致谢

我们来自五湖四海,为着一个相同的目标,相聚在传智.时光飞逝,岁月荏苒,转眼已至毕业季.为了帮助组员高效复习,加深理解,数羊战队各位队员对题目进行了整理.由于时间有限,能力略有欠缺,不足之处还请见谅.错误之处还请指出(反正笔者是不会改的),多做交流,希望大家共通学习共同进步!

特此对参与整理的人员进行感谢,以下排名不分先后:

陈政宏----"跟我喝酒的都能拿 10K+"
陈佩施----"有什么姿势是我现在做不到的？"
陈章会----"来,我帮你换个头"
邱云阳----"美国老大爷"
王月功----"树 tree 新 new 风 bee"
叶蓝梦----"我是一名大四的学生"
杨尚博----"大哥,要纪录片吗"
朱林峰----"停车做爱峰林朱,霜叶红于二月花"
查景蓝----"昏斗罗:婵姐的手好滑啊~"

感谢各位任课老师、婷姐、少爷对我们的敦敦教导!

数羊战队

2016-11-20

目录

一、HTML/CSS

1、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?

一般都会在主流浏览器中进行测试,包括 IE,Chrome,Safari,Firefox 和 Opera.

内核:

IE: trident 内核(1997 年的 IE4 中首次被采用,沿用至今.腾讯,猎豹,360 等浏览器都是用了 IE 的内核)

Firefox:gecko 内核 (开源内核,因为火狐用户最多,被称为火狐内核,是一个跨平台内核,windows,linux 和 mac os 中可以使用)

Chrome:Blink(基于 webkit,Google 与 Opera Software 共同开发) (2013 年之前是用 webkit)

Safari:webkit 内核 (苹果公司的内核,开源代码,不受 IE,火狐限制. 遨游浏览器,塞班手机浏览器,安卓默认浏览器都是用 webkit 作为内核.另外 google chrome ,360 极速浏览器和搜狗高速浏览器高速模式也使用 webkit 作为内核.)

Opera:以前是 presto 内核(渲染速度达到极致,ie 内核速度的 3 倍,但是兼容性很差),Opera 现已改用 Google Chrome 的 Blink 内核

未来:Mozilla 与三星也达成合作协议开发"下一代"浏览器渲染引擎 Servo.

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

<!DOCTYPE> 声明.

DOCTYPE,简称为 DTDs,英文 Document type,中文"文档类型".是一种标准通用标记语言的文档类型声明,它的目的是要告诉标准通用标记语言解析器,它应该使用什么样的文档类型定义(DTD)来解析文档.(重点:告诉浏览器按照何种规范解析页面)

没有定义 doctype 会开启怪异模式.怪异模式下,很多的 css 效果都会受到影响.

现在有用于在 Web 浏览器中的布局引擎的三种模式:

1.quirks mode: 怪异模式

2.almost standards mode:准标准模式 (只有极少数的怪异行为执行.)

3. full standards mode:完整标准模式(一直努力希望致力于描述 HTML 和 CSS 规范.)——In full standards mode , the behavior is (hopefully) the behavior described by the HTML and CSS specifications.)

区分严格模式与混杂模式的意义在于:DOCTYPE 不存在或格式不正确会导致文档以混杂模式呈现.

3、Quirks 模式是什么?它和 Standards 模式有什么区别?

答案 1:从 IE6 开始,引入了 standard 模式.而在 IE6 之前 CSS 还不够成熟,所以 IE5 等之前的浏览器对 CSS 的支持很差,都是基于旧的布局方式写的,为了保证既能提供新的渲染机制又能保证就的页面能够正常显示,此时就传入了一个参数,DTD.就是文档声明.在 IE6 之前,是没有 DTD 的,那么久假定,如果有 DTD,那就是标准模式,如果没有 DTD,那就是以原来的旧模式进行渲染,这种兼容,就是怪异模式.

答案 2:标准模式(standard 模式)是指,浏览器按 W3C 标准解析执行代码;怪异模式(Quirks 模式)则是使用浏览器自己的方式解析执行代码,因为不同浏览器解析执行的方式不一样,所以我们称之为怪异模式.浏览器解析时到底使用标准模式还是怪异模式,与你网页中的 DTD 声明直接相关,DTD 声明定义了标准文档的类型(标准模式解析)文档类型,会使浏览器使用相应的方式加载网页并显示,忽略 DTD 声明,将使网页进入怪异模式(quirks mode).

4、谈谈以前端角度出发做好 SEO 需要考虑什么?

Meta 标签优化

主要包括主题(Title),网站描述(Description),和关键词(Keywords).其他还有 Author(作者),Category(目录),Language(编码语种)等.

放置关键词

关键词分析和选择是 SEO 最重要的工作之一.首先要给网站确定主关键词(一般在 5 个上下),然后针对这些关键词进行优化,包括关键词密度(Density),相关度(Relavancy),突出性(Prominency)等等.

付费的搜索引擎

搜索引擎登录

将网站提交(submit)到搜索引擎.如果你的是商业网站,主要的搜索引擎和目录都会要求你付费来获得收录(比如 Yahoo 要 299 美元),但是好消息是(至少到目前为止)最大的搜索引擎 Google 目前还是免费,而且它主宰着 60%以上的搜索市场.

链接交换和链接广泛度(Link Popularity)

其它网站到你的网站的链接越多,你也就会获得更多的访问量.更重要的是,你的网站的外部链接数越多,会被搜索引擎认为它的重要性越大,从而给你更高的排名.

合理的标签使用.

5、对 WEB 标准以及 W3C 的理解与认识.

web 标准简单来说可以分为结构、表现和行为.其中结构主要是有 HTML 标签组成.或许通俗点说,在页面 body 里面我们写入的标签都是为了页面的结构.表现即指 css 样式表,通过 css 可以是页面的结构标签更具美感.行为是指页面和用户具有一定的交互,同时页面结构或者表现发生变化,主要是有 js 组成.

web 标准三层分离,使其更具有模块化.但一般产生行为时,就会有结构或者表现的变化,也使这三者的界限并不那么清晰.

W3C 对 web 标准提出了规范化的要求,也就是在实际编程中的一些代码规范:包含如下几点:

1.对于结构要求:(标签规范可以提高搜索引擎对页面的抓取效率,对 SEO 很有帮助)

1)标签字母要小写

2)标签要闭合

3)标签不允许随意嵌套

2.对于 css 和 js 来说

1)尽量使用外链 css 样式表和 js 脚本.是结构、表现和行为分为三块,符合规范.同时提高页面渲染速度

2)样式尽量少用行间样式表,使结构与表现分离,标签的 id 和 class 等属性命名要做到见文知义,标签越少,加载越快,用户体验提高,代码维护简单,便于改版

3)不需要变动页面内容,便可提供打印版本而不需要复制内容,提高网站易用性.

6、HTML 与 XHTML——二者有什么区别?

粗略可以分为两大类比较:一个是功能上的差别,另外是书写习惯的差别.关于功能上的差别,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.

因为 XHTML 的语法较为严谨,所以需要特别注意 XHTML 的规则.

1.所有标签都必须小写不能大小写穿插其中,也不能全部都是大写

2.标签必须成双成对

3.标签顺序必须正确

标签由外到内,一层层包覆着,所以假设你先写 div 后写 h1,结尾就要先写 h1 后写 div.只要记住一个原则"先进后出",先弹出的标签要后结尾.

4.所有属性都必须使用双引号

5.不允许使用 target="_blank"

从 XHTML 1.1 开始全面禁止 target 属性,如果想要有开新窗口的功能,就必须改写为 rel="external",并搭配 JavaScript 实现此效果.

7、知道什么是微格式吗?谈谈理解.在前端构建中应该考虑微格式吗?

简化版本记忆:

微格式(microformats):是一种让机器可读的语义化 XHTML 词汇的集合,是结构化数据的开放标准.是为特殊应用而制定的特殊格式.

包含数据的结构化的 XHTML 代码块的定义格式,由于是 XHTML 代码块,所以很适合人类阅读,由于是结构化的,又很容易被机器处理,很容易和外部进行数据通信. 需要作的仅仅是为现有的(X)HTML 元素添加元数据和其他属性.所以,我们无需抛开已有的工作习惯,因为微格式提供的解决方法是符合当前我们行为和习惯模式的.

优点:将智能数据添加到网页上,让网站内容在搜索引擎结果界面可以显示额外的提示.(应用范例:豆瓣); 在爬取 Web 内容时,

能够更为准确地识别内容块的语义.

一句话,根据特定格式给现有标签添加语义化属性,对搜索引擎友好,提供额外显示信息,方便代码阅读

8、div+css 的布局较 table 布局有什么优点?

1.能够使代码精简: css 文件可以在网站的任意一个页面进行调用,避免了使用 table 表格修改部分页面.

2.提升网页访问速度:代码少,其浏览访问速度自然得以提升,从而提升了网站的用户体验度.

3.有利于 SEO 优化:对搜索引擎很是友好,简洁、结构化的代码更加有利于突出重点和适合搜索引擎抓取.另网络爬虫不喜欢页面有态度 css 代码,所以外部调用 CSS 有益于优化

4.浏览器兼容性不好:DIV+CSS 更容易出现多种浏览器不兼容的问题,主要原因是不同的浏览器对 web 标准默认值不同.

回答问题可以从下面 table 优缺点引入

table 优点:开发时间短(使用 DW 开发速度快);纯 table 各浏览器不会有兼容问题;内容可自适应;在搜索引擎排名能靠前.

table 缺点:如果布局变更,需要重新开发;如果 table 里有 div ul 等,可能会出现浏览器兼容问题;加载速度慢;table 套 table,会害死维护人员的.

9、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(成功访问后网站的域名、IP 地址信息缓存到本地

CDN 缓存(内容分发网络,选择一个离用户最近的 CDN 边缘节点来响应用户的请求)

浏览器缓存(存储最近访问的过的页面,再次请求时,从本地磁盘显示文档来加速页面的阅览,节约网络的资源加速浏览)

服务器缓存(将需要频繁访问的网络内容存放在离用户较近、访问速度更快的系统中,来提高访问速度).

10、大量图片加载很慢,有哪些方法优化 这些图片的加载,给用户更好的体验.

1.图片懒加载,在页面上的未可视区域可以添加一个滚动条事件,判断图片位置与浏览器顶端的距离与页面的距离,如果前者小于后者,优先加载. (大型电商网站常见)

2.图片预加载: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.

3.如果图片为 css 图片,可以使用 CSSsprite 精灵图、SVGsprite 矢量图、Iconfont 字体图标、Base64 等技术.

4.缩略图: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.

11、知道的网页制作会用到的图片格式有哪些?

1.PSD 格式,是 Photoshop 专用文件.非压缩的原始文件保存格式.常用于保留原始信息,文件较大,不便于传输,一般用于备份.

2.GIF 图片格式, 256 种的颜色,无损压缩,支持透明度,

3.PNG 图片格式,有 PNG-8、PNG-24 两种,比 gif 更好,但旧的浏览器和程序可能不支持 PNG 文件.

4.JPEG 图片格式,目前最流行的图像格式,可以把文件压缩到最小的格式,下载速度快.存在一定损耗

5.BMP 图片格式,Windows 系统标准图像,缺点--占用磁盘空间过大.在单机上比较流行

6.webp 是由谷歌推出的新一代图片格式,在压缩方面比当前 JPEG 格式更优越,同时提供了有损压缩与无损压缩的图片文件格式,派生自图像编码格式 VP8

优点: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%.

缺点:编解码速度偏慢.但是由于减少了文件体积,缩短了加载的时间,实际上文件的渲染速度反而变快了.

浏览器支持不全.因此,目前可行的解决方法只能是同时提供两套图片.

12、前端页面有哪三层构成,分别是什么?作用是什么?

1.结构层:由 HTML 或 XHTML 之类的标记语言负责创建,仅负责语义的表达.解决了页面"内容是什么"的问题.

2.表示层:由 CSS 负责创建,解决了页面"如何显示内容"的问题.

3.行为层:由脚本负责.解决了页面上"内容应该如何对事件作出反应"的问题.

13、css 的基本语句构成是?

选择器{属性 1:值 1;属性 2:值 2;……}

14、CSS 选择符有哪些?哪些属性可以继承?优先级算法如何计算? CSS3 新增伪类有那些?

选择符:

1.id 选择器(# myid)、2.类选择器(.myclassname)、3.标签选择器(div, h1, p)、4.子选择器(ul < li)、5.后代选择器(li a)、6.通配符选择器(*)、7.属性选择器(a[rel = "external"])、8.相邻选择器(h1 + p)、9.伪类选择器(a: hover, li: nth - child)

可继承: font-size font-family color, UL LI DL DD DT;

不可继承 :border padding margin width height ;

优先级为:

!important > 行内样式 > id > class > tag > 通配符 > 继承 > 默认

权重算法:(不标准,但可以帮助理解)

用 1 表示派生选择器的优先级、用 10 表示类选择器的优先级、用 100 表示 ID 选择器的优先级

eg:div.test1 .span var 优先级 1+10 +10 +1、span#xxx .songs li 优先级 1+100 + 10 + 1、#xxx li 优先级 100 +1

15、简介盒子模型.

CSS 的盒子模型有两种:IE 盒子模型、标准的 W3C 盒子模型模型.

内容(content)、填充(padding)、边框(border)、边界(margin)都是盒模型的基本属性.

区别:

标准的 W3C 盒子模型模型: 设置的 border 值只包括 content 和 padding.

IE 盒子模型:设置的 border 值包括 content,padding 和 border 值.

16、有哪项方式可以对一个 DOM 设置它的 CSS 样式?

外部样式表:link 标签引入一个外部 css 文件

内部样式表:将 css 代码放在 <head> 标签内部

内联样式:将 css 样式直接定义在 HTML 元素内部

17、CSS 中可以通过哪些属性定义,使得一个 DOM 元素不显示在浏览器可视范围内?

基本的: 设置 display :none;(不占位置),或者设置 visibility :hidden(占位置)

使用 display:none 属性后,HTML 元素(对象)的宽度、高度等各种属性值都将"丢失";而使用 visibility:hidden 属性后,HTML 元素(对象)仅仅是在视觉上看不见(完全透明),而它所占据的空间位置仍然存在.

技巧性: 设置宽高为 0,设置透明度为 0,设置 z-index 位置在-1000

18、img 的 alt 与 title 有何异同? strong 与 em 的异同?

alt 属性是在你的图片因为某种原因不能加载时在页面显示的提示信息,它会直接输出在原本加载图片的地方

title 属性是在你鼠标悬停在该图片上时显示一个小提示,鼠标离开就没有了,有点类似 jQuery 的 hover.HTML 的绝大多数标签都支持 title 属性,title 属性就是专门做提示信息的.

默认显示样式上:默认样式是斜体,默认样式是粗体.

语义上:用来局部强调,则是全局强调.从视觉上考虑,的强调是有顺序的,阅读到某处时,才会注意到.的强调则是一种随意无顺序的,看见某文时,立刻就凸显出来的关键词句 【注】:在实际开发中,我们看到一般大网站都只是使用标签就够了,比较少使用标签.此外,使用和这两个标签可以为 SEO 关键词排名加分.

19、简述一下 src 与 href 的区别.

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

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.

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

补充:link 和@import 的区别,两者都是外部引用 CSS 的方式,但是存在一定的区别:

1.link 是 HTML 标签,除了加载 CSS 外,还可以定义 RSS 等其他事务;@import 属于 CSS 范畴,只能加载 CSS.

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

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

4.link 支持使用 Javascript 控制 DOM 去改变样式;而@import 不支持.

20、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(先计算结果后继承,和数值+%一样).

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

21、行内元素和块级元素的具体区别是什么?行内元素的 padding 和 margin 可设置吗?

一、块级元素:block element

块级元素默认占一整行,可嵌套块级元素或行内元素;一般作为容器出现,用来组织结构,特例:<form>只能包含块级元素

DIV 是最常用的块级元素,元素样式的 display:block 都是块级元素

二、行内元素:inline element

也叫内联元素、内嵌元素等;行内元素一般都是基于语义级(semantic)的基本元素,只能容纳文本或其他内联元素

元素样式的 display : inline 的都是行内元素

三、block(块)元素的特点

①、总是在新行上开始;

②、高度,行高以及外边距和内边距都可控制;

③、宽度缺省是它的容器的 100%,除非设定一个宽度.

④、它可以容纳内联元素和其他块元素

四、inline 元素的特点

①、和其他元素都在一行上;

②、高,行高及外边距和内边距不可改变;

③、宽度就是它的文字或图片的宽度,不可改变

④、内联元素只能容纳文本或者其他内联元素

对行内元素,需要注意如下

设置宽度 width 无效.

设置高度 height 无效,可以通过 line-height 来设置.

设置 margin 只有左右 margin 有效,上下无效.

设置 padding 只有左右 padding 有效,上下则无效.注意元素范围是增大了,但是对元素周围的内容是没影响的.

五、常见的块状元素

Address、blockquote、center、dir、div、dl、fieldset、form 、h1 到 h6、hr、input、prompt、 menu、noframes、noscript、ol、p、pre、table、ul

六、常见的内联元素

a、abbr、acronym、b、bdo、big、br、cite、code、dfn、em、 font、i、img、input、 kbd、label、q、s、samp、select、small、span、strike、strong、sub、sup、textarea、u

七、空元素

没有内容的 HTML 内容被称为空元素.空元素是在开始标签中关闭的.
<hr> <input> <img>

22、rgba()和 opacity 的透明效果有什么不同?

rgba()和 opacity 都能实现透明效果.

opacity 作用于元素,以及元素内的所有内容的透明度

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！)

23、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:0; 垂直上缩成 1 行

水平方向:letter-spacing:字体大小 font-size 值取负; 水平上缩成 1 列

letter-spacing 还可以用于消除 inline-block 元素间的换行符空格间隙

24、BFC 是什么?

BFC(块级格式化上下文),一个独立的渲染区域,只有块级元素参与,规定了内部块级元素的布局方式,且该区域与外部无关.

BFC 元素特性表现原则:内部子元素再怎么变动都不会影响外部元素

布局规则:

1.内部 BOX 会在垂直方向上一个接一个放置,所以是从上向下排列

2.元素的 margin 左边和父级的左边紧挨,浮动元素也是这样,所以是从左向右排列

3.BOX 垂直方向上距离由 margin 决定,但同一个 BFC 内的两个相邻元素的 margin 会出现外边距合并(overflow 可以解决外边距合并)

4.BFC 是页面上一个隔离的独立容器,内容子元素不影响外部元素

5.计算 BFC 高度时,浮动元素也参与计算(overflow 清除浮动以后可以撑开盒子高度)

BFC 的生成:

根元素 body、浮动 float 不为 none、定位 position:absolute/fixed、overflow、display:inline-block/flex/table-cell

24、如何垂直居中一个浮动元素?

1.已知浮动元素的宽高

设定父元素为相对定位,浮动元素为绝对定位 top 和 left 为 50%,再设置浮动元素的 margin-left/top 值为浮动元素宽高一半的负值.

2.不知道浮动元素的宽高

设定父元素为相对定位,浮动元素为绝对定位.且 left/right/top/bottom 设置为 0,再给浮动元素设置 margin auto.

另:如何居中 div? 答:默认 div 有宽度,设置 margin:0 auto;

25、列出 display 的值及作用.position 的值, relative 和 absolute 定位原点是?

display:

block---块级元素默认显示方式.单独占一行,能设置宽高 margin 和 padding 值;

inline-block---能像块级元素一样设置宽高,但也能像行内元素一样,和其他 div 平列一行.

inline---行内元素的默认显示方式.

none---缺省值(不存在页面中)

position :

relative ----相对定位,相对于其正常位置进行定位.

absolute ----绝对定位,相对于非 static 定位第一个直接父元素进行定位.

fixed ---- (老 IE 不支持)-----生成绝对定位的元素,相对于浏览器窗口进行定位.

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

26、absolute 的 containing block 计算方式跟正常流有什么不同?(即 absolute 的元素,浏览器是怎么找到最终的位置的)

css 核心:包含块(Containing Block)是视觉格式化模型的一个重要概念,它与框模型类似,也可以理解为一个矩形,而这个矩形

的作用是为它里面包含的元素提供一个参考,元素的尺寸和位置的计算往往是由该元素所在的包含块决定的.

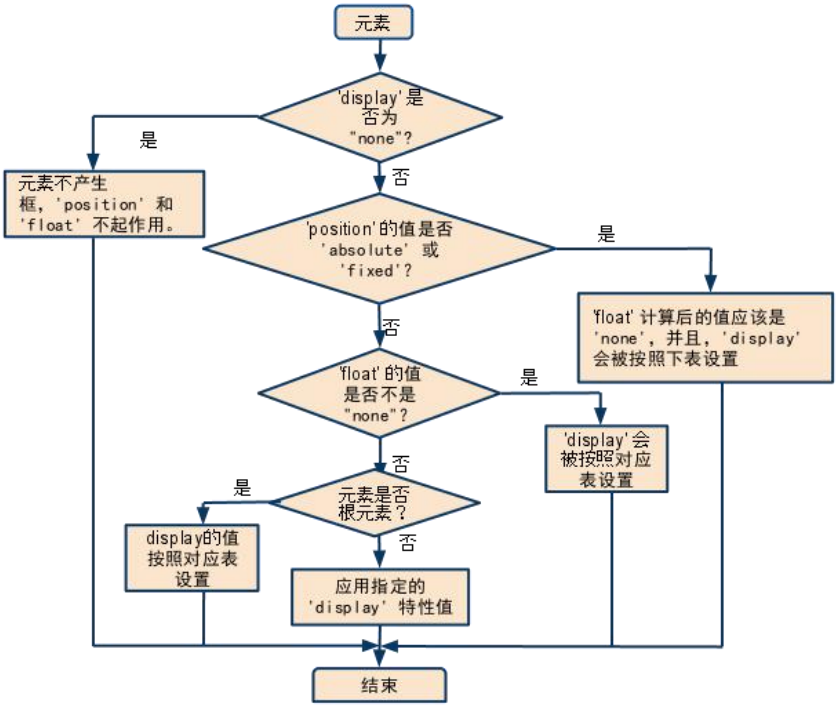
包含块简单说就是定位参考框,或者定位坐标参考系,元素一旦定义了定位显示(相对、绝对、固定)都具有包含块性质,它所包含的定位元素都将以该包含块为坐标系进行定位和调整.

确定一个元素的 containing block,由 position 属性确定:

1. static(默认的)/relative:简单说就是它的父元素的内容框(即去掉 padding 的部分)
2. absolute: 向上找最近的定位为 absolute/relative 的元素
3. fixed: 它的 containing block 一律为根元素(html/body),根元素也是 initial containing block

27、position 跟 display、margin collapse、overflow、float 相互叠加后会怎样?

1、'display'、'position' 和 'float' 的相互关系



转换对应表:

设定值	计算值
inline-table	table
inline, run-in, table-row-group, table-column, table-column-group, table-header-group, table-footer-group, table-row, table-cell, table-caption, inline-block	block
其他	同设定值

说明:一个元素 div

1.'display'的值为'none'那么'position'和'float'不起作用.在这种情况下,元素不产生框.因此浮动和定位无效.

2.'position'的值是'absolute'或'fixed'框就是绝对定位的, 浮动失效,'display' 会被按规则重置.由'top','right','bottom'和'left'属性和该框的包含块确定.

3.'float' 的值不是"none"该框浮动并且'display'会被按照转换对应表设置.

4.元素是根元素如果元素是根元素,'display' 的值按照转换对应表设置.

5.否则,应用指定的 'display' 特性值.

2、position 跟 display、overflow、float 下的 margin collapse(外边距合并)

1.两个或多个毗邻的普通流中的块元素垂直方向上的 margin 会折叠.

2.浮动元素、inline-block 元素、绝对定位元素的 margin 不会和垂直方向上其他元素的 margin 折叠.

3.创建了块级格式化上下文 BFC 的元素,不和它的子元素发生 margin 折叠.

4.元素自身的 margin-bottom 和 margin-top 相邻时也会折叠.

3、外边距重叠就是 margin-collapse.

外边距重叠是指两个垂直相邻的块级元素,当上下两个边距相遇时,起外边距会产生重叠现象,且重叠后的外边距,等于其中较大者.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

防止外边距重叠解决方案:

外层元素 padding 代替、外层元素 overflow:hidden、内层元素透明边框 border:1px solid transparent、内层元素加 float:left;或 display:inline-block、内层元素绝对定位 postion:absolute、内层元素 padding:1px;

28、描述一个"reset"的 CSS 文件并如何使用它.知道 normalize.css 吗?说明不同之处?

引入:不同的浏览器对一些元素有不同的默认样式,不处理时在不同的浏览器下会存在风险,显示差异,甚至布局被打乱,(初始化 CSS 会对搜索引擎优化造成小影响)

可以用 Normalize.css 来代替重置样式文件.但它没有重置所有的样式风格,仅提供了一套合理的默认样式值.既能让众多浏览器达到一致和合理,但又不扰乱其他的东西(如粗体的标题). 在这一方面,无法做每一个复位重置.它也确实有些超过一个重置,它处理了我们永远都不用考虑的怪癖,像 HTML 的 audio 元素不一致或 line-height 不一致.

29、什么是 Css Hack? ie6、7、8 的 hack 分别是什么? 常用 hack 的技巧 ?

CSS hack 由于不同厂商的浏览器,或者同一厂商但是不同版本的浏览器,对 CSS 的解析认识不完全一样,因此会导致生成的页面效果不一样,得不到我们所需要的页面效果. 这个时候我们就需要针对不同的浏览器去写不同的 CSS,让它能够同时兼容不同的浏览器,能在不同的浏览器中也能得到我们想要的页面效果.

简单的说,CSS hack 为了使 CSS 代码兼容不同的浏览器.当然,我们也可以反过来利用 CSS hack 为不同版本的浏览器定制编写不同的 CSS 效果.

书写顺序,一般是将识别能力强的浏览器的 CSS 写在后面.

/*firefox*/ - /*all ie*/ - /*ie8*/ - /*ie7*/ - /*ie6*/ - /*ie9*/ - /*opera*/ - /*chrome and safari*/

技巧:渐进识别的方式,从总体中逐渐排除局部.

首先,巧妙的使用"\9"这一标记,将 IE 游览器从所有情况中分离出来.

接着,再次使用"+"将 IE8 和 IE7、IE6 分离开来,这样 IE8 已经独立识别.

30、Sass、LESS 是什么?大家为什么要使用他们?

CSS 预处理器是一种语言用来为 CSS 增加一些编程的的特性,无需考虑浏览器的兼容性问题,例如你可以在 CSS 中使用变量、简单的程序逻辑、函数等等在编程语言中的一些基本技巧,可以让你的 CSS 更见简洁,适应性更强,代码更直观等诸多好处.SASS、LESS 是最常见的 CSS 预处理器框架.

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).

优点:

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.

31、html 常见兼容性问题?

1.IE 的双边距 BUG

问题:块级元素 float 后设置横向 margin,ie6 显示的 margin 比设置的较大.

解决:给块级元素设置_display:inline;(_ 只有 ie6 能识别)

2.超链接访问过后 hover 样式就不出现的问题是什么?如何解决?

问题:被点击访问过的超链接样式不在具有 hover 和 active 了

解决:改变 CSS 属性的排列顺序: L-V-H-A(link,visited,hover,active)

3.3 个像素问题

问题:在 IE7 中两个 div 是紧挨着的,但是在 IE6 中会出现两个 div 之间出现 3px 左右的间隙,这就是传说中的"IE 3px bug".

解决:这是由于使用 float 引起的使用 dislpay:inline -3px 可以解决.

4.IE z-index 问题

问题:父元素有定位,子元素即便设置 z-index 为多少,都会显示在最上面.

解决:那是因为 z-index 是有继承的.父元素有定位,默认 z-index 为 1.那么子元素也就 z-index 为 1. 要想改变,就得给父元素设置 position:relative,后再设置 z-index.

5.IE5-8 不支持 opacity

解决: 使用滤镜 filter

filter: alpha(opacity=60); /* for IE5-7 */

-ms-filter: "progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"; /* for IE 8*/

6.IE6 不支持 PNG 透明背景

解决: IE6 下使用 gif 图片

7.Png 透明

问题:用透明 PNG 图片作为链接的背景,图像的透明部分变得不能点击.

解决:在兼容 ie 的代码中,给 a 标签设置背景,而 background 的 url()设置为当前页面上.

8.Min-height 最小高度

问题:没办法设置最小高度

解决:用!Important 解决.代码: height:auto !important ; height:最小 px;

9.select 在 ie6 下遮盖 使用 iframe 嵌套

解决:用 iframe 当做 div 的底. div 可以遮盖 iframe,iframe 可以遮盖 select.

10.没法定义 1px 左右的宽度容器

问题:IE6 默认的行高造成的

解决:使用 over:hidden,zoom:0.08, line-height:1px 可以解决.

11.浏览器默认的 margin 和 padding 不同.

解决: 加一个全局的 *{ margin:0; padding:0;} 或者 css 初始化

12.a(有 href 属性)标签嵌套下的 img 标签,在 IE 下会有边框.

解决: a img{border:none;}

13. input 中 button 会有默认的边框,阴影

解决:用 border:none; 解决.在 IE6 以下,还需要加上 border-color:transparet;

32、请用 Css 写一个简单的幻灯片效果页(c3 的 animation 动画)

```
<style>
    .ani{
        width:100px;
        height:100px;
        overflow: hidden;
        box-shadow:0 0 5px rgba(0,0,0,1);
        background-position: center;
        -webkit-animation: loops 10s infinite;
```

```
    }
    @-webkit-keyframes loops {
        0% { background:url(images/1.jpg) no-repeat; }
        33% { background:url(images/2.jpg) no-repeat; }
        66% { background:url(images/3.jpg) no-repeat; }
        /*说明:这里最后一帧如果和最初一帧不同,会出现跳跃*/
        100% { background:url(images/1.jpg) no-repeat; }
    }
</style>
```

33、书写代码,实现 table 表格的隔行变色

方案 1:js 里的隔行变色,js 代码核心为 i%2 时的取值,进行操作 style

方案 2:使用 c3 选择器 E:nth-of-type(2n)、E:nth-of-type(2n+1)

二、HTML5/CSS3

1、CSS3 有哪些新特性?

