**HTML问题**

## **HTML5语义化**

什么是语义化？就是用合理、正确的标签来展示内容，比如h1~h6定义标题。

### **好处**

易于用户阅读，样式丢失的时候能让页面呈现清晰的结构。

有利于SEO，搜索引擎根据标签来确定上下文和各个关键字的权重。

方便其他设备解析，如盲人阅读器根据语义渲染网页

有利于开发和维护，语义化更具可读性，代码更好维护，与CSS3关系更和谐。

daqianduan.com/6549.htm

## **为什么最好把 CSS 的<link>标签放在<head></head>之间？为什么最好把 JS 的<script>标签恰好放在</body>之前，有例外情况吗？**

**把<link>放在<head>中**

把<link>标签放在<head></head>之间是规范要求的内容。此外，这种做法可以让页面逐步呈现，提高了用户体验。将样式表放在文档底部附近，会使许多浏览器（包括 Internet Explorer）不能逐步呈现页面。一些浏览器会阻止渲染，以避免在页面样式发生变化时，重新绘制页面中的元素。这种做法可以防止呈现给用户空白的页面或没有样式的内容。

**把<script>标签恰好放在</body>之前**

脚本在下载和执行期间会阻止 HTML 解析。把<script>标签放在底部，保证 HTML 首先完成解析，将页面尽早呈现给用户。

例外情况是当你的脚本里包含document.write()时。但是现在，document.write()不推荐使用。同时，将<script>标签放在底部，意味着浏览器不能开始下载脚本，直到整个文档（document）被解析。也许，对此比较好的做法是，<script>使用defer属性，放在<head>中。

## **什么是渐进式渲染（progressive rendering）？**

渐进式渲染是用于提高网页性能（尤其是提高用户感知的加载速度），以尽快呈现页面的技术。

在以前互联网带宽较小的时期，这种技术更为普遍。如今，移动终端的盛行，而移动网络往往不稳定，渐进式渲染在现代前端开发中仍然有用武之地。

一些举例：

图片懒加载——页面上的图片不会一次性全部加载。当用户滚动页面到图片部分时，JavaScript 将加载并显示图像。

确定显示内容的优先级（分层次渲染）——为了尽快将页面呈现给用户，页面只包含基本的最少量的 CSS、脚本和内容，然后可以使用延迟加载脚本或监听DOMContentLoaded/load事件加载其他资源和内容。

异步加载 HTML 片段——当页面通过后台渲染时，把 HTML 拆分，通过异步请求，分块发送给浏览器。更多相关细节可以在这里找到。

## **viewport**

Viewport ：字面意思为视图窗口，在移动web开发中使用。表示将设备浏览器宽度虚拟成一个特定的值（或计算得出），这样利于移动web站点跨设备显示效果基本一致。移动版的 Safari 浏览器最新引进了 viewport 这个 meta tag，让网页开发者来控制 viewport 的大小和缩放，其他手机浏览器也基本支持。

在移动端浏览器当中，存在着两种视口，一种是可见视口（也就是我们说的设备大小），另一种是视窗视口（网页的宽度是多少）。举个例子：如果我们的屏幕是320像素 \* 480像素的大小（iPhone4），假设在浏览器中，320像素的屏幕宽度能够展示980像素宽度的内容。那么320像素的宽度就是可见视口的宽度，而能够显示的980像素的宽度就是视窗视口的宽度。

为了显示更多的内容，大多数的浏览器会把自己的视窗视口扩大，简易的理解，就是让原本320像素的屏幕宽度能够容下980像素甚至更宽的内容（将网页等比例缩小）。

### **Viewport属性值**

width 设置layout viewport 的宽度，为一个正整数，或字符串"width-device"

initial-scale 设置页面的初始缩放值，为一个数字，可以带小数

minimum-scale 允许用户的最小缩放值，为一个数字，可以带小数

maximum-scale 允许用户的最大缩放值，为一个数字，可以带小数

height 设置layout viewport 的高度，这个属性对我们并不重要，很少使用

user-scalable 是否允许用户进行缩放，值为"no"或"yes", no 代表不允许，yes代表允许这些属性可以同时使用，也可以单独使用或混合使用，多个属性同时使用时用逗号隔开就行了。

## **Reflow和Repaint**

### **Reflow**

当涉及到DOM节点的布局属性发生变化时，就会重新计算该属性，浏览器会重新描绘相应的元素，此过程叫Reflow（回流或重排）。

### **Repaint**

当影响DOM元素可见性的属性发生变化 (如 color) 时, 浏览器会重新描绘相应的元素, 此过程称为Repaint（重绘）。因此重排必然会引起重绘。

### **引起Repaint和Reflow的一些操作**

调整窗口大小

字体大小

样式表变动

元素内容变化，尤其是输入控件

CSS伪类激活，在用户交互过程中发生

DOM操作，DOM元素增删、修改

width, clientWidth, scrollTop等布局宽高的计算

### **Repaint和Reflow是不可避免的，只能说对性能的影响减到最小，给出下面几条建议：**

避免逐条更改样式。建议集中修改样式，例如操作className。

避免频繁操作DOM。创建一个documentFragment或div，在它上面应用所有DOM操作，最后添加到文档里。设置display:none的元素上操作，最后显示出来。

避免频繁读取元素几何属性（例如scrollTop）。绝对定位具有复杂动画的元素。

绝对定位使它脱离文档流，避免引起父元素及后续元素大量的回流

harttle.land/2015/08/11  
blueidea.com/tech/web/2

## **img中的alt和元素的title属性作用**

img的alt属性  
如果无法显示图像，浏览器将显示alt指定的内容

元素title属性  
在鼠标移到元素上时显示title的内容

## **href和src区别**

href  
href标识超文本引用，用在link和a等元素上，href是引用和页面关联，是在当前元素和引用资源之间建立联系  
若在文档中添加href ，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 方式加载 CSS，而不是使用 @import 方式。

src  
src表示引用资源，替换当前元素，用在img，script，iframe上，src是页面内容不可缺少的一部分。  
当浏览器解析到src ，会暂停其他资源的下载和处理（图片不会暂停其他资源下载），直到将该资源加载、编译、执行完毕，类似于将所指向资源应用到当前内容。这也是为什么建议把 js 脚本放在底部而不是头部的原因。

blog.csdn.net/lhjuejian

## **浏览器的渲染过程**

解析HTML生成DOM树。

解析CSS生成CSSOM规则树。

将DOM树与CSSOM规则树合并在一起生成渲染树。

遍历渲染树开始布局，计算每个节点的位置大小信息。

将渲染树每个节点绘制到屏幕。

baijiahao.baidu.com/s?

## **为何会出现浏览器兼容问题**

同一产品，版本越老 bug 越多

同一产品，版本越新，功能越多

不同产品，不同标准，不同实现方式

### **处理兼容问题的思路**

要不要做

产品的角度（产品的受众、受众的浏览器比例、效果优先还是基本功能优先）

成本的角度 (有无必要做某件事)

2.做到什么程度 \* 让哪些浏览器支持哪些效果

3..如何做 *根据兼容需求选择技术框架/库(jquery)*根据兼容需求选择兼容工具(html5shiv.js、respond.js、css reset、normalize.css、Modernizr) \* 条件注释、CSS Hack、js 能力检测做一些修补

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。

github.com/jirengu/fron

## **doctype有什么用**

doctype是一种标准通用标记语言的文档类型声明，目的是告诉标准通用标记语言解析器要使用什么样的文档类型定义（DTD）来解析文档。

声明是用来指示web浏览器关于页面使用哪个HTML版本进行编写的指令。声明必须是HTML文档的第一行，位于html标签之前。

浏览器本身分为两种模式，一种是标准模式，一种是怪异模式，浏览器通过doctype来区分这两种模式，doctype在html中的作用就是触发浏览器的标准模式，如果html中省略了doctype，浏览器就会进入到Quirks模式的怪异状态，在这种模式下，有些样式会和标准模式存在差异，而html标准和dom标准值规定了标准模式下的行为，没有对怪异模式做出规定，因此不同浏览器在怪异模式下的处理也是不同的，所以一定要在html开头使用doctype。

## **行内元素和块级元素有哪些**

### **行内元素**

一个行内元素只占据它对应标签的边框所包含的空间  
一般情况下，行内元素只能包含数据和其他行内元素

b, big, i, small, tt  
abbr, acronym, cite, code, dfn, em, kbd, strong, samp, var  
a, bdo, br, img, map, object, q, script, span, sub, sup  
button, input, label, select, textarea

### **块级元素**

占据一整行，高度、行高、内边距和外边距都可以改变，可以容纳块级标签和其他行内标签

header,form,ul,ol,table,article,div,hr,aside,figure,canvas,video,audio,footer

## **iframe框架有那些优缺点**

### **优点：**

iframe能够原封不动的把嵌入的网页展现出来。

如果有多个网页引用iframe，那么你只需要修改iframe的内容，就可以实现调用的每一个页面内容的更改，方便快捷。

网页如果为了统一风格，头部和版本都是一样的，就可以写成一个页面，用iframe来嵌套，可以增加代码的可重用。

如果遇到加载缓慢的第三方内容如图标和广告，这些问题可以由iframe来解决。

### **缺点：**

搜索引擎的爬虫程序无法解读这种页面

框架结构中出现各种滚动条

使用框架结构时，保证设置正确的导航链接。

iframe页面会增加服务器的http请求

## **label标签有什么作用**

label 标签通常是写在表单内，它关联一个控件，使用 label 可以实现点击文字选取对应的控件。

<input type**=**"checkbox" id**=**"test">  
<label for**=**"test" >test</label>

## **HTML5的form如何关闭自动完成功能**

将不想要自动完成的 form 或 input 设置为 autocomplete=off

MDN

## **DOM和BOM有什么区别**

DOM

Document Object Model，文档对象模型

DOM 是为了操作文档出现的 API，document 是其的一个对象

DOM和文档有关，这里的文档指的是网页，也就是html文档。DOM和浏览器无关，他关注的是网页本身的内容。

BOM

Browser Object Model，浏览器对象模型

BOM 是为了操作浏览器出现的 API，window 是其的一个对象

window 对象既为 javascript 访问浏览器提供API，同时在 ECMAScript 中充当 Global 对象

## **CSS问题**

### **CSS 选择器的优先级是如何计算的？**

浏览器通过优先级规则，判断元素展示哪些样式。优先级通过 4 个维度指标确定，我们假定以a、b、c、d命名，分别代表以下含义：

a表示是否使用内联样式（inline style）。如果使用，a为 1，否则为 0。

b表示 ID 选择器的数量。

c表示类选择器、属性选择器和伪类选择器数量之和。

d表示标签（类型）选择器和伪元素选择器之和。

优先级的结果并非通过以上四个值生成一个得分，而是每个值分开比较。a、b、c、d权重从左到右，依次减小。判断优先级时，从左到右，一一比较，直到比较出最大值，即可停止。所以，如果b的值不同，那么c和d不管多大，都不会对结果产生影响。比如0，1，0，0的优先级高于0，0，10，10。

