第⼀部分：基础篇

⼀、HTML、HTTP、web综合问题

1. 前端需要注意哪些SEO

合理的 title 、 description 、 keywords ：搜索对着三项的权重逐个减⼩， title 值强调重点即可，重要关键词出现不要超过2次，⽽且要靠前，不同⻚⾯ title 要有所不同； description 把⻚⾯内容⾼度概括，⻓度合适，不可过分堆砌关键词，不同⻚⾯description 有所不同； keywords 列举出重要关键词即可

语义化的 HTML 代码，符合W3C规范：语义化代码让搜索引擎容易理解⽹⻚

重要内容 HTML 代码放在最前：搜索引擎抓取 HTML 顺序是从上到下，有的搜索引擎对抓取⻓度有限制，保证重要内容⼀定会被抓取

重要内容不要⽤ js 输出：爬⾍不会执⾏js获取内容少⽤ iframe ：搜索引擎不会抓取 iframe 中的内容

⾮装饰性图⽚必须加 alt

###### 提⾼⽹站速度：⽹站速度是搜索引擎排序的⼀个重要指标

1. **<img>** 的 **title** 和 **alt** 有什么区别

###### 通常当⿏标滑动到元素上的时候显示

alt 是 <img> 的特有属性，是图⽚内容的等价描述，⽤于图⽚⽆法加载时显示、读屏器阅读图⽚。可提图⽚⾼可访问性，除了纯装饰图⽚外都必须设置有意义的值，搜索引擎会 重点分析。

1. HTTP的⼏种请求⽅法⽤途

GET ⽅ 法

发送⼀个请求来取得服务器上的某⼀资源

POST ⽅ 法

###### 向 URL 指定的资源提交数据或附加新的数据

PUT ⽅ 法

跟 POST ⽅法很像，也是想服务器提交数据。但是，它们之间有不同。 PUT 指定了资源在服务器上的位置，⽽ POST 没有

HEAD ⽅ 法

###### 只请求⻚⾯的⾸部

DELETE ⽅ 法

###### 删除服务器上的某资源

OPTIONS ⽅ 法

它⽤于获取当前 URL 所⽀持的⽅法。如果请求成功，会有⼀个 Allow 的头包含类似 “GET,POST” 这样的信息

TRACE ⽅ 法

TRACE ⽅法被⽤于激发⼀个远程的，应⽤层的请求消息回路

CONNECT ⽅ 法

把请求连接转换到透明的 TCP/IP 通道

#### 从浏览器地址栏输⼊url到显示⻚⾯的步骤

###### 基础版本

浏览器根据请求的 URL 交给 DNS 域名解析，找到真实 IP ，向服务器发起请求；

服务器交给后台处理完成后返回数据，浏览器接收⽂件（ HTML、JS、CSS 、图象等）； 浏览器对加载到的资源（ HTML、JS、CSS 等）进⾏语法解析，建⽴相应的内部数据结构

（ 如 HTML 的 DOM ）；

###### 载⼊解析到的资源⽂件，渲染⻚⾯，完成。详细版

1. 在浏览器地址栏输⼊URL

###### 浏览器查看缓存，如果请求资源在缓存中并且新鲜，跳转到转码步骤

* 1. 如果资源未缓存，发起新请求
  2. 如果已缓存，检验是否⾜够新鲜，⾜够新鲜直接提供给客户端，否则与服务器进⾏验 证。
  3. 检验新鲜通常有两个HTTP头进⾏控制 Expires 和 Cache-Control ： HTTP1.0提供Expires，值为⼀个绝对时间表示缓存新鲜⽇期

HTTP1.1增加了Cache-Control: max-age=,值为以秒为单位的最⼤新鲜时间

1. 浏览器解析URL获取协议，主机，端⼝，path
2. 浏览器组装⼀个HTTP（GET）请求报⽂
3. 浏览器获取主机ip地址，过程如下：

###### 浏览器缓存

* 1. 本机缓存
  2. hosts⽂件

###### 路由器缓存

* 1. ISP DNS缓存
  2. DNS递归查询（可能存在负载均衡导致每次IP不⼀样）

1. 打开⼀个socket与⽬标IP地址，端⼝建⽴TCP链接，三次握⼿如下：
   1. 客户端发送⼀个TCP的SYN=1，Seq=X的包到服务器端⼝
   2. 服务器发回SYN=1， ACK=X+1， Seq=Y的响应包
   3. 客户端发送ACK=Y+1， Seq=Z
2. TCP链接建⽴后发送HTTP请求
3. 服务器接受请求并解析，将请求转发到服务程序，如虚拟主机使⽤HTTP Host头部判断请求的服务程序
4. 服务器检查HTTP请求头是否包含缓存验证信息如果验证缓存新鲜，返回304等对应状态码

###### 处理程序读取完整请求并准备HTTP响应，可能需要查询数据库等操作

1. 服务器将响应报⽂通过TCP连接发送回浏览器
2. 浏览器接收HTTP响应，然后根据情况选择关闭TCP连接或者保留重⽤，关闭TCP连接的四次握⼿如下：
   1. 主动⽅发送Fin=1， Ack=Z， Seq= X报⽂
   2. 被动⽅发送ACK=X+1， Seq=Z报⽂
   3. 被动⽅发送Fin=1， ACK=X， Seq=Y报⽂
   4. 主动⽅发送ACK=Y， Seq=X报⽂
3. 浏览器检查响应状态吗：是否为1XX，3XX， 4XX， 5XX，这些情况处理与2XX不同

###### 如果资源可缓存，进⾏缓存

1. 对响应进⾏解码（例如gzip压缩）
2. 根据资源类型决定如何处理（假设资源为HTML⽂档）
3. 解析HTML⽂档，构件DOM树，下载资源，构造CSSOM树，执⾏js脚本，这些操作没有严格的先后顺序，以下分别解释
4. 构建DOM树：
   1. Tokenizing：根据HTML规范将字符流解析为标记
   2. Lexing：词法分析将标记转换为对象并定义属性和规则
   3. DOM construction：根据HTML标记关系将对象组成DOM树
5. 解析过程中遇到图⽚、样式表、js⽂件，启动下载
6. 构建CSSOM树：
   1. Tokenizing：字符流转换为标记流
   2. Node：根据标记创建节点
   3. CSSOM：节点创建CSSOM树
7. [根据DOM树和CSSOM树构建渲染树](https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction) :
   1. 从DOM树的根节点遍历所有可⻅节点，不可⻅节点包括：1） script , meta 这样本身不可⻅的标签。2)被css隐藏的节点，如 display: none
   2. 对每⼀个可⻅节点，找到恰当的CSSOM规则并应⽤

###### 发布可视节点的内容和计算样式

1. js解析如下：
   1. 浏览器创建Document对象并解析HTML，将解析到的元素和⽂本节点添加到⽂档中，此 时document.readystate为loading
   2. HTML解析器遇到没有async和defer的script时，将他们添加到⽂档中，然后执⾏⾏内或外部脚本。这些脚本会同步执⾏，并且在脚本下载和执⾏时解析器会暂停。这样就可 以⽤document.write()把⽂本插⼊到输⼊流中。同步脚本经常简单定义函数和注册事件 处理程序，他们可以遍历和操作script和他们之前的⽂档内容
   3. 当解析器遇到设置了async属性的script时，开始下载脚本并继续解析⽂档。脚本会在它下载完成后尽快执⾏，但是解析器不会停下来等它下载。异步脚本禁⽌使⽤document.write()，它们可以访问⾃⼰script和之前的⽂档元素
   4. 当⽂档完成解析，document.readState变成interactive
   5. 所有defer脚本会按照在⽂档出现的顺序执⾏，延迟脚本能访问完整⽂档树，禁⽌使⽤document.write()
   6. 浏览器在Document对象上触发DOMContentLoaded事件
   7. 此时⽂档完全解析完成，浏览器可能还在等待如图⽚等内容加载，等这些内容完成载⼊ 并且所有异步脚本完成载⼊和执⾏，document.readState变为complete，window触发load事件
2. 显示⻚⾯（HTML解析过程中会逐步显示⻚⾯） 详细简版

###### 从浏览器接收 url 到开启⽹络请求线程（这⼀部分可以展开浏览器的机制以及进程与线程之间的关系）

* 1. 开启⽹络线程到发出⼀个完整的 HTTP 请求（这⼀部分涉及到dns查询， TCP/IP 请求， 五层因特⽹协议栈等知识）

###### 从服务器接收到请求到对应后台接收到请求（这⼀部分可能涉及到负载均衡，安全拦截以 及后台内部的处理等等）

* 1. 后台和前台的 HTTP 交互（这⼀部分包括 HTTP 头部、响应码、报⽂结构、 cookie 等知识，可以提下静态资源的 cookie 优化，以及编码解码，如 gzip 压缩等）
  2. 单独拎出来的缓存问题， HTTP 的缓存（这部分包括http缓存头部， ETag ， catch- control 等 ）
  3. 浏览器接收到 HTTP 数据包后的解析流程（解析 html -词法分析然后解析成 dom 树、解析 css ⽣成 css 规则树、合并成 render 树，然后 layout 、 painting 渲染、复合图层的合成、 GPU 绘制、外链资源的处理、 loaded 和 DOMContentLoaded 等）
  4. CSS 的可视化格式模型（元素的渲染规则，如包含块，控制框， BFC ， IFC 等概念）
  5. JS 引擎解析过程（ JS 的解释阶段，预处理阶段，执⾏阶段⽣成执⾏上下⽂， VO ，作

###### ⽤域链、回收机制等等）

* 1. 其它（可以拓展不同的知识模块，如跨域，web安全， hybrid 模式等等内容）

#### 如何进⾏⽹站性能优化

content ⽅ ⾯

减少 HTTP 请求：合并⽂件、 CSS 精灵、 inline Image

减少 DNS 查询： DNS 缓存、将资源分布到恰当数量的主机名减少 DOM 元素数量

Server ⽅ ⾯

使 ⽤ CDN

配置 ETag

对组件使⽤ Gzip 压缩

Cookie ⽅ ⾯

减⼩ cookie ⼤⼩

css ⽅ ⾯

将样式表放到⻚⾯顶部不使⽤ CSS 表达式

使⽤ <link> 不使⽤ @import

Javascript ⽅ ⾯

###### 将脚本放到⻚⾯底部

将 javascript 和 css 从外部引⼊压 缩 javascript 和 css

###### 删除不需要的脚本

减少 DOM 访问

###### 图⽚⽅⾯

优化图⽚：根据实际颜⾊需要选择⾊深、压缩优化 css 精灵

不要在 HTML 中拉伸图⽚

1. HTTP状态码及其含义

1XX ：信息状态码

100 Continue 继续，⼀般在发送 post 请求时，已发送了 http header 之后服务端将返回此信息，表示确认，之后发送具体参数信息

2XX ：成功状态码

1. OK 正常返回信息
2. Created 请求成功并且服务器创建了新的资源
3. Accepted 服务器已接受请求，但尚未处理

3XX ：重定向

1. Moved Permanently 请求的⽹⻚已永久移动到新位置。
2. Found 临时性重定向。
3. See Other 临时性重定向，且总是使⽤ GET 请求新的 URI 。
4. Not Modified ⾃从上次请求后，请求的⽹⻚未修改过。

4XX ：客户端错误

1. Bad Request 服务器⽆法理解请求的格式，客户端不应当尝试再次使⽤相同的内容发起请求。
2. Unauthorized 请求未授权。

403 Forbidden 禁⽌访问。

404 Not Found 找不到如何与 URI 相匹配的资源。

5XX: 服务器错误

500 Internal Server Error 最常⻅的服务器端错误。

503 Service Unavailable 服务器端暂时⽆法处理请求（可能是过载或维护）。

#### 语义化的理解

###### ⽤正确的标签做正确的事情！

HTML 语义化就是让⻚⾯的内容结构化，便于对浏览器、搜索引擎解析； 在没有样式 CSS 情况下也以⼀种⽂档格式显示，并且是容易阅读的。

###### 搜索引擎的爬⾍依赖于标记来确定上下⽂和各个关键字的权重，利于 SEO 。使阅读源代码的⼈对⽹站更容易将⽹站分块，便于阅读维护理解

1. 介绍⼀下你对浏览器内核的理解？

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

渲染引擎：负责取得⽹⻚的内容（ HTML 、 XML 、图像等等）、整理讯息（例如加⼊

###### CSS 等），以及计算⽹⻚的显示⽅式，然后会输出⾄显示器或打印机。浏览器的内核的不同对于⽹⻚的语法解释会有不同，所以渲染的效果也不相同。所有⽹⻚浏览器、电⼦邮件 客户端以及其它需要编辑、显示⽹络内容的应⽤程序都需要内核

JS 引擎则：解析和执⾏ javascript 来实现⽹⻚的动态效果

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

1. html5有哪些新特性、移除了那些元素？

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

绘 画 canvas

⽤于媒介回放的 video 和 audio 元素

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

sessionStorage 的数据在浏览器关闭后⾃动删除

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

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

新 的 技 术 webworker 、 websocket 、 Geolocation

###### 移除的元素：

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

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

⽀持 HTML5 新标签：

IE8/IE7/IE6 ⽀持通过 document.createElement ⽅法产⽣的标签可以利⽤这⼀特性让这些浏览器⽀持 HTML5 新标签

###### 浏览器⽀持新标签后，还需要添加标签默认的样式

当然也可以直接使⽤成熟的框架、⽐如 html5shim

1. **HTML5** 的离线储存怎么使⽤，⼯作原理能不能解释⼀下？

###### 在⽤户没有与因特⽹连接时，可以正常访问站点或应⽤，在⽤户与因特⽹连接时，更新⽤ 户机器上的缓存⽂件

原理： HTML5 的离线存储是基于⼀个新建的 .appcache ⽂件的缓存机制(不是存储技

###### 术)，通过这个⽂件上的解析清单离线存储资源，这些资源就会像 cookie ⼀样被存储了下来。之后当⽹络在处于离线状态下时，浏览器会通过被离线存储的数据进⾏⻚⾯展示

如何使⽤：

⻚⾯头部像下⾯⼀样加⼊⼀个 manifest 的属性； 在 cache.manifest ⽂件的编写离线存储的资源

在离线状态时，操作 window.applicationCache 进⾏需求实现

json

CACHE MANIFEST #v0.11

CACHE:

js/app.js css/style.css NETWORK:

resourse/logo.png FALLBACK:

/offline.html

1. 浏览器是怎么对 **HTML5** 的离线储存资源进⾏管理和加载的呢

在线的情况下，浏览器发现 html 头部有 manifest 属性，它会请求 manifest ⽂件，如果是第⼀次访问 app ，那么浏览器就会根据manifest⽂件的内容下载相应的资源并且进⾏离线存储。如果已经访问过 app 并且资源已经离线存储了，那么浏览器就会使⽤离线的资源加载⻚⾯，然后浏览器会对⽐新的 manifest ⽂件与旧的 manifest ⽂件，如果⽂件没有发⽣改变，就不做任何操作，如果⽂件改变了，那么就会重新下载⽂件中的资源并进⾏ 离线存储。

###### 离线的情况下，浏览器就直接使⽤离线存储的资源。

1. 请描述⼀下 **cookies** ， **sessionStorage** 和 **localStorage** 的区别？

cookie 是⽹站为了标示⽤户身份⽽储存在⽤户本地终端（Client Side）上的数据（通常经过加密）

cookie数据始终在同源的http请求中携带（即使不需要），记会在浏览器和服务器间来回 传递

sessionStorage 和 localStorage 不会⾃动把数据发给服务器，仅在本地保存

###### 存储⼤⼩：

cookie 数据⼤⼩不能超过4k

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

###### 有期时间：

localStorage 存储持久数据，浏览器关闭后数据不丢失除⾮主动删除数据

sessionStorage 数据在当前浏览器窗⼝关闭后⾃动删除

cookie 设置的 cookie 过期时间之前⼀直有效，即使窗⼝或浏览器关闭

1. iframe有那些缺点？

iframe 会阻塞主⻚⾯的 Onload 事件

###### 搜索引擎的检索程序⽆法解读这种⻚⾯，不利于 SEO

iframe 和主⻚⾯共享连接池，⽽浏览器对相同域的连接有限制，所以会影响⻚⾯的并⾏加载

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

javascript 动态给 iframe 添加 src 属性值，这样可以绕开以上两个问题

1. WEB标准以及W3C标准是什么?

标签闭合、标签⼩写、不乱嵌套、使⽤外链 css 和 js 、结构⾏为表现的分离

1. xhtml和html有什么区别?

⼀个是功能上的差别

主要是 XHTML 可兼容各⼤浏览器、⼿机以及 PDA ，并且浏览器也能快速正确地编译⽹

###### ⻚

另外是书写习惯的差别

XHTML 元素必须被正确地嵌套，闭合，区分⼤⼩写，⽂档必须拥有根元素

1. Doctype作⽤? 严格模式与混杂模式如何区分？它们有何意义?

⻚⾯被加载的时， link 会同时被加载，⽽ @imort ⻚⾯被加载的时， link 会同时被加载，⽽ @import 引⽤的 CSS 会等到⻚⾯被加载完再加载 import 只在 IE5 以上才能识别，⽽ link 是 XHTML 标签，⽆兼容问题 link ⽅式的样式的权重 ⾼于 @import 的权重

<!DOCTYPE> 声明位于⽂档中的最前⾯，处于 <html> 标签之前。告知浏览器的解析器， ⽤什么⽂档类型 规范来解析这个⽂档

###### 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

在混杂模式中，⻚⾯以宽松的向后兼容的⽅式显示。模拟⽼式浏览器的⾏为以防⽌站点⽆ 法⼯作。 DOCTYPE 不存在或格式不正确会导致⽂档以混杂模式呈现

1. ⾏内元素有哪些？块级元素有哪些？ 空(void)元素有那些？⾏内元素和块级元素有什么区别？

⾏ 内 元 素 有 ： a b span img input select strong

块级元素有： div ul ol li dl dt dd h1 h2 h3 h4… p

空 元 素 ： <br> <hr> <img> <input> <link> <meta>

###### ⾏内元素不可以设置宽⾼，不独占⼀⾏块级元素可以设置宽⾼，独占⼀⾏

1. HTML全局属性(global attribute)有哪些

class :为元素设置类标识

data-\* : 为元素增加⾃定义属性draggable : 设置元素是否可拖拽id : 元素 id ，⽂档内唯⼀

lang : 元素内容的的语⾔style : ⾏ 内 css 样 式 title : 元素相关的建议信息

1. Canvas和SVG有什么区别？

svg 绘制出来的每⼀个图形的元素都是独⽴的 DOM 节点，能够⽅便的绑定事件或⽤来修改。 canvas 输出的是⼀整幅画布

###### svg 输出的图形是⽮量图形，后期可以修改参数来⾃由放⼤缩⼩，不会失真和锯⻮。⽽

canvas 输出标量画布，就像⼀张图⽚⼀样，放⼤会失真或者锯⻮

1. HTML5 为什么只需要写 <!DOCTYPE HTML>

HTML5 不基于 SGML ，因此不需要对 DTD 进⾏引⽤，但是需要 doctype 来规范浏览器的⾏为

⽽ HTML4.01 基于 SGML ,所以需要对 DTD 进⾏引⽤，才能告知浏览器⽂档所使⽤的⽂档类型

1. 如何在⻚⾯上实现⼀个圆形的可点击区域？

svg

border-radius

纯 js 实现 需要求⼀个点在不在圆上简单算法、获取⿏标坐标等等

#### ⽹⻚验证码是⼲嘛的，是为了解决什么安全问题

###### 区分⽤户是计算机还是⼈的公共全⾃动程序。可以防⽌恶意破解密码、刷票、论坛灌⽔ 有效防⽌⿊客对某⼀个特定注册⽤户⽤特定程序暴⼒破解⽅式进⾏不断的登陆尝试

1. viewport

js

<meta name="viewport" content="width=device-width,initial-scale=1.0,minimu

// width 设置viewport宽度，为⼀个正整数，或字符串‘device-width’