简化:选择器、RGBA、阴影、盒模型、边框圆角、边框图片、渐变、背景、变换、过渡、动画、列布局、伸缩布局、媒体查询

1、css3 选择器:

属性选择器:E[attr]、E[attr=val]、E[attr*=val]、E[attr^=val]、E[attr$=val];

伪类选择器:E:only-of-type、E:only-child、E:first-child、E:last-child、E:nth-child(n)、E:nth-last-child(n)、E:nth-of-type(n)、E:nth-last-of-type(n){n 范围[0,+∞),即所有子元素,0 和负数在伪类选择器中获取不到;-n+5 表示[1,5]}、E:empty、E:target、E:enabled、E:disabled 控制表单控件的禁用状态、E:checked 单选框或复选框被选中

伪元素选择器:E::befor、E::after、E::first-letter、E::first-line、E::selection(c3 引入)

2、颜色:

[面试]RGBA:该透明度 Alpha 不会被继承,区别于不透明度 opacity

HSLA:Hue(色调),0(或 360)表示红色,120 表示绿色,240 表示蓝色,取值 0 - 360;Saturation(饱和度),取值 0.0% - 100.0%;Lightness(亮度),取值 0.0% - 100.0%;Alpha 透明度,取值 0~1.

3、文本阴影:

text-shadow:none | <shadow> [, <shadow>]* 可以多阴影,逗号隔开

<shadow> = <length>{2,3} && <color>? {Xoffset 可负、Yoffset 可负、blur 不可负}、颜色

4、盒模型:

box-sizing:content-box(标准模式盒模型) | border-box(怪异模式盒模型)

5、盒子阴影:

box-shadow:none | <shadow> [, <shadow>]* 可以有多个阴影

<shadow> = inset? && <length>{2,4} && <color>? 内阴影、{Xoffset 可负、Yoffset 可负、blur 不可负、外延可负}、颜色

6、边框圆角:

border-radius:[<length> | <percentage>]{1,4} [/ [<length> | <percentage>]{1,4}]?

按上左(top-left)、上右(top-right)、下右(bottom-right)、下左(bottom-left)的顺序作用于四个角

7、边框图片:

border-image:<' border-image-source '> || <' border-image-slice '> [/ <' border-image-width '> | / <' border-image-width '>? / <' border-image-outset '>]? || <' border-image-repeat '>

source:图像路径 url

[面试]slice:内偏移(分割方式) [<number> | <percentage>]{1,4} && fill?指定从上,右,下,左方位来分隔图像,将图像分成 4 个角,4 条边和中间区域共 9 份,中间区域始终是透明的(即没图像填充),加上关键字 fill 后按 border-image-repeat 设定填充

width:边框厚度,将裁切的图片缩放至设定厚度,然后在边框中显示,超出 border-width 部分不显示

outset:扩张,设置后图像在原本基础上向外延展设定值后再显示,不允许负值(少用)

repeat:平铺,默认 stretch 拉伸;repeat 平铺但不缩放;round 平铺且自适应缩放大小;space 平铺且自适应缩放间距

8、渐变:

线性渐变 linear-gradient:([<point> || <angle>,]? <stop>, <stop> [, <stop>]*)

径向渐变 radial-gradient:([[<shape> || <size>] [at <position>]? , | at <position>,]?<color-stop>[, <color-stop>]+)

注意:渐变色不是单一颜色,不能使用 backgroud-color 设置,只能使用 background 设置

间隔分明实现,red 0%,red 33.3%,green 33.3%,green 66.6%,blue 66.6%,blue 100%

[面试]径向渐变中的 size:渐变终止的地方(要能看到明显的起点终点来判定是哪种,默认最远角)

closest-side:最近边、farthest-side:最远边、closest-corner:最近角、farthest-corner:最远角

9、背景:

background-size:auto/number/percentage/cover/contain

cover 自动等比缩放,直到某方向完全显示;contain 自动等比缩放,直到图片完全显示

[面试]移动端实现较小图片拥有较大响应区:下面连个属性都设置成 context-box,配合 padding 值增大响应区

background-origin:padding-box/border-box/content-box; 设置定位原点

background-clip: padding-box/border-box/content-box; 设置显示区域

10、过渡:

transition: property duration timing-function delay;

过渡属性(必要,全属性为 all)、过渡时间(必要)、过渡曲线(常用 linear 匀速)、延时事件

11、变换:

坐标轴:x 向右为正,y 向下文正,z 向屏幕外为正

3D 变换比 2D 变换相比:多个 Z 轴,合写语法需要写上默认值,此处不做详细说明

translate 平移:正值向坐标轴正向平移,负值向坐标轴反向平移;默认参考元素左上角

scale 缩放:比值,1 为不缩放,>1 为放大,<1 为缩小;默认参考元素中心点

rotate 旋转:正向面对某坐标轴,正值为顺时针旋转,负值为逆时针旋转;默认参考元素中心点

skew 斜切/翻转/扭曲(2D 独有):正值向坐标轴正向拉伸,默认元素中心点固定,拉伸右下角点,面积保持不变进行扭曲

transform-origin 改变参考点

12、动画

定义:@keyframes 动画序列{ 关键帧{ 属性:目标值 }.... }

关键帧可以用 from to 关键字,也可以用百分比

引用:animation: 动画序列名 持续时间 过渡类型 延迟时间 循环次数 是否反向 动画之外状态

过渡类型 animation-timing-function: linear | ease-in-out | steps(n); 实现步进,此时动画不连续

循环次数 animation-iteration-count:n | infinte; 可设定具体次数或无限循环

[面试]动画外状态 animation-fill-mode: forwards | backwords | both;

说明:forwards 在动画结束后保持最后状态(100%状态),backwords 会在延迟开始之前先执行最初状态(0%状态,无延迟时,没有明显效果),both 会存在上述两种情况

是否反向 animation-direction: alternate 交替 | reverse 反向 | normal 正常

常用简写:animation: move 5s [2s] linear infinite [alternate]

13、列布局:

column-count:列数、column-width:列宽、column-gap:列间隙、column-span:跨列列数、column-rule:列间样式

[面试]列高度的平衡:如果设定列的最大高度,这个时候,文本内容会从第一列开始填充,然后第二列,第三列,造成多列,列数会发生改变.(所以列高可以大于分列后的默认高度,但不要小于)

14、伸缩布局:

父盒子属性设定:

display:flex;　　设置给父容器盒子,子元素都会自动的变成伸缩项(flex item)

justify-content　　设置主轴方向的子元素对齐方式,常设 center 实现居中(默认水平居中)

align-items　　设置侧轴方向的子元素对齐方式,常设 center 实现居中(默认垂直居中)

flex-direction　　定义弹性盒子元素的排列方向(主轴);row 水平主轴、column 垂直主轴

flex-wrap　　控制 flex 容器是单行或者多行;nowrap 单行,wrap 多行

子元素属性设定:

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

flex-grow:默认值是 0,保持 CSS 设定尺寸,设置后平分剩余空间,加在 css 设定尺寸上

flex-shrink　:默认值是 1,一起缩小,设为 0 则保持 CSS 设定,设置后平分溢出部分,减在 css 设定尺寸上

15、媒体查询

查询到当前屏幕的宽度,针对不同的屏幕宽度设置不同的样式来适应不同屏幕.重置浏览器大小时,页面也会根据浏览器的宽度和高度重新渲染页面.

@media screen [not|only]? and (media feature) { CSS-Code; }

media feature :max(min)-height (width),为了实现向上兼容,常用 min-width,从小写到大

媒体查询调用不同 css 文件:<link rel="stylesheet" media=" screen and|not|only (media feature)" href="mystylesheet.css">

2、html5 有哪些新特性、移除了那些元素?如何处理 HTML5 新标签的浏览器兼容问题?如何区分 HTML 和 HTML5?

1、新特性:

语义特性,本地存储特性,设备兼容特性,连接特性,网页多媒体特性,三维、图形及特效特性,性能与集成特性

a.语义化更好的内容标签(header,nav,footer,aside,article,section)

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

c.音频、视频 API(audio,video)　　自定义播放器案例

d.地理定位(Geolocation) API　　掌握获取、使用第三方 API 方法

e.拖拽释放(Drag and drop) API　　给元素设置 draggable = "true" 属性使其可拖动,链接和图片默认可拖动

f. WEB 存储 API　　只能存储字符串

localStorage 长期存储,浏览器关闭后数据不丢失;sessionStorage 临时存储,不共享,关闭页面数据丢失

g.画布(Canvas) API

h.新的技术 webworker, websocket, Geolocation(作为了解)

webworker:可以在浏览器后台运行 JavaScript, 而不占用浏览器自身线程.Web Worker 可以提高应用的总体性能,并且提升用户体验.受限于不能访问 DOM 节点、全局变量和函数、window 和 document,但可以使用定时器和 XMLHttpRequest 实现 Ajax 通信.可分为两种类型:专用线程 dedicated web worker(单页面),以及共享线程 shared web worker(多页面).

websocket:html5 中的新通讯协议,实现了浏览器与服务器全双工通信(full-duplex).一开始的握手需要借助 HTTP 请求完成.前期的网站实时通讯是通过轮询技术实现,定期发送请求,比较占带宽.IE10+实现了该协议.

Geolocation:地理定位,html5 提供了 api

2、移除的元素:

纯表现的元素:basefont,big,center,font,dir,strike,

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

3、HTML5 新标签兼容性处理:

a.IE8/IE7/IE6 支持通过 document.createElement 方法产生的标签,但会把新标签解析成行内元素,还需要添加标签默认的样式 display:block

b.引入兼容性处理文件 html5shiv.min.js(bootstrap 框架中有使用)

```
<!--[if lt IE 9]>
<script> src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>
```

```
<![endif]-->
```

4、区分 html 和 html5:

查看 DOCTYPE 声明:<!DOCTYPE html>为 html5 标准

3、HTML5 和 CSS3 的新标签

HTML5:canvas , audio , video , datalist , keygen ,meter, header, footer, nav, section, article, aside,

CSS3:RGBA, text-shadow, box-shadow, border-radius, border-image,border-color, transform, transition, animation, linear-gradient, radial-gradient, flex

4、如何在 HTML5 页面中嵌入音频、视频?

HTML 5 包含嵌入音频文件的标准方式,支持的格式包括 MP3、Wav 和 Ogg:

方法 1:<audio src="../mp3/See.mp3" controls autoplay></audio>

方法 2:<audio controls>

<source src="jamshed.mp3" type="audio/mpeg"> //资源外部引入

Your browser does'nt support audio embedding feature.

</audio>

HTML5 定义了嵌入视频的标准方法,支持的格式包括:MP4、WebM 和 Ogg:

用法同上,可以设置 width、height 属性

标签属性说明:

autoplay:自动播放; controls:显示控件; src:播放文件 URL; loop:循环播放; preload:预加载

5、HTML5 引入什么新的表单标签、属性、输入类型、事件?

标签:datalist 配合 input 的 list 属性指定、keygen 验证用户数据、output 展示内容不可修改

属性:placeholder 占位符、autofocus 获取焦点、multiple 文件上传多选、required 必填项、pattern 正则表达式、 autocomplete 自动完成、form 指定表单项属于哪个表单、novalidate 关闭验证

输入类型:email、tel、url、number、search、range、color、time、datatime

事件:oninput 输入内容时触发,可用于移动端输入字数统计、onkeyup 按键弹起时触发、oninvalid 验证不通过时触发

6、 HTML5 Canvas 元素有什么用?

Canvas 元素用于在网页上绘制图形,属于 html5,ie9-不支持,自身只是一个标签,没有任何绘制方法,绘图是通过 js 实现,该对象提供了各种绘图 api.常用 2d 库 konvajs.

7、语义化的理解?

1、利于 SEO,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, 页面是否对爬虫容易理解非常重要,因为爬虫很大程度上会忽略用于表现的标记,而只注重语义标记

2、让页面的内容结构化,便于各种终端根据其自身的条件来合适地显示页面,虽然部分 HTML 标签存在默认样式,甚至浏览器都各有默认样式,这种表现虽然不是语义化 HTML 结构的有点,但同样是为了更好的表达 HTML 语义

3、便于维护,方便阅读源码, 去掉或样式丢失的时候能让页面呈现清晰的结构

4、便于团队开发和维护,W3C 给我们定了一个很好的标准,在团队中大家都遵循这个标准,可以减少很多差异化的东西,方便开发和维护,提高开发效率,甚至实现模块化开发

总之一句话,用正确的标签做正确的事情!

8、写/描述一段语义的 html 代码

```
<!--以博客文章为例-->
<article><!--article 是一个特殊的 section 标签,它比 section 具有更明确的语义, 它代表一个独立的、完整的相关内容块-->
    <header><!--文档头部-->
```

```
        <h1>The Very First Rule of Life</h1><!--标题-->
        <p><time pubdate datetime="2009-10-09T14:28-08:00"></time></p>
    </header>
    <p>...</p>
    <p>...</p>
    <!--以下评论区,article 是独立的,但是嵌套 article 的内外层是相关的,所以这里使用嵌套-->
    <section><!--与 div 相似,但它有更进一步的语义,section 用作一段有专题性的内容,一般在它里面会带有标题
                section 典型的应用场景应该是文章的章节、标签对话框中的标签页、或者论文中有编号的部分-->
        <h2>Comments</h2>
        <article>
            <footer>
                <p>Posted by: George Washington</p>
                <p><time pubdate datetime="2009-10-10T19:10-08:00"></time></p>
            </footer>
            <p>Yeah! Especially when talking about your lobbyist friends!</p>
        </article>
        ....
    </section>
</article>
<aside></aside><!--侧边栏,nav 和 asid 也是特殊的 section-->
```

9、Web Storage 本地存储(Local Storage、SessionStorage)和 cookies(储存在用户本地终端上的数据)之间的区别是什么?

Cookies:服务器和客户端都可以访问;大小只有 4KB 左右;设定有效期(不设定则关闭浏览器删除),过期后将会删除;cookie 还需要指定作用域,不可以跨域调用;需要前端开发者自己封装 setCookie,getCookie;作用是与服务器进行交互,作为 HTTP 规范的一部分而存在,不可或缺.

本地存储:只有本地浏览器端可访问数据,服务器不能访问本地存储,可通过 POST 或者 GET 的通讯发送到服务器;每个域 5MB;localStorage 用于持久化的本地存储,除非主动删除数据,否则数据是永远不会过期的;sessionStorage 用于本地存储一个会话中的数据,这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁,是会话级别的存储.Web Storage 拥有 setItem,getItem,removeItem,clear 等方法,是为了在本地"存储"数据而生.

补充:如何实现浏览器内多个标签页之间的通信?

答案:调用 localstorge、cookies 等本地存储方式

10、 HTML5 存储类型有什么区别?~~问题是否有问题?~~

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

11 、如何对网站的文件和资源进行优化?

文件合并(目的是减少 http 请求):Web 性能优化最佳实践中最重要的一条是减少 HTTP 请求,它也是 YSlow 中比重最大的一条规则(YSlow 是雅虎发布的基于 firefox 的网站性能平分工具,共 23 条规则).减少 HTTP 请求的方案主要有合并 JavaScript 和 CSS 文件、CSS Sprites、图像映射 (Image Map,一幅图中多个区域指定不同 url)、使用 Data URI 来编码图片(包含页面但无需额外 http 请求的 url,IE 不支持)、字体图标(将所有图标打包成字体库,减少请求;具有矢量性,可保证清晰度;使用灵活,便于维护,@font-face 声明字体)

文件最小化/文件压缩:目的是直接减少文件下载的体积

使用 CDN(内容分发网络) 托管:其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节,使内

容传输的更快、更稳定.通过在网络各处放置节点服务器所构成的在现有的互联网基础之上的一层智能虚拟网络,CDN 系统能够实时地根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上

缓存的使用:多个域名提供缓存

12、为什么利用多个域名来存储网站资源会更有效?

主流的原因:

1. CDN 缓存更方便(内容分发网络,选择一个离用户最近的 CDN 边缘节点来响应用户的请求)

2.突破浏览器并发限制(像地图之类的需要大量并发下载图片的站点,这个非常重要)

3. Cookieless, 节省带宽,尤其是上行带宽 一般比下行要慢

4.节约主域名的连接数,从而提高客户端网络带宽的利用率,优化页面响应.因为老的浏览器(IE6 是典型),针对同一个域名只允许同时保持两个 HTTP 连接.将图片等资源请求分配到其他域名上,避免了大图片之类的并不一定重要的内容阻塞住主域名上其他后续资源的连接(比如 ajax 请求).

非常规原因:

5.对于 UGC 的内容和主站隔离,防止不必要的安全问题(上传 js 窃取主站 cookie 之类的),因此要求用户内容的域名必须不是自己主站的子域名,而是一个完全独立的第三方域名.

13、你怎么来实现页面设计图,你认为前端应该如何高质量完成工作? 一个满屏品字布局如何设计?

首先按照头部、body、底部进行大致划分,然后再功能进行细分,在明确整体结构的情况下考虑局部实现,先实现大致结构,再进行局部微调实现设计图纸复现.

沟通很重要,要积极与设计师,产品经理进行沟通,确认用户实际需求,避免无谓的返工;技术上,做好的页面结构,注意页面重构和用户体验,通过 CSS hack 处理浏览器兼容、编写优美的代码格式,针对服务器的优化、拥抱 HTML5.

```
<style>
    *{
        margin: 0;
        border: 0;
    }
    .d1, .d2, .d3{
        height: 300px;              /*高度自适应需要配合 js*/
    }
    .d1{
        width: 100%;
        background-color: #FF0000;
    }
    .d3{
        float: left;
        width: 50%;
        background-color: #0099FF;
    }
    .d2{
        float: left;
        width: 50%;
        background-color: #4eff00;
    }
```

```
</style>
<div class="d1">上</div>
<div class="d2">右</div>
<div class="d3">左</div>
```

14、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 progressive enhancement: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 graceful degradation: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.

扩展:产品经理看到 IE6,7,8 网页效果相对高版本现代浏览器少了很多圆角,阴影(CSS3),要求兼容(使用图片背景,放弃 CSS3),你会如何说服他?

年初微软已经放弃对 IE8 的更新和支持,虽然现在还有部分用户在使用 IE8,但是随着市场占有率越来越低,虽然还在考虑对其兼容,但是没必要因为对齐兼容而影响更多客户的体验.

15、请谈一下你对网页标准和标准制定机构重要性的理解.

网页标准和标准制定机构都是为了能让 web 发展的更‘健康’,开发者遵循统一的标准,降低开发难度,开发成本,SEO 也会更好做,也不会因为滥用代码导致各种 BUG、安全问题,最终提高网站易用性.

16、什么是响应式设计?

网页制作的过程中让不同的设备有不同的尺寸和不同的功能.包括弹性网格和布局、图片、CSS media query 媒体查询的使用等.响应式网页设计就是一个网站能够兼容多个终端,而不必多次开发.

17、用 H5+CSS3 解决下导航栏最后一项掉下来的问题

```
box-sizing:border-box;      //不用再计算总宽高
```

18、请用 CSS 实现:一个矩形内容,有投影,有圆角, hover 下状态慢慢变透明.

```
div {
    width: 100px;
    height: 100px;
    border: 1px solid #ccc;
    border-radius: 20px;
    box-shadow: 5px 5px 10px #e92322;
    background-color: rgba(222,222,222,.8);
    opacity: 1;
    transition: all linear 5.5s;
}
div:hover {
    opacity: 0;   /*修改背景色 rgba 不会影响阴影*/
}
```

19、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,用来插入生成内容.最常见的应用是利用伪类清除浮动.

```
.clearfix:after {
```

```
    content: "";      /*这里利用到了 content 属性*/
    height: 0;
    line-height: 0;
    display: block;
    visibility: hidden;
    clear: both;
}
.clearfix {
    *zoom: 1;          /*兼容 ie6,7,现在少用*/
}
```

css 计数器(序列数字字符自动递增)吗?如何通过 css content 属性实现 css 计数器?

```
<style>
    .counter {
        counter-reset: wangxiaoer 0;      /*普照源:用于重置计数器,并设置计数起始数字,正负整数值,默认 0*/
    }
    .counter span:before {
        counter-increment: wangxiaoer 1;/*计数递增,在普照源 reset 基础上实现累加,是先加,所以第一次是 0+1=1,也可以负值*/
                           /*普照规则:普照源(counter-reset)唯一,每普照(counter-increment)1 次,普照源增加 1 次计数值*/
        content: counter(wangxiaoer);     /*只要有 increment,对应计数器值就会变化,counter()只是输出而已,参数 2 可设置样式*/
    }
</style>
<div class="counter">
    <span></span>          <!--生成 1-->
    <span></span>          <!--生成 2-->
    <span></span>          <!--生成 3-->
    <span></span>          <!--生成 4-->
    <span></span>          <!--生成 5-->
</div>
```

20、解释一下这个 css 选择器什么发生什么?

[role=nav]>ul a:not([href^-mailto]){}

定义了 role 属性,并且值为 nav 的任何元素,其子元素列表下的除邮箱链接之外的所有链接元素

三、JS 基础

1、javascript 的 typeof 返回哪些数据类型

number、boolean、string 、underfind、object、function

```
console.log(typeof function);          //会报错,这是个关键字
console.log(typeof string);            //'undefined'
console.log(typeof undefined);         //'undefined'
console.log(typeof null);              //object,比较特殊,要记住
console.log(typeof isNaN);             //function,是个判断函数
console.log(typeof isNaN(123));        //boolean,结果是布尔型
console.log(typeof []);                //object,数组是对象
console.log(Array.isArray());          //false,返回是布尔值
```

```
console.log(Array.isArray);                //function isArray() { [native code] }
console.log(toString.call([]));            //'[object Array]'
console.log([].constructor);               //function Array() { [native code] }
```

2、例举 3 种强制类型转换和 2 种隐式类型转换??

强制:parseInt,parseFloat,Number()

隐式:+string 可以转成数字, a+""可以转成字符串,!var 可以转成布尔值

3、传统事件绑定和符合 W3C 标准的事件绑定有什么区别?

DOM0 传统事件绑定,如 onclick

会出现同一元素同名事件的覆盖,不支持 DOM 事件流事件捕获阶段

DOM1 没有事件绑定规范

IE 绑定:attachEvent("onclick",function(){}),不支持捕获

DOM2 事件绑定:addEventListener("click",function(){},true);参数 3 为 true 支持捕获,参数 3 为 false 支持冒泡

不会出现同名事件覆盖,支持完整事件流

事件流:捕获->目标->冒泡

事件绑定是指把事件注册到具体的元素之上,普通事件指的是可以用来注册的事件

4、事件冒泡和事件委托

事件冒泡:当一个元素上的事件被触发的时候,比如说鼠标点击了一个按钮,同样的事件将会在那个元素的所有祖先元素中被触发.这一过程被称为事件冒泡;这个事件从原始元素开始一直冒泡到 DOM 树的最上层.

阻止事件冒泡和捕获: 标准浏览器 e. stopPropagation(); IE9 之前 event.canceBubble=true;

阻止默认事件:为了不让 a 点击之后跳转,我们就要给他的点击事件进行阻止

return false; e.preventDefault();

事件委托:让利用事件冒泡的原理,让自己的所触发的事件,让他的父元素代替执行!

5、IE 和标准下有哪些兼容性的写法

var event = event || window.event

document.documentElement.clientWidth || document.body.clientWidth

var target = event.srcElement||event.target

6、call 和 apply 的区别

call 和 apply 都是为了改变某个函数运行时的 context 即上下文而存在的,换句话说,就是为了改变函数体内部 this 的指向.因为 JavaScript 的函数存在「定义时上下文」和「运行时上下文」以及「上下文是可以改变的」这样的概念.

二者的作用完全一样,只是接受参数的方式不太一样,参数 1 是 this ,all 需要把参数按顺序传递进去,而 apply 则是把参数放在数组里,若第一个参数为 null 那么函数中的 this 指向 window

Bind 也可以改变当前 this 指向;返回的是个函数.

7、b 继承 a 的方法

继承:原型继承,借用构造函数继承,组合继承,拷贝继承.

一般最简单面试会用的继承方式:原型继承:b.prototype=new a();

8、添加删除替换插入到某个节点的方法

```
obj.appendChild()          //追加子节点
obj.insertBefore()         //插入子节点,参数 2 为 null 时等同于 appendChild
obj.replaceChild()         //替换子节点,参数:新节点、旧节点(应该是没用过)
```

obj.removeChild()　　//删除子节点

此外,jQuery 还提供了 append、perpend、after、before,appendTo、prependTo、insertAfter、insertBefore

9、javascript 的本地对象、内置对象和宿主对象

本地对象为 array、obj、regexp 等可以 new 实例化

内置对象为 Global、Math 等不可以实例化的(也是本地对象,Global 不存在,但是类似于 isNaN()、parseInt()和 parseFloat()方法都是他的方法)

宿主对象为浏览器提供的对象,所有的 BOM 和 DOM,如 document,window

10、JavaScript 是一门什么样的语言,它有哪些特点?

弱类型:声明变量都使用 var(ES6 中有 let 和 const)

动态类型:声明时类型不定,运行时才明确变量类型;对象可以随意添加属性、方法

解析型:遇到一行解析一行(区别是编译型全解析后再执行)

脚本语言:不需要编译,直接执行

11、如何判断某变量是否为数组数据类型

方法一.能力检测,判断是否拥有数组的方法,可能兼容问题

方法二.obj instanceof Array 在某些 IE 版本中不正确,判断是否为 Array 的实例

方法三.在 ES5 中定义了新方法 Array.isArray(), 保证其兼容性,最好的方法如下:

```
isArray : function (obj) {
    return    Array.isArray ? Array.isArray(obj) : Object.prototype.toString.call(arg)==="[object Array]"
},
```

12、看下列代码输出为何?解释原因(变量提升和简单数据类型)

```
var a;
console.log(a);              //undefined,声明变量未赋值就是这个
console.log(typeof a);       //"undefined",类型也是
console.log(b);              //  报错,b is not defined,没有 var 不会变量提升
b=10;
console.log(typeof b);       //"number"
```

13、undefined 会在以下三种情况下产生

1、一个变量定义了却没有被赋值

2、想要获取一个对象上不存在的属性或者方法:

3、一个数组中没有被赋值的元素

注意区分 undefined 跟 not defnied(语法错误)是不一样的

14、看下列代码, 输出什么?解释原因(==判断和隐式转换)

```
//对于 0、空字符串的判断,建议使用 "===" ."==="会先判断两边的值类型,类型不匹配时为 false
var undefined;                          //此时 undefined 变量的值就是 undefined
console.log(undefined   == null);       // true,undefined 与 null 相等,但不恒等(===)
console.log(1 == true);                 // true,此时会把布尔类型的值转换为数字类型 true=1 false=0
console.log(2 == true);                 // false,尝试将 boolean 转换为 number,0 或 1
console.log(0 == false);                // true,尝试将 boolean 转换为 number,0 或 1
console.log(0 == '');                   // true,尝试将 Object 转换成 number 或 string,取决于另外一个对比量的类型
```

```
console.log([] == false);                    // true
console.log([] == ![]);                      // true
console.log(NaN == NaN);                     // false,特例,唯一一个不等于自己的


var foo = "11"+2-"1";                        //先拼接字符串,然后隐式转换成数字运算
console.log(foo);                            //111
console.log(typeof foo);                     //number
```

15、看代码给答案.

```
var a = new Object();
a.value = 1;
b = a;                                       //引用数据类型,a 和 b 是指向同一个堆中数据
b.value = 2;
console.log(a.value);                        //2
```

16、写一个 function 将字符串 foo="get-element-by-id"转化成驼峰表示法"getElementById"

```
function combo(msg) {
    var arr = msg.split("-");                //字符串用-来分割成数组
    //第一个元素不作处理,从第二个开始遍历
    for (var i = 1; i <arr.length; i++) {
        //遍历转换元素,首字母大写
        arr[i] = arr[i].charAt(0).toUpperCase() + arr[i].substr(1, arr[i].length - 1);
    }
    msg = arr.join("");                      //拼接字符串并返回
    return msg;
}
var str = 'get-element-by-id';
console.log(combo(str));                     //getElementById
```

17、实现对该数组的倒序排列、降序排列(冒泡)

```
var arr=[2,1,3];
console.log(arr.reverse())           //翻转数组
for(var i=0;i<arr.length;i++){       //数组冒泡
    for(var j=0;j<arr.length-i-1;j++){
        if(arr[j]<arr[j+1]){
            [arr[j],arr[j+1]]=[arr[j+1],arr[j]]
        }
    }
}
console.log(arr);
```

18、日期对象的常用方法

```
var date = new Date();
var year = date.getFullYear();         //获取年
var month = date.getMonth();           //获取月份获取的月份从 0 开始
```

```
var day = date.getDate();                  //获取日,每个月当中的第几天
var week    = date.getDay();                 //获取星期几    0 代表星期日    6 代表星期六
var hour = date.getHours();                //获取小时  0 - 23
var minutes = date.getMinutes();       //获取分    0 - 59
var second = date.getSeconds();         //获取秒  0 - 59
var ms = date.getMilliseconds();       //获取毫秒  0-999
var tatolms = date.getTime();             //获取时间的总的毫秒数
console.log(tatolms.toString().length); //13 位
//案例
function DateDemo() {
    var d, s = "今天日期是:";
    d = new Date();
    s += d.getMonth() + "/";
    s += d.getDate() + "/";
    s += d.getFullYear();
    return s;
}
console.log(DateDemo());       //今天日期是:10/18/2016
```

19、为了保证页面输出安全,我们经常需要对一些特殊的字符进行转义,请写一个函数 escapeHtml ,将<、 >、 &、 "、进行转义(正则、字符串替换 replace)

```
function escapeHtml(str) {
//[<>"&]:中括号中字符只要其中的一个出现就代表满足条件
//replace 第二个参数传递一个回调函数,回调函数中参数就是匹配结果,如果匹配不到就是  null
    return str.replace(/[<>"&]/g, function (match) {
        switch (match) {
            case '<':
                return '&lt;';          //return 会结束,省去 break
            case '>':
                return '&gt;';
            case '&':
                return '&amp;';
            case '"':
                return '&quot;';
        }
    });
}
str = '"';
console.log(escapeHtml(str));
```

20、foo = foo||bar ,这行代码是什么意思?为什么要这样写?