当出现优先级相等的情况时，最晚出现的样式规则会被采纳。如果你在样式表里写了相同的规则（无论是在该文件内部还是其它样式文件中），那么最后出现的（在文件底部的）样式优先级更高，因此会被采纳。

在写样式时，我会使用较低的优先级，这样这些样式可以轻易地覆盖掉。尤其对写 UI 组件的时候更为重要，这样使用者就不需要通过非常复杂的优先级规则或使用!important的方式，去覆盖组件的样式了。

### **参考**

smashingmagazine.com/20

sitepoint.com/web-found

### **重置（resetting）CSS 和 标准化（normalizing）CSS 的区别是什么？你会选择哪种方式，为什么？**

**重置（Resetting）**：重置意味着除去所有的浏览器默认样式。对于页面所有的元素，像margin、padding、font-size这些样式全部置成一样。你将必须重新定义各种元素的样式。

**标准化（Normalizing）**：标准化没有去掉所有的默认样式，而是保留了有用的一部分，同时还纠正了一些常见错误。

当需要实现非常个性化的网页设计时，我会选择重置的方式，因为我要写很多自定义的样式以满足设计需求，这时候就不再需要标准化的默认样式了。

### **参考**

stackoverflow.com/quest

### **请阐述Float定位的工作原理。**

浮动（float）是 CSS 定位属性。浮动元素从网页的正常流动中移出，但是保持了部分的流动性，会影响其他元素的定位（比如文字会围绕着浮动元素）。这一点与绝对定位不同，绝对定位的元素完全从文档流中脱离。

CSS 的clear属性通过使用left、right、both，让该元素向下移动（清除浮动）到浮动元素下面。

如果父元素只包含浮动元素，那么该父元素的高度将塌缩为 0。我们可以通过清除（clear）从浮动元素后到父元素关闭前之间的浮动来修复这个问题。

有一种 hack 的方法，是自定义一个.clearfix类，利用伪元素选择器::after清除浮动。另外还有一些方法，比如添加空的<div></div>和设置浮动元素父元素的overflow属性。与这些方法不同的是，clearfix方法，只需要给父元素添加一个类，定义如下：

.**clearfix**::after {  
 **content**: '';  
 **display**: **block**;  
 **clear**: **both**;  
}

值得一提的是，把父元素属性设置为overflow: auto或overflow: hidden，会使其内部的子元素形成块格式化上下文（Block Formatting Context），并且父元素会扩张自己，使其能够包围它的子元素。

### **参考**

css-tricks.com/all-abou

### **请阐述z-index属性，并说明如何形成层叠上下文（stacking context）。**

CSS 中的z-index属性控制重叠元素的垂直叠加顺序。z-index只能影响position值不是static的元素。

没有定义z-index的值时，元素按照它们出现在 DOM 中的顺序堆叠（层级越低，出现位置越靠上）。非静态定位的元素（及其子元素）将始终覆盖静态定位（static）的元素，而不管 HTML 层次结构如何。

层叠上下文是包含一组图层的元素。在一组层叠上下文中，其子元素的z-index值是相对于该父元素而不是 document root 设置的。每个层叠上下文完全独立于它的兄弟元素。如果元素 B 位于元素 A 之上，则即使元素 A 的子元素 C 具有比元素 B 更高的z-index值，元素 C 也永远不会在元素 B 之上.

每个层叠上下文是自包含的：当元素的内容发生层叠后，整个该元素将会在父层叠上下文中按顺序进行层叠。少数 CSS 属性会触发一个新的层叠上下文，例如opacity小于 1，filter不是none，transform不是none。

### **参考**

css-tricks.com/almanac/

philipwalton.com/articl

developer.mozilla.org/e

### **请阐述块格式化上下文（Block Formatting Context）及其工作原理。**

块格式上下文（BFC）是 Web 页面的可视化 CSS 渲染的部分，是块级盒布局发生的区域，也是浮动元素与其他元素交互的区域。

一个 HTML 盒（Box）满足以下任意一条，会创建块格式化上下文：

float的值不是none.

position的值不是static或relative.

display的值是table-cell、table-caption、inline-block、flex、或inline-flex。

overflow的值不是visible。

在 BFC 中，每个盒的左外边缘都与其包含的块的左边缘相接。

两个相邻的块级盒在垂直方向上的边距会发生合并（collapse）。更多内容请参考边距合并（margin collapsing）。

### **参考**

developer.mozilla.org/e

sitepoint.com/understan

### **有哪些清除浮动的技术，都适用哪些情况？**

空div方法：<div style="clear:both;"></div>。

Clearfix 方法：上文使用.clearfix类已经提到。

overflow: auto或overflow: hidden方法：上文已经提到。

在大型项目中，我会使用 Clearfix 方法，在需要的地方使用.clearfix。设置overflow: hidden的方法可能使其子元素显示不完整，当子元素的高度大于父元素时。

### **请解释什么是雪碧图（css sprites），以及如何实现？**

雪碧图是把多张图片整合到一张上的图片。它被运用在众多使用了很多小图标的网站上（Gmail 在使用）。实现方法：

使用生成器将多张图片打包成一张雪碧图，并为其生成合适的 CSS。

每张图片都有相应的 CSS 类，该类定义了background-image、background-position和background-size属性。

使用图片时，将相应的类添加到你的元素中。

好处：

减少加载多张图片的 HTTP 请求数（一张雪碧图只需要一个请求）。但是对于 HTTP2 而言，加载多张图片不再是问题。

提前加载资源，防止在需要时才在开始下载引发的问题，比如只出现在:hover伪类中的图片，不会出现闪烁。

### **参考**

css-tricks.com/css-spri

### **如何解决不同浏览器的样式兼容性问题？**

在确定问题原因和有问题的浏览器后，使用单独的样式表，仅供出现问题的浏览器加载。这种方法需要使用服务器端渲染。

使用已经处理好此类问题的库，比如 Bootstrap。

使用 autoprefixer 自动生成 CSS 属性前缀。

使用 Reset CSS 或 Normalize.css。

### **如何为功能受限的浏览器提供页面？使用什么样的技术和流程？**

优雅的降级：为现代浏览器构建应用，同时确保它在旧版浏览器中正常运行。

Progressive enhancement - The practice of building an application for a base level of user experience, but adding functional enhancements when a browser supports it.

渐进式增强：构建基于用户体验的应用，但在浏览器支持时添加新增功能。

利用 caniuse.com 检查特性支持。

使用 autoprefixer 自动生成 CSS 属性前缀。

使用 Modernizr进行特性检测。

### **有什么不同的方式可以隐藏内容（使其仅适用于屏幕阅读器）？**

这些方法与可访问性（a11y）有关。

visibility: hidden：元素仍然在页面流中，并占用空间。

width: 0; height: 0：使元素不占用屏幕上的任何空间，导致不显示它。

position: absolute; left: -99999px：将它置于屏幕之外。

text-indent: -9999px：这只适用于block元素中的文本。

Metadata：例如通过使用 Schema.org，RDF 和 JSON-LD。

WAI-ARIA：如何增加网页可访问性的 W3C 技术规范。

即使 WAI-ARIA 是理想的解决方案，我也会采用绝对定位方法，因为它具有最少的注意事项，适用于大多数元素，而且使用起来非常简单。

### **参考**

w3.org/TR/wai-aria-1.1/

developer.mozilla.org/e

a11yproject.com/

### **除了screen，你还能说出一个 @media 属性的例子吗？**

all  
适用于所有设备。

print  
为了加载合适的文档到当前使用的可视窗口. 需要提前咨询 paged media（媒体屏幕尺寸）, 以满足个别设备网页尺寸不匹配等问题。

screen  
主要适用于彩色的电脑屏幕

speech  
解析speech这个合成器. 注意: CSS2已经有一个相似的媒体类型叫aural.  
developer.mozilla.org/z

### **编写高效的 CSS 应该注意什么？**

首先，浏览器从最右边的选择器，即关键选择器（key selector），向左依次匹配。根据关键选择器，浏览器从 DOM 中筛选出元素，然后向上遍历被选元素的父元素，判断是否匹配。选择器匹配语句链越短，浏览器的匹配速度越快。避免使用标签和通用选择器作为关键选择器，因为它们会匹配大量的元素，浏览器必须要进行大量的工作，去判断这些元素的父元素们是否匹配。

BEM (Block Element Modifier) methodology recommends that everything has a single class, and, where you need hierarchy, that gets baked into the name of the class as well, this naturally makes the selector efficient and easy to override. BEM (Block Element Modifier)原则上建议为独立的 CSS 类命名，并且在需要层级关系时，将关系也体现在命名中，这自然会使选择器高效且易于覆盖。

搞清楚哪些 CSS 属性会触发重新布局（reflow）、重绘（repaint）和合成（compositing）。在写样式时，避免触发重新布局的可能。

### **参考**

developers.google.com/w

csstriggers.com/

### **使用 CSS 预处理的优缺点分别是什么？**

优点：

提高 CSS 可维护性。

易于编写嵌套选择器。

引入变量，增添主题功能。可以在不同的项目中共享主题文件。

通过混合（Mixins）生成重复的 CSS。

Splitting your code into multiple files. CSS files can be split up too but doing so will require a HTTP request to download each CSS file.

