# 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，不同页面 `title` 要有所不同； `description` 把页面内容高度概括， 长度合适，不可过分堆砌关键词，不同页面 `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页
- 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长度有限制，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

### 2 `<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。

### 3 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
  - 只请求页面的首部
- 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

## 4 从浏览器地址栏输入url到显示页面的步骤

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，渲染页面，完成。

详细版

1. 在浏览器地址栏输入URL
2. 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

3. 浏览器解析URL获取协议， 主机，端口， path
4. 浏览器组装一个HTTP（GET）请求报文
5. 浏览器获取主机ip地址， 过程如下：
    1. 浏览器缓存
    2. 本机缓存
    3. hosts文件
    4. 路由器缓存
    5. ISP DNS缓存
    6. DNS递归查询（可能存在负载均衡导致每次IP不一样)
6. 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
7. TCP链接建立后发送HTTP请求
8. 服务器接受请求并解析，将请求转发到服务程序， 如虚拟主机使用工TTP 工ost头部判断请求的服务程序
9. 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
10. 处理程序读取完整请求并准备工TTP响应， 可能需要查询数据库等操作
11. 服务器将响应报文通过TCP连接发送回浏览器
12. 浏览器接收工TTP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
13. 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
14. 如果资源可缓存， 进行缓存
15. 对响应进行解码（例如gzip压缩)
16. 根据资源类型决定如何处理（假设资源为工TML文档)
17. 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
18. 构建DOM树：
    1. Tokenizing：根据工TML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：根据工TML标记关系将对象组成DOM树
19. 解析过程中遇到图片 、样式表 、js文件，启动下载
20. 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

21. 根据DOM树和CSSOM树构建渲染树：
    1. 从DOM树的根节点遍历所有可见节点，不可见节点包括：1) `script`，`meta` 这样本身不可见的标签。2)被css隐藏的节点， 如 `display: none`
    2. 对每一个可见节点，找到恰当的CSSOM规则并应用
    3. 发布可视节点的内容和计算样式
22. 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
23. 显示页面（HTML解析过程中会逐步显示页面)

详细简版

1. 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)

2. 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)

3. 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)

4. 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)

5. 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)

6. 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)

7. 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)

8. 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)

9. 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)

## 5 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
  - 将样式表放到页面顶部
  - 不使用 CSS 表达式
  - 使用 <link> 不使用 @import
- Javascript 方面
  - 将脚本放到页面底部
  - 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

## 6 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
  - 200 OK 正常返回信息
  - 201 Created 请求成功并且服务器创建了新的资源
  - 202 Accepted 服务器已接受请求，但尚未处理
- 3XX ： 重定向
  - 301 Moved Permanently 请求的网页已永久移动到新位置。
  - 302 Found 临时性重定向。
  - 303 See Other 临时性重定向，且总是使用 GET 请求新的 URI 。
  - 304 Not Modified 自从上次请求后，请求的网页未修改过。
- 4XX ： 客户端错误
  - 400 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。
  - 401 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 最常见的服务器端错误。
  - 503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

## 7 语义化的理解

- 用正确的标签做正确的事情!
- HTML 语义化就是让页面的内容结构化，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

## 8 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：负责取得网页的内容（ HTML 、 XML 、图像等等）、整理讯息（例如加入 CSS 等)， 以及计算网页的显示方式，然后会输出至显示器或打印机 。浏览器的内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同 。所有网页浏览器 、电子邮件客户端以及其它需要编辑 、显示网络内容的应用程序都需要内核
- JS 引擎则：解析和执行 javascript 来实现网页的动态效果
- 最开始渲染引擎和 JS 引擎并没有区分的很明确，后来JS引擎越来越独立， 内核就倾向于只指渲染引擎

## 9 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 长期存储数据， 浏览器关闭后数据不丢失
  - sessionStorage 的数据在浏览器关闭后自动删除
  - 语意化更好的内容元素， 比如 article 、 footer 、 header 、 nav 、 section
  - 表单控件， calendar 、 date 、 time 、 email 、 url 、 search
  - 新的技术 webworker 、 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

