 是其中重要的概念，如果顶层组件初始化 props，那么 React 会向下遍历整颗组件树，重新渲染相关的子组件。其中 state 表示的是每个组件中内部的状态，这些状态只在组件内部改变。

把组件看成是一个函数，那么他接受 props 作为参数，内部由 state 作为函数的内部参数，返回一个虚拟 dom 的实现。

3. react 生命周期函数和 react 组件的生命周期

参考答案:

React 的组件在第一次挂在的时候首先获取父组件传递的 props，接着获取初始的 state 值，接着经历挂载阶段的三个生命周期函数，也就是 ComponentWillMount, render, ComponentDidMount，这三个函数分别代表组件将会挂载，组件渲染，组件挂载完毕三个阶段，在组件挂载完成后，组件的 props 和 state 的任意改变都会导致组件进入更新状态，在组件更新阶段，如果是 props 改变，则进入 ComponentWillReceiveProps 函数，接着进入 ComponentShouldUpdate 进行判断是否需要更新，如果是 state 改变则直接进入 ComponentShouldUpdate 判定，这个默认是 true，当判定不需要更新的话，组件继续运行，需要更新的话则依次进入 ComponentWillMount, render, ComponentDidMount 三个函数，当组件卸载时，会首先进入生命周期函数 ComponentWillUnmount，之后才进行卸载，如图



React 的生命周期函数：

初始化阶段：

`getDefaultProps` 获取实例的默认属性，

`getInitialState` 获取每个实例的初始化状态，

`ComponentWillMount`：组件将被装载，渲染到页面上，`render`：组件在这里生成虚拟的 DOM 节点，

`ComponentDidMount`：组件真正被装载之后

运行中状态：

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

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

`shouldComponentUpdate` 这个方法用来判断是否需要调用 `render` 方法重新描绘 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。

`componentWillUpdate`: 组件即将更新不能修改属性和状态 `render`: 组件重新描绘

`componentDidUpdate`: 组件已经更新

销毁阶段: `componentWillUnmount`: 组件即将销毁

4. react 和 Vue 的原理，区别，亮点，作用

参考答案:

曾经看过 vue 作者尤雨溪的一个专访，他说过这样一段话(大概内容): 做框架的时候我们也很纠结，到底是定制内容少一点好还是定制内容多一点好。定制少了，很多人不知道一些情况应该怎么处理，所以他就乱来，写的代码乱七八糟，性能也不好，然后他就会认为你的框架没做好，有的人还去网上喷你。但是当大家经验越来越丰富，反而希望受到框架的限制越少越好。因为随着经验的增加，大家都知道了各种场景下应该怎么处理，优化自己的代码。限制越少，自我发挥的空间就越大。

最终我们可以看到，纠结之后，vue 的选择居于 react 与 angular 之间，框架自身的语法比 react 多一点，但是又比 angular 少一点。

也正是由于选择的不同，所呈现出来的写法与思考方式就一定会有所差异，不论优劣，但肯定会导致不同的偏好。

react 的简单在于，它的核心 API 其实非常少。所以我们会看到很多地方在说 react 其实是一个 UI 库，并不是一个完整的框架。他只是告诉我们如何创建组件以及组件之间如何进行数据传递。甚至于创建组件的方式正是使用 ES6 的 class 语法(`createClass` 将会在 react 16 被丢弃)。

因此开发中 react 的使用对于 ES6 的语法依赖非常高。因为 react 自身本来就没有多少强限制的语法。我们只需要掌握组件里的 props, state, ref, 生命周期, 就好像没有过多额外的知识了。就连如果想要在 jsx 模板来遍历渲染, 还得使用原生的 map 方法。而 react 的高阶组件, 理解之后发现, 其实就是 JavaScript 函数式编程中所涉及到的思维方式。

所以在我看来, react 的最大特点就是简单并且与原生 JavaScript 非常接近。即给开发者带来的束缚非常少。一个功能的实现, 如果你知道使用原生 JavaScript 如何实现, 那么你就一定能够很轻松的知道使用 react 如何实现。

当然, 核心 API 简单并不代表上手容易。在使用之初, 如果你经验缺乏, 那么你用 react 写出来的页面, 性能可能会非常差。因为无意识的, 你的组件可能会有非常多的多余的渲染。

比如很多人在学习 react 的时候, 会接触到一个倒计时的例子, 这个例子使用修改组件中 state 的方式来实现。但是其实后来大家会慢慢知道, 这种方式是非常错误的。因为 state 的每次修改, 都会导致组件及其所有子组件的重新渲染。这是成本非常高的行为。当然, 我还知道很多人, 在调试 react 的时候, 由于高频的重复渲染直接把浏览器都卡死的。这些问题都是尤雨溪所担心的限制过少带来的。

网上有的自以为牛逼的人, 用着 react/vue 这样的框架, 其实写着很烂的代码, 恐怖的是他们还嘲讽这嘲讽那的。还遇到过一个人, 口口声声说自己用了 angular 好多年, 说 angular 真的好垃圾啊, 性能好差啊, 什么什么的, 各种黑, 结果连 track by 都不会用。而 react 由于没有真正意义上的双向绑定。因此在处理一些复杂场景会非常麻烦, 比如复杂的表单验证。

而相对而言，vue 提供的能力则更多一点，这些便捷的能力会让初学者感到非常幸福，因为很多效果只需要一些简单的代码既可以实现。我大概列举几条我个人认为非常棒的能力：

统一管理的计算属性

JavaScript 的表达式非常便利，无论是 vue 还是 react，表达式的能力是必不可少的。但正如 vue 官方文档所说，在模板中放入太多的逻辑会让模板过重且难以维护。而 vue 的组件中提供了一个计算属性来统一管理表达式。

```
<template>
<div id="example">
<p>Original message: "{{ message }}"</p>
<p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
</template>
<script>
export default {
  name: 'example',
  data () {
    return {
      message: 'Hello'
    }
  },
  computed: {
    reversedMessage: function() {
```



```
return this.message.split('').reverse().join('')
}
}
}
</script>
```

class 的动态语法让我感觉非常爽

在实践中我们会发现非常多这样的场景，需要根据不同的状态来决定一个元素 class 的具体值。而如果仅仅只是简单的表达式或者条件判断在 jsx 模板中，例如下面这个样子就会让人感觉非常难受

```
<p className={active ? 'note active' : 'note'}></p>
```

当稍微复杂一点的逻辑还这样处理就是难受到忍不了了。而 vue 中支持的语法则非常轻松的搞定了这个问题。

```
// 可以放在任何你觉得舒服的位置
```

```
const pcls = {
  active: active,
  note: true
}
<p class={pcls}></p>
```

这样我们继续添加更多的 class 名也不会造成额外的复杂度了。

当然，这仅仅只是一个工具方法就能搞定的问题，在使用 react 时，大家可以借助 classnames 来完成同样的功能。但 vue 是直接支持了。

双向绑定

由于 react 并不支持双向绑定，因此在复杂的表单验证时实现起来非常痛苦。而 vue 在以单向数据流为核心的同时，又没有完全抛弃双向绑定，这让在这样复杂的表单验证场景开发效率比 react 高出非常多。这也是 vue 省事的一个方面。

修饰符

我们在写事件处理逻辑时，常常需要 e.preventDefault 等操作。vue 提供的修饰符功能可以帮助我们省去这些代码，极为方便。用多了就会发现，真 TM 好用。

```
<!-- 阻止单击事件冒泡 -->
<a v-on:click.stop="doThis"></a>
<!-- 提交事件不再重载页面 -->
1<form v-on:submit.prevent="onSubmit"></form>
<!-- 修饰符可以串联 -->
<a v-on:click.stop.prevent="doThat"></a>
<!-- 只有修饰符 -->
<form v-on:submit.prevent></form>
<!-- 添加事件侦听器时使用事件捕获模式 -->
<div v-on:click.capture="doThis">...</div>
<!-- 只当事件在该元素本身（而不是子元素）触发时触发回调 -->
<div v-on:click.self="doThat">...</div>
```

当然，还有按键修饰符等，可以去官网进一步查看学习。

vue 提供的方便可爱的语法糖还有很多，就不细说，大家可以在官网上一一体验。正如文章开头所说，vue 会有一些语法限制，而这些语法限制在某种程度上来说降低了我们的开发成本，提高了开发效率。这大概也就是很多人认为 vue 更加简

单易学的原因所在吧。

就从学习难易程度上来说，react 之所以上手更加困难，主要的原因并不在于 react 本身，而在于围绕 react 的丰富的生态圈。正是由于 react 本身足够简单，所以我们需要掌握的 react 组件就更多。比如 react-router，react-redux 等。而且很多好用的，功能特别棒的组件在我们涉猎不广的时候都不知道。