将代码分割成多个文件。不进行预处理的 CSS，虽然也可以分割成多个文件，但需要建立多个 HTTP 请求加载这些文件。

缺点：

需要预处理工具。

重新编译的时间可能会很慢。

### **对于你使用过的 CSS 预处理，说说喜欢和不喜欢的地方？**

喜欢：

绝大部分优点上题以及提过。

Less 用 JavaScript 实现，与 NodeJS 高度结合。

**Dislikes:**

我通过node-sass使用 Sass，它用 C ++ 编写的 LibSass 绑定。在 Node 版本切换时，我必须经常重新编译。

Less 中，变量名称以@作为前缀，容易与 CSS 关键字混淆，如@media、@import和@font-face。

### **如何实现一个使用非标准字体的网页设计？**

使用@font-face并为不同的font-weight定义font-family。

### **解释浏览器如何确定哪些元素与 CSS 选择器匹配。**

这部分与上面关于编写高效的 CSS 有关。浏览器从最右边的选择器（关键选择器）根据关键选择器，浏览器从 DOM 中筛选出元素，然后向上遍历被选元素的父元素，判断是否匹配。选择器匹配语句链越短，浏览器的匹配速度越快。

例如，对于形如p span的选择器，浏览器首先找到所有<span>元素，并遍历它的父元素直到根元素以找到<p>元素。对于特定的<span>，只要找到一个<p>，就知道'`已经匹配并停止继续匹配。

### **参考**

stackoverflow.com/quest

### **描述伪元素及其用途。**

CSS 伪元素是添加到选择器的关键字，去选择元素的特定部分。它们可以用于装饰（:first-line，:first-letter）或将元素添加到标记中（与 content:...组合），而不必修改标记（:before，:after）。

:first-line和:first-letter可以用来修饰文字。

上面提到的.clearfix方法中，使用clear: both来添加不占空间的元素。

使用:before和after展示提示中的三角箭头。鼓励关注点分离，因为三角被视为样式的一部分，而不是真正的 DOM。如果不使用额外的 HTML 元素，只用 CSS 样式绘制三角形是不太可能的。

### **参考**

css-tricks.com/almanac/

### **说说你对盒模型的理解，以及如何告知浏览器使用不同的盒模型渲染布局。**

CSS 盒模型描述了以文档树中的元素而生成的矩形框，并根据排版模式进行布局。每个盒子都有一个内容区域（例如文本，图像等）以及周围可选的padding、border和margin区域。

CSS 盒模型负责计算：

块级元素占用多少空间。

边框是否重叠，边距是否合并。

盒子的尺寸。

盒模型有以下规则：

块级元素的大小由width、height、padding、border和margin决定。

如果没有指定height，则块级元素的高度等于其包含子元素的内容高度加上padding（除非有浮动元素，请参阅下文）。

如果没有指定width，则非浮动块级元素的宽度等于其父元素的宽度减去父元素的padding。

元素的height是由内容的height来计算的。

元素的width是由内容的width来计算的。

默认情况下，padding和border不是元素width和height的组成部分。

### **参考**

smashingmagazine.com/20

### **\* { box-sizing: border-box; }会产生怎样的效果？**

元素默认应用了box-sizing: content-box，元素的宽高只会决定内容（content）的大小。

box-sizing: border-box改变计算元素width和height的方式，border和padding的大小也将计算在内。

元素的height = 内容（content）的高度 + 垂直方向的padding + 垂直方向border的宽度

元素的width = 内容（content）的宽度 + 水平方向的padding + 水平方向border的宽度

### **display的属性值都有哪些？**

none, block, inline, inline-block, table, table-row, table-cell, list-item.

### **inline和inline-block有什么区别？**

我把block也加入其中，为了获得更好的比较。

| | block | inline-block | inline | | ------------------------------- | ----------------------------------------------------------- | ------------------------------------------ | ------------------------------------------------------------------------------------------------------------------ | | 大小 | 填充其父容器的宽度。| 取决于内容。| 取决于内容。| | 定位 | 从新的一行开始，并且不允许旁边有 HTML 元素（除非是float） | 与其他内容一起流动，并允许旁边有其他元素。| 与其他内容一起流动，并允许旁边有其他元素。| | 能否设置width和height | 能 | 能 | 不能。设置会被忽略。| | 可以使用vertical-align对齐 | 不可以 | 可以 | 可以 | | 边距（margin）和填充（padding） | 各个方向都存在 | 各个方向都存在 | 只有水平方向存在。垂直方向会被忽略。尽管border和padding在content周围，但垂直方向上的空间取决于'line-height' | | 浮动（float） | - | - | 就像一个block元素，可以设置垂直边距和填充。|

### **relative、fixed、absolute和static四种定位有什么区别？**

经过定位的元素，其position属性值必然是relative、absolute、fixed或sticky。

static：默认定位属性值。该关键字指定元素使用正常的布局行为，即元素在文档常规流中当前的布局位置。此时 top, right, bottom, left 和 z-index 属性无效。

relative：该关键字下，元素先放置在未添加定位时的位置，再在不改变页面布局的前提下调整元素位置（因此会在此元素未添加定位时所在位置留下空白）。

absolute：不为元素预留空间，通过指定元素相对于最近的非 static 定位祖先元素的偏移，来确定元素位置。绝对定位的元素可以设置外边距（margins），且不会与其他边距合并。

fixed：不为元素预留空间，而是通过指定元素相对于屏幕视口（viewport）的位置来指定元素位置。元素的位置在屏幕滚动时不会改变。打印时，元素会出现在的每页的固定位置。fixed 属性会创建新的层叠上下文。当元素祖先的 transform 属性非 none 时，容器由视口改为该祖先。

sticky：盒位置根据正常流计算(这称为正常流动中的位置)，然后相对于该元素在流中的 flow root（BFC）和 containing block（最近的块级祖先元素）定位。在所有情况下（即便被定位元素为 table 时），该元素定位均不对后续元素造成影响。当元素 B 被粘性定位时，后续元素的位置仍按照 B 未定位时的位置来确定。position: sticky 对 table 元素的效果与 position: relative 相同。

### **参考**

developer.mozilla.org/e

### **你使用过哪些现有的 CSS 框架？你是如何改进它们的？**

**Bootstrap**：更新周期缓慢。Bootstrap 4 已经处于 alpha 版本将近两年了。添加了在页面中广泛使用的微调按钮组件。

**Semantic UI**：源代码结构使得自定义主题很难理解。非常规主题系统的使用体验很差。外部库的路径需要硬编码（hard code）配置。变量重新赋值没有 Bootstrap 设计得好。

**Bulma**：需要很多非语义的类和标记，显得很多余。不向后兼容，以至于升级版本后，会破坏应用的正常运行。

### **你了解 CSS Flex 和 Grid 吗？**

Flex 主要用于一维布局，而 Grid 则用于二维布局。

### **Flex**

flex容器中存在两条轴， 横轴和纵轴， 容器中的每个单元称为flex item。

在容器上可以设置6个属性： *flex-direction*flex-wrap *flex-flow*justify-content *align-items*align-content

注意：当设置 flex 布局之后，子元素的 float、clear、vertical-align 的属性将会失效。

### **Flex 项目属性**

有六种属性可运用在 item 项目上: 1. order 2. flex-basis 3. flex-grow 4. flex-shrink 5. flex 6. align-self

### **Grid**

CSS网格布局用于将页面分割成数个主要区域，或者用来定义组件内部元素间大小、位置和图层之间的关系。

像表格一样，网格布局让我们能够按行或列来对齐元素。但是，使用CSS网格可能还是比CSS表格更容易布局。例如，网格容器的子元素可以自己定位，以便它们像CSS定位的元素一样，真正的有重叠和层次。

### **响应式设计与自适应设计有何不同？**

响应式设计和自适应设计都以提高不同设备间的用户体验为目标，根据视窗大小、分辨率、使用环境和控制方式等参数进行优化调整。

响应式设计的适应性原则：网站应该凭借一份代码，在各种设备上都有良好的显示和使用效果。响应式网站通过使用媒体查询，自适应栅格和响应式图片，基于多种因素进行变化，创造出优良的用户体验。就像一个球通过膨胀和收缩，来适应不同大小的篮圈。

自适应设计更像是渐进式增强的现代解释。与响应式设计单一地去适配不同，自适应设计通过检测设备和其他特征，从早已定义好的一系列视窗大小和其他特性中，选出最恰当的功能和布局。与使用一个球去穿过各种的篮筐不同，自适应设计允许使用多个球，然后根据不同的篮筐大小，去选择最合适的一个。

### **参考**

developer.mozilla.org/e

mediumwell.com/responsi

css-tricks.com/the-diff

### **你有没有使用过视网膜分辨率的图形？当中使用什么技术？**

我倾向于使用更高分辨率的图形（显示尺寸的两倍）来处理视网膜显示。更好的方法是使用媒体查询，像@media only screen and (min-device-pixel-ratio: 2) { ... }，然后改变background-image。

对于图标类的图形，我会尽可能使用 svg 和图标字体，因为它们在任何分辨率下，都能被渲染得十分清晰。

还有一种方法是，在检查了window.devicePixelRatio的值后，利用 JavaScript 将<img>的src属性修改，用更高分辨率的版本进行替换。

### **参考**

sitepoint.com/css-techn

### **什么情况下，用translate()而不用绝对定位？什么时候，情况相反。**

translate()是transform的一个值。改变transform或opacity不会触发浏览器重新布局（reflow）或重绘（repaint），只会触发复合（compositions）。而改变绝对定位会触发重新布局，进而触发重绘和复合。transform使浏览器为元素创建一个 GPU 图层，但改变绝对定位会使用到 CPU。因此translate()更高效，可以缩短平滑动画的绘制时间。

