2021-2022最新版前端工程师面试题手册

目录

1| 前端基础........................................................................................................................ 11

1.1|HTTP/HTML/浏览器................................................................................................ 11





































































说一下http和https......................................................................................... 11

tcp三次握手，一句话概括............................................................................ 12

TCP和UDP的区别.........................................................................................12

WebSocket的实现和应用................................................................................12

HTTP请求的方式，HEAD方式....................................................................13

一个图片url访问后直接下载怎样实现？....................................................13

说一下webQuality（无障碍）...................................................................... 14

几个很实用的BOM属性对象方法?..............................................................14

说一下HTML5dragapi...................................................................................15

说一下http2.0.................................................................................................. 15

补充400和401、403状态码........................................................................15

fetch发送2次请求的原因............................................................................. 16

Cookie、sessionStorage、localStorage的区别................................................ 16

说一下webworker............................................................................................17

对HTML语义化标签的理解..........................................................................17

iframe是什么？有什么缺点？.......................................................................17

Doctype作用?严格模式与混杂模式如何区分？它们有何意义?................17

Cookie如何防范XSS攻击..............................................................................18

Cookie和session的区别.................................................................................18

一句话概括RESTFUL.....................................................................................18

讲讲viewport和移动端布局...........................................................................18

click在ios上有300ms延迟，原因及如何解决？......................................18

addEventListener参数......................................................................................18

cookiesessionStoragelocalStorage区别...........................................................19

cookiesession区别...........................................................................................19

介绍知道的http返回的状态码......................................................................19

http常用请求头...............................................................................................21

强，协商缓存.................................................................................................. 24

讲讲304............................................................................................................25

强缓存、协商缓存什么时候用哪个..............................................................25

前端优化...........................................................................................................26

GET和POST的区别.......................................................................................26

301和302的区别........................................................................................... 27

HTTP支持的方法............................................................................................27

1



















































如何画一个三角形.......................................................................................... 27

状态码304和200........................................................................................... 27

说一下浏览器缓存.......................................................................................... 28

HTML5新增的元素.........................................................................................28

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？....28

cookie和session的区别，localstorage和sessionstorage的区别.................29

常见的HTTP的头部.......................................................................................29

HTTP2.0 的特性..............................................................................................30

cache-control的值有哪些...............................................................................30

浏览器在生成页面的时候，会生成那两颗树？..........................................30

csrf和xss的网络攻击及防范.........................................................................30

怎么看网站的性能如何.................................................................................. 31

介绍HTTP协议(特征).....................................................................................31

输入URL到页面加载显示完成发生了什么?.............................................. 31

说一下对Cookie和Session的认知，Cookie有哪些限制？....................... 31

描述一下XSS和CRSF攻击？防御方法？..................................................32

知道304吗，什么时候用304？...................................................................32

具体有哪些请求头是跟缓存相关的..............................................................32

cookie和session的区别..................................................................................32

cookie有哪些字段可以设置...........................................................................33

cookie有哪些编码方式？...............................................................................33

除了cookie，还有什么存储方式。说说cookie和localStorage的区别.....33

浏览器输入网址到页面渲染全过程..............................................................34

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？...........................................................................................................................34













http常见的请求方法.......................................................................................35

get和post的区别............................................................................................ 35

说说302，301，304的状态码......................................................................35

web性能优化................................................................................................... 35

浏览器缓存机制.............................................................................................. 35

post和get区别................................................................................................ 36

1.2|CSS............................................................................................................................36

























说一下css盒模型............................................................................................36

画一条0.5px的线............................................................................................37

link标签和import标签的区别.......................................................................37

transition和animation的区别.........................................................................38

Flex布局...........................................................................................................38

BFC（块级格式化上下文，用于清楚浮动，防止margin重叠等）..........39

垂直居中的方法.............................................................................................. 39

关于JS动画和css3动画的差异性................................................................41

说一下块元素和行元素.................................................................................. 41

多行元素的文本省略号.................................................................................. 41

visibility=hidden,opacity=0，display:none......................................................42

双边距重叠问题（外边距折叠）..................................................................42

2

































position属性比较............................................................................................42

浮动清除...........................................................................................................43

css3新特性.......................................................................................................43

CSS选择器有哪些，优先级呢.......................................................................43

清除浮动的方法，能讲讲吗..........................................................................44

怎么样让一个元素消失，讲讲......................................................................44

介绍一下盒模型.............................................................................................. 44

position相关属性.............................................................................................45

css动画如何实现.............................................................................................45

如何实现图片在某个容器中居中的？..........................................................46

如何实现元素的垂直居中.............................................................................. 46

CSS3中对溢出的处理.....................................................................................46

float的元素，display是什么..........................................................................46

隐藏页面中某个元素的方法..........................................................................46

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...................................................................................................................46



























什么是BFC...................................................................................................... 47

calc属性...........................................................................................................47

有一个width300，height300，怎么实现在屏幕上垂直水平居中..............47

display：table和本身的table有什么区别....................................................48

position属性的值有哪些及其区别.................................................................48

z-index的定位方法.........................................................................................48

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？...49

对CSS的新属性有了解过的吗？..................................................................49

用的最多的css属性是啥？........................................................................... 49

line-height和height的区别............................................................................49

设置一个元素的背景颜色，背景颜色会填充哪些区域？..........................49

知道属性选择器和伪类选择器的优先级吗..................................................49

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

49





用css实现一个硬币旋转的效果................................................................... 50

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...................................................................................................................51













CSS画正方体，三角形...................................................................................51

overflow的原理................................................................................................ 53

清除浮动的方法.............................................................................................. 53

box-sizing的语法和基本用处........................................................................ 54

使元素消失的方法有哪些？..........................................................................54

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。..................................................................54









说说盒子模型.................................................................................................. 54

display............................................................................................................... 54

怎么隐藏一个元素.......................................................................................... 55

display:none和visibilty:hidden的区别...........................................................5

3





















相对布局和绝对布局，position:relative和obsolute。..................................55

flex布局............................................................................................................55

block、inline、inline-block的区别。........................................................... 56

css的常用选择器.............................................................................................56

css布局.............................................................................................................56

css定位.............................................................................................................57

relative定位规则..............................................................................................57

垂直居中...........................................................................................................58

css预处理器有什么........................................................................................ 58

1.3|JavaScript.................................................................................................................. 58



























get请求传参长度的误区.................................................................................58

补充get和post请求在缓存方面的区别.......................................................58

说一下闭包...................................................................................................... 59

说一下类的创建和继承.................................................................................. 59

如何解决异步回调地狱.................................................................................. 61

说说前端中的事件流...................................................................................... 61

如何让事件先冒泡后捕获.............................................................................. 61

说一下事件委托.............................................................................................. 62

说一下图片的懒加载和预加载......................................................................62

mouseover和mouseenter的区别.....................................................................62

JS的new操作符做了哪些事情..................................................................... 62

改变函数内部this指针的指向函数（bind，apply，call的区别）........... 62

JS的各种位置，比如 clientHeight,scrollHeight,offsetHeight,以及 scrollTop,

offsetTop,clientTop的区别？...................................................................................63









































JS拖拽功能的实现..........................................................................................63

异步加载JS的方法.........................................................................................63

Ajax解决浏览器缓存问题..............................................................................64

JS的节流和防抖..............................................................................................64

JS中的垃圾回收机制......................................................................................64

eval是做什么的...............................................................................................65

如何理解前端模块化...................................................................................... 66

说一下CommonJS、AMD和CMD.................................................................66

对象深度克隆的简单实现.............................................................................. 67

实现一个once函数，传入函数参数只执行一次.........................................67

将原生的ajax封装成promise........................................................................ 67

JS监听对象属性的改变..................................................................................68

如何实现一个私有变量，用getName方法可以访问，不能直接访问......68

==和===、以及Object.is的区别...................................................................69

setTimeout、setInterval和requestAnimationFrame之间的区别...................69

实现一个两列等高布局，讲讲思路..............................................................70

自己实现一个bind函数.................................................................................70

用setTimeout来实现setInterval..................................................................... 70

JS怎么控制一次加载一张图片，加载完后再加载下一张.........................71

代码的执行顺序.............................................................................................. 72

4



































































如何实现sleep的效果（es5或者es6）.......................................................72

简单的实现一个promise.................................................................................73

Function.\_proto\_(getPrototypeOf)是什么？......................................................73

实现JS中所有对象的深度克隆（包装对象，Date对象，正则对象）....74

简单实现Node的Events模块....................................................................... 76

箭头函数中this指向举例.............................................................................. 77

JS判断类型......................................................................................................77

数组常用方法.................................................................................................. 77

数组去重...........................................................................................................78

闭包有什么用................................................................................................. 78

事件代理在捕获阶段的实际应用..................................................................78

去除字符串首尾空格...................................................................................... 78

性能优化...........................................................................................................79

来讲讲JS的闭包吧.........................................................................................79

能来讲讲JS的语言特性吗.............................................................................79

如何判断一个数组(讲到typeof差点掉坑里)................................................80

你说到typeof，能不能加一个限制条件达到判断条件...............................80

JS实现跨域......................................................................................................80

JS基本数据类型..............................................................................................80

JS深度拷贝一个元素的具体实现..................................................................80

之前说了ES6set可以数组去重，是否还有数组去重的方法.....................81

重排和重绘，讲讲看...................................................................................... 81

JS的全排列......................................................................................................81

跨域的原理...................................................................................................... 82

不同数据类型的值的比较，是怎么转换的，有什么规则..........................82

null==undefined为什么.................................................................................82

this的指向哪几种...........................................................................................83

暂停死区...........................................................................................................83

AngularJS双向绑定原理.................................................................................83

写一个深度拷贝.............................................................................................. 84

简历中提到了requestAnimationFrame，请问是怎么使用的.......................85

有一个游戏叫做FlappyBird，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）....................................... 85



编写代码，满足以下条件：（1）Hero("37er");执行结果为 Hi!Thisis37er

（2）Hero("37er").kill(1).recover(30);执行结果为 Hi! This is 37er Kill 1 bug

Recover30bloods（3）Hero("37er").sleep(10).kill(2)执行结果为 Hi!Thisis37er//

等待10s后 Kill2bugs //注意为bugs（双斜线后的为提示信息，不需要打印）

85











什么是按需加载.............................................................................................. 86

说一下什么是virtualdom............................................................................... 86

webpack用来干什么的....................................................................................86

ant-design优点和缺点....................................................................................87

JS中继承实现的几种方式，..........................................................................87

5

















写一个函数，第一秒打印1，第二秒打印2................................................87

Vue的生命周期...............................................................................................88

简单介绍一下symbol...................................................................................... 88

什么是事件监听.............................................................................................. 89

介绍一下promise，及其底层如何实现.........................................................89

说说C++,Java，JavaScript这三种语言的区别.............................................90

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..........................91

















什么是js的闭包？有什么作用，用闭包写个单例模式.............................95

promise+Generator+Async的使用...................................................................95

事件委托以及冒泡原理。.............................................................................. 99

写个函数，可以转化下划线命名到驼峰命名..............................................99

深浅拷贝的区别和实现................................................................................ 100

JS中string的startwith和indexof两种方法的区别................................... 101

JS字符串转数字的方法................................................................................101

letconstvar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........................................................................102







ES6箭头函数的特性.....................................................................................102

setTimeout和Promise的执行顺序...............................................................102

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.........................................................................................................................103





平时是怎么调试JS的...................................................................................103

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................................................103



















setTimeout(fn,100);100毫秒是如何权衡的.................................................. 105

JS的垃圾回收机制........................................................................................105

写一个newBind函数，完成bind的功能。...............................................105

怎么获得对象上的属性：比如说通过Object.key（）..............................106

简单讲一讲ES6的一些新特性....................................................................106

call和apply是用来做什么？.......................................................................106

了解事件代理吗，这样做有什么好处........................................................107

如何写一个继承？........................................................................................ 107

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).............................. 108



















给两个构造函数A和B，如何实现A继承B？........................................108

问能不能正常打印索引................................................................................ 109

如果已经有三个promise，A、B和C，想串行执行，该怎么写？.........109

知道private和public吗............................................................................... 109

基础的js.........................................................................................................109

async和await具体该怎么用？....................................................................110

知道哪些ES6，ES7的语法......................................................................... 110

promise和await/async的关系.......................................................................110

JS的数据类型................................................................................................110

6









JS加载过程阻塞，解决方法。....................................................................110

JS对象类型，基本对象类型以及引用对象类型的区别...........................110

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？.........................................................................................................................111





















































怎么实现一个计算一年中有多少周？........................................................111

面向对象的继承方式.................................................................................... 111

JS的数据类型................................................................................................112

引用类型常见的对象.................................................................................... 112

es6的常用......................................................................................................113

class.................................................................................................................113

口述数组去重................................................................................................ 113

继承.................................................................................................................113

call和apply的区别.......................................................................................114

es6的常用特性..............................................................................................115

箭头函数和function有什么区别................................................................. 115

new操作符原理.............................................................................................115

bind,apply,call.................................................................................................115

bind和apply的区别......................................................................................115

数组的去重.................................................................................................... 115

闭包.................................................................................................................116

promise实现...................................................................................................116

assign的深拷贝..............................................................................................117

说promise，没有promise怎么办................................................................ 118

事件委托.........................................................................................................119

箭头函数和function的区别......................................................................... 119

arguments........................................................................................................ 119

箭头函数获取arguments...............................................................................119

Promise............................................................................................................119

事件代理.........................................................................................................120

Eventloop.........................................................................................................120

2| 前端核心...................................................................................................................... 120

2.1| 服务端编程............................................................................................................ 120











JSONP的缺点................................................................................................ 120

跨域（jsonp，ajax）..................................................................................... 120

如何实现跨域................................................................................................ 121

dom是什么，你的理解？.............................................................................121

关于dom的api有什么.................................................................................121

2.2|Ajax.........................................................................................................................121







ajax返回的状态.............................................................................................121

实现一个Ajax................................................................................................122

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................................... 122



写出原生Ajax................................................................................................124

7





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？

124





Fetch和Ajax比有什么优缺点？.................................................................125

原生JS的ajax............................................................................................... 125

2.3| 移动web开发....................................................................................................... 125























































知道PWA吗..................................................................................................125

移动布局方案................................................................................................ 125

Rem,Em..........................................................................................................126

flex布局及优缺点..........................................................................................127

Rem布局及其优缺点....................................................................................128

百分比布局.................................................................................................... 129

移动端适配1px的问题.................................................................................130

移动端性能优化相关经验............................................................................131

toB 和 toC 项目的区别................................................................................132

移动端兼容性................................................................................................ 132

小程序.............................................................................................................133

2X图 3X图适配...........................................................................................133

图片在安卓上，有些设备模糊问题............................................................134

固定定位布局键盘挡住输入框内容............................................................134

click的300ms延迟问题和点击穿透问题...................................................134

phone及ipad下输入框默认内阴影.............................................................135

防止手机中页面放大和缩小........................................................................135

px、em、rem、%、vw、vh、vm这些单位的区别.....................................135

移动端适配-dpr浅析...................................................................................136

移动端扩展点击区域.................................................................................... 136

上下拉动滚动条时卡顿、慢........................................................................136

长时间按住页面出现闪退............................................................................136

ios和android下触摸元素时出现半透明灰色遮罩.................................... 136

active兼容处理 即 伪类：active失效....................................................... 137

webkitmask兼容处理....................................................................................137

transiton闪屏..................................................................................................138

圆角bug..........................................................................................................138

3| 前端进阶...................................................................................................................... 138

3.1| 前端工程化........................................................................................................... 138













Babel的原理是什么?....................................................................................138

如何写一个babel插件?................................................................................139

你的git工作流是怎样的?............................................................................143

rebase 与 merge的区别?..............................................................................148

gitreset、gitrevert 和 gitcheckout 有什么区别........................................149

webpack和gulp区别（模块化与流的区别）............................................ 150

3.2|Vue 框架................................................................................................................151





有使用过Vue吗？说说你对Vue的理解...................................................151

说说Vue的优缺点........................................................................................151

8

























Vue和React有什么不同？使用场景分别是什么？................................. 151

什么是虚拟DOM？.......................................................................................152

请描述下vue的生命周期是什么？.............................................................152

vue如何监听键盘事件？..............................................................................155

watch怎么深度监听对象变化......................................................................156

删除数组用delete和Vue.delete有什么区别？......................................... 156

watch和计算属性有什么区别？..................................................................156

Vue双向绑定原理.........................................................................................157

v-model是什么？有什么用呢？..................................................................157

axios是什么？怎样使用它？怎么解决跨域的问题？...............................157

在vue项目中如何引入第三方库（比如jQuery）？有哪些方法可以做到？

157

















说说VueReactangularjsjquery的区别.......................................................157

Vue3.0 里为什么要用 ProxyAPI 替代 definePropertyAPI？..................158

Vue3.0 编译做了哪些优化？.......................................................................158

Vue3.0新特性 ——CompositionAPI 与 React.js 中 Hooks的异同点..159

Vue3.0是如何变得更快的？（底层，源码）........................................... 161

vue要做权限管理该怎么做？如果控制到按钮级别的权限怎么做？.....162

vue在created和mounted这两个生命周期中请求数据有什么区别呢？162

说说你对proxy的理解..................................................................................162

3.3|React 框架............................................................................................................. 162





































angularJs和React区别................................................................................. 162

redux中间件...................................................................................................163

redux有什么缺点...........................................................................................163

React组件的划分业务组件技术组件？......................................................163

React生命周期函数...................................................................................... 163

React性能优化是哪个周期函数？..............................................................164

为什么虚拟dom会提高性能?......................................................................164

diff算法?........................................................................................................165

React性能优化方案...................................................................................... 165

简述flux 思想............................................................................................... 165

React项目用过什么脚手架？Mern?Yeoman?............................................165

你了解React吗？......................................................................................... 166

React解决了什么问题？..............................................................................166

React的协议？.............................................................................................. 166

了解shouldComponentUpdate吗？...............................................................166

React 的工作原理？.....................................................................................166

使用 React 有何优点？...............................................................................166

展示组件(Presentational component)和容器组件(Container component)之间

有何不同？.............................................................................................................167

类组件(Class component)和函数式组件(Functional component)之间有何不

同？.........................................................................................................................167







(组件的)状态(state)和属性(props)之间有何不同？.....................................167

应该在 React 组件的何处发起 Ajax 请求？........................................... 168

9











在 React 中，refs 的作用是什么？...........................................................168

何为高阶组件(higherordercomponent)？.....................................................168

使用箭头函数(arrowfunctions)的优点是什么？..........................................168

为什么建议传递给 setState 的参数是一个 callback 而不是一个对象？

168









除了在构造函数中绑定 this，还有其它方式吗？....................................169

怎么阻止组件的渲染？................................................................................ 169

当渲染一个列表时，何为 key？设置 key 的目的是什么？...................169

何为 JSX ？.................................................................................................. 169

3.4|Angular 框架..........................................................................................................170











Angular中组件之间通信的方式.................................................................. 170

Angualr的八大组成部分并简单描述.......................................................... 170

Angular中常见的生命周期的钩子函数?.................................................... 170

Angular中路由的工作原理...........................................................................171

解释rjx在Angular中的使用场景...............................................................171

10



1| 前端基础

1.1|HTTP/HTML/浏览器

 说一下http和https

参考回答：

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

务器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，来确保数组的传输，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：

Https协议需要ca证书，费用较高。

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。

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

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服务器通信时有以下几个步骤，如图所示。

客户使用httpsurl访问服务器，则要求web 服务器建立ssl链接。

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。

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。

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。

web服务器通过自己的私钥解密出会话密钥。

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；

11



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。

 tcp三次握手，一句话概括

参考回答：

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：

<imgwidth="487"alt="2018-07-1034211"src="https://user-images.githubusercontent.com/

17233651/42496289-1c6d668a-8458-11e8-98b3-65db50f64d48.png">

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确认我们

再看看每次握手的作用：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

 TCP和UDP的区别

参考回答：

（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。

（6）TCP是面向连接的可靠性传输，而UDP是不可靠的。

 WebSocket的实现和应用

参考回答：

(1)什么是WebSocket?

12



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/chatHTTP/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：

1Upgrade:webSocket

2Connection:Upgrade

告诉服务器发送的是websocket

1Sec-WebSocket-Key:x3JJHMbDL1EzLkh9GBhXDw==

2Sec-WebSocket-Protocol:chat,superchat

3Sec-WebSocket-Version:13

 HTTP请求的方式，HEAD方式

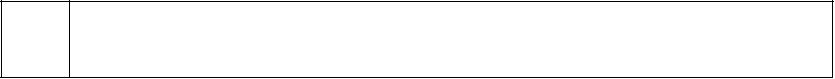
参考回答：

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。

 一个图片url访问后直接下载怎样实现？

13



参考回答：

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

 说一下webQuality（无障碍）

参考回答：

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：

<imgsrc="person.jpg" alt="thisisaperson"/>

有时候浏览器会无法显示图像。具体的原因有：

用户关闭了图像显示

浏览器是不支持图形显示的迷你浏览器

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。

 几个很实用的BOM属性对象方法?

参考回答：

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

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

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

14



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

URLlocation.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

 说一下HTML5dragapi

参考回答：

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

 说一下http2.0

参考回答：

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

 补充400和401、403状态码

参考回答：

(1)400状态码：请求无效

产生原因：

15



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

 fetch发送2次请求的原因

参考回答：

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。

 Cookie、sessionStorage、localStorage的区别

参考回答：

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：

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。

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

16



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。

 说一下webworker

参考回答：

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。webworker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webworker：

检测浏览器对于webworker的支持性

创建webworker文件（js，回传函数等）

创建webworker对象

 对HTML语义化标签的理解

参考回答：

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。

 iframe是什么？有什么缺点？

参考回答：

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：

会阻塞主页面的onload事件

搜索引擎无法解读这种页面，不利于SEO

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。

 Doctype作用?严格模式与混杂模式如何区分？它们有何意义?

参考回答：

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。

17



 Cookie如何防范XSS攻击

参考回答：

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>.....