// device-width 设备宽度

// height 设置viewport⾼度，⼀般设置了宽度，会⾃动解析出⾼度，可以不⽤设置

// initial-scale 默认缩放⽐例（初始缩放⽐例），为⼀个数字，可以带⼩数

// minimum-scale 允许⽤户最⼩缩放⽐例，为⼀个数字，可以带⼩数

// maximum-scale 允许⽤户最⼤缩放⽐例，为⼀个数字，可以带⼩数

// user-scalable 是否允许⼿动缩放

###### 延伸提问

怎样处理 移动端 1px 被 渲染成 2px 问题局部处理

mate 标签中的 viewport 属性 ， initial-scale 设置为 1

rem 按照设计稿标准⾛，外加利⽤ transfrome 的 scale(0.5) 缩⼩⼀倍即可；

###### 全局处理

mate 标签中的 viewport 属性 ， initial-scale 设置为 0.5 rem 按照设计稿标准⾛即可

#### 渲染优化

禁⽌使⽤ iframe （阻塞⽗⽂档 onload 事件）

iframe 会阻塞主⻚⾯的 Onload 事件

搜索引擎的检索程序⽆法解读这种⻚⾯，不利于SEO

iframe 和主⻚⾯共享连接池，⽽浏览器对相同域的连接有限制，所以会影响⻚⾯的并

###### ⾏加载

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

javascript

动态给 iframe 添加 src 属性值，这样可以绕开以上两个问题

禁⽌使⽤ gif 图⽚实现 loading 效果（降低 CPU 消耗，提升渲染性能）

使⽤ CSS3 代码代替 JS 动画（尽可能避免重绘重排以及回流）

对于⼀些⼩图标，可以使⽤base64位编码，以减少⽹络请求。但不建议⼤图使⽤，⽐较耗费 CPU

###### ⼩图标优势在于

减 少 HTTP 请 求避免⽂件跨域 修改及时⽣效

⻚⾯头部的 <style></style> <script></script> 会阻塞⻚⾯；（因为 Renderer

进程中 JS 线程和渲染线程是互斥的）

⻚⾯中空的 href 和 src 会阻塞⻚⾯其他资源的加载 (阻塞下载进程)

⽹⻚ gzip ， CDN 托管， data 缓存 ，图⽚服务器

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

⽤ innerHTML 代替 DOM 操作，减少 DOM 操作次数，优化 javascript 性能

当需要设置的样式很多时设置 className ⽽不是直接操作 style

少⽤全局变量、缓存 DOM 节点查找的结果。减少 IO 读取操作

###### 图⽚预加载，将样式表放在顶部，将脚本放在底部 加上时间戳

对普通的⽹站有⼀个统⼀的思路，就是尽量向前端优化、减少数据库操作、减少磁盘 IO

1. meta viewport相关

html

<!DOCTYPE html> <!--H5标准声明，使⽤ HTML5 doctype，不区分⼤⼩写-->

<head lang=”en”> <!--标准的 lang 属性写法-->

<meta charset=’utf-8′> <!--声明⽂档使⽤的字符编码-->

<meta http-equiv=”X-UA-Compatible” content=”IE=edge,chrome=1″/> <!--优先使

<meta name=”description” content=”不超过150个字符”/> <!--⻚⾯描述-->

<meta name=”keywords” content=””/> <!-- ⻚⾯关键词-->

<meta name=”author” content=”name, [email@gmail.com”](mailto:email@gmail.com)/> <!--⽹⻚作者-->

<meta name=”robots” content=”index,follow”/> <!--搜索引擎抓取-->

<meta name=”viewport” content=”initial-scale=1, maximum-scale=3, minimum-sc

<meta name=”apple-mobile-web-app-title” content=” 标 题 ”> <!--iOS 设 备 begin--

<meta name=”apple-mobile-web-app-capable” content=”yes”/> <!--添加到主屏后的是否启⽤ WebApp 全屏模式，删除苹果默认的⼯具栏和菜单栏-->

<meta name=”apple-itunes-app” content=”app-id=myAppStoreID, affiliate-data=

<!--添加智能 App ⼴告条 Smart App Banner（iOS 6+ Safari）-->

<meta name=”apple-mobile-web-app-status-bar-style” content=”black”/>

<meta name=”format-detection” content=”telphone=no, email=no”/> <!--设置苹果

<meta name=”renderer” content=”webkit”> <!-- 启⽤360浏览器的极速模式(webkit)--

<meta http-equiv=”X-UA-Compatible” content=”IE=edge”> <!--避免IE使⽤兼容模

<meta http-equiv=”Cache-Control” content=”no-siteapp” /> <!--不让百度转码-

<meta name=”HandheldFriendly” content=”true”> <!--针对⼿持设备优化，主要是针

<meta name=”MobileOptimized” content=”320″> <!--微软的⽼式浏览器-->

<meta name=”screen-orientation” content=”portrait”> <!--uc强制竖屏-->

<meta name=”x5-orientation” content=”portrait”> <!--QQ强制竖屏-->

<meta name=”full-screen” content=”yes”> <!--UC强制全屏-->

<meta name=”x5-fullscreen” content=”true”> <!--QQ强制全屏-->

<meta name=”browsermode” content=”application”> <!--UC应⽤模式-->

<meta name=”x5-page-mode” content=”app”> <!-- QQ应⽤模式-->

<meta name=”msapplication-tap-highlight” content=”no”> <!--windows phone

设置⻚⾯不缓存-->

<meta http-equiv=”pragma” content=”no-cache”>

<meta http-equiv=”cache-control” content=”no-cache”>

<meta http-equiv=”expires” content=”0″>

#### 你做的⻚⾯在哪些流览器测试过？这些浏览器的内核分别是什么?

IE : trident 内 核

Firefox ： gecko 内 核

Safari : webkit 内 核

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

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

1. div+css的布局较table布局有什么优点？

###### 改版的时候更⽅便 只要改 css ⽂件。

⻚⾯加载速度更快、结构化清晰、⻚⾯显示简洁。表现与结构相分离。

易于优化（ seo ）搜索引擎更友好，排名更容易靠前。

1. a：img的alt与title有何异同？b：strong与em的异同？

alt(alt text) :为不能显示图像、窗体或 applets 的⽤户代理（ UA ）， alt 属性⽤来指定替换⽂字。替换⽂字的语⾔由 lang 属性指定。(在IE浏览器下会在没有 title 时把 alt 当 成 tool tip 显 示 )

title(tool tip) :该属性为设置该属性的元素提供建议性的信息

strong :粗体强调标签，强调，表示内容的重要性

em :斜体强调标签，更强烈强调，表示内容的强调点

1. 你能描述⼀下渐进增强和优雅降级之间的不同吗

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

优雅降级：⼀开始就构建完整的功能，然后再针对低版本浏览器进⾏兼容。

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

1. 为什么利⽤多个域名来存储⽹站资源会更有效？

CDN 缓存更⽅便

###### 突破浏览器并发限制

节约 cookie 带宽

###### 节约主域名的连接数，优化⻚⾯响应速度防⽌不必要的安全问题

1. 简述⼀下src与href的区别

src ⽤于替换当前元素，href⽤于在当前⽂档和引⽤资源之间确⽴联系。

src 是 source 的缩写，指向外部资源的位置，指向的内容将会嵌⼊到⽂档中当前标签所在位置；在请求 src 资源时会将其指向的资源下载并应⽤到⽂档内，例如 js 脚本，

img 图⽚和 frame 等元素

<script src ="js.js"></script> 当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执⾏完毕，图⽚和框架等元素 也如此，类似于将所指向资源嵌⼊当前标签内。这也是为什么将js脚本放在底 部⽽不是头部

href 是 Hypertext Reference 的缩写，指向⽹络资源所在位置，建⽴和当前元素（锚点）或当前⽂档（链接）之间的链接，如果我们在⽂档中添加

<link href="common.css" rel="stylesheet"/> 那么浏览器会识别该⽂档为 css ⽂件，就会并⾏下载资源并且不会停⽌对当前⽂档的处理。这也是为什么建议使⽤ link ⽅式来加载 css ，⽽不是使⽤ @import ⽅式

#### 知道的⽹⻚制作会⽤到的图⽚格式有哪些？

png-8 、 png-24 、 jpeg 、 gif 、 svg

但是上⾯的那些都不是⾯试官想要的最后答案。⾯试官希望听到是

Webp , Apng 。（是否有关注新技术，新鲜事物）

Webp： WebP 格式，⾕歌（google）开发的⼀种旨在加快图⽚加载速度的图⽚格式。图

⽚压缩体积⼤约只有 JPEG 的 2/3 ，并能节省⼤量的服务器带宽资源和数据空间。

Facebook Ebay 等知名⽹站已经开始测试并使⽤ WebP 格式。

在质量相同的情况下，WebP格式图像的体积要⽐JPEG格式图像⼩ 40% 。

Apng：全称是 “Animated Portable Network Graphics” , 是PNG的位图动画扩展，可以实现png格式的动态图⽚效果。04年诞⽣，但⼀直得不到各⼤浏览器⼚商的⽀持，直到

⽇前得到 iOS safari 8 的⽀持，有望代替 GIF 成为下⼀代动态图标准

1. 在css/js代码上线之后开发⼈员经常会优化性能，从⽤户刷新⽹⻚开始，⼀次js请求⼀般情况下有哪些地⽅会有缓存处理？

dns 缓存， cdn 缓存，浏览器缓存，服务器缓存

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

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

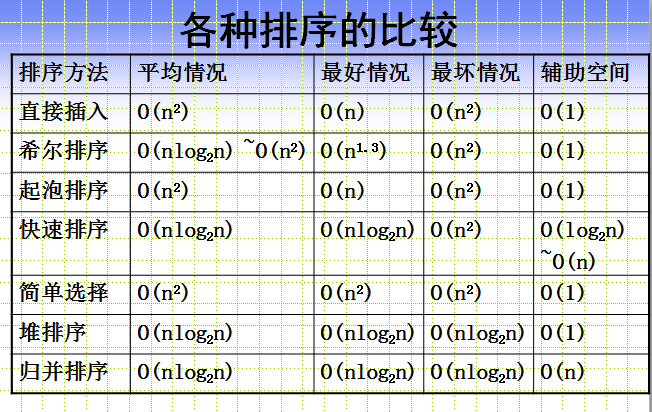
如果为幻灯⽚、相册等，可以使⽤图⽚预加载技术，将当前展示图⽚的前⼀张和后⼀张优 先下载。

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

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

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

1. 常⻅排序算法的时间复杂度,空间复杂度



1. web开发中会话跟踪的⽅法有哪些

cookie session

url 重 写

隐 藏 input ip 地址

1. HTTP request报⽂结构是怎样的
2. ⾸⾏是Request-Line包括：请求⽅法，请求URI，协议版本，CRLF
3. ⾸⾏之后是若⼲⾏请求头，包括general-header，request-header或者entity-header， 每个⼀⾏以CRLF结束
4. 请求头和消息实体之间有⼀个CRLF分隔
5. 根据实际请求需要可能包含⼀个消息实体 ⼀个请求报⽂例⼦如下：

GET /Protocols/rfc2616/rfc2616-sec5.html HTTP/1.1 Host: [www.w3.org](http://www.w3.org/)

Connection: keep-alive Cache-Control: max-age=0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/ User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, Referer: [https://www.google.com.hk/](http://www.google.com.hk/)

Accept-Encoding: gzip,deflate,sdch Accept-Language: zh-CN,zh;q=0.8,en;q=0.6 Cookie: authorstyle=yes

If-None-Match: "2cc8-3e3073913b100"

If-Modified-Since: Wed, 01 Sep 2004 13:24:52 GMT name=qiu&age=25

1. HTTP response报⽂结构是怎样的

⾸⾏是状态⾏包括：HTTP版本，状态码，状态描述，后⾯跟⼀个CRLF

⾸⾏之后是若⼲⾏响应头，包括：通⽤头部，响应头部，实体头部响应头部和响应实体之间⽤⼀个CRLF空⾏分隔

最后是⼀个可能的消息实体 响应报⽂例⼦如下：

HTTP/1.1 200 OK

Date: Tue, 08 Jul 2014 05:28:43 GMT

Server: Apache/2

Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT ETag: "40d7-3e3073913b100"

Accept-Ranges: bytes Content-Length: 16599

Cache-Control: max-age=21600

Expires: Tue, 08 Jul 2014 11:28:43 GMT

P3P: [policyref="http://www.w3.org/2001/05/P3P/p3p.xml"](http://www.w3.org/2001/05/P3P/p3p.xml) Content-Type: text/html; charset=iso-8859-1

{"name": "qiu", "age": 25}

⼆、CSS部分

1. css sprite是什么,有什么优缺点

概念：将多个⼩图⽚拼接到⼀个图⽚中。通过 background-position 和元素尺⼨调节需要显示的背景图案。

优点：

减少 HTTP 请求数，极⼤地提⾼⻚⾯加载速度

###### 增加图⽚信息重复度，提⾼压缩⽐，减少图⽚⼤⼩

更换⻛格⽅便，只需在⼀张或⼏张图⽚上修改颜⾊或样式即可实现

缺点：

###### 图⽚合并麻烦

维护麻烦，修改⼀个图⽚可能需要从新布局整个图⽚，样式

1. **display: none;** 与 **visibility: hidden;** 的 区 别

###### 联系：它们都能让元素不可⻅

区别：

display:none ;会让元素完全从渲染树中消失，渲染的时候不占据任何空间； visibility: hidden ;不会让元素从渲染树消失，渲染师元素继续占据空间，只是内容不可⻅

display: none ;是⾮继承属性，⼦孙节点消失由于元素从渲染树消失造成，通过修改

⼦孙节点属性⽆法显示 ；visibility: hidden; 是继承属性，⼦孙节点消失由于继承了 hidden ，通过设置 visibility: visible; 可以让⼦孙节点显式

修改常规流中元素的 display 通常会造成⽂档重排。修改 visibility 属性只会造成本元素的重绘。

读屏器不会读取 display: none ;元素内容；会读取 visibility: hidden; 元素内容

1. **link** 与 **@import** 的区别
2. link 是 HTML ⽅式， @import 是CSS⽅式
3. link 最⼤限度⽀持并⾏下载， @import 过多嵌套导致串⾏下载，出现 FOUC (⽂档样式短暂失效)
4. link 可以通过 rel="alternate stylesheet" 指定候选样式
5. 浏览器对 link ⽀持早于 @import ，可以使⽤ @import 对⽼浏览器隐藏样式
6. @import 必须在样式规则之前，可以在css⽂件中引⽤其他⽂件
7. 总体来说： link 优于 @import
8. 什么是FOUC?如何避免

Flash Of Unstyled Content ：⽤户定义样式表加载之前浏览器使⽤默认样式显示⽂档，⽤户样式加载渲染之后再从新显示⽂档，造成⻚⾯闪烁。

解决⽅法：把样式表放到⽂档的 <head>

1. 如何创建块级格式化上下⽂(block formatting context),BFC有什么

#### ⽤

###### 创建规则：

根元素

浮动元素（ float 不取值为 none ）

绝对定位元素（ position 取值为 absolute 或 fixed ）

display 取 值 为 inline-block 、 table-cell 、 table-caption 、 flex 、

inline-flex 之⼀的元素

overflow 不取值为 visible 的元素

作⽤：

###### 可以包含浮动元素不被浮动元素覆盖

阻⽌⽗⼦元素的 margin 折叠

## display、float、position的关系

如果 display 取值为 none ，那么 position 和 float 都不起作⽤，这种情况下元素不产⽣框

否则，如果 position 取值为 absolute 或者 fixed ，框就是绝对定位的， float 的计算值为 none ， display 根据下⾯的表格进⾏调整。

否则，如果 float 不是 none ，框是浮动的， display 根据下表进⾏调整否则，如果元素是根元素， display 根据下表进⾏调整

其他情况下 display 的值为指定值

总结起来：绝对定位、浮动、根元素都需要调整 **display**

#### 清除浮动的⼏种⽅式，各⾃的优缺点

⽗ 级 div 定 义 height

结尾处加空 div 标签 clear:both

⽗级 div 定义伪类 :after 和 zoom

⽗ 级 div 定 义 overflow:hidden

⽗级 div 也浮动，需要定义宽度结尾处加 br 标签 clear:both

###### ⽐较好的是第3种⽅式，好多⽹站都这么⽤

1. 为什么要初始化CSS样式?

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

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

1. css3有哪些新特性

新增各种 css 选择器圆 ⻆ border-radius 多列布局

###### 阴影和反射

⽂字特效 text-shadow

###### 线性渐变

旋 转 transform

CSS3新增伪类有那些？

p:first-of-type 选择属于其⽗元素的⾸个 <p> 元素的每个 <p> 元素。p:last-of-type 选择属于其⽗元素的最后 <p> 元素的每个 <p> 元素。p:only-of-type 选择属于其⽗元素唯⼀的 <p> 元素的每个 <p> 元素。p:only-child 选择属于其⽗元素的唯⼀⼦元素的每个 <p> 元素。p:nth-child(2) 选择属于其⽗元素的第⼆个⼦元素的每个 <p> 元素。

:after 在元素之前添加内容,也可以⽤来做清除浮动。

:before 在元素之后添加内容。

:enabled 已启⽤的表单元素。

:disabled 已禁⽤的表单元素。

:checked 单选框或复选框被选中。

1. display有哪些值？说明他们的作⽤

block 转换成块状元素。inline 转换成⾏内元素。none 设置元素不可⻅。

inline-block 象⾏内元素⼀样显示，但其内容象块类型元素⼀样显示。

list-item 象块类型元素⼀样显示，并添加样式列表标记。

table 此元素会作为块级表格来显示

inherit 规定应该从⽗元素继承 display 属性的值

1. 介绍⼀下标准的CSS的盒⼦模型？低版本IE的盒⼦模型有什么不同的？

有两种， IE 盒⼦模型、 W3C 盒⼦模型；

盒模型： 内容(content)、填充( padding )、边界( margin )、 边框( border )； 区 别： IE 的c ontent 部分把 border 和 padding 计算了进去;

1. CSS优先级算法如何计算？

###### 优先级就近原则，同权重情况下样式定义最近者为准载⼊样式以最后载⼊的定位为准

优先级为: !important > id > class > tag ; !important ⽐ 内联优先级⾼

1. 对BFC规范的理解？

###### 它决定了元素如何对其内容进⾏定位,以及与其他元素的关系和相互作⽤

1. 谈谈浮动和清除浮动

浮动的框可以向左或向右移动，直到他的外边缘碰到包含框或另⼀个浮动框的边框为⽌。 由于浮动框不在⽂档的普通流中，所以⽂档的普通流的块框表现得就像浮动框不存在⼀样。浮动的块框会漂浮在⽂档普通流的块框上

1. position的值， relative和absolute定位原点是

absolute ：⽣成绝对定位的元素，相对于 static 定位以外的第⼀个⽗元素进⾏定位

fixed ：⽣成绝对定位的元素，相对于浏览器窗⼝进⾏定位relative ：⽣成相对定位的元素，相对于其正常位置进⾏定位static 默认值。没有定位，元素出现在正常的流中

inherit 规定从⽗元素继承 position 属性的值

1. display:inline-block 什么时候不会显示间隙？(携程)

###### 移除空格

使⽤ margin 负值

使⽤ font-size:0 letter-spacing

word-spacing

1. PNG\GIF\JPG的区别及如何选

GIF

8 位像素， 256 ⾊

###### ⽆损压缩

⽀持简单动画

⽀持 boolean 透明适合简单动画

JPEG

颜⾊限于 256

###### 有损压缩

可控制压缩质量不⽀持透明

适合照⽚

PNG

有 PNG8 和 truecolor PNG

PNG8 类似 GIF 颜⾊上限为 256 ，⽂件⼩，⽀持 alpha 透明度，⽆动画适合图标、背景、按钮

1. ⾏内元素float:left后是否变为块级元素？

⾏内元素设置成浮动之后变得更加像是 inline-block （⾏内块级元素，设置成这个属性的元素会同时拥有⾏内和块级的特性，最明显的不同是它的默认宽 度不是 100% ），这时候给⾏内元素设置 padding-top 和 padding-bottom 或者 width 、 height 都是有效果的