当使用translate()时，元素仍然占据其原始空间（有点像position：relative），这与改变绝对定位不同。

### **参考**

paulirish.com/2012/why-

### **其他答案**

neal.codes/blog/front-e

quizlet.com/28293152/fr

peterdoes.it/2015/12/03

### **行内元素、块级元素区别**

行内元素：和其他元素都在一行上，高度、行高及外边距和内边距都不可改变，文字图片的宽度不可改变，只能容纳文本或者其他行内元素；其中img是行元素

块级元素：总是在新行上开始，高度、行高及外边距和内边距都可控制，可以容纳内敛元素和其他元素；行元素转换为块级元素方式：display：block；

### **一边固定宽度一边宽度自适应**

可以使用flex布局 复制下面的HTML和CSS代码 用浏览器打开可以看到效果

<div class="wrap">  
 <div class="div1"></div>  
 <div class="div2"></div>  
</div>  
  
.wrap {  
 display: flex;  
 justify-content: space-between;  
}  
.div1 {  
 min-width: 200px;  
}  
.div2 {  
 width: 100%;  
 background: #e6e6e6;  
}  
html,  
body,  
div {  
 height: 100%;  
 margin: 0;  
}

### **水平垂直居中的方式**

### **flex**

// 父容器  
display: flex;  
justify-content: center;  
align-items: center;

### **position**

// 父容器  
position: relative;  
  
// 子容器  
position:absolute;  
margin:auto;  
top:0;  
bottom:0;  
left:0;  
right:0;

### **position+transform**

// 父容器  
position: relative;  
  
// 子容器  
position: absolute;  
top: 50%;  
left: 50%;  
transform: translate(-50%, -50%);

### **table-cell**

<div class="box">  
 <div class="content">  
 <div class="inner"></div>  
 </div>  
</div>  
  
html, body {  
 height: 100%;  
 width: 100%;  
 margin: 0;  
}  
.box {  
 display: table;  
 height: 100%;  
 width: 100%;  
}  
.content {  
 display: table-cell;  
 vertical-align: middle;  
 text-align: center;  
}  
.inner {  
 background-color: #000;  
 display: inline-block;  
 width: 200px;  
 height: 200px;  
}

### **display:none、visibile:hidden、opacity:0的区别**

| |是否隐藏 |是否在文档中占用空间|是否会触发事件| |-|-|-|-| |display: none|是|否|否| |visibile: hidden|是|是|否| |opacity: 0|是|是|是|

### **CSS中link和@import的区别**

link属于HTML标签，而@import是CSS提供的

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

import只在IE5以上才能识别，而link是HTML标签，无兼容问题

link方式的样式的权重 高于@import的权重

### **如何用css实现瀑布流布局**

利用column-count和break-inside这两个CSS3属性即可，复制如下代码即可察看效果

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="utf-8">  
 <style>  
 body {  
 margin: 0;  
 }  
 .waterfall-container {  
 /\*分几列\*/  
 column-count: 2;  
 width: 100%;  
 /\* 列间距 \*/  
 column-gap: 10px;  
 }  
  
 .waterfall-item {  
 break-inside: avoid;  
 width: 100%;  
 height: 100px;  
 margin-bottom: 10px;  
 background: #ddd;  
 column-gap: 0;  
 text-align: center;  
 color: #fff;  
 font-size: 40px;  
 }  
 </style>  
</head>  
<body>  
 <div class="waterfall-container">  
 <div class="waterfall-item" style="height: 100px">1</div>  
 <div class="waterfall-item" style="height: 300px">2</div>  
 <div class="waterfall-item" style="height: 400px">3</div>  
 <div class="waterfall-item" style="height: 100px">4</div>  
 <div class="waterfall-item" style="height: 300px">5</div>  
 <div class="waterfall-item" style="height: 600px">6</div>  
 <div class="waterfall-item" style="height: 400px">7</div>  
 <div class="waterfall-item" style="height: 300px">8</div>  
 <div class="waterfall-item" style="height: 700px">9</div>  
 <div class="waterfall-item" style="height: 100px">10</div>  
 </div>  
</body>  
</html>

## **文本超出部分显示省略号**

### **单行**

overflow: hidden;  
text-overflow:ellipsis;  
white-space: nowrap;

### **多行**

display: -webkit-box;  
-webkit-box-orient: vertical;  
-webkit-line-clamp: 3; // 最多显示几行  
overflow: hidden;

## **利用伪元素画三角**

.info-tab {  
 position: relative;  
}  
.info-tab::after {  
 content: '';  
 border: 4px solid transparent;  
 border-top-color: #2c8ac2;  
 position: absolute;  
 top: 0;  
}

## **已知父级盒子的宽高，子级img宽高未知，想让img铺满父级盒子且图片不能变形**

需要用到css的object-fit属性

div {  
 **width**: 200**px**;  
 **height**: 200**px**;  
}  
img {  
 **object-fit**: **cover**;  
 **width**: 100**%**;  
 **height**: 100**%**;  
}

MDN

## **iframe的作用**

iframe是用来在网页中插入第三方页面，早期的页面使用iframe主要是用于导航栏这种很多页面都相同的部分，这样在切换页面的时候避免重复下载。

优点 1. 便于修改，模拟分离，像一些信息管理系统会用到。2. 但现在基本不推荐使用。除非特殊需要，一般不推荐使用。

缺点 1. iframe的创建比一般的DOM元素慢了1-2个数量级 2. iframe标签会阻塞页面的的加载，如果页面的onload事件不能及时触发，会让用户觉得网页加载很慢，用户体验不好，在Safari和Chrome中可以通过js动态设置iframe的src属性来避免阻塞。3. iframe对于SEO不友好，替换方案一般就是动态语言的Incude机制和ajax动态填充内容等。

## **css hack是什么**

由于不同的浏览器，比如Internet Explorer 6,Internet Explorer 7,Mozilla Firefox等，对CSS的解析认识不一样，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。

这个针对不同的浏览器写不同的CSS code的过程，就叫CSS hack,也叫写CSS hack。

具体请看：  
cnblogs.com/Renyi-Fan/p

## **过渡与动画的区别是什么**

transition  
可以在一定的时间内实现元素的状态过渡为最终状态，用于模拟以一种过渡动画效果，但是功能有限，只能用于制作简单的动画效果而动画属性

animation  
可以制作类似Flash动画，通过关键帧控制动画的每一步，控制更为精确，从而可以制作更为复杂的动画。

## **什么是外边距合并**

外边距合并指的是，当两个垂直外边距相遇时，它们将形成一个外边距。

合并后的外边距的高度等于两个发生合并的外边距的高度中的较大者。

## **去除inline-block元素间间距的方法**

移除空格

使用margin负值

使用font-size:0

letter-spacing

word-spacing

更详细的介绍请看:去除inline-block元素间间距的N种方法

## **JavaScript问题**

## **同源策略**

同源策略可防止 JavaScript 发起跨域请求。源被定义为 URI、主机名和端口号的组合。此策略可防止页面上的恶意脚本通过该页面的文档对象模型，访问另一个网页上的敏感数据。

## **跨域**

原因  
浏览器的同源策略导致了跨域

作用  
用于隔离潜在恶意文件的重要安全机制

解决

jsonp ，允许 script 加载第三方资源

反向代理（nginx 服务内部配置 Access-Control-Allow-Origin \*）

cors 前后端协作设置请求头部，Access-Control-Allow-Origin 等头部信息

iframe 嵌套通讯，postmessage

zhuanlan.zhihu.com/p/41  
跨域资源共享 CORS 阮一峰

## **JSONP**

这是我认为写得比较通俗易懂的一篇文章 直接转载过来  
blog.csdn.net/hansexplo

## **域名收敛**

PC 时代为了突破浏览器的域名并发限制。有了域名发散。  
浏览器有并发限制，是为了防止DDOS攻击。  
域名收敛：就是将静态资源放在一个域名下。减少DNS解析的开销。  
域名发散：是将静态资源放在多个子域名下，就可以多线程下载，提高并行度，使客户端加载静态资源更加迅速。  
域名发散是pc端为了利用浏览器的多线程并行下载能力。而域名收敛多用与移动端，提高性能，因为dns解析是是从后向前迭代解析，如果域名过多性能会下降，增加DNS的解析开销。

## **事件绑定的方式**

嵌入dom

<button onclick="func()">按钮</button>

直接绑定

btn.onclick = function(){}

事件监听

btn.addEventListener('click',function(){})

## **事件委托**

事件委托利用了事件冒泡，只指定一个事件处理程序，就可以管理某一类型的所有事件。所有用到按钮的事件（多数鼠标事件和键盘事件）都适合采用事件委托技术， 使用事件委托可以节省内存。

<ul>  
 <li>苹果</li>  
 <li>香蕉</li>  
 <li>凤梨</li>  
</ul>  
  
// good  
document.querySelector('ul').onclick = (event) => {  
 let target = event.target  
 if (target.nodeName === 'LI') {  
 console.log(target.innerHTML)  
 }  
}  
  
// bad  
document.querySelectorAll('li').forEach((e) => {  
 e.onclick = function() {  
 console.log(this.innerHTML)  
 }  
})

## **事件循环**

事件循环是一个单线程循环，用于监视调用堆栈并检查是否有工作即将在任务队列中完成。如果调用堆栈为空并且任务队列中有回调函数，则将回调函数出队并推送到调用堆栈中执行。

## **事件模型**

DOM0  
直接绑定

<input onclick="sayHi()"/>  
  
btn.onclick = function() {}  
btn.onclick = null

DOM2  
DOM2级事件可以冒泡和捕获 通过addEventListener绑定 通过removeEventListener解绑

// 绑定  
btn.addEventListener('click', sayHi)  
// 解绑  
btn.removeEventListener('click', sayHi)