短路表达式,等同于 if(!foo) foo = bar;如果 foo 存在,值不变,否则把 bar 的值赋给 foo.

作为"&&"和"||"操作符的操作数表达式,这些表达式在进行求值时,只要最终的结果已经可以确定是真或假,求值过程便告终止,这称之为短路求值.

21、生成一个 10 到 100 的 10 个随机数组成的数组,并排序

```
var arr = [];
for (var i = 0; i <10; i++) {
    var result = Math.floor(Math.random() * (100 - 10)) + 10;
    arr.push(result);
}
arr.sort();             //默认升序
console.log(arr);
```

28、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} .

```
function serilizeUrl(url) {
    var result = {};
    url = url.split("?")[1];                    //获取?后的内容--字符串
    var map = url.split("&");                   //将字符串用&分割成数组进行遍历
    for (var i = 0, len = map.length; i < len; i++) {
        var key = map[i].split("=")[0];         //将每个内容用=分割,分别为键和值
        var value = map[i].split("=")[1];
        result[key] = value;                    //将获取的键值对添加给{}
    }
    return result;
}
console.log(serilizeUrl('http://item.taobao.com/item.htm?a=1&b=2&c=&d=xxx&e'));
```

29、完成下拉菜单功能,要求能够动态根据下拉列表的选项变化,更新图片的显示.

```
//onchange 监听下拉列表改变
document.getElementById('sel').onchange = function () {
    //获取获取当前的 value 进行字符串拼接后赋值给 src,改变图片
    document.getElementById('pic').src = "img/" + this.value + ".jpg";
};
```

30、截取字符串 abcdefg 的 efg

```
var str = "abcdefg";
console.log(str.charAt(1));         //b          单字符
console.log(str.slice(1));          //bcdefg     多字符,一个参数就获取到最后
console.log(str.slice(1,3));        //bc         多字符,二个参数获取 1-3,不包括 3
console.log(str.slice(1,-1));       //bcdef      多字符,参数 2 允许负值
console.log(str.substring(1));      //bcdefg     同上,不允许负值
console.log(str.substring(1,3));    //bc
console.log(str.substr(1));         //bcdefg
console.log(str.substr(1,3));       //bcd        参数 2 是长度
```

31、列举浏览器对象模型 BOM 里常用对象,并列举 window 对象的常用方法

对象:window、document、location、screen、history、navigator

方法:alert()、confirm()、prompt()、open()、close()

补充:

window 对象是 BOM 中所有对象的核心;

history 对象是用来把窗口的浏览历史用文档和文档状态列表的形式表示

back()与 forward()与浏览器的"后退","前进"功能一样.history.go(-2);后退两个历史记录;

location 对象是解析 URL,

location.reload()　重新加载页面

location.replace()　本窗口载入新文档

location.assign()　本窗口载入新文档

navigator 浏览器

navigator.appName　Web 浏览器全称

navigator.appVersion　Web 浏览器厂商和版本的详细字符串

navigator.userAgent 客户端绝大部分信息

navagator.platform　浏览器运行所在的操作系统

screen 窗口

screen.availWidth　可用的屏幕宽度

screen.availHeigh　可用的屏幕高度

32、简述列举文档对象模型 DOM 里 document 的常用的查找访问节点的方法并做简单说明

Document.getElementById	根据元素 id 查找元素
Document.getElementByClassName	根据元素类查找元素(存在兼容问题),得到的是对象数组
Document.getElementTagName	根据指定的元素名查找元素,得到的是对象数组
Document.querySelectorAll	通过 CSS 选择器获取元素,以类数组形式存在

33、希望获取到页面中所有的 checkbox 怎么做?(不使用第三方框架)

```
var inputs = document.getElementsByTagName('input')
var checkBoxList = [];
var len = inputs.length;                    //缓存到局部变量
while (len--) {                             //使用 while 的效率会比 for 循环更高
    if (inputs[len].type == 'checkbox') {
        checkBoxList.push(inputs[len]);
    }
}
console.log(checkBoxList);
```

34、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(转义引号用"\""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.

邮箱的正则匹配公式:var regMail = /^([a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+((.[a-zA-Z0-9_-]{2,3}){1,2})$/;

35、写一个 function 清除字符串前后的空格. (兼容所有浏览器)

```
if (!String.prototype.trim) {                    //如果 String 对象原型中没有 trim 方法就添加
    String.prototype.trim = function () {
```

```
        return this.replace(/^\s+/, "").replace(/\s+$/, "");    // \s 匹配空白字符:回车、换行、制表符 tab 空格
    }
}
```

36、Javascript 中 callee 和 caller 的作用?(递归)

arguments.callee:获得当前函数的引用,尚未出现命名函数之前,为了在匿名函数中实现递归而出现

caller 是返回一个对函数的引用,该函数调用了当前函数

```
var a = function() {
    console.log(a.caller);                    //由于是 b 调用了,所以这里打印的是 b,也就是 function(){a();}
}
var b = function() {
    a();
}
b();
```

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.

```
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 {
            result[n] = arguments.callee(n - 1) + arguments.callee(n - 2);    //argument.callee()表示 fn()
            return result[n];
        }
    }
}
```

37、Javascript 中,以下哪条语句一定会产生运行错误?

变量命名规范:由字母、数字、_、$组成,数字不能开头,不能使用关键字保留字、要区分大小写

1、js中的关键字：

break	case	catch	continue	default
delete	do	else	finally	for
function	if	in	instanceof	new
return	switch	this	throw	try
typeof	var	void	while	with

2、js中的保留字：

abstract	boolean	byte	char	class
const	debugger	double	enum	export
extends	final	float	goto	implements
import	int	interface	long	native
package	private	protected	public	short
static	super	synchronized	throws	transient
volatile				

```
var _变量=NaN;
var 0bj = [];       错误原因:变量以数字开头
var obj = //;       错误原因://表示注释,同时也可表示正则字面量,但是不完全
var obj = {};
```

38、以下两个变量 a 和 b ,a+b 的哪个结果是 NaN ?

```
var a=undefined; b=NaN;
console.log(a+b);    //NaN
var a="123"; b=NaN
console.log(a+b);    //123NaN
var a=NaN,b='undefined'
console.log(a+b);    //NaNundefined
```

39、a++和++a 的区别?

```
//++在前,先加后赋值.++在后,先赋值后运算
var a=10; b=20; c=4;
console.log(++b+c+a++); //35    21+4+10
//for 循环中的自加
for (i = 0, j = 0; i < 10, j < 6; i++, j++) {
    k = i + j;
}
console.log(k);                    //10 5+5,执行了 6 次,每次都是执行完表达式后再自加
```

40、实现检索当前页面中的表单元素中的所有文本框,并将它们全部清空的 javascript 语句是:

```
var forms = document.forms;
for (var i = 0; i < forms.length; i++) {              //遍历所有表单
    var formEles = forms[i].elements;
    for (var j = 0; j < formEles.length; j++) {//每个 form 中的元素
        if (formEles[j].type == "text")
            formEles[j].value = "";
    }
}
```

41、考察数组方法,以及 undefined 元素对数组的影响

```
var arr = new Array(1 ,3 ,5);
arr[4]='z';
console.log(arr[3]);        //undefined
console.log(arr);           //[1, 3, 5, 4: "z"],undefined 不显示,后续内容会显示索引值
arr2 = arr. reverse();   //["z", 2: 5, 3: 3, 4: 1];反向后,1 不显示
console.log(arr);           //["z", 2: 5, 3: 3, 4: 1],arr 本身也会反向
arr3 = arr.concat(arr2);//拼接字符串
console.log(arr3);          //["z", 2: 5, 3: 3, 4: 1, 5: "z", 7: 5, 8: 3, 9: 1]    1 和 6 不显示
```

42、按钮弹框提示,确认用户是否退出当前页面,确认之后关闭窗口;

```
function closeWin(){
```

```
    if(confirm("确定要退出吗")){
        window.close();          //Scripts may close only the windows that were opened by it.
    }                            //这种关闭方式只能关闭使用 window.open()打开的页面
}
```

43、写出去除 html 标签字符串中标签部分的正则表达式

```
var str = "<div>这里是 div<p>里面的段落</p></div>";
var reg = /<\/?\w+\/?>/g;          //注意/需要转译,前面加个\
console.log(str.replace(reg,""));
```

44、JavaScript 的数据类型都有什么?

基本数据类型:String,Boolean,Number,Undefined,Null

引用数据类型:Object(Array,Date,RegExp,Function)

45、javascript 中==和===的区别是什么?举例说明.

==只判断数据值

===判断数据类型和值

46、简述创建函数的几种方式

第一种(函数声明): function sum1(num1,num2){ return num1+num2; }

第二种(函数表达式) : var sum2 = function(num1,num2){ return num1+num2; }

第三种(函数对象方式) : var sum3 = new Function("num1","num2","return num1+num2");

匿名函数: function(){}:只能自己执行自己

47、Javascript 如何实现继承?

原型链继承,借用构造函数继承,组合继承,寄生式继承,寄生组合继承

48、Javascript 创建对象的几种方式?

工厂方式,构造函数方式,原型模式,混合构造函数原型模式,动态原型方式

49、Script 标签在页面最底部 body 封闭之前和封闭之后有什么区别?浏览器会如何解析它们?

如果放在 body 封闭之前,将会阻塞其他资源的加载

如果放在 body 封闭之后,不会影响 body 内元素的加载

50、iframe 的优缺点?(这个不太懂)

优点:

1. 解决加载缓慢的第三方内容如图标和广告等的加载问题(如早起的导航栏)
2. Security sandbox(安全沙箱)
3. 并行加载脚本
4. 几乎没有兼容问题,是浏览器的标准规范

缺点:

1. iframe 会阻塞主页面的 Onload 事件
2. 即时内容为空,加载也需要时间
3. 没有语意

51、请你谈谈 Cookie 的弊端?

1.Cookie`数量和长度的限制.每个 domain 最多只能有 20 条 cookie,每个 cookie 长度不能超过 4KB,否则会被截掉.

2.安全性问题.如果 cookie 被人拦截了,那人就可以取得所有的 session 信息.即使加密也与事无补, 因为拦截者并不需要知道 cookie 的意义, 他只要原样转发 cookie 就可以达到目的了.

3.有些状态不可能保存在客户端.例如,为了防止重复提交表单,我们需要在服务器端保存一个计数器.如果我们把这个计数器保存在客户端,那么它起不到任何作用.

52、写一个获取和设置 css 样式的函数

```
function getStyle(obj, attr, value) {
    if (!value) {                    //获取
        if (obj.currentStyle) {      //ie
            return obj.currentStyle(attr);
        } else {                     // 标准浏览器
            window.getComputedStyle(obj, null)[ attr];      //dom,参数 2 是给伪元素用的,传入伪元素字符串
        }
    }else {                          //设置,如果是 json 还需要遍历
        obj.style[attr] = value;
    }
}
```

53、字符串反转,如将 '12345678' 变成 '87654321'

```
    var str = '12345678';
    str = str.split('').reverse().join('');          //转成数组,反转数组,拼接字符串
```

54、成五个不同的随机数

```
var num = [];
for(var i = 0; i <5; i++){
    num[i] = Math.floor(Math.random()*10) + 1; //范围是 [1, 10]
    for(var j = 0; j < i; j++){
        if(num[i] == num[j]){                //判断是否重复
            i--;
        }
    }
}
```

55、去掉数组中重复的数字

```
//思路:每遍历一次就和之前的所有做比较,不相等则放入新的数组中！这里虽然用了原型,但其实执行函数是一样的
Array.prototype.unique = function(){
    var len = this.length, newArr = [], flag = 1;          //标志位,判断是否重复,1 表示不重复
    for(var i = 0; i < len; i++, flag = 1){
        for(var j = 0; j < i; j++){
            if(this[i] == this[j]){
                flag = 0;                                  //找到相同的数字后,不执行添加数据
            }
        }
        flag ? newArr.push(this[i]) : '';
```

```
    }
    return newArr;
};
```

56、阶乘函数 9*8*7*6*5 … *1(从指定数字一直乘到 1)

```
//原型方法
Number.prototype.N = function(){
    var re = 1;                         //从 1 开始乘
    for(var i = 1; i <= this; i++){ //一直到执行数字 this
        re *= i;
    }
    return re;
}
var num = 5;
console.log(num.N());
```

57、window.location.search/.hash/.reload()

window.location.search 返回值是当前 url 地址中问号加上后面的内容,类型为字符串.如果 url 地址没有问号,则返回空字符串""

https://www.qcloud.com/act/try?t=cvm　　截取后的字符串就是?t=cvm

window.location.hash 返回的是锚点.#号及后面的内容,类型是字符串

http://localhost:63342/20161025/test.html#div　　返回值为"#div" ,类型为 string

window.location.reload() 作用是刷新当前页面

58、阻止事件冒泡

```
function stopPropagation(e) {
    e = e || window.event;
    if(e.stopPropagation) {     //DOM 阻止冒泡方法
        e.stopPropagation();
    } else {
        e.cancelBubble = true;  //IE 阻止冒泡方法
    }
}
```

59、javascript 中的垃圾回收机制

在 Javascript 中,如果一个对象不再被引用,那么这个对象就会被 GC 回收.如果两个对象互相引用,而不再被第 3 者所引用,那么这两个互相引用的对象也会被回收.因为函数 a 被 b 引用,b 又被 a 外的 c 引用,函数 a 执行后不会被回收

60、函数作为普通函数和构造函数的区别

```
function f1(){
    var tmp = 1;
    this.x = 3;
    console.log(tmp);
    console.log(this.x);
}
var obj = new f1();         //1 3,因为实例化时执行,this 指向实例对象
```

```
console.log(obj.x);          //3,实例化时指定 x=3
console.log(f1());           //1 3 undefined,会调用,作为普通函数时,this 指向 window,但是没有返回值
```

61、引用数据类型

```
var o1 = new Object();
var o2 = o1;
o2.name = "CSSer";
console.log(o1.name);                    //CSSer ,因为是引用数据类型,变量按照引用访问,2 个变量都指向同一个对象

function changeObjectProperty (o) {      //引用数据类型作为实参传入后,会影响复杂数据类型的数据
    o.siteUrl = "http://www.csser.com/";
    o = new Object();                    //重新赋值后,o 成为了局部变量,不再影响原对象 o
    o.siteUrl = "http://www.popcg.com/";
}
var CSSer = new Object();
changeObjectProperty(CSSer);
console.log(CSSer.siteUrl); //http://www.csser.com
```

62、输出多少?

```
var a = 6;
setTimeout(function () {
    console.log(a);      //undefined,提升了 var a,但是没有赋值      如果没有 var a = 666,那么由于异步执行,这里是 66
    var a = 666;         //此时定义了局部变量,var 会出现变量提升
    console.log(a);      //666,根据作用域链查找,优先查找局部变量   如果没有 var a = 666,那么由于异步执行,这里是 66
}, 1000);
a = 66;                  //这个是不会被变量提升的,没有 var
```

63、JS 的继承属性,this 指向和 call 方法

```
window.color = 'red';
var o = {color: 'blue'};
function sayColor(){
    console.log(this.color);
}
sayColor();                  //red    此时 this 是指向 window
sayColor.call(this);         //red    this 指向的是 window 对象
sayColor.call(window);       //red
sayColor.call(o);            //blue   this 指向 o 对象
```

64、精度问题:JS 精度不能精确到 0.1,所以,同时存在于值和差值中

```
var n = 0.3, m = 0.2, i = 0.2, j = 0.1;
console.log((n - m) == (i - j));          //false  精度问题,js 中浮点数精度存在精度问题,会有误差
console.log((n-m) == 0.1);                //false
console.log((i-j) == 0.1);                //true   这是特例?
```

65、加减运算(+的字符串拼接、-的隐式转换)

```
console.log('5' + 3);                //53 string  字符串拼接
console.log('5' +'3');               //53 string  字符串拼接
console.log('5' - 3);                //2 number   隐式转换
console.log('5' -'3');               //2 number   隐式转换
```

66、对象的方法,实例对象的方法

```
function foo(){
    foo.a = function(){console.log(1)};
    this.a = function(){console.log(2)};
    a = function(){console.log(3)};
    var a = function(){console.log(4)};
};
foo.prototype.a = function(){console.log(5)};
foo.a = function(){console.log(6)};
foo.a();                     //6  最后一个重置了 a 方法
var obj = new foo();
obj.a();                     //2 new 时只看 this,其他的不会实例化出现
foo.a();                     //1  实例化时虽然不会出现在实例对象中,但是执行了 foo.a 重新声明函数
```

67、函数作为普通函数和构造函数执行时的区别

```
var a = 5;
function test() {
    a = 0;
    console.log(a);          //作为普通函数执行时,a=0,作为构造函数执行时,a=0
    console.log(this.a);     //作为普通函数时,this 为 window,所以 a=5,作为构造函数执行时,未赋值 undefined
    var a;                   //变量提升,没任何影响
    console.log(a);          //同 1 相同,因为上面的 var 变量提升了
}
test();                      // 0, 5, 0          作为普通函数执行
new test();                  // 0, undefined, 0  作为构造函数执行
```

68、计算字符串字节数(charCodeAt()得到的是 unCode 码,大于 255 就是汉字)

```
new function(s){
    if(!arguments.length||!s) return null;   //未传参或者传入的参数可以转为 false 则返回 null
    if(""==s) return 0;                      //和上面的!s 重复
    var l=0;
    for(var i=0;i<s.length;i++){
        if (s.charCodeAt(i)>255){            //charCodeAt()得到的是 unCode 码,汉字的 unCode 码大于 255bit 就是两个字节
            l+=2;
        } else {
            l+=1;
        }
    }
    console.log(l);                          //12,都不是汉字
```

```
}("hello world!");
```

69、声明对象,添加属性,输出属性

```
var obj = {
    name: 'leipeng',
    showName: function(){
        console.log(this.name);                    //'leipeng',this 指向 obj
    }
};
obj.showName();
```

70、正则表达式

```
var reg = /^[a-zA-Z_][a-zA-Z0-9_]{4,19}/,    //以字母或下划线开头,后面字母下划线数字出现 4-19 次
        name1 = 'leipeng',
        name2 = '0leipeng',
        name3 = '你好  leipeng',
        name4 = 'hi';
console.log(reg.test(name1));    //true
console.log(reg.test(name2));    //false
console.log(reg.test(name3));    //false
console.log(reg.test(name4));    //false
```

71、JS 中的使用 call 实现借用继承

```
//定义父类
function Parent(name, money){
    this.name = name;
    this.money = money;
    this.info = function(){
        console.log('姓名: '+this.name+'  钱: '+ this.money);
    }
}
//定义子类
function Children(name){
    Parent.call(this, name);
    this.info = function(){
        console.log('姓名: '+this.name);
    }
}
//实例化类
var per = new Parent('parent', 800000000000);
var chi = new Children('child');
per.info();
chi.info();
```

72、typeof 的返回类型有哪些?

```
console.log(typeof [1, 2]);          //object
console.log(typeof 'leipeng');       //string
var i = true;
console.log(typeof i);               //boolean
console.log(typeof 1);               //number
var a;
console.log(typeof a);               //undefined
function a(){;};
console.log(typeof a)                //function
```

73、解析 URL 成一个对象(键值对)

```
String.prototype.urlQueryString = function() {
    var url = this.split('?')[1].split('&'),                    //先获取?后的内容,再用&分割成数组
            len = url.length;
    var rst = {};
    for (var i = 0; i < len; i += 1) {
        var cell = url[i].split('='),
                key = cell[0],                                  //键
                val = cell[1];                                  //值
        rst['' + key + ''] = val;
    }
    return rst;
}
```

74、你如何优化自己的代码?

代码重用

避免全局变量(命名空间,封闭空间,模块化 mvc..)

拆分函数避免函数过于臃肿:单一职责原则

适当的注释,尤其是一些复杂的业务逻辑或者是计算逻辑,都应该写出这个业务逻辑的具体过程

内存管理,尤其是闭包中的变量释放

75、需要将变量"abcd"的值修改为" a+b+c+d "

```
var e1 = "abcd";
var arr = e1.split("");                 //分割成数组
var e2 = e1[0];                         //第一个不变
for(var i = 1 ;i < arr.length ; i++){   //后面都先加一个+
    e2 += "+" + arr[i];
}
console.log(e2);
```

76、怎样实现两栏等高?

```
.box {
    overflow: hidden;               /*父盒子设置溢出清除*/
}
.div1 {
```

```
    height: 200px;
    width: 50px;
    float: left;
    background-color: skyblue;
    margin-bottom: -100px;          /*浮动的子元素设置绝对值相等的 padding-bottom 正值和 margi-bottom 负值*/
    padding-bottom: 100px;
}
.div2 {
    height: 150px;
    width: 50px;
    float: left;
    background-color: lightseagreen;
    margin-bottom: -100px;
    padding-bottom: 100px;
}
```

77、实现在文本域里输入文字按下 enter 键时不换行,而是替换成" {{enter}} "(只考虑行尾)

```
var textarea = document.getElementById("textarea");
textarea.onkeydown = function (e) {
    e.preventDefault();                 //为了阻止 enter 键的默认换行效果
    console.log(e.keyCode);             //13
    if (e.keyCode == 13) {
        console.log(1);
        textarea.value += "{{enter}}";
    }
}
```

78、以下代码中 end 字符串在什么时候输出

```
var t = true;
setTimeout(function () {
    console.log(123);
    t = false;
}, 1000);
while (t) {}          //死循环,上面的 setTimeout 也不会执行,下面的也不会执行
console.log('end');
```

79、请用原生 js 实现 jquery 的 get\post,以及跨域情况下

```
function AJAX() {
    var xhr = null;
    if(window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
    } else {
        try{
            xhr = new ActiveXObject('Microsoft.XMLHttp');
        }catch(e){
```

```
            xhr = new ActiveXObject('msxml2.xmlhttp');
        }
    }
    //get 请求
    this.get = function(url,success,fail){
        xhr.open("GET", "1.jsp",true);
        xhr.onreadystatechange=function(){
            if(xhr.readyState==4) {
                alert(xhr.status);
                if(xhr.status==200) {
                    var txt = xhr.responseText;
                    txt = eval("("+txt+")");
                    var ch = txt.charAt(0);
                    if(ch=="<") { //xml 类型
                        var xml = xhr.responseXML;
                        success(eval("("+xml+")"));//【慎重使用 eval】
                    } else if(ch=="["||ch=="{") {//json 类型
                        txt = eval("("+txt+")");//【慎重使用 eval】
                        success(txt);
                    } else {
                        success(txt);
                    }
                } else {
                    if(fail) {
                        fail(xhr.status);
                    }
                }
            }
        };
        xhr.send(null);
    };
    //post 请求
    this.post = function (url,param,success,fail) {
        xhr.open("POST", "1.jsp",true);
        xhr.onreadystatechange=function(){
            if(xhr.readyState==4) {
                alert(xhr.status);
                if(xhr.status==200) {
                    var txt = xhr.responseText;
                    var ch = txt.charAt(0);
                    if(ch=="<") { //xml 类型
                        var xml = xhr.responseXML;
                        success(eval("("+xml+")"));//【慎重使用 eval】
                    } else if(ch=="["||ch=="{") {//json 类型
```

```
                txt = eval("("+txt+")");//【慎重使用 eval】
                success(txt);
            } else {
                success(txt);
            }
        } else {
            if(fail) {
                fail(xhr.status);
            }
        }
    }
    };
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xhr.send(param);
};
}
```

80、简述 readyonly 与 disabled 的区别

readonly 只针对 input(text/password)和 textarea 有效

disabled 对于所有的表单元素都有效

当表单以 POST 或 GET 的方式提交的话, disabled 元素的值不会被传递出去,而 readonly 会将该值传递出去

81、判断一个字符串出现次数最多的字符,统计这个次数并输出

```
function get (str) {
    for(var i = 0,obj = {},k ;i < str.length ; i++){
        k = str.charAt(i);
        obj[k] ? obj[k]++ : obj[k]=1;          /*如果已经存在,则计数+1,否则定义为 1 次*/
        var key,m = 0,j ;
        for( key in obj ) {                    //判断最大值
            if(obj[key] > m){
                m = obj[key];
                j = key;
            }
        }
    }
    console.log(m , j);
}
var str = "ssdffssggg";
get(str);                                      //4 "s"
```

82、写出 3 个使用 this 的典型应用

构造函数中的 this,原型对象中的 this,对象字面量中的 this

83、请尽可能详尽的解释 ajax

思路:先解释异步,再解释 ajax 如何使用(后面还有专题)

Ajax 的原理简单来说通过 XmlHttpRequest 对象来向服务器发异步请求,从服务器获得据,然后用 javascript 来操作 DOM 而更新页面.这其中最关键的一步就是从服务器获得请求数据.要清楚这个过程和原理,我们必须对 XMLHttpRequest 有所了解.XMLHttpRequest 是 ajax 的核心机制,它是在 IE5 中首先引入的,是一种支持异步请求的技术.简单的说,也就是 javascript 可以及时向服务器提出请求和处理响应,而不阻塞用户.达到无刷新的效果

84、为什么扩展 javascript 内置对象不是好的做法?

因为扩展内置对象会影响整个程序中所使用到的该内置对象的原型属性

85、如果设计中使用了非标准的字体,你该如何去实现?

先通过 font-face 定义字体,再引用

```
@font-face
{
  font-family:myFirstFont;
  src: url('Sansation_Light.ttf'),
       url('Sansation_Light.eot'); /* IE9+ */
};
```

163. 用 css 分别实现某个 div 元素上下居中和左右居中

知道 div 的宽高:不知道 div 的宽高

```
width: 200px;                    margin:atuo;
height: 200px;                   position:absolute;
position: absolute;              top: 0;
top: 50%;                        bottom: 0;
left:50%;                        left: 0;
margin-top:-100px;               right: 0;
margin-left: -100px;
```

85、HTTP 协议中,GET 和 POST 有什么区别?分别使用什么场景?

get 传送的数据长度有限制,post 没有

get 通过 url 传递,在浏览器地址栏可见,post 是在报文中传递

post 一般用于表单提交.

get 一般用于简单的数据查询,严格要求不是那么高的场景

86、HTTP 状态消息 200 302 304 403 404 500 分表表示什么?

200:请求已成功,请求所希望的响应头或数据体将随此响应返回

302:请求的资源临时从不同的 URL 响应请求.由于这样的重定向是临时的,客户端应当继续向原有地址发送以后的请求,只有在 Cache-Control 或 Expires 中进行了指定的情况下,这个响应才是可缓存的.

304:如果客户端发送了一个带条件的 GET 请求且该请求已被允许,而文档的内容(自上次一来或者根据请求的条件)并没有改变,则服务器应当返回这个状态码.304 响应禁止包含消息体,因此始终以消息头后的第一个空行结尾.

403:服务器已经理解请求,但是拒绝执行它.

404:请求失败,请求所希望得到的资源未被在在服务器上发现.

500:服务器遇到了一个未曾预料的状况,导致了它无法完成对请求的处理.一般来说,这个问题都会在服务器端的源代码出现错误时出现.

87、HTTP 协议中,header 信息里面,怎么控制页面失效时间(last-modified, cache-control, Expires 分别代表什么)

HTTP(HyperTextTransferProtocol)即超文本传输协议,目前网页传输的的通用协议.HTTP 协议采用了请求/响应模型,浏览器或其他客户端发出请求,服务器给与响应.就整个网络资源传输而言,包括 message-header 和 message-body 两部分.首先传递 message-header,即 httpheader 消息.http header 消息通常被分为 4 个部分:general header, request header, response header, entity header.但是这种分法就理解而言,感觉界限不太明确.根据维基百科对 http header 内容的组织形式,大体分为 Request 和 Response 两部分.

last-modified 处在 Response 中,代表请求资源的最后修改时间.客户可以通过 If-Modified-Since 请求头提供一个日期,该请求将被视为一个条件 GET,只有改动时间迟于指定时间的文档才会返回,否则返回一个 304(Not Modified)状态.Last-Modified 也可用 setDateHeader 方法来设置.

Expires 处在 Response 中,代表响应过期的日期和时间.

cache-control 指定请求和响应遵循的缓存机制请求时的缓存指令包括 no-cache、no-store、max-age、 max-stale、min-fresh、only-if-cached,响应消息中的指令包括 public、private、no-cache、no- store、no-transform、must-revalidate、proxy-revalidate、max-age.各个消息中的指令含义如下:Public 指示响应可被任何缓存区缓存.Private 指示对于单个用户的整个或部分响应消息,不能被共享缓存处理.这允许服务器仅仅描述当用户的部分响应消息,此响应消息对于其他用户的请求无效.no-cache 指示请求或响应消息不能缓存.no-store 用于防止重要的信息被无意的发布.在请求消息中发送将使得请求和响应消息都不使用缓存.max-age 指示客户机可以接收生存期不大于指定时间(以秒为单位)的响应.min-fresh 指示客户机可以接收响应时间小于当前时间加上指定时间的响应.max-stale 指示客户机可以接收超出超时期间的响应消息.如果指定 max-stale 消息的值,那么客户机可以接收超出超时期指定值之内的响应消息.

88、HTTP 协议目前常用的有哪几个?KEEPALIVE 从哪个版本开始出现的?

超文本传输协议已经演化出了很多版本,它们中的大部分都是向下兼容的.在 RFC 2145 中描述了 HTTP 版本号的用法.客户端在请求的开始告诉服务器它采用的协议版本号,而后者则在响应中采用相同或者更早的协议版本.

0.9 已过时.只接受 GET 一种请求方法,没有在通讯中指定版本号,且不支持请求头.由于该版本不支持 POST 方法,所以客户端无法向服务器传递太多信息.