#### 在⽹⻚中的应该使⽤奇数还是偶数的字体？为什么呢？

###### 偶数字号相对更容易和 web 设计的其他部分构成⽐例关系

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

单冒号( : )⽤于 CSS3 伪类，双冒号( :: )⽤于 CSS3 伪元素

###### ⽤于区分伪类和伪元素

1. 如果需要⼿动写动画，你认为最⼩时间间隔是多久，为什么？（阿

⾥）

多数显示器默认频率是 60Hz ，即 1 秒刷新 60 次，所以理论上最⼩间隔为

1/60\*1000ms ＝ 16.7ms

1. CSS合并⽅法

避免使⽤ @import 引⼊多个 css ⽂件，可以使⽤ CSS ⼯具将 CSS 合并为⼀个 CSS ⽂件，例如使⽤ Sass\Compass 等

1. CSS不同选择器的权重(CSS层叠的规则)

！important 规则最重要，⼤于其它规则

⾏内样式规则，加 1000

对于选择器中给定的各个 ID 属性值，加 100

###### 对于选择器中给定的各个类属性、属性选择器或者伪类选择器，加 10

对于选择其中给定的各个元素标签选择器，加1

如果权值⼀样，则按照样式规则的先后顺序来应⽤，顺序靠后的覆盖靠前的规则

1. 列出你所知道可以改变⻚⾯布局的属性

position 、 display 、 float 、 width 、 height 、 margin 、 padding 、

top 、 left 、 right 、 `

1. CSS在性能优化⽅⾯的实践

css 压缩与合并、 Gzip 压缩

css ⽂件放在 head ⾥、不要⽤ @import

###### 尽量⽤缩写、避免⽤滤镜、合理使⽤选择器

1. CSS3动画（简单动画的实现，如旋转等）

依 靠 CSS3 中 提 出 的 三 个 属 性 ： transition 、 transform 、 animation transition ：定义了元素在变化过程中是怎么样的，包含 transition-property 、

transition-duration 、 transition-timing-function 、 transition-delay 。transform ：定义元素的变化结果，包含 rotate 、 scale 、 skew 、 translate 。animation ：动画定义了动作的每⼀帧（ @keyframes ）有什么效果，包括 animation-

name ， animation-duration 、 animation-timing-function 、 animation-

delay 、 animation-iteration-count 、 animation-direction

1. base64的原理及优缺点

优点可以加密，减少了 HTTTP 请求缺点是需要消耗 CPU 进⾏编解码

1. ⼏种常⻅的CSS布局

流体布局

css

.left {

float: left; width: 100px; height: 200px; background: red;

}

.right {

float: right; width: 200px; height: 200px; background: blue;

}

.main {

margin-left: 120px; margin-right: 220px; height: 200px; background: green;

}

html

<div class="container">

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

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

<div class="main"></div>

</div>

###### 圣杯布局

css

.container {

margin-left: 120px; margin-right: 220px;

}

.main {

float: left; width: 100%; height:300px; background: green;

}

.left {

position: relative; left: -120px; float: left; height: 300px; width: 100px; margin-left: -100%; background: red;

}

.right {

position: relative; right: -220px; float: right; height: 300px; width: 200px;

margin-left: -200px; background: blue;

}

html

<div class="container">

<div class="main"></div>

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

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

</div>

###### 双⻜翼布局

css

.content {

float: left; width: 100%;

}

.main {

height: 200px; margin-left: 110px; margin-right: 220px; background: green;

}

.main::after {

content: ''; display: block; font-size:0; height: 0;

zoom: 1; clear: both;

}

.left {

float:left; height: 200px; width: 100px;

margin-left: -100%; background: red;

}

.right {

float: right; height: 200px; width: 200px;

margin-left: -200px; background: blue;

}

html

<div class="content">

<div class="main"></div>

</div>

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

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

## stylus/sass/less区别

均具有“变量”、“混合”、“嵌套”、“继承”、“颜⾊混合”五⼤基本特性

Scss 和 LESS 语法较为严谨， LESS 要求⼀定要使⽤⼤括号“{}”， Scss 和 Stylus 可以通过缩进表示层次与嵌套关系

Scss ⽆全局变量的概念， LESS 和 Stylus 有类似于其它语⾔的作⽤域概念

Sass 是基于 Ruby 语⾔的，⽽ LESS 和 Stylus 可以基于 NodeJS NPM 下载相应库后进⾏编译；

1. postcss的作⽤

###### 可以直观的理解为：它就是⼀个平台。为什么说它是⼀个平台呢？因为我们直接⽤它，感 觉不能⼲什么事情，但是如果让⼀些插件在它上⾯跑，那么将会很强⼤

PostCSS 提供了⼀个解析器，它能够将 CSS 解析成抽象语法树

通过在 PostCSS 这个平台上，我们能够开发⼀些插件，来处理我们的 CSS ，⽐如热⻔的 ： autoprefixer

postcss 可以对sass处理过后的 css 再处理 最常⻅的就是 autoprefixer

1. css样式（选择器）的优先级

###### 计算权重确定

!important

###### 内联样式

后写的优先级⾼

1. ⾃定义字体的使⽤场景

宣传/品牌/ banner 等固定⽂案字体图标

1. 如何美化CheckBox

<label> 属 性 for 和 id

隐藏原⽣的 <input>

:checked + <label>

1. 伪类和伪元素的区别

###### 伪类表状态

伪元素是真的有元素

前者单冒号，后者双冒号

1. **base64** 的使⽤

⽤于减少 HTTP 请求适⽤于⼩图⽚

base64 的体积约为原图的 4/3

1. ⾃适应布局

思路：

左侧浮动或者绝对定位，然后右侧 margin 撑开使⽤ <div> 包含，然后靠负 margin 形成 bfc 使 ⽤ flex

1. 请⽤CSS写⼀个简单的幻灯⽚效果⻚⾯

知道是要⽤ CSS3 。使⽤ animation 动画实现⼀个简单的幻灯⽚效果

css

/\*\*css\*\*/

.ani{

width:480px; height:320px; margin:50px auto; overflow: hidden;

box-shadow:0 0 5px rgba(0,0,0,1); background-size: cover; background-position: center;

-webkit-animation-name: "loops";

-webkit-animation-duration: 20s;

-webkit-animation-iteration-count: infinite;

}

@-webkit-keyframes "loops" { 0% {

background:[url(http://d.hiphotos.baidu.com/image/w%3D400/sign=c01e6](http://d.hiphotos.baidu.com/image/w%3D400/sign%3Dc01e6)

}

25% {

background:[url(http://b.hiphotos.baidu.com/image/w%3D400/sign=edee1](http://b.hiphotos.baidu.com/image/w%3D400/sign%3Dedee1)

} 50% {

background:[url(http://b.hiphotos.baidu.com/image/w%3D400/sign=937da](http://b.hiphotos.baidu.com/image/w%3D400/sign%3D937da)

} 75% {

background:[url(http://g.hiphotos.baidu.com/image/w%3D400/sign=7d375](http://g.hiphotos.baidu.com/image/w%3D400/sign%3D7d375)

} 100% {

background:[url(http://c.hiphotos.baidu.com/image/w%3D400/sign=cfb23](http://c.hiphotos.baidu.com/image/w%3D400/sign%3Dcfb23)

}

}

1. 什么是外边距重叠？重叠的结果是什么？

外边距重叠就是margin-collapse

###### 在CSS当中，相邻的两个盒⼦（可能是兄弟关系也可能是祖先关系）的外边距可以结合成

⼀个单独的外边距。这种合并外边距的⽅式被称为折叠，并且因⽽所结合成的外边距称为 折叠外边距。

折叠结果遵循下列计算规则：

###### 两个相邻的外边距都是正数时，折叠结果是它们两者之间较⼤的值。 两个相邻的外边距都是负数时，折叠结果是两者绝对值的较⼤值。两个外边距⼀正⼀负时，折叠结果是两者的相加的和。

1. rgba()和opacity的透明效果有什么不同？

rgba() 和 opacity 都能实现透明效果，但最⼤的不同是 opacity 作⽤于元素，以及元素内的所有内容的透明度，

⽽ rgba() 只作⽤于元素的颜⾊或其背景⾊。（设置 rgba 透明的元素的⼦元素不会继承透明效果！）

1. css中可以让⽂字在垂直和⽔平⽅向上重叠的两个属性是什么？

垂直⽅向： line-height

⽔平⽅向： letter-spacing

1. 如何垂直居中⼀个浮动元素？

css

/\*\*⽅法⼀：已知元素的⾼宽\*\*/

#div1{

background-color:#6699FF; width:200px; height:200px;

position: absolute; //⽗元素需要相对定位

top: 50%;

left: 50%;

margin-top:-100px ; //⼆分之⼀的height，width margin-left: -100px;

}

/\*\*⽅法⼆:\*\*/ #div1{

width: 200px; height: 200px;

background-color: #6699FF; margin:auto;

position: absolute; //⽗元素需要相对定位

left: 0;

top: 0;

right: 0;

bottom: 0;

}

如何垂直居中⼀个 **<img>** ?（⽤更简便的⽅法。）

css

#container /\*\*<img>的容器设置如下\*\*/

{

display:table-cell; text-align:center; vertical-align:middle;

}

1. px和em的区别

px 和 em 都是⻓度单位，区别是， px 的值是固定的，指定是多少就是多少，计算⽐较容易。 em 得值不是固定的，并且 em 会继承⽗级元素的字体⼤⼩。

浏览器的默认字体⾼都是 16px 。所以未经调整的浏览器都符合: 1em=16px 。那么

12px=0.75em , 10px=0.625em 。

1. Sass、LESS是什么？⼤家为什么要使⽤他们？

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

CSS 。

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

###### 为什么要使⽤它们？

结构清晰，便于扩展。

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

可以轻松实现多重继承。

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

1. 知道css有个content属性吗？有什么作⽤？有什么应⽤？

css的 content 属性专⻔应⽤在 before/after 伪元素上，⽤于来插⼊⽣成内容。最常⻅的应⽤是利⽤伪类清除浮动。

css

/\*\*⼀种常⻅利⽤伪类清除浮动的代码\*\*/

.clearfix:after {

content:"."; //这⾥利⽤到了content属性display:block;

height:0; visibility:hidden; clear:both;

}

.clearfix {

\*zoom:1;

}

#### ⽔平居中的⽅法

元素为⾏内元素，设置⽗元素 text-align:center

如果元素宽度固定，可以设置左右 margin 为 auto ;

如果元素为绝对定位，设置⽗元素 position 为 relative ，元素设

left:0;right:0;margin:auto;

使⽤ flex-box 布局，指定 justify-content 属性为center

display 设置为 tabel-ceil

#### 垂直居中的⽅法

将显示⽅式设置为表格， display:table-cell ,同时设置 vertial-align：middle

使⽤ flex 布局，设置为 align-item：center

绝对定位中设置 bottom:0,top:0 ,并设置 margin:auto

绝对定位中固定⾼度时设置 top:50%，margin-top 值为⾼度⼀半的负值

⽂本垂直居中设置 line-height 为 height 值

1. 如何使⽤CSS实现硬件加速？

硬件加速是指通过创建独⽴的复合图层，让GPU来渲染这个图层，从⽽提⾼性能，

⼀般触发硬件加速的 CSS 属性有 transform 、 opacity 、 filter ，为了避免2D动画在 开始和结束的时候的 repaint 操作，⼀般使⽤ tranform:translateZ(0)

1. 重绘和回流（重排）是什么，如何避免？

###### DOM的变化影响到了元素的⼏何属性（宽⾼）,浏览器重新计算元素的⼏何属性，其他元素的⼏何

属性和位置也会受到影响，浏览器需要重新构造渲染树，这个过程称为重排，浏览器将受 到影响的部分

重新绘制到屏幕上的过程称为重绘。引起重排的原因有添加或者删除可⻅的DOM元素，

元素位置、尺⼨、内容改变， 浏览器⻚⾯初始化，

浏览器窗⼝尺⼨改变，重排⼀定重绘，重绘不⼀定重排， 减少重绘和重排的⽅法：

不在布局信息改变时做 DOM 查询

使⽤ cssText 或者 className ⼀次性改变属性使 ⽤ fragment

###### 对于多次重排的元素，如动画，使⽤绝对定位脱离⽂档流，让他的改变不影响到其他元素

1. 说⼀说css3的animation

css3的 animation 是css3新增的动画属性，这个css3动画的每⼀帧是通过 @keyframes

来声明的， keyframes 声明了动画的名称，通过 from 、 to 或者是百分⽐来定义

每⼀帧动画元素的状态，通过 animation-name 来引⽤这个动画，同时css3动画也可以定义动画运⾏的时⻓、动画开始时间、动画播放⽅向、动画循环次数、动画播放的⽅式，

这些相关的动画⼦属性有： animation-name 定义动画名、 animation-duration 定义动画播放的时⻓、 animation-delay 定义动画延迟播放的时间、 animation- direction 定义 动画的播放⽅向、 animation-iteration-count 定义播放次数、animation-fill-mode 定义动画播放之后的状态、 animation-play-state 定义播放状态，如暂停运⾏等、 animation-timing-function

###### 定义播放的⽅式，如恒速播放、艰涩播放等。

1. 左边宽度固定，右边⾃适应

左侧固定宽度，右侧⾃适应宽度的两列布局实现

html结构

html

<div class="outer">

<div class="left">固定宽度</div>

<div class="right">⾃适应宽度</div>

</div>

在外层 div （类名为 outer ）的 div 中，有两个⼦ div ，类名分别为

left 和 right ，其中 left 为固定宽度，⽽ right 为⾃适应宽度

⽅法1：左侧div设置成浮动：float: left，右侧div宽度会⾃拉升适应

css

.outer {

width: 100%; height: 500px;

background-color: yellow;

}

.left {

width: 200px; height: 200px;

background-color: red; float: left;

}

.right {

height: 200px; background-color: blue;

}

⽅法2：对右侧:div进⾏绝对定位，然后再设置right=0，即可以实现宽度⾃适应

绝对定位元素的第⼀个⾼级特性就是其具有⾃动伸缩的功能，当我们将width 设置为 auto 的时候（或者不设置，默认为 auto ），绝对定位元素会根据其 left 和 right ⾃动伸缩其⼤⼩

css

.outer {

width: 100%; height: 500px;

background-color: yellow; position: relative;

}

.left {

width: 200px; height: 200px;

background-color: red;

}

.right {

height: 200px; background-color: blue; position: absolute; left: 200px;

top:0; right: 0;

}

⽅法3：将左侧div进⾏绝对定位，然后右侧div设置margin-left: 200px

css

.outer {

width: 100%; height: 500px;

background-color: yellow; position: relative;

}

.left {

width: 200px; height: 200px;

background-color: red; position: absolute;

}

.right {

height: 200px; background-color: blue; margin-left: 200px;

}

⽅法4：使⽤flex布局

css

.outer {

width: 100%; height: 500px;

background-color: yellow; display: flex;

flex-direction: row;

}

.left {

width: 200px; height: 200px;

background-color: red;

}

.right {

height: 200px; background-color: blue; flex: 1;

}

#### 两种以上⽅式实现已知或者未知宽度的垂直⽔平居中

css

/\*\* 1 \*\*/

.wraper {

position: relative;

.box {

position: absolute; top: 50%;

left: 50%; width: 100px; height: 100px;

margin: -50px 0 0 -50px;

}

}

/\*\* 2 \*\*/

.wraper {

position: relative;

.box {

position: absolute; top: 50%;

left: 50%;

transform: translate(-50%, -50%);

}

}

/\*\* 3 \*\*/

.wraper {

.box {

display: flex;

justify-content:center; align-items: center; height: 100px;

}

}

/\*\* 4 \*\*/

.wraper { display: table;

.box {

display: table-cell; vertical-align: middle;

}

}

1. 如何实现⼩于12px的字体效果

transform:scale() 这个属性只可以缩放可以定义宽⾼的元素，⽽⾏内元素是没有宽⾼的，我们可以加上⼀个 display:inline-block ;

transform: scale(0.7);

css 的属性，可以缩放⼤⼩

# 三、JavaScript

#### 闭包

###### 闭包就是能够读取其他函数内部变量的函数

闭包是指有权访问另⼀个函数作⽤域中变量的函数，创建闭包的最常⻅的⽅式就是在⼀个 函数内创建另⼀个函数，通过另⼀个函数访问这个函数的局部变量,利⽤闭包可以突破作⽤ 链域

闭包的特性：

函数内再嵌套函数

内部函数可以引⽤外层的参数和变量参数和变量不会被垃圾回收机制回收

说说你对闭包的理解

使⽤闭包主要是为了设计私有的⽅法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增⼤内存使⽤量，使⽤不当很容易造成内存泄露。在js中，函数即 闭包，只有函数才会产⽣作⽤域的概念

闭包 的最⼤⽤处有两个，⼀个是可以读取函数内部的变量，另⼀个就是让这些变量始终保持在内存中

闭包的另⼀个⽤处，是封装对象的私有属性和私有⽅法

好处：能够实现封装和缓存等；

坏处：就是消耗内存、不正当使⽤会造成内存溢出的问题使⽤闭包的注意点

由于闭包会使得函数中的变量都被保存在内存中，内存消耗很⼤，所以不能滥⽤闭包，否 则会造成⽹⻚的性能问题，在IE中可能导致内存泄露

解决⽅法是，在退出函数之前，将不使⽤的局部变量全部删除

1. 说说你对作⽤域链的理解

作⽤域链的作⽤是保证执⾏环境⾥有权访问的变量和函数是有序的，作⽤域链的变量只能 向上访问，变量访问到 window 对象即被终⽌，作⽤域链向下访问变量是不被允许的

简单的说，作⽤域就是变量与函数的可访问范围，即作⽤域控制着变量与函数的可⻅性和

⽣命周期

1. JavaScript原型，原型链 ? 有什么特点？

每个对象都会在其内部初始化⼀个属性，就是 prototype (原型)，当我们访问⼀个对象的属性时

如果这个对象内部不存在这个属性，那么他就会去 prototype ⾥找这个属性，这个prototype ⼜会有⾃⼰的 prototype ，于是就这样⼀直找下去，也就是我们平时所说的原型链的概念

关系： instance.constructor.prototype = instance. proto

特点：

JavaScript 对象是通过引⽤来传递的，我们创建的每个新对象实体中并没有⼀份属于

###### ⾃⼰的原型副本。当我们修改原型时，与之相关的对象也会继承这⼀改变

当我们需要⼀个属性的时， Javascript 引擎会先看当前对象中是否有这个属性， 如果没有的

就会查找他的 Prototype 对象是否有这个属性，如此递推下去，⼀直检索到 Object 内建对象

#### 请解释什么是事件代理

事件代理（ Event Delegation ），⼜称之为事件委托。是 JavaScript 中常⽤绑定事件的常⽤技巧。顾名思义，“事件代理”即是把原本需要绑定的事件委托给⽗元素，让⽗元 素担当事件监听的职务。事件代理的原理是DOM元素的事件冒泡。使⽤事件代理的好处是可以提⾼性能

可以⼤量节省内存占⽤，减少事件注册，⽐如在 table 上代理所有 td 的 click 事件就

###### ⾮常棒

可以实现当新增⼦对象时⽆需再次对其绑定

1. Javascript如何实现继承？

###### 构造继承

原型继承

实例继承

拷⻉继承

原型 prototype 机制或 apply 和 call ⽅法去实现较简单，建议使⽤构造函数与原型混合⽅式

js

function Parent(){ this.name = 'wang';

}

function Child(){

this.age = 28;

}

Child.prototype = new Parent();//继承了Parent，通过原型var demo = new Child();

alert(demo.age); alert(demo.name);//得到被继承的属性

1. 谈谈This对象的理解

this 总是指向函数的直接调⽤者（⽽⾮间接调⽤者） 如果有 new 关键字， this 指向 new 出来的那个对象

在事件中， this 指向触发这个事件的对象，特殊的是， IE 中的 attachEvent 中的

this 总是指向全局对象 Window

#### 事件模型

W3C 中定义事件的发⽣经历三个阶段：捕获阶段（ capturing ）、⽬标阶段

（ targetin ）、冒泡阶段（ bubbling ）

###### 冒泡型事件：当你使⽤事件冒泡时，⼦级元素先触发，⽗级元素后触发 捕获型事件：当你使⽤事件捕获时，⽗级元素先触发，⼦级元素后触发DOM 事件流：同时⽀持两种事件模型：捕获型事件和冒泡型事件

阻⽌冒泡：在 W3c 中，使⽤ stopPropagation() ⽅法；在IE下设置 cancelBubble =

true

阻⽌捕获：阻⽌事件的默认⾏为，例如 click - <a> 后的跳转。在 W3c 中，使⽤

preventDefault() ⽅法，在 IE 下设置 window.event.returnValue = false

1. new操作符具体⼲了什么呢?