DOM3  
DOM3具有更多事件类型 DOM3级事件在DOM2级事件的基础上添加了更多的事件类型，全部类型如下：

UI事件，当用户与页面上的元素交互时触发，如：load、scroll  
焦点事件，当元素获得或失去焦点时触发，如：blur、focus  
鼠标事件，当用户通过鼠标在页面执行操作时触发如：dbclick、mouseup  
滚轮事件，当使用鼠标滚轮或类似设备时触发，如：mousewheel  
文本事件，当在文档中输入文本时触发，如：textInput  
键盘事件，当用户通过键盘在页面上执行操作时触发，如：keydown、keypress  
合成事件，当为IME（输入法编辑器）输入字符时触发，如：compositionstart  
变动事件，当底层DOM结构发生变化时触发，如：DOMsubtreeModified

jianshu.com/p/3acdf5f71

## **如何自定义事件**

原生提供了3个方法实现自定义事件

createEvent，设置事件类型，是 html 事件还是 鼠标事件

initEvent 初始化事件，事件名称，是否允许冒泡，是否阻止自定义事件

dispatchEvent 触发事件

MDN

## **target和currentTarget区别**

event.target  
返回触发事件的元素

event.currentTarget  
返回绑定事件的元素

## **prototype和proto的关系是什么**

所有的对象都拥有**proto**属性，它指向对象构造函数的prototype属性

let obj = {}  
obj.\_\_proto\_\_ === Object.prototype // true  
  
function Test(){}  
test.\_\_proto\_\_ == Test.prototype // true

所有的函数都同时拥有**proto**和protytpe属性 函数的**proto**指向自己的函数实现 函数的protytpe是一个对象 所以函数的prototype也有**proto**属性 指向Object.prototype

function func() {}  
func.prototype.\_\_proto\_\_ === Object.prototype // true

Object.prototype.**proto**指向null

Object.prototype.\_\_proto\_\_ // null

## **原型继承**

所有的JS对象都有一个prototype属性，指向它的原型对象。当试图访问一个对象的属性时，如果没有在该对象上找到，它还会搜寻该对象的原型，以及该对象的原型的原型，依次层层向上搜索，直到找到一个名字匹配的属性或到达原型链的末尾。

## **继承**

JS高程第3版 第6章 继承 寄生组合式继承

function SuperType(name) {  
 this.name = name  
 this.colors = ['red']  
}  
  
SuperType.prototype.sayName = function() {  
 console.log(this.name)  
}  
// 继承实例属性  
function SubType(name, age) {  
 SuperType.call(this, name)  
 this.age = age  
}  
  
function inheritPrototype(subType, superType) {  
 let prototype = Object.create(superType.prototype)  
 prototype.constructor = subType  
 subType.prototype = prototype  
}  
// 继承原型方法  
inheritPrototype(SubType, SuperType)  
  
// 定义自己的原型方法  
SubType.prototype.sayAge = function() {  
 console.log(this.age)  
}

## **闭包**

闭包是指有权访问另一个函数作用域中的变量的函数。

function sayHi(name) {  
 return () => {  
 console.log(`Hi! ${name}`)  
 }  
}  
const test = sayHi('xiaoming')  
test() // Hi! xiaoming

虽然sayHi函数已经执行完毕，但是其活动对象也不会被销毁，因为test函数仍然引用着sayHi函数中的变量name，这就是闭包。  
但也因为闭包引用着另一个函数的变量，导致另一个函数已经不使用了也无法销毁，所以闭包使用过多，会占用较多的内存，这也是一个副作用。

## **有一个函数，参数是一个函数，返回值也是一个函数，返回的函数功能和入参的函数相似，但这个函数只能执行3次，再次执行无效，如何实现**

这个题目是考察闭包的使用

**function** sayHi() {  
 console.log('hi')  
}  
  
**function** threeTimes(fn) {  
 **let** times **=** 0  
 **return** () => {  
 **if** (times**++** **<** 3) {  
 fn()  
 }  
 }  
}  
  
**const** newFn **=** threeTimes(sayHi)  
newFn()  
newFn()  
newFn()  
newFn()  
newFn() *// 后面两次执行都无任何反应*

通过闭包变量 times 来控制函数的执行

## **实现add函数,让add(a)(b)和add(a,b)两种调用结果相同**

function add(a, b) {  
 if (b === undefined) {  
 return function(x) {  
 return a + x  
 }  
 }  
  
 return a + b  
}

## **Ajax**

Ajax(asynchronous JavaScript and XML)是使用客户端上的许多 Web 技术，创建异步 Web 应用的一种 Web 开发技术。借助 Ajax，Web 应用可以异步（在后台）向服务器发送数据和从服务器检索数据，而不会干扰现有页面的显示和行为。通过将数据交换层与表示层分离，Ajax 允许网页和扩展 Web 应用程序动态更改内容，而无需重新加载整个页面。实际上，现在通常将 JSON 替换为 XML，因为 JavaScript 对 JSON 有原生支持优势。  
XMLHttpRequest API 经常用于异步通信。此外还有最近流行的fetch API。

let xmlhttp  
if (window.XMLHttpRequest) {  
 // IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码  
 xmlhttp = new XMLHttpRequest()  
} else {  
 // IE6, IE5 浏览器执行代码  
 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP")  
}  
xmlhttp.onreadystatechange = () => {  
 if (xmlhttp.readyState === 4 && xmlhttp.status === 200) {  
 document.getElementById("myDiv").innerHTML = xmlhttp.responseText  
 }  
}  
xmlhttp.open("GET", "/ajax/test.txt", true)  
xmlhttp.send()

## **使用Ajax的优缺点分别是什么**

**优点**

交互性更好。来自服务器的新内容可以动态更改，无需重新加载整个页面。

减少与服务器的连接，因为脚本和样式只需要被请求一次。

状态可以维护在一个页面上。JavaScript 变量和 DOM 状态将得到保持，因为主容器页面未被重新加载。

基本上包括大部分 SPA 的优点。

**缺点**

动态网页很难收藏。

如果 JavaScript 已在浏览器中被禁用，则不起作用。

有些网络爬虫不执行 JavaScript，也不会看到 JavaScript 加载的内容。

基本上包括大部分 SPA 的缺点。

github.com/yangshun/fro

## **Ajax和Fetch区别**

ajax是使用XMLHttpRequest对象发起的，但是用起来很麻烦，所以ES6新规范就有了fetch，fetch发一个请求不用像ajax那样写一大堆代码。

使用fetch无法取消一个请求，这是因为fetch基于Promise，而Promise无法做到这一点。

在默认情况下，fetch不会接受或者发送cookies

fetch没有办法原生监测请求的进度，而XMLHttpRequest可以

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

fetch由于是ES6规范，兼容性上比不上XMLHttpRequest

## **变量提升**

var会使变量提升，这意味着变量可以在声明之前使用。let和const不会使变量提升，提前使用会报错。变量提升（hoisting）是用于解释代码中变量声明行为的术语。使用var关键字声明或初始化的变量，会将声明语句“提升”到当前作用域的顶部。但是，只有声明才会触发提升，赋值语句（如果有的话）将保持原样。

## **使用let、var和const创建变量有什么区别**

用var声明的变量的作用域是它当前的执行上下文，它可以是嵌套的函数，也可以是声明在任何函数外的变量。let和const是块级作用域，意味着它们只能在最近的一组花括号（function、if-else 代码块或 for 循环中）中访问。

**function** foo() {  
 *// 所有变量在函数中都可访问* **var** bar **=** 'bar';  
 **let** baz **=** 'baz';  
 **const** qux **=** 'qux';  
  
 console.log(bar); *// bar* console.log(baz); *// baz* console.log(qux); *// qux*}  
  
console.log(bar); *// ReferenceError: bar is not defined*console.log(baz); *// ReferenceError: baz is not defined*console.log(qux); *// ReferenceError: qux is not defined***if** (**true**) {  
 **var** bar **=** 'bar';  
 **let** baz **=** 'baz';  
 **const** qux **=** 'qux';  
}  
  
*// 用 var 声明的变量在函数作用域上都可访问*console.log(bar); *// bar  
// let 和 const 定义的变量在它们被定义的语句块之外不可访问*console.log(baz); *// ReferenceError: baz is not defined*console.log(qux); *// ReferenceError: qux is not defined*

var会使变量提升，这意味着变量可以在声明之前使用。let和const不会使变量提升，提前使用会报错。

console.log(foo); *// undefined*  
**var** foo **=** 'foo';  
  
console.log(baz); *// ReferenceError: can't access lexical declaration 'baz' before initialization*  
**let** baz **=** 'baz';  
  
console.log(bar); *// ReferenceError: can't access lexical declaration 'bar' before initialization*  
**const** bar **=** 'bar';

用var重复声明不会报错，但let和const会。

**var** foo **=** 'foo';  
**var** foo **=** 'bar';  
console.log(foo); *// "bar"*  
**let** baz **=** 'baz';  
**let** baz **=** 'qux'; *// Uncaught SyntaxError: Identifier 'baz' has already been declared*

let和const的区别在于：let允许多次赋值，而const只允许一次。

*// 这样不会报错。***let** foo **=** 'foo';  
foo **=** 'bar';  
  
*// 这样会报错。***const** baz **=** 'baz';  
baz **=** 'qux';

github.com/yangshun/fro

## **对象浅拷贝和深拷贝有什么区别**

在 JS 中，除了基本数据类型，还存在对象、数组这种引用类型。基本数据类型，拷贝是直接拷贝变量的值，而引用类型拷贝的其实是变量的地址。

let o1 = {a: 1}  
let o2 = o1

在这种情况下，如果改变 o1 或 o2 其中一个值的话，另一个也会变，因为它们都指向同一个地址。

o2.a = 3  
console.log(o1.a) // 3

而浅拷贝和深拷贝就是在这个基础之上做的区分，如果在拷贝这个对象的时候，只对基本数据类型进行了拷贝，而对引用数据类型只是进行了引用的传递，而没有重新创建一个新的对象，则认为是浅拷贝。反之，在对引用数据类型进行拷贝的时候，创建了一个新的对象，并且复制其内的成员变量，则认为是深拷贝。

## **怎么实现对象深拷贝**

let o1 = {a:{  
 b:1  
 }  
}  
let o2 = JSON.parse(JSON.stringify(o1))

另一种方法

function deepCopy(s) {  
 const d = {}  
 for (let k in s) {  
 if (typeof s[k] == 'object') {  
 d[k] = deepCopy(s[k])  
 } else {  
 d[k] = s[k]  
 }  
 }  
  
 return d  
}

## **数组去重**

ES5

function unique(arry) {  
 const temp = []  
 arry.forEach(e => {  
 if (temp.indexOf(e) == -1) {  
 temp.push(e)  
 }  
 })  
  
 return temp  
}

ES6

function unique (arr) {  
 return Array.from(new Set(arr))  
}

## **数据类型**

Undefined

Null

Boolean

Number

String

Object

symbol(ES6新增)