## 10 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页面展示
- 如何使用:
  - 页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
```

## 11 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页面，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。

## 12 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

## 13 iframe有那些缺点?

- iframe 会阻塞主页面的 Onload 事件
- 搜索引擎的检索程序无法解读这种页面，不利于 SEO
- iframe 和主页面共享连接池， 而浏览器对相同域的连接有限制，所以会影响页面的并行加载
- 使用 iframe 之前需要考虑这两个缺点。如果需要使用 iframe ， 最好是通过 javascript 动态给 iframe 添加 src 属性值， 这样可以绕开以上两个问题

## 14 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

## 15 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页
- 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

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

- 页面被加载的时， link 会同时被加载， 而 @imort 页面被加载的时， link 会同时被加载， 而 @import 引用的 CSS 会等到页面被加载完再加载 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， 页面以宽松的向后兼容的方式显示 。模拟老式浏览器的行为以防止站点无法工作 。 DOCTYPE 不存在或格式不正确会导致文档以混杂模式呈现

## 17 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

## 18 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

## 19 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，不会失真和锯齿。而 canvas 输出标量画布，就像一张图片一样，放大会失真或者锯齿

## 20 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

## 21 如何在页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

## 22 网页验证码是干嘛的， 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

## 23 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

## 24 渲染优化

- 禁止使用 iframe （阻塞父文档 onload 事件）
  - iframe 会阻塞主页面的 Onload 事件
  - 搜索引擎的检索程序无法解读这种页面，不利于SEO
  - iframe 和主页面共享连接池， 而浏览器对相同域的连接有限制，所以会影响页面的并行加载
  - 使用 iframe 之前需要考虑这两个缺点。如果需要使用 iframe ， 最好是通过 javascript
  - 动态给 iframe 添加 src 属性值， 这样可以绕开以上两个问题
- 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
- 页面头部的 <style></style> <script></script> 会阻塞页面；（因为 Renderer 进程中 JS 线程和渲染线程是互斥的）
- 页面中空的 href 和 src 会阻塞页面其他资源的加载 (阻塞下载进程)
- 网页 gzip ， CDN 托管， data 缓存， 图片服务器
- 前端模板 1S+数据，减少由于 HTML 标签导致的带宽浪费， 前端用变量保存A1AX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

## 25 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" /> <!--页面描述-->
<meta name=" keywords" content=" " /> <!-- 页面关键词-->
<meta name=" author" content=" name, email@gmail .com" /> <!--网页作者-->
<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" /> <!--添加到主屏后的标
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
设置页面不缓存-->
<meta http-equiv=" pragma" content=" no-cache" >
<meta http-equiv=" cache-control" content=" no-cache" >
<meta http-equiv=" expires" content=" 0">
```

## 26 你做的页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)

## 27 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。
- 页面加载速度更快 、结构化清晰 、页面显示简洁。
- 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。

## 28 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

## 29 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页面，保证最基本的功能，然后再针对高级浏览器进行效果 、交互等改进和追加功能达到更好的用户体验。
- 优雅降级：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

## 30 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，优化页面响应速度
- 防止不必要的安全问题

## 31 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

## 32 知道的网页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

## 33 在css/js代码上线之后开发人员经常会优化性能，从用户刷新网页开始， 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

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

- 图片懒加载，在页面上的未可视区域可以添加一个滚动事件， 判断图片位置与浏览器顶端的距离与页面的距离， 如果前者小于后者，优先加载。
- 如果为幻灯片 、相册等， 可以使用图片预加载技术，将当前展示图片的前一张和后一张优先下载。
- 如果图片为css图片， 可以使用 CSSsprite ， SVGsprite ， Iconfont 、 Base64 等技术。
- 如果图片过大， 可以使用特殊编码的图片，加载时会先加载一张压缩的特别厉害的缩略图， 以提高用户体验。
- 如果图片展示区域小于图片的真实大小，则因在服务器端根据业务需要先行进行图片压缩， 图片压缩后大小与展示一致。

## 34 常见排序算法的时间复杂度,空间复杂度

## 各种排序的比较

| 排序方法 | 平均情况 | 最好情况 | 最坏情况 | 辅助空间 |
| --- | --- | --- | --- | --- |
| 直接插入 | $O(n^2)$ | $O(n)$ | $O(n^2)$ | $O(1)$ |
| 希尔排序 | $O(n\log_2 n)$ ~$O(n^2)$ | $O(n^{1.3})$ | $O(n^2)$ | $O(1)$ |
| 起泡排序 | $O(n^2)$ | $O(n)$ | $O(n^2)$ | $O(1)$ |
| 快速排序 | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(n^2)$ | $O(\log_2 n)$ ~$O(n)$ |
| 简单选择 | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | $O(1)$ |
| 堆排序 | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(1)$ |
| 归并排序 | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(n\log_2 n)$ | $O(n)$ |