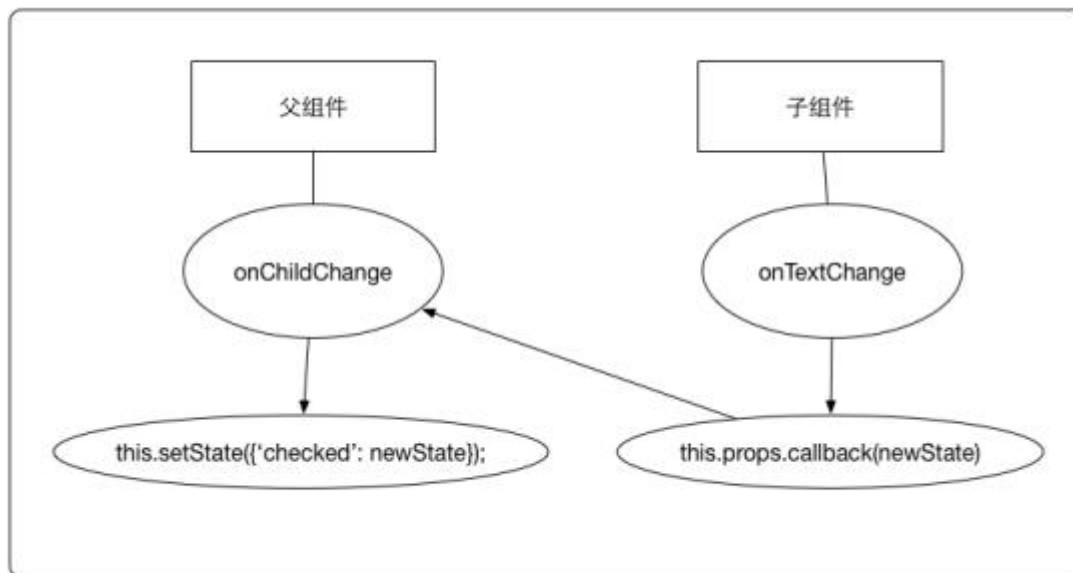
5. reactJs 的组件交流

参考答案：

React 组件之间的交流方式可以分为以下三种

(1) 父组件向子组件传值：主要是利用 props 来进行交流

(2) 子组件向父组件传值：子组件通过控制自己的 state 然后告诉父组件的点击状态。然后在父组件中展示出来，如图



(3) 没有任何嵌套关系的组件之间传值：如果组件之间没有任何关系，组件嵌套层次比较深（个人认为 2 层以上已经算深了），或者你为了一些组件能够订

阅、写入一些信号，不想让组件之间插入一个组件，让两个组件处于独立的关系。对于事件系统，这里有 2 个基本操作步骤：订阅 (subscribe) / 监听 (listen) 一个事件通知，并发送 (send) / 触发 (trigger) / 发布 (publish) / 发送 (dispatch) 一个事件通知那些想要的组件。

6. 有了解过 react 的虚拟 DOM 吗，虚拟 DOM 是怎么对比的呢

参考答案：

当然是使用的 diff 算法，diff 算法有三种优化形式：

- (1) tree diff：将新旧两颗 DOM 树按照层级遍历，只对同级的 DOM 节点进行比较，即同一父节点下的所有子节点，当发现节点已经不存在，则该节点及其子节点会被完全删除，不会进一步比较
- (2) component diff：不同组件之间的对比，如果组件类型相同，暂不更新，否则删除旧的组件，再创建一个新的组件，插入到删除组件的位置
- (3) element diff：在类型相同的组件内，再继续对比组件内部的元素

7. 项目里用到了 react，为什么要选择 react，react 有哪些好处

参考答案：

- (1) 声明式设计
- (2) 高效：通过对 DOM 的模拟，最大限度的减少与 DOM 的交互。
- (3) 灵活：可以与已知的框架或库很好的配合。
- (4) JSX：是 js 语法的扩展，不一定使用，但建议用。

(5) 组件：构建组件，使代码更容易得到复用，能够很好地应用在大项目的开发中。

(6) 单向响应的数据流：React 实现了单向响应的数据流，从而减少了重复代码，这也是解释了它为什么比传统数据绑定更简单。

8. 怎么获取真正的 dom

参考答案：

`ReactDOM.findDOMNode()` 或 `this.refs`

9. 选择 react 的原因

10. react 的生命周期函数

参考答案：

初始化

1) `getDefaultProps()`

设置默认的 props，也可以用 `defaultProps` 设置组件的默认属性。

2) `getInitialState()`

在使用 es6 的 class 语法时是没有这个钩子函数的，可以直接在 constructor 中定义 `this.state`。此时可以访问 `this.props`

3) `componentWillMount()`

组件初始化时只调用，以后组件更新不调用，整个生命周期只调用一次，此时可

以修改 state。

4) render()

react 最重要的步骤，创建虚拟 dom，进行 diff 算法，更新 dom 树都在此进行。

此时就不能更改 state 了。

5) componentDidMount()

组件渲染之后调用，只调用一次。

更新

6) componentWillReceiveProps(nextProps)

组件初始化时不调用，组件接受新的 props 时调用。

7) shouldComponentUpdate(nextProps, nextState)

react 性能优化非常重要的一环。组件接受新的 state 或者 props 时调用，我们可以设置在此对比前后两个 props 和 state 是否相同，如果相同则返回 false 阻止更新，因为相同的属性状态一定会生成相同的 dom 树，这样就不需要创造新的 dom 树和旧的 dom 树进行 diff 算法对比，节省大量性能，尤其是在 dom 结构复杂的时候

8) componentWillUpdate(nextProps, nextState)

组件初始化时不调用，只有在组件将要更新时才调用，此时可以修改 state

9) render()

组件渲染

10) componentDidUpdate()

组件初始化时不调用，组件更新完成后调用，此时可以获取 dom 节点。

卸载

11) componentWillUnmount()

组件将要卸载时调用，一些事件监听和定时器需要在此时清除。

11. setState 之后的流程

参考答案：

在代码中调用 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）。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。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。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。

12. react 高阶组件知道吗？

参考答案：

高阶组件接收 React 组件作为参数，并且返回一个新的 React 组件。高阶组件本质上也是一个函数，并不是一个组件。

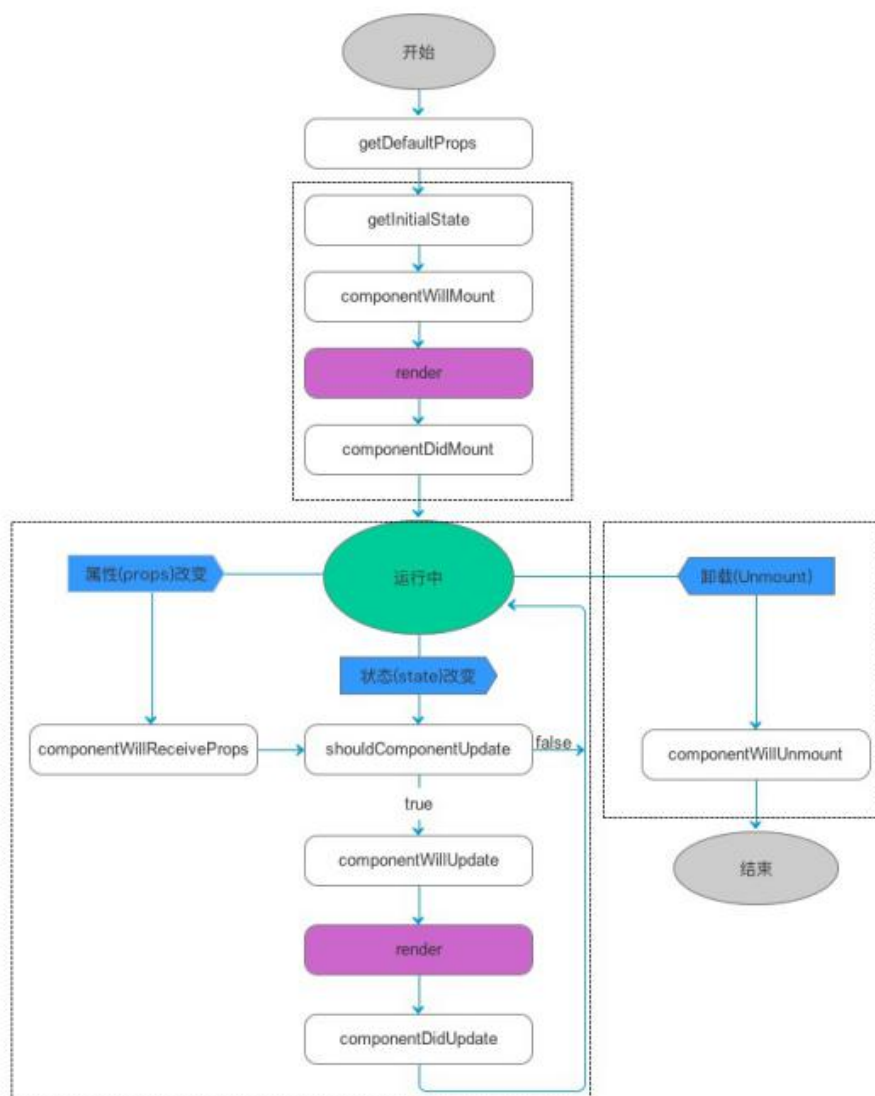
13. React 的生命周期

参考答案：

React 生命周期分为三种状态

1. 初始化
2. 更新
3. 销毁

具体见下图：



14. 说说自己理解的 react

参考答案：

React 是用于构建用户界面的 JavaScript 库。React 可以创建交互式 UI。为应用程序中的每个状态建立的视图，并且 React 将在数据更改时进行更新，呈现正确的组件。另外，我们也可以构建管理自己状态的封装组件，然后将它们组合成

复杂的 UI。因为组件用 JS 编写而不是模板，所以可以通过应用传递数据，并使状态与 DOM 分离

15. react 的组件是通过什么去判断是否刷新的

参考答案：

通过 state 是否改变

第七章 计算机基础

1. TCP 建立连接的三次握手过程

参考答案：

第一次握手：起初两端都处于 CLOSED 关闭状态，Client 将标志位 SYN 置为 1，随机产生一个值 $seq=x$ ，并将该数据包发送给 Server，Client 进入 SYN-SENT 状态，等待 Server 确认；

第二次握手：Server 收到数据包后由标志位 $SYN=1$ 得知 Client 请求建立连接，Server 将标志位 SYN 和 ACK 都置为 1， $ack=x+1$ ，随机产生一个值 $seq=y$ ，并将该数据包发送给 Client 以确认连接请求，Server 进入 SYN-RCVD 状态，此时操作系统为该 TCP 连接分配 TCP 缓存和变量；

第三次握手：Client 收到确认后，检查 ack 是否为 $x+1$ ，ACK 是否为 1，如果正确则将标志位 ACK 置为 1， $ack=y+1$ ，并且此时操作系统为该 TCP 连接分配 TCP 缓存和变量，并将该数据包发送给 Server，Server 检查 ack 是否为 $y+1$ ，ACK

是否为 1，如果正确则连接建立成功，Client 和 Server 进入 ESTABLISHED 状态，完成三次握手，随后 Client 和 Server 就可以开始传输数据。

2. cdn 原理

参考答案：

CDN 的全称是 Content Delivery Network，即内容分发网络。CDN 的基本原理是广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对集中的地区或网络中，在用户访问网站时，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，由缓存服务器直接响