HTTP/1.0 这是第一个在通讯中指定版本号的 HTTP 协议版本,至今仍被广泛采用,特别是在代理服务器中.

HTTP/1.1 当前版本.持久连接被默认采用,并能很好地配合代理服务器工作.还支持以管道方式同时发送多个请求,以便降低线路负载,提高传输速度.

HTTP/1.1 相较于 HTTP/1.0 协议的区别主要体现在:1 缓存处理,2 带宽优化及网络连接的使用,3 错误通知的管理,4 消息在网络中的发送,5 互联网地址的维护,6 安全性及完整性

keep-Alive 功能使客户端到服务器端的连接持续有效,当出现对服务器的后继请求时,Keep-Alive 功能避免了建立或者重新建立连接.市场上的大部分 Web 服务器,包括 iPlanet、IIS 和 Apache,都支持 HTTP Keep-Alive.对于提供静态内容的网站来说,这个功能通常很有用.但是,对于负担较重的网站来说,这里存在另外一个问题:虽然为客户保留打开的连接有一定的好处,但它同样影响了性能,因为在处理暂停期间,本来可以释放的资源仍旧被占用.当 Web 服务器和应用服务器在同一台机器上运行时,Keep- Alive 功能对资源利用的影响尤其突出.**此功能为 HTTP 1.1 预设的功能,HTTP 1.0 加上 Keep-Alive header 也可以提供 HTTP 的持续作用功能.**

89、业界常用的优化 WEB 页面加载速度的方式(可以分别从页面元素展现,请求连接,css,js,服务器等方面介绍)

(1) 优化图像:这个绝对是显而易见的,可以看到图片占据的页面内容分量最重.在现代网页设计中,图片绝对占据了大部分的内容.你需要针对你的页面重新定义图片大小.这能够有效地帮助你减少页面大小.

(2) 减少 DNS 查询(DNS lookups):减少 DNS 查询是一个 WEB 开发人员可以用了页面加载时间快速有效的方法.DNS 查询需要话费很长的时间来返回一个主机名的 IP 地址.而浏览器在查询结束前不会进行任何操作.对于不同的元素可以使用不同的主机名,如 URL、图像、脚本文件、样式文件、FLASH 元素等.具有多种网络元素的页面经常需要进行多个 DNS 查询,因而花费的时间更长.减少不同域名的数量将减少并行下载的数量,加速你的网站

(3) 减少 HTTP 请求:还有一种简单的优化网页速度的方法是,减少 HTTP 请求.当一个网站一下子收到太多的 HTTP 请求,它的访客就会有响应时间延迟的体验,这不仅增加了 CPU 使用率也增加了页面的加载时间.那么,又该如何减少 HTTP 请求?减少网站上的对象数量;最小化网站上的重定向数量;使用 CSS Sprites 技术(只要你需要的那部分图片内容).

(4) 使用内容分发网络(Content Delivery Network CDN):服务器处理大流量是很困难的,这最终会导致页面加载速度变慢.而使用 CDN 就可以解决这一问题,提升页面加载速度.CDN 是位于全球不同地方的高性能网络服务,复制你网络的静态资源,并以最有效的方式来为访客服务.

(5) 把 CSS 文件放在页面顶部,而 JS 文件放在底部:把 CSS 文件在页面底部引入可以禁止逐步渲染,节省浏览器加载和重绘页面元素的资源.JavaScript 是用于功能和验证.把 JS 文件放在页面底部可以避免代码执行前的等待时间,从而提升页面加载速度.这些都是一些减少页面加载时间和提高转换率的方法.在某些情况下,需要 JavaScript 在页面的顶部加载(如某些第三方跟踪脚本).

(6) 压缩 CSS 和 JavaScript:压缩是通过移除不必要的字符(如 TAB、空格、回车、代码注释等),以帮助减少其大小和网页的后续加载时间的过程.这是非常重要的,但是,你还需要保存 JS 和 CSS 的原文件,以便更新和修改代码.

(7) 利用浏览器缓存:浏览器缓存是允许访客的浏览器缓存你网站页面副本的一个功能.这有助于访客再次访问时,直接从缓存中读取内容而不必重新加载.这节省了向服务器发送 HTTP 请求的时间.此外,通过优化您的网站的缓存系统往往也会降低您的网站的带宽和托管费用.

(8) 启用 GZIP 压缩:在服务器上压缩网站的页面是提升网站访问速度非常有效的一种方法.你可以用 gzip 压缩做到这一点.Gzip 是一个减小发送给访客的 HTML 文件、JS 和 CSS 体积的工具.压缩的文件减少了 HTTP 响应时间.据 Yahoo 报道,这大概可以减少 70%的下载时间.而目前 90%的通过浏览器的流量都支持 Gzip 压缩,因此,这是一个提示网站性能有效的选项.

90、列举常用的 web 页面开发,调试以及优化工具

常用的 IDE(集成开发环境,Integrated Development Environment):sublime, webstorm, vscode, hbuilder,

常用的调试工具:Firebug(FF),Web Developer(IE),IETester(浏览器兼容性测试).浏览器自带的控制台?

常用的优化工具:

Google 提供了 PageSpeed 工具,这是一个浏览器插件,可以很好地应用上文中 Google 所提到的 Web 优化实践——帮助你轻松对网站的性能瓶颈进行分析,并为你提供优化建议.

YSlow 是雅虎推出的一款浏览器插件,可以帮助你对网站的页面进行分析,并为你提供一些优化建议,以提高网站的性能.

91、解释什么是 sql 注入,xss 漏洞

结构化查询语言(Structured Query Language)简称 SQL,是一种特殊目的的编程语言,是一种数据库查询和程序设计语言,用于存取数据以及查询、更新和管理关系数据库系统;同时也是数据库脚本文件的扩展名.

所谓 SQL 注入,就是通过把 SQL 命令插入到 Web 表单提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.具体来说,它是利用现有应用程序,将(恶意)的 SQL 命令注入到后台数据库引擎执行的能力,它可以通过在 Web 表单中输入(恶意)SQL 语句得到一个存在安全漏洞的网站上的数据库,而不是按照设计者意图去执行 SQL 语句.

XSS 攻击:跨站脚本攻击(Cross Site Scripting),为不和层叠样式表(Cascading Style Sheets, CSS)的缩写混淆,故将跨站脚本攻击缩写为 XSS.XSS 是一种经常出现在 web 应用中的计算机安全漏洞,它允许恶意 web 用户将代码植入到提供给其它用户使用的页面中.比如这些代码包括 HTML 代码和客户端脚本.攻击者利用 XSS 漏洞旁路掉访问控制——例如同源策略(same origin policy).这种类型的漏洞由于被黑客用来编写危害性更大的网络钓鱼(Phishing)攻击而变得广为人知.对于跨站脚本攻击,黑客界共识是:跨站脚本攻击是新型的"缓冲区溢出攻击",而 JavaScript 是新型的"ShellCode".

92、如何判断一个 js 变量是数组类型

ES5:Array.isArray();

[] instanceof Array;

调用数组的 slice 方法;

Object.prototype.toString.call([]); // "[object Array]"

93、请列举 js 数组类型中常用的方法

方法	描述	FF	IE
concat()	连接两个或更多的数组,并返回结果.	1	4

join()	把数组的所有元素放入一个字符串.元素通过指定的分隔符进行分隔.	1	4
pop()	删除并返回数组的最后一个元素	1	5.5
push()	向数组的末尾添加一个或更多元素,并返回新的长度.	1	5.5
reverse()	颠倒数组中元素的顺序.	1	4
shift()	删除并返回数组的第一个元素	1	5.5
slice()	从某个已有的数组返回选定的元素	1	4
sort()	对数组的元素进行排序	1	4
splice()	删除元素,并向数组添加新元素.	1	5.5
toSource()	返回该对象的源代码.	1	-
toString()	把数组转换为字符串,并返回结果.	1	4
toLocaleString()	把数组转换为本地数组,并返回结果.	1	4
unshift()	向数组的开头添加一个或更多元素,并返回新的长度.	1	6
valueOf()	返回数组对象的原始值	1	4

94、列举常用的 js 框架以及分别适用的领域(更新补充)

jquery:简化了 js 的一些操作,并且提供了一些非常好用的 API

jqueryui, **jqueryeasyui**: 在 jquery 的基础上提供了一些常用的组件:日期,下拉框,表格等

require.js, **sea.js**(阿里的玉帛): 模块化开发使用

zepto: 精简版的 jquery,常用于手机 web 前端开发,提供了一些手机页面实用功能,如 touch

ext.js: 跟 jquery 差不多,但是不开源,也没有 jquery 轻量

angular, **knockoutjs**, **avalon**(去哪儿前总监,司徒正美): MV*框架,适用于单页应用开发(SPA,single page web application)

95、js 中如何实现一个 map

数组的 map 方法:返回一个由原数组中的每个元素调用一个指定方法后的返回值组成的新数组

语法:array .map(callback [, thisArg])

返回值:原数组中的元素经过该方法后返回一个新的元素.

参数:callback

currentValue:callback 的第一个参数,数组中当前被传递的元素.

index:callback 的第二个参数,数组中当前被传递的元素的索引.

array:callback 的第三个参数,调用 map 方法的数组.

thisArg:执行 callback 函数时 this 指向的对象.

实现:

```
Array.prototype.map2 = function(callback) {
    for (var i = 0; i <this.length; i++) {
        this[i] = callback(this[i]);
    };
};
```

96、js 可否实现面向对象编程,如果可以如何实现 js 对象的继承

创建对象的方式:

(1)new Object()　　varobj = new Object();

(2)自定义构造函数　　function Person () {};

(3)对象字面量 varobj = {};

实现继承:

(1)原型继承 student.prototype = new Person();

(2)借用构造函数　function Student () {Person.call(this, name,age)};

(3)组合继承 Person.call(this.name);　Student.prototype = new Person();

(4)克隆继承 object.extend = function(obj1, obj2) {

for (var key in obj1) {

obj2[key] = obj1[key];

};

};

(5)object.create()　　var obj3 = object.create(obj4);

97、约瑟夫环----已知 n 个人(以编号 1,2,3 ••• 分别表示)围坐在一张圆桌周围.从编号为 k 的人开始报数,数到 m 的那个人出列;他的下一个人又从 1 开始报数,数到 m 的那个人又出列;依次规律重复下去,直到圆桌周围的人全部出列.

简单验证了没问题,但是总感觉有点不对劲,请大家再研究一下

```
// n 个人,从 k 开始,数 m 个数
var Josef = function(n, k, m){
    // 将所有参与的人放入一个数组
    var players = [];
    for(var i = 1; i <= n; i++){
        players.push(i);
    };
    var flag = -k + 1;
    // 用 flag 来标记当前玩家数到多少,因为是从第 k 个人开始数,那么 flag 初始值为-k+1
    while(players.length> 1){
        varoutPlayerNum = 0,len = players.length;
        for(var i = 0; i <len; i++){
            flag++;
            if(flag === m){ // 当数到 m 的时候把对应的玩家踢出
                flag = 0;
                if(players.length> 1) {
                    console.log("出局:" + players[i - outPlayerNum]);
                    players.splice(i - outPlayerNum, 1);
                    outPlayerNum++;
                };
            };
        };
    };
    return players[0];
};
console.log("最后一个玩家为:"+Josef(5, 1, 4));
```

98、有 1 到 10w 这 10w 个数,去除 2 个并打乱次序,如何找出那两个数?

位图解决:

假设待处理数组为 A[10w-2],定义一个数组 B[10w],这里假设 B 中每个元素占用 1 比特,并初始化为全 0

```
for(i=0;i <10w-2;i++){
    B[ A[i] ]=1
}
```

那么 B 中为零的元素即为缺少的数据,这种方法的效率非常高,是计算机中最常用的算法之一

99、如何获取对象 a 拥有的所有属性(可枚举的、不可枚举的,不包括继承来的属性)

IE9+ :Objects.keys

或者使用 for in 并过滤出继承的属性

```
for (o in obj) {
    if (obj.hasOwnproterty(o) ) {// 把 o 这个属性放入到一个数组中;
    };
};
```

100、有下面这样一段 HTML 结构,使用 CSS 实现这样的效果:

左边容器无论宽度如何变动,右边容器都能自适应填满父容器剩余的宽度.

```
<style>
    .wrap {                    //父容器
        width: 600px; height: 100px; background-color: red;
        display: flex;
    }
    .left {
        width: 50px; height: 100px; background-color: green;
    }
    .right {
        flex: auto; height: 100px; background-color: blue;
    }
</style>
```

101、如果下面这段代码想要循环输出结果 01234 ,请问输出结果是否正确,如果不正确,请说明为什么,并修改循环内的代码使其输出正确结果

```
for(var i=0;i<5;++i){
    setTimeout(function(){
        console.log(i+"");
    },100*i);
    console.log(0);                    //先输出 5 个 0.然后顺序打印 5 个 5,因为定时器解析到最后执行
};
for(var i = 0; i < 5; i++) {
    console.log(i + "");              //12345
};
```

102、JavaScript 以下哪条语句会产生运行错误

A. varobj = (); B. varobj = []; C. varobj = {}; D. varobj = //;

答案:AD　　bc 是字面量定义

103、以下哪些是 JavaScript 的全局函数(ABCDE)

A. escape　函数可对字符串进行编码,这样就可以在所有的计算机上读取该字符串.

ECMAScript v3 反对使用该方法,应用使用 decodeURI() 和 decodeURIComponent()替代它.

B. parseFloatparseFloat() 函数可解析一个字符串,并返回一个浮点数.

该函数指定字符串中的首个字符是否是数字.如果是,则对字符串进行解析,直到到达数字的末端为止,然后以数字返回该数字,而不是作为字符串.

C. eval 函数可计算某个字符串,并执行其中的的 JavaScript 代码.

D. setTimeout

E. alert

104、关于 IE 的 window 对象表述正确的有(CD)

A. window.opener 属性本身就是指向 window 对象

window.opener 返回打开当前窗口的那个窗口的引用.如果当前窗口是由另一个窗口打开的, window.opener 保留了那个窗口的引用. 如果当前窗口不是由其他窗口打开的, 则该属性返回 null.

B. window.reload()方法可以用来刷新当前页面　　//正确答案:应该是 location.reload 或者 window.location.reload

C. window.location="a.html"和 window.location.href="a.html"的作用都是把当前页面替换成 a.html 页面

D. 定义了全局变量 g;可以用 window.g 的方式来存取该变量

105、描述错误的是 D

A:Http 状态码 302 表示暂时性转移　　//对

B:domContentLoaded 事件早于 onload 事件　　//正确当 onload 事件触发时,页面上所有的 DOM,样式表,脚本,图片,flash 都已经加载完成了.当 DOMContentLoaded 事件触发时,仅当 DOM 加载完成,不包括样式表,图片,flash.

C: IE678 不支持事件捕获

D:localStorage 存储的数据在电脑重启后丢失　　//错误,因为没有时间限制 try...catch 语句.(在 IE5+、Mozilla 1.0、和 Netscape 6 中可用

106、关于 link 和 @import 的区别正确的是 ABD?

A: link 属于 XHTML 标签,而@import 是 CSS 提供的;

B:页面被加载时,link 会同时被加载,而后者引用的 CSS 会等到页面被加载完再加载

C:import 只在 IE5 以上才能识别而 link 是 XHTML 标签,无兼容问题

D: link 方式的样式的权重高于@import 的权重

107、下面正确的是 A

A: 跨域问题能通过 JsonP 方案解决

B:不同子域名间仅能通过修改 window.name 解决跨域　　//还可以通过 script 标签 srcjsonp

C:只有在 IE 中可通过 iframe 嵌套跨域　　//任何浏览器都可以使用 iframe

D:MediaQuery 属性是进行视频格式检测的属性是做响应式的

108、错误的是 AC

A: Ajax 本质是 XMLHttpRequest　　//异步请求 json 和 xml 数据

B: 块元素实际占用的宽度与它的 width、border、padding 属性有关,与 background 无关

C: position 属性 absolute、fixed、relative---会使文档脱标

D: float 属性 left 也会使 div 脱标

109、不用任何插件,如何实现一个 tab 栏切换

通过改变不同层的 css 设置层的显示和隐藏

110、基本数据类型的专业术语及单词拼写

Number, String, Noolean,Null, Undefined

111、变量的命名规范以及命名推荐

(1)变量名只能由英文字母,数字,下划线以及$符号组成,并且数字不能放在名称开头.

(2)变量的命名不能使用 javascript 中的关键字和保留字.

(3)区分大小写

(4)推荐使用驼峰命名

112、三种弹窗的单词以及三种弹窗的功能

alert　弹出一个提示框

confirm　弹出一个询问框,确定或取消

prompt　弹出一个输入框

113、console.log(8 | 1) 输出的值是多少

转换成二进制以后按位取或,8 是 1000,1 是 0001,按位取或以后就是 1001,也就是 9

114、只允许使用+ - - * /和 Math.*,求一个函数 y = f(x, a, b);当 x>100 时返回 a 的值,否则返回 b 的值,不能使用 if else 等条件语句,也不能使用 |, ? : , 数组

```
function fn(x, a, b) {
    var temp = Math.ceil(Math.min(Math.max(x - 100, 0),1));
    // 先想到返回值应该是 a,b 与同一个变量的关系式然后找到这个变量的表达式,全都是套路!!!
    var result = a * temp + b * (1 -temp);    //temp=0 表示小于 100,temp=1 表示大于 100
    return result;
};
console.log(fn(101,3,5));
```

115、JavaScript alert(0.4 * 0.2)结果是多少?浮点数精度问题

结果为 0.08000000000000002

原因,JavaScript 只有一种数字类型 Number,而且在 Javascript 中所有的数字都是以 IEEE-754 标准格式表示的.浮点数的精度问题不是 JavaScript 特有的,因为有些小数以二进制表示位数是无穷的

解决办法:(1)简单粗暴,先都乘以一个较大的数 a 转换为整数然后再除以这个数 a

(2)重写浮点运算的函数(下面是补充别背了)

```
//加法函数,用来得到精确的加法结果
//说明:javascript 的加法结果会有误差,在两个浮点数相加的时候会比较明显.这个函数返回较为精确的加法结果.
//调用:accAdd(arg1,arg2)
//返回值:arg1 加上 arg2 的精确结果
function accAdd(arg1,arg2){
    var r1,r2,m;
    try{r1=arg1.toString().split(".")[1].length}catch(e){r1=0}
    try{r2=arg2.toString().split(".")[1].length}catch(e){r2=0}
    m=Math.pow(10,Math.max(r1,r2));
```

```
    return (arg1*m+arg2*m)/m;
}
//给 Number 类型增加一个 add 方法,调用起来更加方便.
Number.prototype.add = function (arg){
    return accAdd(arg,this);
}
//减法函数
function accSub(arg1,arg2){
    var r1,r2,m,n;
    try{r1=arg1.toString().split(".")[1].length}catch(e){r1=0}
    try{r2=arg2.toString().split(".")[1].length}catch(e){r2=0}
    m=Math.pow(10,Math.max(r1,r2));
    //last modify by deeka
    //动态控制精度长度
    n=(r1>=r2)?r1:r2;
    return ((arg2*m-arg1*m)/m).toFixed(n);
}
///给 number 类增加一个 sub 方法,调用起来更加方便
Number.prototype.sub = function (arg){
    return accSub(arg,this);
}
//乘法函数,用来得到精确的乘法结果
//说明:javascript 的乘法结果会有误差,在两个浮点数相乘的时候会比较明显.这个函数返回较为精确的乘法结果.
//调用:accMul(arg1,arg2)
//返回值:arg1 乘以 arg2 的精确结果
function accMul(arg1,arg2)
{
    var m=0,s1=arg1.toString(),s2=arg2.toString();
    try{m+=s1.split(".")[1].length}catch(e){}
    try{m+=s2.split(".")[1].length}catch(e){}
    return Number(s1.replace(".",""))*Number(s2.replace(".",""))/Math.pow(10,m);
}
//给 Number 类型增加一个 mul 方法,调用起来更加方便.
Number.prototype.mul = function (arg){
    return accMul(arg, this);
};
//除法函数,用来得到精确的除法结果
//说明:javascript 的除法结果会有误差,在两个浮点数相除的时候会比较明显.这个函数返回较为精确的除法结果.
//调用:accDiv(arg1,arg2)
//返回值:arg1 除以 arg2 的精确结果
function accDiv(arg1,arg2){
    var t1=0,t2=0,r1,r2;
    try{t1=arg1.toString().split(".")[1].length}catch(e){}
    try{t2=arg2.toString().split(".")[1].length}catch(e){}
```

```
    with(Math){
        r1=Number(arg1.toString().replace(".",""));
        r2=Number(arg2.toString().replace(".",""));
        return (r1/r2)*pow(10,t2-t1);
    }
}
//给 Number 类型增加一个 div 方法,调用起来更加方便.
Number.prototype.div = function (arg){
    return accDiv(this, arg);
};
```

115、一个 div ,有几种方式得到这个 div 的 jQuery 对象?<div class='aabbcc'id='nodesView'> 想直接获取这个 div 的 dom 对象,如何获取?dom 对象如何转化为 jQuery 对象?

```
var domDiv1 = document.getElementById("nodesView");
var domDiv2 = document.getElementsByClassName("aabbcc")[0];
var domDiv3 = document.querrySelector("#nodesView");
var domDiv4 = document.querrySelector(".aabbcc");
转化为 jQuery 对象 $(domDiv);
```

116、如何显示/隐藏一个 dom 元素?请用原生的 JavaScript 方法实现

```
dom.style.display ="";面试题答案可以,测试有问题
dom.style.display = "none";
dom.style.visibility = "hidden";
dom.style.opacity = 0;
```

117、jQuery 框架中 $.ajax() 常用的参数有哪些

type	类型: String,默认值: GET.
url	类型: String,默认值:当前页地址,发送请求的地址
success	类型:Function,请求成功后的回调函数 参数:由服务器返回,并根据 dataType 参数进行处理后的数据,描述状态的字符串
options	类型:Object,可选,AJAX 请求设置,所有选项都是可选的
async	类型:Boolean,默认值:true,默认设置下所有请求均为异步请求 注意,同步请求将锁住浏览器,用户其他操作必须等待请求完成才可执行
beforeSend(XHR)	类型:Function,发送请求前可修改 XMLHttpRequest 对象的函数,如添加自定义 HTTP 头,XMLHttpRequest 对象是唯一的参数,这是一个 AJAX 事件,如果返回 false 可以取消本次 AJAX 请求
cache	类型:Boolean,默认值:true, dataType 为 script 和 jsonp 时默认为 false,设置为 false 将不缓存此页面
contentType	类型:String,默认值:"application/x-www-form-urlencoded",发送信息至服务器时内容编码类型.默认值适合大多数情况.如果你明确的传递了一个 content-type 给$.ajax()那么它必行会发送给服务器(即使没有数据要发送)
data	类型:String,发送到服务器的数据,将自动转换为请求字符串格式. GET 请求中将附加在 URL 后.查看 processData 选项说明以禁止此自动转换. 必须为 Key/Value 格式,如果为数组,jQuery 将自动为不同值对应同一个名称.

	如:{foo:["bar1", "bar2"]}转换为'&foo=bar1&foo=bar2'
dataFilter	类型:Function,给 AJAX 返回的原始数据的进行预处理的函数.提供 data 和 type 两个参数,data 是 AJAX 返回的原始数据,type 是调用 jQuery.ajax 时提供的 dataType 参数,函数返回的值将由 jQuery 进一步处理
dataType	类型:String,预期服务器返回的数据类型. 如果不指定,jQuery 将自动根据 HTTP 包 MIME 信息来智能判断,比如 XML MIME 类型就被识别为 XML.在 1.4 中,JSON 就会生成一个 JavaScript 对象,而 script 则会执行这个脚本.随后服务器端返回的数据会根据这个值解析后,传递给回调函数.可用值: "xml":返回 XML 文档,可用 jQuery 处理. "html":返回纯文本 HTML 信息;包含的 script 标签会在插入 dom 时执行. "script":返回纯文本 JavaScrip 代码.不会自动缓存结果.除非设置了"cache" 参数. 注意:在远程请求时(不在同一个域下),所有 POST 请求都将转为 GET 请求.(因为将使用 DOM 的 script 标签来加载) "json": 返回 JSON 数据. "jsonp": JSONP 格式.使用 JSONP 形式调用函数时,如 "myurl?callback=?"jQuery 将自动替换 ? 为正确的函数名,以执行回调函数. "text": 返回纯文本字符串
error	类型:Function,默认值:自动判断(xml 或 html), 请求失败时调用此函数

118、写一个 post 请求并带有发送数据和返回数据的样例

```
$.ajax({
    url: "1.html",
    data: {name: "zhangsan", age: 18}, // post 数据
    dataType: "json",
    type: "POST",
    success: function(data) {
        // data: 返回的数据
    },
    error: function() {
        // 异常处理
    }
});
```

119、JavaScript 的循环语句有哪些?

while do-while for for-in

120、作用域-编译期-执行期以及全局局部作用域问题

理解 js 执行主要的两个阶段:预解析和执行期

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.在 JavaScript 中,**作用域**有全局作用域和局部作用域两种(不存在块级作用域,即一对大括号之间的作用域).

(1)全局作用域,在代码中任何地方都能访问到的对象拥有全局作用域,例如:最外层函数和在最外层函数外面定义的变量,所有未定义直接赋值的变量,所有 window 对象的属性.

(2)局部作用域,在函数内部定义的变量.

JavaScript 编译执行特点:JavaScript 引擎,不是逐条解释执行 JavaScript 代码,
而是按照代码块一段段解释执行.所谓代码块就是使用<script>标签分隔的代码段.

当 JavaScript 引擎解析脚本时,它会在预编译期对所有声明的变量和函数进行处理,并且是先预声明变量,再预定义函数.**(编译期,预解析,变量提升)**

在解释过程中,JavaScript 引擎是严格按着作用域机制(scope)来执行的.JavaScript 语法采用的是词法作用域(lexcical scope),也就是说 JavaScript 的变量和函数作用域是在定义时决定的,而不是执行时决定的,由于词法作用域取决于源代码结构,所以 JavaScript 解释器只需要通过静态分析就能确定每个变量、函数的作用域,这种作用域也称为静态作用域(static scope).

JavaScript 引擎通过作用域链(scope chain)把多个嵌套的作用域串连在一起,并借助这个链条帮助 JavaScript 解释器检索变量的值.这个作用域链相当于一个索引表,并通过编号来存储它们的嵌套关系.当 JavaScript 解释器检索变量的值,会按着这个索引编号进行快速查找,直到找到全局对象(global object)为止,如果没有找到值,则传递一个特殊的 undefined 值.**(执行期)**

补充:如果函数引用了外部变量的值,则 JavaScript 引擎会为该函数创建一个闭包体(closure),闭包体是一个完全封闭和独立的作用域,它不会在函数调用完毕后就被 JavaScript 引擎当做垃圾进行回收.闭包体可以长期存在,因此开发人员常把闭包体当做内存中的蓄水池,专门用来长期保存变量的值.只有当闭包体的外部引用被全部设置为 null 值时,该闭包才会被回收.当然,也容易引发垃圾泛滥,甚至出现内存外溢的现象.

121、列出 3 条以上 ff 和 IE 的脚步兼容问题

(1) window.event:

表示当前的事件对象,IE 有这个对象,FF 没有,FF 通过给事件处理函数传递事件对象

(2) 获取事件源

IE 用 srcElement 获取事件源,而 FF 用 target 获取事件源

(3) 添加,移除事件

添加事件

IE:element.attachEvent("onclick", function)

FF:element.addEventListener("click", function, true)

移除事件

element.removeEventListener("click",function, true)

(4) 获取标签的自定义属性

IE:div1.value 或 div1["value"]

FF:可用 div1.getAttribute("value")

(5) document.getElementByName()和 document.all[name]

IE;document.getElementByName()和 document.all[name]均不能获取 div 元素

FF:可以

(6) input.type 的属性

IE:input.type 只读

FF:input.type 可读写

(7) innerTexttextContentouterHTML

IE:支持 innerText, outerHTML

FF:支持 textContent

(8) 是否可用 id 代替 HTML 元素

IE:可以用 id 来代替 HTML 元素

FF:不可以

122、用正则表达式,写出由字母开头,其余由数字、字母、下划线组成的 6~30 的字符串

varreg=/^[a-ZA-Z][\da-zA-Z_]{5,29}/;或者:varreg = /^[a-zA-Z]{1}[\w]{5, 29}/;

修饰符	描述
i	执行对大小写不敏感的匹配
g	执行全局匹配(查找所有匹配而非在找到第一个匹配后停止).

m	执行多行匹配
方括号	方括号用于查找某个范围内的字符
[abc]	查找方括号之间的任何字符.
[^abc]	查找任何不在方括号之间的字符.
元字符	元字符(Metacharacter)是拥有特殊含义的字符
.	查找单个字符,除了换行和行结束符
\w	查找单词字符等同于字符集合[a-zA-Z0-9_]
\W	查找非单词字符
\d	查找数字
\D	查找非数字
\s	查找空白字符
\S	查找非空字符
量词	描述
+	1 个或多个
*	0 个或多个
?	0 个或 1 个
{n}	n 个
{n,}	n 个到多个
{n, m}	n 个到 m 个
^	以什么开头
$	以什么结尾
/^aaa$/	以什么开头和结尾
方法	描述
test	检索字符串中指定的值.返回 true 或 false.
exec	检索字符串中指定的值.返回找到的值,并确定其位置.