 Cookie和session的区别

参考回答：

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。

 一句话概括RESTFUL

参考回答：

就是用URL定位资源，用HTTP描述操作。

 讲讲viewport和移动端布局

参考回答：

响应式布局的常用解决方案对比(媒体查询、百分比、rem和vw/vh）

 click在ios上有300ms延迟，原因及如何解决？

参考回答：

(1)粗暴型，禁用缩放

<metaname="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

 addEventListener参数

参考回答：

addEventListener(event,function,useCapture)

18



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。

 cookiesessionStoragelocalStorage区别

参考回答：

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

 cookiesession区别

参考回答：

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。

 介绍知道的http返回的状态码

参考回答：

100 Continue 继续。客户端应继续其请求

101 SwitchingProtocols 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-AuthoritativeInformation 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 NoContent 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

19



205 ResetContent 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 PartialContent 部分内容。服务器成功处理了部分GET请求

300 MultipleChoices 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 MovedPermanently 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 SeeOther 查看其它地址。与301类似。使用GET和POST请求查看

304 NotModified 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 UseProxy 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 TemporaryRedirect 临时重定向。与302类似。使用GET请求重定向

400 BadRequest 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 PaymentRequired 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 NotFound 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 MethodNotAllowed 客户端请求中的方法被禁止

406 NotAcceptable 服务器无法根据客户端请求的内容特性完成请求

407 ProxyAuthenticationRequired 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 RequestTime-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 LengthRequired 服务器无法处理客户端发送的不带Content-Length的请求信息

412 PreconditionFailed 客户端请求信息的先决条件错误

413 RequestEntityTooLarge 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-URITooLarge 请求的URI过长（URI通常为网址），服务器无法处理

415 UnsupportedMediaType 服务器无法处理请求附带的媒体格式

416 Requestedrangenotsatisfiable 客户端请求的范围无效

417 ExpectationFailed 服务器无法满足Expect的请求头信息

500 InternalServerError 服务器内部错误，无法完成请求

501 NotImplemented 服务器不支持请求的功能，无法完成请求

502 BadGateway 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

20



503 ServiceUnavailable 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 GatewayTime-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 HTTPVersionnotsupported 服务器不支持请求的HTTP协议的版本，无法完成处

理

 http常用请求头

参考回答：

协议头

Accept

说明

可接受的响应内容类型（Content-Types）。

可接受的字符集

Accept-Charset

Accept-Encoding

Accept-Language

Accept-Datetime

可接受的响应内容的编码方式。

可接受的响应内容语言列表。

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。

21



Connection

Cookie

客户端（浏览器）想要优先使用的连接类型

由之前服务器通过Set-Cookie（见下文）设置的一个HTTP协议Cookie

以8进制表示的请求体的长度

Content-Length

Content-MD5

Content-Type

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

请求体的MIME类型 （用于POST和PUT请求中）

Date

Expect

From

Host

发送该消息的日期和时间（以RFC7231中定义的"HTTP日期"格式

来发送）

表示客户端要求服务器做出特定的行为

发起此请求的用户的邮件地址

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。

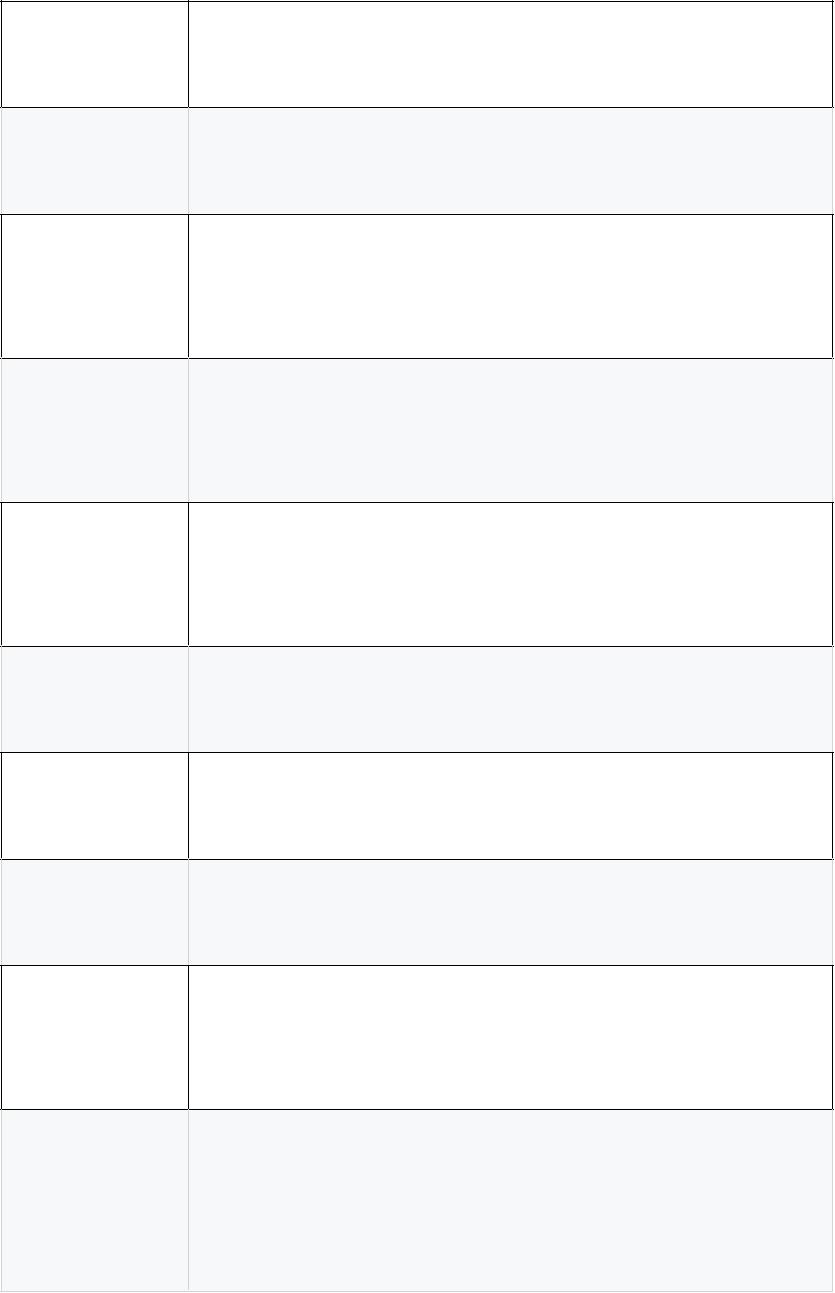
If-Match

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。

22



If-Modified-Since

If-None-Match

If-Range

允许在对应的资源未被修改的情况下返回304未修改

允许在对应的内容未被修改的情况下返回304未修改（ 304Not

Modified ），参考 超文本传输协议 的实体标记

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。

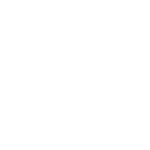
Proxy-Authorization

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。

23



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

Upgrade

Via

浏览器的身份标识字符串

要求服务器升级到一个高版本协议。

告诉服务器，这个请求是由哪些代理发出的。

一个一般性的警告，表示在实体内容体中可能存在错误。

Warning

 强，协商缓存

参考回答：

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

从缓存取

状态码

发送请求到服务器

否，直接从缓存取

强缓存

200（fromcache）

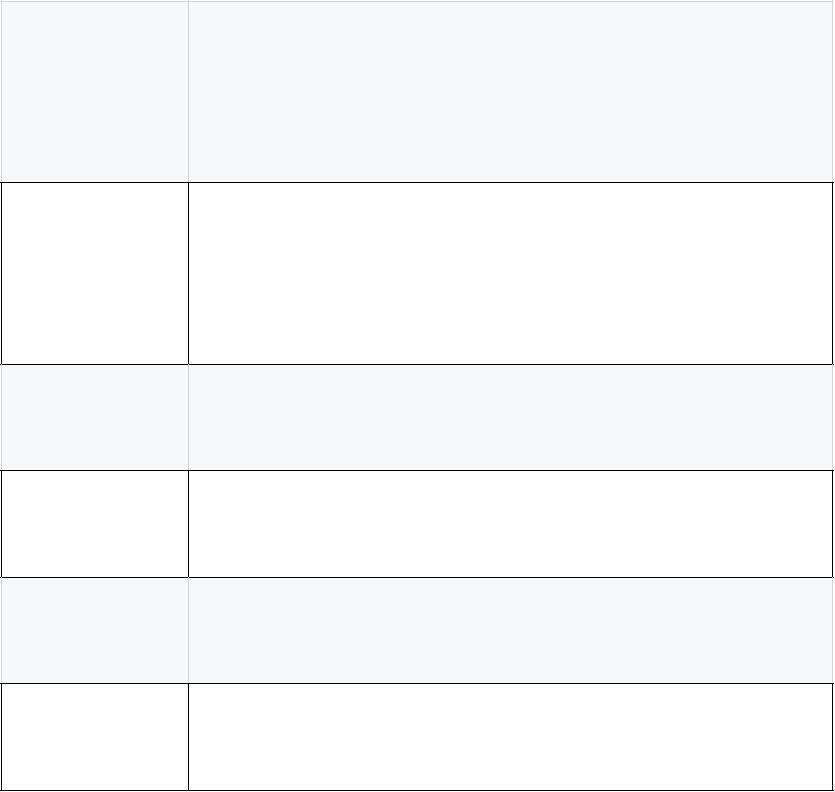
协商缓存

从缓存取

304（notmodified） 是，通过服务器来告知缓存是否可

用

24



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

 讲讲304

参考回答：

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。

 强缓存、协商缓存什么时候用哪个

参考回答：

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。

25



参考https://segmentfault.com/a/1190000008956069

 前端优化

参考回答：

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。

 GET和POST的区别

参考回答：

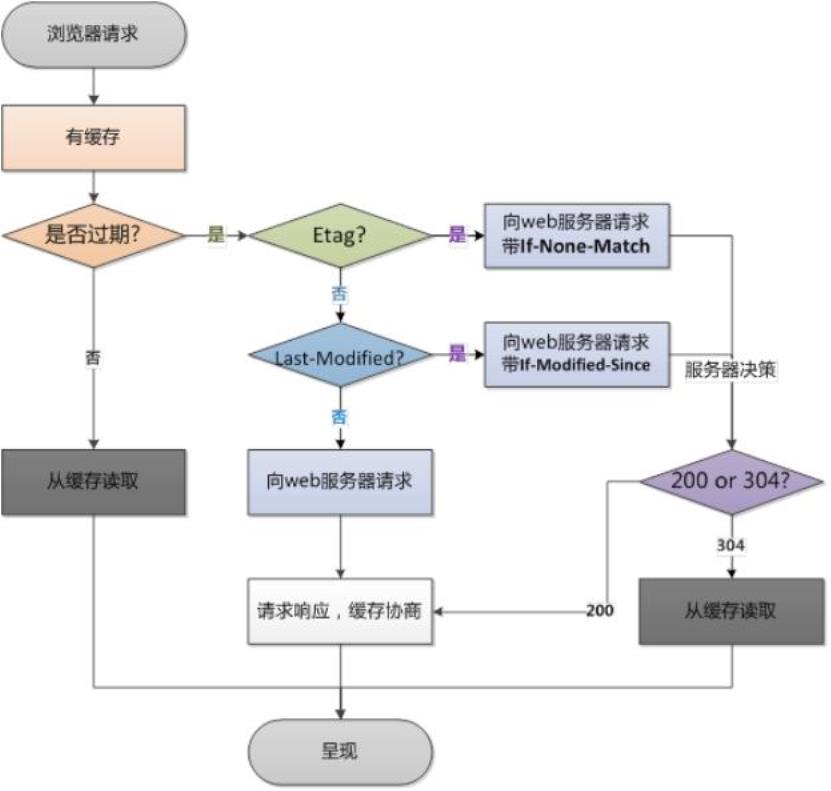
get参数通过url传递，post放在requestbody中。

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

26



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。

 301和302的区别

参考回答：

301MovedPermanently 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。

302Found 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。

 HTTP支持的方法

参考回答：

GET,POST,HEAD,OPTIONS,PUT,DELETE,TRACE,CONNECT

 如何画一个三角形

参考回答：

三角形原理：边框的均分原理

div{

width:0px;

height:0px;

border-top:10pxsolidred;

border-right:10pxsolidtransparent;

border-bottom:10pxsolidtransparent;

border-left:10pxsolidtransparent;

}

 状态码304和200

27



参考回答：

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。即客户端和服务器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。

 说一下浏览器缓存

参考回答：

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

 HTML5新增的元素

参考回答：

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，emial,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vedio，另外还

有地理定位，canvas画布，拖放，多线程编程的webworker和websocket协议。

 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？

参考回答：

这是一个必考的面试问题，

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，

28



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。

 cookie和session的区别，localstorage和sessionstorage的区别

参考回答：

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。

 常见的HTTP的头部

29



参考回答：

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，比如date表示报文创建时间，

请求首部就是请求报文中独有的，如cookie，和缓存相关的如if-Modified-Since

响应首部就是响应报文中独有的，如set-cookie，和重定向相关的location，

实体首部用来描述实体部分，如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。

 HTTP2.0 的特性

参考回答：

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。

 cache-control的值有哪些

参考回答：

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。

 浏览器在生成页面的时候，会生成那两颗树？

参考回答：

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。

 csrf和xss的网络攻击及防范

参考回答：

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ｔａｂ页面访问量攻击者用来制造

30



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，防御方式的话：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，防御的话为

cookie设置httpOnly属性，对用户的输入进行检查，进行特殊字符过滤。

 怎么看网站的性能如何

参考回答：

检测页面加载时间一般有两种方式，一种是被动去测：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，另一种主动监测

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。

 介绍HTTP协议(特征)

参考回答：

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(NextGenerationofHTTP)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。

 输入URL到页面加载显示完成发生了什么?

参考回答：

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

 说一下对Cookie和Session的认知，Cookie有哪些限制？

31



参考回答：

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。

 描述一下XSS和CRSF攻击？防御方法？

参考回答：

XSS, 即为（CrossSiteScripting）, 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（CrossSiteRequestForgery，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：验证 HTTPReferer 字段；在请求地址中添加 token 并

验证；在 HTTP 头中自定义属性并验证。

 知道304吗，什么时候用304？

参考回答：

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。

 具体有哪些请求头是跟缓存相关的

参考回答：

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

 cookie和session的区别

32



参考回答：

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。

 cookie有哪些字段可以设置

参考回答：

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

 cookie有哪些编码方式？

参考回答：

encodeURI（）

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

参考回答：

33



还有localStorage，sessionStorage，indexdDB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，如回话标识。

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。

 浏览器输入网址到页面渲染全过程

参考回答：

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

 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？

参考回答：

html5：

1）标签增删

8个语义元素 headersectionfooterasidenavmainarticlefigure

内容元素mark高亮 progress进度

新的表单控件calanderdatetimeemailurlsearch

新的input类型 colordatedatetimedatetime-localemail

移除过时标签bigfontframeframeset

2）canvas绘图，支持内联SVG。支持MathML

3）多媒体audiovideosourceembedtrack

4）本地离线存储，把需要离线存储在本地的文件列在一个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

动画如animation等。 参考

https://www.cnblogs.com/xkweb/p/5862612.html

34



 http常见的请求方法

参考回答：

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

 get和post的区别

参考回答：

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httpheader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100continue，浏览器再发送data，服

务器响应200ok（返回数据）。

 说说302，301，304的状态码

参考回答：

301 MovedPermanently 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 NotModified 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

 web性能优化

参考回答：

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。

 浏览器缓存机制

参考回答：

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。

35



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

 post和get区别

参考回答：

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httpheader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100continue，浏览器再发送data，服

务器响应200ok（返回数据）。

1.2|CSS

 说一下css盒模型

参考回答：

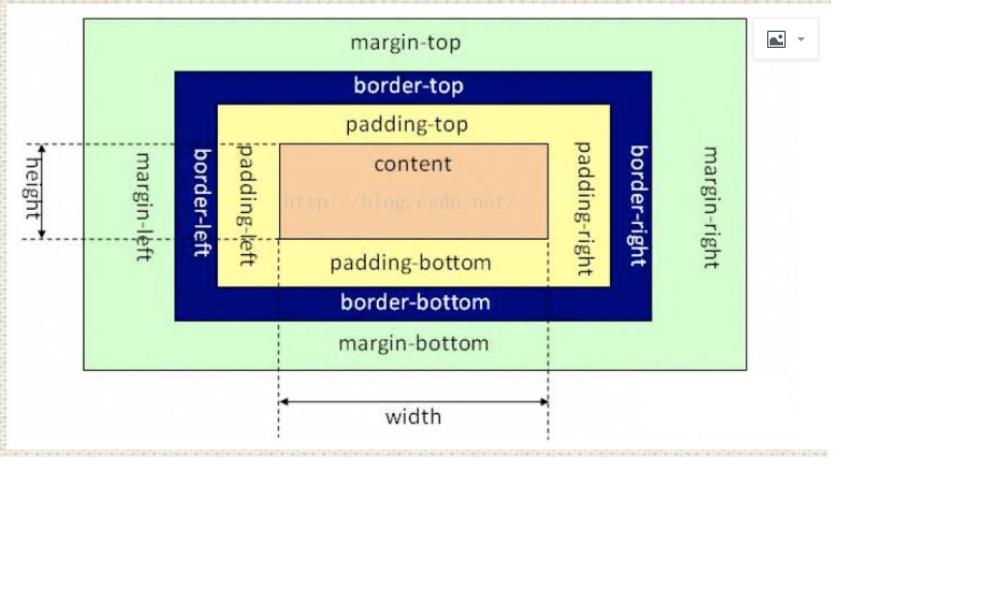
简介：就是用来装页面上的元素的矩形区域。CSS中的盒子模型包括IE盒子模型和标

准的W3C盒子模型。

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

标准盒子模型：

36



IE盒子模型：

区别：从图中我们可以看出，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。

 画一条0.5px的线

参考回答：

采用metaviewport的方式

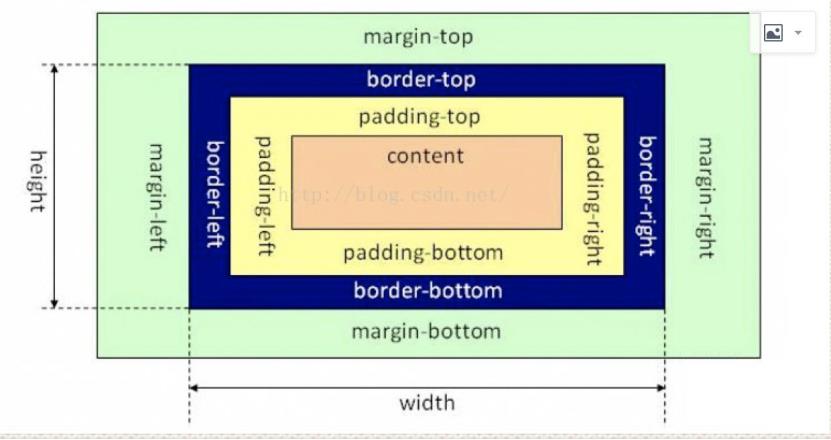
<metaname="viewport"content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no"/>

采用border-image的方式

采用transform:scale()的方式

 link标签和import标签的区别

37



参考回答：

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。

 transition和animation的区别

参考回答：

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。

 Flex布局

参考回答：

http://www.ruanyifeng.com/blog/2015/07/flex-grammar.html?utm\_source=tuicool（语法篇）

http://www.ruanyifeng.com/blog/2015/07/flex-examples.html（实例篇）是FlexibleBox

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，竖直轴线方向

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

38



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flow-shrink为0，则为不缩小

flex-basis属性：定义了在分配多余的空间，项目占据的空间。

flex：是flex-grow和flex-shrink、flex-basis的简写，默认值为01auto。

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

 BFC（块级格式化上下文，用于清楚浮动，防止margin重叠等）

参考回答：

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floatbox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：

根元素

float不为none的元素

position为fixed和absolute的元素

display为inline-block、table-cell、table-caption，flex，inline-flex的元素

overflow不为visible的元素

 垂直居中的方法

参考回答：

(1)margin:auto法

css:

div{

width:400px;

height:400px;

position:relative;

border:1pxsolid#465468;

}

img{

position:absolute;

margin:auto;

top:0;

left:0;

right:0;

bottom:0;

}

html:

<div>

39



<imgsrc="mm.jpg">

</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:2pxsolid#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marin-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:3pxsolid#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;

40



1css:

2.container{

3width:300px;

4height:200px;

5border:3pxsolid#546461;

6display:-webkit-flex;

7display:flex;

8-webkit-align-items:center;

9align-items:center;

10-webkit-justify-content:center;

11justify-content:center;

12}

13.inner{

14border:3pxsolid#458761;

15padding:20px;

16}

 关于JS动画和css3动画的差异性

参考回答：

渲染线程分为mainthread和compositorthread，如果css动画只改变transform和opacity，

这时整个CSS动画得以在compositortrhead完成（而JS动画则会在mainthread执行，然

后出发compositorthread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：

功能涵盖面，JS比CSS大

实现/重构难度不一，CSS3比JS更加简单，性能跳优方向固定

对帧速表现不好的低版本浏览器，css3可以做到自然降级

css动画有天然事件支持

css3有兼容性问题

 说一下块元素和行元素

参考回答：

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

度

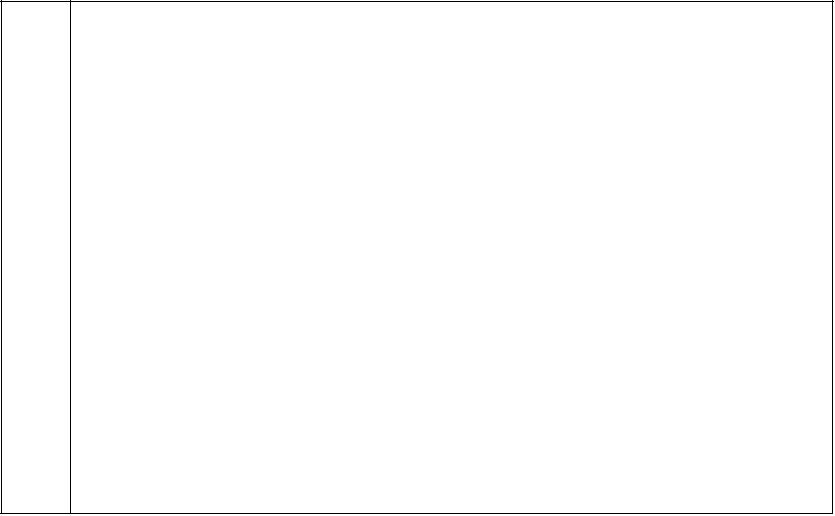
行元素：不会独占一行，width和height会失效，并且在垂直方向的padding和margin

会失

效。

 多行元素的文本省略号

41



参考回答：

1display:-webkit-box

2-webkit-box-orient:vertical

3-webkit-line-clamp:3

4overflow:hidden

 visibility=hidden,opacity=0，display:none

参考回答：

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。

 双边距重叠问题（外边距折叠）

参考回答：

多个相邻（兄弟或者父子关系）普通流的块元素垂直方向marigin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。

 position属性比较

参考回答：

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：

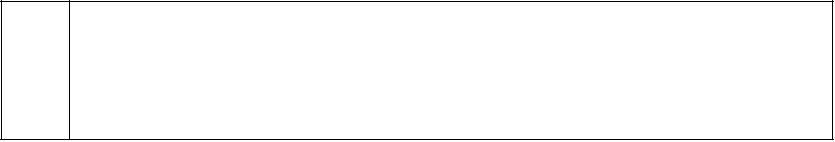
绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那