3. HTTP 的头部包含哪些内容。常见的请求方法（我为什么要说后面的 options，head，connect）

参考答案：

常见的请求方法有 get, post，get 用来请求数据，post 用来提交数据，form 表单使用 get 时数据会以 querystring 形式存在 url 中，因而不安全也存在数据大小限制，而 post 不会，post 将数据存放在 http 报文体中，获取数据应该用 get，提交数据用 post

4. 请求方法 head 特性

参考答案：

Head 只请求页面的首部，head 方法和 get 方法相同，只不过服务器响应时不会

返回消息体，一个 head 请求的响应中，http 头中包含的元信息应该和一个 get 请求的响应消息相同，这种方法可以用来获取请求中隐含的元信息，而不用传输实体本身，这个也经常用来测试超链接的有效性和可用性，

Head 请求有以下特点：

只请求资源的首部，

检查超链接的有效性

检查网页是否被修改

用于自动搜索机器人获取网页的标志信息，获取 rss 种子信息，或者传递安全认证信息等

5. HTTP 状态码，301 和 302 有什么具体区别，200 和 304 的区别

参考答案：

状态码可以按照第一个数字分类：

1 表示信息，2 表示成功，3 表示重定向，4 表示客户端错误，5 表示服务器错误
常见的状态码有：

101 切换协议，200 成功，301 永久重定向，302 临时重定向，304 未修改

301 和 302 的区别：

301：永久移动，请求的网页已永久移动到新的位置，服务器返回此响应，会自动将请求者转到新位置；

302：历史移动，服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来继续以后的请求；

200 和 304:

200 表示成功, 服务器已成功处理了请求, 通常表示为服务器提供了请求的网页;

304 表示未修改, 自从上次请求后, 请求的网页未修改过, 服务器返回此响应时不会返回网页内容;

6. OSI 七层模型

参考答案:

osi 七层模型可以说是面试必考基础了

从上到下分别是:

应用层: 文件传输, 常用协议 HTTP, snmp, FTP ,

表示层: 数据格式化, 代码转换, 数据加密,

会话层: 建立, 解除会话

传输层: 提供端对端的接口, tcp, udp

网络层: 为数据包选择路由, IP, icmp

数据链路层: 传输有地址的帧

物理层: 二进制的形式在物理媒体上传输数据

7. TCP 和 UDP 的区别, 为什么三次握手四次挥手

参考答案:

TCP 和 UDP 之间的区别:

OSI 和 TCP/IP 模型在传输层定义两种传输协议: TCP (或传输控制协议) 和 UDP (或用户数据报协议)。UDP 与 TCP 的主要区别在于 UDP 不一定提供可靠的数

据传输。事实上，该协议不能保证数据准确无误地到达目的地。

为什么 TCP 要进行四次挥手呢？

因为是双方彼此都建立了连接，因此双方都要释放自己的连接，A 向 B 发出一个释放连接请求，他要释放链接表明不再向 B 发送数据了，此时 B 收到了 A 发送的释放链接请求之后，给 A 发送一个确认，A 不能再向 B 发送数据了，它处于 FIN-WAIT-2 的状态，但是此时 B 还可以向 A 进行数据的传送。此时 B 向 A 发送一个断开连接的请求，A 收到之后给 B 发送一个确认。此时 B 关闭连接。A 也关闭连接。

为什么要有 TIME-WAIT 这个状态呢，这是因为有可能最后一次确认丢失，如果 B 此时继续向 A 发送一个我要断开连接的请求等待 A 发送确认，但此时 A 已经关闭连接了，那么 B 永远也关不掉了，所以我们要 TIME-WAIT 这个状态。

当然 TCP 也并不是 100%可靠的。

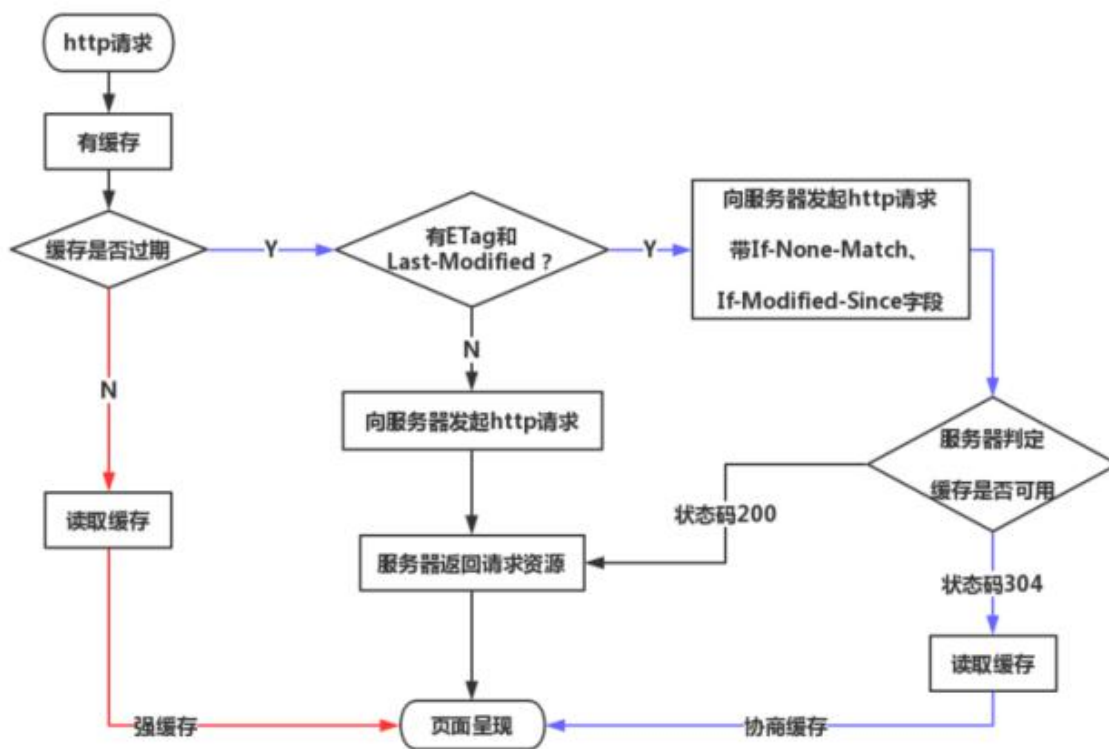
8. HTTP 缓存机制

参考答案：

HTTP 缓存即是浏览器第一次想一个服务器发起 HTTP 请求后，服务器会返回请求的资源，并且在响应头中添加一些有关缓存的字段如：cache-control, expires, last-modified, ETag, Date, 等，之后浏览器再向该服务器请求资源就可以视情况使用强缓存和协商缓存，

强缓存：浏览器直接从本地缓存中获取数据，不与服务器进行交互，

协商缓存：浏览器发送请求到服务器，服务器判断是否可使用本地缓存



9. websocket 和 ajax 的区别是什么，websocket 的应用场景有哪些

参考答案：

WebSocket 的诞生本质上就是为了解决 HTTP 协议本身的单向性问题：请求必须由客户端向服务端发起，然后服务端进行响应。这个 Request-Response 的关系是无法改变的。对于一般的网页浏览和访问当然没问题，一旦我们需要服务端主动向客户端发送消息时就麻烦了，因为此前的 TCP 连接已经释放，根本找不到客户端在哪。

为了能及时从服务器获取数据，程序员们煞费苦心研究出来的各种解决方案其实都是在 HTTP 框架下做的妥协，没法子，浏览器这东西只支持 HTTP，我们有什么

办法。所以大家要么定时去轮询，要么就靠长连接——客户端发起请求，服务端把这个连接攥在手里不回复，等有消息了再回，如果超时了客户端就再请求一次——其实大家也懂，这只是个减少了请求次数、实时性更好的轮询，本质没变。WebSocket 就是从技术根本上解决这个问题的：看名字就知道，它借用了 Web 的端口和消息头来创建连接，后续的数据传输又和基于 TCP 的 Socket 几乎完全一样，但封装了好多原本在 Socket 开发时需要我们手动去做的功能。比如原生支持 wss 安全访问（跟 https 共用端口和证书）、创建连接时的校验、从数据帧中自动拆分消息包等等。

换句话说，原本我们在浏览器里只能使用 HTTP 协议，现在有了 Socket，还是个更好用的 Socket。

了解了 WebSocket 的背景和特性之后，就可以回答它能不能取代 AJAX 这个问题了：

对于服务器与客户端的双向通信，WebSocket 简直是不二之选。如果不是还有少数旧版浏览器尚在服役的话，所有的轮询、长连接等方式早就该废弃掉。那些整合多种双向推送消息方式的库（如 <http://Socket.IO>、SignalR）当初最大的卖点就是兼容所有浏览器版本，自动识别旧版浏览器并采取不同的连接方式，现在也渐渐失去了优势——所有新版浏览器都兼容 WebSocket，直接用原生的就行了。

说句题外话，这点很像 jQuery，在原生 js 难用时迅速崛起，当其他库和原生 js 都吸收了它的很多优势时，慢慢就不那么重要了。

但是，很大一部分 AJAX 的使用场景仍然是传统的请求-响应形式，比如获取 json 数据、post 表单之类。这些功能虽然靠 WebSocket 也能实现，但就像在原本传输数据流的 TCP 之上定义了基于请求的 HTTP 协议一样，我们也要在 WebSocket 之上重新定义一种新的协议，最少也要加个 request id 用来区分每次响应数据

对应的请求吧。

……但是，何苦一层叠一层地造个新轮子呢？直接使用 AJAX 不是更简单、更成熟吗？

另外还有一种情况，也就是传输大文件、图片、媒体流的时候，最好还是老老实实用 HTTP 来传。如果一定要用 WebSocket 的话，至少也专门为这些数据专门开辟个新通道，而别去占用那条用于推送消息、对实时性要求很强的连接。否则会把串行的 WebSocket 彻底堵死的。

所以说，WebSocket 在用于双向传输、推送消息方面能够做到灵活、简便、高效，但在普通的 Request-Response 过程中并没有太大用武之地，比起普通的 HTTP 请求来反倒麻烦了许多，甚至更为低效。