123、写一个函数可以计算 sum(5, 0, -5),输出 0;sum(1,2,3,4),输出 10; arguments

```
functiongetSum() {
var result = 0;
    for (var i = 0; i <arguments.length; i ++) {
        result += arguments[i];
    };
    return result;
};
```

124、《正则》写出正确的正则表达式匹配固话号,区号 3-4 位,第一位为 0,中横线,7-8 位数字,中横线,3-4 位分机号格式的固话号

```
varreg = /^[0][0-9]{2,3}\-\[0-9]{7, 8}\-\[0-9]{3,4}$/;
var reg0 = /^[0]\d{2,3}\-\d{7,8}\-\d{3,4}$/;
```

125、《算法》菲波那切数列

A:农场买了一只羊,第一年是小羊,第二年底生一只,第三年不生,第四年底再生一只,第五年死掉.(找规律)羊的总数:1, 2, 2, 4, 3, 6, 5, 10, 8, 16, 13, 26, 21, 42, …奇数年为 1, 2, 3, 5, 8 Fib((n + 1) / 2) 偶数年为 2, 4, 6, 10, 16 Fib(n / 2) * 2

```
// 函数:获得第 n 年的斐波那契数,1,2,3,5,8,……忽略最初的 0
```

```
var getFib = function() {
    vararr = [1, 1];
    return function fn(n) {
        if(n <arr.length) {
            return arr[n];
        }else {
            arr.push(arr[arr.length- 1] + arr[arr.length- 2]);
            return fn(n);
        };
    };
}();
// 获得第 n 年的羊的总数
function countSheep(n) {
    var count;
    if(n % 2 === 0) {
        count = getFib(n / 2) *2;
    }else {
        count = getFib((n + 1) /2);
    };
    return count;
};
```

126、请写一个正则表达式:要求 6-20 位阿拉伯数和英文字母(不区分大小写)组成

/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[a-zA-Z\d]{6,20}$/

断言匹配(?=.*\d)表示字符串中有数字,(?=.*[a-z])(?=.*[A-Z])则分别表示字符串中有小写字母和大写字母,[a-zA-Z\d]{6,20}表示字母和数字有 6 到 20 位

127、统计 1 到 400 之间的自然数中含有多少个 1?如 1-21 中自然数有 13 个 1

```
var count = 0;                              //定义初始个数
for(var i = 1; i <= 400; i++) {             //从 1 到 400
    for(var j = 0; j < i.toString().length; j++) {  //把每个数拆分成单个数字,如 123 拆分成"1","2","3"
        if(i.toString()[j] == "1")          //如果包含 1
            count++;                        //统计 1 的个数
    }
}
console.log(count); //180
```

127、删除与某个字符相邻且相同的字符,比如"fdaffdaaklfjklja"==>"fdafdaklfjklja"

```
var str="fdafffdaaklfjklja";
var arr=[];
for (var i = 0; i <str.length; i++) {
    arr[i]=str[i]; //考虑到字符串的不可操作性,没有也能实现
};
for (var i = 0; i <arr.length; i++) {
    if(arr[i]===arr[i+1]){
```

```
            arr.splice(i,1);
            i--;
        }
};
str=arr.join("");
console.log(str);
```

128、请写出三种以上的 Firefox 有但 InternetExplorer 没有的属性和函数

1、得到 IFRAME 对象,IE: document.frames["id"]; FF:document.getElementById("content_panel_if").contentWindow;

2、_tbody=_table.childNodes[0];在 FF 中,firefox 会在子节点中包含空白则第一个子节点为空白"", 而 ie 不会返回空白文本节点.

3、模拟点击事件

```
if(document.all){   //ie 下
        document.getElementById("a3").click();
} else{   //非 IE
        var evt = document.createEvent("MouseEvents");
        evt.initEvent("click", true, true);
        document.getElementById("a3").dispatchEvent(evt);
}
```

4、 事件注册 if (isIE){window.attachEvent("onload", init);} else{window.addEventListener("load", init, false);}

129、请写出一个程序,在页面加载完成后动态创建一个 form 表单,并在里面添加一个 input 对象并给它任意赋值后义 post 方式提交到:http://127.0.0.1/save.php

```
// JavaScript 构建一个 form
function MakeForm() {
        // 创建一个 form
        var form1 = document.createElement("form");
        form1.id = "form1";
        form1.name = "form1";
        // 添加到 body 中
        document.body.appendChild(form1);
        // 创建一个输入
        var input = document.createElement("input");
        // 设置相应参数
        input.type = "text";
        input.name = "value1";
        input.value = "1234567";
        // 将该输入框插入到 form 中
        form1.appendChild(input);
        // form 的提交方式
        form1.method = "POST";
        // form 提交路径
        form1.action = "http://127.0.0.1/save.php ";
        // 对该 form 执行提交
        form1.submit();
        // 删除该 form
```

```
    document.body.removeChild(form1);
}
MakeForm();
```

130、简述一下什么叫事件委托以及其原理

使用事件委托技术能让你避免对特定的每个节点添加事件监听器;相反,事件监听器是被添加到它们的父元素上.事件监听器会分析从子元素冒泡上来的事件,找到是哪个子元素的事件.

假设,当每个子元素被点击时,将会有各自不同的事件发生.你可以给每个独立的 li 元素添加事件监听器,但有时这些 li 元素可能会被删除,可能会有新增,监听它们的新增或删除事件将会是一场噩梦,尤其是当你的监听事件的代码放在应用的另一个地方时.此时采用事件委托的方式的话,当子元素的事件冒泡到父 ul 元素时,你可以检查事件对象的 target 属性,捕获真正被点击的节点元素的引用.

131、下列 JavaScript 代码执行后,Num 的值是

```
var iNum = 0;
for(var i = 1; i<10; i++){
    if(i % 5 == 0){
        continue;
    }
    iNum++;
}
console.log(iNum);    //1-9 共 9 个,除去 5,八个数字,相当于八次++,结果为 8
```

132、输出结果是多少?

```
var a;                  //undefined
var b = a * 0;          //NaN
if (b == b) {
    //if(b==NaN)结果为 false [NaN 特殊的是自己不等于自己]
    console.log(b * 2 + "2"- 0 + 4);//true
} else {
    console.log(!b * 2 + "2"- 0 + 4);//false;
    // 1*2+"2"+4=22+4=26
}
2）
var a;                  //重复的 var 声明无效
var b = a * 0;
if (b == b) {           //b=0
    console.log(b * 2 + "2"- 0 + 4);//6
} else {
    console.log(!b * 2 + "2"- 0 + 4);
}
3）
<script>
    var a = 1;
</script>
<script>
```

```
    var a;
    var b = a / 0;
    if (b == b) {    //b=Infinity
        console.log(b * 2 + "2"+ 4);//Infinity24
    } else {
        console.log(!b * 2 + "2"+ 4);
    }
</script>
4)
var t = 10;
function test(t){
    var t = t++;//此时的 t 是一个局部变量
    console.log(t);//10
}
test(t);
console.log(t);//10       //函数内传入的 t 的值改变,不会影响到外面定义的 t 的值(基本数据类型)
5)
var t = 10;
function test(test){
    var t = test++;
}
test(t);
console.log(t);//10
6)
var t = 10;
function test(test){
    t = test++;
}
test(t);
console.log(t);//10
7)
var t = 10;
function test(test){
    t = t + test;
    console.log(t);//undefined+10=NaN
    var t = 3;
}
test(t);
console.log(t);//10
```

133、下列 JavaScript 代码执行后,运行的结果是

```
<body>
<button id='btn'>点击我</button>
</body>
```

```
<script>
    var btn = document.getElementById('btn');
    var handler = {
        id: '_eventHandler',
        exec: function(){
            alert(this.id);//btn;    // 因为 handler.exec 是由 btn 这个按钮执行的
        }
    }
    btn.addEventListener('click', handler.exec);
</script>
```

134、下列 JavaScript 代码执行后,依次 alert 的结果是

```
var obj = {proto: {a:1,b:2}};
function F(){};
F.prototype = obj.proto;
var f = new F();
obj.proto.c = 3;
console.log(F.prototype == obj.proto);          //true
obj.proto = {a:-1, b:-2};                       //字面量重新开辟空间
console.log(F.prototype == obj.proto);          //false
console.log(f.a);                               //1
console.log(f.c);                               //3
delete F.prototype['a'];
console.log(f.a);                               //undefined
console.log(obj.proto.a);                       //-1
```

135、下列 JavaScript 代码执行后的效果是

```
<ul id='list'>
<li>item</li>
<li>item</li>
<li>item</li>
<li>item</li>
<li>item</li>
</ul>
var items = document.querySelectorAll('#list>li');
for(var i = 0;i < items.length; i++){
    setTimeout(function(){
        items[i].style.backgroundColor = '#fee';
    }, 5);
}
```

答案:异常.因为每次循环就执行定时器,但定时器还没计时完毕,就又进行下一次循环,即重新计时.真正执行定时器内的函数是的 i=items.length,所以异常.

136、下列 JavaScript 代码执行后的 li 元素的数量是

```
<ul>
```

```
<li>Item</li>
<li></li>
<li></li>
<li>Item</li>
<li>Item</li>
</ul>
var items = document.getElementsByTagName('li');
for(var i = 0; i< items.length; i++){
    if(items[i].innerHTML == ''){
        items[i].parentNode.removeChild(items[i]);
    }
}
```

答案:4 个 .因为第三个 li 在第二个 li 被清除后,占据的是第二个 li 的位置,所以它被跳过了遍历.

137、程序中捕获异常的方法?

window.error

try{}catch(){}finally{}

138、将字符串"<tr><td>{$id}</td><td>{$name}</td></tr>"中的{$id}替换成 10,{$name}替换成 Tony (使用正则表达式)

```
"<tr><td>{$id}</td><td>{$id}{$name}</td></tr>".replace(/{\$id}/g,'10').replace(/{\$name}/g,'Tony');
```

139、给 String 对象添加一个方法,传入一个 string 类型的参数,然后将 string 的每个字符间间隔空格返回,例如:

```
var str="hello world"; // -> 'h e l l o?w o r l d'
String.prototype.spacify = function(){
    return this.split('').join(' ').replace(/\s{3}/,"?");
};
str.spacify();
```

140、写出函数 DateDemo 的返回结果,系统时间假定为今天

```
function DateDemo(){
    var d, s="今天日期是:";
    var arr=[];
    d = new Date();
    arr[0]=d.getMonth();
    arr[1]=d.getDate();
    arr[2]=d.getFullYear();
    s +=arr.join("/");
    return s;           //今天日期是:月份少 1/日/年
}
```

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

```
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);
```

141、var numberArray=[3,6,2,4,1,5]; (考察基础 API)

实现对该数组的倒排,输出[5,1,4,2,6,3]　　numberArray.reverse();

实现对该数组的降序排列,输出[6,5,4,3,2,1]

```
var newArr=numberArray.sort(function (a,b) {
    return b-a;
});
```

142、把两个数组合并,并删除第二个元素

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

143、写一个 function,清除字符串前后的空格.(兼容所有浏览器)

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

144、数组和字符串

```
var a = "lashou";
var b = a;
outPut(b);                  //lashou
a = "拉手";
outPut(a);                  //拉手
outPut(b);                  //lashou
var a_array = [1, 2, 3];
var b_array = a_array;
outPut(b_array);            //[1,2,3]
a_array[3] = 4;
outPut(a_array);            //[1,2,3,4]
```

```
outPut(b_array);                    //[1,2,3,4]
```

145、下列控制台都输出什么

```
//1
function setName() {
    name = "张三";
}
setName();
console.log(name);                  //"张三"
//2       考察 1、变量声明提升  2、变量搜索机制
var a = 1;
function test() {
    console.log(a);
    var a = 1;
}
test();                             //undefined
//3
var b = 2;
function test2() {
    window.b = 3;
    console.log(b);
}
test2();                            //3
//4
c = 5;//声明一个全局变量  c
function test3() {
    window.c = 3;
    console.log(c);                 //答案:undefined
    var c;
    console.log(window.c);          //答案:3,原因:这里的  c  就是一个全局变量  c
}
test3();
//5
var arr = [];
arr[0] = 'a';
arr[1] = 'b';
arr[10] = 'c';
alert(arr.length);                  //答案:11
console.log(arr[5]);                //答案:undefined
//6
var a = 1;
console.log(a++);                   //答案:1
console.log(++a);                   //答案:3
//7
```

```
console.log(null == undefined);    //答案:true
console.log("1" == 1);             //答案:true,因为会将数字 1 先转换为字符串 1
console.log("1" === 1);            //答案:false,因为数据类型不一致
//8
typeof 1;                          //"number"
typeof "hello";                    //"string"
typeof /[0-9]/;                    //"object"
typeof {};                         //"object"
typeof null;                       //"object"
typeof undefined;                  //"undefined"
typeof [1, 2, 3];                  //"object"
typeof function(){};               //"function"
//9
parseInt(3.14);                    //3
parseFloat("3asdf");               //3
parseInt("1.23abc456");            //1
parseInt(true);                    // NaN
//10      考点:函数声明提前
function bar() {
    return foo;
    foo = 10;
    function foo() {}
    //var foo = 11;
}
alert(typeof bar());               //"function"
//11      考点:函数声明提前
var foo = 1;
function bar() {
    foo = 10;
    return;
    function foo() {}
}
bar();
alert(foo);                        //答案:1
//12
console.log(a);//function a(){}
var a = 3;
function a() {};
console.log(a);                    //3
//13      考点:对 arguments 的操作
function foo(a) {
    arguments[0] = 2;
    alert(a);                      //答案:2,
}
```

```
foo(1);                          //因为:a、arguments 是对实参的访问,b、通过 arguments[i]可以修改指定实参的值
//14
function foo(a) {
    alert(arguments.length);     //答案:3,因为 arguments 是对实参的访问
}
foo(1, 2, 3);
//15
bar();                           //报错
var foo = function bar(name) {
    console.log("hello" + name);
    console.log(bar);
};
//  foo("world");//helloworld//function bar(name){.....};
//  console.log(bar);//报错
console.log(foo.toString());     //function bar(name){.....};
bar();                           //报错
//16
function test() {
    console.log("test 函数");
}
setTimeout(function () {
    console.log("定时器回调函数");
}, 0)
test();     //test 函数 //定时器回调函数
```

四、JS 高级

1、知道什么是 webkit 么?知道怎么用浏览器的各种工具来调试和 debug 代码么?

Webkit 是浏览器引擎,包括 html 渲染和 js 解析功能,手机浏览器的主流内核,与之相对应的引擎有 Gecko(Mozilla Firefox 等使用)和 Trident(也称 MSHTML,IE 使用) .

对于浏览器的调试工具要熟练使用,主要是页面结构分析 Elements 项,后台请求信息查看 Network,逐步调试和逐语句调试 Sources,控制台输出信息 Console,安装其他插件如 AngularJS 帮助 ng 项目调试等.熟练使用这些工具可以快速提高解决问题的效率

2、如何测试前端代码?知道 BDD, TDD, Unit Test 么?知道怎么测试你的前端工程么 (mocha, sinon, jasmin, qUnit..)?开发模式都是什么意思:

TDD:测试驱动开发(Test-Driven Development)

测试驱动开发是敏捷开发中的一项核心实践和技术,也是一种设计方法论.TDD 的原理是在开发功能代码之前,先编写单元测试用例代码,测试代码确定需要编写什么产品代码.TDD 的基本思路就是通过测试来推动整个开发的进行,但测试驱动开发并不只是单纯的测试工作,而是把需求分析,设计,质量控制量化的过程.TDD 首先考虑使用需求(对象、功能、过程、接口等),主要是编写测试用例框架对功能的过程和接口进行设计,而测试框架可以持续进行验证.

BDD:行为驱动开发(Behavior Driven Development)

行为驱动开发是一种敏捷软件开发的技术,它鼓励软件项目中的开发者、QA 和非技术人员或商业参与者之间的协作.主要是从用户的需求出发,强调系统行为.BDD 最初是由 Dan North 在 2003 年命名,它包括验收测试和客户测试驱动等的极限编程的实践,作为对测试驱动开发的回应.使用 BDD 可以解决需求和开发脱节的问题,首先他们都是从用户的需求出发,保证程序实现效果与用户需求一致.

以上两种是开发模式,个人认为在代码编写过程常有按照 TDD 模式对代码进行小单元测试,以保证逻辑正确,而在整体上看按照 BDD 模式测试,保证开发和需求不发生脱节,但其实两种模式的框架都有. Unit Test 单元测试, 测试必须随生产代码的演进而修改,测试越脏就越难修改,所以为了保证其质量和可维护性,测试代码也常使用框架,常见的测试框架如下:

Jasmine:BDD 框架,

不依赖于任何框架,所以适用于所有的 Javascript 代码.使用一个全局函数 describe 来描述每个测试,并且可以嵌套.describe 函数有 2 个参数,一个是字符串用于描述,一个是函数用于测试.在该函数中可以使用全局函数 it 来定义 Specs,也就是单元测试的主要内容, 使用 expect 函数来测试.另外如果想去掉某个 describe,无须注释掉整段代码,只需要在 describe 前面加上 x 即可忽略该 describe.toBe 方法是一个基本的 matcher(匹配器) 用来定义判断规则;时间控制上, 提供了 Clock 方法来模拟时间,以获取 setTimeout 的不同状态;由于异步需要等待回调, 提供了 runs 和 waitsFor 两个方法来完成这个异步的等待

Qunit

jQuery 团队开发的一款测试套件.Qunit 采用断言(Assert)来进行测试,相比于 Jasmine 的 matcher 更加多的类型,Qunit 更集中在测试的度上.deepEqual 用于比较一些纵向数据,比如 Object 或者 Function 等.而最常用的 ok 则直接判断是否为 true.异步方面 Qunit 也很有趣,通过 stop 来停止测试等待异步返回,然后使用 start 继续测试,这要比 Jasmine 的过程化的等待更自由一些,不过有时也许会更难写一些.Qunit 还拥有 3 组 AOP 的方法(done 和 'begin')来对应于整个测试,测试和模块.

Sinon

Sinon 并不是一个典型的单元测试框架,更像一个库,最主要的是对 Function 的测试,包括 Spy 和 Stub 两个部分,Spy 用于侦测 Function,而 Stub 更像是一个 Spy 的插件或者助手,在 Function 调用前后做一些特殊的处理,比如修改配置或者回调.它正好极大的弥补了 Qunit 的不足,所以通常会使用 Qunit+Sinon 来进行单元测试.

值得一提的是,Sinon 的作者 Christian Johansen 就是 *Test-Driven JavaScript Development* 一书的作者,这本书针对 Javascript 很详细的描述了单元测试的每个环节.

Mocha

Mocha 将更多的方法集中在了 describe 和 it 中,比如异步的测试就非常棒,在 it 的回调函数中会获取一个参数 done ,类型是 function,用于异步回调,当执行这个函数时就会继续测试.还可以使用 only 和 skip 去选择测试时需要的部分.Mocha 的接口也一样自由,除了 BDD 风格和 Jasmine 类似的接口,还有 TDD 风格的 (suite test setup teardown suiteSetup suiteTeardown),还有 AMD 风格的 exports,Qunit 风格等.同时测试报告也可以任意组织,无论是列表、进度条、还是飞机跑道这样奇特的样式都可以在 bash 中显示.

3、前端 templating(Mustache, underscore, handlebars) 是干嘛的, 怎么用?

Web 开发的模板引擎是为了使用户界面与业务数据(内容)分离而产生的,它可以生成特定格式的文档,用于网站的模板引擎就会生成一个标准的 HTML 文档.

Mustache 是一个 logic-less (轻逻辑)模板解析引擎,它的优势在于可以应用在 Javascript、PHP、Python、Perl 等多种编程语言中. 需要 data 数据,定义的模板 tpl,然后使用 Mustache.render(tpl, data)方法是用于渲染输出最终的 HTML 代码,在模板中可以使用插值语法来获取数据中的内容

Underscore 封装了常用的 JavaScript 对象操作方法,用于提高开发效率. 正如 jQuery 统一了不同浏览器之间的 DOM 操作的差异,让我们可以简单地对 DOM 进行操作,underscore 则提供了一套完善的函数式编程的接口,让我们更方便地在 JavaScript 中实现函数式编程.同 jq 中的$一样,underscore 会把自身绑定到唯一的全局变量_上,这也是为啥它的名字叫 underscore 的原因(这个我们用过,和我们使用的 template-native 相似,语法上有些区别,如 each 方法需要将 function 当参数传入,和 template-native 的 for 循环有所区别,但是实现效果相同)

Handlebars 是 JavaScript 一个语义模板库,通过对 view 和 data 的分离来快速构建,handlebarsjs 是模板库,模板库的主要作用是:你想要生成某一大片界面,这一片界面有一定规律,比如商品详情,不同商品之间差的只是名称,价格,图片,介绍这些,但是结构一样的,那我们就可以给他预先写个界面模板,里面凡是有可能变的地方,用变量代替,然后每次拿不同的数据代入,生成最终的结果 HTML.一般的模板库都是静态模板.扩展:和 ng 库对比.angularjs 是一种框架,它包含的东西很多,其中,模板这一块也有类似的东西,但它的界面模板不仅仅是模板本身,还包括一些配置,这些配置能被特定的方式解析,从而与数据层进行动态关联.所以这是动态模板.

artTemplate 是比较新一些的前端模板引擎,库分为两种,一个是 template.js,一个是 template-native.js,第一个是简洁语法版

(使用插值语法{{}}),第二个是原生语法版(使用<%%>),两个库的语法是不一样的,大家不要混用,否则会报错的.在 js 中 template("DOM 的 id",数据 data)来连接 js 中的数据和 html 模板

4、简述一下 Handlebars 的基本用法?

Handlebars 是 JavaScript 一个语义模板库,通过对 view 和 data 的分离来快速构建 Web 模板.它采用"Logic-less template"(无逻辑模版)的思路,在加载时被预编译,而不是到了客户端执行到代码时再去编译, 这样可以保证模板加载和运行的速度.Handlebars 兼容 Mustache,你可以在 Handlebars 中导入 Mustache 模板.

表达式是基本单元,使用插值语法,加两个花括号{{value}}, handlebars 模板会自动匹配相应的数值,对象甚至是函数.块表达式可以实现表达式深入操作,有一些内置表达式块,如 each、if、unless(反向 if)、with 等,表达式和 html 对标签一样,这里也要前后对应,如{{#if list}}{{/if}}

{{#with}}一般情况下,Handlebars 模板会在编译的阶段的时候进行 context 传递和赋值.使用 with 的方法,我们可以将 context 转移到数据的一个 section 里面(如果你的数据包含 section). 这个方法在操作复杂的 template 时候非常有用

5、简述一下 Handlerbars 的对模板的基本处理流程, 如何编译的?如何缓存的?

1、工作原理 —— 拼字符串成为代码

2、流程 —— 访问时,转换模板代码为目标源码执行,或者第一次运行时转为目标源码(缓存),后续直接调用源码运行.第一次运行时执行模板->源码拼接,然后将拼接后源码缓存,之后执行模板不再进行重复拼接工作.所以,不管怎么样模板转换与目标代码执行肯定要消耗资源(只是消耗多点少点问题),肯定会比不用模板直接输出 HTML 来的有压力.

3、依赖性—— 访从 1 原理可知,哪方实现的模板引擎,就依赖哪方.而我们前端模板引擎是在客户端实现一种模板解析方式(引擎),用来读取模板内容,分析并转为客户端可执行的程序源码,并运行,实现脱离服务端,在浏览器端渲染页面,而不依赖服务端.但其实一般使用半前置,解决部分服务器压力,也至于过度影响 SEO(模板引擎不支持 seo)

6、用 js 实现千位分隔符?(断言匹配)

实现方法有很多种,可以用三位循环、字符串数组分隔,也可以使用正则

```
function thousandBitSeparator(num) {
    //短路运算判断是否传参
    return num && (num.toString().indexOf('.') != -1 ?
                        //正则表达式断言匹配,只匹配一个位置(?=xxx)表示只匹配符合 xxx 的
                        //replace 的 function 参数$+数字,表示匹配的子表达式
                        //如果转成的字符串后不包含.则匹配的时候加个.
                        num.toString().replace(/(\d)(?=(\d{3})+\.)/g, function($0, $1) {
                            return $1 + ",";
                        }) :
                        num.toString().replace(/(\d)(?=(\d{3}))/g, function($0, $1) {
                            return $1 + ",";
                        }));
}
```

7、检测浏览器版本版本有哪些方式?

```
/简易判断 IE8-浏览器方法
//原理是利用处理数组的 toString 方法的差异,非 IE8-浏览器在数组最后一个字符为逗号时剔除它.
var ie8 = !-[1,];       //非 ie8-,!-[1] --> !-1 --> false,除了那几个都是真,取反就是 false
alert(ie8);


//只判断浏览器,不判断版本,或区分 ie 版本
```

```
function myBrowser(){
    var userAgent = navigator.userAgent; //取得浏览器的 userAgent 字符串
    if (userAgent.indexOf("Opera") > -1) {
        return "Opera"
    }; //判断是否 Opera 浏览器
    if (userAgent.indexOf("Firefox") > -1) {
        return "FF";
    } //判断是否 Firefox 浏览器
    if (userAgent.indexOf("Chrome") > -1){
        return "Chrome";
    } //判断是否 Chrome 浏览器
    if (userAgent.indexOf("Safari") > -1) {
        return "Safari";
    } //判断是否 Safari 浏览器
    if (userAgent.indexOf("compatible") > -1 && userAgent.indexOf("MSIE") > -1 && userAgent.indexOf("Opera") == -1) {
        return "IE";
        //如果需要区分 ie 版本
        var IE5 = IE55 = IE6 = IE7 = IE8 = false;
        var reIE = new RegExp("MSIE (\\d+\\.\\d+);");
        reIE.test(userAgent);           //用正则表达式匹配 userAgent 字符串
        var fIEVersion = parseFloat(RegExp["$1"]);
        IE55 = fIEVersion == 5.5;
        IE6 = fIEVersion == 6.0;
        IE7 = fIEVersion == 7.0;
        IE8 = fIEVersion == 8.0;
        if (IE55) return "IE55";
        if (IE6)    return "IE6";
        if (IE7)    return "IE7";
        if (IE8)    return "IE8";
    } //判断是否 IE 浏览器
}
```

8、给一个 dom 同时绑定两个点击事件,一个用捕获,一个用冒泡,会执行几次事件,然后会先执行冒泡还是捕获,对两种事件模型的理解

会执行两次,既然说到使用冒泡和捕获,自然是 DOM2 标准中的 addEventListener 绑定事件,通过参数 3 来设定,true 表示捕获,flase 表示冒泡(默认).事件分为捕获、

9、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)进行值复制

```
function clone(Obj) {
    var buf;                        //Array 和 Obj 是引用类型,直接赋值只是复制指针,拷贝要遍历执行
    if (Obj instanceof Array) {
        buf = [];                   //创建一个空的数组
        var i = Obj.length;
        while (i--) {               //由于可能存在嵌套,需要递归执行
```

```
            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) {        //for in 遍历属性来拷贝
            buf[k] = clone(Obj[k]);
        }
        return buf;
    }else{                          //普通变量直接赋值
        return Obj;
    }
}
```

10、如何消除一个数组里面重复的元素?

```
function deRepeat(arr){
    var newArr=[];                  /*定义新数组存储不重复元素*/
    var obj={};                     /*借用 obj 键值对方式来表示元素是否存在*/
    var i,l=arr.length;             /*长度缓存,不用每次循环都先获取再判断*/
    for(i=0;i<l;i++){
        if(obj[arr[i]]==undefined){     /*问题:[]不会区分 1 和'1',有 bug*/
            obj[arr[i]]=1;          /*如果在临时 obj 中不存在对应减,则传入键值对*/
            newArr.push(arr[i]);
        }else if(obj[arr[i]]==1)
            continue;               /*如果已经存在于 obj 中,则不处理*/
    }
    return newArr;
}
function deRepeat(arr) {
    for(var i=0; i<arr.length; i++){
        for(var j=i+1; j<arr.length; j++){
            if(arr[i] === arr[j]){      //这里要用===,否则数字 1 和字符串 1 会判定相同而删除
                arr.splice(j,1);
                j--;                    //length 属性动态改变,去掉一个元素后,索引也要改变
            }
        }
    }
    return arr;
}
```

12、小贤是小狗 (Dog) ,叫声很好听 (wow) ,每次看到主人的时候就会乖乖叫一声 (yelp) .从这段描述可以得到以下对象:

```
function Dog() {
    this.wow = function() {     /*叫声是特征,设为属性*/
        console.log('Wow');
```

```
    }
}
Dog.prototype.yelp = function() {/*见到主人触发叫,设为行为*/
    this.wow();
}
```

小芒也是小狗,有一天疯了(MadDog),一看到人就会每隔半秒叫一声(wow)地不停叫唤(yelp).(继承,原型,setInterval)

```
function MadDog() {}
MadDog.prototype = new Dog();     //原型继承
MadDog.prototype.yelp = function() {
    var _this = this;                    //在定时器中 this 指向发生改变,要缓存
    setInterval(function() {
        _this.wow();
    }, 500);
}
```

13、下面这个 ul ,如何点击每一 li 的时候 alert 对应 index? (闭包)

```
// 方法一:自定义属性绑定 index
var lis=document.getElementById('test').getElementsByTagName('li');
for(var i=0;i<lis.length;i++){
    lis[i].index=i;
    lis[i].onclick=function(){
        alert(this.index);
    };
}
//方法二:闭包缓存 index
var lis=document.getElementById('test').getElementsByTagName('li');
for(var i=0;i<lis.length;i++){
    lis[i].onclick=(function(index){
        return function() {
            alert(index);
        }
    })(i);              /*闭包传入 i*/
}
```

14、编写一个 JavaScript 函数, 输入指定类型的选择器((仅需支持 id, class, tagName 三种简单 CSS 选择器,无需兼容组合选择器))可以返回匹配的 DOM 节点,需考虑浏览器兼容性和性能.

```
var select = (function () {
    //简单选择器
    function getById(idName, context) {
        var dom = document.getElementById(idName);
        if (!context) {
            return dom;
```

```
        } else {
            return checkParent(dom, context) ? dom : null;
        }
    }
    //递归判断元素是否拥有指定父级
    function checkParent(dom, parent) {
        if (dom.parentNode === parent) return true;
        if (!dom.parentNode) return false;
        return checkParent(dom.parentNode, parent);
    }
    function getByTag(tagName, context) {
        return context.getElementsByTagName(tagName);
    }
    function getByClass(className, context) {
        //能力检测
        if (document.getElementsByClassName) {
            return context.getElementsByClassName(className)
        } else {
            var doms = document.getElementsByTagName("*");
            var rst = [];
            className = " " + className + " ";
            for (var i = 0; i < doms.length; i++) {
                var dom = doms[i]
                var classString = " " + dom.className + " ";
                //此时判断避免了字符串分割,遍历结果数组
                if (classString.indexOf(className) != -1) {
                    rst.push(dom);
                }
            }
            return rst;
        }
    }
    //基础三合一选择器
    return function get(selector, context) {
        var rst = [];
        //参数保护
        if (!context) {
            context = [document];
        } else if (context.nodeType) {
            context = [context];
        } else if (typeof context === "string") {
            context = get(context);
        }
        //配合正则表达式,遍历执行方法调用
```

```
        var reg = /^(?:#(\w+)|[.](\w+)|(\w+))$/;        //不要有空格
        for (var i = 0; i < context.length; i++) {
            var parent = context[i];
            var regRst = reg.exec(selector);
            var tempRst;
            if (tempRst = regRst[1]) {
                var dom = getById(tempRst, parent);
                //为了避免结果数组中存在 null
                if (dom) {
                    rst.push(dom);
                }
            } else if (tempRst = regRst[2]) {
                rst.push.apply(rst, getByClass(tempRst, parent))
            } else if (tempRst = regRst[3]) {
                rst.push.apply(rst, getByTag(tempRst, parent))
            }
        }
        return rst;
    }
})();
```

15、请评价以下代码并给出改进意见.

```
if(window.addEventListener){          //不需要使用 window.addEventListener 或 document.all  来进行检测浏览器,应该使用能力检测;
    var addListener = function(el,type,listener,useCapture){   //不应该在 if 和 else 语句中声明  addListener  函数,应该先声明;
        el.addEventListener(type,listener,useCapture);
    };
}else if(document.all){
    addListener = function(el,type,listener){
        el.attachEvent("on"+type,function(){
            listener.apply(el);                                 //由于 attachEvent 在 IE 中有  this  指向问题,调用它时需要处理一下
        });
    }
}
```

改进如下:

```
function addEvent(elem, type, handler){
    if(elem.addEventListener){                                  //能力检测
        elem.addEventListener(type, handler, false);
    }else if(elem.attachEvent){
        elem['temp' + type + handler] = handler;
        elem[type + handler] = function(){
            elem['temp' + type + handler].apply(elem);      //apply 改变上下文
        };
        elem.attachEvent('on' + type, elem[type + handler]);
    }else{
```

```
            elem['on' + type] = handler;                    //都不满足直接使用 DOM0 方法绑定
        }
    }
```

16、给 String 对象添加一个方法,传入 string 类型的参数,然后将 string 的每个字符间加个空格返回,例如:

```
String.prototype.spacify = function(){
    //先分割成数组,然后用空格拼接
    return this.split('').join(' ');
};
```

1) 直接在对象的原型上添加方法是否安全?尤其是在 Object 对象

不安全,存在以下情况

容易造成全局污染,有可能和其他库冲突(考虑现在)

有可能出现代码向上不兼容的情况,有可能和新语法冲突(考虑未来)

不利于项目协作开发,易冲突,出了 Bug 不太好定位问题(考虑开发过程)

2) 函数声明与函数表达式的区别?