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：

42



元素先按照普通文档流定位，然后相对于该元素在流中的flowroot（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。

 浮动清除

参考回答：

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<divclass="clear"></div>，并在CSS中赋

予.clear{clear:both;}属性即可清理浮动。亦可使用<brclass="clear"/>或<hrclass="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

参考https://www.cnblogs.com/ForEvErNoME/p/3383539.html

 css3新特性

参考回答：

开放题。CSS3边框如border-radius，box-shadow等；CSS3背景如background-size，

background-origin等；CSS3动画如animation等。

参考https://www.cnblogs.com/xkweb/p/5862612.html

 CSS选择器有哪些，优先级呢

43



参考回答：

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

 清除浮动的方法，能讲讲吗

参考回答：

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<divclass="clear"></div>，并在CSS中赋

予.clear{clear:both;}属性即可清理浮动。亦可使用<brclass="clear"/>或<hrclass="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

参考https://www.cnblogs.com/ForEvErNoME/p/3383539.html

 怎么样让一个元素消失，讲讲

参考回答：

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

 介绍一下盒模型

44



参考回答：

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

 position相关属性

参考回答：

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flowroot（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。

 css动画如何实现

参考回答：

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。

45



 如何实现图片在某个容器中居中的？

参考回答：

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

 如何实现元素的垂直居中

参考回答：

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;

 CSS3中对溢出的处理

参考回答：

cnkOhu

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。

 float的元素，display是什么

参考回答：

display为block

 隐藏页面中某个元素的方法

参考回答：

display:none;visibility:hidden;opacity:0;position移到外部，z-index涂层遮盖等等

 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

参考回答：

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

46



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

 什么是BFC

参考回答：

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

 calc属性

参考回答：

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)；

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

参考回答：

对于行内块级元素，

47



1、父级元素设置text-alig：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

6、视窗居中，vh为视口单位，50vh即是视口高度的50/100，设置margin：50vhauto0，

transform：translate(-50%)

 display：table和本身的table有什么区别

参考回答：

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

 position属性的值有哪些及其区别

参考回答：

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,buttom,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fiexd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

 z-index的定位方法

参考回答：

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

48



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

 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？

参考回答：

可以更改父元素的color

 对CSS的新属性有了解过的吗？

参考回答：

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，rbga等，在字体方

面允许嵌入字体和设置字体阴影，最后还有媒体查讯等

 用的最多的css属性是啥？

参考回答：

用的目前来说最多的是flex属性，灵活但是兼容性方面不强。

 line-height和height的区别

参考回答：

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。

 设置一个元素的背景颜色，背景颜色会填充哪些区域？

参考回答：

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

 知道属性选择器和伪类选择器的优先级吗

参考回答：

属性选择器和伪类选择器优先级相同

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

49



参考回答：

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

 用css实现一个硬币旋转的效果

参考回答：

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:spin2.5slinearinfinite;

}

.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;

}

50



@keyframesspin{

0%{

transform:rotateY(0deg);

}

100%{

transform:rotateY(360deg);

}

}

 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

参考回答：

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：

添加或者删除可见的DOM元素，

元素尺寸位置的改变

浏览器页面初始化，

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：

不在布局信息改变时做DOM查询，

使用csstext,className一次性改变属性

使用fragment

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

 CSS画正方体，三角形

参考回答：

画三角形

#triangle02{

width:0;

height:0;

border-top:50pxsolidblue;

border-right:50pxsolidred;

border-bottom:50pxsolidgreen;

border-left:50pxsolidyellow;

}

画正方体：

<!DOCTYPEhtml>

<htmllang="en">

<head>

<metacharset="UTF-8">

51



<title>perspective</title>

<style>

.wrapper{

width:50%;

float:left;

}

.cube{

font-size:4em;

width:2em;

margin:1.5emauto;

transform-style:preserve-3d;

transform:rotateX(-35deg)rotateY(30deg);

}

.side{

position:absolute;

width:2em;

height:2em;

background:rgba(255,99,71,0.6);

border:1pxsolidrgba(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>

52



<divclass="wrapperw1">

<divclass="cube">

<divclass="sidefront">1</div>

<divclass="sideback">6</div>

<divclass="sideright">4</div>

<divclass="sideleft">3</div>

<divclass="sidetop">5</div>

<divclass="sidebottom">2</div>

</div>

</div>

<divclass="wrapperw2">

<divclass="cube">

<divclass="sidefront">1</div>

<divclass="sideback">6</div>

<divclass="sideright">4</div>

<divclass="sideleft">3</div>

<divclass="sidetop">5</div>

<divclass="sidebottom">2</div>

</div>

</div>

</body>

</html>

 overflow的原理

参考回答：

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，Ablockformattingcontext

isapartofavisualCSSrenderingofaWebpage.Itistheregioninwhichthelayoutofblock

boxesoccursandinwhichfloatsinteractwitheachother.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。

 清除浮动的方法

参考回答：

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

53



 box-sizing的语法和基本用处

参考回答：

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

 使元素消失的方法有哪些？

参考回答：

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。

 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。

参考回答：

margin的外边缘

 说说盒子模型

参考回答：

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

 display

参考回答：

54



https://developer.mozilla.org/zh-CN/docs/Web/CSS/display

 怎么隐藏一个元素

参考回答：

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。

 display:none和visibilty:hidden的区别

参考回答：

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。

 相对布局和绝对布局，position:relative和obsolute。

参考回答：

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。

 flex布局

参考回答：

flex 是 FlexibleBox 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；

55



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

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属性定义了在分配多余空间之前，项目占据的主轴空间（mainsize）。

flex属性是flex-grow,flex-shrink 和 flex-basis的简写，默认值为01auto。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stretch。

参考http://www.ruanyifeng.com/blog/2015/07/flex-grammar.html

 block、inline、inline-block的区别。

参考回答：

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。

 css的常用选择器

参考回答：

id选择器，类选择器，伪类选择器等

 css布局

56



参考回答：

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postion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。

 css定位

参考回答：

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flowroot（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。

 relative定位规则

57



参考回答：

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。

 垂直居中

参考回答：

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;

 css预处理器有什么

参考回答：

less，sass等

1.3|JavaScript

 get请求传参长度的误区

参考回答：

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

 补充get和post请求在缓存方面的区别

参考回答：

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：

58



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。

 说一下闭包

参考回答：

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。

 说一下类的创建和继承

参考回答：

（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

functionAnimal(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，实力化生成对象后，有方法和属性。

（2）类的继承——原型链继承

--原型链继承

functionCat(){}

Cat.prototype=newAnimal();

Cat.prototype.name='cat';

// TestCode

varcat=newCat();

console.log(cat.name);

console.log(cat.eat('fish'));

console.log(cat.sleep());

console.log(catinstanceofAnimal);//true

console.log(catinstanceofCat);//true

59



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）

functionCat(name){

Animal.call(this);

this.name=name||'Tom';

}

//TestCode

varcat=newCat();

console.log(cat.name);

console.log(cat.sleep());

console.log(catinstanceofAnimal);//false

console.log(catinstanceofCat);//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体。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

functionCat(name){

Animal.call(this);

this.name=name||'Tom';

}

Cat.prototype=newAnimal();

Cat.prototype.constructor=Cat;

//TestCode

varcat=newCat();

console.log(cat.name);

console.log(cat.sleep());

console.log(catinstanceofAnimal);//true

console.log(catinstanceofCat);//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

functionCat(name){

Animal.call(this);

this.name=name||'Tom';

}

60



(function(){

// 创建一个没有实例方法的类

varSuper=function(){};

Super.prototype=Animal.prototype;

//将实例作为子类的原型

Cat.prototype=newSuper();

})();

//TestCode

varcat=newCat();

console.log(cat.name);

console.log(cat.sleep());

console.log(catinstanceofAnimal);//true

console.log(catinstanceofCat);//true

较为推荐

 如何解决异步回调地狱

参考回答：

1promise、generator、async/await

 说说前端中的事件流

参考回答：

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。

想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。

 如何让事件先冒泡后捕获

参考回答：

61



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。

 说一下事件委托

参考回答：

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。

 说一下图片的懒加载和预加载

参考回答：

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。

 mouseover和mouseenter的区别

参考回答：

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

 JS的new操作符做了哪些事情

参考回答：

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。

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

62



参考回答：

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。

 JS的各种位置，比如clientHeight,scrollHeight,offsetHeight,以及scrollTop,

offsetTop,clientTop的区别？

参考回答：

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。

 JS拖拽功能的实现

参考回答：

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offetLeft.

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

 异步加载JS的方法

参考回答：

63



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

 Ajax解决浏览器缓存问题

参考回答：

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后面加上时间搓："nowtime="+newDate().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。

 JS的节流和防抖

http://www.cnblogs.com/coco1s/p/5499469.html

 JS中的垃圾回收机制

参考回答：

必要性：由于字符串、对象和数组没有固定大小，所有当他们的大小已知时，才能对他

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

便他们能够被再用，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（GarbageCollection）。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：

64



vara="helloworld";

varb="world";

vara=b;

//这时，会释放掉"helloworld"，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：

functionproblem(){

varobjA=newObject();

varobjB=newObject();

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：

varelement=document.getElementById（’‘）；

varmyObj=newObject();

myObj.element=element;

element.someObject=myObj;

这样就不会有垃圾回收的过程。

 eval是做什么的

参考回答：

65



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）

 如何理解前端模块化

参考回答：

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

 说一下CommonJS、AMD和CMD

参考回答：

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，modules.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(){

varname='Byron';

functionprintName(){

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)；

66



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。

 对象深度克隆的简单实现

参考回答：

functiondeepClone(obj){

varnewObj=objinstanceofArray?[]:{};

for(variteminobj){

vartemple=typeofobj[item]=='object'?deepClone(obj[item]):obj[item];

newObj[item]=temple;

}

returnnewObj;

}

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。

 实现一个once函数，传入函数参数只执行一次

参考回答：

functionones(func){

vartag=true;

returnfunction(){

if(tag==true){

func.apply(null,arguments);

tag=false;

}

returnundefined

}

}

 将原生的ajax封装成promise

参考回答：

var myNewAjax=function(url){

returnnewPromise(function(resolve,reject){

varxhr=newXMLHttpRequest();

xhr.open('get',url);

xhr.send(data);

xhr.onreadystatechange=function(){

67



if(xhr.status==200&&readyState==4){

varjson=JSON.parse(xhr.responseText);

resolve(json)

}elseif(xhr.readyState==4&&xhr.status!=200){

reject('error');

}

}

})

}

 JS监听对象属性的改变

参考回答：

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=newProxy({}，{

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。

 如何实现一个私有变量，用getName方法可以访问，不能直接访问

参考回答：

(1)通过defineProperty来实现

obj={

name:yuxiaoliang,

getName:function(){

returnthis.name

}

}

object.defineProperty(obj,"name",{

//不可枚举不可配置

});

(2)通过函数的创建形式

functionproduct(){

68



varname='yuxiaoliang';

this.getName=function(){

returnname;

}

}

varobj=newproduct();

 ==和===、以及Object.is的区别

参考回答：

(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.js

主要的区别就是+0！=-0 而NaN==NaN

(相对比===和==的改进)

 setTimeout、setInterval和requestAnimationFrame之间的区别

参考回答：

http://www.cnblogs.com/xiaohuochai/p/5777186.html

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

（3）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。

69



 实现一个两列等高布局，讲讲思路

参考回答：

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;

 自己实现一个bind函数

参考回答：

原理：通过apply或者call方法来实现。

(1)初始版本

Function.prototype.bind=function(obj,arg){

vararg=Array.prototype.slice.call(arguments,1);

varcontext=this;

returnfunction(newArg){

arg=arg.concat(Array.prototype.slice.call(newArg));

returncontext.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){

vararg=Array.prototype.slice.call(arguments,1);

varcontext=this;

varbound=function(newArg){

arg=arg.concat(Array.prototype.slice.call(newArg));

returncontext.apply(obj,arg);

}

varF=function(){}

//这里需要一个寄生组合继承

F.prototype=context.prototype;

bound.prototype=newF();

returnbound;

}

 用setTimeout来实现setInterval

参考回答：

(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

70



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。

<imgwidth="626"alt="2018-07-10113643"src="https://user-images.githubusercontent.co

m/17233651/42487876-92656f2c-8435-11e8-8a5f-0a97918039da.png">

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。

functionsay(){

//something

setTimeout(say,200);

}

setTimeout(say,200)

或者

setTimeout(function(){

//dosomething

setTimeout(arguments.callee,200);

},200);

 JS怎么控制一次加载一张图片，加载完后再加载下一张

参考回答：

(1)方法1

<scripttype="text/javascript">

varobj=newImage();

obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";

obj.onload=function(){

alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);

document.getElementById("mypic").innnerHTML="<imgsrc='"+this.src+"'/>";

}

</script>

<divid="mypic">onloading……</div>

(2)方法2

<scripttype="text/javascript">

varobj=newImage();

obj.src="http://www.phpernote.com/uploadfiles/editor/201107240502201179.jpg";

obj.onreadystatechange=function(){

if(this.readyState=="complete"){

71



alert('图片的宽度为：'+obj.width+'；图片的高度为：'+obj.height);

document.getElementById("mypic").innnerHTML="<imgsrc='"+this.src+"'/>";

}

}

</script>

<divid="mypic">onloading……</div>

 代码的执行顺序

参考回答：

setTimeout(function(){console.log(1)},0);

newPromise(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

从promise、process.nextTick、setTimeout出发，谈谈EventLoop中的Jobqueue

 如何实现sleep的效果（es5或者es6）

参考回答：

(1)while循环的方式

functionsleep(ms){

varstart=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

functionsleep(ms){

vartemple=newPromise(

(resolve)=>{

console.log(111);setTimeout(resolve,ms)

});

returntemple

}

sleep(500).then(function(){

72



//console.log(222)

})

//先输出了111，延迟500ms后输出222

(3)通过async封装

functionsleep(ms){

returnnewPromise((resolve)=>setTimeout(resolve,ms));

}

asyncfunctiontest(){

vartemple=awaitsleep(1000);

console.log(1111)

returntemple

}

test();

//延迟1000ms输出了1111

(4).通过generate来实现

function\*sleep(ms){

yieldnewPromise(function(resolve,reject){

console.log(111);

setTimeout(resolve,ms);

})

}

sleep(500).next().value.then(function(){console.log(2222)})

 简单的实现一个promise

首先明确什么是promiseA+规范，参考规范的地址：primise

A+规范

实现一个完美符合Promise/A+规范的Promise

一般不会问的很详细，只要能写出上述文章中的v1.0版本的简单promise即可。

 Function.\_proto\_(getPrototypeOf)是什么？

参考回答：

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。

我们用图可以来明确这个关系：

73



<imgwidth="646"alt="2018-07-1023827"src="https://user-images.githubusercontent.com/

17233651/42493275-f55d0860-844e-11e8-983f-e04189a4f3d8.png">

 实现JS中所有对象的深度克隆（包装对象，Date对象，正则对象）

参考回答：

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

functiondeepClone(obj){

varnewObj=objinstanceofArray?[]:{};

for(variinobj){

newObj[i]=typeofobj[i]=='object'?

deepClone(obj[i]):obj[i];

}

returnnewObj;

}

这种方法可以实现一般对象和数组对象的克隆，比如：

vararr=[1,2,3];

varnewArr=deepClone(arr);

//newArr->[1,2,3]

varobj={

x:1,

y:2

}

varnewObj=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

varnum=newNumber(1);

typeofnum//"object"

varnewNum=deepClone(num);

//newNum-> {} 空对象

//String包装对象

varstr=newString("hello");

typeofstr//"object"

varnewStr=deepClone(str);

//newStr-> {0:'h',1:'e',2:'l',3:'l',4:'o'};

//Boolean包装对象

74



varbol=newBoolean(true);

typeofbol//"object"

varnewBol=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：

functionbaseClone(base){

returnbase.valueOf();

}

//Number

varnum=newNumber(1);

varnewNum=baseClone(num);

//newNum->1

//String

varstr=newString('hello');

varnewStr=baseClone(str);

//newStr->"hello"

//Boolean

varbol=newBoolean(true);

varnewBol=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(){

returnnewDate(this.valueOf());

}

vardate=newDate('2010');

varnewDate=date.clone();

//newDate-> FriJan01201008:00:00GMT+0800

对于正则对象RegExp：

RegExp.prototype.clone=function(){

varpattern=this.valueOf();

varflags='';

flags+=pattern.global?'g':'';

75



flags+=pattern.ignoreCase?'i':'';

flags+=pattern.multiline?'m':'';

returnnewRegExp(pattern.source,flags);

};

varreg=newRegExp('/111/');

varnewReg=reg.clone();

//newReg-> /\/111\//

 简单实现Node的Events模块

参考回答：

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：

varevents=require('events');

vareventEmitter=newevents.EventEmitter();

eventEmitter.on('say',function(name){

console.log('Hello',name);

})

eventEmitter.emit('say','Jonyyu');

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

functionEvents(){

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(vari=0;o<this.handles[eventName].length;i++){

this.handles[eventName][i](obj);

}

}

}

returnthis;

}

76



这样我们就定义了Events，现在我们可以开始来调用：

varevents=newEvents();

events.on('say',function(name){

console.log('Hello',nama)

});

events.emit('say','Jonyyu');

//结果就是通过emit调用之后，输出了Jonyyu

(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：

varevent1=newEvents();

varevent2=newEvents();

event1.on('say',function(){

console.log('Jonyevent1');

});

event2.on('say',function(){

console.log('Jonyevent2');

})

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'Jonyevent1''Jonyevent2'

 箭头函数中this指向举例

参考回答：

vara=11;

functiontest2(){

this.a=22;

letb=()=>{console.log(this.a)}

b();

}

varx=newtest2();

//输出22

定义时绑定。

 JS判断类型

参考回答：

判断方法：typeof()，instanceof，Object.prototype.toString.call()等



组

77



参考回答：

push()，pop()，shift()，unshift()，splice()，sort()，reverse()，map()等

 数组去重

参考回答：

法一：indexOf循环去重

法二：ES6Set去重；Array.from(newSet(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。

 闭包有什么用

参考回答：

（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，则默认的会添加到全

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。

 事件代理在捕获阶段的实际应用

参考回答：

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。

 去除字符串首尾空格

参考回答：

使用正则(^\s\*)|(\s\*$)即可

78



 性能优化

参考回答：

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

 来讲讲JS的闭包吧

参考回答：

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，则默认的会添加到全

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。

 能来讲讲JS的语言特性吗

参考回答：

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

79



 如何判断一个数组(讲到typeof差点掉坑里)

参考回答：

Object.prototype.call.toString()

instanceof

 你说到typeof，能不能加一个限制条件达到判断条件

参考回答：

typeof只能判断是object,可以判断一下是否拥有数组的方法

 JS实现跨域

参考回答：

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

参考https://segmentfault.com/a/1190000011145364

 JS基本数据类型

参考回答：

基本数据类型：undefined、null、number、boolean、string、symbol

 JS深度拷贝一个元素的具体实现

参考回答：

vardeepCopy=function(obj){

if(typeofobj!=='object')return;

varnewObj=objinstanceofArray?[]:{};

for(varkeyinobj){

if(obj.hasOwnProperty(key)){

80



newObj[key]=typeofobj[key]==='object'?deepCopy(obj[key]):obj[key];

}

}

returnnewObj;

}

 之前说了ES6set可以数组去重，是否还有数组去重的方法

参考回答：

法一：indexOf循环去重

法二：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。

 重排和重绘，讲讲看

参考回答：

重绘（repaint或redraw）：当盒子的位置、大小以及其他属性，例如颜色、字体大小等

都确定下来之后，浏览器便把这些原色都按照各自的特性绘制一遍，将内容呈现在页面

上。重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重

新绘制，使元素呈现新的外观。

触发重绘的条件：改变元素外观属性。如：color，background-color等。

注意：table及其内部元素可能需要多次计算才能确定好其在渲染树中节点的属性值，比

同等元素要多花两倍时间，这就是我们尽量避免使用table布局页面的原因之一。

重排（重构/回流/reflow）：当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，

隐藏等改变而需要重新构建, 这就称为回流(reflow)。每个页面至少需要一次回流，就是

在页面第一次加载的时候。

重绘和重排的关系：在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新

构造这部分渲染树，完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程称

为重绘。所以，重排必定会引发重绘，但重绘不一定会引发重排。

 JS的全排列

参考回答：

functionpermutate(str){

varresult=[];

if(str.length>1){

varleft=str[0];

varrest=str.slice(1,str.length);

varpreResult=permutate(rest);

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

for(varj=0;j<preResult[i].length;j++){

vartmp=preResult[i],slice(0,j)+left+preResult[i].slice(j,preResult[i].length);

81



result.push(tmp);

}

}

}elseif(str.length==1){

return[str];

}

returnresult;

}

 跨域的原理

参考回答：

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。

 不同数据类型的值的比较，是怎么转换的，有什么规则

参考回答：

 null==undefined为什么

82



参考回答：

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。

 this的指向哪几种

参考回答：

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。

 暂停死区

参考回答：

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”

 AngularJS双向绑定原理

参考回答：

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

83



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()）

 写一个深度拷贝

参考回答：

functionclone(obj){

varcopy;

switch(typeofobj){

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(vari=0;i<obj.length;i++){

copy.push(clone(obj[i]));

}

}else{

copy={};

for(varjinobj){

copy[j]=clone(obj[j]);

}

}

}

break;

default:

copy=obj;

break;

}

returncopy;

84



}

 简历中提到了requestAnimationFrame，请问是怎么使用的

参考回答：

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。

 有一个游戏叫做FlappyBird，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）

参考回答：

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=16.7ms，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）

 编写代码，满足以下条件：（1）Hero("37er");执行结果为 Hi!Thisis37er

（2）Hero("37er").kill(1).recover(30);执行结果为 Hi!Thisis37erKill1bug

Recover30bloods （3）Hero("37er").sleep(10).kill(2)执行结果为 Hi!Thisis

37er//等待10s后 Kill2bugs //注意为bugs （双斜线后的为提示信息，

不需要打印）

参考回答：

functionHero(name){

leto=newObject();

o.name=name;

o.time=0;

console.log("Hi!Thisis"+o.name);

85



o.kill=function(bugs){

if(bugs==1){

console.log("Kill"+(bugs)+"bug");

}else{

setTimeout(function(){

console.log("Kill"+(bugs)+"bugs");

},1000\*this.time);

}

returno;

};

o.recover=function(bloods){

console.log("Recover"+(bloods)+"bloods");

returno;