每项技术都有自身的优缺点，在适合它的地方能发挥出最大长处，而看到它的几个优点就不分场合地全方位推广的话，可能会适得其反。

我们自己在开发能与手机通信的互联网机器人时就使用了 WebSocket，效果很好。但并不是用它取代 HTTP，而是取代了原先用于通信的基于 TCP 的 Socket。

优点是：

原先在 Socket 连接后还要进行一些复杂的身份验证，同时要阻止未验证的连接发送控制指令。现在不需要了，在建立 WebSocket 连接的 url 里就能携带身份验证参数，验证不通过可以直接拒绝，不用设置状态；

原先自己实现了一套类似 SSL 的非对称加密机制，现在完全不需要了，直接通过

wss 加密，还能顺便保证证书的可信性；

原先要自己定义 Socket 数据格式，设置长度与标志，处理粘包、分包等问题，现在 WebSocket 收到的直接就是完整的数据包，完全不用自己处理；

前端的 nginx 可以直接进行转发与负载均衡，部署简单多了

10. TCP/IP 的网络模型

参考答案：

TCP/IP 模型是一系列网络协议的总称，这些协议的目的是使得计算机之间可以进行信息交换，

TCP/IP 模型四层架构从下到上分别是链路层，网络层，传输层，应用层

链路层的作用是负责建立电路连接，是整个网络的物理基础，典型的协议包括以太网，ADSL 等，

网络层负责分配地址和传送二进制数据，主要协议是 IP 协议，

传输层负责传送文本数据，主要协议是 TCP

应用层负责传送各种最终形态的数据，是直接与用户信息打交道的层，主要协议是 http，ftp 等

11. 知道什么跨域方式吗，jsonp 具体流程是什么，如何实现原生 Jsonp 封装，优化，对于 CORS，服务器怎么判断它该不该跨域呢

参考答案：

常见的跨域方式大概有七种，大致可分为 iframe、api 跨域

1) JSONP，全称为 json with padding，解决老版本浏览器跨域数据访问问题，原理是 web 页面调用 JS 文件不受浏览器同源策略限制，所以通过 script 标签可以进行跨域请求，流程如下：

首先前端设置好回调参数，并将其作为 URL 的参数

服务器端收到请求后，通过该参数获取到回调函数名，并将数据放在参数中返回收到结果后因为是 script 标签，所以浏览器当做脚本运行，

2) cors，全称是跨域资源共享，允许浏览器向跨源服务器发出 XMLHttpRequest 请求，从而克服了 ajax 只能同源使用的策略，实现 cors 的关键是服务器，只要服务器实现了 cors 接口，就可以跨域通信

前端逻辑很简单，正常发起 ajax 请求即可，成功的关键在于服务器 Access-Control-Allow-Origin 是否包含请求页面的域名，如果不包含的话，浏览器将认为这是一次失败的异步请求，将会调用 xhr.onerror 中的函数。

Cors 使用简单，支持 POST 方式，但是存在兼容问题

浏览器将 cors 请求分为两类，简单请求和非简单请求，对于简单请求，浏览器直接发出 cors 请求，就是在头信息之中增加一个 origin 字段，用于说明本次请求来自哪个协议+域名+端口，服务器根据这个值，决定是否同意本次请求，如果服务器同意本次请求，返回的响应中会多出几个头信息字段：

Access-Control-Allow-Origin：返回 origin 的字段或者*

Access-Control-Allow-Credentials, 该字段可选，是一个 bool 值，表示是否允

许发送 cookie,

Access-Control-Expose-Headers

3) 服务器代理:

即当你有跨域的请求操作时发给后端, 让后端帮你代为请求,

此外还有四中不常用的方式, 也可了解下:

location.hash:

Window.name

postMessage

12. 怎么生成 token, 怎么传递

参考答案:

接口特点汇总:

- 1) 因为是非开放性的, 所以所有的接口都是封闭的, 只对公司内部的产品有效;
- 2) 因为是非开放性的, 所以 OAuth 那套协议是行不通的, 因为没有中间用户的授权过程;
- 3) 有点接口需要用户登录才能访问;
- 4) 有点接口不需要用户登录就可访问;

针对以上特点, 移动端与服务端的通信就需要 2 把钥匙, 即 2 个 token。

第一个 token 是针对接口的 (api_token);

第二个 token 是针对用户的 (user_token);

先说第一个 token (api_token)

它的职责是保持接口访问的隐蔽性和有效性, 保证接口只能给自家人用, 怎么做

到？参考思路如下：

现在的接口基本是 mvc 模式，URL 基本是 restful 风格，URL 大体格式如下：

http://blog.snsougou.com/模块名/控制器名/方法名?参数名 1=参数值 1&参数名 2=参数值 2&参数名 3=参数值 3

接口 token 生成规则参考如下：

api_token = md5 (' 模块名' + ' 控制器名' + ' 方法名' + '2017-07-18' + ' 加密密钥') = 770fed4ca2aabd20ae9a5dd774711de2

其中的

- 1) '2013-12-18' 为当天时间，
- 2) '加密密钥' 为私有的加密密钥，手机端需要在服务端注册一个“接口使用者”账号后，系统会分配一个账号及密码，数据表设计参考如下：

字段名	字段类型	注释
client_id	varchar(20)	客户端ID
client_secret	varchar(20)	客户端(加密)密钥

服务端接口校验，PHP 实现流程如下：

<?php

- 1) 获取 GET 参数值

```
$module = $_GET['mod'];  
$controller = $_GET['ctl'];  
$action = $_GET['act'];  
$client_id = $_GET['client_id'];  
$api_token = $_GET['api_token'];
```

2) 根据客户端传过来的 client_id, 查询数据库, 获取对应的 client_secret。

```
$client_secret = getClientSecretById($client_id);
```

3) 服务器重新生成一份 api_token

```
$api_token_server = md5($module.$controller.$action.date('Y-m-d',  
time()).$client_secret);
```

4) 客户端传过来的 api_token 与服务器生成的 api_token 进行校对, 如果不相等, 则表示验证失败。

```
if($api_token != $api_token_server){  
    exit('access deny');  
}
```

5) 验证通过, 返回数据到客户端。

再说第二个 token (user_token), 它的职责是保护用户的用户名及密码多次提交, 以防密码泄露。

如果接口需要用户登录, 其访问流程如下:

1) 用户提交“用户名”和“密码”, 实现登录(条件允许, 这一步最好走 https);

2) 登录成功后, 服务端返回一个 user_token, 生成规则参考如下:

服务端用数据表维护 user_token 的状态, 表设计如下:

| 字段名 | 字段类型 | 注释 |
|-------------|-------------|----------------|
| user_id | int | 用户ID |
| user_token | varchar(36) | 用户token |
| expire_time | int | 过期时间 (Unix时间戳) |

服务端生成 user_token 后，返回给客户端（自己存储），客户端每次接口请求时，如果接口需要用户登录才能访问，则需要把 user_id 与 user_token 传回给服务端，服务端接受到这 2 个参数后，需要做以下几步：

- 1) 检测 api_token 的有效性；
- 2) 删除过期的 user_token 表记录；
- 3) 根据 user_id, user_token 获取表记录，如果表记录不存在，直接返回错误，如果记录存在，则进行下一步；
- 4) 更新 user_token 的过期时间（延期，保证其有效期内连续操作不掉线）；
- 5) 返回接口数据。

那么 token 如何传递呢，ajax 中传递 token 有以下几种方式：

- (1) 放在请求头中：

```
headers: {  
  Accept: "application/json; charset=utf-8",  
  userToken: "" + userToken  
},
```

- (2) 使用 beforeSend 方法设置请求头

```
beforeSend: function(request) {  
  request.setRequestHeader("Authorization", token);  
},
```

13. 操作系统进程和线程的区别

参考答案：

进程，是并发执行的程序在执行过程中分配和管理资源的基本单位，是一个动态概念，竞争计算机系统资源的基本单位。

线程，是进程的一部分，一个没有线程的进程可以被看作是单线程的。线程有时又被称为轻权进程或轻量级进程，也是 CPU 调度的一个基本单位。

14. 什么是进程 线程

参考答案：

进程，是并发执行的程序在执行过程中分配和管理资源的基本单位，是一个动态概念，竞争计算机系统资源的基本单位。

线程，是进程的一部分，一个没有线程的进程可以被看作是单线程的。线程有时又被称为轻权进程或轻量级进程，也是 CPU 调度的一个基本单位。

15. 线程的那些资源共享，那些资源不共享

参考答案：

共享的资源有

- a. 堆由于堆是在进程空间中开辟出来的，所以它是理所当然地被共享的；因此 new 出来的都是共享的（16 位平台上分全局堆和局部堆，局部堆是独享的）
- b. 全局变量 它是与具体某一函数无关的，所以也与特定线程无关；因此也是共

享的

- c. 静态变量虽然对于局部变量来说，它在代码中是“放”在某一函数中的，但是其存放位置和全局变量一样，存于堆中开辟的.bss 和.data 段，是共享的
- d. 文件等公用资源 这个是共享的，使用这些公共资源的线程必须同步。Win32 提供了几种同步资源的方式，包括信号、临界区、事件和互斥体。

