

前端开打掌握的知识内容概要

HTML/CSS:

对 web 标准的理解(结构/表现/行为)、浏览器内核、渲染原理、依赖管理、兼容性、CSS 语法、层次关系、常用属性、布局、选择器、权重、盒模型、Hack、CSS 预处理器、Flexbox、CSS Modules、Document flow、BFC、HTML5 (离线&储存&history、多媒体、webGL、SVG、Cavas)

Javascript:

数据类型、运算、对象、function、继承、闭包、作用域、事件、prototype、RegExp、JSON、Ajax、DOM、BOM、内存泄漏、跨域、异步请求、模板引擎、模块化、Flux、同构、算法、ECMAScript6、Nodejs、HTTP

其他:

主流 MVVM 框架(React\Vue\Angular)、Hybrid App\React Native\Weex、TypeScript、RESTFul、WEB 安全、前端工程化、依赖管理、性能优化、重构、团队协作、可维护、易用性、SEO、UED、前端技术选型、快速学习能力等；

本面试题仅供参考，还是推荐学员用自己的方式去理解，然后用自己的语言表达出来！！

HTML:

Doctype 的作用？

Doctype 是作用于文档最顶部的文档声明，是告诉浏览器是以标准模式还是以怪异模式展示该页面。Doctype 不存在或格式错误都会导致页面以怪异模式展示页面。

标准模式和怪异模式的区别：

标准模式的页面排版和 JS 运作模式都是浏览器支持的最高标准，而怪异模式是向后兼容，模拟老浏览器模式行为，防止页面无法正常工作

行内元素/块级元素/空元素有哪些？

行内元素: a/img/span/b/strong/input/select/section

块级元素: div/p/table/ul/ol/li/h1-h6

空元素: br/hr/img/input/link/meta

介绍一下你对浏览器内核的理解？

浏览器主要分为两个部分:渲染引擎和 JS 引擎

渲染引擎：主要负责获取页面内容和排版渲染页面

JS 引擎：解析和执行 JS 来实现页面的动态效果，以及交互内容

常用浏览器的内核有哪些？

Trident内核：IE,MaxThon,TT,The World,360,搜狗浏览器等。[又称MSHTML]
Gecko内核：Netscape6及以上版本，FF,MozillaSuite/SeaMonkey等
Presto内核：Opera7及以上。 [Opera内核原为：Presto，现为：Blink;]
Webkit内核：Safari,Chrome等。 [Chrome的：Blink (WebKit的分支)]

详细文章：[浏览器内核的解析和对比](<http://www.cnblogs.com/fullhouse/archive/2011/12/19/2293455.html>)

浏览器是怎么对 HTML5 的离线储存资源进行管理和加载的？

在线的情况下，浏览器发现 html 标签有 manifest 属性，它会请求 manifest 文件

如果是第一次访问 app，那么浏览器就会根据 manifest 文件的内容下载相应的资源并且进行离线存储

如果已经访问过 app 且资源已经离线存储了，浏览器会对比新的 manifest 文件与旧的 manifest 文件，如果文件没有发生改变，就不做任何操作。如果文件改变了，那么就会重新下载文件中的资源并进行离线存储

离线的情况下，浏览器就直接使用离线存储的资源。

描述一下 cookies/sessionStorage 和 localStorage 的区别？

cookies 是网站为了表示用户身份而储存在用户本地终端上的数据,Cookies 的数据始终在同源的 http 请求中携带,会在浏览器和服务端中来回传递,大小不

能 4K(通常经过加密,所以不用担心账号被盗,同源策略[同源是指"协议+域名+端口"三者相同]可以防止 XSS 和 CSRF 攻击浏览器,XSS 就是用过浏览器的 cookies,截取用户数据,CSRF 是模拟用户在网页上面的操作,完成数据请求.异步策略牵扯到了 JSONP)

sessionStorage 和 localStorage 的数据都是在本地存储,不会把数据发给服务器,localStorage 是关闭浏览器,数据还存在不会丢失,而 sessionStorage 是离开浏览器后,数据会自动删除.

HTML5 新特性有哪些？如何处理 HTML5 新标签的兼容性问题？如何区分 HTML 和 HTML5？

HTML5 新特性：

绘图方面：加入了 canvas 绘图和 SVG 绘图；

媒体方面： 加入了 video 和 audio 标签

语义化标签： 比如 header、nav、footer、section ['sekʃ(ə)n]、article ['artɪkl]

本地离线存储： localStorage['ləʊʊkl] 和 sessionStorage 两种本地离线缓存

localStorage 是长期储存数据,关闭浏览器后数据不会丢失

sessionStorage 是关闭浏览器后数据自动删除

表单控件： calendar、date、time、email、url、search；

以及一些新技术: webworker / websocket (săkit)/ GelolCation(,jēōlō' kăSHən)

如何区分 HTML 和 HTML5: 通过 Doctype 声明/新增的结构元素/功能元素

简述一下你对 HTML 语义化的理解？

用正确的标签做正确的事情，html 语义化让页面的内容结构更加简单易懂，便于搜索引擎解析，便于阅读维护和理解

HTML5 离线缓存怎么使用，工作原理能不能解释一下？

用户在没有联网时，可以正常访问站点或应用，等用户联网时，更新用户的缓存文件

浏览器是怎么对 HTML5 进行离线缓存资源进行管理和加载的?: 在联网情况下，html 头部有 manifest 属性，会请求 manifest 文件，如果是第一次访问，浏览器会根据 manifest 的内容下载相应的资源并且进行离线缓存，如果不是第一个，会加载成为新的 manifest 文件，新旧 manifest 文件对比，如果一致，则不发生变化，如果不一致，那么会重新下载文件中的资源并进行离线缓存

页面导入样式时,使用 link 和@import 有什么区别？

1/link 属于 XHTML 标签,除了加载 CSS 之外还能用于定义 RSS,@import 是 CSS 提供的,只能用于加载 CSS

2/link 加载的文件,在页面加载的时候,link 文件会同时加载,而@import 引入

的 CSS 文件,是页面在加载完成后再加载的

3/@import 有兼容性问题,IE5 以下的浏览器是无法识别的,而 link 无兼容性问题

Iframe 有哪些缺点?

Iframe 会阻碍页面的 onload 事件

浏览器的搜索引擎一般读无法解读 iframe 页面,不利于 SEO 的搜索

Iframe 和主页面共享链接池,会影响页面的并行加载

使用 js 动态添加 iframe src 属性,可以避免一三问题

Label 的作用是什么?怎么用?

```
<label for="Name">Number:</label>
<input type="text" name="Name" id="Name"/>

<label>Date:<input type="text" name="B"/></label>
```

label 标签是定义表单控制间的关系,当用户点击 label 里面的文字时,浏览器会自动把光标转载表单控件上

HTML5 的 form 如何关闭自动完成功能?

给不想要提示 form 或某个 input 设置 autocomplete = off

如何实现浏览器内多个标签之间的通信?

Websocket[`'sockit'`]/SharedWorker 都是可以将不同线程共享为一个线程,他们的数据也是共享的(没怎么用过,用法不太清楚)

LocalStorage 也可以实现浏览器多个标签页之间的通信

localStorage 在另一个浏览器被添加/删除/修改时,会触发一个事件,我们可以通过对 localStorage 监听事件,控制他的值来进行信息通信

页面可见性有哪些用途?(visibility API)

可以通过 visibilityState 检测当前页是否可见,以及打开网页的时间,可以控制页面在被切换后,停止视频和音频的播放

如何在页面上实现一个圆形的可点击区域?

Border-radius

或者 js 实现,要求一个点在不在圆上的算法,获取鼠标坐标等

网页验证码是干嘛的,解决了什么安全问题?

区分用户是计算机还是人的公告全自动程序,可以防止恶意破解密码,刷票等

Title 和 h1 的区别,b 与 strong 的区别,i 和 em 的区别?

title 属性没有明确的标题,只是 HTML 语义化的一个标签,而 h1 则是层次明确的标题,h1 标签里的文字,字体较大,并且会加粗

b 与 strong 都有加粗字体的作用,strong 只是更加语义化,是加重语气的意思

i 和 em,em 是强化文本的内容,而所有浏览器对重要内容都是以斜体形式显示的,i 则是表示,标签内文本为斜体

** 的 title 和 alt 有什么区别？**

title 是当鼠标划到图片元素时显示的图片描述

alt 是 img 的特有属性，是图片内容的等价描述，用于图片无法加载时显示、读屏器阅读图片。可提高图片可访问性，除了纯装饰性图片外都需要设置有意义的值，搜索引擎会重点分析。

浏览器是怎么对 HTML5 的离线储存资源进行管理和加载的呢？

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

离线的情况下，浏览器就直接使用离线存储的资源

Web 标准以及 W3C 标准是什么？

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

xhtml 和 html 有什么区别？

功能上的差别：

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

书写习惯的差别：

xhtml 元素必须被正确嵌套，闭合、区分大小写，文档必须拥有根元素

Canvas 和 Svg 有什么区别？

svg 绘制出来的每一个图形的元素都是独立的 DOM 节点,能够方便的绑定事件或用来修改。canvas 输出的是一副画布

svg 输出的图形是矢量图形，后期可以修改参数来自由放大缩小，不会失真和锯齿。而 canvas 输出标量画布，就像一张图片一样，放大会失真或者锯齿

Canvas 跨域问题 ？ 如何解决

当 canvas 绘制一张外部链接图片时，我们会遇到跨域问题，在浏览器打开这个页面的时候，会发现这个问题

```
Uncaught DOMException: Failed to execute 'toDataURL' on 'HTMLCanvasElement': Tainted canvases may not be exported.
```

这是由于 canvas 受限于 CORS 策略，会存在跨域问题，虽然可以使用图像，但是绘制到画布上会污染画布，一旦一个画布被污染，就无法提取画布的数据。比如无法使用画布 toBlod()，toDataURL() 或者 getImageData() 方法

解决方案:

1. img 标签上新增了 crossorigin 属性，这个属性决定了图片获取过程

中是否开启 CORS 功能(存在兼容性问题)

```
image.setAttribute( 'crossorigin' , ' anonymous' )
```

2. 将文件读入到 blob 文件对象, 然后使用 URL.createObjectURL 转换成 src 可用的地址

详情访问地址:

https://blog.csdn.net/juse_we/article/details/90639216

CSS:

☆浏览器盒模型?

盒模型分为两种: IE 盒模型和 W3C 盒模型

W3C 标准盒模型: 宽度/padding/border/margin 都是单独分开的

IE 盒模型: 宽度 = 内容宽度+padding+border 是一起的

☆清除浮动的方式

1. 在子元素并级后面添加一个新元素, 添加 clear: both 属性

优点: 通俗易懂, 容易掌握

缺点: 添加无意义空标签, 不方便后期维护

2. 给父元素添加 overflow:hidden

优点: 代码较少, 简单方便

缺点: 不能配合定位使用

3. : after 方法 (作用于浮动元素的父元素)

```
.clearfix:after{

    content:"";

    display: block;

    height:0;

    clear:both;

    visibility:hidden;

}

/* 为兼容 IE6,IE7, 因为 ie6,ie7 不能用 after 伪类 */

.clearfix{

    zoom:1;

}
```

优点：结构和语义化完全正确

缺点：复用方式不当，会造成代码量增加

CSS 选择器有哪些?哪些属性可以继承?

选择器: ID 选择器 (#ID)

Class 选择器 (.class 名)

标签选择器 (标签)

通配符 (*)

相邻选择器 (div+p)

子选择器 (div>p)

后代选择器 (div p)

多个选择器 (div,p,a,ul)

伪类选择器 (a:hover)

拓展内容:

伪类选择器和伪元素的区别:

伪类用于向某些选择器添加特殊效果 (单冒号)

伪元素用于将某个特殊的东西添加到某些元素的前后 (双冒号)

伪类:

伪类	作用
:active	将样式添加到被激活的元素
:focus	将样式添加到被选中的元素
:hover	当鼠标悬浮在元素上方时，向元素添加样式
:link	将特殊的样式添加到未被访问过的链接
:visited	将特殊的样式添加到被访问过的链接
:first-child	将特殊的样式添加到元素的第一个子元素
:lang	允许创作者来定义指定的元素中使用的语言

伪元素:

伪元素	作用
:first-letter	将特殊的样式添加到文本的首字母
:first-line	将特殊的样式添加到文本的首行
:before	在某元素之前插入某些内容
:after	在某元素之后插入某些内容

::after/:after 与 ::before/:before 的区别？

:before 在元素之前添加效果/:after 是在元素之后添加效果

:after/:before 是 CSS2 提出的,兼容 IE8

::after/::before 是 CSS3 为了区分伪类和伪元素的做出的差别,为了避免兼容性问题,习惯性的还是写:after/:before;

可继承样式: font-size/fon-family/color [ul/li/ol/dl/dd/dt]

不可继承样式:width/height/margin/padding/border

CSS 样式优先级算法:

三条标准:

1/就近原则,后加样式优于前面的样式

2/内嵌样式>内联样式>外联样式

3/!Important 大于一切样式

权重计算规则:

内联样式:style=" " -----权值 1000

ID 选择器: #ID -----权值 100

类/伪类/属性选择器 -----权值 10

类型选择器和伪元素 :div/p-----权值 1

继承的样式没有权值

☆CSS3 新特性和伪类有哪些？

新特性:

border-radius(圆角)/box-shadow(阴影)

text-shadow(文字阴影)/线性渐变(line-gradient)/transform 各种
样式(旋转/缩放/定位[xyz]/倾斜)

增加了更多的 CSS 旋转武器,背景颜色加入了 rgba

Border-images/媒体查询/多栏布局

新增伪类:

P: first-of-type 选择属于其父元素中的同类型的第一个 P 元素

P: last-of-type 选择其父元素中的同类型的最后一个 P 元素

::after/::before 在元素之前或之后添加内容

::disabled 控制表单控件的禁用状态

::checked 单选框或复选框被选中

less 的一些优势。

1. 结构清晰，便于扩展
2. 可以方便的屏蔽浏览器私有语法差异
3. 可以轻松实现多重继承
4. 完全兼容 CSS 代码，可以方便的应用到老项目中。Less 知识在 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

了扩展，所以老的 CSS 代码也可以与 Less 代码一同编译

缺点：必须要编译，无论在客户端还是服务器端，都是一种额外的花销

如何居中 div?

水平居中使用 `margin(0 auto)`

垂直居中:`position:absolute;`

`top:50%;`

`Left:50%;`

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

或者考虑 `display:table-cell;`

`Text-align:center;`

`Vertical-align:middle;`

Display 有哪些哪些值?说明他们的作用

`block` 元素转化为块级元素

`inline` 元素转化为行内元素

inline_block 元素转化问行内块元素

None 次元素不会显示,脱离文档流

List-item 元素转化为行内样式,并添加列表样式(如 UL 下的 li)

Table 元素会以块级表格来显示

Inherit 继承父元素 display 属性

Position 的值?

Relative 相对定位(相对于原来位置定位,不脱离文档流)

Absolute 绝对定位(相对于他最近的定位父元素定位,脱离文档流)

Fixed 窗口定位(相对于浏览器窗口进行定位,脱离文档流)

Static ['stætɪk] 默认值,不定位

Inherit 继承父元素的 position 属性

flex 布局以及常用属性

flex 布局可以完美实现响应式布局

1.以下6个属性设置在容器上

flex-direction	row/row-reverse/column/column-reverse	决定主轴的方向（即项目的排列方向）
flex-wrap	wrap/nowrap/wrap-reverse	决定项目排列方式
flex-flow	<flex-direction> <flex-wrap>	前两者简写形式，默认flex-flow:row nowrap
justify-content	flex-start/flex-end/center/space-between/space-around	<p>决定项目在主轴的对齐方式</p> <p>* space-between: 两端对齐，项目之间的间隔都相等。</p> <p>* space-around: 每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。</p>
align-items	flex-start/flex-end/center/baseline/stretch	<p>定义项目在交叉轴上如何对齐</p> <p>* baseline: 项目的第一行文字的基线对齐。</p> <p>* stretch（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。</p>
align-content	flex-start/flex-end/center/space-between/space-around/stretch	定义多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

2.以下6个属性设置在项目上

order	<code>.item{order:1}</code>	定义项目的排列顺序。数值越小，排列越靠前，默认为0。
flex-grow	<code>.item{flex-grow:<number>}</code>	定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大。 (如果所有项目的flex-grow属性都为1，则它们将等分剩余空间（如果有的话）。如果一个项目的flex-grow属性为2，其他项目都为1，则前者占据的剩余空间将比其他项多一倍。)
flex-shrink	<code>.item{flex-shrink:<number>/*default 1*/}</code>	定义项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。(如果所有项目的flex-shrink属性都为1，当空间不足时，都将等比例缩小。如果一个项目的flex-shrink属性为0，其他项目都为1，则空间不足时，前者不缩小。)
flex-basis	<code>.item{flex-basis:length/*default auto*/}</code>	定义在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为auto，即项目的本来大小。(注：它可以设为跟width或height属性一样的值（比如350px），则项目将占据固定空间。)
flex	<code>.item{flex:none}</code>	是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto。后两个属性可选。 该属性有两个快捷值：auto (1 1 auto) 和 none (0 0 auto)。
align-self	<code>.item { align-self: auto flex-start flex-end center baseline stretch; }</code>	允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch。

请解释一下 CSS3 的 flexbox(弹性盒布局模型),以及适用场景？

是一个用于页面布局的新 CSS 功能，规定框内的子元素是否可以伸缩器

尺寸

CSS 打造三角形？

宽度 0,高度 0,边框加宽,给一边加颜色,其余三边使用 transparent

满屏品字布局？

上面 div 宽度 100%;

下面两个宽度 width50%+float/display:inline/inlin-block;

li 与 li 之间有看不见的空白间隙是什么原因引起的？

行内块排列会受到(空格/回车)等的影响,因为空格也属于字符,把字符大小
设置为 0 就 ok 了

为什么要初始化 css 样式？

浏览器的兼容性问题,有些浏览器对标签的默认值是不一样的,如果没有
设置 CSS 初始化,浏览器之间的页面会有差异,最简单的方式:

```
*{ padding ; 0 ;    margin : 0 ; }
```

CSS 中的 visibility 属性的 collapse[kə'ləeps]属性是干嘛的？

- 1、一般情况和 visibility: hidden 一样，不脱离文档流
- 2、在 table 的 tr 元素，脱离文档流
- 3、在 table 的 td 元素中，不脱离文档流

外边距合并是指的什么意思？

是指两个垂直的 margin 相遇,会合并在一起,margin 高度是以最大的 margin 值为准;

移动端的布局用过媒体查询吗?

媒体查询主要用于响应式页面,媒体查询通过页面浏览设备的窗口宽度,完成相应的样式

```
<style>@media(min-width)and(max-width){样式}</style>
```

拓展问题:

响应式页面?

响应式页面主要为了配合各种用户设备的窗口宽度,主要用得到的一个是媒体查询,一个是 bootstrap,一个是 rem 单位,rem 根据页面字体大小等比缩放,可以用 vw/vh+rem,vw/vh 是将窗口大小评分为 100 份;

CSS 媒体查询的原理是什么?

窗口的 onresize 事件,得到窗口大小匹配对应的样式修改

CSS 预处理器(sass 和 less)用过吗?

个人比较喜欢 less,结构清晰,可以与 html 结构保持一致,省去了 css 多层选择器的用法

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。

缺点：

需要预处理工具

重新编译的时间可能会很慢

CSS 优化/提高性能的方法有哪些？

使用 css 预处理器(less/sass),增加代码可复用性,方便项目的协作开发,可维护性.

浏览器是怎么解析 CSS 选择器的？

样式系统优先从关键选择器开始匹配,通过权重,先找祖先元素,再一级一级查下去,如果匹配则使用样式,如果不匹配则放弃

Margin 与 padding 的区别？

Margin 是控制元素与元素之间的距离,padding 是元素与内容之间的距离

Css 如何实现横向滚动与竖向滚动？

横向滚动:父元素:overflow-x:auto; overflow-y:hidden;

竖向滚动:父元素 overflow-x:hidden;overflow-x:auto;

如何设置滚动条样式?

Scrollbar 样式属性,有很多种,很少用,单词没怎么记住;

视觉差效果是如何实现的?

给背景图片添加 background-attachment:fixed 属性,将背景固定在窗口,在使用 background-position:top center 或 0% 0%;后续可以通过 js 修改 background-position 的 top 值,实现背景图片跟随页面上下移动的效果

你对 line-height 如何理解?

line-height 是设置行高的 style 样式,可以增加设置文本行与行之间的上下间距,也可以实现文本在 div 中的垂直居中

设置元素浮动后,元素的 display 值是什么吗?

浮动后,元素的 display 值自动变为 display:block;

怎么让 chrome 支持小于 12px 的文字?

一个是使用图片,不知道的话,就说 12 号字体基本就已经是浏览器的自小号字体,如果字太小,用户阅读内容会很容易产生视觉疲劳感,所以页面中通常是使

用 12px 或者大于 12px 的字体,比如:16/18/24/32 号字体,是比较常用的字体大小

如何设置字体斜体?

i 标签/em 标签/font-style: oblique[ə'blik]

如果需要手写动画,最小时间间隔是多少?

显示器默认 60Hz,一秒刷新 60 次,1000/60,约为 16.7ms

有一个高度自适应的 div,里面有两个 div,一个高度 100px,一个如何自适应高度?

1/父元素 box-sizing:border-box;padding-top:100px;position:relative;

第一个 div position:absolut;

第二个 div height:100%;

Png/jpg/gif 这些图片格式解释一下?

jpg 是正常的图片格式/png 主要设置无背景图片/gif 是动态图片

Style 标签写在 body 前还是 body 后?

正常是写在 body 前的,而且 style 也可以 body 中,但是这回导致 CSS 重新渲染一次页面,占用一定的时间

有什么不同的方式可以隐藏内容？

visiblity: hidden: 元素任然在文档流中，并占用空间；

display: none: 元素脱离文档流，不占用空间；

position: left: -999999px: 将内容至于屏幕之外

text-index: -9999: 只适用于 block 元素中的文本

消除 transition 闪屏

```
.css {  
  
    -webkit-transform-style: preserve-3d;  
  
    -webkit-backface-visibility: hidden;  
  
    -webkit-perspective: 1000;  
  
}
```

过渡动画（在没有启动硬件加速的情况下）会出现抖动的现象，以上的解决方案只是改变视角来启动硬件加速的一种方式；

启动硬件加速的另外一种方式：

```
.css {  
  
    -webkit-transform: translate3d(0,0,0);
```



```
-moz-transform: translate3d(0,0,0);  
  
-ms-transform: translate3d(0,0,0);  
  
transform: translate3d(0,0,0);  
  
}
```

启动硬件加速最常用的方式：translate3D, translateZ, transform

opacity 属性/过渡动画(需要动画执行的过程中才会创建合成层，动画没有开始或结束后元素还会回到之前的状态)

will-change 属性（这个比较偏僻），一般配合 opacity 使用（而且经测试，除了上述可以引发硬件加速的属性外，其它属性并不会变成复合层）

弊端：硬件加速会导致 CPU 性能占用量过大，电池电量消耗加大；因此，尽量避免泛滥使用硬件加速。

CSS 实现单行文本移除显示...

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

还需要加宽度 width 属性来兼容部分浏览器

实现多行文本溢出显示...

`display : -webkit-box ;`

`-webkit-box-orient : vertical ;`

`-webkit-line-clamp : 3 ;`

`overflow : hidden ;`

适用范围：因使用了 Webkit 的 CSS 扩展属性,该方法适用于 Webkit 浏览器以及移动端

注:

`-webkit-line-clamp` 用来限制在一个块元素显示的文本的行数,为了实现该效果,它需要组合其它的 webkit 属性。

常见结合属性:

`display: -webkit-box;` 必须结合的属性, 将对象作为弹性伸缩盒子模式显示。

`-webkit-box-orient` 必须结合的属性, 设置或伸缩伸缩盒对象的子元素排列方式。

溢出显示。。。的另外一种显示方式

纽约时装得分是电风扇飞电风扇地...

发圣诞节的疯狂送积分看见谁开房监控圣
诞节放开就速度快放假水水电费水电 ...