}

o.sleep=function(sleepTime){

o.time=sleepTime;

returno;

}

returno;

}

 什么是按需加载

参考回答：

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。

 说一下什么是virtualdom

参考回答：

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。VirtualDOM 本质上就是在 JS 和 DOM 之间做了一个缓存。

 webpack用来干什么的

参考回答：

webpack是一个现代 JavaScript 应用程序的静态模块打包器(modulebundler)。当

webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependencygraph)，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。

86



 ant-design优点和缺点

参考回答：

优点：组件非常全面，样式效果也都比较不错。

缺点：框架自定义程度低，默认UI风格修改困难。

 JS中继承实现的几种方式，

参考回答：

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，无法实现函数服用，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forin 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，然后通过将父

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

 写一个函数，第一秒打印1，第二秒打印2

参考回答：

两个方法，第一个是用let块级作用域

for(leti=0;i<5;i++){

setTimeout(function(){

console.log(i)

},1000\*i)

}

第二个方法闭包

for(vari=0;i<5;i++){

(function(i){

setTimeout(function(){

87



console.log(i)

},1000\*i)

})(i)

}

 Vue的生命周期

参考回答：

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newVue() 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

 简单介绍一下symbol

参考回答：

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，letid=symbol(“id”)

88



Symbl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。

 什么是事件监听

参考回答：

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。

 介绍一下promise，及其底层如何实现

参考回答：

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，

letpromise1=newPromise(function(resolve,reject){

setTimeout(function(){

resolve('ok')

},1000)

})

promise1.then(functionsuccess(val){

console.log(val)

})

最简单代码实现promise

classPromiseM{

constructor(process){

89



this.status='pending'

this.msg=''

process(this.resolve.bind(this),this.reject.bind(this))

returnthis

}

resolve(val){

this.status='fulfilled'

this.msg=val

}

reject(err){

this.status='rejected'

this.msg=err

}

then(fufilled,reject){

if(this.status==='fulfilled'){

fufilled(this.msg)

}

if(this.status==='rejected'){

reject(this.msg)

}

}

}

//测试代码

varmm=newPromiseM(function(resolve,reject){

resolve('123');

});

mm.then(function(success){

console.log(success);

},function(){

console.log('fail!');

});

 说说C++,Java，JavaScript这三种语言的区别

参考回答：

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。

90



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。

 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

参考回答：

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.protoytpe=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，

91



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对象都会有的一个属性，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：

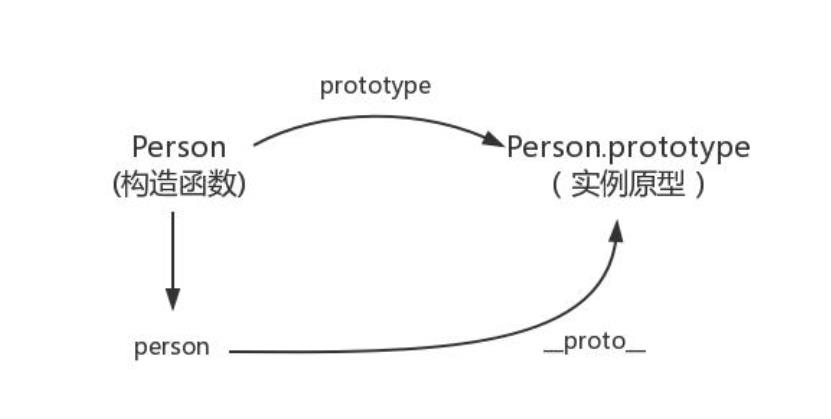
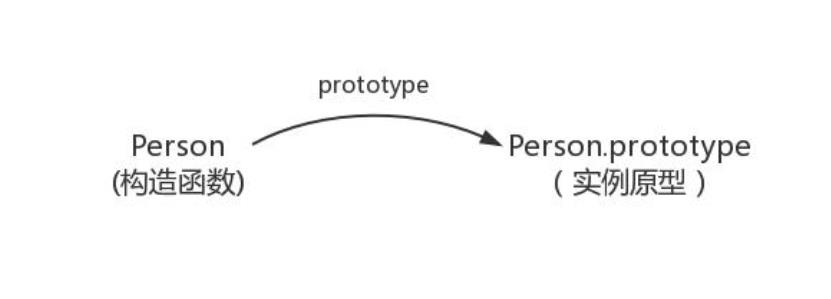
当读取实例的属性时，如果找不到，就会查找与对象相关的原型中的属性，如果还查不

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：

92

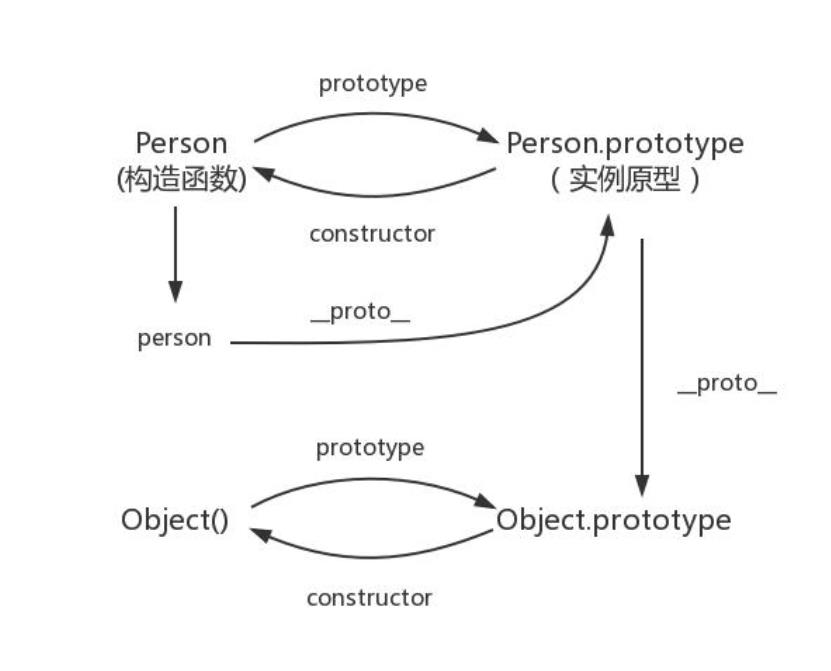


那么Object.prototype的原型呢，我们可以打印console.log(Object.prototype.\_\_proto\_\_===

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：

93



图中这条蓝色的线即是原型链，

最后补充三点：

constructor：

functionPerson(){

}

varperson=newPerson();

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.getPrototype(obj)

继承：

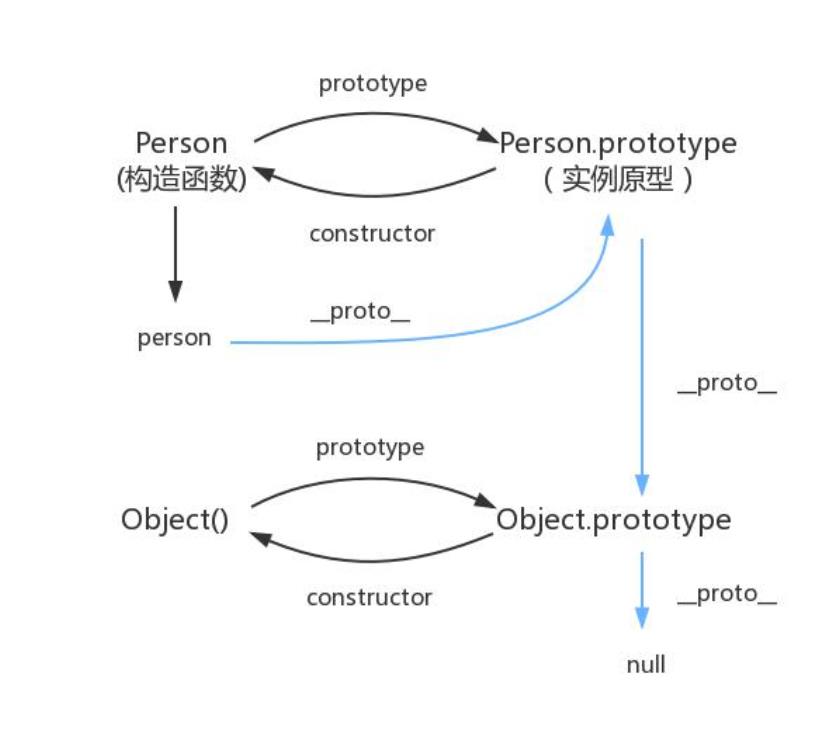
前面说到，每个对象都会从原型继承属性，但是引用《你不知道的JS》中的话，继承意

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。

94



 什么是js的闭包？有什么作用，用闭包写个单例模式

参考回答：

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：

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：

varSingleton=(function(){

varinstance;

varCreateSingleton=function(name){

this.name=name;

if(instance){

returninstance;

}

// 打印实例名字

this.getName();

//instance=this;

//returninstance;

returninstance=this;

}

// 获取实例的名字

CreateSingleton.prototype.getName=function(){

console.log(this.name)

}

returnCreateSingleton;

})();

// 创建实例对象1

vara=newSingleton('a');

// 创建实例对象2

varb=newSingleton('b');

console.log(a===b);

 promise+Generator+Async的使用

参考回答：

Promise

解决的问题:回调地狱

Promise规范:

95



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.

varpromise=newPromise(function(resolve,reject){

setTimeout(function(){

resolve('好哈哈哈哈');

});

});

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.

newPromise((resolve,reject)=>{

thrownewError('错误')

}).then(null,(err)=>{

console.log(err,1);//此处捕获

}).catch((err)=>{

console.log(err,2);

});

// 对比

newPromise((resolve,reject)=>{

thrownewError('错误')

}).then(null,null).catch((err)=>{

console.log(err,2);//此处捕获

});

// 错误示例

96



newPromise((resolve,reject)=>{

resolve('正常');

}).then((val)=>{

thrownewError('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=>{

returnp2()//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.

functionp1(val){

returnnewPromise((resolve,reject)=>{

if(val){

varlen=val.length;//传入null会发生错误,进入catch捕获错

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.

97



functionp1(val){

returnnewPromise((resolve,reject)=>{

val==1?resolve(1):reject()

})

};

functionp2(val){

returnnewPromise((resolve,reject)=>{

val==2?resolve(2):reject();

})

};

letpromimse=newPromise(function(resolve,reject){

resolve(1)

})

.then(function(data1){

returnp1(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){

returnp2(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(){

varo=1;

yieldo++;

yieldo++;

}

vargen=g();

console.log(gen.next());// Object{value:1,done:false}

varxxx=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:

98



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returnPromise.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。

lettimer=asyncfunctiontimer(){

returnnewPromise((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

letsayHi=asyncfunctionsayHi(){

lethi=await'helloworld';

returnhi; //等同于returnPromise.resolve(hi);

}

sayHi().then(result=>{

console.log(result);

});

 事件委托以及冒泡原理。

参考回答：

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

减少内存消耗，节约效率

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。

 写个函数，可以转化下划线命名到驼峰命名

99



参考回答：

publicstaticStringUnderlineToHump(Stringpara){

StringBuilderresult=newStringBuilder();

Stringa[]=para.split("\_");

for(Strings:a){

if(result.length()==0){

result.append(s.toLowerCase());

}else{

result.append(s.substring(0,1).toUpperCase());

result.append(s.substring(1).toLowerCase());

}

}

returnresult.toString();

}

}

 深浅拷贝的区别和实现

参考回答：

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：

vararr=['old',1,true,['old1','old2'],{old:1}]

varnew\_arr=JSON.parse(JSON.stringify(arr));

console.log(new\_arr);

原理是JOSN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

varshallowCopy=function(obj){

// 只拷贝对象

if(typeofobj!=='object')return;

100



// 根据obj的类型判断是新建一个数组还是对象

varnewObj=objinstanceofArray?[]:{};

// 遍历obj，并且判断是obj的属性才拷贝

for(varkeyinobj){

if(obj.hasOwnProperty(key)){

newObj[key]=obj[key];

}

}

returnnewObj;

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~

vardeepCopy=function(obj){

if(typeofobj!=='object')return;

varnewObj=objinstanceofArray?[]:{};

for(varkeyinobj){

if(obj.hasOwnProperty(key)){

newObj[key]=typeofobj[key]==='object'?deepCopy(obj[key]):obj[key];

}

}

returnnewObj;

}

 JS中string的startwith和indexof两种方法的区别

参考回答：

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。

 JS字符串转数字的方法

参考回答：

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

101



 letconstvar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

参考回答：

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。

 ES6箭头函数的特性

参考回答：

ES6 增加了箭头函数，基本语法为

letfunc=value=>value;

相当于

letfunc=function(value){

returnvalue;

};

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

 setTimeout和Promise的执行顺序

参考回答：

首先我们来看这样一道题：

setTimeout(function(){

console.log(1)

},0);

newPromise(function(resolve,reject){

console.log(2)

for(vari=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2103541

要先弄清楚settimeout（fun,0）何时执行，promise何时执行，then何时执行

102



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

 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

参考回答：

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.stopImmediatePropagetion，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

 平时是怎么调试JS的

参考回答：

一般用Chrome自带的控制台

 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

103



参考回答：

JS的基本数据类型有字符串，数字，布尔，数组，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，引用数据类型同事保存在栈区和堆区

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：

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，这个变量会转变成为一个

全局变量，

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还没有定义，因此

undefined==null返回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：

varvalue=1;

functionfoo(){

104



console.log(value);

}

functionbar(){

varvalue=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。

 setTimeout(fn,100);100毫秒是如何权衡的

参考回答：

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

 JS的垃圾回收机制

参考回答：

GC（garbagecollection），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

 写一个newBind函数，完成bind的功能。

参考回答：

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(typeofthis!=="function"){

thrownewError("Function.prototype.bind-whatistryingtobeboundisnotcallable");

105



}

varself=this;

varargs=Array.prototype.slice.call(arguments,1);

varfNOP=function(){};

varfbound=function(){

self.apply(thisinstanceofself?this:context,

args.concat(Array.prototype.slice.call(arguments)));

}

fNOP.prototype=this.prototype;

fbound.prototype=newfNOP();

returnfbound;

}

 怎么获得对象上的属性：比如说通过Object.key（）

参考回答：

从ES5开始，有三种方法可以列出对象的属性

for（letIinobj）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

 简单讲一讲ES6的一些新特性

参考回答：

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

 call和apply是用来做什么？

参考回答：

106



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，

3、调用函数，因为apply，call方法会使函数立即执行

 了解事件代理吗，这样做有什么好处

参考回答：

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

 如何写一个继承？

参考回答：

原型链继承

核心：将父类的实例作为子类的原型

特点：

非常纯粹的继承关系，实例是子类的实例，也是父类的实例

父类新增原型方法/原型属性，子类都能访问到

简单，易于实现

缺点：

要想为子类新增属性和方法，不能放到构造器中

无法实现多继承

来自原型对象的所有属性被所有实例共享

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：

解决了子类实例共享父类引用属性的问题

创建子类实例时，可以向父类传递参数

可以实现多继承（call多个父类对象）

缺点：

实例并不是父类的实例，只是子类的实例

只能继承父类的实例属性和方法，不能继承原型属性/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

107



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：

实例是父类的实例，不是子类的实例

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，然后通过将父类实例作

为子类原型，实现函数复用

特点：

可以继承实例属性/方法，也可以继承原型属性/方法

既是子类的实例，也是父类的实例

不存在引用属性共享问题

可传参

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，然后通过将父类实例作

为子类原型，实现函数复用

参考https://www.cnblogs.com/humin/p/4556820.html

 给出以下代码，输出的结果是什么？原因？ for(vari=0;i<5;i++)

{setTimeout(function(){console.log(i);},1000);}console.log(i)

参考回答：

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。

 给两个构造函数A和B，如何实现A继承B？

参考回答：

functionA(...){} A.prototype...

functionB(...){} B.prototype...

108



A.prototype=Object.create(B.prototype);

// 再在A的构造函数里newB(props);

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

lis[i].addEventListener('click',function(e){

alert(i);

},false)

}

 问能不能正常打印索引

参考回答：

在click的时候，已经变成length了

 如果已经有三个promise，A、B和C，想串行执行，该怎么写？

参考回答：

//promise

A.then(B).then(C).catch(...)

//async/await

(async()=>{

awaita();

awaitb();

awaitc();

})()

 知道private和public吗

参考回答：

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

 基础的js

参考回答：

Function.prototype.a=1;

Object.prototype.b=2;

functionA(){}

109



vara=newA();

console.log(a.a,a.b);//undefined,2

console.log(A.a,A.b);//1,2

 async和await具体该怎么用？

参考回答：

(async()=>{

awaitnewpromise();

})()

 知道哪些ES6，ES7的语法

参考回答：

promise，await/async，let、const、块级作用域、箭头函数

 promise和await/async的关系

参考回答：

都是异步编程的解决方案

 JS的数据类型

参考回答：

字符串，数字，布尔，数组，null，Undefined，symbol，对象。

 JS加载过程阻塞，解决方法。

参考回答：

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

 JS对象类型，基本对象类型以及引用对象类型的区别

参考回答：

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

数据段。基本数据类型有这六种:undefined、null、string、number、boolean、symbol。

110



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)。

 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？

参考回答：

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

 怎么实现一个计算一年中有多少周？

参考回答：

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。

 面向对象的继承方式

参考回答：

原型链继承

核心：将父类的实例作为子类的原型

特点：

非常纯粹的继承关系，实例是子类的实例，也是父类的实例

父类新增原型方法/原型属性，子类都能访问到

简单，易于实现

缺点：

要想为子类新增属性和方法，不能放到构造器中

无法实现多继承

来自原型对象的所有属性被所有实例共享

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：

111



解决了子类实例共享父类引用属性的问题

创建子类实例时，可以向父类传递参数

可以实现多继承（call多个父类对象）

缺点：

实例并不是父类的实例，只是子类的实例

只能继承父类的实例属性和方法，不能继承原型属性/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

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：

实例是父类的实例，不是子类的实例

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，然后通过将父类实例作

为子类原型，实现函数复用

特点：

可以继承实例属性/方法，也可以继承原型属性/方法

既是子类的实例，也是父类的实例

不存在引用属性共享问题

可传参

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，然后通过将父类实例作

为子类原型，实现函数复用

参考https://www.cnblogs.com/humin/p/4556820.html

 JS的数据类型

参考回答：

字符串，数字，布尔，数组，null，Undefined，symbol，对象。

 引用类型常见的对象

112



参考回答：

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

 es6的常用

参考回答：

promise，await/async，let、const、块级作用域、箭头函数

 class

参考回答：

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。

 口述数组去重

参考回答：

法一：indexOf循环去重

法二：ES6Set去重；Array.from(newSet(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。

 继承

参考回答：

原型链继承

核心：将父类的实例作为子类的原型

特点：

非常纯粹的继承关系，实例是子类的实例，也是父类的实例

父类新增原型方法/原型属性，子类都能访问到

简单，易于实现

缺点：

要想为子类新增属性和方法，不能放到构造器中

无法实现多继承

来自原型对象的所有属性被所有实例共享

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）

113



特点：

解决了子类实例共享父类引用属性的问题

创建子类实例时，可以向父类传递参数

可以实现多继承（call多个父类对象）

缺点：

实例并不是父类的实例，只是子类的实例

只能继承父类的实例属性和方法，不能继承原型属性/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

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：

实例是父类的实例，不是子类的实例

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，然后通过将父类实例作

为子类原型，实现函数复用

特点：

可以继承实例属性/方法，也可以继承原型属性/方法

既是子类的实例，也是父类的实例

不存在引用属性共享问题

可传参

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，然后通过将父类实例作

为子类原型，实现函数复用

参考https://www.cnblogs.com/humin/p/4556820.html

 call和apply的区别

参考回答：

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。

114



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。

 es6的常用特性

参考回答：

promise，await/async，let、const、块级作用域、箭头函数

 箭头函数和function有什么区别

参考回答：

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

 new操作符原理

参考回答：

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。

 bind,apply,call

参考回答：

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。

 bind和apply的区别

参考回答：

返回不同：bind返回是函数

参数不同：apply(A,arguments)，bind(A,args1,args2)

 数组的去重

参考回答：

115



法一：indexOf循环去重

法二：ES6Set去重；Array.from(newSet(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。

 闭包

参考回答：

（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，则默认的会添加到全

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。

 promise实现

参考回答：

Promise实现如下

functionPromise(fn){

varstate='pending',

value=null,

callbacks=[];

this.then=function(onFulfilled,onRejected){

returnnewPromise(function(resolve,reject){

handle({

onFulfilled:onFulfilled||null,

onRejected:onRejected||null,

resolve:resolve,

reject:reject

});

});

};

116



functionhandle(callback){

if(state==='pending'){

callbacks.push(callback);

return;

}

varcb=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);

}

functionresolve(newValue){

if(newValue&&(typeofnewValue==='object'||typeofnewValue==='function')){

varthen=newValue.then;

if(typeofthen==='function'){

then.call(newValue,resolve,reject);

return;

}

}

state='fulfilled';

value=newValue;

execute();

}

functionreject(reason){

state='rejected';

value=reason;

execute();

}

functionexecute(){

setTimeout(function(){

callbacks.forEach(function(callback){

handle(callback);

});

},0);

}

fn(resolve,reject);

}

 assign的深拷贝

117



参考回答：

functionclone(obj){

varcopy;

switch(typeofobj){

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）

1if(obj===null){

2copy=null;

3}else{

4if(Object.prototype.toString.call(obj).slice(8,-1)==="Array"){

5copy=[];

6for(vari=0;i<obj.length;i++){

7copy.push(clone(obj[i]));

8}

9}else{

10copy={};

11for(varjinobj){

12copy[j]=clone(obj[j]);

13}

14}

15}

16break;

17default:

18copy=obj;

19break;