## 35 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

## 36 HTTP request报文结构是怎样的

1. 首行是Request-Line包括：请求方法，请求URI，协议版本， CRLF
2. 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
3. 请求头和消息实体之间有一个CRLF分隔
4. 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
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/
```

```
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
```

## 37 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"
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，极大地提高页面加载速度
  - 增加图片信息重复度，提高压缩比，减少图片大小

    - 更换风格方便， 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

## 2 `display: none;` 与 `visibility: hidden;` 的区别

- 联系：它们都能让元素不可见
- 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， 只是内容不可见
    - `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

## 3 `link` 与 `@import` 的区别

1. `link` 是 `HTML` 方式， `@import` 是CSS方式
2. `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)
3. `link` 可以通过 `rel="alternate stylesheet"` 指定候选样式
4. 浏览器对 `link` 支持早于 `@import` ， 可以使用 `@import` 对老浏览器隐藏样式
5. `@import` 必须在样式规则之前， 可以在css文件中引用其他文件
6. 总体来说： `link` 优于 `@import`

## 4 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， 造成页面闪烁。
- 解决方法：把样式表放到文档的 `<head>`

## 5 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

## 6 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`**

## 7 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

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

## 9 css3有哪些新特性

- 新增各种 css 选择器
- 圆角 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。

## 10 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 设置元素不可见。
- 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

## 11 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;

## 12 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

## 13 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

## 14 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

## 15 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

## 16 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

## 17 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

## 18 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

## 19 在网页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

## 20 ::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

## 21 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

## 22 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

## 23 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

## 24 列出你所知道可以改变页面布局的属性

- position 、 display 、 float 、 width 、 height 、 margin 、 padding 、 top 、 left 、 right 、、

## 25 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

## 26 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

## 27 base64的原理及优缺点

- 优点可以加密，减少了 HTTP 请求
- 缺点是需要消耗 CPU 进行编解码

## 28 几种常见的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>
```

## 双飞翼布局

```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>
```

## 29 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;

## 30 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 ， 比如热门的: autoprefixer
- postcss 可以对sass处理过后的 css 再处理 最常见的就是 autoprefixer

## 31 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

## 32 自定义字体的使用场景

- 宣传/品牌/ banner 等固定文案
- 字体图标

## 33 如何美化CheckBox

- <label> 属性 for 和 id
- 隐藏原生的 <input>
- :checked + <label>

## 34 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

## 35 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

## 36 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  包含，然后靠负 margin 形成 bfc
- 使用 flex

## 37 请用CSS写一个简单的幻灯片效果页面

> 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
    }
    25% {
        background:url(http://b.hiphotos.baidu.com/image/w%3D400/sign=edee1
    }
    50% {
        background:url(http://b.hiphotos.baidu.com/image/w%3D400/sign=937da
    }
    75% {
        background:url(http://g.hiphotos.baidu.com/image/w%3D400/sign=7d375
    }
    100% {
        background:url(http://c.hiphotos.baidu.com/image/w%3D400/sign=cfb23
    }
}
```

## 38 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。

## 39 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！）

## 40 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`

## 41 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;
}
```

## 42 px和em的区别

- px 和 em 都是长度单位， 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 。

## 43 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

## 44 知道css有个content属性吗？有什么作用？有什么应用?

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

```css
/**一种常见利用伪类清除浮动的代码**/
.clearfix:after {
    content:".";        //这里利用到了content属性
    display:block;
    height:0;
    visibility:hidden;
    clear:both;
 }
.clearfix {
    *zoom:1;
}
```

## 45 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`

## 46 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

## 47 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)`

## 48 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
  - 添加或者删除可见的DOM元素，
  - 元素位置、尺寸、内容改变，
  - 浏览器页面初始化，

- 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

## 49 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长 、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 定义动画播放的时长 、 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。

## 50 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;
}
```

## 51 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;
  }
}
```

## 52 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

## 1 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，创建闭包的最常见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， 否则会造成网页的性能问题，在IE中可能导致内存泄露
- 解决方法是，在退出函数之前，将不使用的局部变量全部删除

## 2 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见性和生命周期

## 3 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

## 4 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

## 5 Javascript如何实现继承?

- 构造继承
- 原型继承
- 实例继承
- 拷贝继承
- 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
```

## 6 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

## 7 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

## 8 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

## 9 Ajax原理

- Ajax 的原理简单来说是在用户和服务器之间加了一个中间层( 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 而更新页面 。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。

## 10 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.....: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

## 11 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
      };
})();
```