人均 5000 | 长宁区

实现方式：

```
div{
```

```
    position: relative;
```

```
    line-height: 20px;
```

```
    max-height: 40px;
```

```
    overflow: hidden;
```

```
}
```

```
div: after{
```

```
    content : "...";
```

```
    position : absolute ;
```

```
    bottom : 0 ;
```

```
    right : 0 ;
```

```
    padding-left : 40px ;
```

```
background : -webkit-linear-gradient(left , transparent , #fff  
55%);
```

```
background : -o-linear-gradient(left , transparent , #fff 55%);
```

```
background : -moz-linear-gradient(left , transparent , #fff  
55%);
```

```
background : linear-gradient(left , transparent , #fff 55%);  
}
```

此方法也有弊端：就是未超出行的情况下也会出现省略号

注:

1. 将 height 设置为 line-height 的整数倍,防止超出的文字露出。
2. 给 p::after 添加渐变背景可避免文字只显示一半。
3. 由于 ie6-7 不显示 content 内容，所以要添加标签兼容 ie6-7，兼容

ie8 需要将 : : after 替换成 : after

让图文不可复制

```
-webkit-user-select: none;
```

```
-ms-user-select: none;
```

```
-moz-user-select: none;
```

-khtml-user-select: none;

user-select: none;

这些网页为了尊重原创，复制的文本都会被加上一段来源说明，这个是如何做到的呢？拓展：

大致思路：

1. 答案区域监听 copy 事件，并阻止这个事件的默认行为。
2. 获取选中内容（window.getSelection()）加上版权信息，然后设置到剪切板（clipboardData.setData()）

visibility: hidden 与 display: none 的区别？

两个 css 样式都有隐藏元素的效果，但是它们的区别在于：display: none 隐藏元素，可以脱离文档流，而 visibility 隐藏的元素不会脱离文档流，会占有原来的位置。

em, rem, px 的区别？

px 像素单位-----相对长度单位，相对于显示屏分辨率。

特点：IE 无法调整那些使用 px 作为单位的字体大小

国外的大部分网站能够调整的原因在于其使用了 em 或 rem 作为字体单位

Firefox 能够调整 px 和 em、rem，但是 96%以上的中国网民使用 IE 浏览器或（内核）

em-----相对长度单位，相对于当前对象内文本的字体尺寸。如当前对行内文本的字体尺寸未被人为设置，则相对于浏览器的默认字体尺寸（浏览器的默认字体大小是 16px。未经调整的浏览器都符合 1em = 16px）

特点：em 的值不是固定的

em 会继承父级元素的字体大小

rem-----rem 是 CSS3 新增的一个相对访问（root em，根 em），这个单位引起了广泛关注。rem 与 em 的区别在于使用 rem 为元素设定字体大小时，仍然是相对大小，但相对的是 HTML 根元素。这个单位可以根据修改根元素就成比例的调整字体大小。可避免字体大小逐层复合的连锁反应。

css 动画与 js 动画的差异

1. js 动画代码相对复杂一些
2. 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
3. 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

何让一个元素垂直/水平（垂直水平）都居中，请列出你能想到的几种方式？

水平垂直居中 —— 方式一

```
<div class="div-demo"></div><style>
```

```
.div-demo{
```

```
width:100px;
```

```
height:100px;
```

```
background-color:#06c;
```

```
margin: auto;
```

```
position:absolute;
```

```
top: 0;
```

```
left: 0;
```

```
bottom: 0;
```

```
right: 0;
```

```
}</style>
```

水平垂直居中 —— 方式二

```
<style>
```

```
.div-demo{

    width:100px;

    height:100px;

    background-color:#06c;

    margin: auto;

    position:absolute;

    top: 50%;

    left: 50%;

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

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

}</style>
```

水平垂直居中 —— **方式三**，（新旧伸缩盒兼容）

```
<body class="container">

<div class="div-demo"></div>

<style>
```



```
html,body{
```

```
    height:100%;
```

```
}
```

```
.container{
```

```
    display: box;
```

```
    display: -webkit-box;
```

```
    display: flex;
```

```
    display: -webkit-flex;
```

```
    -webkit-box-pack: center;
```

```
    -webkit-justify-content: center;
```

```
    justify-content: center;
```

```
    -webkit-box-align: center;
```

```
    -webkit-align-items: center;
```

```
    align-items: center;
```

```
    }

    .div-demo{

        width:100px;

        height:100px;

        background-color:#06c;

    }

</style>

</body>
```

Chrome、Safari 等浏览器，当表单提交用户选择记住密码后，下次自动填充表单的背景变成黄色，影响了视觉体验是否可以修改

```
input:-webkit-autofill, textarea:-webkit-autofill, select:-webkit-autofill {

    background-color: #fff;//设置成元素原本的颜色

    background-image: none;

    color: rgb(0, 0, 0);

}
```

//方法 2: 由(licongwen)补充 **input:-webkit-autofill** {

-webkit-box-shadow: 0px 0 3px 100px #ccc inset; //背景色

}

浏览器的最小字体为 12px，如果还想再小，该怎么做？

用图片：如果是展示的内容基本是固定不变的话，可以直接切图兼容性也

完美(不到万不得已，不建议)；

找 UI 设计师沟通：为了兼容各大主流浏览器，避免后期设计师来找你撕

逼，主动找 TA 沟通，讲明原因 ——注意语气，好好说话不要激动，

更不能携刀相逼；

CSS3: css3 的样式 transform: scale(0.7)，scale 有缩放功能；

又去找 chrome 复习了一下，说是 “display:table;display: table-cell;”

可以做到，没用过。

给一个 div 设置它的宽度为 100px，然后再设置它的 padding-top 为 20%。问：现在这个 div 有多高？

这题主要考察了对 w3c 标准的了解。如果你亲自去浏览器去试的话会发现这个

div 的高为：316.8(注意：不同分辨率的电脑测试会有不同的效果，这里以我的

电脑 1600x900 为参考)，其实到这里这题已经是解开了，但是可能还有些同学

没明白这个 316.8 是如何计算得来的。别急，请听我细细道来。



如果你搞不懂结果为何是这个的话可能会去查 [w3school](https://www.w3school.com.cn/css/default.asp)，你可能会看到：

<code>auto</code>	浏览器计算内边距。
<code>length</code>	规定以具体单位计的内边距值，比如像素、厘米等。默认值是 <code>0px</code> 。
<code>%</code>	规定基于父元素的宽度的百分比的内边距。
<code>inherit</code>	规定应该从父元素继承内边距。

但是可以这么说上面的所说的是错的，或者说，表述不准确。

例如一下情况：

```
//css
.inner{
    position: absolute;

    width: 100px;

    padding-top: 20%;
}

.mid{
```

```
width: 200px;
}
.wrap{
  position: relative;
  width: 300px;
}
//html
<div class="wrap">
  <div class="mid">
    <div class="inner"></div>
  </div>
</div>
```



如果按照 [w3school](#) 说的，这个 inner 的高应该是 40px，但是实际不是，而是 60px，是以 wrap 的宽度计算的，由此可见，w3school 的说法不成立。

那么，当 padding 设置为 % 时到底以谁为参考呢？

事到如今我也不给大家卖关子了，其实是以**包含块**为参考的。通俗点来说就是谁包含它，它就以谁为参考，在这里 inner 设置了 position: absolute 脱离了原来的文档流，就会去寻找它的祖先元素设置了 position: relative 的元素作为它的包含块。如果还不懂包含块是啥的同学建议仔细阅读我刚刚给的链接，同时还可以参考我在 [segmentfault](#) 上的这个问题。

写一个左中右布局，占满全屏，其中左右两块的固定宽度是200，中间自适应宽度，请写出结构及样式：

```
<style>
  方法一：
  html,body{ margin:0;width:100%; }
  h3{ height: 100px; margin: 20px 0 0; }
  #left,#right{ width:200px;height: 200px;background: #000; position: absolute;top:
120px;}
  #left{left:0px;}
  #right{right: 0px;}
  #center{margin:2px 210px ;background-color: #eee;height: 200px; }
</style>
<style>
  方法二：
  div{
    height: 300px;
  }
  #left,#right{
    width: 200px;
    background: #000;
  }
  #left{
    float:left;
  }
  #right{
    float:right;
  }
  #center{
    margin-left:200px;
    margin-right:200px;
  }
</style>

<h3>实现三列宽度自适应布局</h3>
<div id="left">左边</div>
<div id="right">右边</div>
<div id="center">中间</div>
```

使用左右浮动的方式相对于绝对定位的方法会有一点差异性，并且会有一些小 bug，当中间部分小于内容的情况下，会将右侧的内容挤至下方，可自己试试对比。

CSS sprite 是什么？有什么优缺点？

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

优点：

减少 http 请求数，极大的提高页面加载速度

增加图片信息重复度，提高压缩比，减少图片大小

更换风格方便，只需在一张或几张图片上修改颜色或样式即可实现

缺点：

图片合并麻烦

不方便维护

什么是 FOUC? 如何避免

flash of Unstyle Content：用户定义样式表加载之前浏览器使用默认样式显示文档，用户样式加载渲染之后再重新显示文档，造成页面闪烁

解决方法：把样式表放到文档的 head 中

为什么要初始化 CSS 样式？

因为浏览器的兼容性问题，不同浏览器对有些标签的默认值是不同的，如果没有 css 初始化往往会出现浏览器之间的页面显示差异

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

在网页中的字体大小应该使用偶数还是奇数？为什么呢？

偶数字号相对更容易和 web 设计的其他部分构成比例关系

如果需要手动写动画，你认为最小时间间隔是多久？为什么？（阿里）

多数显示器默认频率是 60Hz，即 1 秒刷新 60 次，所以理论上最小间隔
 $1/60 * 1000\text{ms} = 16.7\text{ms}$

CSS 在性能优化方面的方法？

css 压缩与合并、Gzip 压缩

css 文件放在 head 中，不要使用 @import

尽量用缩写、避免用滤镜、合理使用选择器

base64 的原理及缺点

优点：可以加密，减少了 http 请求

缺点：需要消耗 CPU 进行编解码

stylus、sass、less 区别

均具有变量、混合、嵌套、继承、颜色混合五大基本特性

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

Sass 无全局变量的概念, Less 和 Stylus 有类似于其他语言的作用于概念

Sass 是基于 Ruby 语言的, 而 Less 和 Stylus 可以基于 NodeJS NPM 下载相应库就进行编译

JS:

JS 是一种什么样的语言?

1. 解释性脚本语言, 代码不进行预编译
2. 主要用来向 HTML 页面添加交互行为
3. 可以直接嵌入 HTML 页面, 但单独写成 JS 文件有利于结构和行为的分离
4. 跨平台性, 在绝大多数浏览器的支持下, 可以在多种平台下运行: linux、windows

JS 数据类型有哪些？

栈: (原始数据) string/number/boolean/null/undefined/symbol

堆: (引用数据类型)object (array 和函数属于 object)

数据类型一共 7 (6 种基本类型+1 种引用类型) 种

介绍 JS 有哪些内置对象？

object 是 Javascript 中所有对象的父对象

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

其他对象：Function、Arguments、Math、Date、RegExp、Error

栈与堆的区别？

栈与堆的储存位置不同；

原始数据是储存在栈中简单数据段,体积小,大小固定,属于频繁使用的数据.

引用数据类型是储存在堆中的对象,占据的空间大,如果储存在栈中会影响运行性能,引用数据类型在栈中指明了自己的所在地.当代码解析时,会先从栈中获取地址,然后再从堆中获取实体;

js 中的作用域与变量声明提升

作用域：每一个变量、函数都有其作用的范围，超出范围不得使用，这个叫做作用域

全局变量、局部变量：

全局变量：在全局范围内声明的变量，如 `var a = 1;`

只有赋值没有声明的值，如 `a = 1`（注：如果 `a = 2` 在函数环境中，也是全局变量）

局部变量：写入函数的变量，叫做局部变量，作用范围仅限函数内部

作用：程序安全，内存的释放

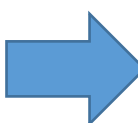
作用域链：

查找变量的过程。先找自己局部环境内部有没有声明或者是函数，如果有，则查看声明有无赋值或者是函数的内容，如果没有，则向上一级查找。

变量声明提升：

在预编译阶段，编译器会把所有定义的变量全部提升到最顶部，即，把变量声明的语句会自动放置在最顶部。

```
var a = 1;
function fn(){
  console.log(a);
  var a = 2;
}
fn(); //undefined
```



```
var a = 1;
function fn(){
  var a;
  console.log(a);
  a = 2;
  //变量提升将声明的变量提升至最顶部
  //赋值在后
  //undefined指向所有未赋值的变量
}
fn(); //undefined
```

`console.log (a)` 何时会打印 1？

当函数内部没有 `a` 这个变量的时候，才会向上一级查找

```
var a = 1;
function fn(){
    console.log(a);
}
fn(); //1
```

如何转化类型？

转数组 `parseFloat()`;

转字符串 `toString()/string()`

数组转字符串 `join()`;

```
var a, b, c;
a = new Array(a, b, c, d, e);
b = a.join('-'); //a-b-c-d-e 使用-拼接数组元素
c = a.join(''); //abcde
```

字符串转数组: `split()`;

```
var str = 'ab+c+de';
var a = str.split('+'); // [ab, c, de]
var b = str.split(''); //[a, b, +, c, +, d, e]
```

什么是面向对象编程及面向过程编程，他们的异同和优缺点

面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个一次调用就可以了

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

面向对象是以功能来划分问题，而不是步骤

面向对象编程思想

基本思想是使用对象、类、继承、封装等基本概念来进行程序设计

优点：易维护

- 采用面向对象思想设计的结构，可读性高，由于继承的存在，即使改变需求，那么维护起来是非常方便你和较低成本的

易扩展

开发工作的重用性、继承性高、降低重复工作量

缩短了开发周期

如何解释 this 在 js 中起的作用？

Js 中的 this,一般取决于调用这个函数的方法

1/如果函数被实例化(new 构造函数名())的情况下,this 指向全新的对象

2/如果是某标签触发什么事件,调用了这个函数,this 指向标签(整个 DOM 节点,包含它的子元素);

3/如果函数使用了 call/apply,this 是作为参数传入对象

4/有时候 this 指向不明确的话,this 会指向 window,ES6 中的箭头函数修改了 this 指向,永远指向作用域

js 中 this 的用法（经典）：

this 是 js 的关键字，随着函数使用场合不同，this 的值会发生变化。但是总

有一个原则，那就是 this 指的是调用函数的那个对象。

//纯粹的函数调用,this 指向全局

```
function test(){  
  
    // this.x = 1;  
  
    console.log(this);  
  
}  
  
test();
```

//作为方法调用,那么 this 就是指向这个上级对象

```
function test1(){  
  
    console.log(this)  
  
}  
  
var o = {}  
  
o.x = test1;  
  
o.x()
```

//构造函数调用,就是生成一个新的对象,这里的 this 指向这个对象

```
function test2(){  
  
    console.log(this)  
  
}  
  
var m = new test2();
```

```
//apply 调用

//this 指向的事 apply 中的第一个参数

var x = 0;

function test3(){

    console.log(this.x);

}

var o = {};

o.x = 1;

o.m =test3;

o.m.apply(); //0

o.m.apply(o); //1
```

☆说说 JS 原型和原型链

原型：函数都要 prototype(显示原型)属性，而 prototype 会自动初始化一个空对象，这个对象就是原型对象

原型对象中会有一个 constructor 属性,这个属性将指向了函数本身

实例化对象都有一个 _proto_(隐式原型)属性，_proto_属性指向原型对象

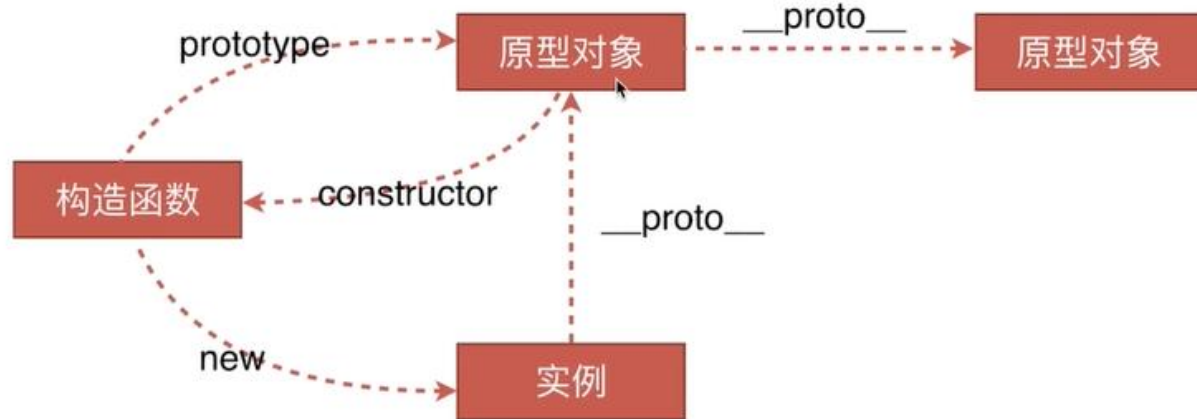
原型链：从实例对象往上找构造这个实例的相关对象，然后这个关联的对象再往上找，找到创造它的上一级的原型对象，以此类推，一直到 object.prototype 原型对象终止,原型链结束.

原型链中的原型对象中的内容,是会被不同的实例,所共有的

如何准确判断一个变量是数组类型？

instanceof 用于判断引用类型属于哪个构造函数的方法

```
var arr = [];
```

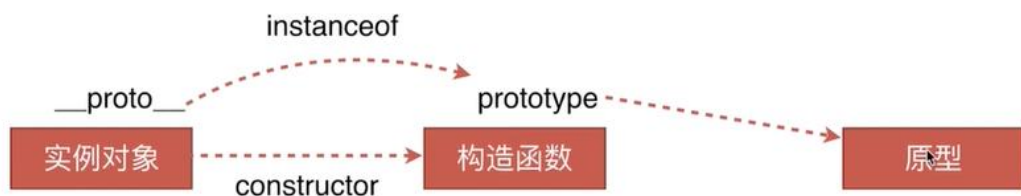


```
arr instanceof Array; //true
```

```
typeof arr; //object
```

typeof 是无法判断是否为数组的

原理:



instanceof 是用来判断实例的__proto__和构造函数的 prototype 是否指向一个原型对象，

但是有一个弊端，只要出现在一条原型链上的，都会返回 true（每个函数都有 prototype，每个对象都有一个内部属性 __proto__，其指向它的原型对象。原型对象也是一个对象，所以也有 __proto__）

这个时候要用实例 __proto__.constructor 更加严谨

```
var arr = [ ];
```

```
console.log(arr instanceof Array); //true
```

```
console.log(arr.__proto__.constructor === Array) //true
```

☆call 和 apply 的区别和作用？

apply 和 call 都是调用一个对象的一个方法，用另一个对象替换当前对象。

相同点：方法的含义是一样的，即方法功能是一样的。并且第一个参数的作用是一样的

不同点：call 可以传入多个参数、apply 只能传入两个参数，所以其第二个参数往往是作为数组形式传入

存在意义：实现（多重）继承

继承的方法有哪些？

原型链继承、构造继承、实例继承、拷贝继承、组合继承、寄生组合继承

继承详情解释：

既然要实现继承，那么我们首先要有一个父类，代码如下：

```
//先定义一个父类  
function Animal(name){
```

```
//属性

this.name = name || 'Animal';

//实例方法

this.sleep = function(){

  console.log(this.name + "正在睡觉!")

}

}

//原型方法

Animal.prototype.eat = function(food){

  console.log(this.name + "正在吃" + food);

}
```

1. 原型链继承

核心：将父类的实例作为子类的原型

```
//原型链继承

function Cat(){ }

Cat.prototype = new Animal();

Cat.prototype.name = "cat";

//Test Code

var cat = new Cat();

console.log(cat.name); //cat

console.log(cat.eat("fish")); //cat 正在吃 fish
```

```
console.log(cat.sleep()); //cat 正在睡觉

console.log(cat instanceof Animal); //true

console.log(cat instanceof Cat); //true
```

特点：

1. 非常纯粹的继承关系，实例是子类的实例，也是父类的实例
2. 父类新增原型方法、原型属性，子类都能够访问到
3. 简单，易于实现

缺点：

1. 要想实现子类新增属性的方法,必须要在 `new Animal()` 这样的语句之后执行,补鞥呢放在构造器中

2. 无法实现多继承
3. 来自原型对象的引用属性是所有实例共享的
4. 创建子类实例时，无法向父类构造函数传参

2. 构造函数

核心：使用父类的构造函数来增强子类实例，等于是赋值父类的实例属性给子类（没用的原型）

```
//构造函数

function Cat(name){

    Animal.call(this);

    this.name = name || "Tom"

}

//Test Code
```

```
var cat = new Cat();

console.log(cat.name); //Tom

console.log(cat.sleep()); //Tom 正在睡觉

console.log(cat instanceof Animal); //false

console.log(cat instanceof Cat); //true
```

特点：

1. 解决了 1 中，子类实例共享父类引用属性的问题
2. 创建子类实例时，可以向父类传递参数
3. 可以实现多继承（call 多个父类对象）

缺点：

1. 实例并不是父类的实例，只是子类的实例
2. 只能继承父类的实例属性与方法，不能继承原型属性、方法
3. 无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

4. 实例继承

核心：为父类实例添加新特性，作为子类实例返回

```
//实例继承

function Cat(name){

    var instance = new Animal();

    instance.name = name || "Tom";

    return instance;

}

//Test Code
```

```
var cat = new Cat();

console.log(cat.name); //Tom

console.log(cat.sleep()); //Tom 正在睡觉!

console.log(cat instanceof Animal); //true

console.log(cat instanceof Cat); //false
```

特点：

1. 不限制调用方法，不管是 new 子类（）还是子类（），返回的对象具有相同的效果

缺点：

1. 实例是父类的实例，不是子类的实例
2. 不支持多继承

4. 拷贝继承

```
//拷贝继承

function Cat(name){

    var animal = new Animal();

    for(var p in animal){

        Cat.prototype[p] = animal[p];

    }

    Cat.prototype.name = name || "Tom"

}

//Test Code

var cat = new Cat();
```

```
console.log(cat.name); //Tom

console.log(cat.sleep()); //Tom 正在睡觉!

console.log(cat instanceof Animal); //false

console.log(cat instanceof Cat); //true
```

特点：

1. 支持多继承

缺点：

1. 效率较低，内存占用高（因为要拷贝父类的属性）
2. 无法获取父类不可枚举的方法（不可枚举方法，不能使用 for in 访问到）

5.组合继承

核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用

```
//组合继承

function Cat(name){

    Animal.call(this);

    this.name = name || "Tom";

}

Cat.prototype = new Animal();

//组合继承也需要修复构造函数的指向问题

Cat.prototype.constructor = Cat;

//Test Code
```

```
var cat = new Cat();

console.log(cat.name); //Tom

console.log(cat.sleep()); //Tom 正在睡觉!

console.log(cat instanceof Animal); //true

console.log(cat instanceof Cat); //true
```

特点：

1. 弥补了方法 2 的缺陷，可以继承实例属性、方法，也可以继承原型属性和方法
2. 既是子类的实例，也是父类的实例
3. 不存在引用属性共享的问题
4. 可传参
5. 函数可复用

缺点：

1. 调用了两次父类构造函数，生成了两份实例（子类实例将子类原型上的那份屏蔽了）

6. 寄生组合继承

核心：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法、属性，避免了组合继承的缺点

```
//寄生组合继承

function Cat(name){

    Animal.call(this);

    this.name = name || "Tom"
```

```
}

(function(){

    //创建一个没有实例方法的类

    var Super = function({});

    Super.prototype = Animal.prototype;

    //将实例作为子类的原型

    Cat.prototype = new Super();

})();

//Test Code

var cat = new Cat();

console.log(cat.name); //Tom

console.log(cat.sleep()); //Tom 正在睡觉!

console.log(cat instanceof Animal); //true

console.log(cat instanceof Cat); //true

//该实现没有修复 constructoe
```

特点:

1. 堪称完美

缺点:

1. 实现较为复杂

☆什么是闭包？闭包有什么作用？

由于在 js 中，变量到的作用域属于函数作用域，在函数执行后作用域会被清除、内存也会随之被回收，但是由于闭包是建立在一个函数内部的子函数，由于其可访问上级作用域的原因，即使上级函数执行完，作用域也不会随之销毁，这时的子函数---也就是闭包，便拥有了访问上级作用域中的变量权限，即使上级函数执行完后，作用域内的值也不会被销毁。

闭包解决了什么：在本质上，闭包就是将函数内部和函数外部连接起来的一座桥梁。

由于闭包可以缓存上级作用域,那么就使得函数外部打破了“函数作用域”的束缚，可以访问函数内部的变量。以平时使用的 Ajax 成功回调为例，这里其实就是个闭包，由于上述的特性，回调就拥有了整个上级作用域的访问和操作能力，提高了几大的便利。开发者不用去写钩子函数来操作审计函数作用域内部的变量了。

闭包有哪些应用场景：

闭包随处可见，一个 Ajax 请求的成功回调，一个事件绑定的回调函数，一个 setTimeout 的延时回调，或者一个函数内部返回另一个匿名函数，这些都是闭包。简而言之，无论使用何种方式对函数类型的值进行传递，当函数在别处被调用时都有闭包的身影

闭包的缺陷：由于闭包打破了函数作用域的束缚，导致里面的数据无法清除销毁，当数据过大时会导致数据溢出

事件代理（事件委托）：

事件代理是将子元素的事件写一个父元素,让父元素代替处理,内部使用 `e.target`,`e.target` 就是触发这个事件的子元素

事件的各个阶段

捕获阶段 ---> 目标阶段 ---> 冒泡阶段

`document` ---> `target` 目标 ---> `document`

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

`true`--->代表该元素在事件的”捕获阶段”(由外向内传递)响应事件

`false` --->表示该元素在事件的”冒泡阶段”(由内向外传递)响应事件

☆**new** 操作符在创建实例的时候经历了哪几个阶段

`new` 创建了一个对象,共经历了 4 个阶段:

1. 创建一个空对象
2. 设置原型链
3. 让实例化对象中的 `this` 指向对象,并执行函数体
4. 判断实例化对象的返回值类型

异步编程的实现方式

-回调函数

优点: 简单、容易理解

缺点：不利于维护，代码耦合高

-事件监听（采用时间驱动模式，取决于某个事件是否发生）

优点：容易理解，可以绑定多个事件，每个时间可以指定多个回调函数

缺点：事件驱动型，流程不够清晰

-发布、订阅（观察者模式）

类似于事件监听，但是可以通过‘消息中心’，了解现在有多少发布者，多少订阅者

- Promise 对象

优点：可以利用 then 方法，进行链式写法；可以书写错误时的回调函数；

缺点：编写和理解，相对比较难

-Generator 函数

优点：函数体内外的数据交换、错误处理机制

缺点：流程管理不方便

-async 函数

优点：内置执行器、更好的语义、更广的适用性、返回的是 Promise，结构清晰

缺点：错误处理机制

对原生 JS 了解程度

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

js 延迟加载的方法有哪些？

defer 和 async、动态创建 DOM 方式（用的最多），按需异步载入 JS

defer 属性：（页面 load 后执行）

script 标签定义了 defer 属性

用途：表明脚本在执行时不会影响页面的构造。也就是所，脚本会被延迟到整个页面解析完毕之后再执行。

```
<script src="XXX.js" defer="defer"></script>
```

async 属性：（页面 load 前执行）

script 标签定义了 async 属性。与 defer 属性类似，都用于改变处理脚本的行为。同样，只适用于外部脚本文件

目的：不让页面等待脚本下载和执行，从而异步加载页面其他内容。异步脚本一定会在页面 load 事件前执行。不能保证脚本会按顺序执行

```
<script src="XXX.js" async></script>
```

动态创建 DOM 方式：

```
1 <script type="text/javascript">
2   function downloadJSAtOnload() {
3       varelement = document.createElement("script");
4       element.src = "defer.js";
5       document.body.appendChild(element);
6   }
7   if (window.addEventListener)
8       window.addEventListener("load",downloadJSAtOnload, false);
9   else if (window.attachEvent)
10      window.attachEvent("onload",downloadJSAtOnload);
11   else window.onload =downloadJSAtOnload;
12 </script>
```

数组从小到大排序？

方法一： sort 方法

```
var array = [1, 4, -8, -3, 6, 12, 9, 8];

function compare(val1, val2) {

    return val1 - val2;

};

array.sort(compare);

document.write(array);
```

方法二：冒泡排序

```
var array = [1, 4, -8, -3, 12, 9];

function sort(arr) {

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

        ////两两比较,如果前一个比后一个大,则交换位置

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

```
        if(arr[i] > arr[j]) {  
            var temp = arr[i];  
            arr[i] = arr[j];  
            arr[j] = temp;  
        }  
    }  
}  
  
sort(array);  
  
console.log(array)
```

查看其他排序方式可以看看：

<https://www.cnblogs.com/real-me/p/7103375.html>

**求从大到小排序可以先使数组从大到小排序,然后添加
reverse()方法，使数组顺序颠倒**

为 string 扩展一个 trim 方法,取掉字符串中的所有空格

方法一：trim（）方法-----仅能取掉字符串首尾空格

```
var str = " a b c "  
  
console.log("trim",str.trim());  
  
//trim 原理  
  
function Trim(str){  
  
    return str.replace(/(^\\s*)|(\\s*$)/g, "");  
}
```

```
}
```

方法二：去除字符中所有的空格

```
str.replace(/\s/ig,"")
```

如何实现数组的随机排序？

最快是给数组添加原生 `sort()` 方法,可以随机数组,如果 `sort()`,方法没有参数的话,就是依照数据的 `unicode` `['juni,kod]`码排序的

可以在 `sort()` 中添加一个比较函数函数:

```
function(a,b){ return Math.random()>.5? -1:1 }
```

`Math.random()`在 0 到 1 之间生成一个随机数

图片懒加载

图片懒加载理解：由于商城图片过多时，就会给图片加一个懒加载的缓冲效果。当图片进入可视化区域的时候才会加载，否则图片只是一个空标签。这样可以优化页面渲染速度，提升用户体验。

思路：将页面中的所有 `img` 属性 `src` 用 `data-src` 代替，当页面滚动至此图片出现在可视区域时，用 `js` 取到该图片的 `data-src` 值赋给 `src`。

所用知识点：

浏览器可视区域的宽高：

js : document.body.clientWidth/clientHeight

jquery: var windowHeight =

\$(window).width()/\$(window).height());

获取滚动条相对于顶部的高度:

js : document.body.scrollTop;

jquery : var scrollTop=\$(window).scrollTop;

获得元素对于浏览器顶部的高度:

js : DOM 元素.offsetTop;

jquery: var imgTop=\$('img').offset().top

判断元素是否出现在浏览器的可视化区域内:

元素相对于顶部的高度 - 浏览器可视化区域的高度 < 小于滚动条

到顶部的高度

成立就代表出现 : 不成立就没出现

怎样排除首屏的图片

元素到顶部距离 - 浏览器的可视化高度 > 0

排除已加载的图片

\$(this).attr('src') != \$(this).attr('data-src') //排除已加载

的图片

Jquery 实现图片懒加载 :

```
<script>
```

```
// 注意: 需要引入 jQuery 和 underscore
```

```
$(function() {
```



```
// 获取 window 的引用:

var $window = $(window);

// 获取包含 data-src 属性的 img，并以 jQuery 对象存入数组:

var lazyImgs = _.map($('img[data-src]').get(), function (i) {

    return $(i);

});

// 定义事件函数:

var onScroll = function() {

    // 获取页面滚动的高度:

    var wtop = $window.scrollTop();

    // 判断是否还有未加载的 img:

    if (lazyImgs.length > 0) {

        // 获取可视区域高度:

        var wheight = $window.height();

        // 存放待删除的索引:

        var loadedIndex = [];

        // 循环处理数组的每个 img 元素:

        _.each(lazyImgs, function ($i, index) {

            // 判断是否在可视范围内:

            if ($i.offset().top - wtop < wheight) {

                // 设置 src 属性:

                $i.attr('src', $i.attr('data-src'));
            }
        });
    }
};

$(window).scroll(onScroll);
```

```
        // 添加到待删除数组:

        loadedIndex.unshift(index);

    }

});

// 删除已处理的对象:

_.each(loadedIndex, function (index) {

    lazyImgs.splice(index, 1);

});

}

};

// 绑定事件:

$window.scroll(onScroll);

// 手动触发一次:

onScroll();

</script>
```

onScroll()函数最后要手动触发一次，因为页面显示时，并未触发 scroll 事件。如果图片已经在可视化区域内，这些图片仍然是 loading 状态，需要手动触发一次，就可以正常显示。

js 中常见的内存泄漏：

1. 内存泄漏会导致一系列问题，比如：运行缓慢、崩溃、高延迟
2. 内存泄漏是指你用不到（访问不到）的变量，依然占据着内存空间，不能

被再次利用起来

3. 意外的全局变量，这些都是不会被回收的变量（除非设置 null 或者被重新赋值），特别是那些用来临时存储大量信息的变量

4. 周期函数一直在运行，处理函数并不会被回收，jq 在移除节点前都会，将事件监听移除

5. js 代码中有对 DOM 节点的引用，dom 节点被移除的时候，引用还维持

深拷贝和浅拷贝的问题：

1. 深拷贝和浅拷贝值针对 Object 和 Array 这样的复杂类型

2. a 和 b 指向了同一块内存，所以修改其中任意一个值，另外一个值也会随之变化，这是浅拷贝

3. a 和 b 指向同一块内存，但是修改其中任意一个值，另外一个调用的变量，不会受到影响，这是深拷贝

4. 浅拷贝：“Object.assign()” 方法用于将所有可枚举的属性的值从一个或多个源对象复制到目标对象，它将返回目标对象

5. 深拷贝：JSON.parse()和 JSON.stringify()给了我们一个基本的解决办法。但是函数不能被正确处理

显示转换与隐式转换

JS 中有 5 中简单的数据类型（也称之为基本数据类型）：undefined、Null、Boolean、Number、String。还有一种复杂的数据-----Object，Object 本质是一组无序的名值对组成的。

对一个值使用 `typeof` 操作符可以返回该值的数据类型，`typeof` 操作符会返回一些令人迷惑但技术上却正确的值，比如调用 `typeof null` 会返回 “object”，应为特殊值 `null` 被认为是一个空的对象引用。

显式转换：主要通过 JS 定义的数据转换方法

各种数据类型及其对应的转化规则：

数据类型	转换为 true 的值	转换为 false 的值
Boolean	true	false
String	任何非空字符串	“ ” (空字符串)
Number	任何非零数字值(包括无穷大)	0 和 NaN
Object	任何对象	null
Underfined	n/a	undefined

隐式转换：是系统默认的，不需要加以声明就可以进行的转换。一般情况下，数据的类型转换通常是由编译系统自动进行的，不需要人工干预

大致规则如下：

1. 对象和布尔值比较

对象和布尔值比较时，对象先转换为字符串，然后再转换为数字，布尔值直接转换为数字

2. 对象和字符串比较

对象和字符串进行比较时，对象转换为字符串，然后两者进行比较

3. 对象和数字比较

对象和数字进行比较时，字符串转换为数字，二者再比较

4. 字符串和数字比较

字符串和数字进行比较时,字符串转换成数字,二者再比较,true=1,
false=0

5. 字符串和布尔值比较

字符串和布尔值进行比较时,二者全部转换成数值再比较

6. 布尔值和数字比较

布尔值和数字进行比较时,布尔转换为数字,二者比较

父元素和子元素分别有点击事件的情况下:

点击父元素只会触发父元素事件,不会影响到子元素,如果点击子元素,会因为冒泡触发父元素的点击事件,可是阻止默认冒泡事件;

mouseover/mouseout 与 mouseenter/mouseleave 的区别与联系

1. mouseover/mouseout 是标准事件,所有浏览器都支持;
mouseenter/mouseleave 是 IE5.5 引入的特有事件,后来被 DOM3 标准采纳,现代浏览器也支持

mouseover/mouseout 是冒泡事件;mouseenter/mouseleave 不冒泡.
需要为多个元素监听鼠标移入/移出事件时,推荐使用 mouseover、mouseout 托管,提高性能

ForEach 和 map 的区别在哪里:

这两个 API 都可以遍历数组

forEach 函数,是给数组中的每个都执行一遍回调函数,不会返回一个值

Map 方法是通过调用数组中的每个元素,映射到新元素中,从而床架一个新数组

```
var a = [ 1,2,3];
var doubled = a.forEach((num,index)=> {
    //用num和/或索引做一些事情。
});
console.log(doubled); //undefined
var c = a.map(function(num){
    return num;
});
console.log(c); //123
```

如果是复合型类型时,如果只改变复合类型的其中某个 value 时,
将可以正常使用

JS 判断设备来源

```
function deviceType(){
```

```
    var ua = navigator.userAgent;
```

```
    var agent = ["Android", "iPhone", "SymbianOS", "Windows Phone", "iP  
ad", "iPod"];
```

```
    for(var i=0; i<len,len = agent.length; i++){
```

```
        if(ua.indexOf(agent[i])>0){
```

```
        break;

    }

}

}

}

deviceType();

window.addEventListener('resize', function(){

    deviceType();

})

//微信的 有些不太一样

function isWeixin(){

    var ua = navigator.userAgent.toLowerCase();

    if(ua.match(/MicroMessenger/i)=='micromessenger'){

        return true;

    }else{

        return false;

    }

}
```

```
}
```

```
}
```

DOM 元素的 e 的 e.getAttribute(propName)和 e.propName 有什么区别和联系？

e.getAttribute()是标准 DOM 操作文档元素属性的方法,具有通用性可在任意文档上使用，返回元素在源文件中设置的属性

e.propName 通常是在 HTML 文档中访问特定元素的特性，浏览器解析元素后生产对应对象，这些对象的特性会根据特定规则结合属性设置得到，对于没有对应特性的属性，只能使用 getAttribute 进行访问

e.getAttribute () 返回值是源文件中设置的值，类型是字符串或者是 null

e.propName 返回值可能是字符串、布尔值、对象、undefined 等

大部分 attribute 与 property 是一一对应关系,修改其中一个会影响另外一个，如 id、title 等属性

一些布尔属性 <input hidden/> 的检测设置需要 hasAttribute 和 removeAttribute 来完成,或者设置对应的 property

像 link 中的 href 属性，转换成 property 的时候需要通过转换得到完整的 url

一些 attribute 和 property 不是一一对应，如：form 控件中 `<input value="hello"/>` 对应的是 `defaultValue`，修改或设置 `value` property 修改的是控件当前值，`setAttribute` 修改 `value` 属性不会改变 `value` property

offsetWidth/offsetHeight、clientWidth/clientHeight 与 scrollWidth/scrollHeight 的区别？

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

`clientWidth`、`clientHeight` 返回值包含 `content+padding`，如果有滚动条，也不包含滚动条

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

focus/blur 与 focusin/focusout 的区别和联系

1. `focus/blur` 不冒泡，`focusin/focusout` 冒泡
2. `focus/blur` 兼容性好，`focusin`、`focusout` 在除 `Firefox` 外的浏览器下都保持良好兼容性，如需使用事件托管，可考虑 `Firefox` 下使用事件捕获

`elem.addEventListener('focus', handler, true)`

3. 可获得焦点的元素:

window/链接被点击或键盘操作/表单控件被点击或键盘操作/设置

tabindex 属性的元素被点击或键盘操作

2.

==与===的区别？

===为等同符，当左边与右边的值与类型都完全相等时，会返回 true；

==为等值符，用来判断值是否相同，不会判断类型是否相同

addEventListener 监听点击事件与 click 事件有什么区别？

addEventListener 事件可以对普通元素进行多个事件处理，click 事件只能使元素运行最新的事件结果

```
window.onload = function(){
    var box = document.getElementById("box");
    box.onclick = function(){
        console.log("我是box1");
    }
    box.onclick = function(){
        box.style.fontSize = "18px";
        console.log("我是box2");
    }
}
```

运行结果：“我是box2”

```
window.onload = function(){  
    var box = document.getElementById("box");  
    box.addEventListener("click",function(){  
        console.log("我是box1");  
    })  
    box.addEventListener("click",function(){  
        console.log("我是box2");  
    })  
}
```

运行结果：我是box1
我是box2

null 和 undefined 的区别？

null 用来表示尚未存在的对象，常用来表示函数企图返回一个不存在对象

undefined 主要指定义了变量，但是并未赋值

NAN （not a Number）不是一个明确数值的数字类型

ECMAScript 认为 undefined 是从 null 派生出来的，他们的值是一样的，但是类型却不一样。

所以

```
null == undefined    //true
```

```
null === undefined   //false
```

如何检查对象中是否存在某个属性？

检查对象中存在某个属性的方式有三种：

第一种使用 in 操作符号，返回 true 或 false:

```
console.log(“属性名” in 对象名);
```

第二种使用 hasOwnProperty 方法，会返回 true 或 false:

```
console.log(对象.hasOwnProperty(“属性名”));
```

第三种是是够括号符号 obj['属性名'], 如果属性存在, 则返回该属性的值, 如果不存在, 则返回 undefined

字符串操作方法。

split () : 用于把一个字符串分割成字符串数组

search () : 用于减缩字符串中指定的子字符串, 或检索与正则表达式相匹配的字符串

indexOf () : 可返回某个字符换字符串中首次出现的位置

substring () : 用于提取字符串中介于两个指定下标之间的字符

trim () : 移除字符串两边的空格

replace () : 替换字符

数组操作方法。

length: 计算数组的长度

索引: 通过索引获取数组中对应值, 同时也可以改变索引对应的值

indexOf: 返回指定元素的位置, 若元素不存在返回-1

slice: 接受 1-2 个参数, 参数对应的是要返回的是要返回项的起始位置和结束位置, 若只有一个参数, 该方法返回从参数指定位置到数组结尾的所有项, 如果还有两个参数, 则返回起始位置到结束位置项, 但是不包括结束位置项, 返回的结果是一个新数组。

push: 向数组末位添加若干元素, 返回添加元素后数组的长度

pop: 删除数组末位最后一个元素, 但会被删除元素

unshift：向数组头部添加若干元素，返回添加元素后的数组长度

shift：删除数组头部的第一个元素，返回被删除的元素

sort：对数组进行排序，返回排序后的数组

reverse：对数组中的数据进行反转，返回反转后的数组

concat：将两个数组合并，返回新数组（可以接受任意元素和数组，并进行拆分放入数组）

join：将数组中的每一个元素用指定的字符串拼接起来

js 的 for 跟 for in 循环它们之间的区别？

遍历数组时的异同：for 循环 数组下标的 typeof 类型:number, for in 循

环数组下标的 typeof 类型:string;

```
var southSu = ['苏南','深圳','18','男'];for(var i=0;i<southSu.length;i++){
```

```
    console.log(typeof i); //number
```

```
    console.log(southSu[i]);// 苏南 ， 深圳 ， 18 ， 男
```

```
}var arr = ['苏南','深圳','18','男','帅气',"@IT 菲酵犯缙  ?-首席填坑官"];for(var k in arr){
```

```
    console.log(typeof k);//string
```

```
    console.log(arr[k]);// 苏南 ， 深圳 ， 18 ， 男 ， 帅气,平头哥联盟-首席填坑官
```

```
}
```

前端技术交流群：104833704

遍历对象时的异同:for 循环 无法用于循环对象,获取不到 obj.length; for

in 循环遍历对象的属性时,原型链上的所有属性都将被访问,解决方案:

使用 hasOwnProperty 方法过滤或 Object.keys 会返回自身可枚举属性

组成的数组

```
Object.prototype.test = '原型链上的属性,本文由平头哥联盟-首席填坑官·苏南分享';var
```

```
southSu = {name:'苏南',address:'深圳',age:18,sex:'男',height:176};for(var i=0;i<s
```

```
outhSu.length;i++){
```

```
    console.log(typeof i); //空
```

```
    console.log(southSu[i]);//空
```

```
}
```

```
for(var k in southSu){
```

```
    console.log(typeof k);//string
```

```
    console.log(southSu[k]);// 苏南 , 深圳 , 18 , 男 , 176 ,本文由平头哥联盟-首席
```

```
    填坑官·苏南分享
```

```
}
```

push()、pop()、shift()、unshift()分别是什么功能?

push 方法 将新元素添加到一个数组中，并返回数组的新长度值。

```
var a=[1,2,3,4]; a.push(5);
```

pop 方法 移除数组中的最后一个元素并返回该元素。

```
var a=[1,2,3,4]; a.pop();
```

shift 方法 移除数组中的第一个元素并返回该元素。

```
var a=[1,2]; alert(a.shift());
```

unshift 方法 将指定的元素插入数组开始位置并返回该数组。

如果用原生 js 给一个按钮绑定两个 click 事件？

使用事件监听，可给一个 DOM 节点绑定多个事件（addEventListener）

拖拽会用到哪些事件？

dragstart---拖拽开始时在被拖拽元素上触发此事件，监听器需要设置拖拽所需数据，操作系统拖拽文件到浏览器时不触发此事件

dragenter---拖拽鼠标进入元素时在该元素上触发，用于给拖放的元素设置视觉反馈，如高亮

dragover---拖拽时鼠标在目标元素上移动时触发，监听器通过组织浏览器默认行为设置元素为可拖放元素

dragleave---拖拽时鼠标移出目标元素时在目标元素上触发，此时监听器可以取消掉前面设置的视觉效果

drag---拖拽期间在被拖拽元素上连续触发

drop---鼠标在拖放目标上释放时，在拖放目标上触发，此时监听器需要收

集数据并且执行所需操作，如果是从操作系统拖放文件到浏览器，需要取消浏览器默认行为

dragend---鼠标在拖放目标上释放时，在拖拽元素上触发，将元素从浏览器拖放到操作系统时不会触发此事件

JS 中定时器有哪些？他们的区别及用法是什么？

setTimeout 只执行一次

setInterval 会一直重复执行

document.write 和 innerHTML 的区别？

document.write 是直接写入到页面的内容流，如果在写之前没有调用 document.open，浏览器会自动调用 open。每次写完关闭后重新调用该函数，会导致页面被重写

innerHTML 则是 DOM 页面元素的一个属性，代表该元素的 html 内容，你可以精确到某一个具体的元素来进行更改。如果想修改 document 的内容，则需要修改 document.documentElement.innerHTML

innerHTML 将内容写入某个 DOM 节点，不会导致页面全部重绘，innerHTML 很多情况下都优于 document.write，其原因在于其允许更精准的控制要刷新页面的那个部分

createElement 与 createDocumentFragment 的区别？

共同点：

1. 添加子元素后返回值都是新添加的子元素
2. 都可以通过 `appendChild` 添加子元素，并且子元素必须是 `node` 类型，不能为文本
3. 若添加的子元素是文档中存在的元素，则通过 `appendChild` 在为其添加子元素时，会从文档中删除之存在的元素

不同点：

1. `createElement` 创建的是元素节点，节点类型为 1，`createDocumentFragment` 创建的是文档碎片，节点类型是 11
2. 通过 `createElement` 新建元素必须指定元素 `tagName`，因为其可用 `innerHTML` 添加子元素。通过 `createDocumentFragment` 则不必
3. 通过 `createElement` 创建的元素是直接插入到文档中，而通过 `createDocumentFragment` 创建的元素插入到文档中的是他的子元素

attribute 和 property 的区别是什么？

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

property 就是 dom 元素在 js 中作为对象拥有的属性；

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

一次性插入 1000 个 div，如何优化插入的性能

使用 Fragment (`document.createDocumentFragment()`)

DocumentFragments 是 DOM 节点，它们不是 DOM 树的一部分。通常的用例是创建文档片段，将元素附加到文档片段，然后将文档片段附加到 DOM 树中。因为文档片段存在于内存中，并不在 DOM 树中，所以讲子元素插入到文档片段时不会引起页面回流。因为使用文档片段会带来更好的性能

先创建一个 div，后续的复制这个元素，避免重复创建元素，再放到元素片段里面

```
var divFragment = document.createDocumentFragment ( ) ;  
  
let div = document.createElement ( “div” ) ;  
  
for ( var i = 0; i <1000; i ++ ) {  
  
    divFragment.append ( div.cloneNode ( ) )  
  
}  
  
document.body.appendChild ( divFragment ) ;
```

JS 中几种常见的高阶函数

高阶函数是对其他函数进行操作的函数，可以将它们作为参数或通过返回它们。简单来说，高阶函数是一个函数，它接收函数作为参数或将函数作为输出返回。

typeof 和 instanceof 有什么区别？

typeof 用于判断数据的基本类型

instanceof 用于判断一个变量是够属于某个对象的实例。也可以用来判断某个构造函数的 prototype 属性是否存在另外一个要检测对象的原型链上

简述同步和异步的区别？

同步就是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程将会一直等待下去，直到返回信息之后才继续执行后面的代码

异步是指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。

window.onload 和 DOMContentLoaded 的区别

```
window.addEventListener( 'load' ,function(){  
    //页面的所有资源加载完才会执行，包括图片和视频等  
})  
  
document.addEventListener( 'DOMContentLoaded' ,function(){  
    //DOM 渲染完即可执行，此时图片,视频还可能没有加载完成  
})
```

document.load 和 document.redy 的区别？

1. load 是当页面所有资源加载完成后（包括 DOM 文档树，css 文件，js 文件，图片资源等），执行一个函数

问题：如果图片资源较多，加载时间较长。onload 后等待执行的函数需要等待较长时间，所以一些效果可能受到影响

2. \$(document).ready()是当 DOM 文档树加载完成后执行一个函数(不包含图片,css 等)，所以比 load 较快执行

注意：在原生 JS 中不包括 ready()这个方法，只有 load 方法就是 onload

事件

解释一下线程和进程的区别？

进程：是资源分配的基本单位，它是程序执行时的一个实例。程序运行时系统就会创建一个进程，并为它分配 CPU 时间，程序开始真正运行。

线程：是程序执行时的最小单位，它是进程的一个执行流，是 CPU 调度和分配的基本单位，一个进程可以由很多个线程组成，线程间共享进程的所有资源，每个线程有自己的堆栈和局部变量。线程由 CPU 独立调度执行，在多 CPU 环境下允许多个线程同时运行。同样多线程也可以实现并发操作，每个请求分配一个线程来处理。

线程和进程有什么区别和优劣势？

1. 进程是资源分配的最小单位，线程是程序执行的最小单位。
2. 进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。而线程是共享进程中的数据，使用相同的地址空间，因此 CPU 切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。
3. 线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC）进行。不过如何处理好同步与互斥是编写多线程程序的难点
4. 但是多进程程序更健壮，多线程程序只要有一个线程死掉，整个进程也死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程有自己独立的地址空间

什么是伪数组，如何将伪数组转换为真数组？

伪数组：无法直接调用数组方法或期望 length 属性有什么特殊的行为，不具有 push、pop 等方法，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 参数等

伪数组的三大特性：

1. 具有 length 属性
2. 按索引方式存储数据
3. 不具有数组的 push, pop 等方法

将伪数组转化为标准数组需要用到数组原型中的方法 slice

例如： `Array.prototype.slice.call(伪数组名称);`

for in 和 for of 的区别

1. for ... in 循环：只能获得对象的键名，不能获得键值

for ... of 循环：允许遍历获取键值

2. 对于普通对象，没有部署原生的 iterator 接口，直接使用 for ... of 会报错

可以使用 for ... in 循环遍历键名

也可以使用 `Object.keys(obj)` 方法将对象的键名生成一个数组，然后遍历这个数组

3. for ... in 循环不仅遍历数字键名，还会遍历手动添加的其他键，甚至包括原型链上的键。for ... of 则不会这样

4. forEach 循环无法中途跳出，break 命令或 return 命令都不能奏效

for ... of 循环可以与 break、continue 和 return 配合使用，跳出循环

5. 无论是 for ... in 还是 for ... of 都不能遍历出 Symbol 类型的值，遍历 Symbol 类型的值需要用 Object.getOwnPropertySymbols() 方法

总之，for ... in 循环主要是为了遍历对象而生，不适用于遍历数组

for ... of 遍历可以用来遍历数、类数组对象、字符串、Set、Map 以及 Generator 对象

判断是否为数组的方法

```
console.log( arr instanceof Array )
```

```
console.log( arr.constructor === Array)
```

```
console.log( Array.isArray( arr ))
```

如何阻止事件冒泡和默认事件？

```
cancelBunnle (IE)
```

```
return false;
```

```
event.preventDefault( );
```

```
event.stopPropagation( );
```

JS 垃圾回收方式

标记清楚：这是 js 最常用的垃圾回收方法，当一个变量进入执行环境时，例如函数中声明一个变量，将其标记为进入环境，当变量离开环境时，（函数执行结束），标记为离开环境

引用计数：跟踪记录每个值被引用的次数，声明一个变量，并将引用类型复

制给这个变量，则这个值的引用次数+1，当变量的值变成了另一个，则这个值的引用次数-1，当这个值的引用次数为 0 的时候，就回收

DOM 节点的增删改查

1. 查找

`getElementById()` `getElementsByClassName()`

`getElementsByTagName()`

`querySelector()` 返回第一个匹配的元素 `querySelectorAll()` 返回全部匹配的元素

2. 插入

`appendChild()` 末尾插入

`insertBefore()` 特定位置插入

3. 替换

`replaceChild()` 接受两个参数，第一个为要插入的节点，第二个为要替换的节点

4. 删除

`removeChild()`

实现函数 A 和函数 B，函数 B 继承函数 A

```
//方式 1
function A(){};
function B(){};
B.prototype = new A()
//方式 2
```

```
function A(){};
function B(){
  A.call(this);
};

// 方式 3
function A(){};
function B(){};
B.prototype = new A();
function B(){
  A.call(this)
}
```

mouseover 和 mouseenter 的区别

mouseover: 当鼠标移入元素或其子元素都会触发事件, 所以有一个重复触发冒泡的过程

mouseenter: 当鼠标移入元素 (除了元素本身, 不包含子元素) 会触发的事件

JS 的各种位置, 比如: clientHeight、scrollHeight、offsetHeight 以及 scrollTop、offsetTop、clientTop 的区别?

clientHeight: 表示的是可视区域的高度, 不包含 border 和滚动条

offsetHeight: 表示可是区域的高度, 包含了 border 和滚动条

scrollHeight: 表示了所有区域的高度, 包含了滚动条被隐藏的部分

clientTop: 表示边框 border 的厚度, 在为指定的情况下, 一般为 0

scrollTop: 滚动后被隐藏的高度, 获取对象相对于 offsetParent 属性指定的父坐标距离顶部的距离

展开（Spread）运算符和剩余（Rest）运算符有什么区别？

展开运算符（spread）是三个点(...)，可以将一个数组转为用逗号分隔的参数序列。说的通俗易懂一点，有点像化骨绵掌，把一个大元素给打散成一个个单独的小元素

剩余运算符也是用三个点(...)表示，它的样子看起来和展开运算符一样,但是它是用于结构数组和对象。在某种程度上，剩余元素和展开元素相反，展开元素会”展开”数组变成多个元素，剩余元素会收集多个元素和”压缩”成一个单一的元素

```
// 拓展运算符
let arr = ["阿里巴巴","阿里妈妈","蚂蚁金服"];
console.log(...arr); //阿里巴巴 阿里妈妈 蚂蚁金服
//剩余运算符
let [webName,...rest] = ["阿里巴巴","阿里妈妈","蚂蚁金服"];
console.log(webName); //阿里巴巴
console.log(rest); //(2) ["阿里妈妈", "蚂蚁金服"]
```

JQuery:

你觉得 jQuery 或 zepto 源码有哪些写的好的地方

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

```
(function( window, undefined ) {  
  
    //用一个函数域包起来，就是所谓的沙箱  
  
    //在这里边 var 定义的变量，属于这个函数域内的局部变量，避免污染全局  
  
    //把当前沙箱需要的外部变量通过函数参数引入进来  
  
    //只要保证参数对内提供的接口的一致性，你还可以随意替换传进来的这个参数  
  
    window.jQuery = window.$ = jQuery;  
  
})( window );
```

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

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

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

jQuery 的实现原理？

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

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

```
window.jQuery = window.$ = jQuery;
```

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

jQuery.fn 的 init 方法返回的这指的是什么对象？为什么要返回这个？

jQuery.fn 的 init 方法返回的这就是 jQuery 对象

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

jQuery.extend 与 jQuery.fn.extend 的区别？

\$.fn.extend () 和 \$.extend () 是 jQuery 为扩展插件提供了两个方法

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