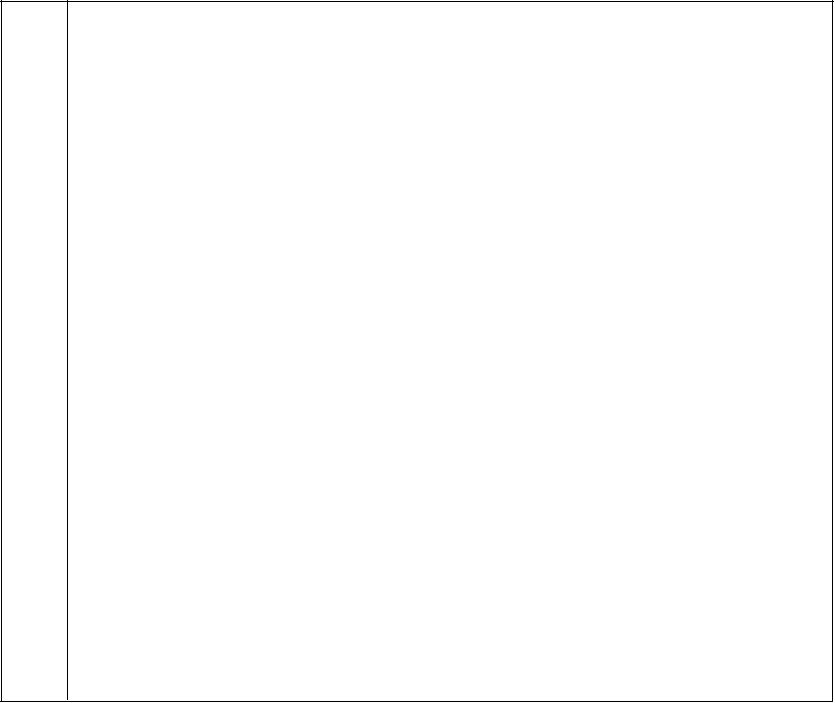
20}

21returncopy;

22}

 说promise，没有promise怎么办

118



参考回答：

没有promise，可以用回调函数代替

 事件委托

参考回答：

把一个元素响应事件（click、keydown......）的函数委托到另一个元素；

优点：减少内存消耗、动态绑定事件。

 箭头函数和function的区别

参考回答：

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

 arguments

参考回答：

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

 箭头函数获取arguments

参考回答：

可用…rest参数获取

 Promise

参考回答：

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”

119



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。

 事件代理

参考回答：

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。

 Eventloop

参考回答：

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microsoft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,UIrendering

microtasks:process.nextTick,Promise,MutationObserver

2| 前端核心

2.1| 服务端编程

 JSONP的缺点

参考回答：

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。

 跨域（jsonp，ajax）

120



参考回答：

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。

 如何实现跨域

参考回答：

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

 dom是什么，你的理解？

参考回答：

文档对象模型（DocumentObjectModel，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。

 关于dom的api有什么

参考回答：

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

2.2|Ajax

 ajax返回的状态

参考回答：

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

121



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

 实现一个Ajax

参考回答：

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。

 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

参考回答：

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

varxhr=newXMLHttpRequest();

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

functionsend(action,arg2){

122



//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

functionsend\_action(command,arg2){

vardtd=$.Deferred();//定义deferred对象

$.post(

"url",

{

command:command

123



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();

});

returndtd.promise();//返回Deferred对象的promise，防止在外部

 写出原生Ajax

参考回答：

Ajax能够在不重新加载整个页面的情况下与服务器交换数据并更新部分网页内容，实现

局部刷新，大大降低了资源的浪费，是一门用于快速创建动态网页的技术，ajax的使用

分为四部分：

1、创建XMLHttpRequest对象varxhr=newXMLHttpRequest();

2、向服务器发送请求，使用xmlHttpRequest对象的open和send方法，

3、监听状态变化，执行相应回调函数

varxhr=newXMLHttpRequest();

xhr.open('get','aabb.php',true);

xhr.send(null);

xhr.onreadystatechange=function(){

if(xhr.readyState==4){

if(xhr.status==200){

console.log(xhr.responseText);

}

}

}

 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？

参考回答：

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）

124



（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。

 Fetch和Ajax比有什么优缺点？

参考回答：

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。

 原生JS的ajax

参考回答：

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。

2.3| 移动web开发

 知道PWA吗

参考回答：

PWA全称ProgressiveWebApp，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 AppManifest 和 ServiceWorker

来实现 PWA 的安装和离线等功能

 移动布局方案

https://juejin.im/post/599970f4518825243a78b9d5#heading-22

125



 Rem,Em

https://blog.csdn.net/romantic\_love/article/details/80875462

一、rem单位如何转换为像素值

1.当使用rem单位的时候，页面转换为像素大小取决于叶根元素的字体大小，即HTML

元素的字体大小。根元素字体大小乘rem的值。例如，根元素的字体大小为16px，那么

10rem就等同于10\*16=160px。

二、em是如何转换成px的

当使用em单位的时候，像素值是将em值乘以使用em单位的元素的字体大小。例如一

个div的字体为18px，设置它的宽高为10em，那么此时宽高就是18px\*10em=180px。

.test{

width:10em;

height:10em;

background-color:#ff7d42;

font-size:18px;

}

一定要记住的是，em是根据使用它的元素的font-size的大小来变化的，而不是根据父

元素字体大小。有些元素大小是父元素的多少倍那是因为继承了父元素中font-size的设

定，所以才起到的作用。

2.em单位的继承效果。

使用em单位存在继承的时候，每个元素将自动继承其父元素的字体大小，继承的效果

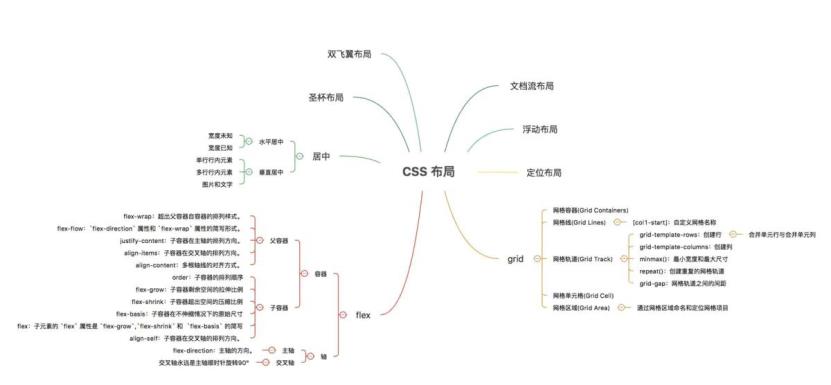
只能被明确的字体单位覆盖，比如px和vw。只要父级元素上面一直有fontsize为em单

位，则会一直继承，但假如自己设置了font-size的单位为px的时候，则会直接使用自

己的px单位的值。

三、根html的元素将会继承浏览器中设置的字体大小

126



除非显式的设置固定值去覆盖。所以html元素的字体大小虽然是直接确定rem的值，

但这个字体大小首先是来源于浏览器的设置。（所以一定要设置html的值的大小，因

为有可能用户的浏览器字体大小是不一致的。）

四、当em单位设置在html元素上时

它将转换为em值乘以浏览器字体大小的设置。

例如：

html{

font-size:1.5em;

}

可以看到，因为浏览器默认字体大小为16px，所以当设置HTML的fontsize的值为1.5em

的售后，其对应的px的值为16\*1.5=24px

所以此时，再设置其他元素的rem的值的时候，其对应的像素值为n\*24px。

例如，test的rem的值为10，

.test{

width:10rem;

height:10rem;

background-color:#ff7d42;

}

可以看到test的font-size继承了html的值24px，而此时宽高为24\*10=240px

总结

1. rem单位翻译为像素值的时候是由html元素的字体大小决定的。此字体大小会

被浏览器中字体大小的设置影响，除非显式的在html为font-size重写一个单位。

2.

em单位转换为像素值的时候，取决于使用它们的元素的font-size的大小，但是有

因为有继承关系，所以比较复杂。

优缺点

em可以让我们的页面更灵活，更健壮，比起到处写死的px值，em似乎更有张力，改

动父元素的字体大小，子元素会等比例变化，这一变化似乎预示了无限可能,

em做弹性布局的缺点还在于牵一发而动全身，一旦某个节点的字体大小发生变化，那

么其后代元素都得重新计算

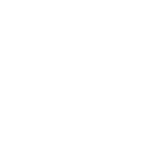
 flex布局及优缺点

https://juejin.im/post/599970f4518825243a78b9d5#heading-22

css3引入的，flex布局；优点在于其容易上手，根据flex规则很容易达到某个布局效果，

然而缺点是：浏览器兼容性比较差，只能兼容到ie9及以上；

127



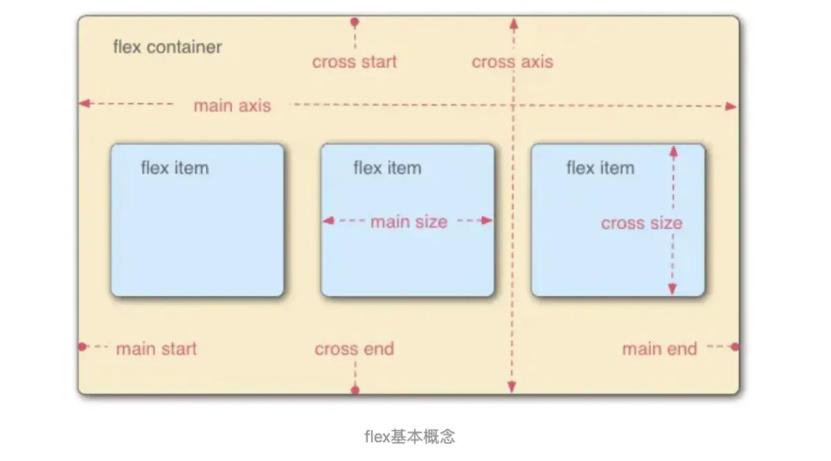
 Rem布局及其优缺点

参考回答：

首先Rem相对于根(html)的font-size大小来计算。简单的说它就是一个相对单例

如:font-size:10px;,那么（1rem=10px）了解计算原理后

128



首先解决怎么在不同设备上设置html的font-size大小。其实rem布局的本质是等比缩

放，一般是基于宽度。

优点

可以快速适用移动端布局 字体图片 高度

缺点

①目前ie不支持，对pc页面来讲使用次数不多；

②数据量大：所有的图片，盒子都需要我们去给一个准确的值；才能保证不同机型的适

配；

③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。

 百分比布局

参考回答：

1、具体分析

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。

padding、border、margin等等属不论是垂直方向还是水平方向，都相对于直接父元素的

width。

除了border-radius外，还有比如translate、background-size等都是相对于自身的。

2、百分比单位布局应用

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：

.trangle{

height:0;

width:100%;

padding-top:75%;

}

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。

129



（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。

 移动端适配1px的问题

https://blog.csdn.net/weixin\_43675871/article/details/84023447

首先，我们了解devicePixelRatio（DPR）这个东西

在window对象中有一个devicePixelRatio属性，他可以反应css中的像素与设备

的像素比。然而1px在不同的移动设备上都等于这个移动设备的1px，这是因为

不同的移动设备有不同的像素密度。有关这个属性，它的官方的定义为：设备物

理像素和设备独立像素的比例，也就是devicePixelRatio= 物理像素 / 独立像素

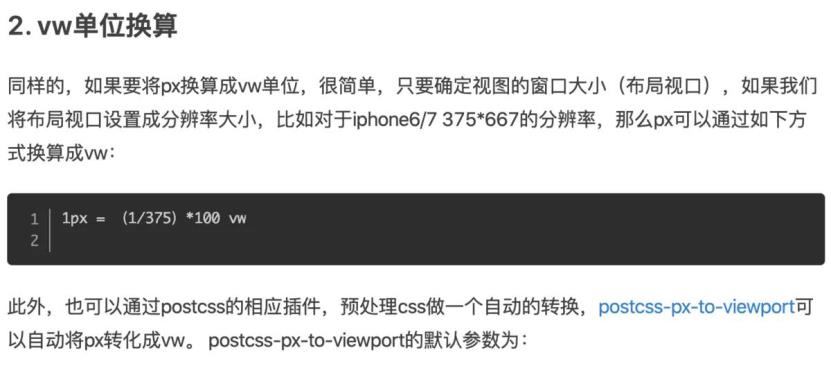
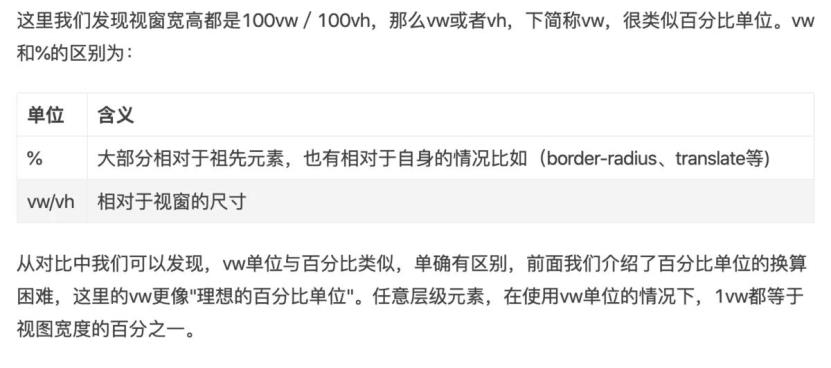
1px变粗的原因：viewport的设置和屏幕物理分辨率是按比例而不是相同的. 移动

端window对象有个devicePixelRatio属性,它表示设备物理像素和css像素的比例,

在retina屏的iphone手机上, 这个值为2或3,css里写的1px长度映射到物理像素

上就有2px或3px那么长

130



1.用小数来写px值 （不推荐）

IOS8下已经支持带小数的px值,mediaquery 对应 devicePixelRatio 有个查询值

-webkit-min-device-pixel-ratio,css可以写成这样

通过-webkit-min-device-pixel-ratio设置。

.border{border:1pxsolid#999}

@mediascreenand(-webkit-min-device-pixel-ratio:2){

.border{border:0.5pxsolid#999}

}

@mediascreenand(-webkit-min-device-pixel-ratio:3){

.border{border:0.333333pxsolid#999}

}

如果使用less/sass的话只是加了1句mixin

缺点: 安卓与低版本IOS不适用, 这个或许是未来的标准写法, 现在不做指望

2、flexible.js

这是淘宝移动端采取的方案,github的地址:https://github.com/amfe/lib-flexible 前面已经

说过1px变粗的原因就在于一刀切的设置 viewport 宽度, 如果能把 viewport 宽度设置

为实际的设备物理宽度,css 里的 1px 不就等于实际 1px 长了么.flexible.js 就是这样

干的.

<metaname=”viewport”>里面的 scale 值指的是对 idealviewport 的缩放,flexible.js 检

测到 IOS 机型, 会算出 scale=1/devicePixelRatio, 然后设置viewport

3、伪类+transform实现

对于解决1px边框问题，我个人觉得最完美的解决办法还是伪类+transform比较好。

原理：是把原先元素的 border 去掉，然后利用 :before 或者 :after 重做 border ，并

transform 的 scale 缩小一半，原先的元素相对定位，新做的 border 绝对定位。

mediaquery

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。

@mediascreenand(min-width:320px){

html{

font-size:50px;

}

}

@media

方便应用广泛 适用于 pc端 手机页面，通常做自适应布局时 我们比较常用。

缺点：相对于代码要重复很多，得知道设备的宽度，手机的分辨率很多所以麻烦了点，

不过性能方面肯定最高； 可能存在闪屏的问题

@media处理手机和pc端界面兼容的问题，在IE 上的访问出现问题，百度方法，找找

css3-mediaquerieshttp://blog.csdn.net/small\_tu/article/details/47317453

 移动端性能优化相关经验

131



https://blog.csdn.net/tangxiujiang/article/details/79791545

 toB 和 toC 项目的区别

参考回答：

toB（business）即面向企业，toC（ customer）即面向普通用户

简单的事情重复去做，重复的事情用心去做，长期坚持，自然功成，无论是B端还是C

端都同样适用。

Tob与Toc的区别

作者 猪八戒网 •10-1110:18:49 阅读 419

oB产品价值何来？

最近团队在toB产品研究的过程中，得出结论，相对于toC产品与服务，toC产品更注重

产品用户的共性而淡化角色关系，而toB产品则更强调面向用户、客户的角色关系，而

淡化共性提取。实际上，这是由服务对象所引起的，C端产品的服务对象，由终端所限，

是一个面向个体的服务。而B端服务使用最终是面向一个系统体系组织，在干系人间配

合使用中发挥产品价值。

一个好的产品toB可以让组织的系统变得更好，最终反哺于系统中的各个单位。

需求动力之不同toC的产品方法论，用户体验是几乎最为重要的需求来源，腾讯此前，

也以"以用户体验为归依"来驱动企业产品打造。

但B端产品则不同，B端在一个商业的背景之下，B端的决策思路是，"以企业获益为归

依"，系统是否有利于企业的生产力，竞争力等，单纯的用户体验，仅能让员工得到片

刻的享受， 但无法说服企业，企业并不会为一个不能"赚钱"的东西买单。

需求动力的不同，引发的这是购买使用决策体系的变化。

toB产品应更考虑 获益与系统性价值，部分情况还有可能会牺牲掉局部个体的利益，对

于使用者而言应该是自律或他律的，toC产品则更考虑的是个体用户的偏好，并长时间

内，基于技术效率的提升，产品的服务中心更多地围绕着更高效地帮助用户的"欲望"释

放进行设计，对于使用者而言是一个释放自我的存在。

 移动端兼容性

https://zhuanlan.zhihu.com/p/28206065

1.IOS移动端click事件300ms的延迟相应

3.h5底部输入框被键盘遮挡问题

10.CSS动画页面闪白,动画卡顿

解决方法:

1.尽可能地使用合成属性transform和opacity来设计CSS3动画，不使用position

的left和top来定位

2.开启硬件加速

9.上下拉动滚动条时卡顿、慢

body{-webkit-overflow-scrolling:

touch;overflow-scrolling:touch;}

Android3+和iOS5+支持CSS3的新属性为overflow-scrolling

132



 小程序

参考回答：

移动端手势

 2X图 3X图适配

参考回答：

实际程序开发当中，我们代码中用的值是指逻辑分辨率pt，而不是像素分辨率px，比如

我们定义一个按钮的高度为45，这个45指的是45pt而不是45px。在非Retina屏下1pt

=1px，4和4.7寸Retina屏下1pt=2px，5.5和x下1pt=3px.我们制作不同尺寸的图片，

比如@1x为22px，则@2x为44px，@3x为66px，命名分别为image.png，在项目的

Assets.xcassets中新建NewImageSet，修改名字为image，并把相应尺寸的图片拖放至相

应位置。

/\* 根据dpr显示2x图/3x图 \*/

.bg-image(@url){

background-image:~"url('@{url}@2x.png')";

@media(-webkit-min-device-pixel-ratio:3),(min-device-pixel-ratio:3){

background-image:~"url('@{url}@3x.png')";

}

}

.bg-color(@color){

background-color:@color;

}

133



 图片在安卓上，有些设备模糊问题

参考回答：

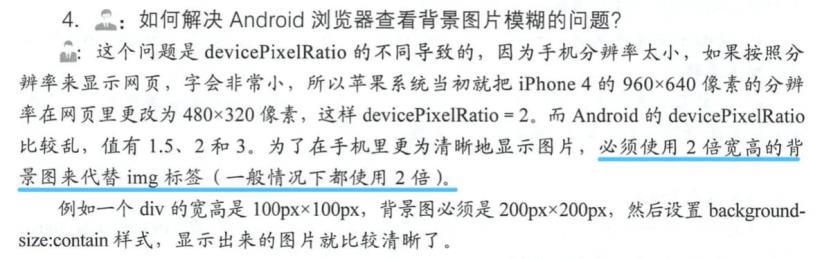
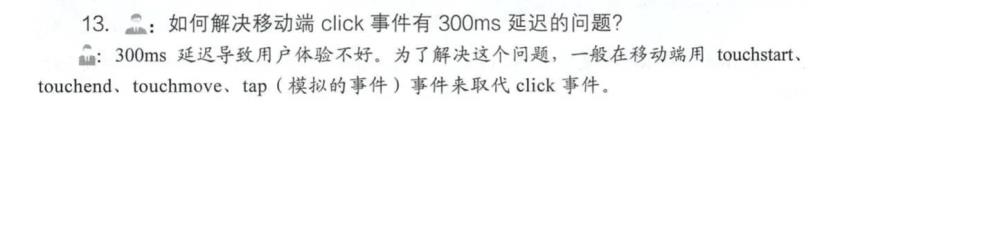
 固定定位布局键盘挡住输入框内容

参考回答：

 click的300ms延迟问题和点击穿透问题

https://www.jianshu.com/p/6e2b68a93c88

134



FastClick 是 FTLabs 专门为解决移动端浏览器 300 毫秒点击延迟问题所开发的一个

轻量级的库。FastClick的实现原理是在检测到touchend事件的时候，会通过DOM自定

义事件立即出发模拟一个click事件，并把浏览器在300ms之后的click事件阻止掉。

 phone及ipad下输入框默认内阴影

参考回答：

 防止手机中页面放大和缩小

参考回答：

<metaname="viewport"content="user-scalable=no">

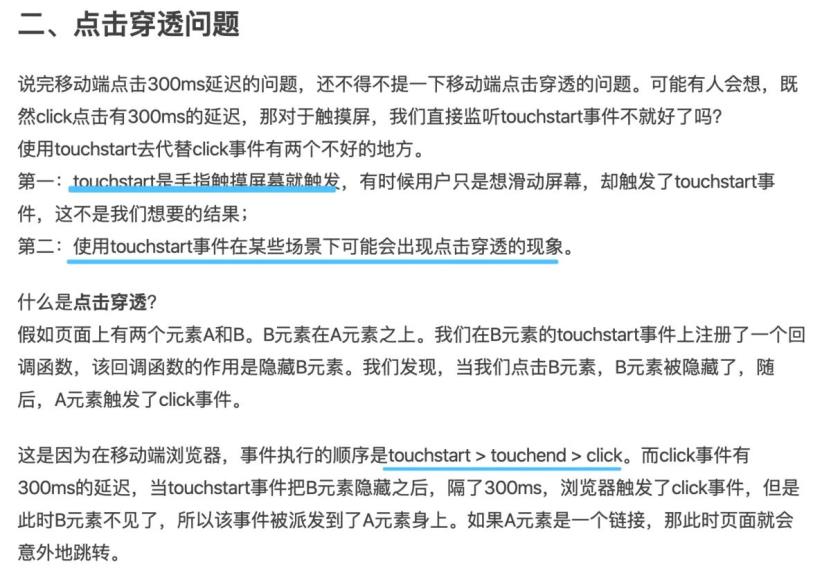
<metaname="viewport"content="initial-scale=1,maximum-scale=1">

 px、em、rem、%、vw、vh、vm这些单位的区别

参考回答：

https://www.jianshu.com/p/ba26509bc5b3

135



 移动端适配-dpr浅析

https://www.jianshu.com/p/cf600c2930cb

dpr= 物理像素 /css像素

在dpr=2； 1px的css像素在设备中是2px的物理像素，这会导致在设备上看上去1px

的边框是2px

解决方法：

用transfrom： scale（）缩小dpr倍数

在meta标签中设定scale缩小两倍

 移动端扩展点击区域

参考回答：

父级代理事件

将a标签设置成块级元素

 上下拉动滚动条时卡顿、慢

参考回答：

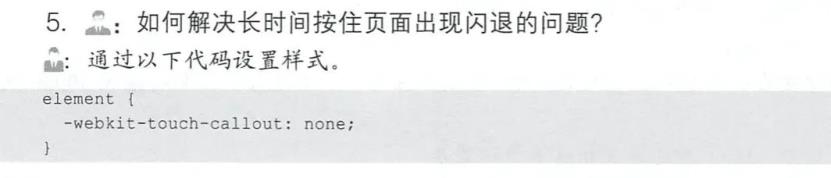
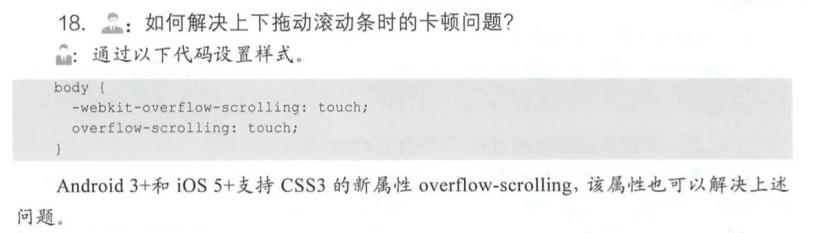
 长时间按住页面出现闪退