创建⼀个空对象，并且 this 变量引⽤该对象，同时还继承了该函数的原型属性和⽅法被加⼊到 this 引⽤的对象中

新创建的对象由 this 所引⽤，并且最后隐式的返回 this

1. Ajax原理

Ajax 的原理简单来说是在⽤户和服务器之间加了—个中间层( AJAX 引擎)，通过XmlHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后⽤ javascrip t 来操作 DOM ⽽更新⻚⾯。使⽤户操作与服务器响应异步化。这其中最关键的⼀步就是从服务器获得请求数据

Ajax 的过程只涉及 JavaScript 、 XMLHttpRequest 和 DOM 。 XMLHttpRequest 是

aja x的核⼼机制

js

/\*\* 1. 创建连接 \*\*/

var xhr = null;

xhr = new XMLHttpRequest()

/\*\* 2. 连 接 服 务 器 \*\*/ xhr.open('get', url, true)

/\*\* 3. 发送请求 \*\*/ xhr.send(null);

/\*\* 4. 接 受 请 求 \*\*/ xhr.onreadystatechange = function(){

if(xhr.readyState == 4){

if(xhr.status == 200){ success(xhr.responseText);

} else {

/\*\* false \*\*/

fail && fail(xhr.status);

}

}

}

ajax 有那些优缺点?

优点：

###### 通过异步模式，提升了⽤户体验.

优化了浏览器和服务器之间的传输，减少不必要的数据往返，减少了带宽占⽤.

Ajax 在客户端运⾏，承担了⼀部分本来由服务器承担的⼯作，减少了⼤⽤户量下的服务器负载。

Ajax 可以实现动态不刷新（局部刷新） 缺点：

###### 安全问题 AJAX 暴露了与服务器交互的细节。对搜索引擎的⽀持⽐较弱。

不容易调试。

1. 如何解决跨域问题?

⾸先了解下浏览器的同源策略 同源策略 /SOP（Same origin policy） 是⼀种约定，由Netscape公司1995年引⼊浏览器，它是浏览器最核⼼也最基本的安全功能，如果缺少了同源策略，浏览器很容易受到 XSS 、 CSFR 等攻击。所谓同源是指"协议+域名+端⼝"三者相同，即便两个不同的域名指向同⼀个ip 地址，也⾮同源

###### 那么怎样解决跨域问题的呢？

通过jsonp跨域

js

var script = document.createElement('script');

script.type = 'text/javascript';

// 传参并指定回调执⾏函数为onBack

script.src = ['http://www.](http://www/) 8080/login?user=admin&callback=onBack';

document.head.appendChild(script);

// 回调执⾏函数

function onBack(res) { alert(JSON.stringify(res));

}

document.domain + iframe跨域

此⽅案仅限主域相同，⼦域不同的跨域应⽤场景

1.）⽗窗⼝：(<http://www.domain.com/a.html)>

html

<iframe id="iframe" src="<http://child.domain.com/b.html>"></iframe>

<script>

document.domain = 'domain.com'; var user = 'admin';

</script>

2.）⼦窗⼝：(<http://child.domain.com/b.html)>

js

document.domain = 'domain.com';

// 获取⽗窗⼝中变量

alert('get js data from parent ---> ' + window.parent.user);

nginx代理跨域

nodejs中间件代理跨域

###### 后端在头部信息⾥⾯设置安全域名

1. 模块化开发怎么做？

###### ⽴即执⾏函数,不暴露私有成员

js

var module1 = (function(){

var \_count = 0;

var m1 = function(){

//...

};

var m2 = function(){

//...

};

return {

m1 : m1, m2 : m2

};

})();

1. 异步加载JS的⽅式有哪些？

defer，只⽀持 IE async ：

创建 script ，插⼊到 DOM 中，加载完毕后 callBack

#### 那些操作会造成内存泄漏？

###### 内存泄漏指任何对象在您不再拥有或需要它之后仍然存在

setTimeout 的第⼀个参数使⽤字符串⽽⾮函数的话，会引发内存泄漏闭包使⽤不当

1. XML和JSON的区别？

###### 数据体积⽅⾯

JSON 相对 于XML 来讲，数据的体积⼩，传递的速度更快些。

###### 数据交互⽅⾯

JSON 与 JavaScript 的交互更加⽅便，更容易解析处理，更好的数据交互

###### 数据描述⽅⾯

JSON 对数据的描述性⽐ XML 较差

###### 传输速度⽅⾯

JSON 的速度要远远快于 XML

1. 谈谈你对webpack的看法

WebPack 是⼀个模块打包⼯具，你可以使⽤ WebPack 管理你的模块依赖，并编绎输出模块们所需的静态⽂件。它能够很好地管理、打包 Web 开发中所⽤到的 HTML 、Javascript 、 CSS 以及各种静态⽂件（图⽚、字体等），让开发过程更加⾼效。对于不同类型的资源， webpack 有对应的模块加载器。 webpack 模块打包器会分析模块间的依赖关系，最后 ⽣成了优化且合并后的静态资源

1. 说说你对AMD和Commonjs的理解

CommonJS 是服务器端模块的规范， Node.js 采⽤了这个规范。 CommonJS 规范加载模块是同步的，也就是说，只有加载完成，才能执⾏后⾯的操作。 AMD 规范则是⾮同步加载模块，允许指定回调函数

AMD 推荐的⻛格通过返回⼀个对象做为模块对象， CommonJS 的⻛格通过对

module.exports 或 exports 的属性赋值来达到暴露模块对象的⽬的

1. 常⻅web安全及防护原理

sql 注⼊原理

###### 就是通过把 SQL 命令插⼊到 Web 表单递交或输⼊域名或⻚⾯请求的查询字符串，最终达到欺骗服务器执⾏恶意的SQL命令

总的来说有以下⼏点

永远不要信任⽤户的输⼊，要对⽤户的输⼊进⾏校验，可以通过正则表达式，或限制⻓ 度，对单引号和双 "-" 进⾏转换等

永远不要使⽤动态拼装SQL，可以使⽤参数化的 SQL 或者直接使⽤存储过程进⾏数据查询存取

永远不要使⽤管理员权限的数据库连接，为每个应⽤使⽤单独的权限有限的数据库连接 不要把机密信息明⽂存放，请加密或者 hash 掉密码和敏感的信息

XSS原理及防范

Xss(cross-site scripting) 攻击指的是攻击者往 Web ⻚⾯⾥插⼊恶意 html 标签或者 javascript 代码。⽐如：攻击者在论坛中放⼀个看似安全的链接，骗取⽤户点击后， 窃取 cookie 中的⽤户私密信息；或者攻击者在论坛中加⼀个恶意表单，当⽤户提交表单的时候，却把信息传送到攻击者的服务器中，⽽不是⽤户原本以为的信任站点

XSS防范⽅法

⾸先代码⾥对⽤户输⼊的地⽅和变量都需要仔细检查⻓度和对 ”<”,”>”,”;”,”’” 等字符做过滤；其次任何内容写到⻚⾯之前都必须加以encode，避免不⼩⼼把 html tag 弄出来。这⼀个层⾯做好，⾄少可以堵住超过⼀半的XSS 攻击

XSS与CSRF有什么区别吗？

XSS 是获取信息，不需要提前知道其他⽤户⻚⾯的代码和数据包。 CSRF 是代替⽤户完成指定的动作，需要知道其他⽤户⻚⾯的代码和数据包。要完成⼀次 CSRF 攻击，受害者必须依次完成两个步骤

登录受信任⽹站 A ，并在本地⽣成 Cookie

在不登出 A 的情况下，访问危险⽹站 B

CSRF的防御

###### 服务端的 CSRF ⽅式⽅法很多样，但总的思想都是⼀致的，就是在客户端⻚⾯增加伪随机数

通过验证码的⽅法

1. ⽤过哪些设计模式？

⼯⼚模式：

⼯⼚模式解决了重复实例化的问题，但还有⼀个问题,那就是识别问题，因为根本⽆法 主要好处就是可以消除对象间的耦合，通过使⽤⼯程⽅法⽽不是 new 关键字

构造函数模式

使⽤构造函数的⽅法，即解决了重复实例化的问题，⼜解决了对象识别的问题，该模式 与⼯⼚模式的不同之处在于

直接将属性和⽅法赋值给 this 对象;

#### 为什么要有同源限制？

###### 同源策略指的是：协议，域名，端⼝相同，同源策略是⼀种安全协议

举例说明：⽐如⼀个⿊客程序，他利⽤ Iframe 把真正的银⾏登录⻚⾯嵌到他的⻚⾯上， 当你使⽤真实的⽤户名，密码登录时，他的⻚⾯就可以通过 Javascript 读取到你的表单中 input 中的内容，这样⽤户名，密码就轻松到⼿了。

1. offsetWidth/offsetHeight,clientWidth/clientHeight与scrollWidth/scrollHeight的区别

offsetWidth/offsetHeight 返回值包含content + padding + border，效果与e.getBoundingClientRect()相同

clientWidth/clientHeight 返回值只包含content + padding，如果有滚动条，也不包含滚动条

scrollWidth/scrollHeight 返回值包含content + padding + 溢出内容的尺⼨

1. javascript有哪些⽅法定义对象

对象字⾯量： var obj = {};

构造函数： var obj = new Object();

Object.create(): var obj = Object.create(Object.prototype);

#### 常⻅兼容性问题？

png24 位的图⽚在iE6浏览器上出现背景，解决⽅案是做成 PNG8

浏览器默认的 margin 和 padding 不同。解决⽅案是加⼀个全局的 \*

{margin:0;padding:0;} 来统⼀,，但是全局效率很低，⼀般是如下这样解决：

css

body,ul,li,ol,dl,dt,dd,form,input,h1,h2,h3,h4,h5,h6,p{ margin:0;

padding:0;

}

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

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

1. 说说你对promise的了解

依照 Promise/A+ 的定义， Promise 有四种状态：

pending: 初始状态, ⾮ fulfilled 或 rejected.

fulfilled: 成功的操作.

rejected: 失败的操作.

settled: Promise 已 被 fulfilled 或 rejected ， 且 不 是 pending

另外， fulfilled 与 rejected ⼀起合称 settled

Promise 对象⽤来进⾏延迟( deferred ) 和异步( asynchronous ) 计算Promise 的构造函数

构造⼀个 Promise ，最基本的⽤法如下：

js

var promise = new Promise(function(resolve, reject) { if (...) { // succeed

resolve(result);

} else { // fails reject(Error(errMessage));

}

});

Promise 实例拥有 then ⽅法（具有 then ⽅法的对象，通常被称为 thenable ）。它的使⽤⽅法如下：

promise.then(onFulfilled, onRejected)

接收两个函数作为参数，⼀个在 fulfilled 的时候被调⽤，⼀个在 rejected 的时候被调⽤，接收参数就是 future ， onFulfilled 对应 resolve , onRejected 对应

reject

1. 你觉得jQuery源码有哪些写的好的地⽅

jquery 源码封装在⼀个匿名函数的⾃执⾏环境中，有助于防⽌变量的全局污染，然后通过传⼊ window 对象参数，可以使 window 对象作为局部变量使⽤，好处是当 jquery 中访问 window 对象的时候，就不⽤将作⽤域链退回到顶层作⽤域了，从⽽可以更快的访问window对象。同样，传⼊ undefined 参数，可以缩短查找 undefined 时的作⽤域链jquery 将⼀些原型属性和⽅法封装在了 jquery.prototype 中，为了缩短名称，⼜赋值给了 jquery.fn ，这是很形象的写法

有⼀些数组或对象的⽅法经常能使⽤到， jQuery 将其保存为局部变量以提⾼访问速度

jquery 实现的链式调⽤可以节约代码，所返回的都是同⼀个对象，可以提⾼代码效率

## vue、react、angular

Vue.js ⼀个⽤于创建 web 交互界⾯的库，是⼀个精简的 MVVM 。它通过双向数据绑定把 View 层和 Model 层连接了起来。实际的 DOM 封装和输出格式都被抽象为了Directives 和 Filters

AngularJS 是⼀个⽐较完善的前端 MVVM 框架，包含模板，数据双向绑定，路由，模块化，服务，依赖注⼊等所有功能，模板功能强⼤丰富，⾃带了丰富的 Angular 指令

react React 仅仅是 VIEW 层是 facebook 公司。推出的⼀个⽤于构建 UI 的⼀个库，能够实现服务器端的渲染。⽤了 virtual dom ，所以性能很好。

1. Node的应⽤场景

特点：

1、它是⼀个 Javascript 运⾏环境

2、依赖于 Chrome V8 引擎进⾏代码解释

###### 3、事件驱动

4、⾮阻塞 I/O

5、单进程，单线程

优点：

###### ⾼并发（最重要的优点）

缺点：

1、只⽀持单核 CPU ，不能充分利⽤ CPU

2、可靠性低，⼀旦代码某个环节崩溃，整个系统都崩溃

1. 谈谈你对AMD、CMD的理解

CommonJS 是服务器端模块的规范， Node.js 采⽤了这个规范。 CommonJS 规范加载模块是同步的，也就是说，只有加载完成，才能执⾏后⾯的操作。 AMD 规范则是⾮同步加载模块，允许指定回调函数

AMD 推荐的⻛格通过返回⼀个对象做为模块对象， CommonJS 的⻛格通过对

module.exports 或 exports 的属性赋值来达到暴露模块对象的⽬的es6模块 CommonJS、AMD、CMD

CommonJS 的规范中，每个 JavaScript ⽂件就是⼀个独⽴的模块上下⽂（ module context ），在这个上下⽂中默认创建的属性都是私有的。也就是说，在⼀个⽂件定义的变量（还包括函数和类），都是私有的，对其他⽂件是不可⻅的。

CommonJS 是同步加载模块,在浏览器中会出现堵塞情况，所以不适⽤

AMD 异步，需要定义回调 define ⽅式

es6 ⼀个模块就是⼀个独⽴的⽂件，该⽂件内部的所有变量，外部⽆法获取。如果你希望外部能够读取模块内部的某个变量，就必须使⽤ export 关键字输出该变量 es6 还可以导出类、⽅法，⾃动适⽤严格模式

1. 那些操作会造成内存泄漏

###### 内存泄漏指任何对象在您不再拥有或需要它之后仍然存在

setTimeout 的第⼀个参数使⽤字符串⽽⾮函数的话，会引发内存泄漏

闭包、控制台⽇志、循环（在两个对象彼此引⽤且彼此保留时，就会产⽣⼀个循环）

1. web开发中会话跟踪的⽅法有哪些

cookie session

url 重 写

隐 藏 input ip 地址

#### 介绍js的基本数据类型

Undefined 、 Null 、 Boolean 、 Number 、 String

#### 介绍js有哪些内置对象

Object 是 JavaScript 中所有对象的⽗对象

数据封装类对象： Object 、 Array 、 Boolean 、 Number 和 String

其 他 对 象 ： Function 、 Arguments 、 Math 、 Date 、 RegExp 、 Error

1. 说⼏条写JavaScript的基本规范

不要在同⼀⾏声明多个变量

请使⽤ ===/!== 来⽐较 true/false 或者数值使⽤对象字⾯量替代 new Array 这种形式

###### 不要使⽤全局函数

Switch 语句必须带有 default 分⽀

If 语句必须使⽤⼤括号

for-in 循环中的变量 应该使⽤ var 关键字明确限定作⽤域，从⽽避免作⽤域污

1. JavaScript有⼏种类型的值

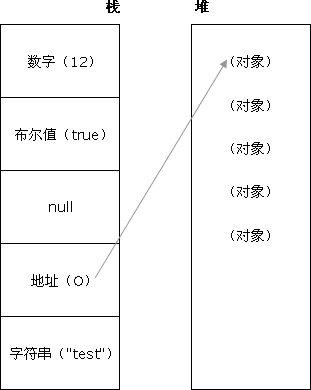
栈：原始数据类型（ Undefined ， Null ， Boolean ， Number 、 String ） 堆：引⽤数据类型（对象、数组和函数）

两种类型的区别是：存储位置不同；

原始数据类型直接存储在栈( stack )中的简单数据段，占据空间⼩、⼤⼩固定，属于被频繁使⽤数据，所以放⼊栈中存储；

###### 引⽤数据类型存储在堆( heap )中的对象,占据空间⼤、⼤⼩不固定,如果存储在栈中，将会影响程序运⾏的性能；引⽤数据类型在栈中存储了指针，该指针指向堆中该实体的起始地 址。当解释器寻找引⽤值时，会⾸先检索其

在栈中的地址，取得地址后从堆中获得实体



1. javascript创建对象的⼏种⽅式

javascript 创建对象简单的说,⽆⾮就是使⽤内置对象或各种⾃定义对象， 当然还可以⽤ JSON ；但写法有很多种，也能混合使⽤

###### 对象字⾯量的⽅式

js

person={firstname:"Mark",lastname:"Yun",age:25,eyecolor:"black"};

⽤ function 来模拟⽆参的构造函数

js

function Person(){}

var person=new Person();//定义⼀个function，如果使⽤new"实例化",该function可person.name="Mark";

person.age="25"; person.work=function(){ alert(person.name+" hello...");

}

person.work();

⽤ function 来模拟参构造函数来实现（⽤ this 关键字定义构造的上下⽂属性）

js

function Pet(name,age,hobby){

this.name=name;//this作⽤域：当前对象this.age=age;

this.hobby=hobby; this.eat=function(){

alert("我叫"+this.name+",我喜欢"+this.hobby+",是个程序员");

}

}

var maidou =new Pet("⻨兜",25,"coding");//实例化、创建对象maidou.eat();//调⽤eat⽅法

###### ⽤⼯⼚⽅式来创建（内置对象）

js

var wcDog =new Object();

wcDog.name=" 旺 财 "; wcDog.age=3; wcDog.work=function(){

alert("我是"+wcDog.name+",汪汪汪 ");

}

wcDog.work();

###### ⽤原型⽅式来创建

js

function Dog(){}

Dog.prototype.name="旺财"; Dog.prototype.eat=function(){

alert(this.name+"是个吃货");

}

var wangcai =new Dog();

wangcai.eat();

###### ⽤混合⽅式来创建

js

function Car(name,price){

this.name=name; this.price=price;

}

Car.prototype.sell=function(){

alert("我是"+this.name+"，我现在卖"+this.price+"万元");

}

var camry =new Car("凯美瑞",27); camry.sell();

1. eval是做什么的

###### 它的功能是把对应的字符串解析成 JS 代码并运⾏

应该避免使⽤ eval ，不安全，⾮常耗性能（ 2 次，⼀次解析成 js 语句，⼀次执⾏） 由 JSON 字符串转换为JSON对象的时候可以⽤ eval，var obj =eval('('+ str +')')

1. null，undefined 的区别

undefined 表示不存在这个值。

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

例如变量被声明了，但没有赋值时，就等于 undefined

null 表示⼀个对象被定义了，值为“空值”

null : 是⼀个对象(空对象, 没有任何属性和⽅法)

###### 例如作为函数的参数，表示该函数的参数不是对象；

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

## ["1", "2", "3"].map(parseInt) 答案是多少

[1, NaN, NaN] 因为 parseInt 需要两个参数 (val, radix) ，其中 radix 表示解析时⽤的基数。

map 传了 3 个 (element, index, array) ，对应的 radix 不合法导致解析失败。

1. javascript 代码中的"use strict";是什么意思

use strict 是⼀种 ECMAscript 5 添加的（严格）运⾏模式,这种模式使得 Javascript 在更严格的条件下运⾏,使 JS 编码更加规范化的模式,消除 Javascript 语法的⼀些不合理、不严谨之处，减少⼀些怪异⾏为

1. JSON 的了解

JSON(JavaScript Object Notation) 是⼀种轻量级的数据交换格式

它是基于 JavaScript 的⼀个⼦集。数据格式简单, 易于读写, 占⽤带宽⼩

JSON 字符串转换为JSON对象:

js

var obj =eval('('+ str +')');

var obj = str.parseJSON(); var obj = JSON.parse(str);

JSON 对象转换为JSON字符串：

var last=obj.toJSONString(); var last=JSON.stringify(obj);