```
$.extend({  
  
    min: function(a, b) { return a < b ? a : b; },  
  
    max: function(a, b) { return a > b ? a : b; }  
});  
  
$.min(2,3); // 2  
  
$.max(4,5); // 5
```

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

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

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

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

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

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

```
$.fn.extend({

    alertValue: function() {

        $(this).click(function(){

            alert($(this).val());

        });

    }

});

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

jQuery 的属性拷贝（extend）的实现原理是什么，如何实现深拷贝？

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

oldObject);

深拷贝（对原始对象属性所引用的对象进行进行递归拷贝）
var newObject
= \$.extend(true, { }, oldObject);

JQuery 的队列是如何实现的？队列可以用在哪些地方？

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

主要应用于 animate()，ajax，其他要按时间顺序执行的事件中

```
var func1 = function(){alert('事件 1');}

var func2 = function(){alert('事件 2');}

var func3 = function(){alert('事件 3');}

var func4 = function(){alert('事件 4');}

// 入栈队列事件

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

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

// 替换队列事件

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

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

// 获取队列事件（返回一个函数数组）

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

// 出栈队列事件并执行

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

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

// 清空整个队列

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

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

bind 直接绑定在目标元素上

live 通过冒泡传播事件，默认文件上，支持动态数据

委托更精确的小范围使用事件代理，性能优于 live

on 是最新的 1.9 版本整合了之前的三种方式的新事件绑定机制

jQuery 一个对象可以同时绑定多个事件，这是如何实现的？

bind on delegate live 进行多事件绑定的原理

针对 jQuery 的优化方法？

缓存频繁操作 DOM 对象

尽量使用 ID 选择器代替类选择器

总是从 #ID 选择器来继承

尽量使用链式操作

在绑定事件上使用时间委托

采用的 jQuery 的内部函数数据 () 来存储数据

使用最新版本的 jQuery

数据请求相关问题：

http 请求方式有哪些？

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD 方法。

HTTP1.1 新增了五种请求方法： OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

http 的状态码有哪些？ 分别说下它们的含义

1XX： 信息状态码

2XX： 成功状态码

3XX： 重定向

4XX： 客户端错误

5XX： 服务器错误

常见状态码：

401 请求要求身份验证。对于需要登录的网页，服务器可能返回此响应

403 服务器已经理解请求，但是拒绝执行它。

404 请求失败

500 服务器遇到一个未曾预料的情况，导致无法完成对请求的处理

503 由于请示服务器维护或者过载，服务器当前无法处理请求

请描述一下 get 与 post 的区别

W3schools 上面的参考答案

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 请求参数会被完整的保留在浏览器历史记录中，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 没有

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

get 与 post 都是 http 协议中的两种发送请求方式，由于 http 的层是 TCP/IP，所以 get 与 post 的底层也是 TCP/IP。而 get 产生一个 TCP 数据包，post 产生两个 TCP 数据包。具体点来说就是：

对于 Get 方式的请求，浏览器会把 http header 和 data 一并发送出去，服务器响应 200（返回数据）

对于 Post 方式的请求，浏览器先发送 header，服务器响应 100，浏览器再

发送 data，服务器响应 200（返回数据）

这样看来 Post 需要两步，时间上消耗的要多一点。所以 get 比 post 更有效

但是：

1. get 与 post 都有自己的语义，不能随便混用
2. 如果网络好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。但是网络环境比较差的情况下，两次包的 TCP 在验证数据包完整性上，有很大的优点

并不是所有浏览器都会在 POST 中发送两次包，Firefox 就只发送一次

get 请求传参长度的误区

实际上 http 协议从未规定 GET/POST 的请求长度限制是多少，对 get 请求参数的限制是来源于浏览器或 Web 服务器，浏览器或者 Web 服务器限制了 url 的长度。为了明确这个概念，需要强调以下几点

1. HTTP 协议为规定 GET 和 POST 的长度限制
2. GET 的最大长度显示是因为浏览器和 web 服务器限制了 url 的长度
3. 不同的浏览器和 web 服务器， 限制长度不一样
4. 要支持 IE，则最大长度为 2083byte，若支持 Chrome，则最大长度

8182byte

get 和 post 请求在缓存方面的区别

get 请求类似于查找的过程，用户获取数据，可以不用每次都与数据库链接，所以可以使用缓存

post 不同，post 做的一般是修改和删除的工作，所以必须与数据库交互，所以不能使用缓存

get 请求更适合于请求缓存

http 和 https 有何区别？如何灵活使用？

http 协议传输的数据都是未加密的，也就是明文的，因此使用 http 协议传输隐私信息非常不安全。为了保证这些隐私数据能加密传输，于是网景公司设计了 ssl(Secure Sockets Layer)协议用于对 http 协议传输的数据进行加密，从而就诞生了 https。

简单来说，https 协议是由 ssl+http 协议构建的可进行加密传输、身份认证的网络协议，要比 http 协议安全。

https 和 http 的主要区别：

一、https 协议需要到 ca 机构申请 ssl 证书(如沃通 CA)，另外沃通 CA 还提供 3 年期的免费 ssl 证书，高级别的 ssl 证书需要一定费用。

二、http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。

三、http 和 https 使用的是完全不同的连接方式，用的端口也不一样，http 是 80 端口，https 是 443 端口。

四、http 的连接很简单，是无状态的；https 协议是由 ssl+http 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全。

五、如果要实现 HTTPS 那么可以淘宝 Gworg 获取 SSL 证书。

什么是 Ajax?为什么使用 Ajax?

Ajax 是一种创建交互网页应用的网页开发技术,相当于在用户和服务器之间加了一个中间层(Ajax 引擎),使用户操作与服务器响应异步化。并不是所有的用户请求都提交给服务器。像一些数据验证和数据处理等都交给 Ajax 引擎自己来做,可以实现页面的局部刷新。

Ajax 包含了一下技术:

1. 基于 web 标准 XHTML+CSS 表示
2. 使用 DOM 进行动态显示以及交互
3. 使用 XML 和 XSLT 进行数据交互及相关操作
4. 使用 XMLHttpRequest 进行一步数据查询、检索;
5. 使用 Javascript 将所有的东西绑定在一起

简述 ajax 的过程。

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
3. 设置响应 HTTP 请求状态变化的函数
4. 发送 HTTP 请求
5. 获取异步调用返回的数据

6. 使用 JavaScript 和 DOM 实现局部刷新

Ajax 优缺点？

Ajax 程序的优势在于：

通过异步模式，提升了用户体验

页面无刷新更新（局部更新），用户体验非常好

Ajax 引擎在客户端运行，承担了一部分本来有服务器承担的工作，从而减少了服务负载

基于标准化的被广泛支持的技术，不需要下载插件或小程序

Ajax 的缺点：

Ajax 不支持浏览器 back 按钮

安全问题，Ajax 暴露了与服务器交互的细节

对搜索引擎的支持比较弱

破坏了程序的异常机制

不容易调试

XMLHttpRequest 通用属性和方法

1. readyState：表示请求状态的整数、取值：

UNSENT(0)：对象已创建

OPENED(1)：open()成功调用，在这个状态下，可以 xhr 设置请求头，或者使用 send () 发送请求

HEADERS——RECEIVED(2)：所有重定向已经自动完成访问, 并且最终响应的 HTTP 头已经收到

LOADING(3)：响应体正在接收

DONE(4)：数据传输完成或者传输产生错误

2. onreadystatechange：readyState 改变时调用的函数
3. status：服务器返回的 HTTP 状态码 (如：200、404)
4. statusText：服务器返回的 HTTP 状态信息 (如：OK、No Content)
5. responseText：作为字符串形式的来自服务器的完整响应式
6. responseXML：Document 对象，表示服务器的响应解析成的 XML 文档
7. abort()：取消异步 HTTP 请求
8. getAllResponseHeaders()：返回一个字符串,包含响应中服务器发送的全部 HTTP 包头。每个报头都是一个用冒号分割名、值对，并且使用一个回车、换行来分割报头行
9. getResponseHeader (headerName)：返回 haedName 对应的报头值
10. open (method, url, asynchronous, [user, password])：初始化准备发送到服务器上的请求。method 是 HTTP 方法，不区分大小写；url

是请求发送的相对或绝对 URL；asynchronous 表示请求是否异步；user 和 password 提供身份验证

11. setRequestHeader (name, value)：设置 HTTP 报头

12. send (body)：对服务器进行初始化。参数 body 包含请求的主体部分，对于 POST 请求为键值对字符串；对于 GET 请求，为 null

Ajax 请求跨域接口，发送了几次请求？

所有跨域的 js 在提交 post 请求的时候，如果服务端设置了可跨域访问，都会默认发送两次请求，第一次预检请求，查询是否支持跨域，第二次踩死真正的 post 提交

跨域的几种方式

首先了解一下同源策略：同源策略、SOP 是一种约定，是浏览器最核心也会最基本的安全功能，如果缺少了同源侧罗，浏览器很容易受到 XSS、CSRF 等攻击。所谓同源是指“协议+端口+域名”三者相同，即便两个不通的域名指向同一个 IP 地址，也非同源。

怎样解决跨域问题？

1. 通过 jsonp 跨域

原生实现:

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

前端技术交流群：104833704

```
script.type = 'text/javascript';

// 传参并指定回调执行函数为 onBack

script.src = 'http://www.....:8080/login?user=admin&callback=onBack

';

document.head.appendChild(script);


// 回调执行函数

function onBack(res) {

    alert(JSON.stringify(res));

}
```

2. document.domain+iframe 跨域

此方案仅限主域相同,子域不同的跨域应用场景

1>父窗口:(http://www.domain.com/a.html)

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

```
<script>
```

```
document.domain = 'domain.com';
```

```
var user = 'admin';
```

```
</script>
```

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

```
<script>
```

```
document.domain = 'domain.com';
```

```
// 获取父窗口中变量
```

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

```
</script>
```

弊端:查看页面渲染优化

3. nginx 代理跨域
4. nodejs 中间件代理跨域
5. 后端在头部信息里面设置安全域名

web 应用从服务器端主动推送 data 大客户端有哪些方式?

- 1) html5 websocket

2) XHR 长轮询

3) 不可见的 iframe

4) <script>标签的长时间链接(可跨域)---http、Jsonp 方式的长轮询

详解:

通常情况下,打开网页或 app 去查询或者刷新, 客户端想服务器发出请求然后返回数据, 客户端与服务端对应的模式是: 客户端请求--服务端响应, 在有些情况下, 服务端会主动推送一些信息到客户端, 如: 新闻订阅等。这个需求类似于日常中使用 QQ 或微信时的新消息提醒一样, 只要有新消息就需要提醒。

Ajax 轮询-----我们自然会想到的是 ajax 轮询, 每 10S 或 30S 轮询一次, 这种方式有一个非常严重的问题。假如有一万个商家打开浏览器, 采用 10S 轮询的方式, 服务器就会承担 1000 的 QPS, 这种方式会对服务器造成极大的性能浪费。

优点: 实现简单

缺点: 加重网络负担, 拖累服务器

Websocket-----websocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行双全工通讯的协议, websocket 只需要简历一次链接, 就可以一直保持连接的状态。

详情可看 <https://www.cnblogs.com/jingmoxukong/p/7755643.html>

XHR 长轮询-----这种方式是比较多的长轮询模式。客户端打开一个到服务端的 ajax 请求然后等待响应; 服务器需要一些特定功能来允许请求被挂起, 只要一有时间发生, 服务端就会在挂起的请求中送回响应并关闭该请求。客户端 js 会在处理完服务器返回的信息后, 再次发出请求, 重新建立连接。

现在浏览已经支持 CROS 的跨域方式请求,因此 HTTP 和 JSONP 的长轮询方式会被慢慢淘汰,建议采用 XHR 长轮询。

优点: 客户端很容易实现良好的错误处理和超时管理, 实现成本与 Ajax 的方式类似

缺点: 需要服务器端有特殊的功能来临时挂起链接。当客户端发起请求过多时, 服务器端会长期保持多个链接, 具有一定的风险。

Iframe-----iframe 通过 HTML 页面里嵌入了一个隐藏帧, 然后将这个隐藏帧的 SRC 属性设为对一个长连接的请求, 服务器端能源源不断的往客户端输入数据。

优点: 每次数据传送都不会关闭连接, 连接只会在通信出现错误时, 或者是连接重建时关闭。

缺点: IE、Firefox 的进度条会显示加载没有完成, IE 则会一直表述正在加载进行中。而且 iframe 会影响页面优化效果。

http 和 Jsonp 方式的长轮询-----把 script 标签附加到页面中让脚本执行。服务器会挂起链接直到有事件发生, 接着把脚本的内容发送回浏览器, 然后重新打开另一个 script 标签来获取下一个事件, 从而实现长轮询的模型。

如何实现浏览器内多个标签页之间的通信? (阿里)

WebSocket、SharedWorker

也可以调用 localStorage、cookies 等本地存储方式

websocket 如何兼容低浏览器? (阿里)

Adobe Flash Socket 、

ActiveX HTMLFile (IE) 、

基于 multipart 编码发送 XHR 、

基于长轮询的 XHR

fetch、ajax、axios 之间的详细区别以及优缺点：

1. JQuery ajax

```
//JQuery ajax

$.ajax({

  type: "POST",

  url: url,

  data: data,

  dataType: dataType,

  success: function() {},

  error: function() {}

})
```

优缺点：

本身针对 MVC 的编程，不符合现在前端 MVVM 的浪潮

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

JQuery 整个项目太大，单纯使用 ajax 却要引入整个 JQuery 非常不合

理（采取个性打包的方案又不能享受 CDN 服务）

2. axios

```
//axios

axios({

  methods: "post",

  url: url,

  data:{

    data_key: data_value,

    ...

  }

})

.then(function(response){

  console.log(response)

})

.catch(function(error){

  console.log(error)

})
```

优缺点：

从 node.js 创建 http 请求

支持 Promise API

客户端支持防止 CSRF

提供了一些并发请求的接口（重要，方便了很多的操作）

3. fetch

```
try{  
  
  let response = await fetch(url);  
  
  let data = response.json();  
  
}catch(e){  
  
  console.log("Oops,error",e)  
  
}
```

优缺点：

符合关注分离，没有讲输入、输出和用事件来跟踪的状态混杂在一个对象内

更好更方便的写法

更加底层，提供了丰富的 API（request，response）

脱离了 XHR，是 ES 规范里的实现方式

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 默认不会带 cookie，需要添加配置项

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

为什么要用 axios？

axios 是一个基于 Promise 用于浏览器和 nodejs 的 HTTP 客户端，本身具

有以下特点

从浏览器中创建 XMLHttpRequest

从 nodejs 发出 http 请求

支持 Promise API

拦截请求和响应

转换请求和响应数据

取消请求

自动转换 JSON 数据

客户端支持防止 CSRF、XSRF

axios 是什么？怎么使用？描述使用它实现登录功能的流程？

请求后台资源的模块。使用 `npm install axios -S` 安装

然后发送的事跨域。需要在配置文件中 `config/index.js` 进行设置。

后台如果是 TP5 则定义一个资源路由 `.js` 中使用 `import` 进行设置，然后 `.get` 或 `.post`。成功返回在 `.then` 函数中，失败返回 `.catch` 函数中

xml 和 json 的区别？

1) 数据体积方面 --- JSON 相对于 XML 来讲，数据体积小，传递的速度更快些

2) 数据交互方面 --- JSON 和 JS 的交互更加方便，更容易解析处理，更好的数据交互

3) XML 对数据描述性比较好

4) JSON 的速度要远远快于 XML

ES6

列举常用的 ES6 特性：

1. let、const
 2. 箭头函数
 3. 类的支持
 4. 字符串模块
 5. symbols
 6. Promises
- 。 。 。

箭头函数需要注意哪些地方？

当要求动态上下文的时候，就不能够使用箭头函数，也就是 this 的固定化。

1. 在使用=>定义函数的时候，this 的指向是定义时所在的对象，而不是使用时所在的对象；
2. 不能够用作构造函数，这就是说，不能够使用 new 命令，否则就会抛出一个错误。
3. 不能够使用 arguments 对象；
4. 不能使用 yield 命令

箭头函数和普通函数之间的区别

1. 箭头函数没有 prototype(原型)，所以箭头函数本身没有 this
2. 箭头函数的 this 在定义的时候继承自外层第一个普通函数的 this。
3. 如果箭头函数外层没有普通函数，严格模式和非严格模式下它的 this 都会指向 window(全局对象)
4. 箭头函数本身的 this 指向不能改变，但可以修改它要继承的对象的 this。
5. 箭头函数的 this 指向全局，使用 arguments 会报未声明的错误。
6. 箭头函数的 this 指向普通函数时,它的 arguments 继承于该普通函数
7. 使用 new 调用箭头函数会报错，因为箭头函数没有 constructor
8. 箭头函数不支持 new.target
9. 箭头函数不支持重命名函数参数,普通函数的函数参数支持重命名
10. 箭头函数相对于普通函数语法更简洁优雅
11. 箭头函数不能当做 Generator 函数,不能使用 yield 关键字(python)

let、const、var

var 声明变量的作用域限制在其声明位置的上下文中，而非声明变量总是全局的

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

let 是更完美的 var，不是全局变量，具有块级函数作用域，大多数情况不会发生变量提升。

1. let 声明的变量具有块级作用域
2. let 声明的变量不能通过 window.变量名访问
3. 形如 for(let x...)的循环是每次迭代都为 x 创建新的绑定
4. let 约束了变量提升
5. let 不允许在相同作用域内重复声明同一个变量名，var 是允许的

const 定义的常量值，不能够重新赋值，如果值是一个对象，可以改变对象里边的属性值。const 变量声明的时候必须初始化

拓展：var 方式定义的变量有什么样的 bug？

1. js 没有块级作用域，在 Js 函数中的 var 声明，其作用域是函数体的全部

var

```
for(var i=0;i<10;i++){  
  
var a = 'a';  
  
}  
  
console.log(a,i); //a 10
```

let

```
for(let i=0;i<10;i++){  
  
let a = 'a';  
  
}  
  
console.log(a,i); //a is not defined
```

2. 循环内变量过度共享

```
for (var i = 0; i < 3; i++) {  
  
    setTimeout(function () {  
  
        console.log(i)  
  
    }, 1000);  
  
} //3 个 3
```

循环本身及三次 timeout 回调均共享唯一的变量 i。当循环结束执行时，i 的值为 3.所以当第一个 timeout 执行时，调用的 i 当然也为 3 了。

Set 数据结构

ES6 中的 Set 方法本身是一个构造函数，它类似于数组，但是成员的值都是唯一的

拓展：数组去重的方法

ES6 set 方法

```
var arr = new Set([1,2,2,3,4]);  
  
console.log([...arr]); // (4) [1, 2, 3, 4]
```

以往去重方法

```
var arr = [1,1,2,2,3,4];

//创建一个空数组用于接收不重复内容的数组

var new_arr = [];

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

    if(new_arr.indexOf(arr[i])!=-1){ //判断 arr[i]在 new_arr 中是否存在相同的内容,
不存在则 push 到数组中

        new_arr.push(arr[i])

    }

}

console.log(new_arr); //(4) [1, 2, 3, 4]
```

箭头函数 this 的指向。

在非箭头函数下，this 指向调用其所在的函数对象，而且是离谁近就指向谁（此对于常规对象，原型链，getter&setter 等都适用）；

构造函数下，this 与被创建的新对象绑定；DOM 事件下，this 指向触发事件的元素；内联事件分为两种情况，bind 绑定，call&apply 等方法改变 this 指向等。而有时 this 也会指向 window

所以 ES6 的箭头函数，修复了原有的 this 指向问题。

手写 ES6 class 继承。

```
//定义一个类
```

```
class Children{
```

```
    constructor(skin,language){

        this.skin = skin;

        this.language = language;

    }

    say(){

        console.log("I'm a Person")

    }

}

class American extends Children{

    constructor(skin,language){

        super(skin,language)

    }

    aboutMe(){

        console.log(this.skin+" "+this.language)

    }

}

var m = new American("张三","中文");

m.say();

m.aboutMe();
```

1)子类没有 constructor

子类 American 继承父类 Person, 子类没用定义 constructor 则默认添加一个, 并且在 constructor 中调用 super 函数, 相当于调用父类的构造函数

数。调用 `super` 函数是为了在子类中获取父类的 `this`，调用之后 `this` 指向子类。也就是父类 `prototype.constructor.call(this)`

2) 子类有 `constructor`

子类必须在 `constructor` 方法中调用 `super` 方法，否则 `new` 实例时会报错。因为子类没有自己的 `this` 对象，而是继承父类的 `this` 对象。如果不调用 `super` 函数，子类就得不到 `this` 对象。`super()` 作为父类的构造函数，只能出现在子类的 `constructor()` 中，但是 `super` 指向父类的原型对象，可以调用父类的属性和方法。

```
const foo = async() => {};
```

ES5 的继承和 ES6 的继承有什么区别？

ES5 的继承实质上是先创建子类的实例对象，然后再将父类的方法添加到 `this` 上(`Parent.apply(this)`)

ES6 继承机制完全不同，实质上是先创建父类的实例对象 `this`(所以必须先调用父类的 `super()` 方法)，然后再用子类的构造函数修改 `this`

具体的：ES6 通过 `class` 关键字定义类，里面有构造方法，类之间通过 `extends` 关键字实现继承。子类必须在 `constructor` 方法中调用 `super` 方法，否则新建实例报错。因为子类没有自己的 `this` 对象，而是继承了父类的 `this` 对象，然后对其进行加工。如果不调用 `super` 方法，子类得不到 `this` 对象

PS：`super` 关键字指代父类的实例，即父类的 `this` 对象。在子类构造函数中，调用 `super` 后，才可使用 `this` 关键字，否则报错

ES5 的继承：

1. 原型链继承
2. 构造函数继承
3. 组合继承
4. 寄生组合继承

ES6 的 extends 继承

super 关键字的使用，新增的静态字段使用，支持对原生构造函数的继承，对 object 继承的差异

ES6 class 的 new 实例和 ES5 的 new 实例有什么区别？

相同点:

1. 实例的属性除非显式定义在其本身(即定义在 this 对象上)，否则都是定义在原型上（即定义在 class 上）
2. 类的所有实例共享一个原型对象

不同点:

1. ES6 用 class 进行实例和普通构造函数进行实例
2. Class 不存在变量提升
3. ES6 为 new 命令引入了一个 new.target 属性(在狗仔函数中) 但会 new 命令作用域的那个构造函数。如果构造函数不是通过 new 命令调用的，new.target 会返回 undefined

generator 生成器函数:

Generator（生成器）是 ES6 标准引入的新数据类型，一个 Generator 看

上去像是一个函数，但可以返回多次。Generator 的声明方式类似一般的函数声明,只是多了个*号,并且一般可以在函数内看到 yield 关键字。

调用 generator 对象有两个方法，一是不断的调用 generator 对象的 next（）方法，第二个方法是直接用 for。。。of 循环迭代 generator 对象。

每调用一次 next，则执行一次 yield 语句，并在该处暂停。return 完成后退出生成器函数，后续如果还有 yield 操作就不再执行了。

```
<script>

function * showWords(){

    yield "one";

    yield "two";

    return 'three'

}

var show = showWords();

console.log(show.next()); //{value: "one", done: false}

console.log(show.next()); //{value: "two", done: false}

console.log(show.next()); //{value: "three", done: true}

console.log(show.next()); //{value: "undefined", done: true}

</script>
```

yield 和 yield*

```
//yield

function * showWords(){

    yield "one";
```

```
    yield showNumber();

    return 'three';
}

// yield *

function * showWords2(){

    yield "one";

    yield* showNumber(2,3);

    return 'three';
}

function * showNumber(a,b){

    yield a+b;

    yield a*b;
}

var show = showWords();

console.log(show.next()); //{value: "one", done: false}

console.log(show.next()); //{value: "showNumber", done: false}

console.log(show.next()); //{value: "three", done: true}

var show2 = showWords2();

console.log(show2.next()); //{value: "one", done: false}

console.log(show2.next()); //{value: 5, done: false}

console.log(show2.next()); //{value: 6, done: true}

console.log(show2.next()); //{value: "three", done: true}
```


什么是 **async/await** 及其如何工作？

async/await 是 JS 中编写异步或非阻塞代码的新方式。它建立在 Promises 之上，让异步代码的可读性和简洁度都更高。**async** 关键声明函数会隐式返回一个 Promise。**await** 关键字只能在 **async function** 中使用，在 **非 async function** 的函数中使用 **await** 关键字都会抛出错误。**await** 关键字在执行下一行代码之前等待右侧表达式（可能是一个 Promise）返回

Promise 和 **async await** 以及它们之间的区别：

promise: 是用于异步编程的解决方案，主要用于异步编程

promise 共有三个状态：pending（执行中）、success（成功）、rejected（失败）

async、await 是将异步强行转换为同步处理。

async/await 与 **promise** 不存在谁代替谁的说法，因为 **async/await** 是寄生于 **Promise**，**Generator** 的语法糖。

区别：

1 **promise** 是 ES6，**async/await** 是 ES7

2 **async/await** 相对于 **promise** 来讲，写法更加优雅

3 **reject** 状态：

- 1) **promise** 错误可以通过 **catch** 来捕捉，建议尾部捕获错误，
- 2) **async/await** 既可以用 **.then** 又可以用 **try-catch** 捕捉

async 函数的基本用法:

async 函数返回一个 Promise 对象，可以使用 then 方法添加回调函数。当函数执行的时候，一旦遇到 await 就会先返回，等到异步操作完成，再接着执行函数体内后面的语句。函数前面的 async 关键字，表明该函数内部有异步操作。调用该函数时，会立即返回一个 Promise 对象。由于 async 函数返回的是 Promise 对象，可以作为 await 命令的参数。

ES6 async 函数有多种使用形式:

```
//函数声明

async function foo(){}

//函数表达式

const foo = async function(){}


//对象的方法

let obj = {async foo(){}

obj.foo().then(...)
```

```
//class 方法

class Storage{

  constructor(){

    this.cachePromise = cache.open("avatars")

  }

  async getAvatar(name){
```

```
const cache = await this.cachePromise;

return cache.match(`/avatars/${name}.jpg`)

}

}
```

```
//箭头函数
```

async 与 generator 的区别？

async 函数是 Generator 函数的语法糖，async 函数就是将 Generator 函数的星号（*）替换成 async，将 yield 替换成 await。

async 函数对 Generator 函数的改进，体现在以下四点：

1、内置执行器

2、更好的语义：async 表示函数里有异步操作，await 表示紧跟在后面的表达式需要等待结果

3、更广的适用性：async 函数的 await 命令后面，可以是 Promise 对象和原始类型的值（数字、字符串和布尔值，但这时等同于同步操作）

4、返回值是 Promise：async 函数的返回值是 Promise 对象，这比 Generator 函数的返回值是 Iterator 对象方便多了，可以用 then 方法指定下一步的操作。

进一步说，async 函数完全可以看作多个异步操作包装成的一个 Promise 对象，而 await 命令就是内部 then 命令的语法糖

简单实现 async/await 中的 async 函数

async 函数的实现原理，就是将 Generator 函数和自动执行器，包装在一个函数里