在 js 中,解析器在向执行环境中加载数据时,会先读取函数声明(预解析的函数提升),并使其在执行任何代码之前可用(可以访问),至于函数表达式,则必须等到解析器执行到它所在的代码行,才会真正被解析执行

函数声明:function func(){}　　函数表达式:var foo = function(){};

而且要求,凡是函数的声明,不允许出现在代码的逻辑结构中或表达式中,函数声明必须独立存在(特例:在 if 语句中执行的 function xxx(){}会被当做表达式)

17、定义一个 lo 方法,让它可以代理 console.log 的方法.

```
//简单的方法:和 jQuery 中的 error 一样,调用系统 api
function log(msg) {
    console.log(msg);//只能打印第一个参数
}
//更好的方法,使用上下文,可以传多个参数
function log(){
    console.log.apply(console, arguments);
};
```

追问:apply 和 call 方法的异同

call 和 apply 都是为了改变某个函数运行时的 context 即上下文而存在的,换句话说,就是为了改变函数体内部 this 的指向.因为 JavaScript 的函数存在「定义时上下文」和「运行时上下文」以及「上下文是可以改变的」这样的概念.

二者的作用完全一样,只是接受参数的方式不太一样,参数 1 是 this ,all 需要把参数按顺序传递进去,而 apply 则是把参数放在数组里

18、在 Javascript 中什么是伪数组?如何将伪数组转化为标准数组?

伪数组(类数组):无法直接调用数组方法或期望 length 属性有什么特殊的行为,但仍可以对真正数组遍历方法来遍历它们.典型的是函数的 argument 参数,还有像调用 getElementsByTagName,document.childNodes 之类的,它们都返回 NodeList 对象都属于伪数组. 可以使用 Array.prototype.slice.call(likeArray)将数组转化为真正的 Array 对象.

```
function list() {
   return Array.prototype.slice.call(arguments);
}
```

19、对作用域上下文和 this 的理解,看下列代码:

```
var User = {
    count: 1,
    getCount: function() {
        return this.count;
    }
};
console.log(User.getCount()); //1
var func = User.getCount;
console.log(func());              //undefined,func 是在 winodw 的上下文中被执行的,所以会访问不到 count 属性.
//为了能确保 Uesr 总是能访问到 func 的上下文,即正确返回 1
//兼容写法,bind 也是用于改变执行上下文的方法,只是返回的是个函数,而不是调用方法
Function.prototype.bind = Function.prototype.bind || function(context){
    var self = this;
    return function(){
        return self.apply(context, arguments);
    };
}
//通过 bind 方法改变执行上下文,此时就能访问
var func = User.getCount.bind(User);
console.log(func());
```

20、原生 JS 的 window.onload 与 Jquery 的 $(document).ready(function(){})有什么不同?如何用原生 JS ready 方法?

首先理解,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.

```
/*
 * 传递函数给 whenReady()
 * 当文档解析完毕且为操作准备就绪时,函数作为 document 的方法调用
 */
var whenReady = (function () {
    var funcs = [];              //当获得事件时,要运行的函数
    var ready = false;           //当触发事件处理程序时,切换为 true,当文档就绪时,调用事件处理程序
    function handler(e) {
        if (ready) return; //确保事件处理程序只完整运行一次
        //如果发生 onreadystatechange 事件, 但其状态不是 complete 的话,那么文档尚未准备好
        if (e.type === 'onreadystatechange' && document.readyState !=='complete') {
            return;
        }
        //运行所有注册函数,注意每次都要计算 funcs.length 以防这些函数的调用可能会导致注册更多的函数
        for (var i = 0; i < funcs.length; i++) {
            funcs[i].call(document);
        }
        //事件处理函数完整执行,切换 ready 状态, 并移除所有函数
```

```
            ready = true;
            funcs = null;
        }
        //为接收到的任何事件注册处理程序
        if (document.addEventListener) {
            document.addEventListener('DOMContentLoaded', handler, false);
            document.addEventListener('readystatechange', handler, false); //IE9+
            window.addEventListener('load', handler, false);
        } else if (document.attachEvent) {
            document.attachEvent('onreadystatechange', handler);
            window.attachEvent('onload', handler);
        }
        //返回  whenReady()函数,闭包
        return function whenReady(fn) {
            if (ready) {
                fn.call(document);
            }
            else {
                funcs.push(fn);
            }
        }
})();
//简化版:
function ready(fn) {
    if (document.addEventListener) {       //DOM 浏览器
        document.addEventListener('DOMContentLoaded', function () {
            //注销事件, 避免反复触发
            document.removeEventListener('DOMContentLoaded', arguments.callee, false);
            fn();
        }, false);
    } else if (document.attachEvent) {    //IE
        document.attachEvent('onreadystatechange', function () {
            if (document.readyState == 'complete') {
                document.detachEvent('onreadystatechange', arguments.callee);
                fn();
            }
        });
    }
};
```

21、(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 S JS)回答出概念即可,下面是几个要点

扩展:这里如果想加一个磁性吸附效果的话,可以在边界判断条件基础上加一定值,就可以实现吸附效果

```
function drag(obj) {
```

```
    //1、mousedown 事件触发后,开始拖拽,加给拖拽对象
    obj.onmousedown = function (ev) {
        var oEvent = ev || event;
        var disX = oEvent.clientX - obj.offsetLeft;
        var disY = oEvent.clientY - obj.offsetTop;
        //2、mousemove 时,需要通过 event.clientX 和 clientY 获取拖拽位置,并实时更新位置
        document.onmousemove = function (ev){
            var oEvent = ev || event;
            l = oEvent.clientX - disX;
            t = oEvent.clientY - disY;
            //增加边框限制
            if (l < 0) {
                l = 0;
            }else if (l > document.documentElement.clientWidth - obj.offsetWidth) {
                l = document.documentElement.clientWidth - obj.offsetWidth;
            }
            if (t < 0) {
                t = 0;
            }else if (t > document.documentElement.clientHeight - obj.offsetHeight) {
                t = document.documentElement.clientHeight - obj.offsetHeight;
            }
            obj.style.left = l + 'px';
            obj.style.top = t + 'px';
        }
        //3、mouseup 时,拖拽结束
        document.onmouseup = function () {
            document.onmousemove = null;
            document.onmouseup = null;
        }
        //阻止默认事件,自动选中文字的现象
        return false;
    }
}
```

22、请实现如下功能

说明:没太看懂需求,这里作为 cookie 操作方式扩展理解,操作需要自己写 api

```
//写 cookie 的方法
function setcookie(name,value,days){
    //参数解释:键、值、生命周期,设置 cookie 要设置生命周期
    var exp = new Date();
    exp.setTime(exp.getTime() + days*24*60*60*1000); //设置过期时间为 days 天
    document.cookie = name + "="+ escape (value) + ";expires=" + exp.toGMTString();
}
//获取 cookie 的方法
```

```
function getCookie(name){
    var result = "";
    var myCookie = ""+document.cookie+";";                  //获取所有 cookie
    var searchName = "+name+";                                  //格式化需要查找的 cookie
    var startOfCookie = myCookie.indexOf(searchName);//查看是否存在
    var endOfCookie;
    if(startOfCookie != -1){
        startOfCookie += searchName.length;
        endOfCookie = myCookie.indexOf(";",startOfCookie);
        result = (myCookie.substring(startOfCookie,endOfCookie));
    }
    return result;
}
(function(){
    var oTips = document.getElementById('tips');
    var page = {
        //检查 tips 的 cookie 是否存在并且允许显示
        check: function(){
            var tips = getCookie('tips');
            //tips 的 cookie 不存在 或 存在且值为 show,则返回 true
            if(!tips || tips == 'show') return true;
            //如果 cookie 存在,且值为 never_show_again,则返回 false
            if(tips == "never_show_again") return false;
        },
        //显示隐藏的方法,隐藏的时候要设置 cookie
        hideTip: function(bNever){
            if(bNever) setcookie('tips', 'never_show_again', 365);
            oTips.style.display = "none";
        },
        showTip: function(){
            oTips.style.display = "block";
        },
        //初始化方法
        init: function(){
            var _this = this;
            if(this.check()){
                _this.showTip();
                setcookie('tips', 'show', 1);
            }
            oTips.onclick = function(){
                _this.hideTip(true);
            };
        }
    };
```

```
    page.init();
})();
```

23、说出以下函数的作用是?空白区域应该填写什么?

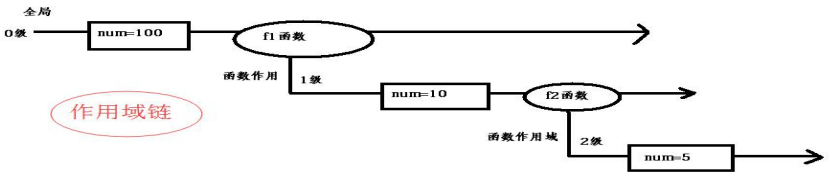
```
//功能实现:使用面向对象思想,写构造函数,其原型中添加一个格式化方法
//可以将方法的参数通过{数字}这样的差值语法,将指定字符串格式化(内容替换),类模板引擎的效果
(function(window){
    function fn(str){
        this.str=str;
    }
    fn.prototype.format = function(){
        var arg = arguments;                    //获取所有实参(这里要填写 arguments)
                                                //指定替换规则,下面这里要写正则表达式
        return this.str.replace(/\{(\d+)\}/ig,function(a,b){
            //先获取{数字}这样的字符串,全局匹配,然后获取通过 b 获取第一个子表达式的数组,就能作为参数组的索引使用
            return arg[b]||"";                  //参数组中存在对应值则返回替换,没有则返回空
        });
    }
    //将方法映射给 window,提供唯一接口,防止污染(jQuery 中也是这样做的)
    window.fn = fn;
})(window);
//测试
(function(){
    var t = new fn('<p><a href="{0}">{1}</a><span>{2}</span></p>');
    console.log(t.format('http://www.alibaba.com','Alibaba','Welcome'));
    //<p><a href="http://www.alibaba.com">Alibaba</a><span>Welcome</span></p>
})();
```

24、Javascript 作用域链?？

理解变量和函数的访问范围和生命周期，全局作用域与局部作用域的区别, JavaScript 中没有块作用域,函数的嵌套形成不同层次的作用域,嵌套的层次形成链式形式,通过作用域链查找属性的规则需要深入理解.

当一个函数内部访问的成员不存在时,则向其外部的函数进行访问,如果再访问不到,继续向外查找,直到全局作用域(script),没有则报错

sacript 内作为 0 级作用域(全局变量和函数),0 级作用域的函数内(全局函数内)为 1 级作用域,其内的第二层函数为 2 级作用域.....访问变量时,从最后内层的作用域开始向 0 级查找



25、谈谈 this 对象的理解.

普通函数中:this ==》window

定时器中:this==》window

构造函数中:this==》当前实例对象

对象的方法中:this==》当前实例对象

原型对象的方法中:this==》当前实例对象

事件处理函数中:this==》事件触发对象

26、eval 是做什么的?

将一个 JavaScript 代码字符串求值成特定的对象, 是一个顶级函数并且跟任何对象无关,参数是字符串表示了一个 JavaScript 表达式,声明, 或声明的序列,解析成对象后并执行

eval() 是一个危险的函数, 它可以像拥有调用者的权力一样调用代码,用字符串来运行 eval(),可能会被黑,如使用恶意代码获取权限,常规用例的安全会被 eval()改变,所以应该避免使用 eval,非常不安全

而且从说明上能看出,耗性能(2 个步骤,一次解析成 js 语句,一次执行)

27、关于事件 IE 与火狐的事件机制有什么区别?如何阻止冒泡?

在 IE 中,事件对象是作为一个全局变量来保存和维护的.所有的浏览器事件,不管是用户触发的,还是其他事件,都会更新 window.event 对象.所以在代码中,只要调用 window.event 就可以获取事件对象, 再 event.srcElement 就可以取得触发事件的元素进行进一步处理.

在 FireFox 中, 事件对象却不是全局对象, 一般情况下是现场发生,现场使用, FireFox 把事件对象自动传给事件处理程序.

存在不同,就要进行兼容性处理,通常在事件执行函数中兼容性获取,e = window.event || e; //后面 e 是执行函数参数

阻止冒泡:window.event? window.event.cancelBubble = true : e.stopPropagation();

阻止默认:window.event? window.event.returnValue = false : e.preventDefault();

原生 javascript 的 return false 只会阻止默认行为,而是用 jQuery 的话则既阻止默认行为又防止对象冒泡

28、什么是闭包(closure),为什么要用它?

简单的理解是函数的嵌套形成闭包,闭包包括函数本身以及它的外部作用域,函数 a 内嵌套 b,且返回 b,当调用函数 a 时,用变量接收函数 b,就形成了闭包

(这里还可以扩展一下作用域和作用域链的问题,js 只有全局作用域、函数作用域,没有块级作用域{},作用域链的查找方式,由内向外查找,函数作用域内的变量,全局不能直接访问,要想访问,就要用闭包....)

优点:使用闭包可以形成独立的空间,缓存数据,延长变量的生命周期

缺点:延长了作用域链,需要释放的变量不能及时释放,可能引发内存泄漏(这里又能扩展,内存泄漏也称作"存储渗漏",用动态存储分配函数动态开辟的空间,在使用完毕后未释放,结果导致一直占据该内存单元.直到程序结束.(其实说白了就是该内存空间使用完毕之后未回收)即所谓内存泄漏.内存泄漏形象的比喻是"操作系统可提供给所有进程的存储空间正在被某个进程榨干",最终结果是程序运行时间越长,占用存储空间越来越多,最终用尽全部存储空间,整个系统崩溃.)

29、javascript 代码中的"use strict";是什么意思?使用它区别是什么?

意思是使用严格模式,使用严格模式,一些不规范的语法将不再支持

严格模式:ECMAscript 5 中添加

设置目的:

消除 Javascript 语法的一些不合理、不严谨之处,减少一些怪异行为;

消除代码运行的一些不安全之处,保证代码运行的安全;

提高编译器效率,增加运行速度;

为未来新版本的 Javascript 做好铺垫

常见区别:

全局变量显式声明,不再支持 v=1;这种全局变量

静态绑定,(动态绑定,js 是动态语言,属性和方法属于哪一个对象,不是在编译时确定的,而是在运行时确定的),涉及到 eval 作用域,其内部定义的变量都是局部变量(原来 js 只有全局作用域和函数作用域)

禁止 this 关键字指向全局对象

禁止在函数内部遍历调用栈

不能删除变量.只有 configurable 设置为 true 的对象属性,才能被删除

禁止对只读属性进行赋值,报错

禁止扩展的对象添加新属性,会报错.

禁止删除一个不可删除的属性,会报错.

禁止对象有多个重名属性,

禁止对 arguments 赋值

arguments 不再追踪参数的变化

禁止使用 arguments.callee

严格模式只允许在全局作用域或函数作用域的顶层声明函数.也就是说,不允许在非函数的代码块内声明函数

严格模式新增了一些保留字:implements, interface, let, package, private, protected, public,static, yield.

30、如何判断一个对象是否属于某个类(在严格来说在 ES6 之前 js 没有类的概念)?

在 ES6 之前 javascriprt 语法中没有类语法,被认为不利于大型面向对象的项目开发.ES6 中终于在千呼万唤中加入了类语法,但实质上还是对原型继承的一种封装,写起来会比较直观,(OO 的三大特性:封装,继承,多态).

instanceof constructor

没查到具体方法,看 instanceof 结果,类和其继承的类都会返回 true

31、new 操作符具体干了什么呢?

1. 创建一个空对象, 并且将 this 变量引向该对象, 同时继承该对象的原型

2. 属性和方法被加入到 this 引用的对象中

3. 新创建的对象由 this 引用, 并且最后隐式的返回 this

32、用原生 JavaScript 的实现过什么功能吗?

此题目主要考擦我们对 JS 的实战经验.其实在学习当中,我们做过蛮多用 JS 原生做的功能,诸如:轮播图、放大镜、瀑布流、旋转木马、框架封装工具类等等.....我们可以针对其中一个功能进行简单地阐述

阐述的顺序

介绍实现的是什么功能,然后再根据其中用到了什么样的技术点,亮点是什么,一些使用原生 JS 实现的功能, 虽然效率高, 但通常需要考虑的是浏览器兼容性问题, 还有一些隐藏的 BUG

实例:瀑布流

```
window.onload = function() {
    var container = document.getElementById("container");          //获取当前容器
    var boxes = container.children;                                //获取所有盒子
    var boxWidth = boxes[0].offsetWidth;                           //width+padding+border 获取盒子的宽度
    var pageWidth = window.innerwidth;                             //获取屏幕的宽度
    var column = Math.floor(pageWidth / boxWidth);                 //获取列数
    waterFall();
    //瀑布流
    function waterFall() {
        var arrHeights = [];                                       //用于存储第一行中每个盒子的高度
        for(var i = 0; i < boxes.length; i++) {
            if(i < column) {
                arrHeights.push(boxes[i].offsetHeight);            //获取第一行的所有盒子的高度存储到数组当中
            } else {
                //先找到第一行中最小的一列,然后放在这一列的下面
```

```
                    //获取第一行最小索引
                    var minHeightIndex = getMinHeight(arrHeights).index;
                    //获取第一行最小高度
                    var minHeightValue = getMinHeight(arrHeights).value;
                    //让每个盒子都有定位,并设置 left 和 top 值
                    boxes[i].style.position = "absolute";
                    boxes[i].style.left = boxes[minHeightIndex].offsetLeft + "px";
                    boxes[i].style.top = minHeightValue + "px";
                    //更新数组的数据,原来的值+新盒子的高度
                    arrHeights[minHeightIndex] = minHeightValue + boxes[i].offsetHeight;
                }
            }
        }
        window.onscroll = function() {
            if(bottomed()) {                                    //判断是否触底
                var config = [
                    {"src" : "images/P_000.jpg"},
                    {"src" : "images/P_001.jpg"},
                    {"src" : "images/P_002.jpg"},
                    {"src" : "images/P_003.jpg"},
                    {"src" : "images/P_004.jpg"},
                    {"src" : "images/P_005.jpg"}
                ];
                //动态添加盒子
                for(var i = 0; i < config.length; i++) {
                    var div = document.createElement("div");
                    div.className = "box";
                    var img = document.createElement("img");
                    img.src = config[i].src;
                    div.appendChild(img);
                    container.appendChild(div);
                }
                waterFall();
            }
        }
        //触底
        function bottomed() {
            var height = scroll().top + getClient().height;
            if(height > boxes[boxes.length - 1].offsetTop) {
                return true;
            }
        }
    }
    //获取盒子的最小高度
```

```
function getMinHeight(arr) {
    var obj = {
        index : 0,
        value : arr[0]
    }
    for(var i = 0; i < arr.length; i++) {
        if(obj.value > arr[i]) {
            obj.value = arr[i];
            obj.index = i;
        }
    }
    return obj;
}
//滚动的兼容函数
function scroll() {
    return {
        top : window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop || 0,
        left : window.pageXOffset || document.documentElement.scrollLeft || document.body.scrollLeft || 0
    }
}
```

33、JavaScript 中,有一个函数,执行时对象查找时,永远不会去查找原型,这个函数是?

hasOwnProperty():用来判断某对象是否含有指定的自身属性

1.用法:obj.hasOwnProperty(prop) --->prop 表示检测的属性名称

2.描述:所有继承了 Object.prototype 的对象都会从原型链上继承到 hasOwnProperty 方法,这个方法可以用来检测一个对象是否含有特定的自身属性,和 in 运算符不同,该方法会忽略掉那些从原型链上继承到的属性.

3.实例:自身属性和继承属性的区别

```
o =newObject();
o.prop ='exists';
o.hasOwnProperty('prop');             // 返回 true
o.hasOwnProperty('toString');         // 返回 false
o.hasOwnProperty('hasOwnProperty');   // 返回 false
```

4.注意点:如果函数中存在一个与 hasOwnPoroperty 方法,此时,原有的 hasOwnPoroperty 方法会被屏蔽掉

34、对 JSON 的了解?

JSON 指的是 JavaScript 对象表示法(JavaScript Object Notation)

1.轻量级数据交互格式

2.可以形成复杂的嵌套格式

3.解析非常方便

4.易于读写,占用带宽小

JSON 的表现形式

```
1.var arrJson = [{"name":"xiaoming","age": 18},{"name":"xiaohong","age": 19}];
2.var strJson = {"name":"Lily","age": 12};
```

35、JS 延迟加载的方式有哪些?

方案一:<script>标签的 async="async"属性(详细参见:script 标签的 async 属性)

方案二:<script>标签的 defer="defer"属性

方案三:动态创建<script>标签

方案四:AJAX eval(使用 AJAX 得到脚本内容,然后通过 eval_r(xmlhttp.responseText) 来运行脚本)

方案五:iframe 方式

36、JS 模块化开发怎么做?

理解 1:所谓的模块(在业内, 模块, 组件等概念还很模糊), 在代码中按照特定的方式对代码进行分解(解耦).分解后代码按照功能零件的形式存在. 如果按照某种逻辑将其组合在一起, 这个时候这个整体可以称之为一个模块.

理解 2:是一种将系统分离成独立功能部分的方法,可将系统分割成独立的功能部分,也就是说由一组高度解耦的、存放在不同模块中的独特功能构成,也避免了全局名称空间污染

例如:

立即执行函数写法

```
var module = (function(){
    //...
})();
```

输入全局变量写法

```
var module = (function(window){
    //...
})(window);
```

模块化管理工具:requirejs,seajs

37、AMD(Modules/Asynchronous-Definition)、CMD(Common Module Definition)规范区别?

1、AMD 是"Asynchronous Module Definition"的缩写,意思就是"异步模块定义".它采用异步方式加载模块,模块的加载不影响它后面语句的运行.所有依赖这个模块的语句,都定义在一个回调函数中,等到加载完成之后,这个回调函数才会运行.

AMD 是提前执行

形式:define(['dep1','dep2'],function(dep1,dep2){...});

2、CMD 是"Common Module Definition"更贴近 Node Modules 规范,一个模块就是一个文件;它推崇**依赖就近**想什么时候 require 就什么时候加载,实现了懒加载,**延迟执行**(as lazy as possible);也没有全局 require,**每个 API 都简单纯粹**

CMD 是延迟执行

形式:define(function(){});

38、requireJS 的核心原理是什么?(如何动态加载的?如何避免多次加载的? 如何缓存的?)

概念:RequireJS 是一个 JavaScript 文件或者模块的加载器.它可以提高 JavaScript 文件的加载速度,避免不必要的堵塞.它针对于在浏览器环境中使用做过专门的优化,但它也可以在其他的 JavaScript 环境中使用,像 Node.js 一样可以在服务器上运行

工作原理:

1.我们在使用 requireJS 时,都会把所有的 js 交给 requireJS 来管理,也就是我们的页面上只引入一个 require.js,把 data-main 指向我们的 main.js.

2.通过我们在 main.js 里面定义的 require 方法或者 define 方法,requireJS 会把这些依赖和回调方法都用一个数据结构保存起来.

3.当页面加载时,requireJS 会根据这些依赖预先把需要的 js 通过 document.createElement 的方法引入到 dom 中,这样,被引入 dom 中的 script 便会运行.

4.由于我们依赖的 js 也是要按照 requireJS 的规范来写的,所以他们也会有 define 或者 require 方法,同样类似第二步这样循环向上查找依赖,同样会把他们村起来.

5.当我们的 js 里需要用到依赖所返回的结果时(通常是一个 key value 类型的 object),requireJS 便会把之前那个保存回调方法的数据结构里面的方法拿出来并且运行,然后把结果给需要依赖的方法.

39、谈一谈你对 ECMAScript6 的了解?

1、let、const 和 块级作用域

let 允许创建块级作用域,ES6 中推荐使用 let 定义变量

同样在块级作用域有效的另一个变量声明方式是 const,它可以声明一个常量.ES6 中,const 声明的常量类似于指针,它指向某个引用,也就是说这个「常量」并非一成不变的

注意点:

let 关键词声明的变量不具备变量提升(hoisting)特性

let 和 const 声明只在最靠近的一个块中(花括号内)有效

当使用常量 const 声明时,请使用大写变量,如:CAPITAL_CASING

const 在声明时必须被赋值

2、箭头函数

ES6 中,箭头函数就是函数的一种简写形式,使用括号包裹参数,跟随一个 =>,紧接着是函数体:

```
var sum = function() {
    return 1 + 1;
};
```

转换后:

```
let sum = () => 1 + 1;
```

3、函数参数默认值

```
let sum = (a, b=1) => a + b;
```

4、Symbol

Symbol 是一种新的数据类型,它的值是唯一的,不可变的.ES6 中提出 symbol 的目的是为了生成一个唯一的标识符,不过你访问不到这个标识符

40、ECMAScript6 怎么写 class 么,为什么会出现 class 这种东西?

1、类

在 ECMAScript5 中,用通过原型的方式来进行模拟类和继承,但在 ECMAScript6 当中,出现了类(class)和继承(extend).ES6 中有 class 语法.值得注意是,这里的 class 不是新的对象继承模型,它只是原型链的语法糖表现形式

extends 允许一个子类继承父类

```
class Person {
    constructor(name) {
        this.name = name;
    }
}
class Student extends Person{
    constructor(name,age) { //子类的 constructor 函数需要调用 super()函数
        super(name);
        this.age = age;
    }
    showInfo() {
        console.log(this.name, this.age);//Charles 18
    }
}
```

```
var stu = new Student("Charles",18);
```

注意点:

类的声明不会提升(hoisting),如果你要使用某个 Class,那你必须在使用之前定义它,否则会抛出一个 ReferenceError 的错误

在类中定义函数不需要使用 function 关键词

2、为什么要有 class 这种东西

a.引入了 Class(类)这个概念,作为对象的模板.通过 class 关键字,可以定义类.基本上,ES6 的 class 可以看作只是一个语法糖,它的绝大部分功能,ES5 都可以做到

b.新的 class 写法只是让对象原型的写法更加清晰、更像面向对象编程的语法而已

41、document.write 和 innerHTML 的区别?

document.write 是重写整个 document, 写入内容是字符串的 html

innerHTML 是 HTMLElement 的属性,是一个元素的内部 html 内容

42、DOM 操作——怎样添加、移除、移动、复制、创建和查找节点?

1.添加节点

createElement()、createTextNode()、craeteDocumentFragment()

2.添加、移除、替换、插入

appendChild()、removeChild()、replaceChild()、insertBefore()

3.查找节点

getElementById()

getElementsByTagName()

getElementsByClassName() -->IE9+

querySelect() -->IE8+

querySelectAll() -->IE8+

43、call()和 apply()的含义和区别?

1、call 和 apply 方法都是 Function.prototype 中的

2、都可以调用函数(可以看成是函数的另一种调用方式)

相同点

1、都是调用函数,如果有返回值可以直接接收

2、调用 call 或 apply 时如果传入的是 null,那么函数的 this 为 window 对象,如果传入的是对象,那么函数的 this 为当前传入的对象

不同点

1、call 传递参数的时候一个一个的传入

2、apply 传递的参数为数组

44、数组和对象有哪些原生方法,列举一下??