#### js延迟加载的⽅式有哪些

defer 和 async 、动态创建 DOM ⽅式（⽤得最多）、按需异步载⼊ js

1. 同步和异步的区别

###### 同步：浏览器访问服务器请求，⽤户看得到⻚⾯刷新，重新发请求,等请求完，⻚⾯刷新， 新内容出现，⽤户看到新内容,进⾏下⼀步操作

异步：浏览器访问服务器请求，⽤户正常操作，浏览器后端进⾏请求。等请求完，⻚⾯不 刷新，新内容也会出现，⽤户看到新内容

1. 渐进增强和优雅降级

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

优雅降级 ：⼀开始就构建完整的功能，然后再针对低版本浏览器进⾏兼容

1. defer和async

defer 并⾏加载 js ⽂件，会按照⻚⾯上 script 标签的顺序执⾏

async 并⾏加载 js ⽂件，下载完成⽴即执⾏，不会按照⻚⾯上 script 标签的顺序执

###### ⾏

1. 说说严格模式的限制

变量必须声明后再使⽤

函数的参数不能有同名属性，否则报错不能使⽤ with 语句

禁⽌ this 指向全局对象

1. attribute和property的区别是什么

attribute 是 dom 元素在⽂档中作为 html 标签拥有的属性；

property 就是 dom 元素在 js 中作为对象拥有的属性。

对于 html 的标准属性来说， attribute 和 property 是同步的，是会⾃动更新的但是对于⾃定义的属性来说，他们是不同步的

1. 谈谈你对ES6的理解

新增模板字符串（为 JavaScript 提供了简单的字符串插值功能） 箭头函数

for-of （⽤来遍历数据—例如数组中的值。）

arguments 对象可被不定参数和默认参数完美代替。

ES6 将p romise 对象纳⼊规范，提供了原⽣的 Promise 对象。增加了 let 和 const 命令，⽤来声明变量。

###### 增加了块级作⽤域。

let 命令实际上就增加了块级作⽤域。

还有就是引⼊ module 模块的概念

## ECMAScript6 怎么写class么

这个语法糖可以让有 OOP 基础的⼈更快上⼿ js ，⾄少是⼀个官⽅的实现了

但对熟悉 js 的⼈来说，这个东⻄没啥⼤影响；⼀个 Object.creat() 搞定继承，⽐

class 简洁清晰的多

1. 什么是⾯向对象编程及⾯向过程编程，它们的异同和优缺点

###### ⾯向过程就是分析出解决问题所需要的步骤，然后⽤函数把这些步骤⼀步⼀步实现，使⽤ 的时候⼀个⼀个依次调⽤就可以了

⾯向对象是把构成问题事务分解成各个对象，建⽴对象的⽬的不是为了完成⼀个步骤，⽽ 是为了描叙某个事物在整个解决问题的步骤中的⾏为

⾯向对象是以功能来划分问题，⽽不是步骤

1. ⾯向对象编程思想

基本思想是使⽤对象，类，继承，封装等基本概念来进⾏程序设计 优点

易维护

采⽤⾯向对象思想设计的结构，可读性⾼，由于继承的存在，即使改变需求，那么维 护也只是在局部模块，所以维护起来是⾮常⽅便和较低成本的

易扩展

开发⼯作的重⽤性、继承性⾼，降低重复⼯作量。缩短了开发周期

1. 对web标准、可⽤性、可访问性的理解

可⽤性（Usability）：产品是否容易上⼿，⽤户能否完成任务，效率如何，以及这过程中

###### ⽤户的主观感受可好，是从⽤户的⻆度来看产品的质量。可⽤性好意味着产品质量⾼，是 企业的核⼼竞争⼒

可访问性（Accessibility）：Web内容对于残障⽤户的可阅读和可理解性

可维护性（Maintainability）：⼀般包含两个层次，⼀是当系统出现问题时，快速定位并解 决问题的成本，成本低则可维护性好。⼆是代码是否容易被⼈理解，是否容易修改和增强 功能。

1. 如何通过JS判断⼀个数组

instanceof ⽅ 法

instanceof 运算符是⽤来测试⼀个对象是否在其原型链原型构造函数的属性

js

var arr = [];

arr instanceof Array; // true

constructor ⽅ 法

constructor 属性返回对创建此对象的数组函数的引⽤，就是返回对象相对应的构造函数

js

var arr = [];

arr.constructor == Array; //true

###### 最简单的⽅法

这种写法，是 jQuery 正在使⽤的

js

Object.prototype.toString.call(value) == '[object Array]'

// 利⽤这个⽅法，可以写⼀个返回数据类型的⽅法

var isType = function (obj) {

return Object.prototype.toString.call(obj).slice(8,-1);

}

ES5 新增⽅法 isArray()

js

var a = new Array(123); var b = new Date();

console.log(Array.isArray(a)); //true console.log(Array.isArray(b)); //false

1. 谈⼀谈let与var的区别

let 命令不存在变量提升，如果在 let 前使⽤，会导致报错如果块区中存在 let 和 const 命令，就会形成封闭作⽤域

###### 不允许重复声明，因此，不能在函数内部重新声明参数

1. map与forEach的区别

forEach ⽅法，是最基本的⽅法，就是遍历与循环，默认有3个传参：分别是遍历的数组内容 item 、数组索引 index 、和当前遍历数组 Array

map ⽅法，基本⽤法与 forEach ⼀致，但是不同的，它会返回⼀个新的数组，所以在callback需要有 return 值，如果没有，会返回 undefined

1. 谈⼀谈你理解的函数式编程

简单说，"函数式编程"是⼀种"编程范式"（programming paradigm），也就是如何编写程序的⽅法论

###### 它具有以下特性：闭包和⾼阶函数、惰性计算、递归、函数是"第⼀等公⺠"、只⽤"表达式"

1. 谈⼀谈箭头函数与普通函数的区别？

函数体内的 this 对象，就是定义时所在的对象，⽽不是使⽤时所在的对象

不可以当作构造函数，也就是说，不可以使⽤ new 命令，否则会抛出⼀个错误

不可以使⽤ arguments 对象，该对象在函数体内不存在。如果要⽤，可以⽤ Rest 参数代替

不可以使⽤ yield 命令，因此箭头函数不能⽤作 Generator 函数

1. 谈⼀谈函数中this的指向

this的指向在函数定义的时候是确定不了的，只有函数执⾏的时候才能确定this到底指向 谁，实际上this的最终指向的是那个调⽤它的对象

《javascript语⾔精髓》中⼤概概括了4种调⽤⽅式：

###### ⽅法调⽤模式

函数调⽤模式

构造器调⽤模式

js

graph LR A-->B

apply/call调⽤模式

#### 异步编程的实现⽅式

###### 回调函数

优点：简单、容易理解

缺点：不利于维护，代码耦合⾼

###### 事件监听(采⽤时间驱动模式，取决于某个事件是否发⽣)：

优点：容易理解，可以绑定多个事件，每个事件可以指定多个回调函数缺点：事件驱动型，流程不够清晰

发布/订阅(观察者模式)

###### 类似于事件监听，但是可以通过‘消息中⼼ʼ，了解现在有多少发布者，多少订阅者

Promise对象

优点：可以利⽤then⽅法，进⾏链式写法；可以书写错误时的回调函数； 缺点：编写和理解，相对⽐较难

Generator函数

###### 优点：函数体内外的数据交换、错误处理机制缺点：流程管理不⽅便

async函数

###### 优点：内置执⾏器、更好的语义、更⼴的适⽤性、返回的是Promise、结构清晰。缺点：错误处理机制

1. 对原⽣Javascript了解程度

数据类型、运算、对象、Function、继承、闭包、作⽤域、原型链、事件、 RegExp 、JSON 、 Ajax 、 DOM 、 BOM 、内存泄漏、跨域、异步装载、模板引擎、前端 MVC 、路由、模块化、 Canvas 、 ECMAScript

1. Js动画与CSS动画区别及相应实现

CSS3 的动画的优点

在性能上会稍微好⼀些，浏览器会对 CSS3 的动画做⼀些优化代码相对简单

###### 缺点

在动画控制上不够灵活兼容性不好

JavaScript 的动画正好弥补了这两个缺点，控制能⼒很强，可以单帧的控制、变换，同时写得好完全可以兼容 IE6 ，并且功能强⼤。对于⼀些复杂控制的动画，使⽤javascript 会⽐较靠谱。⽽在实现⼀些⼩的交互动效的时候，就多考虑考虑 CSS 吧

1. JS 数组和对象的遍历⽅式，以及⼏种⽅式的⽐较

通常我们会⽤循环的⽅式来遍历数组。但是循环是 导致js 性能问题的原因之

⼀。⼀般我们会采⽤下⼏种⽅式来进⾏数组的遍历

for in 循 环

for 循 环

forEach

这⾥的 forEach 回调中两个参数分别为 value ， index forEach ⽆法遍历对象

IE不⽀持该⽅法； Firefox 和 chrome ⽀持

forEach ⽆法使⽤ break ， continue 跳出循环，且使⽤ return 是跳过本次循环

###### 这两种⽅法应该⾮常常⻅且使⽤很频繁。但实际上，这两种⽅法都存在性能问题

在⽅式⼀中， for-in 需要分析出 array 的每个属性，这个操作性能开销很⼤。⽤在key 已知的数组上是⾮常不划算的。所以尽量不要⽤ for-in ，除⾮你不清楚要处理哪些属性，例如 JSON 对象这样的情况

在⽅式2中，循环每进⾏⼀次，就要检查⼀下数组⻓度。读取属性（数组⻓度）要⽐读局部 变量慢，尤其是当 array ⾥存放的都是 DOM 元素，因为每次读取都会扫描⼀遍⻚⾯上的选择器相关元素，速度会⼤⼤降低

1. gulp是什么

###### gulp 是前端开发过程中⼀种基于流的代码构建⼯具，是⾃动化项⽬的构建利器；它不仅能对⽹站资源进⾏优化，⽽且在开发过程中很多重复的任务能够使⽤正确的⼯具⾃动完成

Gulp的核⼼概念：流

###### 流，简单来说就是建⽴在⾯向对象基础上的⼀种抽象的处理数据的⼯具。在流中，定义了

⼀些处理数据的基本操作，如读取数据，写⼊数据等，程序员是对流进⾏所有操作的，⽽ 不⽤关⼼流的另⼀头数据的真正流向

gulp正是通过流和代码优于配置的策略来尽量简化任务编写的⼯作Gulp的特点：

易于使⽤：通过代码优于配置的策略，gulp 让简单的任务简单，复杂的任务可管理构建快速 利⽤ Node.js 流的威⼒，你可以快速构建项⽬并减少频繁的 IO 操作

易于学习 通过最少的 API ，掌握 gulp 毫不费⼒，构建⼯作尽在掌握：如同⼀系列流管道

1. 说⼀下Vue的双向绑定数据的原理

vue.js 则是采⽤数据劫持结合发布者-订阅者模式的⽅式，通过Object.defineProperty() 来劫持各个属性的 setter ， getter ，在数据变动时发布消息给订阅者，触发相应的监听回调

1. 事件的各个阶段

1：捕获阶段 ---> 2：⽬标阶段 ---> 3：冒泡阶段

document ---> target ⽬ 标 ----> document

由此， addEventListener 的第三个参数设置为 true 和 false 的区别已经⾮常清晰了

true 表示该元素在事件的“捕获阶段”（由外往内传递时）响应事件

false 表示该元素在事件的“冒泡阶段”（由内向外传递时）响应事件

## let var const

##### let

###### 允许你声明⼀个作⽤域被限制在块级中的变量、语句或者表达式 let绑定不受变量提升的约束，这意味着let声明不会被提升到当前 该变量处于从块开始到初始化处理的“暂存死区”

var

声明变量的作⽤域限制在其声明位置的上下⽂中，⽽⾮声明变量总是全局的

由于变量声明（以及其他声明）总是在任意代码执⾏之前处理的，所以在代码中的任意位 置声明变量总是等效于在代码开头声明

const

声明创建⼀个值的只读引⽤ (即指针)

###### 基本数据当值发⽣改变时，那么其对应的指针也将发⽣改变，故造成 const 申明基本数据类型时

再将其值改变时，将会造成报错， 例如 const a = 3 ; a = 5 时 将会报错

但是如果是复合类型时，如果只改变复合类型的其中某个 Value 项时， 将还是正常使⽤

1. 快速的让⼀个数组乱序

js

var arr = [1,2,3,4,5,6,7,8,9,10];

arr.sort(function(){

return Math.random() - 0.5;

})

console.log(arr);

#### 如何渲染⼏万条数据并不卡住界⾯

这道题考察了如何在不卡住⻚⾯的情况下渲染数据，也就是说不能⼀次性将⼏ 万条都渲染出来，⽽应该⼀次渲染部分 DOM ，那么就可以通过requestAnimationFrame 来每 16 ms 刷新⼀次

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

<ul>控件</ul>

<script> setTimeout(() => {

// 插⼊⼗万条数据

const total = 100000

// ⼀次插⼊ 20 条，如果觉得性能不好就减少

const once = 20

// 渲染数据总共需要⼏次

const loopCount = total / once let countOfRender = 0

let ul = document.querySelector("ul");

function add() {

// 优化性能，插⼊不会造成回流

const fragment = document.createDocumentFragment(); for (let i = 0; i < once; i++) {

const li = document.createElement("li"); li.innerText = Math.floor(Math.random() \* total); fragment.appendChild(li);

}

ul.appendChild(fragment); countOfRender += 1; loop();

}

function loop() {

if (countOfRender < loopCount) { window.requestAnimationFrame(add);

}

}

loop();

}, 0);

</script>

</body>

</html>

1. 希望获取到⻚⾯中所有的checkbox怎么做？

不使⽤第三⽅框架

js

var domList = document.getElementsByTagName(‘input’) var checkBoxList = [];

var len = domList.length; //缓存到局部变量

while (len--) { //使⽤while的效率会⽐for循环更⾼

if (domList[len].type == ‘checkbox’) { checkBoxList.push(domList[len]);

}

}

1. 怎样添加、移除、移动、复制、创建和查找节点

###### 创建新节点

js

createDocumentFragment() //创建⼀个DOM⽚段

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

###### 添加 移除 替换 插⼊

、 、 、

js

appendChild() //添加

removeChild() //移除

replaceChild() //替换

insertBefore() //插⼊

###### 查找

js

getElementsByTagName() //通过标签名称

getElementsByName() //通过元素的Name属性的值getElementById() //通过元素Id，唯⼀性

#### 正则表达式

正则表达式构造函数 var reg=new RegExp(“xxx”) 与正则表达字⾯量 var reg=// 有什么不同？匹配邮箱的正则表达式？

当使⽤ RegExp() 构造函数的时候，不仅需要转义引号（即 \ ”表示”），并且还需要双反斜杠（即 \\ 表示⼀个 \ ）。使⽤正则表达字⾯量的效率更⾼

###### 邮箱的正则匹配：

js

var regMail = /^([a-zA-Z0-9\_-])+@([a-zA-Z0-9\_-])+((.[a-zA-Z0-9\_-]{2,3}){1,2

1. Javascript中callee和caller的作⽤？

caller 是返回⼀个对函数的引⽤，该函数调⽤了当前函数；

callee 是返回正在被执⾏的 function 函数，也就是所指定的 function 对象的正⽂

那么问题来了？如果⼀对兔⼦每⽉⽣⼀对兔⼦；⼀对新⽣兔，从第⼆个⽉起就 开始⽣兔⼦；假定每对兔⼦都是⼀雌⼀雄，试问⼀对兔⼦，第n个⽉能繁殖成 多少对兔⼦？（使⽤ callee 完成）

js

var result=[];

function fn(n){ //典型的斐波那契数列if(n==1){

return 1;

}else if(n==2){

return 1;

}else{

if(result[n]){

return result[n];

}else{

}

}

}

//argument.callee() 表 示 fn() result[n]=arguments.callee(n-1)+arguments.callee(n-2); return result[n];

## window.onload和$(document).ready

原 ⽣ JS 的 window.onload 与 Jquery 的 $(document).ready(function()

{}) 有什么不同？如何⽤原⽣JS实现Jq的 ready ⽅法？

window.onload() ⽅法是必须等到⻚⾯内包括图⽚的所有元素加载完毕后才能执⾏。

$(document).ready() 是 DOM 结构绘制完毕后就执⾏，不必等到加载完毕

js

function ready(fn){

if(document.addEventListener) { //标准浏览器document.addEventListener('DOMContentLoaded', function() {

//注销事件, 避免反复触发

document.removeEventListener('DOMContentLoaded',arguments.cal fn(); //执⾏函数

}, false);

}else if(document.attachEvent) { //IE document.attachEvent('onreadystatechange', function() {

if(document.readyState == 'complete') { document.detachEvent('onreadystatechange', arguments.calle

fn(); //函数执⾏

}

});

}

};

## addEventListener()和attachEvent()的区别

addEventListener() 是符合W3C规范的标准⽅法; attachEvent() 是IE低版本的⾮标准⽅法

addEventListener() ⽀持事件冒泡和事件捕获; - ⽽ attachEvent() 只⽀持事件冒泡addEventListener() 的第⼀个参数中,事件类型不需要添加 on ; attachEvent() 需要添加 'on'

如果为同⼀个元素绑定多个事件, addEventListener() 会按照事件绑定的顺序依次执⾏,

attachEvent() 会按照事件绑定的顺序倒序执⾏

1. 获取⻚⾯所有的checkbox

js

var resultArr= [];

var input = document.querySelectorAll('input'); for( var i = 0; i < input.length; i++ ) {

if( input[i].type == 'checkbox' ) { resultArr.push( input[i] );

}

}

//resultArr即中获取到了⻚⾯中的所有checkbox

#### 数组去重⽅法总结

⽅法⼀、利⽤ES6 Set去重（ES6中最常⽤）

js

function unique (arr) {

return Array.from(new Set(arr))

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin console.log(unique(arr))

//[1, "true", true, 15, false, undefined, null, NaN, "NaN", 0, "a", {}, {}

⽅法⼆、利⽤for嵌套for，然后splice去重（ES5中最常⽤）

js

function unique(arr){

for(var i=0; i<arr.length; i++){ for(var j=i+1; j<arr.length; j++){

if(arr[i]==arr[j]){ //第⼀个等同于第⼆个，splice⽅法删除

arr.splice(j,1); j--;

}

}

}

return arr;

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin console.log(unique(arr))

//[1, "true", 15, false, undefined, NaN, NaN, "NaN", "a", {…}, {…}]

###### 双层循环，外层循环元素，内层循环时⽐较值。值相同时，则删去这个值。 想快速学习更多常⽤的 ES6 语法

⽅法三、利⽤indexOf去重

js

function unique(arr) {

if (!Array.isArray(arr)) { console.log('type error!') return

}

var array = [];

for (var i = 0; i < arr.length; i++) { if (array .indexOf(arr[i]) === -1) {

array .push(arr[i])

}

}

return array;

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin console.log(unique(arr))

// [1, "true", true, 15, false, undefined, null, NaN, NaN, "NaN", 0, "a"

新建⼀个空的结果数组， for 循环原数组，判断结果数组是否存在当前元素，如果有相同的值则跳过，不相同则 push 进数组

⽅法四、利⽤sort()

js

function unique(arr) {

if (!Array.isArray(arr)) { console.log('type error!') return;

}

arr = arr.sort() var arrry= [arr[0]];

for (var i = 1; i < arr.length; i++) { if (arr[i] !== arr[i-1]) {

arrry.push(arr[i]);

}

}

return arrry;

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin console.log(unique(arr))

// [0, 1, 15, "NaN", NaN, NaN, {…}, {…}, "a", false, null, true, "true", un

利⽤ sort() 排序⽅法，然后根据排序后的结果进⾏遍历及相邻元素⽐对

⽅法五、利⽤对象的属性不能相同的特点进⾏去重

js

function unique(arr) {

if (!Array.isArray(arr)) { console.log('type error!') return

}

var arrry= []; var obj = {};

for (var i = 0; i < arr.length; i++) { if (!obj[arr[i]]) {

arrry.push(arr[i]) obj[arr[i]] = 1

} else {

obj[arr[i]]++

}

}

return arrry;

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin

console.log(unique(arr))