```
function spawn(genF){
  return new Promise(function(resolve,reject){
    const gen = genF();
    function step(nextF){
      let next;
      //ry/catch/finally 语句用于处理代码中可能出现的错误信息。
      try {
        next = nextF()
      }catch(e){
        return reject(e);
      }
      if(next .done){
        return resolve(next.value);
      }
      Promise.resolve(next.value).then(
        function(v){
          step(function(){
            return gen.next(v)
          });
        },
        function(v){
          step(function(){
            return gen.throw(e)
          });
        }
      )
    }
    step(function(){
      return gen.next(undefined);
    });
  })
}
```

有用过 promise 吗？请写出下列代码的执行结果，并写出你的理解思路：

```
setTimeout(()=>{  
  
    console.log(1);  
  
}, 0);  
  
new Promise((resolve)=>{  
  
    console.log(2);  
  
    for(var i = 1; i < 200; i++){  
  
        i = 198 && resolve();  
  
    }  
  
    console.log(3);  
  
}).then(()=>{  
  
    console.log(4);  
  
});console.log(5);
```

首先要讲一下，js 是单线程执行，那么代码的执行就有先后；

有先后，那就要有规则(排队)，不然就乱套了，那么如何分先后呢？大体

分两种：同步、异步；

同步很好理解，就不用多说了(我就是老大,你们都要给我让路)；

异步(定时器[setTimeout , setInterval]、事件、ajax、promise 等),

说到异步又要细分宏任务、微任务两种机制,

宏任务: js 异步执行过程中遇到宏任务, 就先执行宏任务, 将宏任务加入执行的队列(event queue), 然后再去执行微任务;

微任务: js 异步执行过程中遇到微任务, 也会将任务加入执行的队列(event queue), 但是注意这两个 queue 身份是不一样的, 不是你先进来, 就你先出去的(就像宫里的皇上选妃侍寝一样, 不是你先进宫(或先来排队)就先宠幸的), 真执行的时候是先微任务里拿对应回调函数, 然后才轮到宏任务的队列回调执行的;

理解了顺序, 那上面的结果也就不难懂了。

Object.is () 与原来的比较操作符===, ==的区别?

==相等运算符, 比较时会自动进行数据类型转换

===严格相等运算符, 比较时不进行隐式类型转换

Object.is 同值相等算法, 在===基础上对 0 和 NaN 特别处理

```
+0 === -0 //true
```

```
NaN === NaN // false
```

```
Object.is(+0, -0) // false
```

```
Object.is(NaN, NaN) // true
```

介绍一下 Set、Map、WeakSet 和 WeakMap 的区别?

Set： 成员不能重复

只有键值，没有键名

可以遍历，方法有 add, delete, has

WeakSet：成员都是对象

成员都是弱引用，随时可以消失，可以用来保存 DOM 节点，

不容易造成内存泄漏

不能遍历，方法有 add, delete , has

Map: 本质上是键值对的集合,类似集合

可以遍历，方法很多，可以跟各种数据格式转换

WeakMap：只接受对象作为键名（null 除外），不接受其他类型的值作为键名

键名所指向的对象，不计入垃圾回收机制

不能遍历，方法有 get、set、has、delete

ES6 新特性详细介绍说明：

变量声明：const 和 let：

ES6 推荐使用 let 声明局部变量，相比之前的 var（无论声明在何处，都会被视为声明在函数的最顶部），而 let 不会将声明变量提前；

let 表示声明变量，而 const 表示声明常量，两者都为块级作用域；const 声明的变量都会被认为是常量，意思就是它的值被设置完成后就不能再修改了。

注意：let 关键词声明的变量不具备变量提升的特性

const 和 let 声明只在最高进的一个块中（花括号内）有效

const 在声明时必须被赋值

箭头函数：

箭头函数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

箭头函数最直观的三个特点：

不需要 function 关键字来创造

省略 return 关键字

修复了 this 指向

类和继承：

class 和 extend 是一种语法糖，也是基于原型继承实现的

class 和 super calls，实例化，静态方法和构造函数

```
<script>

//class 声明类 内部直接是属性和方法 不用分隔。 constructor

class Person{

  constructor(name, age){

    this.name = name;//实例属性

    this.age = age;

    console.log(name,age)

  }

  sayhi(name){
```



```
//使用 ES6 新特性字符串模块,注意外层包裹符号是``反引号来创建字符串,内部
变量使用${}

    console.log(`this name is ${this.name}`);

    //以往的写法

    console.log('my age is '+this.age)

}

}

class Programmer extends Person{

    constructor(name,age){

        //直接调用父类结构器进行初始化

        super(name,age)

    }

    program(){

        console.log("I'm coding...")

    }

}

var anim = new Person("张三",18);

anim.sayhi();

var wayou = new Programmer("李四",20);

wayou.sayhi();
```

```
wayou.program();
```

```
</script>
```

字符串模板：

ES6 中允许使用反引号`来创建字符串，此方法创建的字符串里面可以包含由`\${}`包裹的变量

```
<script>
```

```
//产生一个随机数
```

```
var num = Math.random();
```

```
//将这个数字输出到 console
```

```
console.log(`输出随机数${num}`);
```

```
</script>
```

增强的对象字面量：

对象字面量被增强了，写法更加简洁与玲姐，同时在定义对象的时候能够做的事情更多了。具体表现在：

1. 可以在对象子面量里面定义原型
2. 定义方法可以不用 function 关键词
3. 直接调用父类方法

```
<script>
```

```
var human = {
```

```
  breathe(){
```

```
    console.log('breathing...');
```

```
  }
```

```
};

var worker = {

  __proto__:human, //设置此对象的原型为 human,想党羽继承 human

  company:'freelancer',

  work(){

    console.log("working..")

  }

}

human.breathe();

//调用继承来的 breathe 方法

worker.breathe();

</script>
```

解构：

自动解析数组或对象中的值。若一个函数要函数要返回多个值，常规的做法是返回一个对象，将每个值作为这个对象的属性返回。在 ES6 中，利用解构这一特性，可以直接返回一个数组，然后数组中的值会自动被解析到对应接收该值得变量中。

```
<script>

var [x,y] = getVal(), //函数返回值解析

  [name, ,age] = ["wayou","male","secret"]; //数组解析

function getVal(){

  return [1,2];
```

```
}

console.log(`x:${x},y:${y}`); //x:1,y:2

console.log(`name:${name},age:${age}`); //name:wayou,age:secrect

</script>
```

参数默认值，不定参数，拓展参数：

默认参数值：

可以在定义函数的时候指定参数的默认值，而不用像以前那样通过逻辑或操作符来达到目的了。

```
<script>

//传统方式设置默认方式

function sayHello(name){

    var name = name||'dude';

    console.log("Hello "+name);

}

sayHello(); //Hello dude

sayHello("wayou"); //Hello wayou


//ES6 的默认参数

function sayHello2(name = "dude"){

    console.log("Hello "+name)

}

}
```

```
sayHello2(); //Hello dude  
  
sayHello2("wayou"); //Hello wayou  
  
</script>
```

不定参数（拓展符）：

不定参数是在函数中使用命名参数同时接收不定数量的未命名参数。这只是一中语法糖，在以前的 JavaScript 代码中我们可以通过 arguments 变量来达到这一目的。不定参数的格式是三个句点后跟代表所有不定参数的变量名。比如下面这个例子中，...x 代表了所有传入 add 函数的参数。

```
<script>  
  
//将所有参数想加的函数  
  
function add(...x){  
  
    return x.reduce((m,n)=>m+n);  
  
}  
  
//传递任意个数的参数  
  
console.log(add(1,2,3)); //输出:6  
  
console.log(add(1,2,3,4,5)); //输出:15  
  
</script>
```

reduce () 方法接收一个函数作为累加器，数组中的每个值（从左到右）开始缩减，最终计算为一个值。

reduce () 可以作为一个高阶函数，用于函数的 compose

注意：reduce () 对于空数组是不会执行回调函数的

拓展符：将一个数组转为用逗号分隔的参数序列。（若数组为空不产生

任何效果)

```
<script>

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

console.log(x); //(6) [1, 2, 3, 4, 5, 6]

console.log(...x); //1 2 3 4 5 6

</script>
```

拓展参数：

拓展参数则是另一种形式的语法糖，它允许传递数组或类数组直接作为函数的参数而不用通过 apply。

```
<script>

var people = ['wayou','john','sherlock'];

//sayHello 函数来接收三个单独的参数

function sayHello(people1,people2,people3){

    console.log(`Hello ${people1},${people2},${people3}`)

}

//以前的方式,如果需要传递数组当参数,我们需要使用函数 apply 方法

sayHello.apply(null,people); //Hello wayou,john,sherlock

sayHello(...people); //Hello wayou,john,sherlock

</script>
```

for of 值遍历：

for in 循环用于遍历数组，类数组或对象，ES6 中新引入的 for of 循环功能相似，不同的是每次循环他提供的不是序号而是值

```
<script>

var someArray = ['a','b','c'];

for(v of someArray){

    console.log(v); //a,b,c

}

</script>
```

iterator/generator:

iterator: 它是这么一个对象，拥有一个 next 方法，这个方法返回一个对象{done, value}，这个对象包含两个属性，一个布尔类型的 done 和包含任意值的 value。

iterable: 这是这么一个对象，拥有一个 obj[@@iterator]方法，这个方法返回一个 iterator

generator: 它是一个特殊的 iterator。反的 next 方法可以接收一个参数并且返回值取决于它的构造函数（generator function）。generator 同时拥有一个 throw 方法。

generator 番薯：即 generator 的构造函数。此函数内可以使用 yield 关键字，在 yield 出现的地方可以通过 generator 的 next 或 throw 方法向外界传递值。generator 函数是通过 function*来声明的。

yield 关键字：它可以暂停函数的执行，随后可以再进入函数继续执行

具 体 详 情 :

<https://blog.domenic.me/es6-iterators-generators-and-iterables/>

模块：

在 ES6 标准中，Javascript 原生支持 module 了。这种将 JS 代码分割成不同功能的小块进行模块化的概念是在一些三方规范中流行起来的，比如 CommonJS 和 AMD 模式。

将不同功能的代码分别写在不同文件中，各模块只需导出公共接口部分，然后通过模块的导入方式可以在其他地方使用。

```
<script>

//单独的 js 文件,如:point.js

module "point" {

  export class Point {

    constructor (x,y){

      publice x = x;

      publice y = y;

    }

  }

}

//在需要引用模块的 js 文件内

//声明引用的模块

module point from './point.js';

//这里可以看出,尽管声明了引用的模块,还是可以通过指定需要的部分进行导入

import Point from "point"
```



```
var origin = new Point(0,0);

console.log(origin)

</script>
```

Map、Set 和 WeakMap、WeakSet

这些是新加的集合类型，提供了更加方便的获取属性值的方法，不用像以前一样用 `hasOwnProperty` 来检查某个属性是属于原型链上的还是当前对象的。同时，在进行属性值添加与获取时有专门的 `get`、`set` 方法。

```
//Sets

var s = new Set();

s.add("hello").add("goodbye").add("hello");

s.size === 2;

s.has("hello")===true;

//Maps

var m = new Map();

m.set("hello",42);

m.set(s,34);

m.get(s) === 34;
```

我们会把对象作为一个对象的键来存放属性值，普通集合类型比如简单对象会阻止垃圾回收器对这些作为属性键存在的对象回收，偶造成内存泄露的危险。而 `weakMap`，`weakSet` 则更加安全些，这些作为属性键的对象如果没有别的变量在引用它们，则会被回收释放掉，具体还看下面的例子。

```
//weak Maps
```

```
var wm = new WeakMap();

wm.set(s,{eatra:42});

wm.size === undefined;


//weak Sets

var ws = new WeakSet();

ws.add({data:42}); //因为添加到 ws 的这个临时对象没有其他变量引用它,所以 ws 不会保存它的值,也就是说这次添加其实没有意思
```

Proxies

proxy 可以监听对象身上发生了什么事情，并在这些事情发生后执行一些相应的操作。一下子让我们对一个对象有了很强的跟踪能力，同时咋数据绑定方面也很有用处。

```
<script>

//定义被侦听的目标对象

var engineer = {name:"Join",salary:50};

//定义处理程序

var interceptor = {

  set:function(receiver,property,value){

    console.log(property,"is changed to",value);

    receiver[property] = value;

  }

}
```

```
//创建代理以进行侦听

engineer = new Proxy(engineer,interceptor);

//做一些改动来触发代理

engineer.salary = 60; //控制台输出:salary is changed to 60

</script>
```

上面代码，我们已经添加了注释，这里进一步解释。对于处理程序，是在被侦听的对象身上发生了相应事件之后，处理程序里面的方法会被调用，上面例子中我们设置了 set 的处理函数，表明。如果我们侦听对象的属性被更改，也就是被 set 了，那这个处理程序就会被调用，同时通过参数能够的值是哪个属性被更改，更改为什么值

symbols

symbols 是一种基本类型，像数字、字符串还有布尔一样。它不是一个对象。symbols 通过调用 symbol 函数产生，接收一个可选的名字参数，该函数返回的 symbol 是唯一的。之后就可以用这个返回值作为对象的键了。symbol 还可以用来创建私有属性，外部无法直接访问偶 symbol 作为键的属性值。

```
<script>

var MyClass = (function() {

    // module scoped symbol

    var key = Symbol("key");

    function MyClass(privateData) {

        this[key] = privateData;
```

```
    }

    MyClass.prototype = {

        doStuff: function() {

            this[key]

        }

    };

    return MyClass;

})();

var c = new MyClass("hello")

console.log(c["key"] === undefined); //true,无法访问该属性,因为是私有的

</script>
```

Math、Number、String、Object 的新 API

对 Math、Number、String 还有 Object 等添加了许多新的 API。

Promises

Promise 是处理异步操作的一种模式，之前在很多三方库中有实现，比如 JQuery 的 deferred 对象。当你发起一个异步请求，并绑定了.when()/.done()等事件处理程序时,其实就是在应用 promise 模式。

```
<script>

    //创建 Promise

    var promise = new Promise(function(resolve,reject){

        //进行一些异步或耗时操作

        if(/*如果成功*/){
```

```
        resolve("stuff worked")

    }else{

        reject(Error("It broke"));

    }

});

//绑定处理程序

promise.then(function(result){

    //promise 成功的话会执行这里

    console.log(result); //"stuff worked"

},function(err){

    //promise 处理失败会执行这里

    console.log(err); //Error:"It broke"

});

</script>
```

Vue

什么是 MVVM?

MVVM 分为 Model、View、ViewModel 三者。

Model 代表数据模型，数据和业务逻辑都在 Model 层中定义；

View 代表 UI 视图，负责数据的展示；

ViewModel 负责监听 Model 中数据的改变并且控制视图的更新，处理用户交互操作；

Model 和 View 并无直接关联，而是通过 ViewModel 来进行联系的，Model 和 ViewModel 之间有着双向数据绑定的联系。因此当 Model 中的数据改变时会触发 View 层的刷新，View 中由于用户交互操作而改变的数据也会在 Model 中同步。

这种模式实现了 Model 和 View 的数据自动同步，因此开发者只需要专注对数据的维护操作即可，而不需要自己操作 dom。

mvvm 和 mvc 的区别？它和其他框架（JQuery）的区别是什么？哪些场景适合？

mvc 和 mvvm 其实区别不大。都是一种设计思想，主要 mvc 中 controller 演变成 mvvm 中的 ViewModel。mvvm 主要解决了 mvc 中大量的 DOM 操作使页面渲染性能降低，加载速度变慢，影响用户体验

区别：vue 数据驱动，通过数据来显示视图层而不是节点操作

场景：数据操作比较多的场景，更加便捷

Vue 的优点是什么？

1. 低耦合：视图 View 可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同给的“view”上，当 View 变化的时候 Model 可以不变，当

Model 变化的时候 View 也可以不变。

2. 可重用性：你可以把一些视图逻辑放在 ViewModel 里面，让很多 view 重用这段视图逻辑

3. 独立开发：开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计

4. 可测试：界面素来是比较难测试的，而现在测试可以针对 ViewModel 来写

Vue.js 的两个核心是什么？

数据驱动、组件系统

Vue 组件之间的传值

-父组件与子组件传值

父组件通过标签上定义传值(:父组件名称="子组件 props 的名称")

子组件通过 props 的方式接受数据

-子组件向父组件传递数据

子组件通过\$emit 方法传递参数给父组件

Vue-cli 中怎么使用自定义组件，又遇到过哪些问题吗？

1) 在 components 文件中创建组件.vue 文件.是 script 中 export default{}

2) 在调用的页面中使用 import XXX from 路径

3) 在调用的组建 components 属性中注册组件，注意大小写（文档不接受-连字符，可以使用驼峰命名）

Vue 如何实现按需加载配合 webpack 设置

webpack 中提供了 require.ensure() 来实现按需加载。以前引入路由是通过 import 这样的方式引入，现在改为 const 定义的方式他引入。

不进行页面按需加载引入方式：import home from './XXX'

进行页面按需加载的引入方式：const home = './XXX'

v-show 和 v-if 指令的共同点和不同点

v-show 指令是通过修改元素的 display 的 CSS 属性让其显示或隐藏

v-if 指令是直接销毁和重建 DOM 节点，达到让元素显示和隐藏的效果

如何让 CSS 只在当前组件中起作用

在当前文档中使用 style 标签，并添加 scope 属性

<keep-alive></keep-alive>的作用是什么？

keep-alive 标签包裹动态组件时，会缓存不活动的组件实例，主要用于保留组件状态或避免重新渲染

聊聊 Keep-alive 的实现原理和缓存策略

原理：

keep-alive 的实现正是用到了 LRU 策略，将最近访问的组件 push 到

this.keys 最后面, this.keys[0] 也就是最久没有被访问到的组件, 当缓存实例超过 max 设置值, 删除 this.keys[0]

LRU 缓存淘汰算法: LRU 算法根据数据的历史访问记录来进行记录, 其核心思想是” 如果数据最近被访问过, 那么将来被访问的几率也更高”

Vue 中引入组件的步骤

1) 采用 ES6 的 import...from...语法或 CommonJS 的 require()方法引入组件

2) 对组件进行注册

```
// 注册

Vue.component('my-component', {

  template: '<div>A custom component!</div>'

})
```

3) 使用组件<my-component></my-component>

Vue 常用的修饰符?

.precent: 提交事件不在重载页面

.stop: 组织但即使单击事件冒泡

.self: 当事件发生在该元素本身而不是子元素的时候会触发

.capture: 事件侦听, 事件发生的时候会调用

什么是 Vue 的计算属性?

在模板中放入太多的逻辑会让模板过重且难以维护，在需要对数据进行复杂处理，且可能多次使用的情况下，尽量采用计算属性的方式。

好处：

1. 使得数据处理结构清晰
2. 依赖于数据，数据更新，处理结果自动更新
3. 计算属性内部部 this 指向 vm 实例
4. 在 template 调用时，直接写计算属性名即可
5. 常用的 getter 方法，获取数据，也可以使用 set 方法改变数据
6. 相较于 methods，不管依赖的数据变不变，methods 都会重新计算，但依赖数据不变的时候，computed 从缓存中获取，不会重新计算

Vue 等单页面应用及其优缺点？

优点：Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件，核心是一个响应的数据绑定系统。MVVM、数据驱动、组件化、轻量、简洁、高效、快速、模块友好

缺点：不支持低版本的浏览器，最低只支持 IE9；不利于 SEO 的优化（如果要支持 SEO，建议通过服务端来进行渲染组件）；第一次加载首页耗时相对长一点；不可以使用浏览器的导航按钮需要自行实现前进后退

指令 v-el 的作用是什么？

提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标，可以是 CSS 选择器，也可以是一个 HTML 元素实例

在 Vue 中使用插件的步骤

采用 ES6 的 `import...from...` 语法或 CommonJS 的 `require()` 方法引入插件

使用全局方法 `Vue.use(plugin)` 使用插件，可以传图一个选项对象

```
Vue.use(MyPlugin,{someOption:true })
```

active-class 是哪个组件的属性？

vue-router 模块的 router-link 组件

说出至少 4 中 vue 当中的指令和它的用法？

v-if：判断是否为真，然后重组、销毁 DOM 节点

v-for：数据循环

v-bind：class 绑定一个属性

v-model：实现双向绑定

v-on：添加事件

v-else：配合 v-if 使用

vue-loader 是什么？使用它的用途有哪些？

解析：vue 文件的一个加载器

用途：js 可以写 es6、style 样式可以 scss、template 可以加等

scss 是什么？在 vue-cli 中安装使用步骤？有几大特性？

scss 是 css 的预编译。

使用步骤：

1. 先安装 css-loader、node-loader、scss-loader 等加载器模块
2. 在 build 目录找到 webpack.base.config.js，在那个 extends 属性中加一个拓展 scss
3. 在同一个文件，配置一个 module 属性
4. 在组件的 style 标签上添加 lang 属性，lang= “scss”

特性：

可以使用变量（\$变量名=值）；

可以用混合器

可以嵌套

Vue 中的 Key 到底有什么用？

Key 是给每一个 vnode 的唯一 id，依靠 key，我们 diff 操作可以更准确、更快速（对于简单列表页面渲染来说 diff 节点也更快，但会产生一些隐藏的副作用，比如可能不会产生过渡效果，或者在某些节点有绑定数据（表单）状态，会出现状态错位）

diff 算法的过程中，先会进行新旧节点的首尾交叉对比，当无法匹配的时候会用新节点的 key 与旧节点进行比对，从而找到相应的旧节点

更准确：因为带 Key 就不会出现重复现象，在 sameNode 函数 `a.key === b.key` 对比中可以避免就地复用的情况。所以会更加准确，如果不加 key 会导致之前节点的状态被保留下来，会产生一系列的 bug

更快速：key 的唯一性可以被 Map 数据结构充分利用，相比于遍历查找的

时间复杂度 $O(n)$ ，Map 的时间复杂度仅仅为 $O(1)$

谈一谈 nextTick 的原理

Vue 的 nextTick 方法的实现原理

1. Vue 用异步队列的方式来控制 DOM 更新和 nextTick 回调先后执行
2. microtask 因为其高优先级特性，能确保队列中的微任务在一次事件循环前辈执行完毕
3. 考虑兼容问题做了 microtask 向 macrotask 的降级方案

详解：

js 执行是单线程的，它是基于事件循环的。

主线程的执行过程就是一个 tick，而所有的异步结果都是通过“任务队列”来调度。消息队列中存放的是一个一个的任务（task）。规范中的 task 分为两大类，分别是 macro task 和 micro task，并且每个 macro task 结束后，都将清空所有 micro task。

在浏览器环境中，常见的 macro task 有 setTimeout、MessageChannel、postMessage、setImmediate

常见的 micro task 有 MutationObserver 和 Promise.then

Vue 在更新 DOM 时是异步执行的，只要侦听到数据变化，Vue 将开启一个队列，并缓冲在同一事件循环中发生的所有数据变更。

在 Vue2.5 的源码中，macrotask 降级的方案依次是：setImmediate、MessageChannel、setTimeout

为什么避免 v-if 和 v-for 用在一起？

当 vue 处理指令时，v-for 比 v-if 具有更高的优先级，通过 v-if 移动到容器元素，不会再重复遍历列表中的每个值，取而代之的事，我们只检查它一次，且不会再 v-if 为否的时候运行 v-for。

VNode 是什么？虚拟 DOM 是什么？

Vue 在羽绒棉上渲染的节点，及其子节点成为“虚拟节”，简称为“VNode”。“虚拟 DOM”是由 Vue 组件树简历起来的整个 VNode 树的称呼

MINI UI 是什么？怎么使用？说出至少三个组件的使用方法？

基于 vue 前端的组件库。使用 npm 安装，然后 import 样式和 js；

vue.use(miniUi)全局引入。在单个组件局部引入；

import {Toast} from 'mini-ui'；

组件一： Toast（‘登录成功’）

组件二： mint-header；

组件三： mint-swiper

自定义指定(v-check/v-focus)的方法有哪些？有哪些钩子函数？还有哪些钩子函数参数？

全局定义指令：在 vue 对象的 directive 方法里面有两个参数，一个是指令名称，另外一个函数。组件内定义指令：directives 钩子函数：bind（绑定事件触发）、inserted（节点插入的时候触发）、update（组件内相关更新）

钩子函数的参数：el、binding

Vue 封装组件的过程

首先，使用 `Vue.extend()` 创建一个组件

再使用 `Vue.component()` 方法注册组件

接着，如果子组件需要数据，可以在 `props` 中接受定义

最后，子组件修改好数据后，想把数据传递给父组件，可以使用 `emit()` 方法

Vue 的双向数据绑定原理是什么？

`vue.js` 是采用数据劫持介入发布-订阅者模式的方法，通过 `Object.defineProperty()` 来劫持各个属性的 `setter` 和 `getter`，在数据变动时发布消息给订阅者，触发相应的监听回调。

`observer`: `Vue` 中使用 `Observer` 类来管理上述响应式化 `Object.defineProperty` 的过程。它的作用是给对象的属性添加 `getter` 和 `setter`，用来依赖收集和派发更新

`Dep`: 用于收集当前响应式对象的依赖关系，每个响应式对象包括子对象都拥有一个 `DEp` 实例（里面 `subs` 是 `Watcher` 实例数组），当数据有变更时，会通过 `dep.notify()` 通知各个 `watcher`

`Watcher`: 观察者对象，实例分为 `watch(render watcher)`, 计算属性 `watcher(computed watcher)`, 侦听器 `watcher(user watcher)` 三种

Watch 和 Dep 的关系

watcher 中实例化了 dep 并向 dep.subs 中添加了订阅者, dep 通过 notify 遍历了 dep.subs 通知每个 watcher 更新。

依赖收集

1. initState 时, 对 computed 属性初始化时, 触发 computed watcher 依赖收集
2. initState 时, 对侦听属性初始化是, 触发 user watcher 依赖收集
3. render () 的过程, 触发 render watcher 依赖收集
4. re-render 时, vm.render()再次执行, 会溢出所有 subs 中的 watcher 的订阅, 重新赋值

派发更新

1. 组件对响应的数据进行了修改, 触发 setter 的逻辑
2. 调用 dep.notify()
3. 遍历所有的 subs(Watcher 实例),调用每一个 watcher 的 update 方法

简单描述每个周期具体适合哪些场景

生命周期钩子的一些使用方法:

beforecreate: 可以在这加个 loading 事件, 在加载实例时触发

created: 初始化完成时的时间写在这里, 如在这结束 loading 事件, 异步请求也适宜在这里调用

mounted：挂载元素，获取到 DOM 节点

updated：如果对数据统一处理，在这里写上相应函数

beforeDestroy：可以做一个确认停止事件的确认框

nextTick：更新数据后立即操作 DOM

打包 Vue 项目命令是什么？

npm run build

在 Vue 中同时发送多个请求怎么操作？

1. 创建两个 Promise，在 Promise 中使用 axios
2. 调用 Promise.all([p1,p2]).then(res =>{ }).catch(err => { }) 方法

举例说明：