1.Array.concat()　　//数组拼接

2.Array.push()　　//向数组尾部添加元素

3.Array.pop()　　//从数组尾部删除元素

4.Array.shift()　　//从数组头部删除元素

5.Array.unshift()　　//向数组头部删除元素

6.Array.slice()　　//返回数组的一部分

7.Array.splice()　　//插入,删除或替换元素

8.Array.reverse()　　//颠倒数组中元素的顺序

9.Array.sort()　　//对数组进行排序

10.Array.toString()　//将数组转换成一个字符串

11.Array.valueOf()　//

12.Array.join()　　//将数组元素连接起来以构建一个字符串

13.Array.length()　//数组的长度

14.Array.toLocalString() //把数组转换成局部字符串

45、JavaScript 中的作用域与变量声明提升?

作用域:

1.全局作用域:使用范围在整个的页面(在 script 标签中)

2.函数作用域:使用方位在该函数内部----(闭包)

3.js 中没有块级作用域

预解析:

1. 把变量的声明和函数的声明提升了,函数内部的变量只能提升到函数内部的顶部

2. 作用域链: 把全局的变量和函数 f1 都看成是 0 级的链,f1 内部的函数和变量堪称是 1 级的,如果函数内部访问变量的时候,先在自己的函数中搜索,如果有则直接使用,如果没有则去上一级查找,找到则使用,找不到则继续向上一级找,一直找到全局作用域,如果还没有则报错

46、如何编写高性能的 JavaScript?

1.使用 DocumentFragment 优化多次 append

2.通过模板元素 clone ,替代 createElement

3.使用一次 innerHTML 赋值代替构建 dom 元素

4.使用 firstChild 和 nextSibling 代替 childNodes 遍历 dom 元素

5.使用 Array 做为 StringBuffer ,代替字符串拼接的操作

6.将循环控制量保存到局部变量

7.顺序无关的遍历时,用 while 替代 for

8.将条件分支,按可能性顺序从高到低排列

9.在同一条件子的多(>2)条件分支时,使用 switch 优于 if

10.使用三目运算符替代条件分支

11.需要不断执行的时候,优先考虑使用 setInterval

47、那些操作会造成内存泄漏?

所谓的内存泄露,就是在您不再拥有或需要它的时候仍然存在

垃圾回收机制会定期扫描对象,并计算引用了每个对象的其他对象的数量.如果一个对象的没有被其他对象引用过,或对该对象的唯一引用是循环的,那么该对象内存即可回收

1.setTimeout 的第一个参数使用字符串而非函数的话,会引发内存泄露

2.闭包

3.控制台日志

4.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)

48、JavaScript 对象的几种创建方式?

1. 工厂模式

工厂模式虽然解决了创建多个相似对象的问题,但却没有解决对象识别的问题(即怎样知道一个对象的类型)

2. 构造函数模式

比起工厂模式

a) 没有显式地创建对象;

b) 直接将属性和方法赋给了 this 对象;

c) 没有 return 语句.

但是使用构造函数的主要问题,就是每个方法都要在每个实例上重新创建一遍

3. 原型模式

好处是可以让所有对象实例共享它所包含的属性和方法

原型模式的最大问题是由其共享的本性所导致的,如果一个实例对象修改了引用类型的值,那么接下来的所有的新实例对象读取到都是已修改后的引用类型的值

4. 混合构造函数和原型模式

构造函数模式用于定义实例属性,而原型模式用于定义方法和共享的属性

每个实例都会有自己的一份实例属性的副本,但同时又共享着对方法的引用,最大限度地节省了内存

5. 动态原型模式

使用动态原型模式时,不能使用对象字面量重写原型.前面已经解释过了,如果在已经创建了实例的情况下重写原型,那么就会切断现有实例与新原型之间的联系

6. 寄生构造函数模式

这种模式的基本思想是创建一个函数,该函数的作用仅仅是封装创建对象的代码,然后再返回新创建的对象

但是返回的对象与构造函数或者与构造函数的原型属性之间没有关系.不推荐使用这种模式

7. 稳妥构造函数模式

稳妥对象最适合在一些安全的环境中(这些环境中会禁止使用 this 和 new),或者在防止数据被其他应用程序(如 Mashup 程序)改动时使用.稳妥构造函数遵循与寄生构造函数类似的模式,

但有两点不同:

1.是新创建对象的实例方法不引用 this;

2.是不使用 new 操作符调用构造函数

49、JavaScript 继承的 6 种方法?

1. 原型链继承

i. 引用类型值的原型属性会被所有实例共享

2. 借用构造函数继承

i. 借助 call()或 apply()方式改变 this 的指向

ii. 解决了继承,属性值可以独立的问题,出现了方法不能共享的问题

3. 组合继承(原型+借用构造)

i. 使用原型链实现对原型属性和方法的继承,而通过借用构造函数来实现对实例属性的继承

ii. instanceof 和 isPrototypeOf()能够用于识别基于组合继承创建的对象

iii. 组合继承最大的问题就是无论什么情况下,都会调用两次超类型构造函数:一次是在创建子类型原型的时候,另一次是在子类型构造函数内部

例:

```
function SuperType(name){
 this.name = name;
 this.colors = ["red", "blue", "green"];
}
SuperType.prototype.sayName = function(){
 alert(this.name);
};
function SubType(name, age){
 SuperType.call(this, name); //第二次调用 SuperType()
```

```
 this.age = age;
}
SubType.prototype = new SuperType(); //第一次调用 SuperType()
SubType.prototype.constructor = SubType;
SubType.prototype.sayAge = function(){
 alert(this.age);
};
```

4. 原型式继承
   i. 必须有一个对象可以作为另一个对象的基础
5. 寄生式继承
   i. 使用寄生式继承来为对象添加函数,会由于不能做到函数复用而降低效率;这一点与构造函数模式类似
6. 寄生组合式继承
   i. 即通过借用构造函数来继承属性,通过原型链的混成形式来继承方法.其背后的基本思路是:不必为了指定子类型的原型而调用超类型的构造函数,我们所需要的无非就是超类型原型的一个副本而已.本质上,就是使用寄生式继承来继承超类型的原型,然后再将结果指定给子类型的原型
   ii. 集寄生式继承和组合继承的优点与一身,是实现基于类型继承的最有效方式

   方式:

```
function object(o) {
    function F(){}
    F.prototype = o;
    return new F();
}//借助原型可以基于已有的对象创建新对象,同时还不必因此创建自定义类型
function inheritPrototype(superType,subType) {
    var prototype = object(superType.prototype); //创建对象(浅复制)
    prototype.constructor = subType; //增强对象
    subType.prototype = prototype; //指定对象
}
function SuperType(name){
    this.name = name;
    this.colors = ["red", "blue", "green"];
}
SuperType.prototype.sayName = function(){
    alert(this.name);
};
function SubType(name, age){
     SuperType.call(this, name);
     this.age = age;
}
inheritPrototype(SubType, SuperType);
SubType.prototype.sayAge = function(){
    alert(this.age);
};
```

总结:这个例子的高效率体现在它只调用了一次 SuperType 构造函数,并且因此避免了在 SubType. prototype 上面创建不必要的、多余的属性.与此同时,原型链还能保持不变;因此,还能够正常使用 instanceof 和 isPrototypeOf().开发人员普遍

认为寄生组合式继承是引用类型最理想的继承方式.

50、eval 是做什么的?

1. 它的功能是把对应的字符串解析成 JS 代码并运行

2. 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)

51、JavaScript 原型,原型链? 有什么特点?

1. 原型对象也是普通的对象,是对象一个自带隐式的 __proto__ 属性,原型也有可能有自

己的原型,如果一个原型对象的原型不为 null 的话,我们就称之为原型链

2. 原型链是由一些用来继承和共享属性的对象组成的(有限的)对象链(基本思想是利用原型让一个引用类型继承另一个引用类型的属性和方法)

理解:每个构造函数都有一个原型对象,原型对象都包含一个指向构造函数的指针,而实例都包含一个指向原型对象的内部指针.那么,假如我们让原型对象等于另一个类型的实例,结果会怎么样呢?显然,此时的原型对象将包含一个指向另一个原型的指针,相应地,另一个原型中也包含着一个指向另一个构造函数的指针.假如另一个原型又是另一个类型的实例,那么上述关系依然成立,如此层层递进,就构成了实 例与原型的链条.这就是所谓原型链的基本概念

3. 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,如果没有的话,就会查找他的 Prototype 对象是否有这个属性

52、事件、IE 与火狐的事件机制有什么区别 ? 如何阻止冒泡?

1. 我们在网页中的某个操作(有的操作对应多个事件).例如:当我们点击一个按钮就会产生一个事件.是可以被 JavaScript 侦测到的行为

2. 事件处理机制:IE 是事件冒泡、firefox 同时支持两种事件模型,也就是:捕获型事件和冒泡型事件

3. e.stopPropagation();注意:旧 ie 的方法:ev.cancelBubble = true;

53、简述一下 Sass、Less,且说明区别?

他们是动态的样式语言,是 CSS 预处理器,CSS 上的一种抽象层.他们是一种特殊的语法/语言而编译成 CSS.

区别:

1.变量符不一样,less 是@,而 Sass 是$;

2.Sass 支持条件语句,可以使用 if{}else{},for{}循环等等.而 Less 不支持;

3.Sass 是基于 Ruby 的,是在服务端处理的,而 Less 是需要引入 less.js 来处理 Less 代码输出 Css 到浏览器

54、说说你对 this 的理解?

在 JavaScript 中,this 通常指向的是我们正在执行的函数本身,或者是,指向该函数所属的对象.

1.普通函数中 this 是 window

2.构造函数中 this 是当前的实例对象

3.对象中的 this 是指向其本身

4.原型对象的方法中 this 是当前的实例对象

5.计时器中 this 是 window

6.事件处理函数,this 是触发该事件的对象

55、你用过 require.js 吗?它有什么特性?

1.实现 js 文件的异步加载,避免网页失去响应;

2.管理模块之间的依赖性,便于代码的编写和维护.

56、谈一下 JS 中的递归函数,并且用递归简单实现阶乘?

递归即是程序在执行过程中不断调用自身的编程技巧,当然也必须要有一个明确的结束条件, 不然就会陷入死循环.

57、外部 JS 文件出现中文字符,会出现什么问题,怎么解决?

1. 会出现乱码,加 charset="GB2312";

2. 另一种解决方式:网页文件和外部 JS 文件都是 UTF8 编码

58、写一个通用的事件侦听器函数?

```
// event(事件)工具集,来源:github.com/markyun
markyun.Event = {
    // 页面加载完成后
    readyEvent : function(fn) {
        if (fn==null) {
            fn=document;
        }
        var oldonload = window.onload;
        if (typeof window.onload != 'function') {
            window.onload = fn;
        } else {
            window.onload = function() {
                oldonload();
                fn();
            };
        }
    },
    // 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.removeEnentListener) {
            element.removeEnentListener(type, handler, false);
        } else if (element.detachEvent) {
            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;
    },
    // 获取 event 对象的引用,取到事件的所有信息,确保随时能使用 event;
    getEvent : function(e) {
        var ev = e || window.event;
        if (!ev) {
            var c = this.getEvent.caller;
            while (c) {
                ev = c.arguments[0];
                if (ev && Event == ev.constructor) {
                    break;
                }
                c = c.caller;
            }
        }
        return ev;
    }
};
```

59、前端开发的优化问题(看雅虎 14 条性能优化原则)

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

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

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

4. 当需要设置的样式很多时设置 className 而不是直接操作 style.
5. 少用全局变量、缓存 DOM 节点查找的结果.减少 IO 读取操作.
6. 避免使用 CSS Expression(css 表达式)又称 Dynamic properties(动态属性).
7. 图片预加载,将样式表放在顶部,将脚本放在底部 加上时间戳.
8. 避免在页面的主体布局中使用 table,table 要等其中的内容完全下载之后才会显示出来,显示比 div+css 布局慢.

五、Ajax

1、Ajax 是什么?它最大的特点是?优缺点?

Ajax (asynchronous javascript and xml),即异步 JavaScript 和 xml,主要用来实现客户端与服务器端的异步数据交互,实现页面的局部刷新.早期浏览器并不能原生支持 ajax,可以使用隐藏帧(iframe)方式变相实现异步效果,后来的浏览器提供了对 ajax 的原生支持 .

ajax 的最大的特点+优点:

Ajax 可以实现异步通信效果,实现页面局部刷新,避免用户不断刷新或者跳转页面,带来更好的用户体验;按需获取数据,节约带宽资源;

ajax 的缺点

1、ajax 不支持浏览器 back 按钮.

2、安全问题, AJAX 暴露了与服务器交互的细节,诸如跨站点脚步攻击、SQL 注入攻击.

3、对搜索引擎的支持比较弱.

4、AJAX 不能很好支持移动设备.

2、如何创建一个 Ajax?简述 ajax 的过程.ajax 的交互模型?

1.创建 XMLHttpRequest 对象,也就是创建一个异步调用对象

2.创建一个新的 HTTP 请求,并指定该 HTTP 请求的方法、URL 及验证信息(即请求报文行)

```
xhr.open('get','5-register-get.php?name='+name);
```

3.设置请求报文头;(Content-Type:请求资源的 MIME 类型)

```
xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded');//get 方式不需要
```

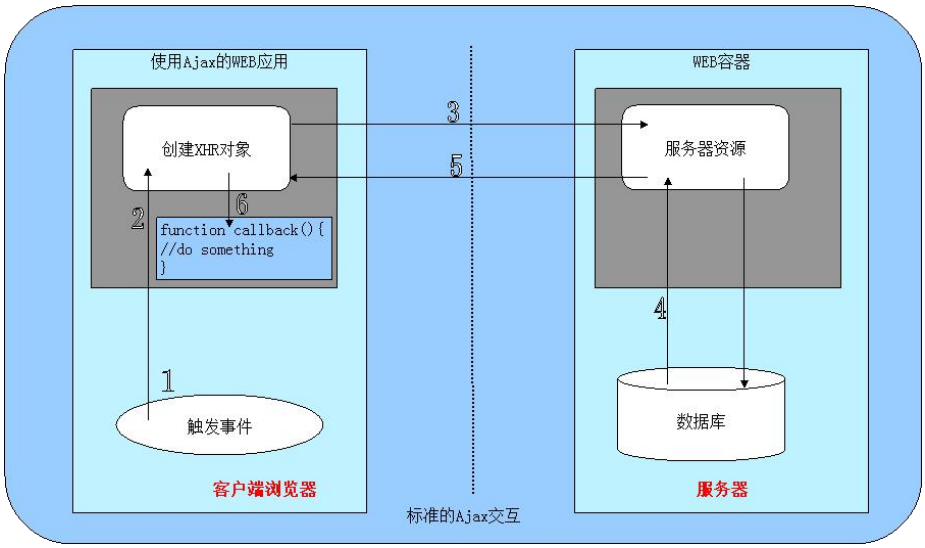
4.设置响应 HTTP 请求状态变化的函数

5. 发送 HTTP 请求

```
xhr.send(null);
```

6.获取异步调用返回的数据

7.使用 JavaScript 和 DOM 实现局部刷新



3、一个页面从输入 URL 到页面加载显示完成,这个过程中都发生了什么?

分为 4 个步骤:

1. 当发送一个 URL 请求时(可能是 Web 页面还是 Web 页面上每个资源的 URL),浏览器都会开启一个线程来处理这个请求,同时在远程 DNS 服务器上启动一个 DNS 查询,这能使浏览器获得请求对应的 IP 地址.

2. 浏览器与远程 Web 服务器通过 TCP 三次握手协商来建立一个 TCP/IP 连接.该握手包括一个同步报文,一个同步-应答报文和一个应答报文,这三个报文在 浏览器和服务器之间传递.该握手首先由客户端尝试建立起通信,而后服务器应答并接受客户端的请求,最后由客户端发出已经被接受的请求报文.

3. 一旦 TCP/IP 连接建立,浏览器会通过该连接向远程服务器发送 HTTP 的 GET 请求.远程服务器找到资源并使用 HTTP 响应返回该资源,值为 200 的 HTTP 响应状态表示一个正确的响应.

4. 此时,Web 服务器提供资源服务,客户端开始下载资源.

4、ajax 请求时,如何解释 json 数据

使用 eval() 或者 JSON.parse() 鉴于安全性考虑,推荐使用 JSON.parse()更靠谱,对数据的安全性更好.

【JSON.parse(): 将 JSON 字符串转换成对象.】

5、同步和异步的区别?

同步:阻塞的-张三叫李四去吃饭,李四一直忙得不停,张三一直等着,直到李四忙完两个人一块去吃饭=浏览器向服务器请求数据,服务器比较忙,浏览器一直等着(页面白屏),直到服务器返回数据,浏览器才能显示页面.

异步:非阻塞的-张三叫李四去吃饭,李四在忙,张三说了一声然后自己就去吃饭了,李四忙完后自己去吃=浏览器向服务器请求数据,服务器比较忙,浏览器可以自如的干原来的事情(显示页面),服务器返回数据的时候通知浏览器一声,浏览器把返回的数据再渲染到页面,局部更新

6、阐述一下异步加载.

1. 异步加载的方案: 动态插入 script 标签
2. 通过 ajax 去获取 js 代码,然后通过 eval 执行
3. script 标签上添加 defer 或者 async 属性
4. 创建并插入 iframe,让它异步执行 js

7、GET 和 POST 的区别,何时使用 POST?

GET:一般用于信息获取,使用 URL 传递参数,请求的参数都暴露在 url 地址当中,如果传递中文参数,需要自己进行编码操作,安全性较低. 对所发送信息的数量也有限制,一般在 2000 个字符,有的浏览器是 8000 个字符;

POST:提交的数据内容存在于 http 请求体中,数据不会暴漏在 url 地址中. 一般用于修改服务器上的资源,对所发送的信息没有限制.

在以下情况中,请使用 POST 请求:

1. 无法使用缓存文件(更新服务器上的文件或数据库)
2. 向服务器发送大量数据(POST 没有数据量限制)
3. 发送包含未知字符的用户输入时,POST 比 GET 更稳定也更可靠

8、请解释一下 JavaScript 的同源策略.

同源策略是客户端脚本(尤其是 Javascript)的重要的安全度量标准.它最早出自 Netscape Navigator2.0,其目的是防止某个文档或脚本从多个不同源装载.所谓同源指的是:协议,域名,端口相同,同源策略是一种安全协议,指一段脚本只能读取来自同一来源的窗口和文档的属性.

9、如何解决跨域问题?

理解跨域的概念:协议、域名、端口都相同才同域,否则都是跨域 .

出于安全考虑,服务器不允许 ajax 跨域获取数据,但是可以跨域获取文件内容.基于这一点,可以动态创建 script 标签,使用标签的 src 属性访问 js 文件的形式,获取 js 脚本,并且这个 js 脚本中的内容是函数调用,该函数调用的参数是服务器返回的数据.为了获取这里的参数数据,需要事先在页面中定义回调函数,在回调函数中处理服务器返回的数据,这就是解决跨域问题的主流解决方案 .

10、解释 jsonp 的原理,以及为什么不是真正的 ajax

Jsonp 并不是一种数据格式,而 json 是一种数据格式,jsonp 是用来解决跨域获取数据的一种解决方案,具体是通过动态创建 script 标签,然后通过标签的 src 属性获取 js 文件中的 js 脚本,该脚本的内容是一个函数调用,参数就是服务器返回的数据,为了处理这些返回的数据,需要事先在页面定义好回调函数,本质上使用的并不是 ajax 技术

11、页面编码和被请求的资源编码如果不一致如何处理?

对于 ajax 请求传递的参数,如果是 get 请求方式,如果传递中文参数,由于不同的浏览器对参数编码的处理方式不同,在有些浏览器会乱码,所以对于 get 请求的参数需要使用 encodeURIComponent 函数对参数进行编码处理,后台开发语言都有相应的解码 api.对于 post 请求不需要进行编码 .

12、为什么利用多个域名来存储网站资源会更有效?

确保用户在不同地区能用最快的速度打开网站;如果某个域名崩溃用户也能通过其他郁闷访问网站;并且不同的资源放到不同的服务器上有利于减轻单台服务器的压力.

13、请说出三种减低页面加载时间的方法

1、减少 http 请求(合并文件,合并图片)

2、压缩 css、js 文件

一般 js、css 文件中存在大量的空格、换行、注释,这些利于阅读,如果能够压缩掉,将会很有利于网络传输.这个压缩比率还是比较高的,一般都有百分五十左右.这个代码压缩对于网页的加载还是很有用的.

3、外部 js、css 文件放在最底下

网页文件的载入是从上到下加载的,很多 Javascript 脚本执行效率较低,或者在网页前面都不需要执行的,如果将这些脚本放置到页面比较靠前的位置,可能导致网站内容载入速度下降或加载不了,将这些脚本放置在网页文件末尾.一定要放置在前面的脚本要改用所谓的"后载入"方式加载,在主体网页加载完成后再加载,防止其影响到主体网页的加载速度.

4、减少 dom 操作,尽可能用变量替代不必要的 dom 操作

5、网址后加斜杠(如 www.campr.com/目录,会判断这个"目录是什么文件类型,或者是目录.)

14、JSON 的优缺点.

JSON 是一种轻量级的数据交换格式,ECMA 的一个子集 .

优点:

1、轻量级、易于人的阅读和编写,json 代码的良好结构,可以很直观地了解存的是什么内容.

2、方便转换.有很多的 json api 提供了 json 字符串转成对象、对象转换成 json 串的方法.

3、便于机器(JavaScript)解析;

4、支持复合数据类型(数组、对象、字符串、数字)

5、方便于传输,较少冗余的字符

缺点:

1、没有 XML 格式这么推广的深入人心和喜用广泛,没有 XML 那么通用性;

2、JSON 格式目前在 Web Service 中推广还属于初级阶段.时

15、HTTP 状态码有那些?分别代表是什么意思?

100-199 用于指定客户端应相应的某些动作.
200-299 用于表示请求成功.

300-399 用于已经移动的文件并且常被包含在定位头信息中指定新的地址信息.	
400-499 用于指出客户端的错误.	
400 语义有误,当前请求无法被服务器理解.	
401 当前请求需要用户验证	
403 服务器已经理解请求,但是拒绝执行它.	
500-599 用于支持服务器错误.	
503 - 服务不可用	
200 OK	//客户端请求成功
301	//资源(网页等)被永久转移到其它 URL
400 Bad Request	//客户端请求有语法错误,不能被服务器所理解
403 Forbidden	//服务器收到请求,但是拒绝提供服务
404 Not Found	//请求资源不存在,输入了错误的 URL
500 Internal Server Error	//内部服务器发生不可预期的错误
503 Server Unavailable	//服务器当前不能处理客户端的请求,一段时间后可能恢复正常

六、流行框架（jQuery、Zepto、Underscore、Angular）

1、JQuery 的源码看过吗?能不能简单概况一下它的实现原理?

看过,在 JQuery 体系中,主要有两种对象,第一种是 JQuery 构造函数,第二种是 JQuery 构造函数产生的的对象,我们称之为 JQuery 对象,他们的主要关系如下:

1)JQuery 对象($('XX')),是 JQuery 原型属性的 init 方法实例出来的对象,调用 JQuery 构造函数时自动返回 init 的实例,以达到不用 new 创建对象的目的.

2)通过改变 init 方法的原型指向 JQuery 的原型以达到继承的目的,这样一来 init 方法中的 this 就可以使用 JQuery 原型对象中的方法

3)将$和 JQuery 暴露到 window 中,用户可以直接通过$或$('xx')调用到 JQuery 构造函数和 JQuery 对象原型中的方法

4)在 JQuery 构造函数和 JQuery 对象原型中有同一个 extend 方法,用于扩展自身的属性和方法.

```
//创建一个入口函数
function Itcast(selector,parent){
  //返回原型对象中F的实例对象
  return new Itcast.prototype.F(selector,parent);
}

var push = Array.prototype.push;//借用添加数组方法
var splice = Array.prototype.splice;
//itcast的原型
Itcast.fn = Itcast.prototype = {
  constructor:Itcast,//原型
  F:function(selector,parent){
    //先清除所有对象
    splice.call(this,0,this.length);
    var ele = selectors(selector,parent);//获取一个dom元素
    //借用数据的添加方法把dom元素放入当前对象中
    push.apply(this,ele);
    return this;
  }
}


Itcast.fn.extend = Itcast.extend = function(){...
}

//封装工具类
Itcast.extend({...
});

//F实例对象的方法
Itcast.fn.extend({...
});

//把原型中的F原型指向Itcast的原型
Itcast.prototype.F.prototype = Itcast.prototype;

//将$和itcast暴露到window上
window.$ = window.itcast = Itcast;
```

2、jQuery 的 slideUp 动画,如果目标元素是被外部事件驱动,当鼠标快速地连续触发外部元素事件, 动画会滞后的反复执行,该如何处理呢?

1、外部事件驱动:JS 是采用事件驱动的机制来响应用户操作的,也就是说当用户对某个 html 元素进行操作的时候,会产生一个事件,该事件会驱动某些函数来处理.事件源--事件类型--(事件对象)--事件处理程序.

2、slideUp()方法:元素由下到上缩短隐藏,只改变元素的高度

3、反复执行的原因--动画队里:动画效果的执行有先后顺序

4、stop()方法:立即停止元素正在进行的动画,如果后面还有动画等待执行,则以当前状态开始接下来的动画.

参数 1:clearQueue,设为 true 时,会把当前元素接下来尚未执行的动画队列都清空

参数 2:gotoEnd,设为 true 时,会让正在的执行的动画直接到达结束时刻的状态

说明:stop(false,true)是结束当前动画直接到达末状态的末状态(后续仍会执行),stop(true,true)是停止当前动画并直接到达当前动画的末状态,并清空动画队列(后续不再执行)

答案:先 stop(true,true)后 slideUp()

3、jQuery.fn 的 init 方法返回的 this 指的是什么对象?为什么要返回 this?

init 作为构造函数调用时,构造函数中的 this 指向构造函数的实例,init 中返回了一个 this,会覆盖作为构造函数的默认返回值,由于原来也是返回了 this,所以不会造成影响,当做连贯操作二次调用 init 方法时{$('body').init()},由于是方法调用,需要手动添加返回值,从而实现后续的链式编程{$('body').init().css(xxx)},否则的话将会返回 undefined,无法进行链式编程.

4、jquery 中如何将数组转化为 json 字符串,然后再转化回来?

没有将数组转化为 json 字符串的方法,只有以下方法

原理:先会判断浏览器支不支持 window.JSON.parse 方法,如果支持就直接使用,如果不支持,就是用正则将对象匹配出来

```
$.fn.stringifyArray = function(array) {
    return JSON.stringify(array)
```

```
}
$.fn.parseArray = function(array) {
    return JSON.parse(array)
}
```

然后调用:

```
$("").stringifyArray(array)
```

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

什么是深拷贝?举个例子

```
var result = $.extend(
  {a:1,c:{age:1,name1:'x'}},
  {b:3,c:{age:2,name2:'aa'}}
);
//这种情况result值为{a:1,b:3,c:{age:2,name2:'aa'}}
```

由于 extend 方法会使数据源对象替换掉目标对象同键值的值,所以上面最终的结果会只遍历一层,c:{age:2,name2:'aa'}会直接替换掉 c:{age:1,name1:'x'}

深入拷贝是 extend 的重载方法,只需要在第一个参数传入一个布尔 true,即可深度拷贝.

```
var result = $.extend(
  {a:1,c:{age:1,name1:'x'}},
  {b:3,c:{age:2,name2:'aa'}}
);
//这种情况result值为{a:1,b:3,c:{age:2,name2:'aa'}}
```

深度拷贝遇到键值是对象时,会执行递归,将对象再一次执行 extend 方法.

结论:属性拷贝是通过克隆继承的原理,将数据来源对象遍历到目标对象上,而深度拷贝则是在此基础上执行了递归操作.

6、jquery.extend 与 jquery.fn.extend 的区别?

当两者都有两个以上的参数时,效果是一样的,既把出第一个外的所有参数对象遍历到第一个参数内部.

当只有一个参数时,jquery.extend 是将参数添加在 jquery 的构造函数对象中,而 jquery.fn.extend 是将参数添加在 jquery.prototype 中.

7、谈一下 Jquery 中的 bind(),live(),delegate(),on()的区别?

bind()不支持动态创建出来的元素事件绑定,1.7 版本后被 on 取代

例子:$(xx).bind(‘click’,function(){})

live()可以支持动态创建出来的元素事件绑定

例子:$('a').live('click', fn);

Delegate()支持动态创建出来的元素事件绑定,原因是事件冒泡机制,因为事件是绑定在父元素上,由子元素触发的

例子:$(xx).delegate(‘a’,’ click’,function(){xx 下面所有 a 都触发})

On()综合上述所有优点,可进行事件委托或直接绑定

$(xx).on(‘click’,’ a’,function(xx 下面所有 a 都触发){})

例子:$(xx).on(‘click’,function(xx 触发){})

8、JQuery 一个对象可以同时绑定多个事件,这是如何实现的?

调用内部选择器 selector 方法获取 dom 元素,然后交给 each 方法遍历 dom 元素去执行内部 add 方法添加事件,添加的事件的时候用了 addEventListner 与 attachEvent 兼容处理做事件注册

9、Jquery 与 jQuery UI 有啥区别?

jQuery 是操作 dom 的库,主要提供的功能是选择器,属性修改和事件绑定等等,它能使用户更方便的处理 HTML,event 实现各种效果.

QueryUI 是基于 jQuery 做的一个 UI 组件库,可兼容多种浏览器,里面有很多底层用户交互、动画、特效的可视控件,我们可以直接用它构建良好的交互性 web 应用

JqueryUI 是 jquery 的一个插件,是在 jQuery 的基础上,利用 jQuery 的扩展性,设计的插件.提供了一些常用的界面元素,诸如对话框、拖动行为、改变大小行为等等.jQuery 本身注重于后端操作,没有漂亮的界面,而 jQuery UI 则补充了前者的不足,他提供了华丽的展示界面,使人更容易接受.既有强大的后端操作,又有华丽的前台

10、jQuery 和 Zepto 的区别?各自的使用场景?

jquery 体积较 zepto 大很多,jquery 处理的兼容比 zepto 要好很多,jquery 主要应用于 PC 端,而 zepto 主要用于移动端(因为一般移动端的浏览器都比较新,几乎不用考虑兼容问题,而且 zepto 体积小,更适合移动端加载).