//[1, "true", 15, false, undefined, null, NaN, 0, "a", {…}] //两个true直接

⽅法六、利⽤includes

js

function unique(arr) {

if (!Array.isArray(arr)) { console.log('type error!') return

}

var array =[];

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

if( !array.includes( arr[i]) ) {//includes 检测数组是否有某个值

array.push(arr[i]);

}

}

return array

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin console.log(unique(arr))

//[1, "true", true, 15, false, undefined, null, NaN, "NaN", 0, "a", {…}

⽅法七、利⽤hasOwnProperty

js

function unique(arr) {

var obj = {};

return arr.filter(function(item, index, arr){

return obj.hasOwnProperty(typeof item + item) ? false : (obj[typeof

})

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,und console.log(unique(arr))

//[1, "true", true, 15, false, undefined, null, NaN, "NaN", 0, "a", {…}]

利⽤ hasOwnProperty 判断是否存在对象属性

⽅法⼋、利⽤filter

js

function unique(arr) {

return arr.filter(function(item, index, arr) {

//当前元素，在原始数组中的第⼀个索引==当前索引值，否则返回当前元素return arr.indexOf(item, 0) === index;

});

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin console.log(unique(arr))

//[1, "true", true, 15, false, undefined, null, "NaN", 0, "a", {…}, {…}]

⽅法九、利⽤递归去重

js

function unique(arr) {

var array= arr;

var len = array.length;

array.sort(function(a,b){ //排序后更加⽅便去重return a - b;

})

function loop(index){ if(index >= 1){

if(array[index] === array[index-1]){ array.splice(index,1);

}

loop(index - 1); //递归loop，然后数组去重

}

}

loop(len-1); return array;

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin console.log(unique(arr))

//[1, "a", "true", true, 15, false, 1, {…}, null, NaN, NaN, "NaN", 0, "a",

⽅法⼗、利⽤Map数据结构去重

js

function arrayNonRepeatfy(arr) {

let map = new Map();

let array = new Array(); // 数组⽤于返回结果

for (let i = 0; i < arr.length; i++) { if(map .has(arr[i])) { // 如果有该key值

map .set(arr[i], true);

} else {

map .set(arr[i], false); // 如果没有该key值

array .push(arr[i]);

}

}

return array ;

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefi console.log(unique(arr))

//[1, "a", "true", true, 15, false, 1, {…}, null, NaN, NaN, "NaN", 0, "a",

创建⼀个空 Map 数据结构，遍历需要去重的数组，把数组的每⼀个元素作为key 存到 Map 中。由于 Map 中不会出现相同的 key 值，所以最终得到的就是去重后的结果

⽅法⼗⼀、利⽤reduce+includes

js

function unique(arr){

return arr.reduce((prev,cur) => prev.includes(cur) ? prev : [...prev,cu

}

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefin console.log(unique(arr));

// [1, "true", true, 15, false, undefined, null, NaN, "NaN", 0, "a", {…}, {

⽅法⼗⼆、[...new Set(arr)]

js

[...new Set(arr)]

//代码就是这么少 （其实，严格来说并不算是⼀种，相对于第⼀种⽅法来说只是简化了代码）

1. （设计题）想实现⼀个对⻚⾯某个节点的拖曳？如何做？（使⽤ 原⽣JS）

给需要拖拽的节点绑定 mousedown , mousemove , mouseup 事件

mousedown 事件触发后，开始拖拽

mousemove 时，需要通过 event.clientX 和 clientY 获取拖拽位置，并实时更新位置

mouseup 时，拖拽结束

###### 需要注意浏览器边界的情况

1. Javascript全局函数和全局变量

全局变量

Infinity 代表正的⽆穷⼤的数值。NaN 指示某个值是不是数字值。undefined 指示未定义的值。

###### 全局函数

decodeURI() 解码某个编码的 URI 。decodeURIComponent() 解码⼀个编码的 URI 组件。encodeURI() 把字符串编码为 URI。 encodeURIComponent() 把字符串编码为 URI 组件。escape() 对字符串进⾏编码。

eval() 计算 JavaScript 字符串，并把它作为脚本代码来执⾏。

isFinite() 检查某个值是否为有穷⼤的数。isNaN() 检查某个值是否是数字。Number() 把对象的值转换为数字。

parseFloat() 解析⼀个字符串并返回⼀个浮点数。parseInt() 解析⼀个字符串并返回⼀个整数。String() 把对象的值转换为字符串。

unescape() 对由 escape() 编码的字符串进⾏解码

1. 使⽤js实现⼀个持续的动画效果

###### 定时器思路

js

var e = document.getElementById('e') var flag = true;

var left = 0; setInterval(() => {

left == 0 ? flag = true : left == 100 ? flag = false : ''

flag ? e.style.left = ` ${left++}px` : e.style.left = ` ${left--}px`

}, 1000 / 60)

requestAnimationFrame

js

//兼容性处理

window.requestAnimFrame = (function(){

return window.requestAnimationFrame || window.webkitRequestAnimationFrame || window.mozRequestAnimationFrame || function(callback){

window.setTimeout(callback, 1000 / 60);

};

})();

var e = document.getElementById("e"); var flag = true;

var left = 0;

function render() {

left == 0 ? flag = true : left == 100 ? flag = false : ''; flag ? e.style.left = ` ${left++}px` :

e.style.left = ` ${left--}px`;

}

(function animloop() { render(); requestAnimFrame(animloop);

})();

使⽤css实现⼀个持续的动画效果

css

animation:mymove 5s infinite;

@keyframes mymove { from {top:0px;} to {top:200px;}

}

animation-name 规定需要绑定到选择器的 keyframe 名称。animation-duration 规定完成动画所花费的时间，以秒或毫秒计。animation-timing-function 规定动画的速度曲线。

animation-delay 规定在动画开始之前的延迟。animation-iteration-count 规定动画应该播放的次数。animation-direction 规定是否应该轮流反向播放动画

1. 封装⼀个函数，参数是定时器的时间，.then执⾏回调函数

js

function sleep (time) {

return new Promise((resolve) => setTimeout(resolve, time));

}

1. 怎么判断两个对象相等？

js

obj={

a:1,

b:2

}

obj2={

a:1,

b:2

}

obj3={

a:1, b:'2'

}

可以转换为字符串来判断

js

JSON.stringify(obj)==JSON.stringify(obj2);//true

JSON.stringify(obj)==JSON.stringify(obj3);//false

#### 项⽬做过哪些性能优化？

减少 HTTP 请求数减 少 DNS 查 询

使 ⽤ CDN 避免重定向图⽚懒加载

减少 DOM 元素数量减 少 DOM 操 作

使 ⽤ 外 部 JavaScript 和 CSS

压缩 JavaScript 、 CSS 、字体、图⽚等优 化 CSS Sprite

使 ⽤ iconfont

###### 字体裁剪

多域名分发划分内容到不同域名尽量减少 iframe 使⽤

避免图⽚ src 为空把样式表放在 link 中

把 JavaScript 放在⻚⾯底部

#### 浏览器缓存

浏览器缓存分为强缓存和协商缓存。当客户端请求某个资源时，获取缓存的流 程如下

先根据这个资源的⼀些 http header 判断它是否命中强缓存，如果命中，则直接从本地获取缓存资源，不会发请求到服务器；

当强缓存没有命中时，客户端会发送请求到服务器，服务器通过另⼀些 request header 验证这个资源是否命中协商缓存，称为 http 再验证，如果命中，服务器将请求返回，但不返回资源，⽽是告诉客户端直接从缓存中获取，客户端收到返回后就会从缓存中获取资 源；

###### 强缓存和协商缓存共同之处在于，如果命中缓存，服务器都不会返回资源； 区别是，强缓存不对发送请求到服务器，但协商缓存会。

当协商缓存也没命中时，服务器就会将资源发送回客户端。

当 ctrl+f5 强制刷新⽹⻚时，直接从服务器加载，跳过强缓存和协商缓存； 当 f5 刷新⽹⻚时，跳过强缓存，但是会检查协商缓存；

###### 强缓存

Expires （该字段是 http1.0 时的规范，值为⼀个绝对时间的 GMT 格式的时间字符串，代表缓存资源的过期时间）

Cache-Control:max-age （该字段是 http1.1 的规范，强缓存利⽤其 max-age 值来判断缓存资源的最⼤⽣命周期，它的值单位为秒）

协商缓存

Last-Modified （值为资源最后更新时间，随服务器response返回）

If-Modified-Since （通过⽐较两个时间来判断资源在两次请求期间是否有过修改，如果没有修改，则命中协商缓存）

ETag （表示资源内容的唯⼀标识，随服务器 response 返回）

If-None-Match （服务器通过⽐较请求头部的 If-None-Match 与当前资源的 ETag 是否⼀致来判断资源是否在两次请求之间有过修改，如果没有修改，则命中协商缓存）

## WebSocket

由于 http 存在⼀个明显的弊端（消息只能有客户端推送到服务器端，⽽服务器端不能主动推送到客户端），导致如果服务器如果有连续的变化，这时只 能使⽤轮询，⽽轮询效率过低，并不适合。于是 WebSocket 被发明出来

相⽐与 http 具有以下有点

###### ⽀持双向通信，实时性更强；

可以发送⽂本，也可以⼆进制⽂件；

协议标识符是 ws ，加密后是 wss ；

###### 较少的控制开销。连接创建后， ws 客户端、服务端进⾏数据交换时，协议控制的数据包头部较⼩。在不包含头部的情况下，服务端到客户端的包头只有 2~10 字节（取决于数据包⻓度），客户端到服务端的的话，需要加上额外的4字节的掩码。⽽ HTTP 协议每次通信都需要携带完整的头部；

⽀持扩展。ws协议定义了扩展，⽤户可以扩展协议，或者实现⾃定义的⼦协议。（⽐如⽀ 持⾃定义压缩算法等）

⽆跨域问题。

实现⽐较简单，服务端库如 socket.io 、 ws ，可以很好的帮助我们⼊⻔。

⽽客户端也只需要参照 api 实现即可

1. 尽可能多的说出你对 Electron 的理解

最最重要的⼀点， electron 实际上是⼀个套了 Chrome 的 nodeJS 程序

###### 所以应该是从两个⽅⾯说开来

Chrome （⽆各种兼容性问题）；

NodeJS （ NodeJS 能做的它也能做）

1. 深浅拷⻉

浅拷⻉

Object.assign

###### 或者展开运算符深拷⻉

可以通过 JSON.parse(JSON.stringify(object)) 来解决

js

let a = {

age: 1,

jobs: {

first: 'FE'

}

}

let b = JSON.parse(JSON.stringify(a)) a.jobs.first = 'native' console.log(b.jobs.first) // FE

###### 该⽅法也是有局限性的

会忽略 undefined

###### 不能序列化函数

不能解决循环引⽤的对象

1. 防抖/节流

防抖

在滚动事件中需要做个复杂计算或者实现⼀个按钮的防⼆次点击操作。可以通 过函数防抖动来实现

js

// 使⽤ underscore 的源码来解释防抖动

/\*\*

* underscore 防抖函数，返回函数连续调⽤时，空闲时间必须⼤于或等于 wait，func 才会执⾏

\*

* @param {function} func 回调函数
* @param {number} wait 表示时间窗⼝的间隔
* @param {boolean} immediate 设置为ture时，是否⽴即调⽤函数
* @return {function} 返回客户调⽤函数

\*/

\_.debounce = function(func, wait, immediate) {

var timeout, args, context, timestamp, result;

var later = function() {

// 现在和上⼀次时间戳⽐较

var last = \_.now() - timestamp;

// 如果当前间隔时间少于设定时间且⼤于0就重新设置定时器

if (last < wait && last >= 0) {

timeout = setTimeout(later, wait - last);

} else {

// 否则的话就是时间到了执⾏回调函数

timeout = null; if (!immediate) {

result = func.apply(context, args); if (!timeout) context = args = null;

}

}

};

return function() { context = this; args = arguments;

// 获得时间戳

timestamp = \_.now();

// 如果定时器不存在且⽴即执⾏函数

var callNow = immediate && !timeout;

// 如果定时器不存在就创建⼀个

if (!timeout) timeout = setTimeout(later, wait); if (callNow) {

// 如果需要⽴即执⾏函数的话 通过 apply 执⾏

result = func.apply(context, args); context = args = null;

}

return result;

};

};

整体函数实现

对于按钮防点击来说的实现

开始⼀个定时器，只要我定时器还在，不管你怎么点击都不会执⾏回调函数。⼀旦定时器 结束并设置为 null，就可以再次点击了

对于延时执⾏函数来说的实现：每次调⽤防抖动函数都会判断本次调⽤和之前的时间间隔，如果⼩于需要的时间间隔，就会重新创建⼀个定时器，并且定时器的延时为设定时间 减去之前的时间间隔。⼀旦时间到了，就会执⾏相应的回调函数

节流

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

js

/\*\*

* underscore 节流函数，返回函数连续调⽤时，func 执⾏频率限定为 次 / wait

\*

* @param {function} func 回调函数
* @param {number} wait 表示时间窗⼝的间隔
* @param {object} options 如果想忽略开始函数的的调⽤，传⼊{leading: fals
* 如果想忽略结尾函数的调⽤，传⼊{trailing: false
* 两者不能共存，否则函数不能执⾏
* @return {function} 返回客户调⽤函数

\*/

\_.throttle = function(func, wait, options) { var context, args, result;

var timeout = null;

// 之前的时间戳

var previous = 0;

// 如果 options 没传则设为空对象

if (!options) options = {};

// 定时器回调函数

var later = function() {

// 如果设置了 leading，就将 previous 设为 0

// ⽤于下⾯函数的第⼀个 if 判断

previous = options.leading === false ? 0 : \_.now();

// 置空⼀是为了防⽌内存泄漏，⼆是为了下⾯的定时器判断

timeout = null;

result = func.apply(context, args); if (!timeout) context = args = null;

};

return function() {

// 获得当前时间戳

var now = \_.now();

// ⾸次进⼊前者肯定为 true

// 如果需要第⼀次不执⾏函数

// 就将上次时间戳设为当前的

// 这样在接下来计算 remaining 的值时会⼤于0

if (!previous && options.leading === false) previous = now;

// 计算剩余时间

var remaining = wait - (now - previous); context = this;

args = arguments;

// 如果当前调⽤已经⼤于上次调⽤时间 + wait

// 或者⽤户⼿动调了时间

// 如果设置了 trailing，只会进⼊这个条件

// 如果没有设置 leading，那么第⼀次会进⼊这个条件

// 还有⼀点，你可能会觉得开启了定时器那么应该不会进⼊这个 if 条件了

// 其实还是会进⼊的，因为定时器的延时

// 并不是准确的时间，很可能你设置了2秒

// 但是他需要2.2秒才触发，这时候就会进⼊这个条件

if (remaining <= 0 || remaining > wait) {

// 如果存在定时器就清理掉否则会调⽤⼆次回调

if (timeout) { clearTimeout(timeout); timeout = null;

}

previous = now;

result = func.apply(context, args); if (!timeout) context = args = null;

} else if (!timeout && options.trailing !== false) {

// 判断是否设置了定时器和 trailing

// 没有的话就开启⼀个定时器

// 并且不能不能同时设置 leading 和 trailing

timeout = setTimeout(later, remaining);

}

return result;

};

};

1. 谈谈变量提升？

当执⾏ JS 代码时，会⽣成执⾏环境，只要代码不是写在函数中的，就是在全局执⾏环境中，函数中的代码会产⽣函数执⾏环境，只此两种执⾏环境

接下来让我们看⼀个⽼⽣常谈的例⼦， var

js

b() // call b

console.log(a) // undefined var a = 'Hello world'

function b() { console.log('call b')

}

变量提升

###### 这是因为函数和变量提升的原因。通常提升的解释是说将声明的代码移动到了顶部， 这其实没有什么错误，便于⼤家理解。但是更准确的解释应该是：在⽣成执⾏环境

时，会有两个阶段。第⼀个阶段是创建的阶段，JS 解释器会找出需要提升的变量和函数，并且给他们提前在内存中开辟好空间，函数的话会将整个函数存⼊内存中，变量 只声明并且赋值为 undefined ，所以在第⼆个阶段，也就是代码执⾏阶段，我们可以直接提前使⽤

###### 在提升的过程中，相同的函数会覆盖上⼀个函数，并且函数优先于变量提升

js

b() // call b second

function b() { console.log('call b fist')

}

function b() {

console.log('call b second')

}

var b = 'Hello world'

复制代码 var 会产⽣很多错误，所以在 ES6 中引⼊了 let 。 let 不能在声明前使⽤，但是这并不是常说的 let 不会提升， let 提升了，在第⼀阶段内存也已经为他开辟好了空间，但是因为这个声明的特性导致了并不能在声 明前使⽤

1. 什么是单线程，和异步的关系

单线程 - 只有⼀个线程，只能做⼀件事

原因 - 避免 DOM 渲染的冲突浏览器需要渲染 DOM

JS 可以修改 DOM 结构

JS 执⾏的时候，浏览器 DOM 渲染会暂停

两段 JS 也不能同时执⾏（都修改 DOM 就冲突了）

webworker ⽀持多线程，但是不能访问 DOM

解决⽅案 - 异步

1. 是否⽤过 jQuery 的 Deferred

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| image.png | image.png | image.png | image.png | image.png | image.png |
| image.png | image.png | image.png | image.png |  |  |

1. 前端⾯试之hybrid

<http://blog.poetries.top/2018/10/20/fe-interview-hybrid/>

1. 前端⾯试之组件化

<http://blog.poetries.top/2018/10/20/fe-interview-component/>

1. 前端⾯试之MVVM浅析

<http://blog.poetries.top/2018/10/20/fe-interview-mvvm/>

1. 实现效果，点击容器内的图标，图标边框变成border 1px solid red，点击空⽩处重置

js

const box = document.getElementById('box'); function isIcon(target) {

return target.className.includes('icon');

}

box.onClick = function(e) { e.stopPropagation();

const target = e.target; if (isIcon(target)) {

target.style.border = '1px solid red';

}

}

const doc = document; doc.onclick = function(e) {

const children = box.children; for(let i; i < children.length; i++) {

if (isIcon(children[i])) { children[i].style.border = 'none';

}

}

}

1. 请简单实现双向数据绑定 **mvvm**

<input id="input"/>

const data = {};

const input = document.getElementById('input'); Object.defineProperty(data, 'text', {

set(value) { input.value = value; this.value = value;

html

js

}

});

input.onChange = function(e) { data.text = e.target.value;

}

1. 实现Storage，使得该对象为单例，并对 **localStorage** 进⾏封装设置值setItem(key,value)和getItem(key)

js

var instance = null;

class Storage {

static getInstance() { if (!instance) {

instance = new Storage();

}

return instance;

}

setItem = (key, value) => localStorage.setItem(key, value), getItem = key => localStorage.getItem(key)

}

1. 说 说 **event loop**

⾸先， js 是单线程的，主要的任务是处理⽤户的交互，⽽⽤户的交互⽆⾮就是响应 DOM 的增删改，使⽤事件队列的形式，⼀次事件循环只处理⼀个事件响应，使得脚本执⾏相对连续，所以有了事件队列，⽤来储存待执⾏的事件， 那么事件队列的事件从哪⾥被 push 进来的呢。那就是另外⼀个线程叫事件触发线程做的事情了，他的作⽤主要是在定时触发器线程、异步 HTTP 请求线程满⾜特定条件下的回调函数 push 到事件队列中，等待 js 引擎空闲的时候去执⾏，当然js引擎执⾏过程中有优先级之分，⾸先js引擎在⼀次事件循环中， 会先执⾏js线程的主任务，然后会去查找是否有微任务microtask（promise） ，如果有那就优先执⾏微任务，如果没有，在去查找宏任务 macrotask（setTimeout、setInterval） 进⾏执⾏

#### 说说事件流

事件流分为两种，捕获事件流和冒泡事件流

###### 捕获事件流从根节点开始执⾏，⼀直往⼦节点查找执⾏，直到查找执⾏到⽬标节点 冒泡事件流从⽬标节点开始执⾏，⼀直往⽗节点冒泡查找执⾏，直到查到到根节点

事件流分为三个阶段，⼀个是捕获节点，⼀个是处于⽬标节点阶段，⼀个是冒泡阶段

1. 为什么 **canvas** 的图⽚为什么过有跨域问题
2. 我现在有⼀个 **canvas** ，上⾯随机布着⼀些⿊块，请实现⽅法，计算canvas上有多少个⿊块

https://[www.jianshu.com/p/f54d265f7aa4](http://www.jianshu.com/p/f54d265f7aa4)