## **内置函数(原生函数)**

String

Number

Boolean

Object

Function

Array

Date

RegExp

Error

Symbol

原始值 "I am a string" 并不是一个对象，它只是一个字面量，并且是一个不可变的值。如果要在这个字面量上执行一些操作，比如获取长度、访问其中某个字符等，那需要将其 转换为 String 对象。幸好，在必要时语言会自动把字符串字面量转换成一个 String 对象，也就是说你并不需要 显式创建一个对象。

## **如何判断数组与对象**

Array.isArray([]) *// true*Array.isArray({}) *// false*  
**typeof** [] *// "object"***typeof** {} *// "object"*  
Object.prototype **==** [].\_\_proto\_\_ *// false*Object.prototype **==** {}.\_\_proto\_\_ *// true*Array.prototype **==** [].\_\_proto\_\_ *// true*Array.prototype **==** {}.\_\_proto\_\_ *// false*

## **自动分号**

有时 JavaScript 会自动为代码行补上缺失的分号，即自动分号插入（Automatic SemicolonInsertion，ASI）。  
因为如果缺失了必要的 ; ，代码将无法运行，语言的容错性也会降低。ASI 能让我们忽略那些不必要的 ; 。  
请注意，ASI 只在换行符处起作用，而不会在代码行的中间插入分号。  
如果 JavaScript 解析器发现代码行可能因为缺失分号而导致错误，那么它就会自动补上分 号。并且，只有在代码行末尾与换行符之间除了空格和注释之外没有别的内容时，它才会 这样做。

## **浮点数精度**

css88.com/archives/7340

## **cookie、localStorage、sessionStorage区别**

|特性 | cookie | localStorage | sessionStorage| |-|-|-|-| |由谁初始化 | 客户端或服务器，服务器可以使用Set-Cookie请求头。| 客户端 | 客户端 | |数据的生命周期|一般由服务器生成，可设置失效时间，如果在浏览器生成，默认是关闭浏览器之后失效 |永久保存，可清除 | 仅在当前会话有效，关闭页面后清除| |存放数据大小|4KB|5MB|5MB| |与服务器通信|每次都会携带在HTTP头中，如果使用cookie保存过多数据会带来性能问题|仅在客户端保存|仅在客户端保存| |用途|一般由服务器生成，用于标识用户身份|用于浏览器缓存数据|用于浏览器缓存数据| | 访问权限 | 任意窗口 | 任意窗口 | 当前页面窗口 |

## **自执行函数?用于什么场景？好处?**

自执行函数:1、声明一个匿名函数2、马上调用这个匿名函数。  
作用：创建一个独立的作用域。

好处：防止变量弥散到全局，以免各种js库冲突。隔离作用域避免污染，或者截断作用域链，避免闭包造成引用变量无法释放。利用立即执行特性，返回需要的业务函数或对象，避免每次通过条件判断来处理

场景：一般用于框架、插件等场景

## **多个页面之间如何进行通信**

有如下几个方式： *cookie*web worker \* localeStorage和sessionStorage

## **css动画和js动画的差异**

代码复杂度，js 动画代码相对复杂一些

动画运行时，对动画的控制程度上，js 能够让动画，暂停，取消，终止，css动画不能添加事件

动画性能看，js 动画多了一个js 解析的过程，性能不如 css 动画好

zhuanlan.zhihu.com/p/41

## **如何实现文件断点续传**

断点续传最核心的内容就是把文件“切片”然后再一片一片的传给服务器，但是这看似简单的上传过程却有着无数的坑。

首先是文件的识别，一个文件被分成了若干份之后如何告诉服务器你切了多少块，以及最终服务器应该如何把你上传上去的文件进行合并，这都是要考虑的。

因此在文件开始上传之前，我们和服务器要有一个“握手”的过程，告诉服务器文件信息，然后和服务器约定切片的大小，当和服务器达成共识之后就可以开始后续的文件传输了。

前台要把每一块的文件传给后台，成功之后前端和后端都要标识一下，以便后续的断点。

当文件传输中断之后用户再次选择文件就可以通过标识来判断文件是否已经上传了一部分，如果是的话，那么我们可以接着上次的进度继续传文件，以达到续传的功能。有了HTML5 的 File api之后切割文件比想想的要简单的多的多。

只要用slice 方法就可以了

var packet = file.slice(start, end);

参数start是开始切片的位置，end是切片结束的位置 单位都是字节。通过控制start和end 就可以是实现文件的分块

如

file.slice(0,1000);  
file.slice(1000,2000);  
file.slice(2000,3000);  
// ......

在把文件切成片之后，接下来要做的事情就是把这些碎片传到服务器上。如果中间掉线了，下次再传的时候就得先从服务器获取上一次上传文件的位置，然后以这个位置开始上传接下来的文件内容。

cnblogs.com/zhwl/p/3580

## **new一个对象经历了什么**

function Test(){}  
const test = new Test()

创建一个新对象：

const obj = {}

设置新对象的constructor属性为构造函数的名称，设置新对象的**proto**属性指向构造函数的prototype对象

obj.constructor = Test  
obj.\_\_proto\_\_ = Test.prototype

使用新对象调用函数，函数中的this被指向新实例对象

Test.call(obj)

将初始化完毕的新对象地址，保存到等号左边的变量中

## **bind、call、apply的区别**

call和apply其实是一样的，区别就在于传参时参数是一个一个传或者是以一个数组的方式来传。  
call和apply都是在调用时生效，改变调用者的this指向。

let name = 'Jack'  
const obj = {name: 'Tom'}  
function sayHi() {console.log('Hi! ' + this.name)}  
  
sayHi() // Hi! Jack  
sayHi.call(obj) // Hi! Tom

bind也是改变this指向，不过不是在调用时生效，而是返回一个新函数。

const newFunc = sayHi.bind(obj)  
newFunc() // Hi! Tom

## **请简述JavaScript中的this。**

JS 中的this是一个相对复杂的概念，不是简单几句能解释清楚的。粗略地讲，函数的调用方式决定了this的值。我阅读了网上很多关于this的文章，Arnav Aggrawal 写的比较清楚。this取值符合以下规则：

在调用函数时使用new关键字，函数内的this是一个全新的对象。

如果apply、call或bind方法用于调用、创建一个函数，函数内的 this 就是作为参数传入这些方法的对象。

当函数作为对象里的方法被调用时，函数内的this是调用该函数的对象。比如当obj.method()被调用时，函数内的 this 将绑定到obj对象。

如果调用函数不符合上述规则，那么this的值指向全局对象（global object）。浏览器环境下this的值指向window对象，但是在严格模式下('use strict')，this的值为undefined。

如果符合上述多个规则，则较高的规则（1 号最高，4 号最低）将决定this的值。

如果该函数是 ES2015 中的箭头函数，将忽略上面的所有规则，this被设置为它被创建时的上下文。

想获得更深入的解释，请查看他在 Medium 上的文章。

github.com/yangshun/fro

## **如何确定this指向**

如果要判断一个运行中函数的 this 绑定，就需要找到这个函数的直接调用位置。找到之后就可以顺序应用下面这四条规则来判断 this 的绑定对象。1. 由 new 调用？绑定到新创建的对象。2. 由 call 或者 apply （或者 bind ）调用？绑定到指定的对象。3. 由上下文对象调用？绑定到那个上下文对象。4. 默认：在严格模式下绑定到 undefined ，否则绑定到全局对象。

一定要注意，有些调用可能在无意中使用默认绑定规则。如果想“更安全”地忽略 this 绑定，你可以使用一个 DMZ 对象，比如 ø = Object.create(null) ，以保护全局对象。  
ES6 中的箭头函数并不会使用四条标准的绑定规则，而是根据当前的词法作用域来决定this ，具体来说，箭头函数会继承外层函数调用的 this 绑定（无论 this 绑定到什么）。这其实和 ES6 之前代码中的 self = this 机制一样

参考：《你不知道的JavaScript》

## **== 和 ===的区别是什么**

==是抽象相等运算符，而===是严格相等运算符。==运算符是在进行必要的类型转换后，再比较。===运算符不会进行类型转换，所以如果两个值不是相同的类型，会直接返回false。使用==时，可能发生一些特别的事情，例如：

1 **==** '1'; *// true*1 **==** [1]; *// true*1 **==** **true**; *// true*0 **==** ''; *// true*0 **==** '0'; *// true*0 **==** **false**; *// true*

如果你对==和===的概念不是特别了解，建议大多数情况下使用===

## **箭头函数和普通函数有什么区别**

函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象，用callapplybind也不能改变this指向

不可以当作构造函数，也就是说，不可以使用new命令，否则会抛出一个错误。

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

不可以使用yield命令，因此箭头函数不能用作 Generator 函数。

箭头函数没有原型对象prototype

## **首屏时间、白屏时间**

Performance 接口可以获取到当前页面中与性能相关的信息。  
该类型的对象可以通过调用只读属性 Window.performance 来获得。  
白屏时间：

performance.timing.responseStart - performance.timing.navigationStart

首屏时间

window.onload = () => {  
 new Date() - performance.timing.responseStart  
}

developer.mozilla.org/z

## **当你在浏览器输入一个地址后发生了什么**

github.com/skyline75489

## **页面大量图片，如何优化加载，优化用户体验**

图片懒加载。在页面的未可视区域添加一个滚动事件，判断图片位置与浏览器顶端的距离与页面的距离，如果前者小于后者，优先加载。

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

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

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

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

jianshu.com/p/5d82bba9e

## **js网络请求性能优化之防抖与节流**

防抖(debounce)  
在函数需要频繁触发时，只有当有足够空闲的时间时，才执行一次。就好像在百度搜索时，每次输入之后都有联想词弹出，这个控制联想词的方法就不可能是输入框内容一改变就触发的，他一定是当你结束输入一段时间之后才会触发。

节流(thorttle)  
预定一个函数只有在大于等于执行周期时才执行，周期内调用不执行。就好像你在淘宝抢购某一件限量热卖商品时，你不断点刷新点购买，可是总有一段时间你点上是没有效果，这里就用到了节流，就是怕点的太快导致系统出现bug。