参考回答：

 ios和android下触摸元素时出现半透明灰色遮罩

参考回答：

136



 active兼容处理 即 伪类：active失效

https://blog.csdn.net/diaobuwei1238/article/details/101716814

将 :visited放到最后，则会导致以下结果：若链接已经被访问过，a:visited会覆盖:active

和:hover的样式声明，链接将总是呈现为紫色，无论鼠标悬停还是按下激活，链接都将

保持为紫色。

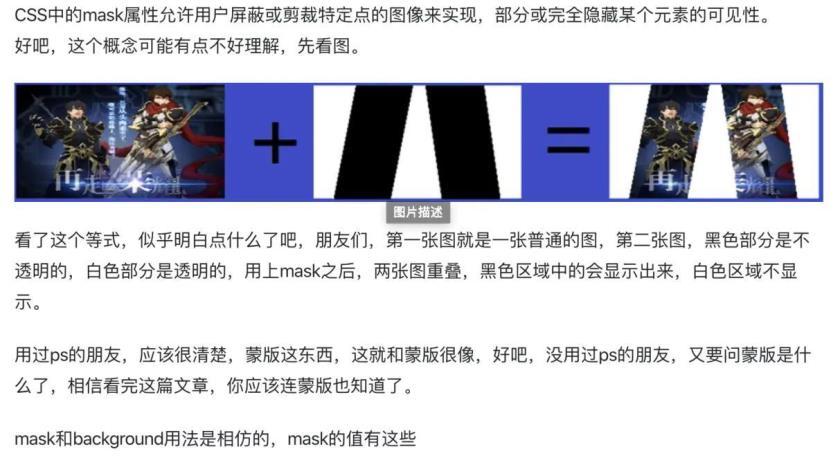
基于此原因，上述代码必须按照顺序定义，一般称为LVHA-order::link—:visited

—:hover—:active，为方便记忆，可记为“LOVEHATE”

 webkitmask兼容处理

https://segmentfault.com/a/1190000011838367

137



 transiton闪屏

参考回答：

//设置内联的元素在3D空间如何呈现：保留

3D-webkit-transform-style:preserve-3D;

//设置进行转换的元素的背面在面对用户时是否可见：隐藏

-webkit-backface-visibility:hidden;

 圆角bug

参考回答：

作者：Aniugel

链接：https://www.jianshu.com/p/610123c6ed45

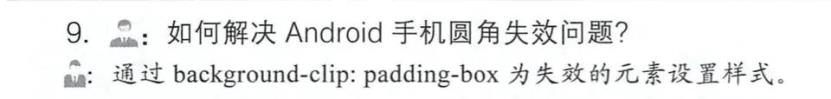
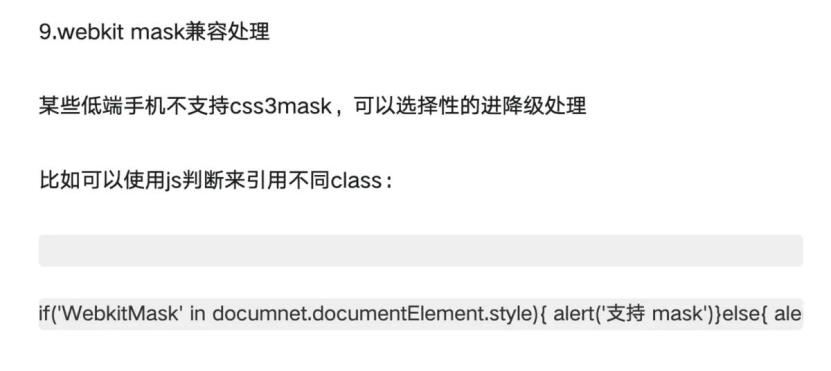
来源：简书

3| 前端进阶

3.1| 前端工程化

 Babel的原理是什么?

138



参考回答：

babel 的转译过程也分为三个阶段，这三步具体是：

o

o

解析 Parse: 将代码解析生成抽象语法树( 即AST)，即词法分析与语法分

析的过程

转换 Transform: 对于 AST 进行变换一系列的操作，babel 接受得到

AST 并通过 babel-traverse 对其进行遍历，在此过程中进行添加、更新

及移除等操作

o

生成 Generate: 将变换后的 AST 再转换为 JS 代码, 使用到的模块是

babel-generator

 如何写一个babel插件?

参考回答：

Babel解析成AST，然后插件更改AST，最后由Babel输出代码

那么Babel的插件模块需要你暴露一个function，function内返回visitor

module.export=function(babel){

return{

visitor:{

}

}

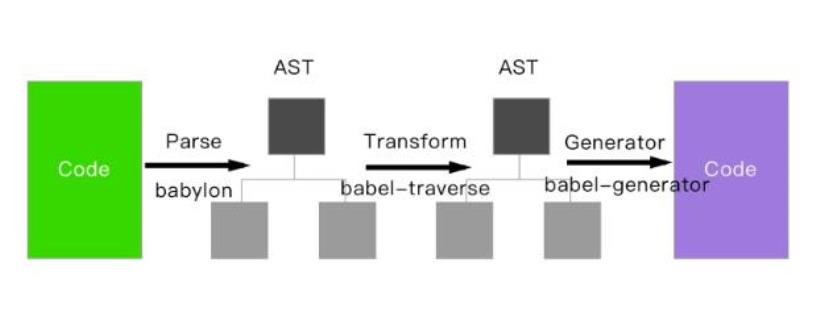
}

visitor是对各类型的AST节点做处理的地方，那么我们怎么知道Babel生成了的

AST有哪些节点呢？

很简单，你可以把Babel转换的结果打印出来，或者这里有传送门:ASTexplorer

139



这里我们看到constresult=1+2中的1+1是一个BinaryExpression节点，那么在

visitor中，我们就处理这个节点

varbabel=require('babel-core');

vart=require('babel-types');

constvisitor={

BinaryExpression(path){

constnode=path.node;

letresult;

// 判断表达式两边，是否都是数字

if(t.isNumericLiteral(node.left)&&t.isNumericLiteral(node.right)){

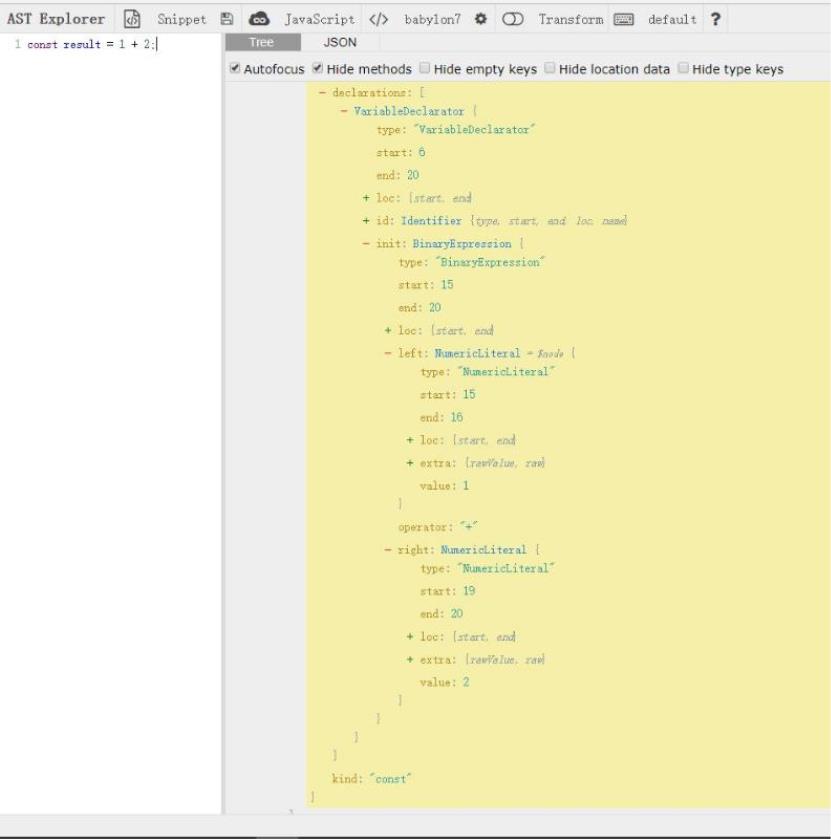
// 根据不同的操作符作运算

switch(node.operator){

case"+":

result=node.left.value+node.right.value;

140



break

case"-":

result=node.left.value-node.right.value;

break;

case"\*":

result= node.left.value\*node.right.value;

break;

case"/":

result= node.left.value/node.right.value;

break;

case"\*\*":

leti=node.right.value;

while(--i){

result=result||node.left.value;

result= result\*node.left.value;

}

break;

default:

}

}

// 如果上面的运算有结果的话

if(result!==undefined){

// 把表达式节点替换成number字面量

path.replaceWith(t.numericLiteral(result));

}

}

};

module.exports=function(babel){

return{

visitor

};

}

插件写好了，我们运行下插件试试

constbabel=require("babel-core");

constresult=babel.transform("constresult=1+2;",{

plugins:[

require("./index")

]

});

console.log(result.code);//constresult=3;

与预期一致，那么转换constresult=1+2+3+4+5;呢?

结果是:constresult=3+3+4+5;

这就奇怪了，为什么只计算了1+2之后，就没有继续往下运算了?

我们看一下这个表达式的AST树

141



你会发现Babel解析成表达式里面再嵌套表达式。

表达式( 表达式( 表达式( 表达式(1+2)+3)+4)+5)

而我们的判断条件并不符合所有的，只符合1+2

// 判断表达式两边，是否都是数字

if(t.isNumericLiteral(node.left)&&t.isNumericLiteral(node.right)){}

那么我们得改一改

第一次计算1+2之后，我们会得到这样的表达式

表达式( 表达式( 表达式(3+3)+4)+5)

其中3+3又符合了我们的条件， 我们通过向上递归的方式遍历父级节点

又转换成这样:

表达式( 表达式(6+4)+5)

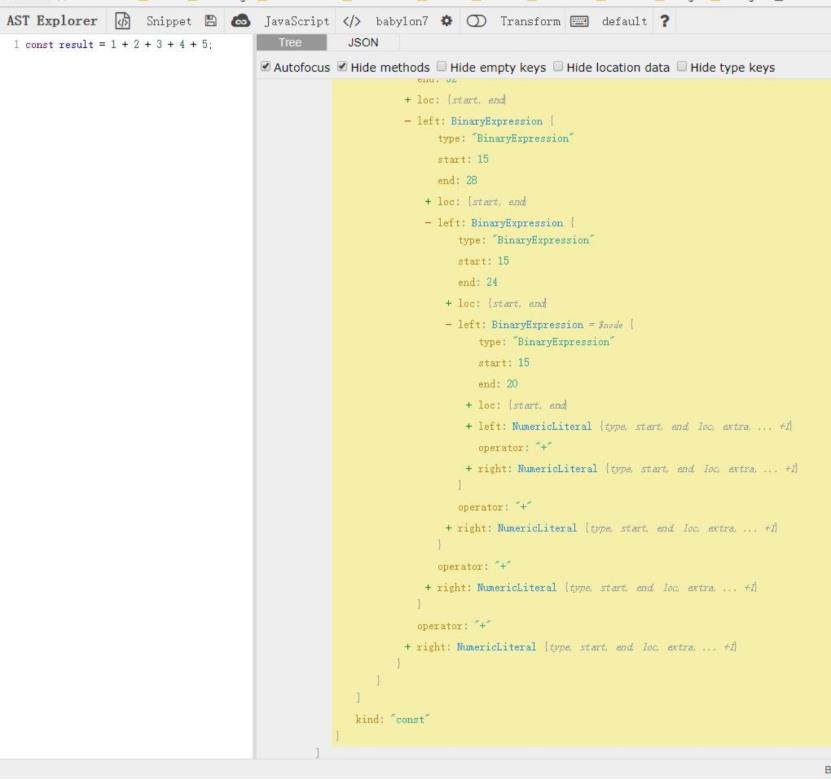
表达式(10+5)

15

// 如果上面的运算有结果的话

if(result!==undefined){

142



// 把表达式节点替换成number字面量

path.replaceWith(t.numericLiteral(result));

letparentPath=path.parentPath;

// 向上遍历父级节点

parentPath&&visitor.BinaryExpression.call(this,parentPath);

}

到这里，我们就得出了结果constresult=15;

那么其他运算呢:

constresult=100+10-50>>>constresult=60;

constresult=(100/2)+50>>>constresult=100;

constresult=(((100/2)+50\*2)/50)\*\*2>>>constresult=9;

 你的git工作流是怎样的?

参考回答：

GitFlow 是由 VincentDriessen 提出的一个 git操作流程标准。包含如下几个关键分支：

master主分支develop主开发分支，包含确定即将发布的代码

feature新功能分支，一般一个新功能对应一个分支，对于功能的拆分需要比较合理，以

避免一些后面不必要的代码冲突

release发布分支，发布时候用的分支，一般测试时候发现的

bug 在这个分支进行修复hotfixhotfix 分支，紧急修 bug 的时候用

GitFlow 的优势有如下几点：



并行开发：GitFlow 可以很方便的实现并行开发：每个新功能都会建立一个新

的feature分支，从而和已经完成的功能隔离开来，而且只有在新功能完成开发

的情况下，其对应的feature分支才会合并到主开发分支上（也就是我们经常说

的develop分支）。另外，如果你正在开发某个功能，同时又有一个新的功能需

要开发，你只需要提交当前feature的代码，然后创建另外一个feature分支并完

成新功能开发。然后再切回之前的feature分支即可继续完成之前功能的开发。

协作开发：GitFlow 还支持多人协同开发，因为每个feature分支上改动的代码

都只是为了让某个新的feature可以独立运行。同时我们也很容易知道每个人都

在干啥。





发布阶段：当一个新feature开发完成的时候，它会被合并到develop分支，这

个分支主要用来暂时保存那些还没有发布的内容，所以如果需要再开发新

的feature，我们只需要从develop分支创建新分支，即可包含所有已经完成

的feature。



支持紧急修复：GitFlow 还包含了hotfix分支。这种类型的分支是从某个已经发

布的 tag 上创建出来并做一个紧急的修复，而且这个紧急修复只影响这个已经

发布的 tag，而不会影响到你正在开发的新feature。

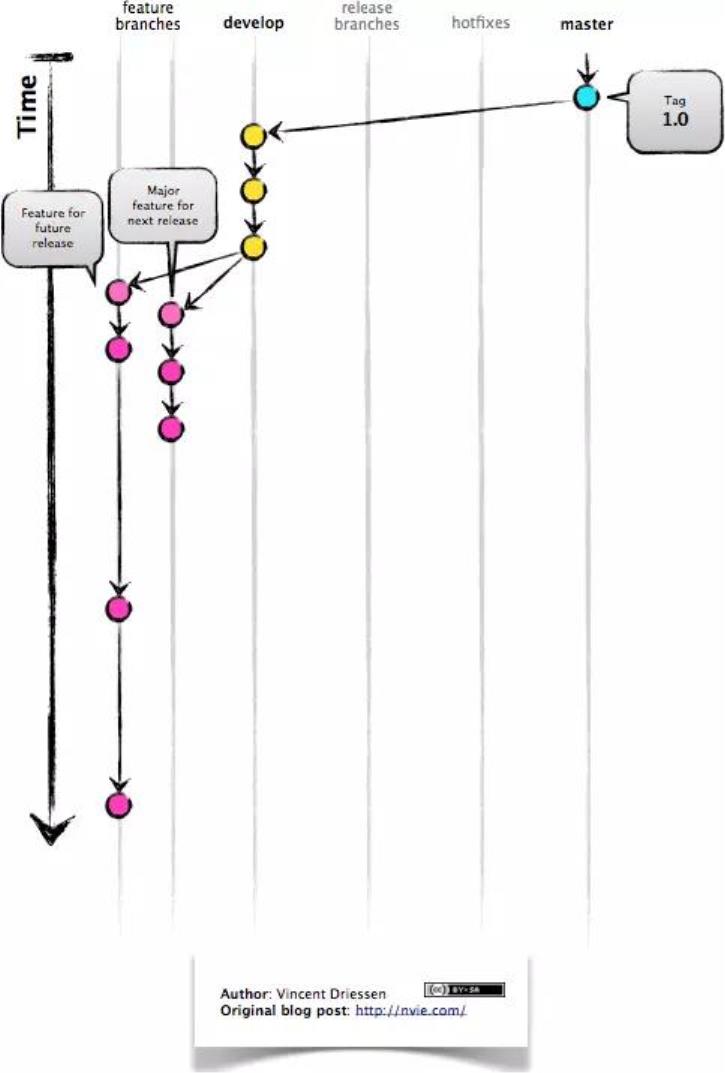
然后就是 GitFlow 最经典的几张流程图，一定要理解：

143



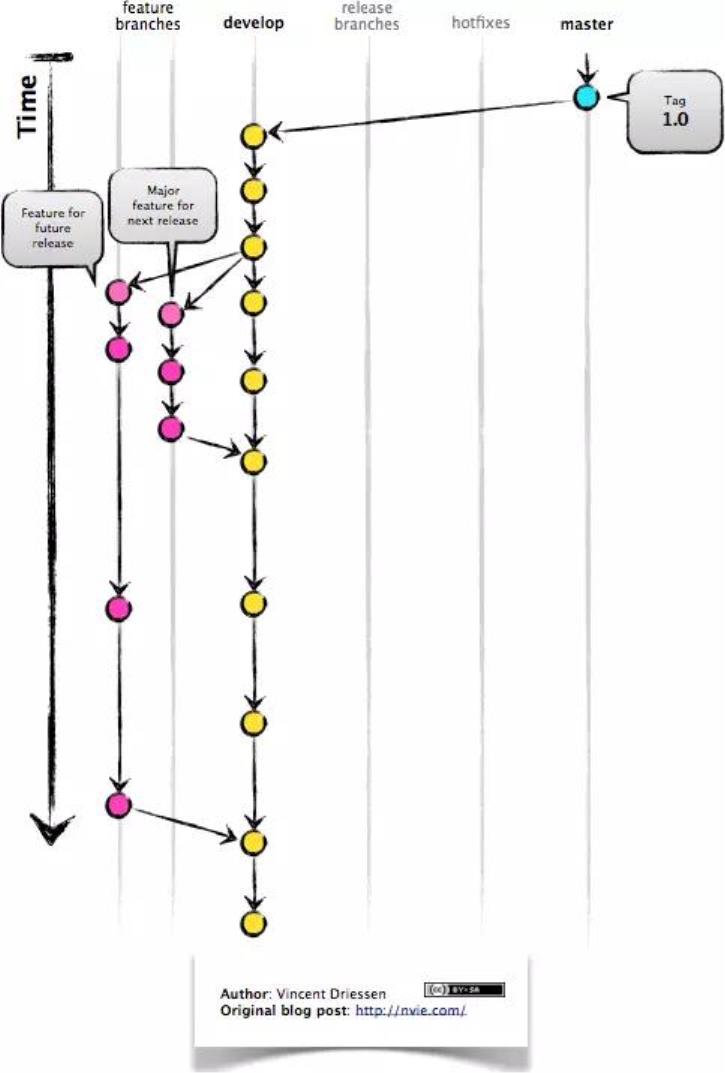
feature分支都是从develop分支创建，完成后再合并到develop分支上，等待发布。

144



当需要发布时，我们从develop分支创建一个release分支

145



然后这个release分支会发布到测试环境进行测试，如果发现问题就在这个分支直接进

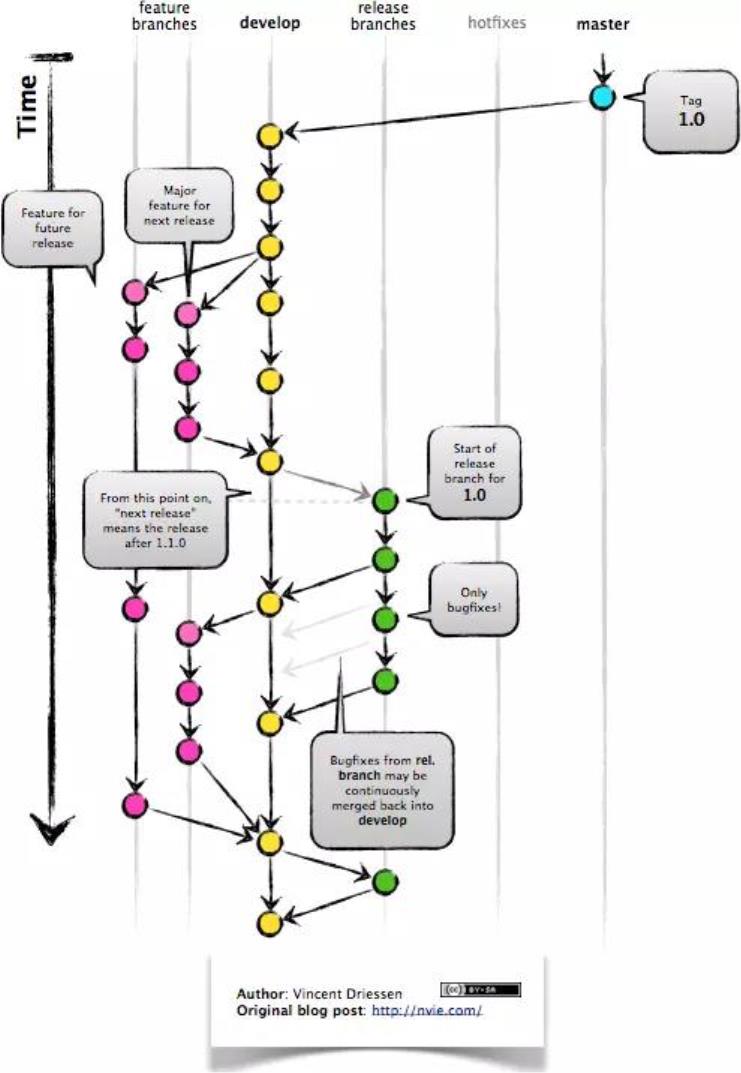
行修复。在所有问题修复之前，我们会不停的重复发布->测试->修复->重新发布->重