1. 请⼿写实现⼀个 **promise**

##### https://segmentfault.com/a/1190000013396601

1. 说说从输⼊URL到看到⻚⾯发⽣的全过程，越详细越好

###### ⾸先浏览器主进程接管，开了⼀个下载线程。

然后进⾏HTTP请求（DNS查询、IP寻址等等），中间会有三次捂⼿，等待响应，开始下载响应报⽂。

将下载完的内容转交给Renderer进程管理。

Renderer进程开始解析css rule tree和dom tree，这两个过程是并⾏的，所以⼀般我会把link标签放在⻚⾯顶部。

解析绘制过程中，当浏览器遇到link标签或者script、img等标签，浏览器会去下载这些内 容，遇到时候缓存的使⽤缓存，不适⽤缓存的重新下载资源。

css rule tree和dom tree⽣成完了之后，开始合成render tree，这个时候浏览器会进⾏layout，开始计算每⼀个节点的位置，然后进⾏绘制。

绘制结束后，关闭TCP连接，过程有四次挥⼿

1. 描述⼀下 **this**

this ，函数执⾏的上下⽂，可以通过 apply ， call ， bind 改变 this 的指向。对于匿名函数或者直接调⽤的函数来说，this指向全局上下⽂（浏览 器为window，NodeJS为 global ），剩下的函数调⽤，那就是谁调⽤它， this 就指向谁。当然还有es6的箭头函数，箭头函数的指向取决于该箭头函数声明的位置，在哪⾥声明， this 就指向哪⾥

#### 说⼀下浏览器的缓存机制

浏览器缓存机制有两种，⼀种为强缓存，⼀种为协商缓存

对于强缓存，浏览器在第⼀次请求的时候，会直接下载资源，然后缓存在本地，第⼆次请 求的时候，直接使⽤缓存。

对于协商缓存，第⼀次请求缓存且保存缓存标识与时间，重复请求向服务器发送缓存标识 和最后缓存时间，服务端进⾏校验，如果失效则使⽤缓存

协商缓存相关设置

Exprires ：服务端的响应头，第⼀次请求的时候，告诉客户端，该资源什么时候会过期。 Exprires 的缺陷是必须保证服务端时间和客户端时间严格同步。

Cache-control：max-age ：表示该资源多少时间后过期，解决了客户端和服务端时间必须同步的问题，

If-None-Match/ETag ：缓存标识，对⽐缓存时使⽤它来标识⼀个缓存，第⼀次请求的时候，服务端会返回该标识给客户端，客户端在第⼆次请求的时候会带上该标识与服务端进

⾏对⽐并返回 If-None-Match 标识是否表示匹配。

Last-modified/If-Modified-Since ：第⼀次请求的时候服务端返回 Last-modified 表明请求的资源上次的修改时间，第⼆次请求的时候客户端带上请求头 If-Modified- Since ，表示资源上次的修改时间，服务端拿到这两个字段进⾏对⽐

1. 现在要你完成⼀个Dialog组件，说说你设计的思路？它应该有什 么功能？

该组件需要提供 hook 指定渲染位置，默认渲染在body下⾯。然后改组件可以指定外层样式，如宽度等

组件外层还需要⼀层 mask 来遮住底层内容，点击 mask 可以执⾏传进来的 onCancel 函数关闭 Dialog 。

另外组件是可控的，需要外层传⼊ visible 表示是否可⻅。

然后 Dialog 可能需要⾃定义头head和底部 footer ，默认有头部和底部，底部有⼀个确认按钮和取消按钮，确认按钮会执⾏外部传进来的 onOk 事件，然后取消按钮会执⾏外部传进来的 onCancel 事件。

当组件的 visible 为 true 时候，设置 body 的 overflow 为 hidden ，隐藏 body 的滚动条，反之显示滚动条。

###### 组件⾼度可能⼤于⻚⾯⾼度，组件内部需要滚动条。

只有组件的 visible 有变化且为 ture 时候，才重渲染组件内的所有内容

1. **caller** 和 **callee** 的区别

callee

caller 返回⼀个函数的引⽤，这个函数调⽤了当前的函数。使⽤这个属性要注意

###### 这个属性只有当函数在执⾏时才有⽤

如果在 javascript 程序中，函数是由顶层调⽤的，则返回 null functionName.caller: functionName 是当前正在执⾏的函数。

js

function a() {

console.log(a.caller)

}

##### callee

callee 放回正在执⾏的函数本身的引⽤，它是 arguments 的⼀个属性

使⽤callee时要注意:

###### 这个属性只有在函数执⾏时才有效

它有⼀个 length 属性，可以⽤来获得形参的个数，因此可以⽤来⽐较形参和实参个数是否⼀致，即⽐较 arguments.length 是否等于 arguments.callee.length

###### 它可以⽤来递归匿名函数。

js

function a() {

console.log(arguments.callee)

}

1. ajax、axios、fetch区别

##### jQuery ajax

js

$.ajax({

type: 'POST', url: url, data: data,

dataType: dataType, success: function () {}, error: function () {}

});

优缺点：

本身是针对 MVC 的编程,不符合现在前端 MVVM 的浪潮

基于原⽣的 XHR 开发， XHR 本身的架构不清晰，已经有了 fetch 的替代⽅案

JQuery 整个项⽬太⼤，单纯使⽤ ajax 却要引⼊整个 JQuery ⾮常的不合理（采取个性化打包的⽅案⼜不能享受CDN服务）

##### axios

js

axios({

method: 'post', url: '/user/12345', data: {

firstName: 'Fred', lastName: 'Flintstone'

}

})

.then(function (response) { console.log(response);

})

.catch(function (error) { console.log(error);

});

优缺点：

从浏览器中创建 XMLHttpRequest

从 node.js 发 出 http 请 求

⽀ 持 Promise API

###### 拦截请求和响应

转换请求和响应数据取消请求

⾃动转换 JSON 数据

客户端⽀持防⽌ CSRF/XSRF

##### fetch

js

try {

let response = await fetch(url); let data = response.json(); console.log(data);

} catch(e) {

console.log("Oops, error", e);

}

优缺点：

fetcht 只对⽹络请求报错，对 400 ， 500 都当做成功的请求，需要封装去处理

fetch 默认不会带 cookie ，需要添加配置项

fetch 不⽀持 abort ，不⽀持超时控制，使⽤ setTimeout 及 Promise.reject 的实现的超时控制并不能阻⽌请求过程继续在后台运⾏，造成了量的浪费

fetch 没有办法原⽣监测请求的进度，⽽XHR可以

四、jQuery

1. 你觉得jQuery或zepto源码有哪些写的好的地⽅

jquery源码封装在⼀个匿名函数的⾃执⾏环境中，有助于防⽌变量的全局污染，然后通过传⼊window对象参数，可以使window对象作为局部变量使⽤，好处是当jquery中访问window对象的时候，就不⽤将作⽤域链退回到顶层作⽤域了，从⽽可以更快的访问window对象。同样，传⼊undefined参数，可以缩短查找undefined时的作⽤域链

js

(function( window, undefined ) {

//⽤⼀个函数域包起来，就是所谓的沙箱

//在这⾥边var定义的变量，属于这个函数域内的局部变量，避免污染全局

//把当前沙箱需要的外部变量通过函数参数引⼊进来

//只要保证参数对内提供的接⼝的⼀致性，你还可以随意替换传进来的这个参数window.jQuery = window.$ = jQuery;

})( window );

jquery将⼀些原型属性和⽅法封装在了jquery.prototype中，为了缩短名称，⼜赋值给了jquery.fn，这是很形象的写法

有⼀些数组或对象的⽅法经常能使⽤到，jQuery将其保存为局部变量以提⾼访问速度jquery实现的链式调⽤可以节约代码，所返回的都是同⼀个对象，可以提⾼代码效率

1. jQuery 的实现原理

(function(window, undefined) {})(window);

jQuery 利⽤ JS 函数作⽤域的特性，采⽤⽴即调⽤表达式包裹了⾃身，解决命名空间和变量污染问题

window.jQuery = window.$ = jQuery;

在闭包当中将 jQuery 和 $ 绑定到 window 上，从⽽将 jQuery 和 $ 暴露为全局变量

1. **jQuery.fn** 的 **init** ⽅法返回的 **this** 指的是什么对象

jQuery.fn 的 init ⽅法 返回的 this 就是 jQuery 对象

⽤户使⽤ jQuery() 或 $() 即可初始化 jQuery 对象，不需要动态的去调⽤ init ⽅法

## jQuery.extend 与 jQuery.fn.extend 的区别

$.fn.extend() 和 $.extend() 是 jQuery 为扩展插件提拱了两个⽅法

$.extend(object) ; // 为jQuery添加“静态⽅法”（⼯具⽅法）

js

$.extend({

min: function(a, b) { return a < b ? a : b; }, max: function(a, b) { return a > b ? a : b; }

});

$.min(2,3); // 2

$.max(4,5); // 5

##### $.extend([true,] targetObject, object1[, object2]); // 对targt对象进⾏扩展

js

var settings = {validate:false, limit:5};

var options = {validate:true, name:"bar"};

$.extend(settings, options); // 注意：不⽀持第⼀个参数传 false

// settings == {validate:true, limit:5, name:"bar"}

$.fn.extend(json) ; // 为jQuery添加“成员函数”（实例⽅法）

js

$.fn.extend({

alertValue: function() {

$(this).click(function(){ alert($(this).val());

});

}

});

$("#email").alertValue();

1. jQuery 的属性拷⻉(extend)的实现原理是什么，如何实现深拷⻉

浅拷⻉（只复制⼀份原始对象的引⽤） var newObject = $.extend({}, oldObject);

深拷⻉（对原始对象属性所引⽤的对象进⾏进⾏递归拷⻉） var newObject =

$.extend(true, {}, oldObject);

1. jQuery 的队列是如何实现的

jQuery 核⼼中有⼀组队列控制⽅法，由 queue()/dequeue()/clearQueue() 三个⽅法组成。

主要应⽤于 animate() ， ajax ，其他要按时间顺序执⾏的事件中

js

var func1 = function(){alert('事件1');}

var func2 = function(){alert('事件2');} var func3 = function(){alert('事件3');} var func4 = function(){alert('事件4');}

// ⼊栈队列事件

$('#box').queue("queue1", func1); // push func1 to queue1

$('#box').queue("queue1", func2); // push func2 to queue1

// 替换队列事件

$('#box').queue("queue1", []); // delete queue1 with empty array

$('#box').queue("queue1", [func3, func4]); // replace queue1

// 获取队列事件（返回⼀个函数数组）

$('#box').queue("queue1"); // [func3(), func4()]

// 出栈队列事件并执⾏

$('#box').dequeue("queue1"); // return func3 and do func3

$('#box').dequeue("queue1"); // return func4 and do func4

// 清空整个队列

$('#box').clearQueue("queue1"); // delete queue1 with clearQueue

## jQuery 中的 bind(), live(), delegate(), on()的区别

bind() 直接绑定在⽬标元素上

live() 通过冒泡传播事件，默认 document 上，⽀持动态数据

delegate() 更精确的⼩范围使⽤事件代理，性能优于 live

on() 是最新的 1.9 版本整合了之前的三种⽅式的新事件绑定机制

#### 是否知道⾃定义事件

事件即“发布/订阅”模式，⾃定义事件即“消息发布”，事件的监听即“订阅订阅” JS 原⽣⽀持⾃定义事件，示例：

js

document.createEvent(type); // 创建事件

event.initEvent(eventType, canBubble, prevent); // 初始化事件target.addEventListener('dataavailable', handler, false); // 监 听 事 件target.dispatchEvent(e); // 触发事件

jQuery ⾥的 fire 函数⽤于调⽤ jQuery ⾃定义事件列表中的事件

1. jQuery 通过哪个⽅法和 Sizzle 选择器结合的

Sizzle 选择器采取 Right To Left 的匹配模式，先搜寻所有匹配标签，再判断它的⽗节点

jQuery 通过 $(selecter).find(selecter); 和 Sizzle 选择器结合

1. jQuery 中如何将数组转化为 JSON 字符串，然后再转化回来

js

// 通过原⽣ JSON.stringify/JSON.parse 扩展 jQuery 实现

$.array2json = function(array) { return JSON.stringify(array);

}

$.json2array = function(array) {

// $.parseJSON(array); // 3.0 开始，已过时

return JSON.parse(array);

}

// 调 ⽤

var json = $.array2json(['a', 'b', 'c']); var array = $.json2array(json);

1. jQuery ⼀个对象可以同时绑定多个事件，这是如何实现的

js

$("#btn").on("mouseover mouseout", func);

$("#btn").on({ mouseover: func1, mouseout: func2, click: func3

});

1. 针对 jQuery 的优化⽅法

缓存频繁操作 DOM 对象

尽量使⽤ id 选择器代替 class 选择器总是从 #id 选择器来继承

###### 尽量使⽤链式操作

使⽤时间委托 on 绑定事件

采⽤ jQuery 的内部函数 data() 来存储数据使⽤最新版本的 jQuery

1. jQuery 的 slideUp 动画，当⿏标快速连续触发, 动画会滞后反复执

⾏，该如何处理呢

在触发元素上的事件设置为延迟处理：使⽤ JS 原⽣ setTimeout ⽅法在触发元素的事件时预先停⽌所有的动画，再执⾏相应的动画事件：

$('.tab').stop().slideUp();

1. jQuery UI 如何⾃定义组件

通过向 $.widget() 传递组件名称和⼀个原型对象来完成

$.widget("ns.widgetName", [baseWidget], widgetPrototype);

## jQuery 与 jQuery UI、jQuery Mobile 区别

jQuery 是 JS 库，兼容各种PC浏览器，主要⽤作更⽅便地处理 DOM 、事件、动画、

AJAX

jQuery UI 是建⽴在 jQuery 库上的⼀组⽤户界⾯交互、特效、⼩部件及主题

jQuery Mobile 以 jQuery 为基础，⽤于创建“移动Web应⽤”的框架

1. jQuery 和 Zepto 的区别？ 各⾃的使⽤场景

jQuery 主要⽬标是 PC 的⽹⻚中，兼容全部主流浏览器。在移动设备⽅⾯，单独推出

##### `jQuery Mobile

Zepto 从⼀开始就定 位移动设备，相对更轻量级。它的 API 基本兼容 jQuery`，但对PC浏览器兼容不理想

1. jQuery对象的特点

只有 JQuery 对象才能使⽤ JQuery ⽅法

JQuery 对象是⼀个数组对象

# 五、Bootstrap

1. 什么是Bootstrap？以及为什么要使⽤Bootstrap？

Bootstrap 是⼀个⽤于快速开发 Web 应⽤程序和⽹站的前端框架。

Bootstrap 是 基 于 HTML 、 CSS 、 JAVASCRIPT 的

Bootstrap 具有移动设备优先、浏览器⽀持良好、容易上⼿、响应式设计等优点，所以

Bootstrap 被⼴泛应⽤

1. 使⽤Bootstrap时，要声明的⽂档类型是什么？以及为什么要这样声 明？

使⽤ Bootstrap 时，需要使⽤ HTML5 ⽂档类型（ Doctype ）。 <!DOCTYPE html>

因为 Bootstrap 使⽤了⼀些 HTML5 元素和 CSS 属性，如果在 Bootstrap 创建的⽹

⻚开头不使⽤ HTML5 的⽂档类型（ Doctype ），可能会⾯临⼀些浏览器显示不⼀致的问题，甚⾄可能⾯临⼀些特定情境下的不⼀致，以致于代码不能通过 W3C 标准的验证

1. 什么是Bootstrap⽹格系统

Bootstrap 包含了⼀个响应式的、移动设备优先的、不固定的⽹格系统，可以随着设备或视⼝⼤⼩的增加⽽适当地扩展到 12 列。它包含了⽤于简单的布局选项的预定义类，也包含了⽤于⽣成更多语义布局的功能强⼤的混合类

响应式⽹格系统随着屏幕或视⼝（ viewport ）尺⼨的增加，系统会⾃动分为最多 12

###### 列。

1. Bootstrap ⽹格系统（Grid System）的⼯作原理
   1. ⾏必须放置在 .container class 内，以便获得适当的对⻬（ alignment ）和内边距（ padding ）。

###### 使⽤⾏来创建列的⽔平组。

* 1. 内容应该放置在列内，且唯有列可以是⾏的直接⼦元素。
  2. 预定义的⽹格类，⽐如 .row 和 .col-xs-4 ，可⽤于快速创建⽹格布局。 LESS

###### 混合类可⽤于更多语义布局。

* 1. 列通过内边距（ padding ）来创建列内容之间的间隙。该内边距是通过 .rows 上的外边距（ margin ）取负，表示第⼀列和最后⼀列的⾏偏移。
  2. ⽹格系统是通过指定您想要横跨的⼗⼆个可⽤的列来创建的。例如，要创建三个相等 的列，则使⽤三个 .col-xs-4

1. 对于各类尺⼨的设备，Bootstrap设置的class前缀分别是什么

超⼩设备⼿机（ <768px ）： .col-xs-\*

⼩型设备平板电脑（ >=768px ）： .col-sm-\*

中型设备台式电脑（ >=992px ）： .col-md-\*

⼤型设备台式电脑（ >=1200px ）： .col-lg-\*

1. Bootstrap ⽹格系统列与列之间的间隙宽度是多少

间隙宽度为 30px （⼀个列的每边分别是 15px ）

1. 如果需要在⼀个标题的旁边创建副标题，可以怎样操作

在元素两旁添加 <small> ，或者添加 .small 的 class

1. ⽤Bootstrap，如何设置⽂字的对⻬⽅式？

class="text-center" 设置居中⽂本class="text-right" 设置向右对⻬⽂本class="text-left" 设置向左对⻬⽂本

1. Bootstrap如何设置响应式表格？

增 加 class="table-responsive"

1. 使⽤Bootstrap创建垂直表单的基本步骤？
2. 向⽗ <form> 元素添加 role="form" ；
3. 把标签和控件放在⼀个带有 class="form-group" 的 <div> 中，这是获取最佳间距所必需的；
4. 向所有的⽂本元素 <input> 、 <textarea> 、 <select> 添加 class="form- control"
5. 使⽤Bootstrap创建⽔平表单的基本步骤？
6. 向⽗ <form> 元素添加 class="form-horizontal" ；
7. 把标签和控件放在⼀个带有 class="form-group" 的 <div> 中；
8. 向标签添加 class="control-label" 。
9. 使⽤Bootstrap如何创建表单控件的帮助⽂本？

增加 class="help-block" 的 span 标签或 p 标签。

1. 使⽤Bootstrap激活或禁⽤按钮要如何操作？

激活按钮：给按钮增加 .active 的 class

禁⽤按钮：给按钮增加 disabled="disabled" 的属性

1. Bootstrap有哪些关于的class？
2. .img-rounded 为图⽚添加圆⻆
3. .img-circle 将图⽚变为圆形
4. .img-thumbnail 缩略图功能
5. .img-responsive 图⽚响应式 (将很好地扩展到⽗元素)
6. Bootstrap中有关元素浮动及清除浮动的class？
7. class="pull-left" 元素浮动到左边
8. class="pull-right" 元素浮动到右边
9. class="clearfix" 清除浮动
10. 除了屏幕阅读器外，其他设备上隐藏元素的class？

##### `class="sr-only"``

1. Bootstrap如何制作下拉菜单？
2. 将下拉菜单包裹在 class="dropdown" 的 <div> 中；
3. 在触发下拉菜单的按钮中添加： class="btn dropdown-toggle" id="dropdownMenu1" data-toggle="dropdown"
4. 在包裹下拉菜单的ul中添加： class="dropdown-menu" role="menu" aria- labelledby="dropdownMenu1"
5. 在下拉菜单的列表项中添加： role="presentation" 。其中，下拉菜单的标题要添加 class="dropdown-header" ，选项部分要添加 tabindex="-1" 。
6. Bootstrap如何制作按钮组？以及⽔平按钮组和垂直按钮组的优先 级？
7. ⽤ class="btn-group" 的 <div> 去包裹按钮组； class="btn-group- vertical" 可设置垂直按钮组。
8. btn-group 的优先级⾼于 btn-group-vertical 的优先级。
9. Bootstrap如何设置按钮的下拉菜单？