```
getInfo(){
  //创建 promise,在 promise 中调用 axios , then 里使用 resolve 回调, catch 里使用 reject
  回调
  var p1 = new Promise((resolve,reject) => {
    this.$axios.get(httpUrl.getUser).then(res => {
      resolve(res);
    }).catch(err =>{
      reject (err);
    })
  })

  var p2 = new Promise((resolve,reject) => {
    this.$axios.get(httpUrl.getCompany).then(res => {
      resolve(res);
    }).catch(err =>{
      reject (err);
    })
  })

  //调用 Promise.all().then(res => {})
  Promise.all([p1,p2]).then(res => {
    console.log(res);
  })
}
```

```
  })  
}
```

vue 不使用 v-model 的时候怎么监听数据变化

可以使用 watch 监听数据的变化

computed 的实现原理

computed 本质是一个惰性求值的观察者

computed 内部实现了一个惰性的 watcher, 也就是 computed watcher, computed watcher 不会立刻求值, 同时持有一个 dep 实例

其内部通过 this.dirty 属性标记计算属性是否需要重新求值

当 computed 的依赖状态发生改变时, 就会通知这个惰性的 watcher

computed watcher 通过 this.dep.subs.length 判断有没有订阅者

有的话, 会重新计算, 然后对比新旧值, 如果变化了, 会重新渲染(Vue 想确保不仅仅是计算属性依赖的值发生变化, 而是当计算属性最终计算的值发生变化时才会触发渲染, watcher 重新渲染, 本质上是一种优化。)

没有的话, 仅仅把 this.dirty = true (当计算属性依赖于其他数据时, 属性并不会立即重新计算, 只有之后其他地方需要读取属性的时候, 它才会真正计算, 即具备 lazy(懒计算)特性)

computed 和 watch 有什么区别及运用场景?

区别:

computed 计算属性: 依赖其它属性值, 并且 computed 的值有缓存, 只有

它依赖的属性值发生改变，下一次获取 computed 的值时才会重新计算 computed 的值

watch 侦听器：更多的是[观察]的作用，无缓存性，类似于某些数据的监听回调，每当监听的数据变化时都会执行回调进行后续操作

运用场景

的那个我们需要进行数值计算，并且依赖于其他数据时，应用使用 computed，因为可以利用 computed 的缓存性，避免每次获取值时，都要重新计算

当我们需要的数据变化时执行异步或开销较大的操作时，应该使用 watch，使用 watch 选项允许我们执行异步操作(访问一个 API),限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的

为什么在 Vue3.0 才用了 Proxy，抛弃了 object.defineProperty？

object.defineProperty 本身有一定的监控到数组下标变化的能力，但是在 Vue 中，从性能/体验的性价比考虑，尤大大就弃用了这个特性(Vue 为什么不能检测数组变动) 为了解决这个问题，经过 vue 内部处理后可以使用以下几种方法来监听数组

push/pop/shift/unshift/splice/sort/reverse

由于只针对了以上 7 种方法进行了 hack 处理，所以其他数组的属性也是检测不到的，还是具有一定的局限性。

Object.defineProperty 只能劫持对象的属性，因此我么需要对每个对象的每个属性进行遍历。Vue2.0 中通过递归+遍历 data 对象来实现对数据的监控，如果属性值也是对象 Proxy 可以劫持整个对象，并返回一个新的对象.Proxy 不仅可以代理对象，还可以代理数组。还可以代理动态增加的属性

Vue 是如何对数组方法进行变异的？

Vue 通过原型拦截的方式重写了数组的 7 个方法，首先获取到这个数组的 ob，也就是它的 Observer 对象，如果有新的值，就调用 observeArray 对新的值进行监听，然后手动调用 notify，通知 render watcher 执行 update

Vue 组件 data 为什么必须是函数？

因为组件是可以复用的，JS 里对象是引用关系，如果组件 data 是一个对象，那么子组件中的 data 属性值会互相污染，产生副作用。

所以一个组件的 data 选项必须是一个函数，因此每个实例可以维护一份被返回对象的独立拷贝。new Vue 的实例是不会被复用的，因此不存在以上问题。

说说 Vue 的渲染过程

调用 compile 函数，生成 render 函数字符串，编译过程如下：

1. parse 函数解析 template 生成 ast(抽象语法树)

2. optimize 函数优化静态节点(标记不需要每次都更新的内容, diff 算法会直接跳过静态节点，从而减少比较的过程，优化了 patch 的过程)

generate 函数生成 render 函数字符串

1. 调用 new Watcher 函数，监听数据的变化，当数据发生变化时，render 函数执行生成 vnode 对象
2. 调用 patch 方法，对比新旧 vnode 对象，通过 DOM diff 算法添加、修改、删除真正的 DOM 元素

Vue 与 Angular 以及 React 的区别？

与 AngularJS 的区别：

相同点：都支持指令：内置指令和自定义指令；都支持过滤器；内置过滤器和自定义过滤器；都支持双向数据绑定；都不支持低端浏览器

不同点：AngularJS 的学习成本高，比如增加了 Dependency Injection 特性，而 Vue.js 本身提供的 API 都比较简单、直观；在性能上，AngularJS 依赖对数据做脏检查，所以 Watcher 越多越慢；Vue.js 使用基本依赖追踪的观察并且使用异步队列更新，所有的数据都是独立触发的

与 React 的区别：

相同点：React 采用特殊的 JSX 语法，Vue.js 在组件开发中也推崇编写 .vue 特殊文本格式，对文本内容都有一些约定，两者都需要编译后使用；中心思想相同：一切都是组件，组件实例之间可以嵌套；都提供合理的钩子函数，可以让开发者定制化地去处理需求；都不内置列数 Ajax/Route 等功能到核心包，而是以插件的方式加载；在组件开发中都支持 mixins 的特性。

不同点：React 采用的 Virtual DOM 会对渲染出来的结果做脏检查；Vue.js 在模板中提供了指令，过滤器等，可以非常方便，快捷地操作 Virtual DOM.

vue3.0 今年发布了，请你说一下他们之间在响应式的实现上有什么区别？

vue2 采用的是 `defineProperty` 去定义 `get,set` 来劫持和监听数据, vue3 该用了 `proxy`, 也代表着 vue 放弃了兼容 ie

像 vue-router、vuex 他们都是作为 vue 插件，请说一下他们分别都是如何在 vue 中生效的？

通过 vue 的插件系统，用 `vue.mixin` 混入到全局，在每个组件的生命周期的某个阶段注入组件实例

请你说一下 vue 的设计架构

vue2 采用的是典型的混入式架构，类似于 `express` 和 `jquery`，各部分分模块开发，再通过一个 `mixin` 去混入到最终暴露到全局的类上

Vue 项目优化方式:

1. 组件优化 -- 提高组件复用性
2. vue-router 优化 --- 使用路由懒加载
3. v-if 和 v-show/ `computed` 和 `watch` 区分使用场景
4. v-for 遍历必须添加 `key` 值，同时避免使用 `v-if`
5. 事件销毁 --- 在声明周期销毁阶段，销毁一些不必要的数据、定时器等
6. 图片懒加载
7. 第三方插件按需引入

8. 提取公共代码

9. 减少 ES6 转为 ES5 的冗余代码

还有很多，剩余详情地址：

https://blog.csdn.net/qq_37939251/article/details/100031285

vue 首屏加载过慢怎么优化？

vue 作为一个单页面应用，如果不对路由进行处理，在加载首页的时候，就会将所有组件全部加载，并向服务器请求数据，这必将拖慢加载速度。

解决方案：

1. vue-router 懒加载

就是按需加载组件，只有当路由被访问时才会加载对应组件，而不是在加载首页的时候就加载，项目越大，对首页加载的速度提升越明显

2. 使用 CDN 加速

在做项目时，我们会用到很多库，在爱用 CDN 加载可以加快加载速度

3. gzip 压缩

方法一：使用 Nginx 反向代理，配置 nginx.conf 文件

方法二：使用 node 压缩，需要使用 compression 库

4. 异步加载组件

详细学习方法地址：<https://segmentfault.com/a/1190000012138052>

5. 服务端渲染

使用 pug/jade/ejs/vue 通过应用框架 Nuxt 等等都可以实现后端渲染，并且后端渲染还能对 seo 优化起到作用

生命周期相关面试题

生命总共分为 8 个阶段创建前/后、载入前/后、更新前/后、销毁前/后

创建前/后：

在 beforeCreate 阶段，vue 实例的挂载元素 el 和数据对象 data 都为 undefined，还未初始化。

在 created 阶段，vue 实例的数据 data 有了，el 还没有

载入前/后：

在 beforeMount 阶段，vue 实例的 \$el 和 data 都初始化了,单还没有挂载之前都是虚拟的 demo 阶段,data.message 还未替换。

在 mounted 阶段,vue 实例挂载完后,data.message 成功渲染。

更新前/后：当 data 变化时,户触发 beforeUpdate 和 update 方法。

销毁前/后：

在执 destroy 方法后，对 data 的改变不会再触发周期函数，说明此时 vue 实例已经结束了事件监听以及和 dom 的绑定,但是 dom 结构依然存在。

什么是 vue 生命周期？

vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、挂载 DOM→渲染、更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。

vue 生命周期的作用是什么？

生命周期中有多个事件钩子,让我们在控制整个 Vue 实例的过程中更容易形成好的逻辑

vue 生命周期总共有几个阶段？

总共可以分 8 个阶段：创建前/后、载入前/后、更新前/后、销毁前/后

第一次页面加载会触发哪几个钩子？

第一次页面加载时会触发 beforeCreate、created、beforeMount、mounted 这几个钩子

请列举出 3 个 Vue 常用的声明周期钩子函数

created：实例已经创建完成之后调用，在这一步，实例已经完成数据观测、属性和方法的运算，watch、event 事件回调，然而，挂载阶段还没有开始，\$el 属性目前还不可见

mounted：el 被新创建的 vm.\$el 替换，并挂载到实例上去之后调用该钩子，如果 root 实例挂在了一个文档内元素，当 mounted 被调用时 vm.\$el 也在文档内。

activated：keep-alive 组件激活时调用

DOM 渲染在那个周期中已完成？

DOM 渲染在 mounted 中就已经完成了

Vue-router

Vue-router 的跳转原理：

vue-router 实现单页面路由跳转，提供了三种方式：hash 方式、history 模式、abstract 模式，根据 mode 参数来决定采用哪一种方式

- hash: 使用 URL hash 值来作路由。默认模式。
- history: 依赖 HTML5 History API 和服务器配置。查看 HTML5 History 模式。

- abstract: 支持所有 JavaScript 运行环境，如 Node.js 服务器端

路由之间的跳转:

声明式（标签跳转）：<router-view>标签用于展示路由组件，DOM 节点中使用 v-link 进行跳转，或使用 router-link 标签

编程式（js 跳转）

怎么定义 vue-router 的动态路由以及如何获取传过来的动态参数？

在 router 目录下的 index.js 文件中，对 path 属性加上/: id

使用 router 对象的 params id

Vue 中,如何用 watch 去监听 router 变化

当路由发生变化的时候，在 watch 中写具体的业务逻辑

```
let vm = new Vue({  
  
  el:"#app",  
  
  data:{},  
  
  router,  
  
  watch:{  
  
    $router(to,from){  
  
      console.log(to.path);  
  
    }  
  
  }  
  
})
```

vue-router 有哪几种导航钩子？以及它的参数？

三种，一是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。

第二种：组件内的钩子

第三种：单独路由独享组件

beforeRouteEnter、afterEnter、beforeRouterUpdate、beforeRouteLeave

参数：有 to（去的那个路由）、**from**（离开的路由）、**next**（一定要用这个函数才能去到下一个路由，如果不用就拦截）最常用就这几种

Vue 的路由实现: hash 模式和 history 模式(Vue 的两种状态)

1. **hash**——即地址栏 URL 的#符号,特点：通过 window.onhashchange 的监听，匹配不同的 url 路径，进行解析，加载不同的组件，然后动态的渲染出区域内的 html 内容，不会被包含在 HTTP 请求中,对后端完全没有影响

HashHistory 有两个方法：

HashHistory.push（）是将路由添加到浏览器访问历史的栈顶

hashHistory.replace（）是替换掉当前栈顶的路由

因为 hash 发生变化的 url 都会被浏览器历史访问栈记录下来,这样一来，尽管浏览器没有请求服务器，但是页面状态和 url 一一关联起来的，浏览器还是可以进行前进后退的

2. **history** —— 利用了 HTML5 History Interface 中新增的 pushState

() 和 `replaceState` () 方法。这两个方式应用于浏览器的历史记录栈，提供了对历史记录의 修改功能。history 模式不怕页面的前进和后退，就怕刷新，当刷新时，如果服务器没有相应的响应或者资源，就会刷出 404，而 hash 模式不会

\$router 和 \$route 的区别

1. \$route 从当前 router 跳转对象里面可以获取 name、path、query、params 等 (<router-link> 传的 参数 有 this.\$route.query 或者 this.\$route.params 接收)
2. \$router 为 VueRouter 实例。想要导航到不同 URL，则使用 \$router.push 方式，返回上一个 history 也是使用 \$router.go 方法

Vuex:

Vuex 是一个专门为 vue.js 应用设计的状态管理架构，统一管理和维护各个 Vue 组件的可变化状态 (可以理解为 Vue 组件中的那些 data)

Vuex 的五大属性:

Vuex 有五个核心概念: state、getters、mutations、actions、modules

state => 基本数据

getters => 从基本数据派生的数据

mutations => 提交更改数据的方法，同步!

actions => 像一个装饰器，包裹 mutations，使之可以异步

modules => 模块化 Vuex

不用 Vuex 会带来什么问题？

- 1) 可维性会下降，想修改数据要维护三个地方
- 2) 可读性会下降，因为一个组件里的数据，根本就看不出来是从哪来的；
- 3) 增加耦合，大量的上传派发，会让耦合性大大增加。而 Vue 用 Component 就是为了减少耦合

React

react 的优势以及特点

优势:

1. 实现对虚拟 DOM 的操作，速度快，提高了 web 性能
2. 组件化、模块化。react 里每一个模块都是一个组件，组件化开发，可维护性高
3. 单向数据流，比较有序，有便于管理，它随着 react 视图库的开发而被 Facebook 概念化
4. 跨浏览器兼容：虚拟 DOM 帮助我们解决了跨浏览器问题，它为我们提供了标准化 API，甚至在 IE8 中都是没有问题的

不足:

1. react 中只是 MVC 模块的 View 部分，要依赖以内很多其它模块开发
2. 当父组件进行重新渲染操作时，即使子组件的 props 或 state 没有做出任何改变，也会同样进行重新渲染

特点:

1. 声明式设计: React 采用声明范式, 可以轻松描述应用
2. 高效: react 通过对 DOM 的模拟, 最大限度的减少与 DOM 的交互
3. 灵活: react 可以与已知的库或框架很好的配合

React 中的 props 和 state 的用法

state 是一种数据结构, 用于组件挂载时所需数据的默认值。state 可能会随着时间的推移而发生突变, 但多数时候是作为用户事件行为的结果

Props 则是组件的配置。props 由父组件传递给子组件, 并且就子组件而言, props 是不可变的

react 组件之间如何通信?

父子:

父传子: props

子传父: 子调用父组件中的函数并传参

兄弟: 利用 redux 实现

所有关系都通用的方法, 利用 PubSub.js 订阅

为什么虚拟 DOM 会提高性能?

虚拟 dom 相当于在 js 和真实 dom 中间加了一个缓存, 利用 dom diff 算法避免了没有必要的 dom 操作, 从而提高性能。

具体实现步骤如下:

用 js 对象结构表示 DOM 数的结构，然后这个树构造一个真正的 DOM 树，插到文档中

当状态变更的时候，重新构造一颗树的对象树。然后用新的树和旧的树进行比较，记录两棵树的差异。

把 2 所记录的差异应用到步骤 1 所构建的真正的 DOM 数上，视图就更新了。

react 生命周期函数：

react 声明周期分为了挂载、更新、卸载三个阶段

挂载阶段：

Mounting Component
constructor()
static getDerivedStateFromProps(nextProps, prevState)
render()
componentDidMount()

constructor ()

当页面加载时，最初也是唯一一次被调用

用于做一些初始化状态的操作

唯一可以修改 state 的地方（因为 react 一般想要修改 state，一般会调用 setState 方法）

static getDerivedStateFromProps (nextProps, prevState)

当 state 需要从 props 初始化时使用

每次 render 或者 rerender 的时候都会调用

尽量不要使用：维护两者一致性会增加复杂度

典型场景：表单控件获取默认值

render()

这是类组件中唯一必需的方法

每次 react 更新并提交到 DOM 的时候调用

用于为组件编写 JSX

componentDidMount()

在构建组件并将其添加到 DOM 上(渲染后)只执行一次

可用于获取数据并在渲染后立即显示

典型场景：获取外部资源

更新时:

Updating State and Props
static getDerivedStateFromProps(nextProps, prevState)
shouldComponentUpdate(nextProps, nextState)
render()
getSnapshotBeforeUpdate(prevProps, prevState)
componentDidUpdate(previousProps, previousState, snapshot)

static getDerivedStateFromProps(nextProps,prevState)

每次页面 render 之前调用, state 已经更新

典型场景：获取 render 之前的 DOM 状态

很少使用：将 props 复制到 state

shouldComponentUpdate(nextProps, nextState)

返回一个布尔类型的值，默认返回 true

在渲染(render)之前或组件接受到新的 state 和 props 的时候马上执行

在每次重新渲染之前调用，但不是初始化操作时调用

决定 Virtual Dom 是否要重绘

对于性能优化，可以做一些不必要重新渲染的操作

render()

对于一个类组件，这是唯一必需的方法

对于更新，如果 shouldComponentUpdate() 返回 false，则不会调用 render()

getSnapshotBeforeUpdate(prevProps, prevState)

此方法允许我们捕获在呈现该组件之前未存储在改状态中的某些属性。(即，如果用户滚到页面上的特定位置，并且我们想要存储该位置并在以后使用它)

在 react 更新并提交新的内容到 DOM 之前调用

很少使用但可以捕获可能快速变化的数据

卸载时:

Unmounting Component

componentWillUnmount()

componentWillUnmount

在从 DOM 卸载组件之前调用它，做一些回收类的操作，如清除定时器等

react 性能优化是哪个周期函数？

`shouldComponentUpdate` 这个方法用来判断是否需要调用 `render` 方法重新描绘 `dom`。因为 `dom` 的描绘非常消耗性能，如果我们能在 `shouldComponentUpdate` 方法中能够写出更优化的 `dom diff` 算法，可以极大的提高性能。

详情参考：<https://segmentfault.com/a/1190000006254212>

在生命周期中的哪一步你应该发起 AJAX 请求？

我们应当将 AJAX 请求放到 `componentDidMount` 函数中执行，主要原因有下：

React 下一代调用算法 `Fiber` 会通过开始或停止渲染的方式优化应用性能，其会影响到 `componentWillMount` 的触发次数。对于 `componentWillMount` 这个生命周期函数的调用次数会不确定，React 可能会多次频繁调用 `componentWillMount`。如果我们将 AJAX 请求放到 `componentWillMount` 函数中，那么显而易见其会被触发多次，自然也就不是最好的选择

如果我们将 AJAX 请求放置在生命周期的其他函数中，我们并不能保证请求仅在组件挂载完毕后会要求响应。如果我们的数据请求在组件挂载之前就完成，并且调用了 `setState` 函数将数据添加到组件状态中，对于为挂载的组件则会报错。而在 `componentDidMount` 函数中进行 AJAX 请求则

能有效避免这个问题。

概述一下 REact 中的事件处理逻辑

为了解决跨浏览器兼容性问题，React 会将浏览器原生事件封装为合成事件，传入设置的时间处理中。这里的合成事件提供了与原生事件相同的接口，不过它们屏蔽了底层浏览器的细节差异，保证了行为的一致性。另外有意思的是，react 并没有直接将事件附着到子元素上，而是以单一事件监听器的方式将所有的时间发送到顶层进行处理。这样 React 在更新 DOM 的时候就不需要考虑如何处理附着在 DOM 上的事件监听器，最终达到优化性能的目的

如何告诉 React 它应该编译生产环境版本？

通常情况下我们会使用 Webpack 的 DefinePlugin 方法将 NODE_ENV 变量这是为 production。编译版本中 React 会忽略 propTypes 验证以及其他的警告信息，同时还降低了代码库的大小，React 使用了 Uglify 插件来移除生产环境下不必要的注释等信息

调用 setState 之后发生了什么？

在调用 setState 函数之后，React 会将传入的参数对象与组件当前的状态合并，然后触发所谓的调和过程（Reconciliation）。经过调和过程，React 会以相对高效的方法根据新的状态构建 React 元素树并且着手重新渲染整个 UI 界面。React 得到元素树之后，React 会自动计算出新的树与老树的节点差异，然后根据差异对界面进行最小化重渲染。在差异计算算法中，React 能够相对精确

的知道哪些位置发生了改变以及应该如何改变，这就保证了按需更新，而不是全部重新渲染。

react 的 setState 的原理及用法

原理：当调用 setState 时，它并不会立即改变，而是会把要修改的状态放入一个任务队列，等到事件循环结束时，再合并更新，因此 setState 有异步，合并更新两个特性。

setState 为什么是异步的？

1. 保证内部的一致性

因为 props 要等到父组件渲染过后才能拿到，也就是不能同步更新的，state 处于统一性设置成了异步更新

2. 性能优化

举个栗子：假如你正在一个聊天窗口输入，如果来了一条新消息又要 render，那就会阻塞你的当前操作，导致延迟

3. 支持 state 在幕后渲染

异步可以使 state 在幕后更新，而不影响你当前旧页面的交互，提升用户体验。

另外，setState 在原生事件，setTimeout,setInterval,promise 等异步操作中，state 会同步更新

传入 setState 函数的第二个参数的作用是什么？

该函数会在 `setState` 函数调用完成并且组件开始重新渲染的时候被调用，我们可以用该函数来监听渲染是否完成：

```
this.setState(  
  {  
    username:"tylermcginnis33",  
    () => .console.log('setState has finished and the component has  
re-rendered');  
  }  
)
```

shouldComponentUpdate 的作用是啥以及为何它这么重要？

`shouldComponentUpdate` 允许我们手动地判断是否进行组件更新，根据组件的应用场景设置函数的合理返回值能够帮我们避免不必要的更新。

createElement 与 cloneElement 的区别是什么？

`createElement` 函数是 JSX 编译之后使用的创建 React Element 的函数，而 `cloneElement` 则是用于复制某个元素并传入新的 Props。

为什么我们需要使用 React 提供的 Children API 而不是 JS 的 map？

这个是 react 最新版的 API，主要是为了使 React 能在更多的不同环境下更快、更容易构建。于是把 react 分成了 react 和 react-dom 两个部分。这样就为 web 版的 react 和移动端的 React Native 共享组件铺平了道路,也就是说我们可以跨平台使用想用的 react 组件.

React 中的 Element 与 Component 的区别是？

React Element 是描述屏幕上所见内容的数据结构，是对于 UI 的对象表述，典型的 React Element 就是利用 JSX 构建的声明式代码片然后被转化为 createElement 的调用组合。而 React Component 则是可以接收参数输入并且返回某个 React Element 的函数或者类。

新的 react 包含了：`React.createElement`、`.createClass`、`.Component`、`.propTypes`、`.children` 以及其他元素和组件类。这些都是你需要构建组件时的助手。

而 react-dom 包含了 `ReactDOM.render`、`.unmountComponentAtNode` 和

`.findDOMNode` 在 `react-dom/server`，有 `ReactDOMServer.renderToString` 和 `.renderToStaticMarkup` 服务器端渲染支持。

在什么情况下你会优先选择使用 class Component 而不是 functional Component？

在组件主要包含内部状态或者使用到生命周期函数的时候使用 class

Component，否则使用函数式组件，否则使用函数式组件。

React 中 refs 的作用是什么？

refs 是 React 提供给我们安全访问 DOM 元素或者某个组件实例的句柄。

我们可以为元素添加 ref 属性然后在回调函数中接受该元素在 DOM 树中的句柄，该值会作为回调函数的第一个参数返回：

```
class CustomForm extends Component{

  handleSubmit = () => {

    console.log("Input Value",this.input.value)

  }

  render(){

    return(

      <form onSubmit = {this.handleSubmit}>

        <input type="text" ref={(input) => this.input = input}/>

        <button type="submit">Submit</button>

      </form>

    )

  }

}
```

上述代码中的 input 域包含了一个 ref 属性，该属性声明的回调函数会接收 input 对应的 DOM 元素，我们将其绑定到 this 指针以便在其他的类函数中使用。另外值得一提的是，refs 并不是类组件的专属，函数式组件同样能够利用闭包暂存其

值：

```
function CustomForm({handleSubmit}){

  let inputElement

  return (

    <form onSubmit={()=>handleSubmit(inputElement.value)}>

      <input type='text' ref={(input)=>inputElement=input}/>

      <button type='submit'>Submit</button>

    </form>

  )

}
```

React 中 keys 的作用是什么？

Keys 是 React 用于追踪哪些列表中元素被修改、被添加或者被移除的辅助标识。

```
render(){

  return(

    <ul>

      {this.state.todoItems.map(({task,uid})=>{

        return <li key={uid}>{task}</li>

      })}

    </ul>

  )