区别  
在发生持续触发事件时，防抖设置事件延迟并在空闲时间去触发事件，而节流则是隔一定的时间触发一次。

具体请看：  
blog.csdn.net/jacoox/ar

## **如何做到修改url参数页面不刷新**

HTML5引入了 history.pushState() 和 history.replaceState() 方法，它们分别可以添加和修改历史记录条目。

**let** stateObj **=** {  
 foo**:** "bar",  
};  
  
history.pushState(stateObj, "page 2", "bar.html");

假设当前页面为 foo.html，执行上述代码后会变为 bar.html，点击浏览器后退，会变为 foo.html，但浏览器并不会刷新。 pushState() 需要三个参数: 一个状态对象, 一个标题 (目前被忽略), 和 (可选的) 一个 URL. 让我们来解释下这三个参数详细内容：

状态对象 — 状态对象 state 是一个 JavaScript 对象，通过 pushState () 创建新的历史记录条目。无论什么时候用户导航到新的状态，popstate 事件就会被触发，且该事件的 state 属性包含该历史记录条目状态对象的副本。状态对象可以是能被序列化的任何东西。原因在于 Firefox 将状态对象保存在用户的磁盘上，以便在用户重启浏览器时使用，我们规定了状态对象在序列化表示后有640k的大小限制。如果你给 pushState() 方法传了一个序列化后大于 640k 的状态对象，该方法会抛出异常。如果你需要更大的空间，建议使用 sessionStorage 以及 localStorage.

标题 — Firefox 目前忽略这个参数，但未来可能会用到。传递一个空字符串在这里是安全的，而在将来这是不安全的。二选一的话，你可以为跳转的 state 传递一个短标题。

URL — 该参数定义了新的历史URL记录。注意，调用 pushState() 后浏览器并不会立即加载这个 URL，但可能会在稍后某些情况下加载这个 URL，比如在用户重新打开浏览器时。新URL不必须为绝对路径。如果新URL是相对路径，那么它将被作为相对于当前 URL 处理。新 URL 必须与当前URL同源，否则 pushState() 会抛出一个异常。该参数是可选的，缺省为当前 URL。

## **格式化金钱，每千分位加逗号**

**function** format(str) {  
 **let** s **=** ''  
 **let** count **=** 0  
 **for** (**let** i **=** str.length **-** 1; i **>=** 0; i**--**) {  
 s **=** str[i] **+** s  
 count**++**  
 **if** (count **%** 3 **==** 0 **&&** i **!=** 0) {  
 s **=** ',' **+** s  
 }  
 }  
 **return** s  
}  
**function** format(str) {  
 **return** str.replace(/(\d)(?=(?:\d{3})+$)/g, '$1,')  
}

## **请用js去除字符串空格**

### **去除所有空格**

str.replace(/\s/g, '')

### **去除两边空格**

str.replace(/^\s+|\s+$/g, '')  
// 原生方法  
str.trim()

## **HTTP问题**

## **RESTful**

REST 指的是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是 RESTful。

GET  
get方法在Rest中主要用于获取资源，能够发送参数，不过有限制，且参数都会以?开头的形 式附加在URL尾部。规范的get方法处理器应该是幂等的，也就是说对一个资源不论发送多少次get请求都不会更改数据或造成破坏。

POST  
post方法在Rest请求中主要用于添加资源，参数信息存放在请求报文的消息体中相对安全，且可发送较大信息

PUT  
put方法在Rest中主要用于更新资源，因为大多数浏览器不支持put和delete，会自动将put和delete请求转化为get和post. 因此为了使用put和delete方法, 需要以post发送请求，在表单中使用隐藏域发送真正的请求。put方法的参数是同post一样是存放在消息中的，同样具有安全性，可发送较大信息。put方法是幂等的，对同一URL资源做出的同一数据的任意次put请求其对数据的改变都是一致的。

DELETE  
Delete在Rest请求中主要用于删除资源，因为大多数浏览器不支持put和delete，会自动将put和delete请求转化为get和post。因此为了使用put和delete方法,需要以post发送请求，在表单中使用隐藏域发送真正的请求。Delete方法的参数同post一样存放在消息体中,具有安全性，可发送较大信息 Delete方法是幂等的，不论对同一个资源进行多少次delete请求都不会破坏数据

blog.csdn.net/jnshu\_it/

## **GET和POST的区别**

GET产生一个TCP数据包；POST产生两个TCP数据包。

GET在浏览器回退时是无害的，而POST会再次提交请求。

GET产生的URL地址可以被Bookmark，而POST不可以。

GET请求会被浏览器主动cache，而POST不会，除非手动设置。

GET请求只能进行url编码，而POST支持多种编码方式。

GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

GET请求在URL中传送的参数是有长度限制的，而POST么有。

对参数的数据类型，GET只接受ASCII字符，而POST没有限制。

GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。

GET参数通过URL传递，POST放在Request body中。

## **Accept和Content-Type**

Accept 请求头用来告知客户端可以处理的内容类型，这种内容类型用MIME类型来表示。服务器使用 Content-Type 应答头通知客户端它的选择。

Accept: text/html  
Accept: image/\*  
Accept: text/html, application/xhtml+xml, application/xml;q=0.9, \*/\*;q=0.8

1.Accept属于请求头， Content-Type属于实体头。  
Http报头分为通用报头，请求报头，响应报头和实体报头。  
请求方的http报头结构：通用报头|请求报头|实体报头  
响应方的http报头结构：通用报头|响应报头|实体报头

2.Accept代表发送端（客户端）希望接受的数据类型。  
比如：Accept：text/xml;  
代表客户端希望接受的数据类型是xml类型

Content-Type代表发送端（客户端|服务器）发送的实体数据的数据类型。  
比如：Content-Type：text/html;  
代表发送端发送的数据格式是html。

二者合起来，  
Accept:text/xml；  
Content-Type:text/html  
即代表希望接受的数据类型是xml格式，本次请求发送的数据的数据格式是html。

## **状态码**

| 状态码 | 类别 | 描述 | | -- | -- | -- | | 1xx | Informational（信息状态码） | 接受请求正在处理 | | 2xx | Success（成功状态码） | 请求正常处理完毕 | | 3xx | Redirection（重定向状态码） | 需要附加操作已完成请求 | | 4xx | Client Error（客户端错误状态码） | 服务器无法处理请求 | | 5xx | Server Error（服务器错误状态码） | 服务器处理请求出错 |

## **HTTP缓存**

segmentfault.com/a/1190

## **如何处理不让别人盗用你的图片，访问你的服务器资源**

http header, 对refer做判断看来源是不是自己的网站，如果不是就拒绝

通过session校验，如果不通过特定服务生成cookie和session就不能请求得到资源

## **Http与Https的区别**

HTTP 的URL 以http:// 开头，而HTTPS 的URL 以https:// 开头

HTTP 是不安全的，而 HTTPS 是安全的

HTTP 标准端口是80 ，而 HTTPS 的标准端口是443

在OSI 网络模型中，HTTP工作于应用层，而HTTPS 的安全传输机制工作在传输层

HTTP 无法加密，而HTTPS 对传输的数据进行加密

HTTP无需证书，而HTTPS 需要CA机构wosign的颁发的SSL证书

zhuanlan.zhihu.com/p/33

## **什么是Http协议无状态协议?怎么解决Http协议无状态协议?**

无状态协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息也就是说，  
当客户端一次HTTP请求完成以后，客户端再发送一次HTTP请求，HTTP并不知道当前客户端是一个”老用户“。

可以使用Cookie来解决无状态的问题，Cookie就相当于一个通行证，第一次访问的时候给客户端发送一个Cookie，  
当客户端再次来的时候，拿着Cookie(通行证)，那么服务器就知道这个是”老用户“。

zhuanlan.zhihu.com/p/33

## **常用的HTTP方法有哪些**

GET：用于请求访问已经被URL（统一资源标识符）识别的资源，可以通过URL传参给服务器。

POST：用于传输信息给服务器，主要功能与Get方法类似，但一般推荐POST方式。

PUT：传输文件，报文主体包含文件内容，保存到对应URL位置。

HEAD：获取报文首部，与GET方法类似，只是不返回报文主体，一般用于验证URL是否有效。

DELET：删除文件，与PUT方法相反，删除对应URL位置的文件。OPTIONS：查询相应URL支持的HTTP方法。

## **一次完整的HTTP请求所经历的7个步骤**

HTTP通信机制是在一次完整的HTTP通信过程中，Web浏览器与Web服务器之间将完成下列7个步骤：

建立TCP连接

在HTTP工作开始之前，Web浏览器首先要通过网络与Web服务器建立连接，该连接是通过TCP来完成的，该协议与IP协议共同构建 Internet，即著名的TCP/IP协议族，因此Internet又被称作是TCP/IP网络。HTTP是比TCP更高层次的应用层协议，根据规则， 只有低层协议建立之后才能，才能进行更层协议的连接，因此，首先要建立TCP连接，一般TCP连接的端口号是80。

Web浏览器向Web服务器发送请求行

一旦建立了TCP连接，Web浏览器就会向Web服务器发送请求命令。例如：GET /sample/hello.jsp HTTP/1.1。

Web浏览器发送请求头

浏览器发送其请求命令之后，还要以头信息的形式向Web服务器发送一些别的信息，之后浏览器发送了一空白行来通知服务器，它已经结束了该头信息的发送。

Web服务器应答

客户机向服务器发出请求后，服务器会客户机回送应答， HTTP/1.1 200 OK ，应答的第一部分是协议的版本号和应答状态码。

Web服务器发送应答头

正如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档。

Web服务器向浏览器发送数据

Web服务器向浏览器发送头信息后，它会发送一个空白行来表示头信息的发送到此为结束，接着，它就以Content-Type应答头信息所描述的格式发送用户所请求的实际数据。

Web服务器关闭TCP连接

一般情况下，一旦Web服务器向浏览器发送了请求数据，它就要关闭TCP连接，然后如果浏览器或者服务器在其头信息加入了这行代码：

Connection:keep-alive

TCP连接在发送后将仍然保持打开状态，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

建立TCP连接->发送请求行->发送请求头->（到达服务器）发送状态行->发送响应头->发送响应数据->断TCP连接

juejin.im/post/5a8102e0

## **Vue问题**

## **什么是mvvm**

MVVM最早由微软提出来，它借鉴了桌面应用程序的MVC思想，在前端页面中，把Model用纯JavaScript对象表示，View负责显示，两者做到了最大限度的分离 把Model和View关联起来的就是ViewModel。  
ViewModel负责把Model的数据同步到View显示出来，还负责把View的修改同步回Model  
View 和 Model 之间的同步工作完全是自动的，无需人为干涉（由viewModel完成，在这里指VUE）  
因此开发者只需关注业务逻辑，不需要手动操作DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理

### **ViewModel如何编写？**

需要用JavaScript编写一个通用的ViewModel，这样，就可以复用整个MVVM模型了

一个MVVM框架和jQuery操作DOM相比有什么区别？我们先看用jQuery实现的修改两个DOM节点的例子：

<!-- HTML -->  
<p>Hello, <span id="name">Bart</span>!</p>  
<p>You are <span id="age">12</span>.</p>  
  
Hello, Bart!  
  