## 12 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

## 13 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

## 14 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

## 15 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

## 16 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 推荐的风格通过返回一个对象做为模块对象， CommonJS 的风格通过对 module.exports 或 exports 的属性赋值来达到暴露模块对象的目的

## 17 常见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页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， 或限制长度，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 页面里插入恶意 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长度和对 ”<”,”>”,”;”,”’” 等字符做过滤；其次任何内容写到页面之前都必须加以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页面的代码和数据包 。 CSRF 是代替用户完成指定的动作， 需要知道其他用户页面的代码和数据包 。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，就是在客户端页面增加伪随机数
- 通过验证码的方法

## 18 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;

## 19 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 把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过 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。

## 20 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

## 21javascript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);`

## 22 常见兼容性问题?

- 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.

## 23 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

## 24 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

## 25 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。

## 26 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

## 27 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 推荐的风格通过返回一个对象做为模块对象， CommonJS 的风格通过对 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，对其他文件是不可见的。
  - 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

## 28 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）

## 29 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

## 30 介绍js的基本数据类型

- `Undefined`、`Null`、`Boolean`、`Number`、`String`

## 31 介绍js有哪些内置对象

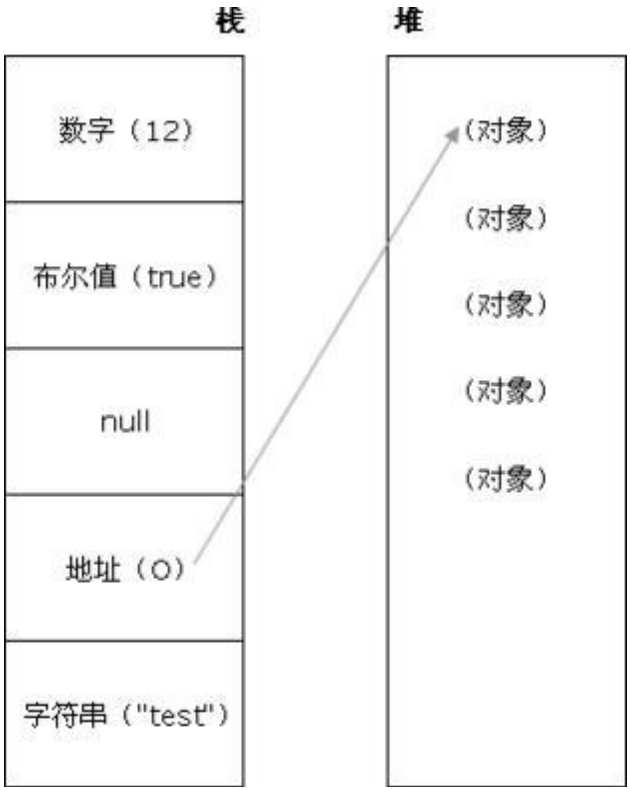
- `Object` 是 `JavaScript` 中所有对象的父对象
- 数据封装类对象：`Object`、`Array`、`Boolean`、`Number` 和 `String`
- 其他对象：`Function`、`Arguments`、`Math`、`Date`、`RegExp`、`Error`

## 32 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

## 33 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




## 34javascript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

```
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("麦兜",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();
```

## 35 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 +')')

## 36 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

## 37 ["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。

## 38javascript 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

## 39 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);
```

## 40js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

## 41 同步和异步的区别

- 同步：浏览器访问服务器请求，用户看得到页面刷新， 重新发请求,等请求完， 页面刷新，新内容出现，用户看到新内容,进行下一步操作
- 异步：浏览器访问服务器请求，用户正常操作， 浏览器后端进行请求 。等请求完， 页面不刷新，新内容也会出现，用户看到新内容

## 42 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页面，保证最基本的功能，然后再针对高级浏览器进行效果 、交互等改进和追加功能达到更好的用户体验。
- 优雅降级： 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

## 43 defer和async

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

## 44 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

## 45 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

## 46 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

## 47 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， 这个东西没啥大影响；一个 Object.creat() 搞定继承， 比 class 简洁清晰的多

## 48 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

## 49 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

## 50 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， 是从用户的角度来看产品的质量 。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。

# 51 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
```