在⼀个 .btn-group 中放置按钮和下拉菜单即可。

1. Bootstrap中的输⼊框组如何制作？
2. 把前缀或者后缀元素放在⼀个带有 class="input-group" 中的 <div> 中
3. 在该 <div> 内，在 class="input-group-addon" 的 <span> ⾥⾯放置额外的内容；
4. 把 <span> 放在 <input> 元素的前⾯或后⾯。
5. Bootstrap中的导航都有哪些？
6. 导航元素：有 class="nav nav-tabs" 的标签⻚导航，还有 class="nav nav- pills" 的胶囊式标签⻚导航；
7. 导 航 栏 ： class="navbar navbar-default" role="navigation" ；
8. ⾯包屑导航： class="breadcrumb"
9. Bootstrap中设置分⻚的class？

默认的分⻚： class="pagination"

默认的翻⻚： class="pager"

1. Bootstrap中显示标签的class？

class="label"

1. Bootstrap中如何制作徽章？

<span class="badge">26</span>

1. Bootstrap中超⼤屏幕的作⽤是什么？

设置 class="jumbotron" 可以制作超⼤屏幕，该组件可以增加标题的⼤⼩并增加更多的外边距

六 微信⼩程序

、

#### 微信⼩程序有⼏个⽂件

WXSS (WeiXin Style Sheets) 是⼀套样式语⾔，⽤于描述 WXML 的组件样式， js

逻辑处理，⽹络请求 json ⼩程序设置，如⻚⾯注册，⻚⾯标题及 tabBar 。

app.json 必须要有这个⽂件，如果没有这个⽂件，项⽬⽆法运⾏，因为微信框架把这个作为配置⽂件⼊⼝，整个⼩程序的全局配置。包括⻚⾯注册，⽹络设置，以及⼩程序的window 背景⾊，配置导航条样式，配置默认标题。

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

#### 微信⼩程序怎样跟事件传值

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

1. ⼩程序的 wxss 和 css 有哪些不⼀样的地⽅？

wxss 的图⽚引⼊需使⽤外链地址

没有 Body ；样式可直接使⽤ import 导⼊

#### ⼩程序关联微信公众号如何确定⽤户的唯⼀性

使⽤ wx.getUserInfo ⽅法 withCredentials 为 true 时 可获取

encryptedData ，⾥⾯有 union\_id 。后端需要进⾏对称解密

1. 微信⼩程序与vue区别

###### ⽣命周期不⼀样，微信⼩程序⽣命周期⽐较简单

数据绑定也不同，微信⼩程序数据绑定需要使⽤ {{}} ， vue 直接 : 就可以

显示与隐藏元素， vue 中，使⽤ v-if 和 v-show 控制元素的显示和隐藏，⼩程序中，使⽤ wx-if 和 hidden 控制元素的显示和隐藏

事 件 处 理 不 同 ， ⼩ 程 序 中 ， 全 ⽤ bindtap(bind+event) ， 或 者 catchtap(catch+event) 绑定事件, vue： 使⽤ v-on:event 绑定事件，或者使⽤@event 绑定事件

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

七、webpack相关

#### 打包体积 优化思路

提取第三⽅库或通过引⽤外部⽂件的⽅式引⼊第三⽅库代码压缩插件 UglifyJsPlugin

服务器启⽤gzip压缩

按需加载资源⽂件 require.ensure

优 化 devtool 中 的 source-map

剥离 css ⽂件，单独打包

###### 去除不必要插件，通常就是开发环境与⽣产环境⽤同⼀套配置⽂件导致

1. 打包效率

开发环境采⽤增量构建，启⽤热更新

开发环境不做⽆意义的⼯作如提取 css 计算⽂件hash等配置 devtool

选择合适的 loader

个 别 loader 开 启 cache 如 babel-loader

###### 第三⽅库采⽤引⼊⽅式提取公共代码

优化构建时的搜索路径 指明需要构建⽬录及不需要构建⽬录模块化引⼊需要的部分

1. Loader

编写⼀个loader

loader 就是⼀个 node 模块，它输出了⼀个函数。当某种资源需要⽤这个loader 转换时，这个函数会被调⽤。并且，这个函数可以通过提供给它的this 上 下 ⽂ 访 问 Loader API 。 reverse-txt-loader

js

// 定 义

module.exports = function(src) {

//src是原⽂件内容（abcde），下⾯对内容进⾏处理，这⾥是反转var result = src.split('').reverse().join('');

//返回JavaScript源码，必须是String或者Buffer return `module.exports = '${result}'`;

}

//使⽤

{

test: /\.txt$/, use: [

{

'./path/reverse-txt-loader'

}

]

},

1. 说⼀下webpack的⼀些plugin，怎么使⽤webpack对项⽬进⾏优化

###### 构建优化

减少编译体积 ContextReplacementPugin 、 IgnorePlugin 、 babel-plugin- import 、 babel-plugin-transform-runtime

并⾏编译 happypack 、 thread-loader 、 uglifyjsWebpackPlugin 开启并⾏

缓 存 cache-loader 、 hard-source-webpack-plugin 、 uglifyjsWebpackPlugin 开启缓存、 babel-loader 开启缓存

预 编 译 dllWebpackPlugin && DllReferencePlugin 、 auto-dll-webapck-plugin

###### 性能优化

减少编译体积 Tree-shaking 、 Scope Hositing hash 缓 存 webpack-md5-plugin

拆 包 splitChunksPlugin 、 import() 、 require.ensure

### ⼋ 编程题

、

1. 写⼀个通⽤的事件侦听器函数

js

// event(事件)⼯具集，来源：github.com/markyun

markyun.Event = {

// 视能⼒分别使⽤dom0||dom2||IE⽅式 来绑定事件

// 参数： 操作的元素,事件名称 ,事件处理程序

addEvent : function(element, type, handler) { if (element.addEventListener) {

//事件类型、需要执⾏的函数、是否捕捉

element.addEventListener(type, handler, false);

} else if (element.attachEvent) { element.attachEvent('on' + type, function() {

handler.call(element);

});

} else {

element['on' + type] = handler;

}

},

// 移除事件

removeEvent : function(element, type, handler) { if (element.removeEventListener) {

element.removeEventListener(type, handler, false);

} else if (element.datachEvent) { element.detachEvent('on' + type, handler);

} else {

element['on' + type] = null;

}

},

// 阻⽌事件 (主要是事件冒泡，因为IE不⽀持事件捕获) stopPropagation : function(ev) {

if (ev.stopPropagation) { ev.stopPropagation();

} else {

ev.cancelBubble = true;

}

},

// 取消事件的默认⾏为

preventDefault : function(event) { if (event.preventDefault) {

event.preventDefault();

} else {

event.returnValue = false;

}

},

// 获取事件⽬标

getTarget : function(event) {

return event.target || event.srcElement;

}

1. 如何判断⼀个对象是否为数组

js

function isArray(arg) {

if (typeof arg === 'object') {

return Object.prototype.toString.call(arg) === '[object Array]';

}

return false;

}

#### 冒泡排序

每次⽐较相邻的两个数，如果后⼀个⽐前⼀个⼩，换位置

js

var arr = [3, 1, 4, 6, 5, 7, 2];

function bubbleSort(arr) {

for (var i = 0; i < arr.length - 1; i++) { for(var j = 0; j < arr.length - i - 1; j++) {

if(arr[j + 1] < arr[j]) { var temp;

temp = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] = temp;

}

}

}

return arr;

}

console.log(bubbleSort(arr));

#### 快速排序

###### 采⽤⼆分法，取出中间数，数组每次和中间数⽐较，⼩的放到左边，⼤的放到右边

js

var arr = [3, 1, 4, 6, 5, 7, 2];

function quickSort(arr) { if(arr.length == 0) {

return []; // 返回空数组

}

var cIndex = Math.floor(arr.length / 2); var c = arr.splice(cIndex, 1);

var l = [];

var r = [];

for (var i = 0; i < arr.length; i++) { if(arr[i] < c) {

l.push(arr[i]);

} else {

r.push(arr[i]);

}

}

return quickSort(l).concat(c, quickSort(r));

}

console.log(quickSort(arr));

1. 编写⼀个⽅法 求⼀个字符串的字节⻓度

假设：⼀个英⽂字符占⽤⼀个字节，⼀个中⽂字符占⽤两个字节

js

function GetBytes(str){

var len = str.length; var bytes = len;

for(var i=0; i<len; i++){

if (str.charCodeAt(i) > 255) bytes++;

}

return bytes;

}

alert(GetBytes("你好,as"));

1. bind的⽤法，以及如何实现bind的函数和需要注意的点

bind 的作⽤与 call 和 apply 相同，区别是 call 和 apply 是⽴即调⽤函数，⽽

bind 是返回了⼀个函数，需要调⽤的时候再执⾏。 ⼀个简单的 bind 函数实现如下

js

Function.prototype.bind = function(ctx) { var fn = this;

return function() { fn.apply(ctx, arguments);

};

};

1. 实现⼀个函数clone

可以对 JavaScript 中的5种主要的数据类型,包括 Number 、 String 、

Object 、 Array 、 Boolean ）进⾏值复

###### 考察点1：对于基本数据类型和引⽤数据类型在内存中存放的是值还是指针这⼀区别是否清 楚

考察点2：是否知道如何判断⼀个变量是什么类型的

考察点3：递归算法的设计

js

// ⽅法⼀：

Object.prototype.clone = function(){

var o = this.constructor === Array ? [] : {}; for(var e in this){

o[e] = typeof this[e] === "object" ? this[e].clone() : th

}

return o;

}

//⽅法⼆：

/\*\*

* 克隆⼀个对象
* @param Obj
* @returns

\*/

function clone(Obj) { var buf;

if (Obj instanceof Array) {

buf = []; //创建⼀个空的数组var i = Obj.length;

while (i--) {

buf[i] = clone(Obj[i]);

}

return buf;

}else if (Obj instanceof Object){

buf = {}; //创建⼀个空对象

for (var k in Obj) { //为这个对象添加新的属性

buf[k] = clone(Obj[k]);

}

return buf;

}else{ //普通变量直接赋值

return Obj;

}

}

1. 下⾯这个ul，如何点击每⼀列的时候alert其index

考察闭包

<ul id=”test”>

<li>这是第⼀条</li>

<li>这是第⼆条</li>

<li>这是第三条</li>

</ul>

// ⽅法⼀：

var lis=document.getElementById('2223').getElementsByTagName('li'); for(var i=0;i<3;i++)

{

html

js

lis[i].index=i; lis[i].onclick=function(){

alert(this.index);

}

//⽅法⼆：

var lis=document.getElementById('2223').getElementsByTagName('li'); for(var i=0;i<3;i++)

{

lis[i].index=i; lis[i].onclick=(function(a){

return function() { alert(a);

}

})(i);

}

1. 定义⼀个log⽅法，让它可以代理console.log的⽅法

js

可⾏的⽅法⼀：

function log(msg) { console.log(msg);

}

log("hello world!") // hello world!

如果要传⼊多个参数呢？显然上⾯的⽅法不能满⾜要求，所以更好的⽅法是：

js

function log(){

console.log.apply(console, arguments);

};

#### 输出今天的⽇期

以 YYYY-MM-DD 的⽅式，⽐如今天是2014年9⽉26⽇，则输出2014-09-26

js

var d = new Date();

// 获取年，getFullYear()返回4位的数字

var year = d.getFullYear();

// 获取⽉，⽉份⽐较特殊，0是1⽉，11是12⽉

var month = d.getMonth() + 1;

// 变成两位

month = month < 10 ? '0' + month : month;

// 获取⽇

var day = d.getDate();

day = day < 10 ? '0' + day : day; alert(year + '-' + month + '-' + day);

1. ⽤js实现随机选取10–100之间的10个数字，存⼊⼀个数组，并排序

js

var iArray = [];

funtion getRandom(istart, iend){

var iChoice = istart - iend +1;

return Math.floor(Math.random() \* iChoice + istart;

}

for(var i=0; i<10; i++){

iArray.push(getRandom(10,100));

}

iArray.sort();

1. 写⼀段JS程序提取URL中的各个GET参数

有 这 样 ⼀ 个 URL ： [http://item.taobao.com/item.htm?](http://item.taobao.com/item.htm) a=1&b=2&c=&d=xxx&e ，请写⼀段JS程序提取URL中的各个GET参数(参数名和

参数个数不确定)，将其按 key-value 形式返回到⼀个 json 结构中，如

{a:'1', b:'2', c:'', d:'xxx', e:undefined}

js

function serilizeUrl(url) {

var result = {};

url = url.split("?")[1]; var map = url.split("&");

for(var i = 0, len = map.length; i < len; i++) { result[map[i].split("=")[0]] = map[i].split("=")[1];

}

return result;

}

1. 写⼀个 **function** ，清除字符串前后的空格

使⽤⾃带接⼝ trim() ，考虑兼容性：

js

if (!String.prototype.trim) { String.prototype.trim = function() {

return this.replace(/^\s+/, "").replace(/\s+$/,"");

}

}

// test the function

var str = " \t\n test string ".trim(); alert(str == "test string"); // alerts "true"

1. 实现每隔⼀秒钟输出1,2,3...数字

js

for(var i=0;i<10;i++){ (function(j){

setTimeout(function(){ console.log(j+1)

},j\*1000)

})(i)

}

1. 实现⼀个函数，判断输⼊是不是回⽂字符串

js

function run(input) {

if (typeof input !== 'string') return false;

return input.split('').reverse().join('') === input;

}

16、数组扁平化处理

实现⼀个 flatten ⽅法，使得输⼊⼀个数组，该数组⾥⾯的元素也可以是数组，该⽅法会输出⼀个扁平化的数组

js

function flatten(arr){

return arr.reduce(function(prev,item){

return prev.concat(Array.isArray(item)?flatten(item):item);

},[]);

}

### 九 其他

、

#### 负载均衡

多台服务器共同协作，不让其中某⼀台或⼏台超额⼯作，发挥服务器的最⼤作

⽤

http 重定向负载均衡：调度者根据策略选择服务器以302响应请求，缺点只有第⼀次有效果，后续操作维持在该服务器 dns负载均衡：解析域名时，访问多个 ip 服务器中的⼀个（可监控性较弱）

###### 反向代理负载均衡：访问统⼀的服务器，由服务器进⾏调度访问实际的某个服务器，对统

⼀的服务器要求⼤，性能受到 服务器群的数量

1. CDN

内容分发⽹络，基本思路是尽可能避开互联⽹上有可能影响数据传输速度和稳 定性的瓶颈和环节，使内容传输的更快、更稳定。

1. 内存泄漏

定义：程序中⼰动态分配的堆内存由于某种原因程序未释放或⽆法释放引发的 各种问题。

js中可能出现的内存泄漏情况

结果：变慢，崩溃，延迟⼤等，原因：

###### 全局变量

dom 清空时，还存在引⽤

ie 中使⽤闭包定时器未清除

###### ⼦元素存在引起的内存泄露避免策略

减少不必要的全局变量，或者⽣命周期较⻓的对象，及时对⽆⽤的数据进⾏垃圾回收； 注意程序逻辑，避免“死循环”之类的 ；

避免创建过多的对象 原则：不⽤了的东⻄要及时归还。减少层级过多的引⽤

1. babel原理

ES6、7 代码输⼊ -> babylon 进⾏解析 -> 得到 AST （抽象语法树）->

plugin ⽤b abel-traverse 对 AST 树进⾏遍历转译 ->得到新的 AST 树->

⽤ babel-generator 通 过 AST 树 ⽣ 成 ES5 代 码

1. js⾃定义事件

三要素： document.createEvent() event.initEvent() element.dispatchEvent()

js

// (en:⾃定义事件名称，fn:事件处理函数，addEvent:为DOM元素添加⾃定义事件，triggerEv

window.onload = function(){

var demo = document.getElementById("demo"); demo.addEvent("test",function(){console.log("handler1")}); demo.addEvent("test",function(){console.log("handler2")}); demo.onclick = function(){

this.triggerEvent("test");

}

}

Element.prototype.addEvent = function(en,fn){ this.pools = this.pools || {};

if(en in this.pools){ this.pools[en].push(fn);

}else{

this.pools[en] = []; this.pools[en].push(fn);

}

}

Element.prototype.triggerEvent = function(en){ if(en in this.pools){

var fns = this.pools[en];

for(var i=0,il=fns.length;i<il;i++){ fns[i]();

}

}else{

return;

}

}

#### 前后端路由差别

###### 后端每次路由请求都是重新访问服务器

前端路由实际上只是 JS 根据 URL 来操作 DOM 元素，根据每个⻚⾯需要的去服务端请求数据，返回数据后和模板进⾏组合

⼗、综合

#### 谈谈你对重构的理解

###### ⽹站重构：在不改变外部⾏为的前提下，简化结构、添加可读性，⽽在⽹站前端保持⼀致的⾏为。也就是说是在不改变UI的情况下，对⽹站进⾏优化， 在扩展的同时保持⼀致的UI对于传统的⽹站来说重构通常是：

表格( table )布局改为 DIV+CSS

使⽹站前端兼容于现代浏览器(针对于不合规范的 CSS 、如对IE6有效的)对于移动平台的优化

针对于 SEO 进⾏优化

#### 什么样的前端代码是好的

###### ⾼复⽤低耦合，这样⽂件⼩，好维护，⽽且好扩展。具有可⽤性、健壮性、可靠性、宽容性等特点

遵循设计模式的六⼤原则

1. 对前端⼯程师这个职位是怎么样理解的？它的前景会怎么样

前端是最贴近⽤户的程序员，⽐后端、数据库、产品经理、运营、安全都近 实现界⾯交互

提升⽤户体验

基于NodeJS，可跨平台开发

前端是最贴近⽤户的程序员，前端的能⼒就是能让产品从 90分进化到 100 分，甚⾄更好， 与团队成员， UI 设计，产品经理的沟通；

###### 做好的⻚⾯结构，⻚⾯重构和⽤户体验；

1. 你觉得前端⼯程的价值体现在哪

为简化⽤户使⽤提供技术⽀持（交互部分） 为多个浏览器兼容性提供⽀持

为提⾼⽤户浏览速度（浏览器性能）提供⽀持

为跨平台或者其他基于webkit或其他渲染引擎的应⽤提供⽀持 为展示数据提供⽀持（数据接⼝）

1. 平时如何管理你的项⽬

先期团队必须确定好全局样式（ globe.css ），编码模式( utf-8 ) 等； 编写习惯必须⼀致（例如都是采⽤继承式的写法，单样式都写成⼀⾏）； 标注样式编写⼈，各模块都及时标注（标注关键样式调⽤的地⽅）；

###### ⻚⾯进⾏标注（例如 ⻚⾯ 模块 开始和结束）；

CSS 跟 HTML 分⽂件夹并⾏存放，命名都得统⼀（例如 style.css ）；

JS 分⽂件夹存放 命名以该 JS 功能为准的英⽂翻译。

图⽚采⽤整合的 images.png png8 格式⽂件使⽤ - 尽量整合在⼀起使⽤⽅便将来的管理

#### 组件封装

⽬的：为了重⽤，提⾼开发效率和代码质量 注意：低耦合，单⼀职责，可复⽤性，可维护性 常⽤操作

分析布局初步开发化繁为简组件抽象

⼗⼀、⼀些常⻅问题

###### ⾃我介绍

⾯试完你还有什么问题要问的吗你有什么爱好?

你最⼤的优点和缺点是什么?

你为什么会选择这个⾏业，职位? 你觉得你适合从事这个岗位吗? 你有什么职业规划?

###### 你对⼯资有什么要求? 如何看待前端开发？

未来三到五年的规划是怎样的？

你的项⽬中技术难点是什么？遇到了什么问题？你是怎么解决的？ 你们部⻔的开发流程是怎样的

你认为哪个项⽬做得最好？

说下⼯作中你做过的⼀些性能优化处理最近在看哪些前端⽅⾯的书？

平时是如何学习前端开发的？ 你最有成就感的⼀件事

你为什么要离开前⼀家公司？ 你对加班的看法

你希望通过这份⼯作获得什么？