新测试这个流程。

发布结束后，这个release分支会合并到develop和master分支，从而保证不会有代码

丢失。

146



master分支只跟踪已经发布的代码，合并到master上的 commit 只能来自release分支

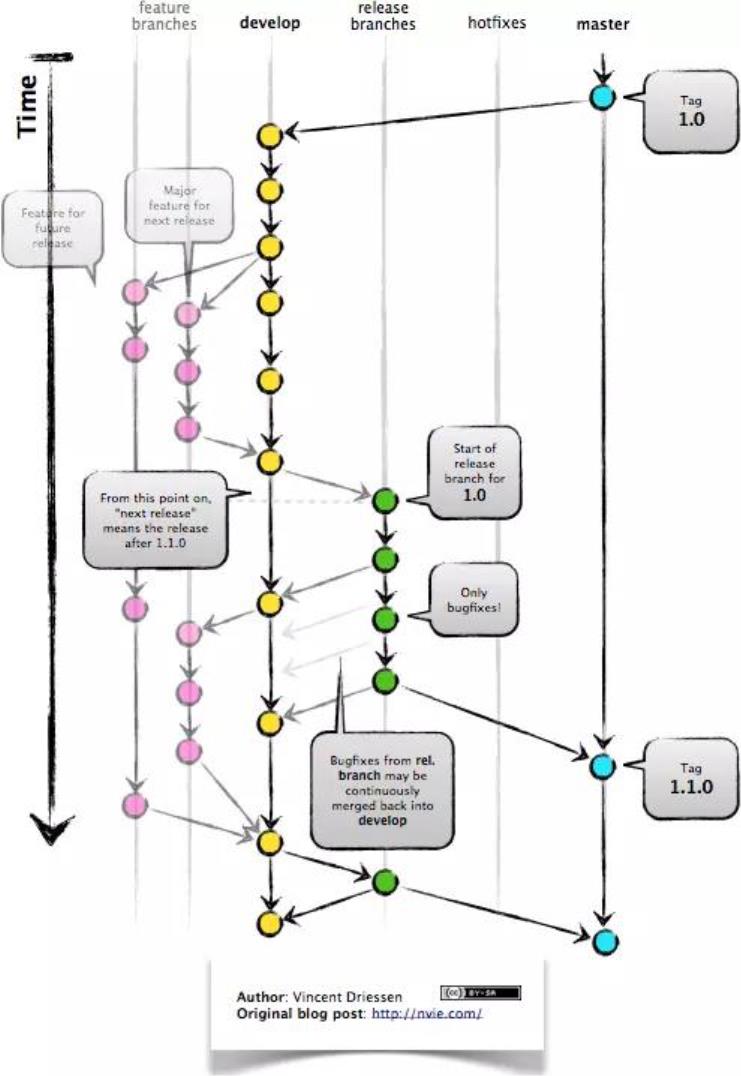
和hotfix分支。

hotfix分支的作用是紧急修复一些 Bug。

它们都是从master分支上的某个 tag 建立，修复结束后再合并到develop和master分

支上。

147



 rebase 与 merge的区别?

参考回答：

gitrebase 和 gitmerge 一样都是用于从一个分支获取并且合并到当前分支.

假设一个场景,就是我们开发的[feature/todo]分支要合并到master主分支,那么用rebase或

者merge有什么不同呢?

o

marge 特点：自动创建一个新的commit 如果合并的时候遇到冲突，仅需

要修改后重新commit

o

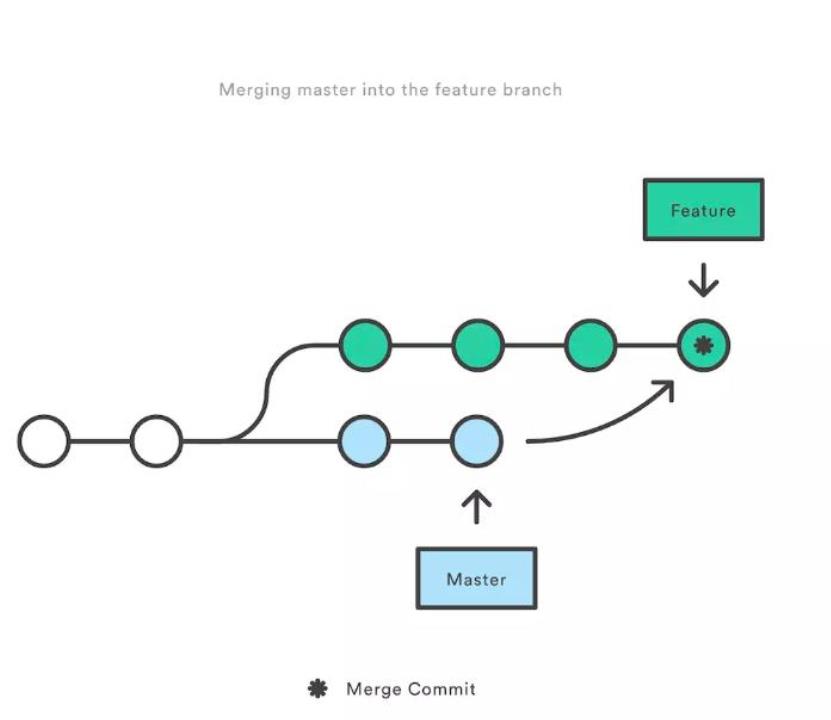
o

优点：记录了真实的commit情况，包括每个分支的详情

缺点：因为每次merge会自动产生一个mergecommit，所以在使用一些

git 的GUItools，特别是commit比较频繁时，看到分支很杂乱。

148



o

o

o

rebase 特点：会合并之前的commit历史

优点：得到更简洁的项目历史，去掉了mergecommit

缺点：如果合并出现代码问题不容易定位，因为re-write了history

因此,当需要保留详细的合并信息的时候建议使用gitmerge，特别是需要将分支合并进入

master分支时；当发现自己修改某个功能时，频繁进行了gitcommit提交时，发现其实

过多的提交信息没有必要时，可以尝试gitrebase.

 gitreset、gitrevert 和 gitcheckout 有什么区别

参考回答：

这个问题同样也需要先了解 git 仓库的三个组成部分：工作区（WorkingDirectory）、

暂存区（Stage）和历史记录区（History）。

o

工作区：在 git 管理下的正常目录都算是工作区，我们平时的编辑工作

都是在工作区完成

o

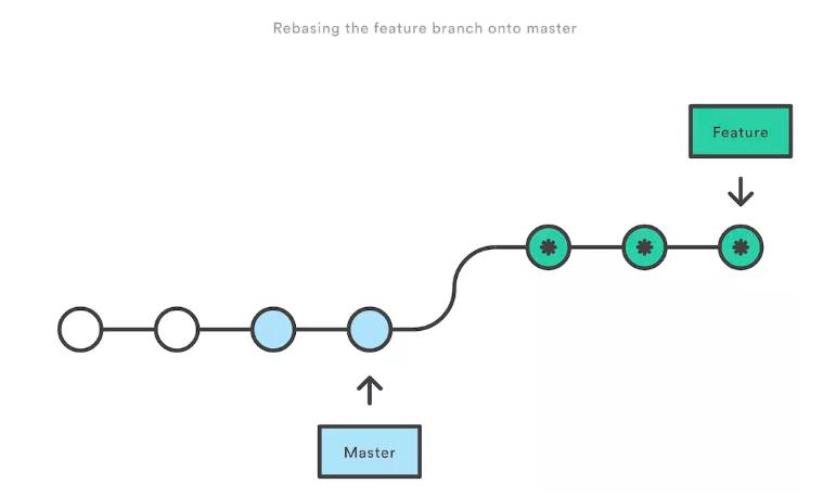
o

暂存区：临时区域。里面存放将要提交文件的快照

历史记录区：gitcommit 后的记录区

三个区的转换关系以及转换所使用的命令：

149



gitreset、gitrevert 和 gitcheckout的共同点：用来撤销代码仓库中的某些更改。

然后是不同点：

首先，从 commit 层面来说：

o

gitreset 可以将一个分支的末端指向之前的一个 commit。然后再下次 git

执行垃圾回收的时候，会把这个 commit 之后的 commit 都扔掉。gitreset

还支持三种标记，用来标记 reset 指令影响的范围：

.

.

.

--mixed：会影响到暂存区和历史记录区。也是默认选项

--soft：只影响历史记录区

--hard：影响工作区、暂存区和历史记录区

注意：因为 gitreset 是直接删除 commit 记录，从而会影响到其他开发人员的分

支，所以不要在公共分支（比如 develop）做这个操作。

.

gitcheckout 可以将 HEAD 移到一个新的分支，并更新工作目录。

因为可能会覆盖本地的修改，所以执行这个指令之前，你需要

stash 或者 commit 暂存区和工作区的更改。

o

gitrevert 和 gitreset 的目的是一样的，但是做法不同，它会以创建新的

commit 的方式来撤销 commit，这样能保留之前的 commit 历史，比较安

全。另外，同样因为可能会覆盖本地的修改，所以执行这个指令之前，

你需要 stash 或者 commit 暂存区和工作区的更改。

然后，从文件层面来说：

o

gitreset 只是把文件从历史记录区拿到暂存区，不影响工作区的内容，而

且不支持 --mixed、--soft 和 --hard。

o

o

gitcheckout 则是把文件从历史记录拿到工作区，不影响暂存区的内容。

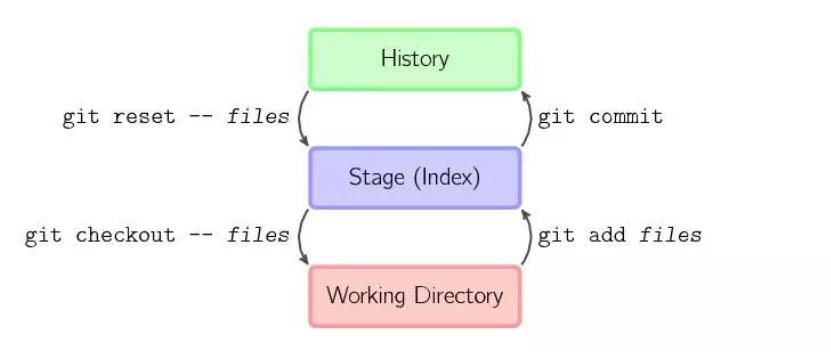
gitrevert 不支持文件层面的操作。

版权声明：本文为取经猿作者的原创文章，转载请附上原文出处链接及本声明

原文链接：https://www.teaspect.com/detail/5623?pn=21

 webpack和gulp区别（模块化与流的区别）

150



参考回答：

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。

3.2|Vue 框架

 有使用过Vue吗？说说你对Vue的理解

参考回答：

Vue是一个构建数据驱动的渐进性框架，它的目标是通过API实现响应数据绑定和视图

更新。

 说说Vue的优缺点

参考回答：

优点：

1、数据驱动视图，对真实dom进行抽象出virtualdom（本质就是一个js对象），

并配合diff算法、响应式和观察者、异步队列等手段以最小代价更新dom，渲染

页面

2、组件化，组件用单文件的形式进行代码的组织编写，使得我们可以在一个文

件里编写html\css（scoped属性配置css隔离）\js 并且配合Vue-loader之后，支

持更强大的预处理器等功能

3、强大且丰富的API提供一系列的api能满足业务开发中各类需求

4、由于采用虚拟dom，让Vuessr先天就足

5、生命周期钩子函数，选项式的代码组织方式，写熟了还是蛮顺畅的，但仍然

有优化空间（Vue3composition-api）

6、生态好，社区活跃

缺点：

1、由于底层基于Object.defineProperty实现响应式，而这个api本身不支持IE8

及以下浏览器

2、csr的先天不足，首屏性能问题（白屏）

3、由于百度等搜索引擎爬虫无法爬取js中的内容，故spa先天就对seo优化心

有余力不足（谷歌的puppeteer就挺牛逼的，实现预渲染底层也是用到了这个工

具）

 Vue和React有什么不同？使用场景分别是什么？

151



参考回答：

1、Vue是完整一套由官方维护的框架，核心库主要有由尤雨溪大神独自维护，而React

是不要脸的书维护（很多库由社区维护），曾经一段时间很多人质疑Vue的后续维护性，

似乎这并不是问题。

2、Vue上手简单，进阶式框架，白话说你可以学一点，就可以在你项目中去用一点，

你不一定需要一次性学习整个Vue才能去使用它，而React，恐怕如果你这样会面对项

目束手无策。

3、语法上Vue并不限制你必须es6+完全js形式编写页面，可以视图和js逻辑尽可能分

离，减少很多人看不惯React-jsx的恶心嵌套，毕竟都是作为前端开发者，还是更习惯

于html干净。

4、很多人说React适合大型项目，适合什么什么，Vue轻量级，适合移动端中小型项目，

其实我想说，说这话的人是心里根本没点逼数，Vue完全可以应对复杂的大型应用，甚

至于说如果你React学的不是很好，写出来的东西或根本不如Vue写的，毕竟Vue跟着

官方文档撸就行，自有人帮你规范，而React比较懒散自由，可以自由发挥

5、Vue在国内人气明显胜过React，这很大程度上得益于它的很多语法包括编程思维更

符合国人思想。

 什么是虚拟DOM？

参考回答：

虚拟 dom 是相对于浏览器所渲染出来的真实 dom 的，在react，vue等技术出现之前，

我们要改变页面展示的内容只能通过遍历查询 dom 树的方式找到需要修改的 dom 然

后修改样式行为或者结构，来达到更新 ui 的目的。

这种方式相当消耗计算资源，因为每次查询 dom 几乎都需要遍历整颗 dom 树，如果

建立一个与 dom 树对应的虚拟 dom 对象（ js 对象），以对象嵌套的方式来表示 dom

树，那么每次 dom 的更改就变成了 js 对象的属性的更改，这样一来就能查找 js 对象

的属性变化要比查询 dom 树的性能开销小。

 请描述下vue的生命周期是什么？

参考回答：

o

生命周期就是vue从开始创建到销毁的过程，分为四大步（创建，挂载，

更新，销毁），每一步又分为两小步，如beforeCreate，created。beforeCreate

前，也就是newVue的时候会初始化事件和生命周期；beforeCreate和

created之间会挂载Data，绑定事件；接下来会根据el挂载页面元素，如

果没有设置el则生命周期结束，直到手动挂载；el挂载结束后，根据

templete/outerHTML(el)渲染页面；在beforeMount前虚拟DOM已经创建完

成；之后在mounted前，将vm.$el替换掉页面元素el;mounted将虚拟dom

挂载到真实页面（此时页面已经全部渲染完成）；之后发生数据变化时

触发beforeUpdate和updated进行一些操作；最后主动调用销毁函数或者

组件自动销毁时beforeDestroy，手动撤销监听事件，计时器等；destroyed

时仅存在Dom节点，其他所有东西已自动销毁。这就是我所理解的vue

的一个完整的生命周期。

152

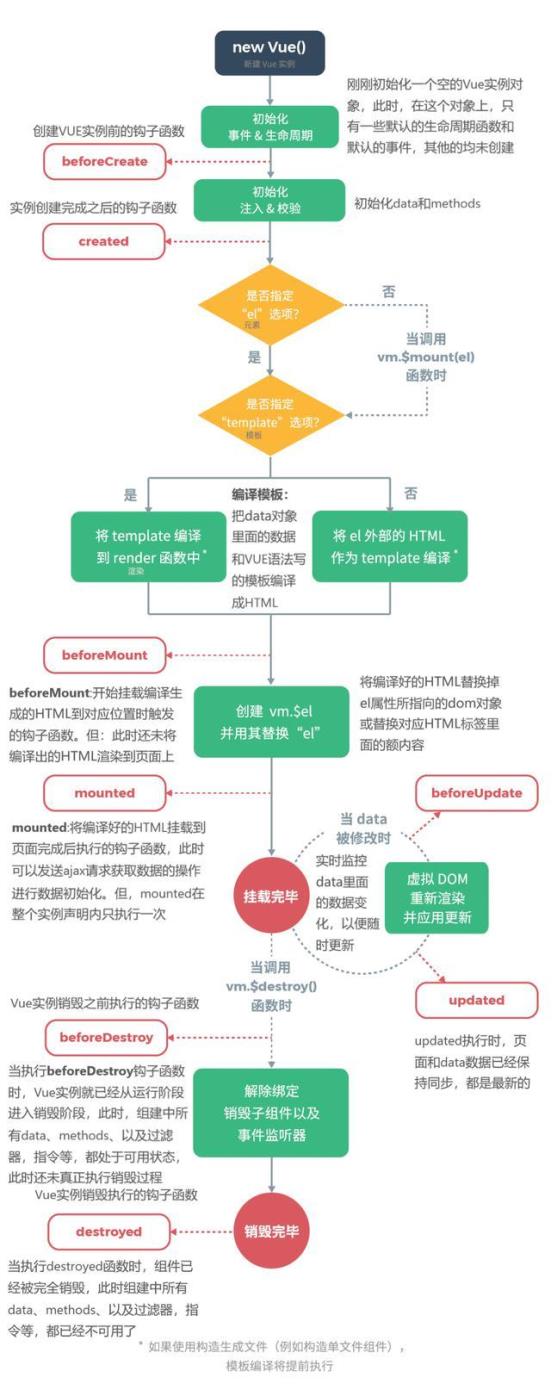


o

153



154



 vue如何监听键盘事件？

参考回答：

1. @keyup.方法

<template>

<inputref="myInput"type="text"value="helloworld"autofocus

@keyup.enter="handleKey">

</template>

<script>

exportdefault{

methods:{

handleKey(e){

console.log(e)

}

}

}

</script>

2.

addEventListener

<script>

exportdefault{

mounted(){

document.addEventListener('keyup',this.handleKey)

},

beforeDestroy(){

document.removeEventListener('keyup',this.handleKey)

},

methods:{

handleKey(e){

console.log(e)

}

}

}

</script><script>

exportdefault{

mounted(){

document.addEventListener('keyup',this.handleKey)

},

beforeDestroy(){

document.removeEventListener('keyup',this.handleKey)

},

methods:{

handleKey(e){

155



console.log(e)

}

}

}

</script>

 watch怎么深度监听对象变化

参考回答：

deep设置为true 就可以监听到对象的变化

letvm=newVue({

el:"#first",

data:{msg:{name:'北京'}},

watch:{

msg:{

handler(newMsg,oldMsg){

console.log(newMsg);

},

immediate:true,

deep:true

}

}

})

 删除数组用delete和Vue.delete有什么区别？

参考回答：





delete：只是被删除数组成员变为 empty/undefined，其他元素键值不变

Vue.delete：直接删了数组成员，并改变了数组的键值（对象是响应式的，确保

删除能触发更新视图，这个方法主要用于避开 Vue 不能检测到属性被删除的限

制）

 watch和计算属性有什么区别？

参考回答：

通俗来讲，既能用computed 实现又可以用 watch 监听来实现的功能，推荐用 computed，

重点在于computed 的缓存功能

computed计算属性是用来声明式的描述一个值依赖了其它的值，当所依赖的值或者变量

改变时，计算属性也会跟着改变；

watch 监听的是已经在 data 中定义的变量，当该变量变化时，会触发 watch 中的方法。

156



 Vue双向绑定原理

参考回答：

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)。

 v-model是什么？有什么用呢？

参考回答：

一则语法糖，相当于v-bind:value="xxx" 和 @input，意思是绑定了一个value属性的值，

子组件可对value属性监听，通过$emit('input',xxx)的方式给父组件通讯。自己实现

v-model方式的组件也是这样的思路。

 axios是什么？怎样使用它？怎么解决跨域的问题？

参考回答：

axios 的是一种异步请求，用法和ajax类似，安装npminstallaxios--save 即可使用，请

求中包括get,post,put,patch,delete等五种请求方式，解决跨域可以在请求头中添加

Access-Control-Allow-Origin，也可以在index.js文件中更改proxyTable配置等解决跨域

问题。

 在vue项目中如何引入第三方库（比如jQuery）？有哪些方法可以做到？

参考回答：

1、绝对路径直接引入

在index.html中用script引入

<scriptsrc="./static/jquery-1.12.4.js"></script>

然后在webpack中配置external

externals:{'jquery':'jQuery'}

在组件中使用时import

import$from'jquery'

2 、在webpack中配置alias

resolve:{extensions:['.js','.vue','.json'],alias:{'@':resolve('src'),'jquery':

resolve('static/jquery-1.12.4.js')}}

然后在组件中import

3、在webpack中配置plugins

plugins:[newwebpack.ProvidePlugin({$:'jquery'})]