# 52 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

## 53 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`

## 54 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"第一等公民"、只用"表达式"

## 55 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

## 56 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

```
graph LR
A-->B
```

- apply/call调用模式

## 57 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’， 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

## 58 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

## 59 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

## 60 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见且使用很频繁 。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，就要检查一下数组长度 。读取属性 （数组长度） 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页面上的选择器相关元素， 速度会大大降低

## 61 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

## 62 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

## 63 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

## 64 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

## 65 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);
```

## 66 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页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>
```

## 67 希望获取到页面中所有的checkbox怎么做?

> 不使用第三方框架

```
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]);
    }
}
```

## 68 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
```

# 69 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
```

# 70 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];
          }
     }
 }
```

## 71 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() 方法是必须等到页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
            }
        });
    }
};
```

## 72 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

## 73 获取页面所有的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即中获取到了页面中的所有checkbox
```

## 74 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

```
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)]

```
[...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)
```

## 75 （设计题）想实现一个对页面某个节点的拖曳？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

## 76 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

## 77 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  5 s  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

## 78 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));
}
```

## 79 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
```

## 80 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 放在页面底部

## 81 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 强制刷新网页时， 直接从服务器加载，跳过强缓存和协商缓存；
- 当 f5 刷新网页时，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)

## 82 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（取决于数据包长度)， 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门。而客户端也只需要参照 api 实现即可

## 83 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）

## 84 深浅拷贝

浅拷贝

- Object.assign
- 或者展开运算符

深拷贝

- 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

## 85 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

```
/**
 * 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;
  };
};
```

## 86 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

## 87 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

## 88 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

## 89 前端面试之hybrid

> http://blog.poetries.top/2018/10/20/ EncodeStudio-hybrid/

## 90 前端面试之组件化

> http://blog.poetries.top/2018/10/20/ EncodeStudio-component/

## 91 前端面试之MVVM浅析

> http://blog.poetries.top/2018/10/20/ EncodeStudio-mvvm/

## 92 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';
    }
  }
}
```

## 93 请简单实现双向数据绑定 mvvm

html

```
<input id="input"/>
```

js

```
const data = {};
const input = document.getElementById( 'input');
Object.defineProperty(data, 'text', {
  set(value) {
    input.value = value;
    this.value = value;
  }
});
input.onChange = function(e) {
  data.text = e.target.value;
}
```

## 94 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)
}
```

## 95 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

## 96 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

## 97 为什么 canvas 的图片为什么过有跨域问题

## 98 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

## 99 请手写实现一个 promise

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

## 100 说说从输入URL到看到页面发生的全过程，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标签放在页面顶部。
- 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

## 101 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

## 102 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

## 103 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` 表示是否可见 。
- 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。
- 组件高度可能大于页面高度， 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

## 104 `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)
}
```

## 105 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

## 2 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

## 3 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

## 4 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();
```

## 5 jQuery 的属性拷贝(extend)的实现原理是什么， 如何实现深拷贝

- 浅拷贝（只复制一份原始对象的引用） `var newObject = $.extend({}, oldObject);`
- 深拷贝（对原始对象属性所引用的对象进行进行递归拷贝） `var newObject = $.extend(true, {}, oldObject);`

## 6 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
```

## 7 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

## 8 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

## 9 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

## 10jQuery 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);
```