独享的资源有：

- a. 栈 栈是独享的
- b. 寄存器 这个可能会误解，因为电脑的寄存器是物理的，每个线程去取值难道不一样吗？其实线程里存放的是副本，包括程序计数器 PC

16. 操作系统里面进程和线程的区别

参考答案：

进程是具有一定独立功能的程序，他是系统进行资源分配调度的一个独立单位，线程是进程的一个实体，是 cpu 调度分派的基本单位，线程之间基本上不拥有系统资源

一个程序至少有一个进程，一个进程至少有一个线程，资源分配给进程，同一个进程下所有线程共享该进程的资源

17. Linux 查询进程指令，查询端口，杀进程

参考答案：

查询进程：

ps 命令用于查看当前正在运行的进程。

grep 是搜索

例如： `ps -ef | grep java`

表示查看所有进程里 CMD 是 java 的进程信息

`ps -aux | grep java`

-aux 显示所有状态

ps

杀死进程：

`kill -9[PID]`

18. 进程间的通信方式有哪些？

参考答案：

总共有八种，面试中只要能大概答上三四种方式的原理就可以了

- (1) 无名管道：半双工的通信方式，数据只能单向流动且只能在具有亲缘关系的进程间使用
- (2) 高级管道：将另一个程序当作一个新的进程在当前程序进程中启动，则这个进程算是当前程序的子进程，
- (3) 有名管道，：也是半双工的通信方式，但是允许没有亲缘进程之间的通信
- (4) 消息队列：消息队列是有消息的链表，存放在内核中，并由消息队列标识符标识，消息队列克服了信号传递信息少，管道只能承载无格式字节流以及缓冲区大小受限的缺点
- (5) 信号量：信号量是一个计数器，可以用来控制多个进程对共享资源的访问，它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该

资源，

- (6) 信号：用于通知接受进程某个事件已经发生
- (7) 共享内存：共享内存就是映射一段能被其他进程所访问的内存。这段共享内存由一个进程创建，但是多个进程可以访问，共享内存是最快的 IPC 方式，往往与其他通信机制配合使用
- (8) 套接字：可用于不同机器之间的进程通信

19. Redis 和 mysql

参考答案：

(1) 类型上

从类型上来说，mysql 是关系型数据库，redis 是缓存数据库

(2) 作用上

mysql 用于持久化的存储数据到硬盘，功能强大，但是速度较慢

redis 用于存储使用较为频繁的数据到缓存中，读取速度快

(3) 需求上

mysql 和 redis 因为需求的不同，一般都是配合使用。

第八章 算法与数据结构

1. 二叉树层序遍历

参考答案:

思路: 先建立一棵二叉树。再进行队列遍历

```
function tree(obj) {
  var obj = obj.split('');
  obj.pop();
  var newobj = [];
  for (var i = 0; i < obj.length; i++) {
    newobj.push(obj[i].replace(' ',''));
  }
  var root = {
    value: null, left: null, right: null, have: 0
  }
  var u;
  for(var i = 0; i < newobj.length; i++) {
    var a1 = newobj[i].split(',')[0];
    var a2 = newobj[i].split(',')[1];
    u = root;
    if(a2!==''){
      for (var j = 0; j < a2.length; j++) {
        if(a2[j]=== 'L') {
          if(u.left === null){
            u.left = newnode();
            u = u.left;
          }else {
            u = u.left;
          }
        } else if(a2[j]=== 'R') {
          if(u.right === null){
            u.right = newnode();
            u = u.right;
          }else{
            u = u.right;
          }
        }
      }
    }
  }
}
```

```
}
}
if(u.have === 1) {
} else{
u.value = a1;
u.have = 1;
}
}else {
root.value = a1;
u.have = 1;
}
}
return root;
}
//建立新结点

function newnode() {
return {value: null, left: null, right: null,have:0};
}
//队列遍历

function bfs() {
var root =
tree(' (11,LL) (7,LLL) (8,R) (5,) (4,L) (13,RL) (2,LLR) (1,RRR) (4,RR)' );
var front = 0, rear = 1, n=0;
var q = [], ans=[];
q[0] = root;
while(front < rear) {
var u = q[front++];
if(u.have!==1) {
return;
}
ans[n++] = u.value;
if(u.left!==null) {
q[rear++] = u.left;
}
if(u.right!==null) {
q[rear++] = u.right;
}
}
}
```

```
console.log(ans.join(' '));  
}  
bfs();
```

2. B 树的特性，B 树和 B+树的区别

参考答案：

一个 m 阶的 B 树满足以下条件：

每个结点至多拥有 m 棵子树；

根结点至少拥有两颗子树（存在子树的情况下）；

除了根结点以外，其余每个分支结点至少拥有 $m/2$ 棵子树；

所有的叶结点都在同一层上；

有 k 棵子树的分支结点则存在 $k-1$ 个关键码，关键码按照递增次序进行排列；

关键字数量需要满足 $\text{ceil}(m/2)-1 \leq n \leq m-1$ ；

B 树和 B+树的区别：

以一个 m 阶树为例。

关键字的数量不同；B+树中分支结点有 m 个关键字，其叶子结点也有 m 个，其关键字只是起到了一个索引的作用，但是 B 树虽然也有 m 个子结点，但是其只拥有 $m-1$ 个关键字。

存储的位置不同；B+树中的数据都存储在叶子结点上，也就是其所有叶子结点的数据组合起来就是完整的数据，但是 B 树的数据存储在每一个结点中，而不仅仅存储在叶子结点上。

分支结点的构造不同；B+树的分支结点仅仅存储着关键字信息和儿子的指针（这里的指针指的是磁盘块的偏移量），也就是说内部结点仅仅包含着索引信息。

查询不同；B 树在找到具体的数值以后，则结束，而 B+树则需要通过索引找到叶子结点中的数据才结束，也就是说 B+树的搜索过程中走了一条从根结点到叶子

结点的路径。

3. 尾递归

参考答案：

如果一个函数中所有递归形式的调用都出现在函数的末尾，我们称这个递归函数是尾递归的。当递归调用是整个函数体中最后执行的语句且它的返回值不属于表达式的一部分时，这个递归调用就是尾递归。

4. 如何写一个大数阶乘？递归的方法会出现什么问题？

参考答案：

```
function factorial(n) {  
  return n > 1 ? n * factorial(n-1) : 1;  
}
```

递归方法会有计算溢出的问题

5. 把多维数组变成一维数组的方法

参考答案：

法一：递归

```
function flatten(arr) {  
  var result = [];  
  for (var i = 0, len = arr.length; i < len; i++) {  
    if (Array.isArray(arr[i])) {  
      result = result.concat(flatten(arr[i]))  
    }  
  }  
}
```

```
else {  
  result.push(arr[i])  
}  
}  
return result;  
}
```

法二：toString

```
function flatten(arr) {  
  return arr.toString().split(',').map(function(item) {  
    return +item  
  })  
}
```

法三：reduce

```
function flatten(arr) {  
  return arr.reduce(function(prev, next) {  
    return prev.concat(Array.isArray(next) ? flatten(next) : next)  
  }, [])  
}
```

法四：rest 运算符

```
function flatten(arr) {  
  while (arr.some(item => Array.isArray(item))) {  
    arr = [].concat(...arr);  
  }  
  return arr;  
}
```

6. 知道的排序算法 说一下冒泡快排的原理

参考答案：

冒泡排序：重复地走访过要排序的元素列，依次比较两个相邻的元素，如果他们的顺序（如从大到小、首字母从 A 到 Z）错误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素已经排序完成。

快速排序：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

7. Heap 排序方法的原理？复杂度？

参考答案：

堆排序（英语：Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

复杂度： $O(n \lg n)$

8. 几种常见的排序算法，手写

参考答案：

基本排序算法：冒泡，选择，插入，希尔，归并，快排

冒泡排序：

```
function bubbleSort(data) {  
  var temp=0;  
  for(var i=data.length;i>0;i--){
```



```
for(var j=0;j<i-1;j++){
  if(data[j]>data[j+1])
  {
    temp=data[j];
    data[j]=data[j+1];
    data[j+1]=temp;
  }
}
return data;
}
```

选择排序:

```
function selectionSort(data) {
  for(var i=0;i<data.length;i++){
    var min=data[i];
    var temp;
    var index=1;
    for(var j=i+1;j<data.length;j++){
      if(data[j]<min)
      {
        temp=data[j];
        data[j]=min;
        min=temp;
      }
    }
    temp=data[i];
    data[i]=min;
    data[index]=temp
  }
}
```

插入排序:

```
function insertSort(data) {  
  var len=data.length;  
  for(var i=0;i<len;i++){  
    var key=data[i];  
    var j=i-1;  
    while(j>=0&&data[j]>key){  
      data[j+1]=data[j];  
      j--;  
    }  
    data[j+1]=key;  
  }  
  return data;  
}
```

希尔排序:

```
function shallSort(array) {  
  var increment = array.length;  
  var i  
  var temp; //暂存  
  
  do {  
  
    //设置增量  
  
    increment = Math.floor(increment / 3) + 1;  
    for (i = increment ; i < array.length; i++) {  
      if ( array[i] < array[i - increment]) {  
        temp = array[i];  
        for (var j = i - increment; j >= 0 && temp < array[j]; j -= increment)  
        {  
          array[j + increment] = array[j];  
        }  
        array[j + increment] = temp;  
      }  
    }  
  } while (increment > 1);  
}
```

```
}  
array[j + increment] = temp;  
}  
}  
}  
while (increment > 1)  
return array;  
}
```

归并排序:

```
function mergeSort ( array ) {  
  var len = array.length;  
  if( len < 2 ){  
    return array;  
  }  
  var middle = Math.floor(len / 2),  
      left = array.slice(0, middle),  
      right = array.slice(middle);  
  return merge(mergeSort(left), mergeSort(right));  
}  
function merge(left, right)  
{  
  var result = [];  
  while (left.length && right.length) {  
    if (left[0] <= right[0]) {  
      result.push(left.shift());  
    } else {  
      result.push(right.shift());  
    }  
  }  
  while (left.length)  
    result.push(left.shift());  
}
```

```
while (right.length)
result.push(right.shift());
return result;
}
```

快速排序:

```
function quickSort(arr) {
if(arr.length==0)
return [];
var left=[];
var right=[];
var pivot=arr[0];
for(var i=0;i<arr.length;i++) {
if(arr[i]<pivot) {
left.push(arr[i]);
}
else{
right.push(arr[i]);
}
}
return quickSort(left).concat(pivot,quickSort(right));
}
```

9. 数组的去重，尽可能写出多个方法

参考答案:

1) 首先介绍最简单的双层循环方法:

```
var array = ['1','2',1,'1','4','9','1'];
function unique(array) {
var res=[];
for(var i=0,arraylen=array.length;i<array.length;i++) {
```

```
for(var j=0,reslen=array.length;j<array.length;j++){
  if(array[i]==res[j])
    break;
}
if(j===reslen)
{
  res.push(array[i])
}
}
return res;
}
```

```
console.log(unique(array));
```