You are 12.

用jQuery修改name和age节点的内容：

var name = 'Homer';  
var age = 51;  
  
$('#name').text(name);  
$('#age').text(age);

如果我们使用MVVM框架来实现同样的功能，我们首先并不关心DOM的结构，而是关心数据如何存储。最简单的数据存储方式是使用JavaScript对象：

var person = {  
 name: 'Bart',  
 age: 12  
}

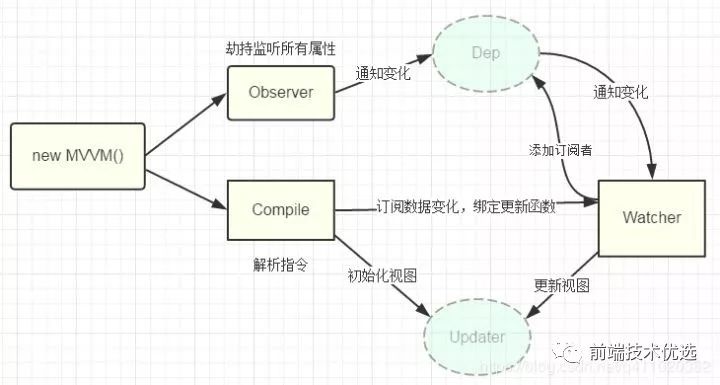
我们把变量person看作Model，把HTML某些DOM节点看作View，并假定它们之间被关联起来了。

要把显示的name从Bart改为Homer，把显示的age从12改为51，我们并不操作DOM，而是直接修改JavaScript对象：

person.name = 'Homer';  
person.age = 51;

执行上面的代码，我们惊讶地发现，改变JavaScript对象的状态，会导致DOM结构作出对应的变化！这让我们的关注点从如何操作DOM变成了如何更新JavaScript对象的状态，而操作JavaScript对象比DOM简单多了！

这就是MVVM的设计思想：关注Model的变化，让MVVM框架去自动更新DOM的状态，从而把开发者从操作DOM的繁琐步骤中解脱出来！下图可以很好的解释view viewModel model之间的关系



## **Vue的优点是什么**

mvvm的优点即是vue的优点，在这里再总结一下：  
数据和视频之间的同步工作完全是自动的，无需人为干涉，所以开发者只需关注业务逻辑，不需要手动操作DOM, 不需要关注数据状态的同步问题， 复杂的数据状态维护完全由 MVVM 来统一管理，节省了很多精力。

## **对于生命周期的理解**

创建一个Vue实例，是一个漫长的过程，要经历初始化，数据合并，模板解析，数据渲染等等一系列过程。所以，为了能实现在这个过程里面插入自己想要提前做的事情，就有了生命周期钩子函数。

### **举个例子：**

一辆公交车，从出发点A站到终点站B，中间有很多站点，公交车每到一个站点，就得停下来，  
等待客人上车，然后再驶往下一个站点，一直到终点站为止。  
A和B之间的站点，就像是这个路程的生命周期。每一个站点都是一个不同的生命周期（站点名不同），  
只要到了站点，就得执行该站点对应的生命周期函数，  
只不过每个站点的生命周期函数都是一样的（等待客人上车）。

Vue中的生命周期也是一样，对应了Vue实例从创建到结束之间的每一个过程。例如，Vue的beforeCreate周期，指的就是Vue在实例初始化之后，数据观测 (data observer) 和 event/watcher 事件配置之前被调用。

至于Vue具体的生命周期函数有哪些，请看官网API文档

## **组件传值**

父组件通过prop向子组件传值

子组件通过事件向父组件传值

子组件与子组件之间不能直接传值，需要通过父组件来做间接传值，在这种情况下推荐使用vuex

具体例子请看官方文档

## **vue数据绑定原理**

Vue的数据双向绑定都是依据Object.defineProperty()这一方法来做的  
Object.defineProperty到底有什么作用呢？MDN

Object.defineProperty(obj, prop, descriptor)  
  
obj  
要在其上定义属性的对象。  
  
prop  
要定义或修改的属性的名称。  
  
descriptor  
将被定义或修改的属性描述符。

简单来说 这个方法可以定义一个对象某个属性的描述符

我们需要用到的就是描述符当中的getter和setter

const obj = {a:1}  
obj.a // 1  
  
obj.a = 2

像上面代码中的两个操作 读取和赋值 就是在访问obj.a的getter和setter  
当我们输入obj.a时 就是在访问obj对象a属性的getter 当输入obj.a = 2 时就是在访问obj对象a属性的setter

Object.defineProperty(obj, 'a', {  
 get : function(){  
 return val  
 },  
 set : function(newValue){  
 val = newValue  
 },  
 enumerable : true,  
 configurable : true  
})

getter和setter都是一个函数 我们还可以这样做 例如

get: function() {  
 // 每次访问obj.a时都会执行这段代码  
 console.log('hello, 你在读取a的值')  
 return val  
}  
set: function(newValue) {  
 val = newValue  
 // 每次给obj.a赋值时都会执行这段代码  
 console.log('你设置了a的值')  
}

Vue的双向数据绑定就是根据上面的原理来实现的 只要在读取值时收集观察者 在赋值时触发观察者更新函数 就可以实现数据变更 从而实现DOM重新渲染

说到这可能还不是很明白 不要急 慢慢来 先看一下这段代码 复制放到HTML文件里自己运行一下 然后打开网页 在控制台里输入data.user.name看看 会有惊喜

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="utf-8">  
 <title>动态数据绑定（一）</title>  
</head>  
<body>  
 <script>  
 var data = {  
 user: {  
 name: 'xiaoming',  
 age: 18,  
 occupation: 'frontend'  
 },  
 address: {  
 city: 'shaoguan'  
 }  
 };  
 function Observer(data) {  
 this.data = data;  
 this.walk(data);  
 }  
 Observer.prototype = {  
 walk: function(obj) {  
 var value,  
 key;  
 for (key in obj) {  
 if (obj.hasOwnProperty(key)) {  
 value = obj[key];  
 if (typeof value === 'object') {  
 new Observer(value);  
 }  
 this.convert(key, value);  
 }  
 }  
 },  
 convert: function(key, value) {  
 Object.defineProperty(this.data, key, {  
 get : function(){  
 console.log("你访问了" + key);  
 return value;  
 },  
 set : function(newValue){  
 value = newValue;  
 console.log('你设置了' + key + '=' + value);  
 }  
 });  
 }  
 }  
 var example = new Observer(data);  
 </script>  
</body>  
</html>

## **vue-router原理**

说简单点，vue-router的原理就是通过对URL地址变化的监听，继而对不同的组件进行渲染。  
每当URL地址改变时，就对相应的组件进行渲染。原理是很简单，实现方式可能有点复杂，主要有hash模式和history模式。  
如果想了解得详细点，建议百度或者阅读源码。

## **vuex原理**

vuex的原理其实非常简单，它为什么能实现所有的组件共享同一份数据？  
因为vuex生成了一个store实例，并且把这个实例挂在了所有的组件上，所有的组件引用的都是同一个store实例。  
store实例上有数据，有方法，方法改变的都是store实例上的数据。由于其他组件引用的是同样的实例，所以一个组件改变了store上的数据， 导致另一个组件上的数据也会改变，就像是一个对象的引用。  
如果对vuex的实现有兴趣，可以看看我自己造的一个vue轮子对应的vuex插件。它实现了除vuex模块外的所有功能。

## **v-if和v-show的区别**

v-if 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

v-if 也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用v-if 较好。

cn.vuejs.org/v2/guide/c

## **vue怎么实现页面的权限控制**

利用 vue-router 的 beforeEach 事件，可以在跳转页面前判断用户的权限（利用 cookie 或 token），是否能够进入此页面，如果不能则提示错误或重定向到其他页面，在后台管理系统中这种场景经常能遇到。

## **keep-alive有什么作用**

在 Vue 中，每次切换组件时，都会重新渲染。如果有多个组件切换，又想让它们保持原来的状态，避免重新渲染，这个时候就可以使用 keep-alive。 keep-alive 可以使被包含的组件保留状态，或避免重新渲染。

## **计算属性有什么作用**

先来看一下计算属性的定义：  
当其依赖的属性的值发生变化的时，计算属性会重新计算。反之则使用缓存中的属性值。  
计算属性和vue中的其它数据一样，都是响应式的，只不过它必须依赖某一个数据实现，并且只有它依赖的数据的值改变了，它才会更新。

## **$route和$router的区别**

$route 是路由信息对象，包括path，params，hash，query，fullPath，matched，name 等路由信息参数。

而 $router 是路由实例对象，包括了路由的跳转方法，钩子函数等

## **watch的作用是什么**

watch 主要作用是监听某个数据值的变化。和计算属性相比除了没有缓存，作用是一样的。

借助 watch 还可以做一些特别的事情，例如监听页面路由，当页面跳转时，我们可以做相应的权限控制，拒绝没有权限的用户访问页面。