## 11jQuery 一个对象可以同时绑定多个事件， 这是如何实现的

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

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

## 12 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

## 13jQuery 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();
```

## 14jQuery 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);

## 15jQuery 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

## 16jQuery 和 Zepto 的区别？ 各自的使用场景

- jQuery 主要目标是 PC 的网页中，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

## 17jQuery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

## 2 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 创建的网页开头不使用 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

## 3 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。

## 4 Bootstrap 网格系统（Grid System）的工作原理

- （1）行必须放置在 .container class 内，以便获得适当的对齐（alignment）和内边距（padding）。
- （2）使用行来创建列的水平组。
- （3）内容应该放置在列内，且唯有列可以是行的直接子元素。
- （4）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。
- （5）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。
- （6）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

## 5 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-*

## 6 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）

## 7 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`

## 8 用Bootstrap， 如何设置文字的对齐方式?

- `class="text-center"` 设置居中文本
- `class="text-right"` 设置向右对齐文本
- `class="text-left"` 设置向左对齐文本

## 9 Bootstrap如何设置响应式表格?

> 增加 `class="table-responsive"`

## 10 使用Bootstrap创建垂直表单的基本步骤?

- （1）向父 `<form>` 元素添加 `role="form"` ；
- （2）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;
- （3）向所有的文本元素 `<input>` 、 `<textarea>` 、 `<select>` 添加 `class="form-control"`

## 11 使用Bootstrap创建水平表单的基本步骤?

- （1）向父 `<form>` 元素添加 `class="form-horizontal"` ；
- （2）把标签和控件放在一个带有 `class="form-group"` 的 `<div>` 中;
- （3）向标签添加 `class="control-label"` 。

## 12 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。

## 13 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

## 14 Bootstrap有哪些关于的class?

- （1） .img-rounded 为图片添加圆角
- （2） .img-circle 将图片变为圆形
- （3） .img-thumbnail 缩略图功能
- （4） .img-responsive 图片响应式 (将很好地扩展到父元素)

## 15 Bootstrap中有关元素浮动及清除浮动的class?

- （1） class="pull-left" 元素浮动到左边
- （2） class="pull-right" 元素浮动到右边
- （3） class="clearfix" 清除浮动

## 16 除了屏幕阅读器外， 其他设备上隐藏元素的class?

> 、class="sr-only"、、

## 17 Bootstrap如何制作下拉菜单?

- （1）将下拉菜单包裹在 class="dropdown" 的  中;
- （2）在触发下拉菜单的按钮中添加： class="btn dropdown-toggle" id="dropdownMenu1" data-toggle="dropdown"
- （3）在包裹下拉菜单的ul中添加： class="dropdown-menu" role="menu" aria-labelledby="dropdownMenu1"
- （4）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" 。

## 18 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?

- （1）用 class="btn-group" 的  去包裹按钮组； class="btn-group-vertical" 可设置垂直按钮组。
- （2） btn-group 的优先级高于 btn-group-vertical 的优先级。

## 19 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。

## 20 Bootstrap中的输入框组如何制作?

- （1）把前缀或者后缀元素放在一个带有 class="input-group" 中的  中
- （2）在该  内，在 class="input-group-addon" 的  里面放置额外的内容;
- （3）把  放在 <input> 元素的前面或后面。

## 21 Bootstrap中的导航都有哪些?

- （1）导航元素：有 class="nav nav-tabs" 的标签页导航， 还有 class="nav nav-pills" 的胶囊式标签页导航;
- （2）导航栏： class="navbar navbar-default" role="navigation" ;
- （3）面包屑导航： class="breadcrumb"

## 22 Bootstrap中设置分页的class?

- 默认的分页： class="pagination"
- 默认的翻页： class="pager"

## 23 Bootstrap中显示标签的class?

> class="label"

## 24 Bootstrap中如何制作徽章?

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

## 25 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

## 1 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， 如页面注册， 页面标题及 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 。包括页面注册， 网络设置， 以及小程序的 window 背景色， 配置导航条样式， 配置默认标题。
- app.js 必须要有这个文件，没有也是会报错！但是这个文件创建一下就行 什么都不需要写以后我们可以在这个文件中监听并处理小程序的生命周期函数 、声明全局变量。
- app.wxss 配置全局 css

## 2 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

## 3 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

### 4 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

### 5 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

### 1 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

### 2 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

## 3 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'
        }
    ]
},
```

## 4 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;
        }
```

## 2 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;
}
```

## 3 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));
```

## 4 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));
```

## 5 编写一个方法 求一个字符串的字节长度

- 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"));
```

## 6 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);
    };
};
```

## 7 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;
          }
      }
```

## 8 下面这个ul， 如何点击每一列的时候alert其index

> 考察闭包

```html
<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>
```

```js
// 方法一:
var lis=document.getElementById( '2223').getElementsByTagName( 'li');
for(var i=0;i<3;i++)
{
    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);
}
```

## 9 定义一个log方法，让它可以代理console.log的方法

可行的方法一:

```js
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);
};
```

## 10 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);
```

## 11 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();
```

## 12 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?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;
}
```

## 13 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"
```

## 14 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)
}
```

## 15 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

## 1 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

## 2 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。

## 3 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， 或者生命周期较长的对象，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：不用了的东西要及时归还。
- 减少层级过多的引用

## 4 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

## 5 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;
    }
}
```

## 6 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，根据每个页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

## 1 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Im6有效的)

  对于移动平台的优化
  针对于 `SEO` 进行优化

## 2 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

## 3 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；
- 做好的页面结构， 页面重构和用户体验；

## 4 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)

## 5 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)；
- 页面进行标注 （例如 页面 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

## 6 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、一些常见问题

- 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?
- 你们部门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?