2) 用 indexOf 简化内层循环: indexOf 函数返回某个指定的字符在字符串中第一次出现的位置

```
var array = ['1','2',1,'1','4','9','1'];
function unique(array){
  var res=[];
  for(var i=0,len=array.length;i<len;i++){
    var current=array[i];
    if(res.indexOf(current)===-1)
    {
      res.push(current);
    }
  }
  return res;
}
console.log(unique(array));
```

排序后去重

```
var array = ['1','2',1,'1','4','9','1'];
function unique(array) {
```

// res 用来存储结果

```
var res=[];
var sortArray = array.concat().sort();
console.log(sortArray);
var seen;
for(var i=0,len=sortArray.length;i<len;i++){
  if(!i||seen!==sortArray[i]){
    res.push(sortArray[i]);
  }
  seen=sortArray[i];
}
return res;
}
console.log(unique(array)); //
```

ES6 的方法，使用 set 和 map 数据结构，以 set 为例，它类似于数组，但是成员的值都是唯一的，没有重复的值，很适合这个题目

```
var array = ['1','2',1,'1','4','4','1'];
function unique(array) {
  // res 用来存储结果
```

```
  return Array.from(new Set(array));
}
```

```
console.log(unique(array));
```

或者更简化点

```
var array = ['1','2',1,'1','4','4','1'];
function unique(array) {
  // res 用来存储结果
```

```
  return [...new Set(array)];
}
```

```
console.log(unique(array));
```

10. 如果有一个大的数组，都是整型，怎么找出最大的前 10 个数

参考答案：

排序数组，输出前 10 个

11. 知道数据结构里面的常见的数据结构

参考答案：

常见的数据结构有链表，栈，队列，树，更深一点的就还有图，但是考的不怎么多

12. 找出数组中第 k 大的数组出现多少次，比如数组【1， 2， 4， 4， 3， 5】第二大的数字是 4，出现两次，所以返回 2

参考答案：

对数组进行排序，找到第 k 大的数，然后看第 k 大的数有几个，返回

13. 合并两个有序数组

参考答案：

即是采用归并排序即可

14. 给一个数，去一个已经排好序的数组中寻找这个数的位置（通过快速查找，二分查找）

参考答案：

```
function binarySearch(target, arr, start, end) {  
  var start = start || 0;  
  var end = end || arr.length-1;  
  var mid = parseInt((start+(end-start)/2));  
  if(target==arr[mid]) {  
    return mid;  
  }else if(target>arr[mid]) {  
    return binarySearch(target, arr, mid+1, end);  
  }else {  
    return binarySearch(target, arr, start, mid-1);  
  }  
  return -1;  
}
```

第九章 设计模式

1. 设计模式：单例，工厂，发布订阅

参考答案：

单例模式：在它的核心结构中值包含一个被称为单例的特殊类。一个类只有一个实例，即一个类只有一个对象实例。

工厂模式：在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。

发布订阅模式：在软件架构中，发布订阅是一种消息范式，消息的发送者（称为发布者）不会将消息直接发送给特定的接收者（称为订阅者）。而是将发布的信息分为不同的类别，无需了解哪些订阅者（如果有的话）可能存在。同样的，订阅者可以表达对一个或多个类别的兴趣，只接收感兴趣的消息，无需了解哪些发布者（如果有的话）存在。

2. 看过一些设计模式的书？你觉得设计模式怎么样？

参考答案：

JS 中常用的设计模式中，我最常用的是装饰者模式，在不改变元对象的基础上，对这个对象进行包装和拓展（包括添加属性和方法），从而使这个对象可以有更复杂的功能。

第十章 项目

1. 介绍一个做过的项目

参考回答：

- (1) 项目涉及的技术栈和相关的技术点。
- (2) 遇到的问题。
- (3) 项目是否做过优化，包括了代码、开发效率、性能、体验等相关领域。
- (4) 项目存在的一些问题。
- (5) 这个项目达成的成果。

(6) 这个项目给我带来的成长。

2. 遇到的难题，怎么解决

参考回答：

- (1) 难点分析, 任务拆分, 关联自己已有知识储备, 能否解决.
- (2) 搜索引擎查找相关博客/问答平台是否遇到并给出的解答方案.
- (3) 咨询自己的技术上级/组内技术高手, 看能否帮忙解答
- (4) 在公司内部小组群/大群, 提出自己的难点, 看是否有人遇到.
- (5) 借助 QQ 群/微信群/问答平台, 提出相关问题来探讨.

3. 简单的自我介绍

参考回答：

首先请报出自己的姓名和身份。可能应试者与面试考官打招呼时，已经将此告诉了对方，而且考官们完全可以从你的报名表、简历等材料中了解这些情况，但仍请你主动提及。这是礼貌的需要，还可以加深考官对你的印象。

其次，你可以简单地介绍一下你的学历、工作经历等基本个人情况。请提供给考官关于你个人情况的基本的、完整的信息，如：学历、工作经历、家庭概况、兴趣爱好、理想与报负等。 这部分的陈述务必简明扼要、抓住要点。例如介绍自己的学历，一般只需谈本专科以上的学历。工作单位如果多，选几个有代表性的或者你认为重要的介绍，就可以了，但这些内容一定要和面试及应考职位有关系。请保证叙述的线索清晰，一个结构混乱、内容过长的开场白，会给考官们留下杂

乱无章、个性不清晰的印象，并且让考官倦怠，削弱对继续进行的面试的兴趣和注意力。

4. 项目的同源处理，跨越相关

参考答案：

（1）什么是跨域？

跨域，是指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 实施的安全限制。

（2）什么是同源策略？

同源策略（Same origin policy）是一种约定，它是浏览器最核心也最基本的安全功能，它是由 Netscape 提出的一个著名的安全策略。

同源策略是浏览器的行为，是为了保护本地数据不被 JavaScript 代码获取回来的数据污染，因此拦截的是客户端发出的请求回来的数据接收，即请求发送了，服务器响应了，但是无法被浏览器接收。

（3）其主要限制以下几个方面：

- a. Cookie 、LocalStorage 和 IndexDB 无法读取
- b. 无法获取或操作另一个资源的 DOM
- c. AJAX 请求不能发送

那么什么是同源呢？所谓的同源，就是指两个页面具有相同的协议，主机（也常说域名），端口，三个要素缺一不可。

（4）常见跨域方案

- a. 通过 jsonp 跨域
- b. document.domain+iframe 跨域
- c. location.hash + iframe
- d. window.name + iframe 跨域
- e. postMessage 跨域
- f. 跨域资源共享（CORS）
- g. nginx 代理跨域
- h. nodejs 中间件代理跨域
- i. WebSocket 协议跨域

5. 遇到过什么安全问题，怎么解决的

参考答案：

前端常见的安全方面问题：

- (1) iframe
- (2) opener
- (3) CSRF（跨站请求伪造）
- (4) XSS（跨站脚本攻击）
- (5) ClickJacking（点击劫持）
- (6) HSTS（HTTP 严格传输安全）
- (7) CND 劫持

1) iframe

- a、如何让自己的网站不被其他网站的 iframe 引用？

```
// 检测当前网站是否被第三方 iframe 引用// 若相等证明没有被第三方引用，  
若不等证明被第三方引用。当发现被引用时强制跳转百度。if(top.location !=  
self.location){  
    top.location.href = 'http://www.baidu.com'  
}
```

b、如何禁用，被使用的 iframe 对当前网站某些操作？

sandbox 是 html5 的新属性，主要是提高 iframe 安全系数。iframe 因安全问题而臭名昭著，这主要是因为 iframe 常被用于嵌入到第三方中，然后执行某些恶意操作。

现在有一场景：我的网站需要 iframe 引用某网站，但是不想被该网站操作 DOM、不想加载某些 js（广告、弹框等）、当前窗口被强行跳转链接等，我们可以设置 sandbox 属性。如使用多项用空格分隔。

allow-same-origin：允许被视为同源，即可操作父级 DOM 或 cookie 等

allow-top-navigation：允许当前 iframe 的引用网页通过 url 跳转链接或加载

allow-forms：允许表单提交

allow-scripts：允许执行脚本文件

allow-popups：允许浏览器打开新窗口进行跳转

“ ”：设置为空时上面所有允许全部禁止

2) opener

如果在项目中需要 打开新标签 进行跳转一般会有两种方式

```
// 1) HTML -> <a target='_blank' href='http://www.baidu.com'>  
// 2) JS -> window.open('http://www.baidu.com')  
/*
```

- * 这两种方式看起来没有问题，但是存在漏洞。
- * 通过这两种方式打开的页面可以使用 `window.opener` 来访问源页面的 `window` 对象。
- * 场景：A 页面通过 `<a>` 或 `window.open` 方式，打开 B 页面。但是 B 页面存在恶意代码如下：
`*window.opener.location.replace('https://www.baidu.com')` 【此代码仅针对打开新标签有效】
- * 此时，用户正在浏览新标签页，但是原来网站的标签页已经被导航到了百度页面。
- * 恶意网站可以伪造一个足以欺骗用户的页面，使得进行恶意破坏。
- * 即使在跨域状态下 `opener` 仍可以调用 `location.replace` 方法。
- */

a、``

`a 标签跳转 url`

`<!--`

通过 `rel` 属性进行控制：
`noopener`：会将 `window.opener` 置空，从而源标签页不会进行跳转（存在浏览器兼容问题）
`noreferrer`：兼容老浏览器/火狐。禁用 HTTP 头部 `Referer` 属性（后端方式）。
`nofollow`：SEO 权重优化，详情见 https://blog.csdn.net/qq_33981438/article/details/80909881

`-->`

b、`window.open()`

`<button onclick='openurl("http://www.baidu.com")'>click 跳转</button>`

```
function openurl(url) {  
    var newTab = window.open();
```

```
newTab.opener = null;  
newTab.location = url;}
```

3) CSRF / XSRF (跨站请求伪造)