}
```

```
}
```

在开发过程中，我们需要保证某个元素的 key 在其统计元素中具有唯一性。在 React Diff 算法中 React 会借助元素的 Key 值来判断该元素是新近创建的还是被移动而来的元素，从而减少不必要的元素重渲染。此外 React 还需要借助 key 值来判断元素与本地状态的关联关系，因此我们绝不可忽视转换函数中 key 的重要性。

diff 算法？

把树结构按照层级分解，值比较同级元素。

给列表结构的每个单元添加唯一的 key 属性，方便比较

React 只会匹配项通 class 的 component（这里面的 class 指的是组件的名称）

合并操作，调用 component 的 setState 方法的时候，React 将其标记为 dirty 到每一个事件循环结束，React 检查所有标记 dirt component 重新绘制

选择性子树渲染，开发人员可以重写 shouldComponentUpdate 提高 diff 的性能

参考链接：<https://segmentfault.com/a/1190000000606216>

React 性能优化方案？

1. 重写 shouldComponentUpdate 来避免不必要的 dom 操作
2. 使用 production 版本的 react.js
3. 使用 key 来帮助 React 识别列表中所有子组件的最小变化

参考链接：<https://segmentfault.com/a/1190000006254212>

react 怎么从虚拟 dom 中拿出真实 dom

Refs 是 React 提供给我们的安全访问 DOM 元素或者某个组件实例的句柄。

我们可以为元素添 ref 属性,然后在回调函数中接受该元素在 DOM 树中的句柄,

该值会作为回调函数的第一个参数返回,或者 ref 可以传字符串。

例如: `<input ref=((input)=>{return this.name=input}) />`, `this.name.value` 取值

或者 `<input ref="name" />`, `this.refs.name` 取值

简述 flux 思想

flux 的最大特点,就是数据的“单向流动”

1. 用户访问 View
2. View 发出用户的 Action
3. Dispatcher 收到 Action, 要求 Store 进行相应的更新
4. Store 更新后, 发出一个“change”事件
5. View 收到“change”事件后, 更新页面

参考链接：<http://www.ruanyifeng.com/blog/2016/01/flux.html>

React 项目用过什么脚手架? Mern? Yeoman?

Mern: Mern 是脚手架的工具, 它可以很容易的使用 Mongo, Express, React and NodeJS 生成同构 JS 应用。它最大限度的减少安装时间, 并得到您使用的成熟技术来加速开发。

React 组件的划分-业务组件和技术组件？

根据组件的职责，通常把组件分为 UI 组件和容器组件

UI 组件负责 UI 的呈现，容器组件负责管理数据和逻辑

两者通过 React-redux 提供 connect 方法联系起来。

具体使用方法参考地址：

http://www.ruanyifeng.com/blog/2016/09/redux_tutorial_part_three_react-redux.html

react 高阶组件是什么？有什么作用？

高阶组件就是接受一个组件作为参数，在函数中对组件做一系列的处理，随后返回一个新的组件作为返回值。

作用：1、代码复用，逻辑抽象，抽离底层准备（bootstrap）代码

2、渲染劫持

3、State 抽象和更改

4、Props 更改

PureComponent（纯组件）的重要性的使用场景

PureComponent 改变了生命周期方法 shouldComponentUpdate，并且它会自动检查组件是否需要重新渲染。这时，只有 PureComponent 检测到 state 或者 props 发生变化时，PureComponent 才会调用 render 方法，PureComponent 仅仅是浅比较(shallow comparison)，所以改变组件内部的 props 或者 state，它将不会发挥作用。

使用 PureComponent 的最佳情况就是展示组件，它既没有子组件，也没有依赖应用的全局状态。

react hook:

Hook 是 React16.8 的新增特性，可以在不编写 class 的情况下使用 state 以及其他的 React 特性

react hook 最重要的三大基础:

useEffect //万能的生命周期，每次都触发的声明周期，可以重复写多个的生命周期

useState //创建值、创建改变值的方法

useContent //react 自带的 redux（mobx），方便组件之间传值

React 目前提供的 Hook

hook	用途
useState	设置和改变state，代替原来的state和setState
useEffect	代替原来的生命周期，componentDidMount，componentDidUpdate 和 componentWillUnmount 的合并版
useLayoutEffect	与 useEffect 作用相同，但它会同步调用 effect
useMemo	控制组件更新条件，可根据状态变化控制方法执行,优化传值
useCallback	useMemo优化传值，usecallback优化传的方法，是否更新
useRef	跟以前的ref，一样，只是更简洁了
useContext	上下文爷孙及更深组件传值
useReducer	代替原来redux里的reducer,配合useContext一起使用
useDebugValue	在 React 开发者工具中显示自定义 hook 的标签，调试使用。
useImperativeHandle	可以让你在使用 ref 时自定义暴露给父组件的实例值。

详细学习地址: https://blog.csdn.net/weixin_43648947/article/details/102838142

React-router:

react-router 的原理:

react-router 就是控制不同的 url 渲染不同的组件。react-router 在 history 库的基础上，实现了 URL 与 UI 的同步。

原理：DOM 渲染完成之后，给 window 添加 onhashchange 事件监听页面 hash 的变化，并且在 state 属性中添加了 route 属性，代表当前页面的路由。

具体步骤:

当点击链接，页面 hash 改变时，触发绑定在 window 上的 onhashchange 事件；

在 onhashchange 事件中改变组件的 state 中的 route 属性，react 组件的 state 属性改变时，自动重新渲染页面；

页面随着 state 中的 route 属性改变，自动根据不同的 hash 给 Child 变量赋值不同的组件，进行渲染。

用过 React-router 吗？router 3.X 和 router 4.X 区别在哪？

router 3.X

- 路由集中在一处；
- 布局和页面的层叠由层叠的 <Route> 组件控制；
- 布局和页面组件是路由的一部分；

React Router 4

- 不再提倡中心化路由,路由不集中在一起。取之的是路由存在于布局和 UI 之间。

- 不需要再在嵌套组件中使用 {props.children}
- location.query 属性没有了，现在通过 'query-string' 模块进行转换获取

```
import queryString from 'query-string'
```

```
let query=this.query=queryString.parse(location.search);
```

react-router 里 hashHistory 和 browserHistory 的区别

react-router 提供了三种方式来实现路由，并没有默认的路由，需要在声明路由的时候，显式指定所使用的路由。

browserHistory（官方推荐）、hashHistory、createMemoryHistory

区别：使用 hashHistory,浏览器的 url 是这样的：/#/user/liuna?_k=adseis

使用 browserHistory,浏览器的 url 是这样的：/user/liuna

这样看起来当然是 browserHistory 更好一些,但是它需要 server 端支持。

使用 hashHistory 时，因为有 # 的存在，浏览器不会发送 request,react-router 自己根据 url 去 render 相应的模块。

使用 browserHistory 时，从 / 到 /user/liuna, 浏览器会向 server 发送 request, 所以 server 要做特殊请求，比如用的 express 的话，你需要 handle 所有的路由 app.get('*', (req, res) => { ... }), 使用了 nginx 的话，nginx 也要做相应的配置。

如果只是静态页面，就不需要用 browserHistory,直接 hashHistory 就好了。

redux:

redux 的实现原理

redux 作为一个通用模块，主要还是用来处理应用中 state 的变更，通过 react-redux 做连接，可以在 react+redux 的项目中将两者结合的更好

react-redux 是一个轻量级的封装库，它主要通过两个核心方法实现：

Provider: 从最外部封装了整个应用，并向 connect 模块传递 store

connect:

1. 包装原组件，将 state 和 action 通过 props 的方式传入到原组件内部。
2. 监听 store tree 的变化，使其包装的原组件可以相应 state 变化

redux 中间件:

中间件提供第三方插件的模式，自定义拦截 action->reducer 的过程.变为 action->middlewares->reducer。这种机制让我们改变数据流，实现如异步 action，action 过滤，日志输出，异常报告等功能

redux 常见的中间件:

redux-logger: 提供日志输出

redux-thunk: 处理异步操作

redux-promise: 处理异步操作，sctionCreator 的返回值是 promise

redux 有什么缺点?

1. 一个组件所需要的数据，必须由父组件传过来，而不能像 flux 中直接从 store 取
2. 当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件

还是会重新 render，可能会有效率影响，或者需要写复杂的 shouldComponentUpdate 进行判断。

redux 三大原则：

- 1、单一数据源：整个应用的 **state** 被储存在一棵 object tree 中，并且这个 object tree 只存在于唯一一个 **store** 中。
- 2、State 是只读的：唯一改变 state 的方法就是触发 **action**，action 是一个用于描述已发生事件的普通对象。这样确保了视图和网络请求都不能直接修改 state，相反它们只能表达想要修改的意图。
- 3、使用纯函数来执行修改：为了描述 action 如何改变 state tree，你需要编写 reducers。Reducer 只是一些纯函数，它接收先前的 state 和 action，并返回新的 state。

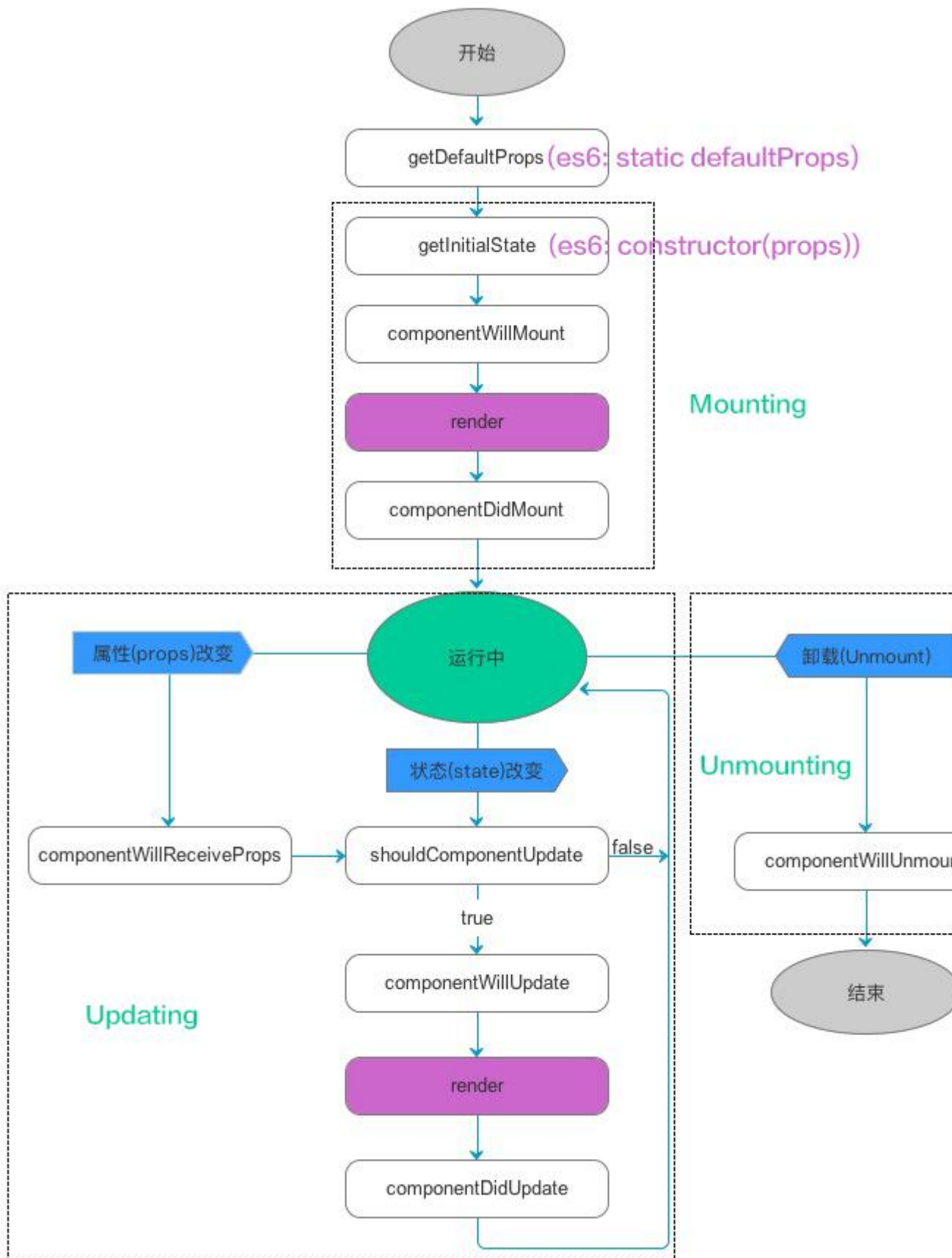
React Native：

React Native 相对于原生的 IOS 和 Android 有哪些优势？

1. 性能媲美原生 APP
2. 使用 JavaScript 编码，只要学习这一种语言
3. 绝大部分代码安卓与 IOS 都能共用
4. 组件化开发，代码重用性高
5. 跟编写网页一般，修改代码后即可自动刷新，不需要慢慢编译，节省了很多编译等待时间
6. 支持 APP 热更新，更新无需重新安装 App

缺点：内存占用相对较高，版本不稳定，一直在更新。

React Native 组件的生命周期：



1. 当页面第一次加载时，会依次调用：

constructor ()

componentWillMount ()：这个函数调用时机是在组件创建，并初始化了状态之后，在第一次绘制 render () 之前，可以在这里做一些业务初始化操作，也可以设置组件状态。这个函数在整个生命周期中只被调用一次。

render ()：组件渲染

componentDidMount ()：虚拟 DOM 已经构建完成，你可以在这个函数开始获取其中的元素或者子组件了。需要注意的是，RN 框架是先调用了子组件的 componentDidMount ()，然后调用父组件的函数。从这个函数开始，就尅和 JS 其他框架交互了，例如设置计时器 setTimeout 或者 setInterval,或者发起网络请求.这个函数也是只被调用一次.这个函数之后,就进入了稳定运行状态,等待事件触发.

2. 页面状态 state 更改时：

shouldComponentUpdate()：当组件接收到新的属性和状态改变

componentWillUpdate()：如果组件状态或者属性改变，并且上面的 shouldComponentUpdate(...) 返回为 true，就会开始准备更新组件

render()：组件渲染

componentDidUpdate()：属性和状态完成更新

3. 属性(props) 改变时：

componentWillReceiveProps()：组件收到的新属性(props)

4. 组件关闭

componentWillUnmount()：组件要被从界面上移除的时候，在这个函数

中可以做一些组件相关的清理工作，例如取消计时器、网络请求等

注意：shouldComponentUpdate（）默认返回 true，true 表示需要更新，继续走后面的更新流程

当返回值为 false 时，则不更新，直接进入等待状态，componentWillUpdate（）、render（）、componentDidUpdate（）都不会执行

微信小程序：

简单描述一下微信小程序的相关文件类型？

微信小程序结构主要由四个文件类型，如下：wxml、wxss、js、json、app.json、app.js、app.wxss

1. wxml 是框架设计的一套标签语言，结合基础组件、结合基础组件、事件系统，可以构建出页面结构。内部主要是微信自己定义的一套组件。

2. wxss 是一套样式语言，用于描述 wxml 的组件样式
3. js 逻辑处理，网络请求
4. json 小程序设置，如页面注册，页面标题及 tabBar。
5. app.json

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

6. app.js

必须要有这个文件，没有的话也会报错，但这个文件创建一下就行，什么都不需要写，以后我们可以在这个文件中监听并处理小程序的生命周期函数、声明全局变量。

7. app.wxss

可以用于设置通用样式

你是怎么封装微信小程序的数据请求？

1. 将所有的接口放在同一的 js 文件中并导出
2. 在 app.js 中创建封装请求数据的方法
3. 在子页面中调用封装的方法请求数据

有哪些参数传值的方法？

1. 给 HTML 元素添加 data-* 属性来传递我们需要的值，然后通过 e.currentTarget.dataset 或 onload 的 param 参数获取。但 data-名称不能有

大写字母和不可以存放对象

2. 设置 id 的方法标识来传递通过 e.currentTarget.id 获取设置的 id 值，然后通过设置全局对象的方式来传递数值

3. 在 navigator 中添加参数传值

你使用过哪些方法，来提高微信小程序的应用速度？

1. 提高页面加载速度
2. 用户行为预测
3. 减少默认 data 的大小
4. 组件化方案（wepy）
5. 让后台减少联合查询数据库 加强接口请求速度
6. 控制图片大小 部分图片可放在服务器（小程序大小限制）

小程序和原生 App 哪个好？

小程序除了拥有公众号的低开发成本、低获客成本低以及无需下载等优势，在服务请求延时与用户使用体验是都得到了较大幅度的提升，使得其能够承载跟负责的服务功能以及使用户获得更好的用户体验。

简述微信小程序原理？

微信小程序采用 Javascript、wxml、wxss 三种技术进行开发，从技术讲和现有的前端开发差不多，但深入挖掘的话却又有所不同。

Javascript：首先 Javascript 的代码是运行在微信 App 中的，并不是运行

在浏览器中，因此一些 H5 技术的应用，需要微信 App 提供对应的 API 支持，而这限制了 H5 技术的应用，且不能称为严格的 H5。同理，微信提供的独有的某些 API，H5 也不支持或支持的不是特别好。

wxml：wxml 微信自己基于 xml 语法开发的，因此开发时，只能使用微信提供的现有标签，HTML 的标签是无法使用的。

wxss：wxss 具有 css 的大部分特性，但并不是所有的都支持，而且支持哪些，不支持哪些并没有详细的文档。

微信的框架，是数据驱动的架构模型，它的 UI 和数据是分离的，所有的页面更新，都需要通过对数据的更改来实现。

小程序分为两个部分 webview 和 appService。其中 webview 主要用来展现 UI，appService 有来处理业务逻辑、数据及接口调用。它们在两个进程中运行，通过系统层 JSBridge 实现通信，实现 UI 的渲染、事件的处理

分析微信小程序的优劣势

优势：

1. 无需下载，通过搜索和扫一扫就可以打开
2. 良好的用户体验，打开速度快
3. 开发成本要比 App 低
4. 安卓上可以添加到桌面，与原生 App 差不多
5. 为用户提供了良好的安全保障，小程序的发布，微信拥有一套严格的审查流程，不能通过审查的小程序是无法发布到线上的

劣势：

1. 限制比较多，页面大小不能超过 2M，不能打开超过 10 个层级的页面。
2. 样式单一，小程序的部分组件已经是成型的了，样式不可以修改。例如：幻灯片、导航
3. 推广面窄，不能分享朋友圈，只能通过分享给朋友，附近小程序推广。其中附近小程序也受到微信的限制。
4. 依托于微信，无法开发后台管理功能

微信小程序与 H5 的区别？

1. 运行环境 传统的 HTML5 的运行环境是浏览器，包括 webview，而微信小程序的运行环境并非完整的浏览器，是微信开发团队基于浏览器内核完全重构的一个内置解析器，针对小程序专门做了优化，配合自己定义的开发语言标准，提升了小程序的性能。
2. 开发成本不同 只在微信中运行，所以不用再去顾虑浏览器兼容性问题，不用担心生产环境中出现不可预料的奇妙 bug
3. 获取系统级权限不同 系统级权限都可以和微信小程序无缝衔接
4. 应用在生产环境的运行流畅度 长久以来，当 HTML5 应用面对复杂的业务逻辑或者丰富的页面交互时，它的体验总是不尽人意，需要不断的对项目优化来提升用户体验，但是由于微信小程序运行环境独立

怎么解决小程序的异步请求问题？

在回调函数中调用下一个组件的函数(app.js)

```
success: function (info) {  
  
    that.apirtnCallback(info)  
  
}
```

index.js

```
onLoad: function () {  
  
    app.apirtnCallback = res => {  
  
        console.log(res)  
  
    }  
  
}
```

小程序的双向绑定和 vue 哪里不一样

小程序直接 this.data 的属性是不可以同步到视图的, 必须调用
this.setDate({ noBind:true })

小程序的 wxss 和 css 有哪些不一样的地方?

1. wxss 的图片引入需要使用外链地址
2. 没有 Body; 样式可以值使用 import 导入

小程序关联微信公众号如何确定用户的唯一性?

使用 wx.getUserInfo 方法 withCredentials 为 true 时 可获取
encryptedData, 里面有 union_id。后端需要进行对称解密

使用 webview 直接加载要注意哪些事项?

1. 必须要在小程序后台使用管理员添加业务域名；
2. h5 页面跳转至小程序的脚本必须是 1.3.1 以上；
3. 微信分享只可以都是小程序的主名称了，如果要自定义分享的内容，需小程序版本在 1.7.1 以上；
4. h5 的支付不可以是微信公众号的 appid，必须是小程序的 appid，而且用户的 openid 也必须是用户和小程序的。

小程序调用后台接口遇到哪些问题？

1. 数据的大小有限制，超过范围会直接导致整个小程序崩溃，除非重启小程序；
2. 小程序不可以直接渲染文章内容页这类型的 html 文本内容，若需显示要借住插件，但插件渲染会导致页面加载变慢，所以最好在后台对文章内容的 html 进行过滤，后台直接处理批量替换 p 标签 div 标签为 view 标签，然后其它的标签让插件来做，减轻前端的时间。

小程序写自定义的组件，要考虑什么？

微信小程序的版本不能太低，否则整个界面会显示一片空白

小程序怎么获取用户授权信息？

在页面中加入一个 **button 按钮**，并将 **open-type** 属性设置为 **getUserInfo**。

小程序如何分享卡片信息？

```
onShareAppMessage: function () {  
  
  return {  
  
    title: '自定义分享标题',  
  
    desc: '自定义分享描述',  
  
    path: '/page/user?id=123'  
  
  }  
  
}
```

小程序框架 wepy 的特性和特点？

- 1 类 Vue 开发风格
- 2 支持自定义组件开发
- 3 支持引入 NPM 包
- 4 支持 Promise 支持 ES2015+ 特性，如 Async Functions 支持多种编译器，5 5
Less/Sass/Styus、Babel/Typescript、Pug
- 6 支持多种插件处理，文件压缩，图片压缩，内容替换等
- 7 小程序细节优化，如请求列队，事件优化等 可同时进发 10 个 request 请求

Wepy 组件之间的通信与交互方式？

wepy.component 基类提供三个方法\$broadcast，\$emit，\$invoke

小程序分包是什么？有什么作用？

提示：跟 vue 路由懒加载有类似的作用，提升用户体验度，减少首屏渲染速

度

<https://developers.weixin.qq.com/miniprogram/dev/framework/subpackages/basic.html>
<https://blog.csdn.net/acmdown/article/details/80037660>

IOS/Andriod 浏览器适配问题整理:

相关知识点:

移动端、兼容/适配、IOS 点击事件 300ms 延迟、点击穿透、定位失败...

手机浏览器特有事件:

onTouchmove、onTouched、onTouchstart、onTouchcancel

使用 Zepto 的原因:

JQuery 适用于 PC 端桌面环境，桌面环境更加复杂，jQuery 需要考虑的因素非常多，尤其表现在兼容性上面。

与 PC 端相比，移动端的发展远不及 PC 端，手机上的宽度永远比不上 PC 端。

PC 端下载 JQuery 到本地只需要 1-3 秒 (90+K)，但是移动端就慢了很多，2G 网络下你会看到一大片空白网页在加载，这样用户可能就没打开的欲望了。

Zepto 解决了这个问题：只有不到 10K 的大小，2G 网络环境也毫无压力，表现不逊色于 JQuery。

所以移动端开发首选框架，个人推荐 zepto.js

渐进增强和优雅降级:

渐进增强: 针对低版本浏览器进行构建页面，保证最基本的功能，然后再针

对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。

优雅降级：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容

IOS 移动端 click 事件 300ms 的延迟响应

移动设备上的 web 网页是有 300ms 延迟的,这样往往会造成按钮点击延迟甚至是点击失败。此问题是由于区别单机事件和双击屏幕缩放的历史原因造成的。

原因：2007 年苹果发布首款 iphone 上 IOS 系统搭载的 safari 为了将适用于 PC 端上大屏幕的网页能比较好的展示在手机端上，使用了双击缩放的方案。比如手机上用浏览器打开一个 PC 上的网页，可能看到的页面内容虽然可以撑满整个屏幕，但是字体、图片都很小看不清，此时可以快速双击屏幕上的某一部分，你就能看清部分放大后的内容，再次双击后能回到原始状态。原因就在浏览器需要如何判断快速点击，当用户在屏幕上点击某一个元素的时候，浏览器并不能确定用户是单纯的点击还是双击该部分区域进行缩放操作。所以，捕获第一次单机后，浏览器会先 hold 一段时间，如果在这段时间内，用户未进行下一次点击，则浏览器会作为单击事件处理，如果在这段时间里用户进行了第二次单击事件，则会执行页面局部区域缩放操作。在 IOS safari 下，这个时间段大致为 300ms，这就是延迟的原因。

解决方案：

fastclick 可以解决在手机上点击事件的 300ms 延迟

zepto 的 touch 模块，tap 事件也是为了解决在 click 的延迟问题

触摸事件的响应顺序为 touchstart -> touchmove -> touchend ->

click，也可以通过绑定 ontouchstart 事件，加快事件的响应，解决 300ms 延迟问题

一些情况下，对非可点击元素（如 label，span）监听 click 事件，IOS 不会触发

css 增加 cursor: pointer;

三星手机遮罩层下的 input、select、a 等元素可以被点击和 focus（点击穿透）

问题发现于三星手机，这个在特定需求下才会有：

首先需求是浮层操作，在三星上被遮罩的元素依然可以获取 focus、click、change


解决方案：

第一种：通过层显示以后加入对应的 class 名控制，截断显示层下方可以获取焦点元素的事件获取

第二种：通过将可获取焦点元素加入的 disabled 属性，也可以利用属性加 dom 锁定的方式（disabled 的一种变换方式）

安卓浏览器看背景图时，有些设备会模糊

原因：因为手机分辨率太小，如果按照分辨率来显示网页，这样字会非常小，所以苹果当初就把 iPhone 4 的 960*640 分辨率，在网页里值显示了 480*320，这样 devicePixelRatio =2。现在安卓的比较多，有 1.5 的，也有 2 或 3 的。

解决方案：想让图片在手机里显示更为清晰，必须使用 2X 的背景图来代替 标签（一般情况都是用 2 倍）

例如：一个 div 的宽高是 100*100，背景图必须是 200*200，然后使用 background-size:contain; 这样显示出来的图片比较清晰

当输入框在最底部，点击软键盘后输入框内被遮挡

```
//浏览器当前高度
var oHeight = $(document).height();
$(window).resize(function(){
    if($(document).height() < oHeight ){
        $("#footer").css("position","static");
    }else{
        $("#footer").css("position","absolute");
    }
})
```

关于 Web 移动端 Fixed 布局的解决方案，这篇文章不错

<http://efe.baidu.com/blog/mobile-fixed-layout/>

消除 reansition 闪屏

/*设置内嵌的元素在 3D 空间如何呈现：保留 3D*/

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

/*(设置进行转换的元素的背面在面对用户时是否可见：隐藏)*/

-webkit-backface-visibility: hidden;

CSS3 动画页面闪白，动画卡顿

解决方案：

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

使用 position 的 left 和 top 来定位

2. 开启硬件加速

```
-webkit-transform: translate3d(0,0,0);  
-moz-transform: translate3d(0, 0, 0);  
-ms-transform: translate3d(0, 0, 0);  
transform: translate3d(0, 0, 0);
```

阻止旋转屏幕时自动调整字体大小

```
html, body, form, fieldset, p, div, h1, h2, h3, h4, h5, h6 {  
  -webkit-text-size-adjust: none;  
}
```

Input 的 placeholder 会出现文本位置偏上的情况

PC 端： 设置 line-height = height;

移动端： 设置 line-height: normal;

往返缓存问题：

点击浏览器的回退，有时候不会自动执行 js，特别是 mobile safari 中。这与往返缓存有关系，解决方案： window.onunload = function(){ };

calc 的兼容性处理

CSS3 中的 calc 变量

在 IOS6 浏览器中必须加-webkit-前缀，目前的 FF 浏览器已经无需-moz-前缀。

Android 浏览器目前仍然不支持 caic，所以要在之前添加一个保守尺寸：

```
div{  
  width: 95%;
```

```
width: -webkit-calc(100% -50px);  
width:calc(100% -50px);  
}
```

加上一个 CSS3 的属性后，让所关联的元素事件监听失效

pointer-events: none;

```
#box{  
  width: 100px;  
  height: 100px;  
  background: #000;  
  pointer-events: none;  
}  
<div id="box"></div>  
<script>  
  var box = document.getElementById("box");  
  box.onclick = function(){  
    console.log("123");  
  }  
</script>
```

防止手机中网页放大和缩小