11、针对 jQuery 的优化方法?

1)优先使用 id 选择器,因为 id 选择器是最快的,如果涉及到子元素选择,也建议从最近的 id 开始选择,比如:$("#div1 input"),这样就可以快速从 id 为 div1 的盒子内部查找 input

2)在 class 前面使用 tag 标签,jquery 中第二快的选择器是标签选择器,我们可以使用 tagname 来限制类选择器的范围,如:$("input.on")

3)将 jquery 对象缓存起来,因为每一次$(div)都会 new 一个新的对象

4)尽量减少或压缩 dom 操作的次数,特别注意是 for 循环的内部

5)合理使用冒泡,没有必要为每个元素都绑定事件,只需要向父元素绑定事件然后使用 e.target 去捕捉触发的子元素即可

6)jquery 使用的是$(document).ready,它可以实现在页面渲染时,其他元素还没下载完成就立即执行,如果你发现页面一直是载入中的状态,很有可能是$(document).ready 函数引起的,我们可以通过将 jquery 函数绑定到$(window).load 事件的方法来减少页面载入时 cpu 使用率,他会在所有 html 被下载后才执行

7)给选择器一个上下文,jquery 中的选择器有第二个参数,可以限制选择器在 dom 中的搜索范围,如:$("div","#context")

8)尽量不要使用 live()方法,live()可以给动态添加的 dom 元素添加事件,但他会重新去使用选择器获取所有满足条件的元素然后重新绑定,因此最好是自己手动给新添加的元素绑定事件

12、Zepto 的点透问题如何解决?

点透主要是由于两个 div 重合,例如:一个 div 调用 show(),一个 div 调用 hide();这个时候当点击上面的 div 的时候就会影响到下面的那个 div;

解决办法主要有两种:

引入 fastclick 的库, 并且在 dom ready 时初始化在 body 上

```
$(function(){
    new FastClick(document.body);
})
```

通过事件阻止默认行为

```
$divTapAbove.on('touchend',function(e){ // 改变了事件名称，tap是在body上才被触发，而touchend是原生的事件，在dom本身
    $divTapAbove.hide()
    $output.html($output.html() + 'tap<br/>')
    e.preventDefault(); // 阻止"默认行为"
})
```

13、知道各种 JS 框架(Angular, Backbone, Ember, React, Meteor, Knockout...)么?能讲出他们各自的优点和缺点么?

angular(1.X 版本)优点:

1)模板功能强大丰富,并且是声明式的,自带了丰富的 Angular 指令

2)是一个比较完善的前端 MV*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

3) 自定义 Directive,比 jQuery 插件还灵活,但是需要深入了解 Directive 的一些特性,简单的封装容易,复杂一点官方没有提供详细的介绍文档,我们可以通过阅读源代码来找到某些我们需要的东西,如:在 directive 使用 $parse

4) ng 模块化比较大胆的引入了 Java 的一些东西(依赖注入),能够很容易的写出可复用的代码,对于敏捷开发的团队来说

非常有帮助,我们的项目从上线到目前,UI 变化很大,在摸索中迭代产品,但是 js 的代码基本上很少改动

Angular 缺点:

1)验证功能错误信息显示比较薄弱,需要写很多模板标签,没有 JQuery Validate 方便,所以我们自己封装了验证的错误信息提示,详细参考 why520crazy/w5c-validator-angular • GitHub

2)ngView 只能有一个,不能嵌套多个视图,虽然有 angular-ui/ui-router • GitHub 解决,但是貌似 ui-router 对于 URL 的控制不是很灵活,必须是嵌套式的(也许我没有深入了解或者新版本有改进)

3)ng 提倡在控制器里面不要有操作 DOM 的代码,对于一些 JQuery 插件的使用,如果想不破坏代码的整洁性,需要写一些 directive 去封装一下 JQ 插件,但是现在有很多插件的版本已经支持 Angular 了,如:jQuery File Upload Demo

4)Angular 太笨重了,没有让用户选择一个轻量级的版本,当然 1.2.X 后,Angular 也在做一些更改,比如把 route,animate 等模块独立出去,让用户自己去选择

14、Underscore 对哪些 JS 原生对象进行了扩展以及提供了哪些好用的函数方法?

.each、.map 等,用法例子:

```
_.map([1, 2, 3], function(num){ return num * 3; });
=> [3, 6,9]
```

好用的函数方法_.template,用法例子

```
var compiled = _.template("hello: <%= name %>");
compiled({name: 'moe'});
=> "hello: moe"
```

15、使用过 angular 吗?angular 中的过滤器是干什么用的

"过滤器"是用来过滤变量的值,或者格式化输出,得到自己所期望的结果或格式的东东.

例子:number 过滤器将数字格式化成文本,它的参数是可选的,用来控制小数点后的截取位数,如果传入的是一个非数字字符,会返回空字符串

可以这样使用:{{ 3600 | number:2}},返回结果为:3,600.00

七、移动 web 开发

1、移动端常用类库及优缺点

1. jQuery Mobile 是 jQuery 发布的针对手机和平板设备、经过触控优化的 Web 框架.它基于 jQuery,在不同移动设备平台上可提供统一的用户界面.该框架基于渐近增强技术,并利用 HTML5 和 CSS3 特性.Query Mobile 继承了 jQuery 的优势,并且提供了丰富的适合手机应用的 UI 组件.jQuery Mobile 还有很多的第三方扩展.

2. Zepto.js 是支持移动 WebKit 浏览器的 JavaScript 框架,具有与 jQuery 兼容的语法.相对于 jQuery Mobile 更加轻量级,大小为 2-5k 的库,通过不错的 API 处理绝大多数的基本工作.

3. Sencha Touch 做的 Web App 看起来更像 Native App,用户界面组件和丰富的数据管理,全部基于最新的 HTML 5 和 CSS3 的 WEB 标准,全面兼容 Android 和 iOS 设备.Sencha Touch 提供了超过 50 个组件.

4.GMU(Global Mobile UI)是百度前端通用组开发的移动端组件库,具有代码体积小、简单、易用等特点,组件内部处理了很多移动端的 bug,覆盖机型广,能大大减少开发交互型组件的工作量,非常适合移动端网站项目. 该组件基于 zepto 的 mobile UI 组件库,提供 webapp、pad 端简单易用的 UI 组件!

5. Frozen UI 是腾讯 ISUX 团队(社交用户体验设计团队)根据最新的手机 QQ 设计规范制作的移动端 Web 框架,包括 CSS 基础样式和组件、JavaScript 基础组件和一些动画效果库.

6. Ionic 提供了一个免费且开源的移动优化 HTML,CSS 和 JS 组件库,来构建高交互性应用.基于 Sass 构建和 AngularJS 优化.Ionic 既是一个 CSS 框架也是一个 Javascript UI 库.许多组件需要 Javascript 才能产生神奇的效果,尽管通常组件不需要编码,通过框架扩展可以很容易地使用,比如我们的 AngularIonic 扩展.

7. Junior 为前端框架,用来构建基于 HTML5 的移动 Web 应用,外观与行为跟本地应用相似.它采用针对移动性能优化的 CSS3 转换,支持旋转灯箱效果,包含多样的 Ratchet UI 组件.整个框架使用 Zepto(类似 jQuery 语法的轻量级移动设备 js 类库),且整合了 backbone.js 的视图和路由. Junior 十分易于使用,且提供详细的文档及案例,便于学习.

2、移动端最小触控区域是多大?

IOS 上 44 * 44px

Android 上 48 * 48dp

可以给文字、图标、按钮等可操作元素扩展触控区域, padding、line-height 等都是不错的方式.

3、移动端的点击事件的有延迟,时间是多久,为什么会有?怎么解决这个延时?

click 有 300ms 延迟,为了实现 safari 的双击事件的设计,浏览器要知道你是不是要双击操作,

解决 1:直接禁用缩放<meta name="viewport" content="width=device-width, user-scalable=no">

解决 2:使用 fastclick 库

解决 3: 用 css 设置-ms-touch-action: none,那么对应的元素在被点击之后,浏览器不会启动缩放操作,也就避免了这个 300ms 延迟

八、NodeJs

1、对 Node 的优点和缺点提出了自己的看法

优点:

1) 因为 Node 是基于事件驱动和无阻塞的,所以非常适合处理并发请求,因此构建在 Node 上的代理服务器相比其他技术实现(如 Ruby)的服务器表现要好得多.

2) 与 Node 代理服务器交互的客户端代码是由 javascript 语言编写的,因此客户端和服务器端都用同一种语言编写,这是非常美妙的事情

缺点:

1) Node 是一个相对新的开源项目,所以不太稳定,它总是一直在变.

2)容易写出糟糕的代码,callback 的执行流程有时并不是很符合直觉,需要定期 review 和重构来加以避免.

3) 单线程,要考虑的东西比较多,一旦这个线程死了,就全站死了

2、需求:实现一个页面操作不会整页刷新的网站,并且能在浏览器前进、后退时正确响应.给出你的技术实现方案?

通过 window.location.hash=hash 这个语句来调整地址栏的地址,使得浏览器里边的"前进"、"后退"按钮能正常使用(实质上欺骗了浏览器).然后再根据 hash 值的不同来显示不同的面板(用户可以收藏对应的面板了)

```
<p><input type="text" value="0" id="oTxt" /></p>
<p><input type="button" value="+"  id="oBtn" /></p>
<script>
  var otxt = document.getElementById("oTxt");
  var oBtn = document.getElementById("oBtn");
  var n = 0;
  oBtn.addEventListener("click",function(){
    n++;
    add();
  },false);
  get();
  function add(){
    if("onhashchange" in window){
      window.location.hash = "#"+n;
    }
  }
  function get(){
    if("onhashchange" in window){
      window.addEventListener("hashchange",function(e){
        var hashVal = window.location.hash.substring(1);
        if(hashVal){
          n = hashVal;
        }else{
          n=0;
        }
        otxt.value = n;
      },false);
    }
  }
</script>
```

3、Node.js 的适用场景?

1)实时应用:如 在线聊天,通知推送等

2)分布式应用:通过高效的并行 IO 使用已有的数据

3)工具类应用:海量的,小到前端压缩部署(如 grunt),大到桌面图形应用程序

4)游戏类应用:领域对实时和并发有很高的要求的项目

5)利用稳定接口提升 web 渲染能力

6)前后端编程语言环境统一

4、(如果会用 node)知道 route, middleware, cluster, nodemon, pm2, server-side rendering 么?

Route(路由):设置路由,app.get(‘路由名称’,function(req,res){执行代码});

Middleware(中间件):中间件(middleware)函数能够访问请求对象 (req)、响应对象 (res) 以及应用程序的请求/响应循环中的下一个中间件(middleware)函数.下一个中间件函数通常由名为 next 的变量来表示.中间件最大的好处就是 AOP(面向切面编程)

```
//中间件(middleware)函数能够访问请求对象 (req)、响应对象 (res) 以及应用程序的请求/响应循环中的下一个中间件(middleware)
函数。下一个中间件函数通常由名为 next 的变量来表示。
//next就是下一个middleware
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});

//卧槽，我就是上面那个里面的next
app.use(function (req, res, next) {
  console.log('I am the fucking "next" mentioned above');
  next();
});
```

Cluster:多进程模块

Nodemon:用来监控你 node.js 源代码的任何变化和自动重启你的服务器

pm2:带有负载均衡功能的 Node 应用的进程管理器

server-side:服务器模块,可以实时更新内容

rendering:模板渲染

5、解释一下 Backbone 的 MVC 实现方式?

Backbone 将数据呈现为模型, 你可以创建模型、对模型进行验证和销毁,甚至将它保存到服务器. 当 UI 的变化引起模型属性改变时,模型会触发"change"事件; 所有显示模型数据的视图会接收到该事件的通知,继而视图重新渲染. 你无需查找 DOM 来搜索指定 id 的元素去手动更新 HTML.

6、什么是"前端路由"?什么时候适合使用"前端路由"?"前端路由"有哪些优点和缺点?

前端路由就是把不同路由对应不同的内容或页面的任务交给前端来做,之前是通过服务端根据 url 的不同返回不同的页面实现的.主要在单页面应用,大部分页面结构不变,只改变部分内容的时候使用.

优点:用户体验好,不需要每次都从服务器全部获取,快速展现给用户

缺点:使用浏览器的前进,后退键的时候会重新发送请求,没有合理地利用缓存,单页面无法记住之前滚动的位置,无法在前进,后退的时候记住滚动的位置

八、前端概括性问题

1、常使用的库有哪些?常用的前端开发工具?开发过什么应用或组件?

常用的库:jquery,angular,underscore.js 等

常用的前端开发工具:sublime webstorm 等

开发过插件:jquery 插件,轮播图等

2、对 BFC 规范的理解?

BFC 有一下特性:

1)内部的 Box 会在垂直方向,从顶部开始一个接一个地放置.

2)Box 垂直方向的距离由 margin 决定.属于同一个 BFC 的两个相邻 Box 的 margin 会发生叠加

3)每个元素的 margin box 的左边, 与包含块 border box 的左边相接触(对于从左往右的格式化,否则相反).即使存在浮动也是如此.

4)BFC 的区域不会与 float box 叠加.

5)BFC 就是页面上的一个隔离的独立容器,容器里面的子元素不会影响到外面的元素,反之亦然.

6)计算 BFC 的高度时,浮动元素也参与计算.

那么我们怎样做就可以触发 BFC 呢

float 除了 none 以外的值 overflow 除了 visible 以外的值(hidden,auto,scroll)

display (table-cell,table-caption,inline-block, flex, inline-flex)

position 值为(absolute,fixed) fieldset 元素

fieldset 元素

BFC 可以解决的问题

1)margin 叠加的问题,我们将某个元素放到我们新建的 BFC 里面就可以避免 margin 叠加、

2)对于左右布局的元素,我们可以给右侧的元素添加 overflow:hidden 或者 auto,左侧的是 float:left

3)可以清除浮动,计算 BFC 高度,浮动元素不会撑开父元素的高度,我们可以让父元素触发 BFC,即使用 overflow:hidden

3、99%的网站都需要被重构是那本书上写的?

《应用 web 标准进行设计》

4、WEB 应用从服务器主动推送 Data 到客户端有那些方式?

1)html5 websoket

2)WebSocket 通过 Flash

3)XHR 长时间连接

4)XHR Multipart Streaming

5)不可见的 Iframe

6)<script>标签的长时间连接(可跨域)

5、加班的看法

救急不救穷,如果项目实在是急,是可以加班的

6、平时如何管理你的项目,如何设计突发大规模并发架构?

1)先期团队必须确定好全局样式(globe.css),编码模式(utf-8) 等

2)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);

3)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);

4)页面进行标注(例如 页面 模块 开始和结束);

5)CSS 跟 HTML 分文件夹并行存放,命名都得统一(例如 style.css)

6)JS 分文件夹存放 命民以该 JS 功能为准英文翻译;

7)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

设计突发大规模并发架构需要了解一下几点

1)负载均衡系统:cluster 模块用于方便的创建共享端口的多进程模式

2)反向代理系统: 实现反向代理的静态资源分离,首先是判断 request 来判断是否请求静态文件,如果是请求静态文件,就找到项目下的具体文件,直接用 fs 读取,如果不是请求静态文件那么就用 proxy.web 来请求内部的服务器(我的是 127.0.0.1:81)来实现

3)Web 服务器系统:node 本身就具有高并发的优势

4)数据库集群与分布:mongoDB 的分片技术和 mysql 的读写分离等

7、那些操作会造成内存泄漏?

1)内存泄露是指一块被分配的内存既不能使用,又不能回收,直到浏览器进程结束,由于浏览器垃圾回收方法有 bug,会产生内存泄露.

2)当页面中元素被移除或替换时,若元素绑定的事件仍没被移除,在 IE 中不会作出恰当处理,此时要先手工移除事件,不然会存在内存泄露.

3)在闭包中引入闭包外部的变量时,当闭包结束时此对象无法被垃圾回收(GC).

4)Delete 一个 Object 的属性,而还有其他变量保持这个属性的引用

8、你说你热爱前端,那么应该 WEB 行业的发展很关注吧?说说最近最流行的一些东西吧?

微信小程序、angular2、reactJS 等

9、移动端(比如:Android IOS)怎么做好用户体验?

1)明确重点突出的内容:可以通过小箭头或者悬浮消息来说明功能,不惜一切代价地要让用户更方便地找到他们想要的东西.

2)菜单和导航简单化:传统的桌面网页会在页面顶部突出显示菜单栏.但是在移动设备上这会占用宝贵的屏幕空间.要解决这个问题,可以使用手风琴式下拉菜单,或者在手机屏幕的左上角或右上角显示菜单图标

3)流式布局:移动设备的尺寸种类很多 .但千万不要偷懒只设计了一个 320 像素的宽度.不管你承认与否——176、240、320、360,~480-600(横向)都是常见的设备宽度.灵活和流式的布局方可确保网页正确展示在不同的屏幕尺寸上

4)触屏设计:触屏设计需要关注光标的精确,必须考虑所有形状和大小的手指产生各种类型的压力,在触碰屏幕时产生的不同响应.您需要确保表单、按钮和其他需要触摸或手势输入的元件足够大,以避免与相邻元件的重叠,或者误解触摸事件.

5)表单最小化: 用最少的字段来获取所需要的数据.尽可能预填充有默认值的字段.自动填充常用字段.例如,使用可视化的日历,而不是让用户键入日期.对于那些不止一屏的长表单,最好设置进度条,用来说明用户已经完成了多少步骤,还需要多少步骤才能结束.

6)放弃图片:使用图片来实现特殊效果越少越好,这会占用网站的空间,增加页面的加载时间

7)充分利用手机的特定功能:GPS,Gyrometers 和其他桌面设备不具备的感应器,如"滑动解锁",以及打电话的功能,这些都是你可以利用的手机特定功能,在它们的基础上打破陈规来实现更好的移动体验.

10、你所知道的页面性能优化方法有那些?

1)减少 HTTP 请求

2)减少 DNS 查找

3)避免重定向

4)使用 Ajax 缓存

5)延迟载入组件

6)代码压缩合并

7)预先载入组件

8)减少 DOM 元素数量

9)切分组件到多个域

10)最小化 iframe 的数量

11)不要出现 http 404 错误

11、除了前端以外还了解什么其它技术么?你最最厉害的技能是什么?

个人情况而定,比如喝酒,吹牛逼

12、AMD(Modules/Asynchronous-Definition)、CMD(Common Module Definition)规范区别?

AMD 是 RequireJS 在推广过程中对模块定义的规范化产出.

CMD 是 SeaJS 在推广过程中对模块定义的规范化产出.

这些规范的目的都是为了 JavaScript 的模块化开发,特别是在浏览器端的.

目前这些规范的实现都能达成浏览器端模块化开发的目的

区别:

1)对于依赖的模块,AMD 是提前执行,CMD 是延迟执行.不过 RequireJS 从 2.0 开始,也改成可以延迟执行(根据写法不同,处理方式不同).CMD 推崇 as lazy as possible.

2)CMD 推崇依赖就近,AMD 推崇依赖前置.看代码:

```
// CMD
define(function(require, exports, module) {
    var a = require('./a')
    a.doSomething()
    // 此处略去 100 行
    var b = require('./b') // 依赖可以就近书写
    b.doSomething()
    // ...
})

// AMD 默认推荐的是
define(['./a', './b'], function(a, b) { // 依赖必须一开始就写好
    a.doSomething()
    // 此处略去 100 行
    b.doSomething()
    // ...
})
```

虽然 AMD 也支持 CMD 的写法,同时还支持将 require 作为依赖项传递,但 RequireJS 的作者默认是最喜欢上面的写法,

也是官方文档里默认的模块定义写法.

3)AMD 的 API 默认是一个当多个用,CMD 的 API 严格区分,推崇职责单一.比如 AMD 里,require 分全局 require 和局部 require,都叫 require.CMD 里,没有全局 require,而是根据模块系统的完备性,提供 seajs.use 来实现模块系统的加载启动.CMD 里,每个 API 都简单纯粹.

13、谈谈你认为怎样做能使项目做的更好?

1)了解需求:不要以主观去认为需求,

2)代码的可维护性:便于后期开发和维护,拓展

3)清晰的文件目录:便于查找管理文件,优化 I/O

4)上线前严格的测试:确保上线没问题

5)良好的用户体验

14、你对前端界面工程师这个职位是怎么样理解的?它的前景会怎么样?

理解:前端开发是一项很特殊的工作,前端工程师的工作说得轻松,看似轻巧,但做起来绝对不是那么的简单.在开发过程中涵盖的东西非常宽广,既要从技术的角度来思 考界面的实现,规避技术的死角,又要从用户的角度来思考,怎样才能更好地接受技术呈现的枯燥的数据,更好的呈现信息.简单地说,它的主要职能就将网站的数 据和用户的接受更好地结合在一起,为用户呈现一个友好的数据界面.

前景:前端工程师是一个很新的职业,在国内乃至国际上真正开始受到重视的时间不超过 5 年.互联网的发展速度迅猛,网页由 WEB1.0 到 WEB2.0,再 到新生的 HTML5、CSS3,到现在手机、3G 网络等新科技的兴起,网页也由最原先的图文为主,到现在各种各样的基于哀前端技术实现的应用、交互和富媒 体的呈现,更多的信息、更丰富的内容、更友好的体验,已经成为网站前端开发的要求,网站的前端开发发生了翻天可覆地的变化.网站的开发对前端的需要越来越重要,但目前前端工程师需求远大于供给,前端人才非常紧缺.所以高质量的前端开发工程师将会是后五年内一个非常热门的职业,发展的前景非常可观.

15、php 中哪个函数可以打开一个文件,以对文件进行读和写操作?

```php
<?php
$fp = fopen("test.txt", "w");//文件被清空后再写入
if($fp) {
    $count=0;
    for($i=1;$i<=5;$i++) {
        $flag=fwrite($fp,"行".$i." : "."Hello World!\r\n"); //写入文件
        if(!$flag) {
            echo "写入文件失败<br>";
            break;
        }
        $count+=$flag;
    }
    echo "共写入".$count."个字符";
}else{
    echo "打开文件失败";
}
fclose($fp); //关闭文件操作
?>
```

16、PHP 中 rmdir 可以直接删除文件夹吗?

在 php 中 rmdir 是不能直接删除非空目录的

17、phpinset 和 empty 的区别,举例说明

Isset:查看一个变量是否已经被设置并且不为空

Empty:查看一个变量是否为空 ""、0、"0"、NULL、FALSE、array()、$var

18、php 中$_SERVER 变量中如何得到当前执行脚本路径

```
echo $_SERVER['HTTP_HOST'];        //当前请求的 Host: 头部的内容 即域名信信息
echo $_SERVER['PHP_SELF'];         //当前正在执行脚本的文件相对网站根目录地址,就算该文件被其他文件引用也可以正确得到地址
```

echo $_SERVER['SCRIPT_NAME']; //当前正在执行脚本的文件相对网站根目录地址,但当该文件被其他文件引用时,只显示引用文件的相对地址,不显示该被引用脚本的相对地址.

echo $_SERVER['DOCUMENT_ROOT']; //网站相对服务器地址即网站的绝对路径名 #当前运行脚本所在的文档根目录.在服务器配置文件中定义

echo $_SERVER['SCRIPT_FILENAME'];//当前执行脚本的绝对路径名.

19、写一个 php 函数,要求两个日期字符串的天数差,如 2012-02-05~2012-03-06 的日期差数

```php
<?php

    header("Content-Type:text/html;charset=utf-8");

    $time1 = mktime(10,20,30,11,4,2016); //2016-11-4 10:20:30
    $time2 = mktime(18,30,20,11,5,2016); //2000-11-5 18:30:20
    $diff = (int)(($time2-$time1)/(24*3600));
    echo "时间差为: " . $diff . "天<br>";
?>
```

20、一个衣柜中放了许多杂乱的衬衫,如果让你去整理一下,使得更容易找到你想要的衣服;你会怎么做?请写出你的做法和思路?

1)按颜色整理

2)按春夏秋冬整理

3)按自己喜欢的类型整理

4)按穿着场合整理

21、如何优化网页加载速度?

1)合并 Js 文件和 CSS

将 JS 代码和 CSS 样式分别合并到一个共享的文件,这样不仅能简化代码,而且在执行 JS 文件的时候,如果 JS 文件比较多,就需要进行多次"Get"请求,延长加载速度,将 JS 文件合并在一起后,自然就减少了 Get 请求次数,提高了加载速度.

2)Sprites 图片技术

Spriting 是一种网页图片应用处理方式,它是将一个页面涉及到的所有零星图片都包含到一张大图中去,然后利用 CSS 技术展现出来.这样一来,当访问该页面时,载入的图片就不会像以前那样一幅一幅地慢慢显示出来了,可以减少了整个网页的图片大小,并且利用

3)压缩文本和图片

压缩技术如 gzip 可以有效减少页面加载的时间.包括 HTML,XML,JSON(JavaScript 对象符号),JavaScript 和 CSS 等,压缩率都可以在大小 70%左右.文本压缩用得比较多,一般直接在空间开启就行,而图片的压缩就比较随意,很多都是直接上传,其实还有很大的压缩空间.

4)延迟显示可见区域外的内容

为了确保用户可以更快地看见可见区域的网页可以延迟加载或展现可见区域外的内容,为了避免页面变形,可以使用占位符标签制定正确的高度和宽度.比如 WP 的 jQueryImage LazyLoad 插件就可以在用户停留在第一屏的时候,不加载任何第一屏以下的图片信息,只有当用户把鼠标往下滚动的时候,这些图片才开始加载.这样很明显提升可见区域的加载速度,提高用户体验.

5)确保功能图片优先加载

网站主要考虑可用性的重要性,一个功能按钮要提前加载出来,用户进入下载页,一个只需要 8s 时间的下载花了 5s 在等待、寻找下载按钮图片,谁能忍受?

6)重新布置 Call-to-Action 按钮

其实这个和上面一条是差不多的,都是从用户体验速度着手,跳过了网页的整体加载速度.速度没变,只是让一些行为按钮提前,Call-to-Action 按钮一般习惯设计在页面底部,这样的习惯对于用户来说并不总是好的,购买用户需要等到最下面加载出来才能点击下一步操作.可以调整 CTA 按钮的位置或使用滑动的图片按钮.很多大型购物网站的加入购物车就是这种类型.

7)图片格式优化

不恰当的图像格式是一种极为常见的减慢加载速度的罪魁祸首.正确的图片格式可以让图片缩小数倍,如果保存为最佳格式.可以节省大量带宽,减少处理时间时间,大大加快页面加载速度,这是一种很常见的做法.

8)使用 Progressive JPEGs

ProgressiveJPEGs 图片是 JPEG 格式的一个特殊变种,名为"高级 JPEG".在创建高级 JPEG 文件时,数据是这样安排的:在装入图像时,开始只显示一个模糊的图像,随着数据的装入,图像逐步变得清晰.它相当于交织的 GIF 格式的图片.高级 JPEG 主要是考虑到使用调制解调器的慢速网络而设计的,快速网络的使用者通常不会体会到它和正常 JPEG 格式图片的区别.对于网速比较慢的用户,这无疑有很好的体验.

9)精简代码

这个可以说是最直接的一个方法,也是用得比较多的,对网页代码进行瘦身,删除不必要的沉冗代码,比如不必要的空格、换行符、注释等,包括 JS 代码中的无用代码也需要清除.其中对于注释代码的清除可能有些人存在误区,甚至有的在里面堆砌关键词.

10)延迟加载和执行非必要脚本

网页中有很多脚本是在页面完全加载完前都不需要执行的,可以延迟加载和执行非必要脚本.这些脚本可以在 onload 事件之后执行,避免对网页上重要内容的呈现造成影响.这些脚本可能是你自己网页的甲苯,往往更多的是一些第三方脚本,这样的有很多,比如评论、广告、智能推荐、百度云图、分享等等,这些完全可以等主体内容加载完后再执行.

11)使用 AJAX

AJAX 即"Asynchronous Javascript +XML",是指一种创建交互式网页应用的网页开发技术.通过在后台与服务器进行少量数据交换,AJAX 可以使网页实现异步更新.这意味着可以在不重新加载整个网页的情况下,对网页的某部分进行更新.传统的网页(不使用 AJAX)如果需要更新内容,必须重载整个网页面.

12)自动化的页面性能优化

自动化的页面性能优化也就是借助工具了,网站提速工具有很多,如 RadwareFastView

23、工作流程,你怎么来实现页面设计图,你认为前端应该如何高质量完成工作?

按客户要求设计,视情况而定,一般客户都会有 UI 参考

如何高质量完成工作:

1)用好工具,如办公自动化工具,压缩 css,压缩图片等

2)选择适合自己的编辑器,

3)能熟练的用浏览器进行调试

4)拥有自己的项目模板,以后就可以直接在模板上修改

5)积累一些高质量的常用第三方组件

6)积累些常用的代码片段,比如说使用 sublime 可以按几个键就轻松的调出所需代码

24、介绍项目经验、合作开发、独立开发.

合作开发,使用 github 的流程,如:

项目经理会把项目在 git 中分成两个部分,master 主分支和 programer 分支,我们会下载 master 的代码到我们本地的 master 主分支中,然后在本地又创建一个 programer 分支,当天完成任务以后我们会把 programer 分支的内容合并到当前的主分支中,然后在 push 到项目服务器的 programer 分支中,然后向项目经理申请合并代码,测试员测试完以后,就会把我们的代码合并到项目服务器的 master 中了

25、开发过程中遇到困难,如何解决.

1)百度查找

2)查看论坛有没相关经验的帖子,知乎、cnchina、博客园等

3)加入一些技术大牛群,部分是付费的,但是在相关领域几乎是有问必答