你可以这么理解 CSRF 攻击: 攻击者盗用了你的身份, 以你的名义进行恶意请求。它能做的事情有很多包括: 以你的名义发送邮件、发信息、盗取账号、购买商品、虚拟货币转账等。总结起来就是: 个人隐私暴露及财产安全问题。

/* *

阐述 CSRF 攻击思想: (核心 2 和 3)

- * 1、浏览并登录信任网站 (举例: 淘宝)
 - * 2、登录成功后在浏览器产生信息存储 (举例: cookie)
 - * 3、用户在没有登出淘宝的情况下, 访问危险网站
 - * 4、危险网站中存在恶意代码, 代码为发送一个恶意请求 (举例: 购买商品/余额转账)
 - * 5、携带刚刚在浏览器产生的信息进行恶意请求
 - * 6、淘宝验证请求为合法请求 (区分不出是否是该用户发送)
 - * 7、达到了恶意目标
- */

防御措施 (推荐添加 token / HTTP 头自定义属性)

- (1) 涉及到数据修改操作严格使用 post 请求而不是 get 请求
- (2) HTTP 协议中使用 Referer 属性来确定请求来源进行过滤 (禁止外域)
- (3) 请求地址添加 token , 使黑客无法伪造用户请求
- (4) HTTP 头自定义属性验证 (类似上一条)
- (5) 显示验证方式: 添加验证码、密码等

4) XSS/CSS (跨站脚本攻击)

XSS 又叫 CSS (Cross Site Script)，跨站脚本攻击：攻击者在目标网站植入恶意脚本 (js / html)，用户在浏览器上运行时可以获取用户敏感信息 (cookie / session)、修改 web 页面以欺骗用户、与其他漏洞相结合形成蠕虫等。

浏览器遇到 html 中的 script 标签时，会解析并执行其中的 js 代码。

针对这种情况，我们对特殊字符进行转译就好了 (vue/react 等主流框架已经避免类似问题，vue 举例：不能在 template 中写 script 标签，无法在 js 中通过 ref 或 append 等方式动态改变或添加 script 标签)

XSS 类型：

持久型 XSS：将脚本植入到服务器上，从而导致每个访问的用户都会执行

非持久型 XSS：对个体用户某 url 的参数进行攻击

防御措施 (对用户输入内容和服务端返回内容进行过滤和转译)

- a. 现代大部分浏览器都自带 XSS 筛选器，vue / react 等成熟框架也对 XSS 进行一些防护
- b. 即便如此，我们在开发时也要注意和小心
- c. 对用户输入内容和服务端返回内容进行过滤和转译
- d. 重要内容加密传输
- e. 合理使用 get/post 等请求方式
- f. 对于 URL 携带参数谨慎使用
- g. 我们无法做到彻底阻止，但是能增加黑客攻击成本，当成本与利益不符时自然会降低风险

5)、ClickJacking (点击劫持)

ClickJacking 翻译过来被称为点击劫持。一般会利用透明 iframe 覆盖原网页诱导用户进行某些操作达成目的。

防御措施

- a. 在 HTTP 投中加入 X-FRAME-OPTIONS 属性，此属性控制页面是否可被嵌入 iframe 中【DENY：不能被所有网站嵌套或加载；SAMEORIGIN：只能被同域网站嵌套或加载；ALLOW-FROM URL：可以被指定网站嵌套或加载。】
- b. 判断当前网页是否被 iframe 嵌套（详情在第一条 firame 中）

6)、HSTS (HTTP Strict Transport Security: HTTP 严格传输安全)

网站接受从 HTTP 请求跳转到 HTTPS 请求的做法，例如我们输入

“http://www.baidu.com”或“www.baidu.com”最终都会被 302 重定向到“https://www.baidu.com”。这就存在安全风险，当我们第一次通过 HTTP 或域名进行访问时，302 重定向有可能会被劫持，篡改成一个恶意或钓鱼网站。HSTS：通知浏览器此网站禁止使用 HTTP 方式加载，浏览器应该自动把所有尝试使用 HTTP 的请求自动替换为 HTTPS 进行请求。用户首次访问时并不受 HSTS 保护，因为第一次还未形成链接。我们可以通过 浏览器预置 HSTS 域名列表或将 HSTS 信息加入到域名系统记录中，来解决第一次访问的问题。

7)、CDN 劫持

出于性能考虑，前端应用通常会把一些静态资源存放到 CDN（Content Delivery Networks）上面，例如 js 脚本和 style 文件。这么做可以显著提高前端应用的访问速度，但与此同时却也隐含了一个新的安全风险。如果攻击者劫持了 CDN，或者对 CDN 中的资源进行了污染，攻击者可以肆意篡改我们的前端页面，对用户实施攻击。

现在的 CDN 以支持 SRI 为荣，script 和 link 标签有了新的属性 integrity，这个属性是为了防止校验资源完整性来判断是否被篡改。它通过 验证获取文件的哈希值是否和你提供的哈希值一样来判断资源是否被篡改。

使用 SRI 需要两个条件：一是要保证 资源同域 或开启跨域，二是在<script> 中 提供签名 以供校验。

```
<script
  crossorigin="anonymous"
  integrity="sha384-xBuQ/xzmlsLoJpyjoggmTEz8OWUFM0/RC5BsqQBDX2v5cMvDHcMakNTNr
  src="http://lib.baomitu.com/jquery/3.2.1/jquery.min.js">
</script>
```

integrity 属性分为两个部分，第一部分是指定哈希值的生成算法(例: sha384)，第二部分是经过编码的实际哈希值，两者之前用一个短横(-)来分隔

6.让你带领一个小团队完成一个项目，我会怎么做？

参考答案：

根据团队的技术栈，指定开发计划，然后分配任务，定人定点，推进项目的进展。

7. 前端的项目如何进行优化，移动端呢

参考答案：

前端性能优化有七大手段：减少请求数量，减少资源大小，优化网络连接，优化资源加载，减少重绘回流，使用性能更好的 API 和构建优化

1) 减少请求数量：通过减少重定向，使用缓存，不适用 CSS@import，避免使用空的 src 和 href 等手段

2) 减少资源大小：通过压缩 HTML，CSS，JS，图片，此外在安卓下可以使用 webp 格式的图片，它具有更优的图像数据压缩算法，能带来更小的图片体积，还可以开启 gzip，gzip 编码是以后总用来改进 web 应用程序性能的技术，

3) 优化网络连接：使用 CDN，使用 DNS 预解析，并行连接，

- 4) 优化资源加载，通过优化资源加载位置和时机，使用资源预加载 preload 和资源预读取 prefetch
- 5) 减少重绘回流，
避免使用层级较深的 CSS 选择器，以提高 CSS 渲染效率
避免使用 CSS 表达式，
给元素适当的定义高度或最小高度，否则元素的动态内容载入时，会出现页面晃动，造成回流，
不要使用 table 布局，
能用 CSS 实现的效果，尽量使用 CSS 而不用 JS 实现
- 6) 使用性能更好的 api

8.项目中使用了 iframe，说说 iframe 的优缺点

参考答案：

iframe 的优点：

- (1) iframe 能够原封不动地把嵌入的网页展现出来。
- (2) 如果有多个网页调用 iframe，只需要修改 iframe 的内容，就可以实现对调用 iframe 的每一个页面内容的更改，方便快捷。
- (3) 网页如果为了统一风格，头部和版本都是一样的，就可以写成一个页面，用 iframe 来嵌套，可以增加代码的可重用性。
- (4) 如果遇到加载缓慢的第三方内容，如图标和广告等，可以用 iframe 来解决。

iframe 的缺点：

- (1) 会产生很多页面，不容易管理。
- (2) 在几个框架中都出现上下、左右滚动条时，这些滚动条除了会挤占已经非常有限的页面空间外，还会分散访问者的注意力。

- (3) 使用框架结构时，必须保证正确设置所有的导航链接，否则会给访问者带来很大的麻烦。比如被链接的页面出现在导航框架内，这种情况下会导致链接死循环。
- (4) 很多的移动设备（PDA 手机）无法完全显示框架，设备兼容性差。
- (5) iframe 框架页面会增加服务器的 http 请求，对于大型网站是不可取的。

第十一章 职业发展

1.介绍一下前端的学习经历

参考回答：

- (1) 原生 js/html/css 阶段
- (2) html5/css3/jquery/js 进阶阶段
- (3) vue 阶段
- (4) react 阶段

2. 作为一个专业的前端，你觉得应该掌握哪些知识

参考回答：

- (1) 计算机基础（计算机网络，数据结构，算法，编译原理...）
- (2) 前端常用的语言基础（JS，HTML，CSS）也可以看下 TS
- (3) 上面一条的拓展。为了更好的沟通和更全面的认知最好也涉猎一下服务端/客户端知识
- (4) 用于团队协作的工具（如：Git）
- (5) 用于工程化建设的工程化工具（如：Webpack）

3.什么时候接触前端

参考回答：

根据自身实际情况回答

4.大学学过哪些编程的课

参考回答：

《C++语言程序设计》《Java 语言程序设计》，《数据结构》，《编译原理》，《软件工程》，《算法分析与设计》，《数据库系统原理》，《计算机组成原理》，《操作系统》，《互联网程序设计》，《汇编语言与计算机系统组成》，《计算机网络》另外还有很多选修课，就不一一列举了。

6.对未来三年职业的规划

参考答案：

成为一个全栈工程师

7.你一般是通过什么方式学习前端的？

参考答案：

自学，W3school 等网站，阮一峰、冢羽等大神的博客，GitHub，掘金，segmentfault 等交流平台，（根据自身实际情况回答）

8.你还有什么我没问到的优势吗

参考回答：

- 1) 编译原理：TypeScript、Vue 3、跨端框架都强烈依赖这项技术，如果你是月薪超过 20k 的前端，一定要重温一下编译原理，才能看懂这些技术的源码。
- 2) 函数式编程：React 的 Hooks 用到的思想来自于函数式编程，什么是副作用，你能不能说清楚，20k 以上的前端建议重点学习一下。
- 3) 后端知识：前端技术已经有五年没有较大变化了，想要突破瓶颈的前端可以以 Node.js 为抓手（阿里喜欢说这个词）来进入后端编程领域，需要学的包括但不限于关系型数据库、Redis、非关系型数据库、HTTP、TCP/IP、MVC 架构等
- 4) 前端知识巩固：Vue 3 和 React 的源码可以了解一下

9.看过什么书

推荐阅读：

《Head First HTML 与 CSS》、《CSS 揭秘》、《CSS 世界》、《CSS 权威指南（第三版）》、《JavaScript 语言精粹》、《JavaScript 高级程序设计（第三版）》...