全局使用，但在使用eslint情况下会报错，需要在使用了 $ 的代码前添加 /\*

eslint-disable\*/ 来去掉 ESLint 的检查。

 说说VueReactangularjsjquery的区别

157



参考回答：

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



Vue3.0 里为什么要用 ProxyAPI 替代 definePropertyAPI？

参考回答：

响应式优化。

a.definePropertyAPI 的局限性最大原因是它只能针对单例属性做监听。

Vue2.x中的响应式实现正是基于defineProperty中的descriptor，对 data 中的属性做了遍

历 + 递归，为每个属性设置了 getter、setter。

这也就是为什么 Vue 只能对 data 中预定义过的属性做出响应的原因，在 Vue中使用

下标的方式直接修改属性的值或者添加一个预先不存在的对象属性是无法做到setter监

听的，这是defineProperty的局限性。

b.ProxyAPI的监听是针对一个对象的，那么对这个对象的所有操作会进入监听操作，这

就完全可以代理所有属性，将会带来很大的性能提升和更优的代码。

Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须

先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。

c. 响应式是惰性的

在 Vue.js2.x 中，对于一个深层属性嵌套的对象，要劫持它内部深层次的变化，就需要

递归遍历这个对象，执行 Object.defineProperty 把每一层对象数据都变成响应式的，这

无疑会有很大的性能消耗。

在 Vue.js3.0 中，使用 ProxyAPI 并不能监听到对象内部深层次的属性变化，因此它的

处理方式是在 getter 中去递归响应式，这样的好处是真正访问到的内部属性才会变成响

应式，简单的可以说是按需实现响应式，减少性能消耗。

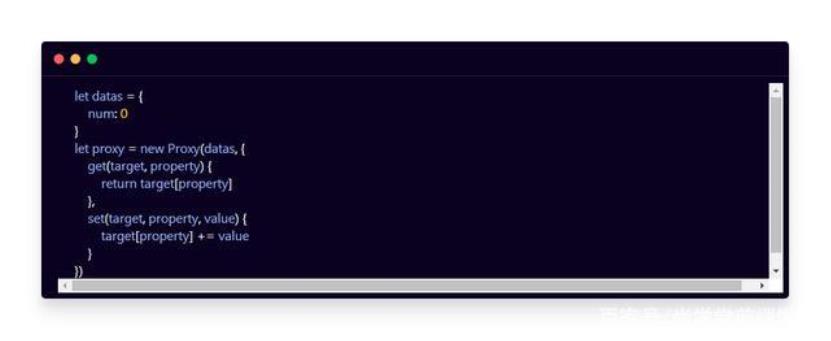
基础用法：



Vue3.0 编译做了哪些优化？

参考回答：

158



a. 生成 Blocktree

Vue.js2.x 的数据更新并触发重新渲染的粒度是组件级的，单个组件内部 需要遍历该组

件的整个 vnode 树。在2.0里，渲染效率的快慢与组件大小成正相关：组件越大，渲染

效率越慢。并且，对于一些静态节点，又无数据更新，这些遍历都是性能浪费。

Vue.js3.0 做到了通过编译阶段对静态模板的分析，编译生成了 Blocktree。Blocktree

是一个将模版基于动态节点指令切割的嵌套区块，每个 区块内部的节点结构是固定的，

每个区块只需要追踪自身包含的动态节点。所以，在3.0里，渲染效率不再与模板大小

成正相关，而是与模板中动态节点的数量成正相关。

b.slot 编译优化

Vue.js2.x 中，如果有一个组件传入了slot，那么每次父组件更新的时候，会强制使子组

件update，造成性能的浪费。

Vue.js3.0 优化了slot的生成，使得非动态 slot中属性的更新只会触发子组件的更新。

动态slot指的是在slot上面使用v-if，v-for，动态slot名字等会导致slot产生运行时动

态变化但是又无法被子组件track的操作。

c.diff算法优化



Vue3.0新特性 ——CompositionAPI 与 React.js 中 Hooks的异同点

参考回答：

a.React.js 中的 Hooks 基本使用

ReactHooks 允许你 "勾入" 诸如组件状态和副作用处理等 React 功能中。Hooks 只能

用在函数组件中，并允许我们在不需要创建类的情况下将状态、副作用处理和更多东西

带入组件中。

React 核心团队奉上的采纳策略是不反对类组件，所以你可以升级 React 版本、在新组

件中开始尝试 Hooks，并保持既有组件不做任何更改。

案例：

159



useState 和 useEffect 是 ReactHooks 中的一些例子，使得函数组件中也能增加状态和

运行副作用。

我们也可以自定义一个Hooks，它打开了代码复用性和扩展性的新大门。

b.VueCompositionAPI 基本使用

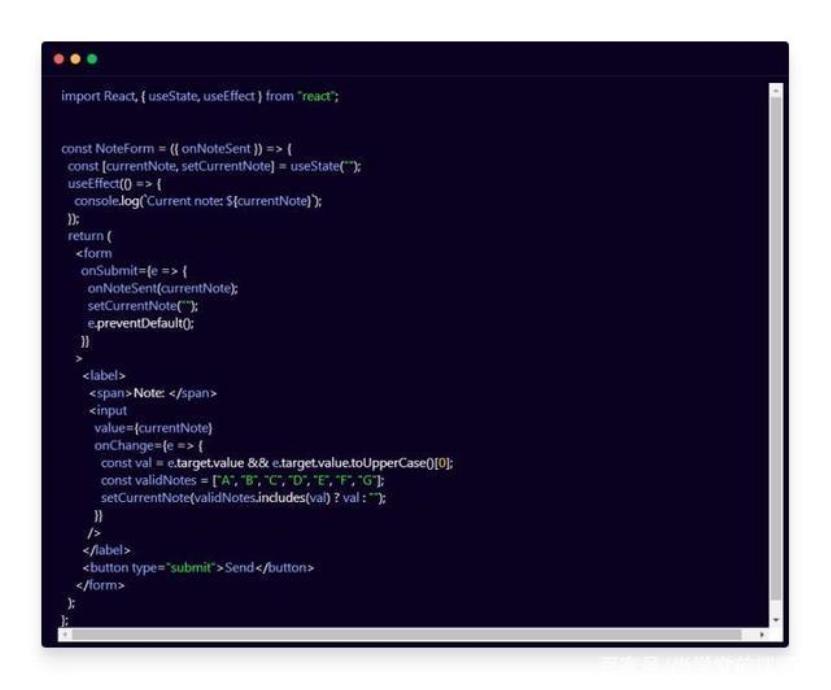
VueCompositionAPI 围绕一个新的组件选项 setup 而创建。setup() 为 Vue 组件提供了

状态、计算值、watcher 和生命周期钩子。

并没有让原来的 API（Options-basedAPI）消失。允许开发者 结合使用新旧两种 API

（向下兼容）。

160



c. 原理

Reacthook 底层是基于链表实现，调用的条件是每次组件被render的时候都会顺序执行

所有的hooks。

Vuehook 只会被注册调用一次，Vue 能避开这些麻烦的问题，原因在于它对数据的响

应是基于proxy的，对数据直接代理观察。（这种场景下，只要任何一个更改data的地

方，相关的function或者template都会被重新计算，因此避开了 React可能遇到的性能

上的问题）。

React 中，数据更改的时候，会导致重新 render，重新render又会重新把hooks重新注

册一次，所以React复杂程度会高一些。

 Vue3.0是如何变得更快的？（底层，源码）

参考回答：

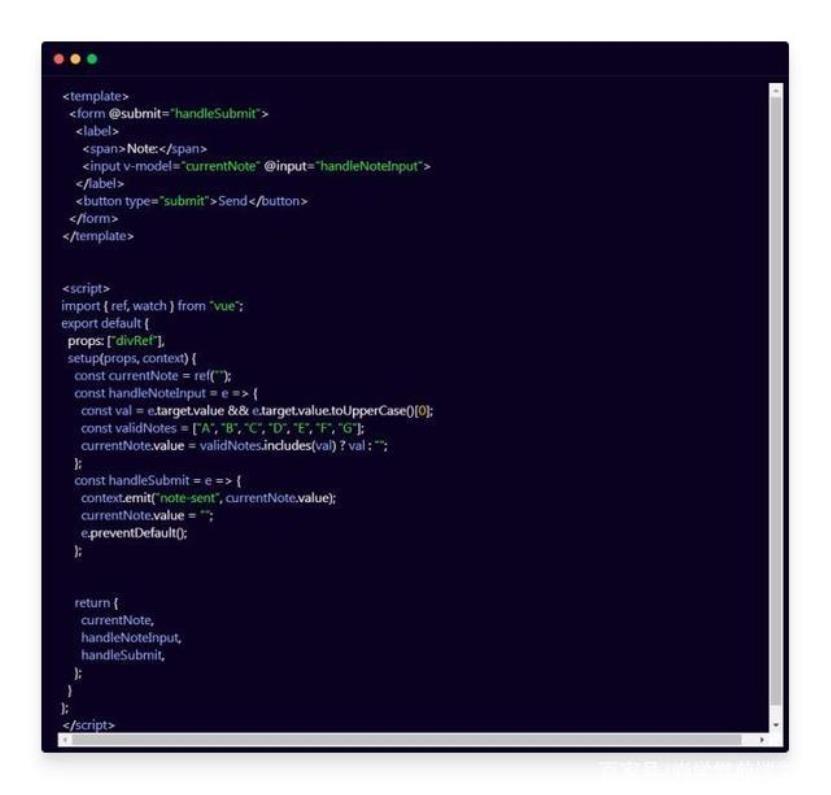
a.diff方法优化

Vue2.x 中的虚拟dom是进行全量的对比。

Vue3.0 中新增了静态标记（PatchFlag）：在与上次虚拟结点进行对比的时候，值对比

带有patchflag的节点，并且可以通过flag 的信息得知当前节点要对比的具体内容化。

161



b.hoistStatic 静态提升

Vue2.x: 无论元素是否参与更新，每次都会重新创建。

Vue3.0: 对不参与更新的元素，只会被创建一次，之后会在每次渲染时候被不停的复用。

c.cacheHandlers 事件侦听器缓存

默认情况下onClick会被视为动态绑定，所以每次都会去追踪它的变化但是因为是同一

个函数，所以没有追踪变化，直接缓存起来复用即可。

原作者姓名： 欧阳呀

 vue要做权限管理该怎么做？如果控制到按钮级别的权限怎么做？

参考回答：

https://panjiachen.github.io/vue-element-admin-site/zh/guide/essentials/permission.html#%E6

%8C%87%E4%BB%A4%E6%9D%83%E9%99%90

 vue在created和mounted这两个生命周期中请求数据有什么区别呢？

参考回答：

看实际情况，一般在created（或beforeRouter）里面就可以，如果涉及到需要页面加载

完成之后的话就用mounted。

在created的时候，视图中的html并没有渲染出来，所以此时如果直接去操作html的dom

节点，一定找不到相关的元素

而在mounted中，由于此时html已经渲染出来了，所以可以直接操作dom节点，（此

时document.getelementById 即可生效了）。

 说说你对proxy的理解

参考回答：

vue的数据劫持有两个缺点:

1、无法监听通过索引修改数组的值的变化

2、无法监听object也就是对象的值的变化

所以vue2.x中才会有$set属性的存在

proxy是es6中推出的新api，可以弥补以上两个缺点，所以vue3.x版本用proxy替换

object.defineproperty。

3.3|React 框架

 angularJs和React区别

参考回 ：

162



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 headersectionfooterslider)

 redux中间件

参考回答：

中间件提供第三方插件的模式，自定义拦截 action->reducer 的过程。变为 action->

middlewares->reducer 。这种机制可以让我们改变数据流，实现如异步 action ，action 过

滤，日志输出，异常报告等功能。

常见的中间件：redux-logger：提供日志输出；redux-thunk：处理异步操作；redux-promise：

处理异步操作；actionCreator的返回值是promise

 redux有什么缺点

参考回答：

1.一个组件所需要的数据，必须由父组件传过来，而不能像flux中直接从store取。

2.当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新

render，可能会有效率影响，或者需要写复杂的shouldComponentUpdate进行判断。

 React组件的划分业务组件技术组件？

参考回答：

根据组件的职责通常把组件分为UI组件和容器组件。UI 组件负责 UI 的呈现，容器组

件负责管理数据和逻辑。两者通过React-Redux 提供connect方法联系起来。

 React生命周期函数

参考回答：

一、初始化阶段：

getDefaultProps:获取实例的默认属性

getInitialState:获取每个实例的初始化状态

componentWillMount：组件即将被装载、渲染到页面上

render:组件在这里生成虚拟的DOM节点

163



componentDidMount:组件真正在被装载之后

二、运行中状态：

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

shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以返回false，接收数据

后不更新，阻止render调用，后面的函数不会被继续执行了）

componentWillUpdate:组件即将更新不能修改属性和状态

render:组件重新描绘

componentDidUpdate:组件已经更新

三、销毁阶段：

componentWillUnmount:组件即将销毁

 React性能优化是哪个周期函数？

参考回答：

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domdiff算法，可以极大的提高性能。

 为什么虚拟dom会提高性能?

参考回答：

虚拟dom相当于在js和真实dom中间加了一个缓存，利用domdiff算法避免了没有必要

的dom操作，从而提高性能。

具体实现步骤如下：

1.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；

2.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2所记录的差异应用到步骤1所构建的真正的DOM树上，视图就更新了。

164



 diff算法?

参考回答：

1.把树形结构按照层级分解，只比较同级元素。

2.给列表结构的每个单元添加唯一的key属性，方便比较。

3.React 只会匹配相同 class 的 component（这里面的class指的是组件的名字）

4.合并操作，调用 component 的 setState 方法的时候,React 将其标记为 dirty.到每一个

事件循环结束,React 检查所有标记 dirty 的 component 重新绘制.

6．选择性子树渲染。开发人员可以重写shouldComponentUpdate提高diff的性能。

 React性能优化方案

参考回答：

1）重写shouldComponentUpdate来避免不必要的dom操作。

2）使用 production 版本的React.js。

3）使用key来帮助React识别列表中所有子组件的最小变化

 简述flux 思想

参考回答：

Flux 的最大特点，就是数据的"单向流动"。

1.用户访问 View

2.View 发出用户的 Action

3.Dispatcher 收到 Action，要求 Store 进行相应的更新

4.Store 更新后，发出一个"change"事件

5.View 收到"change"事件后，更新页面

 React项目用过什么脚手架？Mern?Yeoman?

参考回答：

165



Mern：MERN是脚手架的工具，它可以很容易地使用Mongo,Express,ReactandNodeJS

生成同构JS应用。它最大限度地减少安装时间，并得到您使用的成熟技术来加速开发。

 你了解React吗？

参考回答：

了解，React是facebook搞出来的一个轻量级的组件库，用于解决前端视图层的一些问

题，就是MVC中V层的问题，它内部的Instagram网站就是用React搭建的。

 React解决了什么问题？

参考回答：

解决了三个问题： 1.组件复用问题， 2.性能问题，3.兼容性问题：

 React的协议？

参考回答：

React遵循的协议是“BSD许可证 + 专利开源协议”，这个协议比较奇葩，如果你的产

品跟facebook没有竞争关系，你可以自由的使用React，但是如果有竞争关系，你的React

的使用许可将会被取消

 了解shouldComponentUpdate吗？

参考回答：

React虚拟dom技术要求不断的将dom和虚拟dom进行diff比较，如果dom树比价大，

这种比较操作会比较耗时，因此React提供了shouldComponentUpdate这种补丁函数，如

果对于一些变化，如果我们不希望某个组件刷新，或者刷新后跟原来其实一样，就可以

使用这个函数直接告诉React，省去diff操作，进一步的提高了效率。

 React 的工作原理？

参考回答：

React 会创建一个虚拟 DOM(virtualDOM)。当一个组件中的状态改变时，React 首先会

通过 "diffing" 算法来标记虚拟 DOM 中的改变，第二步是调节(reconciliation)，会用 diff

的结果来更新 DOM。

 使用 React 有何优点？

参考回答：

1.只需查看 render 函数就会很容易知道一个组件是如何被渲染的

166



2.JSX 的引入，使得组件的代码更加可读，也更容易看懂组件的布局，或者组件之间是

如何互相引用的

3.支持服务端渲染，这可以改进 SEO 和性能

4.易于测试

5.React 只关注 View 层，所以可以和其它任何框架(如Backbone.js,Angular.js)一起使用

 展示组件(Presentationalcomponent)和容器组件(Containercomponent)之间有

何不同？

参考回答：

1.展示组件关心组件看起来是什么。展示专门通过 props 接受数据和回调，并且几乎不

会有自身的状态，但当展示组件拥有自身的状态时，通常也只关心 UI 状态而不是数据

的状态。

2.容器组件则更关心组件是如何运作的。容器组件会为展示组件或者其它容器组件提供

数据和行为(behavior)，它们会调用 Fluxactions，并将其作为回调提供给展示组件。容器

组件经常是有状态的，因为它们是(其它组件的)数据源

 类组件(Classcomponent)和函数式组件(Functionalcomponent)之间有何不

同？

参考回答：

1.类组件不仅允许你使用更多额外的功能，如组件自身的状态和生命周期钩子，也能使

组件直接访问 store 并维持状态

2.当组件仅是接收 props，并将组件自身渲染到页面时，该组件就是一个 '无状态组件

(statelesscomponent)'，可以使用一个纯函数来创建这样的组件。这种组件也被称为哑组

件(dumbcomponents)或展示组件

 (组件的)状态(state)和属性(props)之间有何不同？

参考回答：

1.State 是一种数据结构，用于组件挂载时所需数据的默认值。State 可能会随着时间的

推移而发生突变，但多数时候是作为用户事件行为的结果。

2.Props(properties 的简写)则是组件的配置。props 由父组件传递给子组件，并且就子组

件而言，props 是不可变的(immutable)。组件不能改变自身的 props，但是可以把其子组

167



件的 props 放在一起(统一管理)。Props 也不仅仅是数据--回调函数也可以通过 props

传递。

 应该在 React 组件的何处发起 Ajax 请求？

参考回答：

在 React 组件中，应该在 componentDidMount 中发起网络请求。这个方法会在组件第

一次“挂载”(被添加到 DOM)时执行，在组件的生命周期中仅会执行一次。更重要的是，

你不能保证在组件挂载之前 Ajax 请求已经完成，如果是这样，也就意味着你将尝试在

一个未挂载的组件上调用 setState，这将不起作用。在 componentDidMount 中发起网络

请求将保证这有一个组件可以更新了。

 在 React 中，refs 的作用是什么？

参考回答：

Refs 可以用于获取一个 DOM 节点或者 React 组件的引用。何时使用 refs 的好的示例

有管理焦点/文本选择，触发命令动画，或者和第三方 DOM 库集成。你应该避免使用

String 类型的 Refs 和内联的 ref 回调。Refs 回调是 React 所推荐的。

 何为高阶组件(higherordercomponent)？

参考回答：

高阶组件是一个以组件为参数并返回一个新组件的函数。HOC 运行你重用代码、逻辑

和引导抽象。最常见的可能是 Redux 的 connect 函数。除了简单分享工具库和简单的

组合，HOC最好的方式是共享 React 组件之间的行为。如果你发现你在不同的地方写

了大量代码来做同一件事时，就应该考虑将代码重构为可重用的 HOC。

 使用箭头函数(arrowfunctions)的优点是什么？

参考回答：

1. 作用域安全：在箭头函数之前，每一个新创建的函数都有定义自身的 this 值(在构造

函数中是新对象；在严格模式下，函数调用中的 this 是未定义的；如果函数被称为“对

象方法”，则为基础对象等)，但箭头函数不会，它会使用封闭执行上下文的 this 值。

2. 简单：箭头函数易于阅读和书写

3. 清晰：当一切都是一个箭头函数，任何常规函数都可以立即用于定义作用域。开发者

总是可以查找 next-higher 函数语句，以查看 this 的值

 为什么建议传递给 setState 的参数是一个 callback 而不是一个对象？

168



参考回答：

因为 this.props 和 this.state 的更新可能是异步的，不能依赖它们的值去计算下一个

state。

 除了在构造函数中绑定 this，还有其它方式吗？

参考回答：

可以使用属性初始值设定项(propertyinitializers)来正确绑定回调，create-React-app 也是

默认支持的。在回调中你可以使用箭头函数，但问题是每次组件渲染时都会创建一个新

的回调。

 怎么阻止组件的渲染？

参考回答：

在组件的 render 方法中返回 null 并不会影响触发组件的生命周期方法

 当渲染一个列表时，何为 key？设置 key 的目的是什么？

参考回答：

Keys 会有助于 React 识别哪些 items 改变了，被添加了或者被移除了。Keys 应该被赋

予数组内的元素以赋予(DOM)元素一个稳定的标识，选择一个 key 的最佳方法是使用一

个字符串，该字符串能惟一地标识一个列表项。很多时候你会使用数据中的 IDs 作为

keys，当你没有稳定的 IDs 用于被渲染的 items 时，可以使用项目索引作为渲染项的

key，但这种方式并不推荐，如果 items 可以重新排序，就会导致 re-render 变慢



(在构造函数中)调用 super(props) 的目的是什么？

参考回答：

在 super() 被调用之前，子类是不能使用 this 的，在 ES2015 中，子类必须在 constructor

中调用 super()。传递 props 给 super() 的原因则是便于(在子类中)能在 constructor 访问

this.props。

 何为 JSX ？

参考回答：

JSX 是 JavaScript 语法的一种语法扩展，并拥有 JavaScript 的全部功能。JSX 生产

React"元素"，你可以将任何的 JavaScript 表达式封装在花括号里，然后将其嵌入到 JSX

中。在编译完成之后，JSX 表达式就变成了常规的 JavaScript 对象，这意味着你可以在

if 语句和 for 循环内部使用 JSX，将它赋值给变量，接受它作为参数，并从函数中返回

它。

169



3.4|Angular 框架

 Angular中组件之间通信的方式

参考回答：

Propsdown

1.调用子组件,通过自定义属性传值

2.子组件内部通过Input来接收属性的值

Events up

1.在父组件中定义一个有参数的方法

2.调用子组件时,绑定自定义事件和上一步方法

3.子组件内部通过Output和EventEmitter来触发事件并传值.

 Angualr的八大组成部分并简单描述

参考回答：

model是Angular开发中的基本单位,是一个容器,可以包含组件、指令、管道等

Components是可被反复使用的带有特定功能的视图

Templates 是经过指令和管道、组件等增强过的html

Bindings

结合着事件绑定 属性绑定 双向数据绑定等扩展html的功能

Directives分为结构性和属性型指令还有其他模块中比如路由模块中的指令等,

主要是增强html.

Pipes

可以对数据做一个筛选、过滤、格式化从而得到目的数据

Service 将组件、应用中的可共用的部分,比如数据或者方法等 封装成服务以方便服用

DependencyInjection 依赖注入

 Angular中常见的生命周期的钩子函数?

参考回答：

ngOnChanges:当Angular设置其接收当前和上一个对象值的数据绑定属性时响应。

ngOnInit:在第一个ngOnChange触发器之后,初始化组件/指令。这是最常用的方法,

用于从后端服务检索模板的数据。

ngDoCheck：检测并在Angular上下文发生变化时执行。

每次更改检测运行时,会被调用。

ngOnDestroy:在Angular销毁指令/组件之前消除。取消订阅可观察的对象并脱离

事件处理程序,以避免内存泄漏。

组件特定的hooks:

ngAfterContentInit:组件内容已初始化完成

ngAfterContentChecked:在Angular检查投影到其视图中的绑定的外部内容之后。

ngAfterViewInit:Angular创建组件的视图后。

n AfterViewChecked： An ular

查组

图

之

170



 Angular中路由的工作原理

参考回答：

Angular应用程序具有路由器服务的单个实例,并且每当URL改变时,相应的路由就与路

由配置数组

进行匹配。在成功匹配时,它会应用重定向,此时路由器会构建ActivatedRoute对象的树,

同时包含路由器的当前状态。在重定向之前,路由器将通过运行保护(CanActivate)

来检查是否允许新的状态。RouteGuard只是路由器运行来检查路由授权的接口方法。

保护运行后,它将解析路由数据并通过将所需的组件实例化到

<router-outlet></router-outlet>

来激活路由器状态。

 解释rjx在Angular中的使用场景

参考回答：

Rxjs是在微软所提供的一种的异步处理数据的方式,在Angular中处理网络通信时用到了。

创建一个Observable并subsribe

比如:this.http.get('').subscribe((data)=>{})

171