10.比较厉害的技术

参考回答：

- 1) 前端框架和语言层面

9 月份 Vue3.0 发布，声称对 TypeScript 有着更好的开发体验，通过从不同框架级别 TS 支持上，我们可以看出社区的整个风向从 2019 年的大家都去学习应用 TS，变成了大家如何把 TS 用的更好这个方向上来了。

所以我认为今年 TypeScript 的火热程度还是应该排名很靠前的，我今年也使用 TypeScript 重构了 Daruk 的服务框架推出了 2.0 版本，让 TS 开发者拥有更好的 TS 开发体验。

接下来就是两大重磅框架的更新历程对比，Vue3 前面说了一句。而 React 也在十月也发布了 React 17 的 release 版本。这两大主流框架的频繁更新，也说明了社区和作者都在一同演化。

在 Vue 3 中除了更好的支持 TS 外，还更新了 Composition API。而 React 17 主要是集中精力在升级体验上，虽然没有新的 Feature 但是提升了和解决了很多之前版本潜在的问题。

要说哪个最火还是要看个人实际的使用场景和喜好，但是 2020 年来看还没有别的框架可以与之一战。

2) 大前端相关技术栈

今年基于 Chromium 的微软 edge 浏览器也已经推出。google 在 web 端的发展产生了对开发者深刻的影响。Chrome 80+ 也已经发布多个版本，提供了一系列的新特性，比如 Core Web Vitals 标准，Desktop PWA 等都值得我们去关注。

我们说完了浏览器相关的那点技术之后，再聊聊大前端相关的一些技术实践，比如 Flutter。

很多前端在今年已经从 web 开发转型为 Flutter 开发，学习和使用 Dart 技术来构建 UI，这是很多大厂的前端工程师正在经历的事情（包括我的部门也在尝试这个事情），这个趋势应该在未来几年还会持续。

客户端 electron 在今年也有着长足的进展，一年内多次更新版本一路到了 10.1.5。随着疫情影响，国内在线教育的又一波兴起。很多桌面软件，网课软件都在采用这个技术来进行开发，市场上的岗位也开始变多，electron 技术可以说在今年也有火的趋势。

然后我们再看看 BFF 层，nestjs 依然坚挺，越来越多的人开始跳过学习 express 和 koa 开始学习更丰富的 web 框架了，比如 egg 或者我的 daruk，开发者已经在慢慢形成共识，在 web framework 的路上开始越走越远，裸写 nodejs web 服务的时代已经开始慢慢褪去。

不得不提的还有 serverless 在前端的普及，在 2020 年到达了一个新的高潮。阿里云，腾讯云，头条云等等国内的互联网厂商也都开始大玩 serverless 概念。从对内服务开始转向对外服务，普及的势头很猛，也有落地的趋势和场景。今年的 D2 同样也有 serverless 的专场，可见受重视程度非比寻常。

3) 工程化提效和个人素质提升

再离我们近一些的推动生产力的技术，比如据我所知在用 CI/CD 和 pipeline 管理上线流程的公司越来越多，这种去年还可以出去吹一吹的东西，今年也逐步变成了业界标配基础能力，如果不会的同学可要抓紧学习了。

2019 年前大家都疯狂吐槽面试刷 medium 题目没用，而 2020 年后大家开始默认面试某些公司都至少要刷到 medium 程度的题目。这对很多前端来说是一个心智和素质的提升与转变，大家在接触新技术的同时，也慢慢发现，前端整个职业环境的变化，越来越多的公司对人的整体综合素质要求变高了。

11.你学前端怎么坚持下来的

参考回答：

写博客，从总结基础姿势开始

写工具，chrome 插件，vscode 插件，开发成本都不高，重要的是想一些点子，解决一些实际问题，问题不用太大，关键是有用

写点有意思的东西，例如搞个小游戏给自己玩，写个小彩蛋什么的

学点有意思的，就是那种你觉得很有意思，学会了可以干点不一样的事情的东西，例如 app 开发，nodejs 开发等

12.学过哪些框架？

参考回答：

Bootstrap:

让你的页面更简洁、直观、强悍、移动设备优先的前端开发框架, 让 web 开发更迅速、更简单。它还提供了更优雅的 HTML 和 CSS 规范，它即是由动态 CSS 语言 Less 写成。有着丰富的网格布局系统以及丰富的可重用组件，还有强大的支持十几的 JavaScript、jQuery 插件以及组件定制等。

React:

React 可以非常轻松地创建用户交互界面。为你应用的每一个状态设计简洁的视图, 在数据改变时 React 也可以高效地更新渲染界面。

vue.js:

Vue.js 是一个构建数据驱动的 web 界面的渐进式框架。Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

Angular:

Angular 是一款优秀的前端 JS 框架，已经被用于 Google 的多款产品当中。

AngularJS 有着诸多特性，最为核心的是：MVVM、模块化、自动化双向数据绑定、语义化标签、依赖注入等等。

（前端框架还有很多，建议根据自身的实际学习情况回答）

13.你理解的框架

第十二章 Hr 面

1.自我介绍

参考回答：

1) 结合招聘岗位，只讲重点。

简历内容这么多，实际的你，更有很多很多可以描述的东西。但时间有限，没有面试官会听你说个没完。

根据你求职的岗位，说重点即可。

其实简历制作的原则，也是一样。只是自我介绍时间更短，内容更精华。

2) 有理有据，少说空话。

如果你说“自己学习能力强”，这就是一句假大空的话。谁都可以说自己学习能力强。

你如果真的在这方面有突出，就要举一个例子，比如是 1 个月从 0 到 1 考了什么证书等。

3) 有开头有结尾，有逻辑。

开头问候，结尾总结。中间 1、2、3 条理清楚。

4) 特别提醒。

自我介绍中的内容，很可能是面试官后续发问的内容。所以，一是要引起重视，讲最重要的，你最想让面试官知道的内容。而是做好准备，扬长避短，不要给自己挖坑。

比如前面你说自己学习能力强，但是没有举例子。

面试官很可能，顺着你的话问你，怎么证明你学习能力强呢？结果你答不上来，或者是非常普通的成绩，根本不能算是“学习能力强”。那你就是给自己挖坑了。

其实自我介绍也是每个人各有不同，说起来也只能提供大致的思路。和你具体的求职岗位、经历、个人风格有很大的关系。

2.为什么要学习前端

参考回答：

我首先是一个程序员，然后我喜欢并且擅长写 javascript，再然后才是一个所谓的“前端工程师”。

javascript 是一个真正意义上的跨平台语言，浏览器是少有的横跨 PC/移动设备的 GUI 平台，nodejs 也是一个高性能且开发效率高的平台，我相信未来随着计算能力的不断提高，这些技术必然会生出更多炫酷的东西。

3.到现在为止接触过几个项目，有在哪里实习过？

参考回答：

- 1) 梳理之前实习的工作，按岗位相关性强弱排序
- 2) 按照：总结-做什么-做出了什么成果进行讲述
- 3) 如果你的工作不方便量化成果，注重在叙述中展示你的能力。
- 4) 礼貌、大方，对所应聘公司有一定的了解

4.让你收获最多的项目，你做了什么？

参考回答：

项目经验这块因人而异，把觉得做的有亮点的东西挑出来，从四个方面来准备：

功能模块的实现

达到了什么效果

遇到了什么问题，是如何解决的

涉及到的相关知识点

项目经验和知识点 其实是一个双向的过程，要 试图去找到它们之间的联系：
谈到项目经验的时候，可以说：做完 xx 之后，我还去了解了一下 yy 背后的原理，
xxx，这里体现的是你是否有求知欲 。

谈到知识点的时候，可以说：yy 的原理是这样的，在 xx 项目中我是如何应用它
来解决问题的，这里体现的是你是否具备把知识付诸实践的能力。

5.个人的优缺点

参考回答：

- 1) 不宜说自己没缺点。
- 2) 不宜把那些明显的优点说成缺点。

- 3) 不宜说出严重影响所应聘工作的缺点。
- 4) 不宜说出令人不放心、不舒服的缺点。
- 5) 可以说出一些对于所应聘工作“无关紧要”的缺点，甚至是一些表面上看是缺点，从工作的角度看却是优点的缺点。

答：我的工作执行力很高，领导给我安排的任务我也能较快的完成，但是可能在工作的的时候会少一些深入的独立思考，对整个工作的安排缺乏战略眼光，对于这方面能力需要我进一步的努力，我也将继续学习，通过阅读、向前辈请教等改善自己的不足。

6.读不读研

参考回答：

对于大公司，可以答：

我目前没有这方面的打算，但是如果公司的发展需要我提升自己这方面的能力和学历，我会认真考虑考研，以便提高自己的职业技能和素养，更好地为公司谋取利益。

对于小公司，可以答：

我并不打算考研，对于我来说，我更加重视在社会上实际工作经验的积累，我觉得我的工作更加适合实践而不是理论，比起考研，我更加希望能跟公司一起成长，为公司做出贡献

7.说说你最荣耀的事

参考回答：

作为面试官，我们希望了解你最大的闪光点，而不是你的缺点、不足。

对这个问题感到很头疼的，不妨认真想想，你二十多年的人生，有为一件事努力过吗，有满腔热情去做的事情吗，有即便前路艰险也义无反顾的事吗？

如果没有，你不觉得自己的人生是为别人而活，是浑浑噩噩的过日子吗？

关于这个问题，你也无法用一个标准答案来回答。

我见过很多人讲到某些事情时眼睛里散发出的那种神采，也见过无数人试图编造一个故事时的眼神迷离，你真的无法在这种事情上撒谎，也没用必要在这种事情上撒谎。