```
<meta  
name=" viewport"  
content=" width=device-width,initial-scale=1.0,  
maximum-scale=1.0,user-scalable=0" >
```

上下拉动滚动条时卡顿，慢

```
body {  
  -webkit-overflow-scrolling:touch;  
  overflow-scrolling: touch;  
}
```

Android3+ 和 IOS5+支持 CSS3 的新属性：overflow-scrolling

关于图片加载

关于图片加载很慢的问题，在手机开发一般用 canvas 方法加载：

```
<li><canvas></canvas></li>
```

js 动态加载图片和 li 总共举例 17 张图片！

```
var total = 17;
```

```
var zWin = $(window);
```

```
var render = function(){
```

```
    var padding = 2;
```

```
    var winWidth = zWin.width();
```

```
    var picWidth = Math.floor( (winWidth-padding*3)/4 );
```

```
    var tpl = '';
```

```
    for(var i = 1; i <= totla; i++){
```

```
        var p = padding;
```

```
        var imgSrc = 'img/' + i + '.jpg';
```

```
        if( i%4 == 1){
```

```
            p=0;
```

```
        }
```

```
        tpl += '<li
```

```
            style="width:'+picWidth+'px;height:'+picWidth+'px;padding-left:'+p+'px;p
```

```
adding-top:'+padding+'px;"><canvas id="cvs_'+i+'"></canvas></li>';
```

```
        var imageObj = new Image();
```

```
        imageObj.index = i;
```

```
imageObj.onload = function(){  
    var cvs = $('#cvs_'+this.index)[0].getContext('2d');  
  
    cvs.width = this.width;  
  
    cvs.height = this.height;  
  
    cvs.drawImage(this,0,0);  
  
}  
  
imageObj.src = imgSrc;  
  
}  
  
}  
  
render();
```

关于 Zepto 点透

zepto 的 tap 是通过监听绑定在 document 上的 touch 事件来完成 tap 事件来模拟的，同时 tap 事件是冒泡到 document 上触发的。在点击完成时的 tap 事件（touchstart、touchend）需要冒泡到 document 上才会触发。

而在冒泡到 document 之前，用户收的接触屏幕（touchstart）和离开屏幕（touchend）是会触发 click 事件的，因为 click 事件有延迟触发（这就是为什么移动端不用 click 而用 tap 的原因），所以在执行完 tap 事件之后，弹出来的选择组件马上就隐藏了，此时 click 事件还在延迟的 300ms 之中。当 300ms 到来时，click 到的其实不是完成而是隐藏之后的下方的元素。如果正下方的元素绑定的有 click 事件此时便会触发，如果没有绑定 click 事件的话就当没有 click，但是正下方的是 input 输入框（或者 select 选择框或者单选复选框），

点击会默认聚焦而弹出输入键盘，也就出现了上面的点透现象

解决方案：

//(1)引入 fastclick.js，在页面中加入如下 js 代码

```
window.addEventListener( "load", function() {  
    FastClick.attach( document.body );  
}, false );
```

//(2)有 zepto 或者 jQuery 的 js 里面加上

```
$(function() {  
    FastClick.attach(document.body);  
});
```

//(3)当然 require 的话就这样：

```
var FastClick = require('fastclick');  
FastClick.attach(document.body, options);
```

//(4)用 touchend 代替 tap 事件并阻止掉 touchend 的默认行为
preventDefault()

```
$("#cbFinish").on("touchend", function (event) {  
    //很多处理比如隐藏什么的  
    event.preventDefault();  
});
```

//(5)延迟一定的时间(300ms+)来处理事件

```
$("#cbFinish").on("tap", function (event) {  
    setTimeout(function(){  
        //很多处理比如隐藏什么的  
    },320);  
});
```

html5 调用安卓或者 IOS 的拨号功能

H5 提供了自动调用拨号的标签:只要在 a 标签的 href 中添加 tel: 就可以实现

```
<a href="tel:15677776767">点击拨打 15677776767</a>
```

禁止文本选中

```
Element {  
  -webkit-user-select:none;  
  -moz-user-select:none;  
  -khtml-user-select:none;  
  user-select:none;  
}
```

在 IOS 和 Andriod 中，audio 元素和 video 元素无法自动播放

解决方案：触屏播放 `$('html').one('touchstart', function(){ audio.play();})`

fixed 定位缺陷

IOS 下，fixed 元素容易定位出错，软键盘弹出时，影响 fixed 元素的定位。

Android 下，fixed 表现要比 IOS 更好。软键盘弹出时不会影响到 fixed 元素定位

IOS4 下支持：position: fixed

解决方案：使用 iScroll 插件

在移动端修改难看的点击的高亮效果，IOS 和安卓下都有效：

```
*{  
  -webkit-tap-highlight-color:rgba(0,0,0,0);  
}
```

不过此方法在现在 Andriod 浏览器下，只能取掉橙色的背景色，点击产生的高亮边框并没有去掉。

不让安卓手机识别邮箱

```
<meta content="email=no" name="format-detection" />
```

Android 下取消语音按钮

```
input::-webkit-input-speech-button {display: none}
```

禁止 IOS 识别长串数字为电话

```
<meta content="telephone=no" name="format-detection" />
```

禁止 IOS 弹出各种操作窗口

```
-webkit-touch-callout:none
```

IOS 系统 中文输入法 输入英文时，字母之间可能会出现一个六分之一空格

使用正则去掉： `this.value = this.value.replace(/\u2006/g , "");`

IOS 下取消 input 在输入时默认的英文首字母大写

```
<input autocapitalize="off" autocorrect="off" />
```

IOS 下伪类: hover

除<a>标签之外的元素都无效，在 Android 下则有效。类似

```
div#topFloatBar: hover
```

```
#topFloatBar_menu{ display:block }
```

这样的导航显示在 IOS6 点击没有点击效果，只能通过增加点击侦听器给元素增减 class 来控制子元素

IOS 和 Android 下触摸元素是出现半透明灰色遮罩

设置 alpha 值为 0 即可去除半透明灰色遮罩

注：transparent 属性值为 Android 下无效

```
Element {  
  
    -webkit-tap-highlight-color : rgba( 255,255,255, 0);  
  
}
```

去掉 iPhone 和 iPad 下输入框默认内阴影

```
Element {  
  
    -webkit-appearance: none;  
  
}
```

active 兼容处理 即 伪类 : active 失效

方法一：body 添加 ontouchstart

```
<body ontouchstart= " " >
```

方法二：js 给 document 绑定 touchstart 或 touchend 事件

```
document.addEventListener( 'touchstart' ,function( ){ },false )
```


动画定义 3D 启动硬件加速

```
element {  
  
    -webkit-transform : translate3d ( 0,0,0 ) ;  
  
    transform : translate3d ( 0,0,0 ) ;  
  
}
```

注意：3D 变形会消耗更多的内存与功耗

Retina 屏的 1px 边框

```
ELement {  
  
    border-width : thin ;  
  
}
```

圆角 bug

某些 Android 手机圆角失效

解决方案：background-clip : padding-box;

顶部状态栏背景

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

说明：除非你先使用 apple-mobile-web-app-capable 指定全屏模式，否则这个 meta 标签不会起任何作用。

如果 content 设置为 default，则状态栏正常显示。

如果设置为 blank，则状态栏会有一个黑色的背景。

如果设置为 blank-translucent，则状态栏显示为黑色半透明。

如果设置为 default 或 blank，则页面显示在状态栏的下方，即状态栏占据上方部分，页面占据下方部分，二者没有遮挡对方或被遮挡。

如果设置为 blank-translucent，则页面会充满屏幕，其中页面顶部会被状态栏遮盖住（会覆盖页面 20px 高度，而 iphone4 和 itouch4 的 Retina 屏幕为 40px）。

默认值是 default。

设置缓存

```
<meta http-equiv=" Cache-Control" content=" no-cache" >
```

手机页面通常在第一次加载后会进行缓存，然后每次刷新会使用缓存而不是重新向服务器发送请求。如果不希望使用缓存可以设置 no-cache。

桌面图标

```
<link rel="apple-touch-icon"href="touch-icon-iphone.png"/>
```

```
<link
```

```
rel="apple-touch-icon"sizes="76x76"href="touch-icon-ipad.png"/>
```

```
<linkrel="apple-touch-icon"          sizes="120x120"          href="
```

```
touch-icon-iphone-
```

```
retina.png "/>
```

```
<link          rel="apple-touch-icon"          sizes="152x152"
```

```
href="touch-icon-ipad-retina.png
```

"/>

IOS 下针对不同设备定义不同的桌面图标。如果不定义则以当前屏幕截图作为图标。

上面的写法可能大家会觉得会有默认光泽，下面这种设置方法可以去掉光泽效果，还原设计图效果！

```
<link rel="apple-touch-icon-precomposed" href="touch-icon-iphone.png" />
```

图片尺寸可以设定为 57*57 或者 Retina 可以定为 114*114，ipad 尺寸为 72*72

启动画面

IOS 下页面启动加载时显示的画面图片，避免加载时的白屏，可以通过 media 来指定不同的尺寸

```
<link rel="apple-touch-startup-image" href="start.png"/>
<!-- iPhone -->
<link href="apple-touch-startup-image-320x460.png" media="(device-width: 320px)"
rel="apple-touch-startup-image"/>
<!-- iPhone Retina -->
<link href="apple-touch-startup-image-640x920.png" media="(device-width: 320px) and
(-webkit-device-pixel-ratio: 2)" rel="apple-touch-startup-image"/>
<!-- iPhone 5 -->
<link rel="apple-touch-startup-image" media="(device-width: 320px) and (device-height:
568px) and (-webkit-device-pixel-ratio: 2)"
href="apple-touch-startup-image-640x1096.png">
<!-- iPad portrait -->
<link href="apple-touch-startup-image-768x1004.png" media="(device-width: 768px) and
(orientation: portrait)" rel="apple-touch-startup-image"/>
<!-- iPad landscape -->
```

```
<link href="apple-touch-startup-image-748x1024.png" media="(device-width: 768px) and
(orientation: landscape)" rel="apple-touch-startup-image"/>
<!-- iPad Retina portrait-->
<link href="apple-touch-startup-image-1536x2008.png" media="(device-width: 1536px)
and (orientation: portrait) and (-webkit-device-pixel-ratio: 2)"
rel="apple-touch-startup-image"/>
<!-- iPad Retina landscape-->
<link href="apple-touch-startup-image-1496x2048.png" media="(device-width: 1536px)
and (orientation: landscape) and (-webkit-device-pixel-ratio:
2)"rel="apple-touch-startup-image"/>
```

浏览器私有及其它 meta

//QQ 浏览器私有

全屏模式

```
<meta name="x5-fullscreen" content="true">
```

强制竖屏

```
<meta name="x5-orientation" content="portrait">
```

强制横屏

```
<meta name="x5-orientation" content="landscape">
```

应用模式

```
<meta name="x5-page-mode" content="app">
```

//UC 浏览器私有

全屏模式

```
<meta name="full-screen" content="yes">
```

强制竖屏

```
<meta name="screen-orientation" content="portrait">
```

强制横屏

```
<meta name="screen-orientation" content="landscape">
```

应用模式

```
<meta name="browsermode" content="application">
```

//其它

针对手持设备优化，主要是针对一些老的不识别 viewport 的浏览器，比如黑莓

```
<meta name="HandheldFriendly" content="true">
```

微软的老式浏览器

```
<meta name="MobileOptimized" content="320">
```

windows phone 点击无高光

```
<meta name="msapplication-tap-highlight" content="no">
```

IOS 中 input 键盘事件 keyup、keydown、keypress 支持不是很好

用 input search 做模糊搜索的时候，在键盘里面输入关键词，会通过 ajax 后台查询，然后返回数据，然后再对返回的数据进行关键词标红。用 input 监听键盘 keyup 事件，在安卓手机浏览器中是可以的，但是在 ios 手机浏览器中变红很慢，用输入法输入之后，并未立刻相应 keyup 事件，只有在通过删除之后才能相应！

解决办法：

可以使用 html5 的 oninput 事件去代替 keyup

```
<input type="text" id="testInput">
```

```
<script type="text/javascript">
```

```
document.getElementById('testInput').addEventListener('input',function(e){
```

```
    var value = e.target.value;
```

```
});
```

```
</script>
```

然后达到类似 keyup 的效果！

H5 网站 input 设置为 type=number 的问题

H5 网页 input 的 type 设置为 number 一般会产生 3 个问题：

1. maxlength 属性不好用了

2. form 提交的时候，默认会取整。因为 form 提交默认做了表单验证，step 默认为 1

3. 部分安卓手机出现样式问题

问题 1 解决方案：

```
<input type="number" oninput="checkTextLength(this ,10)">
```

```
functioncheckTextLength(obj, length) {  
    if( obj.value.length > length) {  
        obj.value = obj.value.substr(0, length);  
    }  
}
```

问题 2 解决方案：

```
<input type="number" step="0.01" />
```

假如 step 和 min 一起使用，那么数值必须在 min 和 max 之间。

问题 3 解决方案：去除 input 默认样式

```
input[type=number] {  
    -moz-appearance:textfield;  
}  
  
input[type=number]::-webkit-inner-spin-button,  
input[type=number]::-webkit-outer-spin-button {  
    -webkit-appearance:none;  
    margin:0;  
}
```

IOS 设置 input 按钮样式按钮会被默认样式覆盖

解决方案：设置默认样式为 none

```
input,textarea {  
  
    border: 0;  
  
    -webkit-appearance : none ;  
  
}
```

select 下拉选择设置右对齐

```
select option {  
  
    direction: rtl ;  
  
}
```

通过 transform 进行 skew 变形, rotate 旋转 会产生 锯齿现象

```
-webkit-transform: rotate(-4deg) skew(10deg) translateZ(0);  
  
transform: rotate(-4deg) skew(10deg) translateZ(0);  
  
outline: 1px solid rgba(255,255,255,0);
```

关于 IOS 和 OSX 端字体的优化(横竖屏会出现字体加粗不一致的现象)

IOS 浏览器横屏时会重置字体大小，设置 text-size-adjust 为 none 可以解决 IOS 上的问题，但是桌面版 Safari 的字体缩放功能会失效，因此最佳方案是

将 text-size-adjust 为 100%

```
-webkit-text-size-adjust:100%;
```

```
-ms-text-size-adjust:100%;
```

```
text-size-adjust:100%;
```

移动端 H5 audio 的 autoplay 属性失效

这个不是 bug。由于自动播放网页中的音频或视频，会给用户带来一些困扰或者不必要的流量消耗，所以苹果系统和安卓系统通常都会禁止自动播放和使用 JS 的触发播放，必须由用户触发才可以播放

解决方案思路：先通过用户 touchstart 触碰，触发播放并暂停（音频开始加载，后面用 JS 再操作就没问题了）

```
document.addEventListener('touchstart',function() {  
    document.getElementsByTagName('audio')[0].play();  
    document.getElementsByTagName('audio')[0].pause();  
});
```

移动端 H5 的 input data 不支持 placeholder 的问题

```
<input placeholder="Date" class="textbox-n" type="text"  
onfocus="(this.type='date')" id="date">
```

type 为 search 的 input，部分机型会自带 close 按钮样式

有些机型的搜索 input 控件会自带 close 按钮（一个伪元素）。

而为了兼容所有浏览器，通常会自己实现一个，此时取掉原生 close 按钮的方法为：

```
Search::-webkit-search-cancel-button{  
  
    display:none;  
  
}
```

Pc 端兼容性问题：

IE 和 DOM 事件流的区别？

1. 执行顺序不一样
2. 参数不一样，低版本的 ie 没有回调函数，只能进行冒泡
3. 第一个参数是否加 ‘on’ ，低版本 IE 不支持 addEventListener () ，
之处 attachEvent，第一个参数需要加 “on”
4. this 指向问题，IE 指向 Windows，不指向触发的函数

IE 标准下有哪些兼容性写法

```
var ev = ev || window.event;  
  
document.documentElement.clientWidth ||  
  
document.body.clientWidth  
  
var target = ev.srcElement || ev.target
```

其他知识点面试题：

Node 的应用场景

特点：

它是一个 javascript 运行环境

依赖 chrome V8 引擎进行代码解析

事件驱动

非阻塞 I/O

单进程、单线程

优点：高并发（最重要的优点）

缺点：只支持单核 CPU，不能充分利用 CPU

可靠性地，一旦代码某个环节崩溃，整个系统都崩溃

谈谈你对 webpack 的看法

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

态资源

gulp 是什么？

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

Gulp 的核心概念：流

流：简单来说就是建立在面向对象基础上的一种抽象的处理数据的工具。在流中，定义了一些处理数据的基本操作，如读取数据，写入数据等，程序员是对流进行所有操作的，而不用关心流的另一头数据的真正流向

gulp 正是通过流和代码优于配置的策略来尽量简化任务编写的任务

Gulp 的特点：

易于使用：通过代码优于配置的策略，gulp 让简单的任务鸡蛋，复杂的任务可管理

构建快速：利用 nodeJs 流的威力，你可以快速构建项目并减少频繁的 IO 操作

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

常见的 web 安全及防护原理

sql 注入原理：

就是通过把 SQL 命令插入到 web 表单递交或输入域名或页面请求的查

询字符串，最终达到欺骗服务器执行恶意的 SQL 命令

总的来说有以下几点：

- 永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双引号以及-进行转换等
- 永远不要使用动态拼接 SQL，可以使用参数化的 SQL 或者直接使用存储过程进行数据查询存取
- 永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接
- 不要把机密信息明文存放，请加密或者 hash 掉密码和敏感的信息

XSS 原理及防范方法

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

防范方法：首先代码里对用户输入的地方和变量都需要仔细检查长度和对“<>,;. ”等字进行过滤；其次内容和内容写到页面之前都必须加 encode，避免不小心把 html tag 弄出来。这一个层面做好，至少可以堵住超过一半的 XSS 攻击

CSRF 的原理及防御

CSRF 是代替用户完成指定的动作，需要知道其他页面的代码和数据包。要完成一次 CSRF 攻击，受害者必须一次完成两个步骤

登录受信任网站 A，并在本地生成 cookie

在不登出 A 的情况下，访问危险网站 B

防御方法：服务端的 CSRF 方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数；通过验证码的方法

XSS 与 CSRF 两种跨站攻击

1. XSS 跨站脚本攻击，主要是前端层面，用户在输入层面插入攻击脚本，改变页面的显示，或则窃取网站 cookie，预防方法：不相信用户的所有操作，对用户输入进行一个转移，不运行 js 对 cookie 的读写
2. CSRF 跨站请求伪造，以你的名义，发送恶意请求，通过 cookie 加参数等形式过滤
3. 我们没法彻底杜绝攻击，只能提高攻击门槛

common.js AMD CMD 的区别

1. 这些规范的目的都是为了 js 的模块化开发，特别是在浏览器端的
2. 对于依赖的模块，AMD 是提前执行，CMD 是延迟执行
3. CMD 推崇依赖就近，AMD 推崇依赖前置

ES6 模块有 CommonJS 模块的差异

1. CommonJS 模块输出的是一个值的拷贝，ES6 模块输出的是一个值的

引用

2. CommonJS 模块时运行时加载，ES6 模块时编译输出接口

3. ES6 输入的模块变量，知识一个符号链接，所以这个变量是只读的，对它进行重新赋值就会报错

网页验证码是干嘛的，是为了解决什么安全问题

区分用户是计算机还是人的公共全自动程序。可以防止恶意破解密码、刷票、论坛灌水

有效防止黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登陆尝试

图片压缩上传(移动端):

为什么要使用图片压缩上传功能？

在做移动端图片上传的收，用户穿的都是手机本地图片，而本地图片一般都相对比较大，拿 iPhone6 来说，平时拍的图片都是 1-2M，如果直接上传，会占用一定的内存以及耗费大量的流量去长传文件，完整把图片上传显然不是一个很好的办法。目前来说 HTML5 的各种新 API 都在移动端的 webkit 是上得到了较好的实现。

图片压缩上传功能的实现步骤？

在移动端压缩图片并且上传主要用到的是 filereader */faɪl/ 'ri:də/* 、 canvas 以及 formdata 这三个 H5 的 API。

1) 用户使用 input file 上传图片的时候，用 filereader 读取用户上传的图片数据（base64 格式）

2) 把图片数据传入 img 对象，然后将 img 绘制到 canvas 上，再调用 canvas.toDataURL 对图片进行压缩

3) 获取到压缩后的 base64 格式图片数据，转成二进制塞入 formData，在通过 XMLHttpRequest 提交到 formData

仅有三步即可完成图片压缩上传功能，但是实现功能比较复杂，代码参考地址：<https://www.cnblogs.com/axes/p/4603984.html>

webpack 的原理

1. webpack 是把项目当作一个整体，通过给定的一个主文件，webpack 将从这个主文件开始找到你项目当中的所有依赖文件，使用 loaders 来处理它们，最后打包成一个或多个浏览器可识别的 js 文件

2. 通过设置 webpack.config.js 的配置，由于 webpack 是基于 Node 构建的，webpack 配置文件中所有的合法 node 语法都可以用

webpack 的 loader 和 plugin 的区别？

Loader: 用于模块源码的转换，loader 描述了 webpack 如何处理非 JS 模块，并且在 build 中引入这些依赖。loader 可以将文件从不同的语言转换为 JS，或者将内联图转换为 data URL。比如：CSS-Loader，Style-Loader 等

Plugin: 用于解决 loader 无法实现的很多事情，从打包优化和压缩，到重

新定义环境变量，功能强大到可以用来处理各种各样的任务。webpack 提供了很多开箱即用的插件，例如：CommonPlugin 主要用于提取第三方库和公共模块，避免首屏加载的 bundle 文件，或者按需加载的 bundle 文件体积过大，导致加载时间过长，是一把优化的利器。而在多页面应用中，更是能够为每个页面文件的应用程序共享代码创建的 bundle

怎么使用 webpack 对项目进行优化？

构建优化:

1. 减少编译体积 ContextReplacementPlugin 、 IgnorePlugin 、 babel-plugin-import、babel-plugin-transform-runtime。
2. 并行编译 happypack、thread-loader、uglifyWebpackPlugin 开启并行
3. 缓存 cache-loader 、 hard-source-webpack-plugin 、 uglifyjsWebpackPlugin 开启缓存、babel-loader 开启缓存
4. 预编译 dllWebpackPlugin&&DllReferencePlugin 、 auto-dll-webpack-plugin

性能优化:

1. 减少编译体积 Tree-shaking、Scope-Hositing
2. hash 缓存 webpack-md5-plugin
3. 拆包 splitChunksPlugin、import () 、 require.ensure

防抖、节流

防抖： 任务频繁被触发的情况下，只有任务触发的间隔超过指定时间的时候，任务才会执行

节流： 指定时间间隔内只会执行一次任务

详情可以浏览：

<https://www.cnblogs.com/Antwan-Dmy/p/10714445.html#chapter-three>

浏览器的缓存机制

浏览器缓存机制有两种，一种为强缓存，一种为协商缓存

强缓存： 浏览器在第一次请求的时候，会直接下载资源，然后缓存到本地，第二次请求的时候，直接使用缓存

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

强缓存方案：

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

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

协商缓存方案：

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

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

描述一下二叉树，并说明二叉树的几种遍历方式？

先序遍历: 若二叉树非空，访问根节点，遍历左子树，遍历右子树

中序遍历: 若二叉树非空，遍历左子树，访问根节点，遍历右子树

后序遍历: 若二叉树非空，遍历左子树，遍历右子树，访问根节点

项目类问题

如何规避 javascript 多人开发函数重名问题？

尽量少使用全局变量，尽可能的使用局部变量，这样不仅会减少变量重名的几率，更会减少内存开销，因为局部变量一般会在函数结束后自动销毁释放内存，而全局变量会到程序结束后才会被销毁

什么是渐进渲染:

渐进式渲染是用于改进网页性能的技术名称(特别是改善感知的加载时间), 以尽可能快的呈现内容以供显示。

这种技术的例子:

1. 懒加载图片
2. 优先考虑可见内容（或首屏渲染）---仅包括首先在用户浏览器

中呈现的页面数量所需的最小 CSS、内容、脚本，然后使用延迟脚本或

监听 DOMContentLoaded/load event 加载其他资源和内容

3. 异步 HTML 片段---在后端构建页面时将部分 HTML 刷新到浏览器

如何提高页面加载速度？

1. 减少 http 请求
2. 使用 CDN（内容发布网络），可以进行数据备份、扩展存储能力，进行缓存，同时有助于缓和 web 流量峰值压力
3. 压缩组件，代理缓存
4. 将样式表挡在文档头部
5. 将脚本放在底部
6. 避免 CSS 表达式
7. 使用外部的 JS 和 CSS 文件
8. 减少 DNS 查找

页面渲染优化:

1. 禁止使用 iframe(阻塞父文档 onload 事件)

iframe 会阻塞主页面的 onLoad 事件

搜索引擎的检索程序无法解读这种页面，不利于 SEO

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

使用 iframe 之前需要考虑这两个缺点。如果需要使用 iframe，最好是通过 javascript 动态给 iframe 添加 src 属性值。这样可以绕开以上两个问题

2. 禁止使用 gif 图片实现 loading 效果(降低 CPU 消耗，提升渲染性能)
3. 使用 CSS3 代码替代 JS 动画（尽可能避免重绘重排以及回流）
4. 对于一些小图标，可以使用 base64 位编码，以减少网络请求。但不建议大图使用，比较消耗 CPU

小图优势在于：

- 1) 减少 HTTP 请求
- 2) 避免文件跨域
- 3) 修改及时生效
5. css 与 js 文件尽量使用外部文件（因为 Renderer 进程中，JS 线程和渲染线程是互斥的）
6. 页面中空的 href 和 src 会阻塞页面其他资源的加载（阻塞下载进程）
7. 网页 Gzip、CDN 托管、data 缓存、图片服务器
8. 前端模板 JS+数据，减少 HTML 标签导致的宽带浪费，前端用变量把保存 AJAX 请求结果，每次操作本地变量，不用请求，减少请求次数
9. 用 innerHTML 代替 DOM 操作，减少 DOM 操作次数，优化 javascript 性能

前端需要注意哪些 SEO？（搜索引擎优化）：

1. 合理的 title、description、keywords：搜索对着三项的权重逐个减小，

title 值强调中你的那即可，重要关键词出现不要超过 2 次，而且要靠前，不同页面 title 要有所不同；description 把页面内容高度概括，长度合适，不可过分堆砌关键词，不同页面 description 有所不同；keywords 列举出重要关键词即可

2. 语义化的 HTML 代码，符合 W3C 规范：语义化代码让搜索引擎容易理解网页。

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

4. 重要内容不要用 js 输出：爬虫不会执行 JS 获取内容

5. 非装饰性图片必须加 alt

6. 少用 iframe：搜索引擎不会抓取 iframe 中的内容

7. 提高网站速度：网站速度是搜索引擎排序的一个重要指标

前后端分离的项目如何 seo？（偏难）

先去 www.baidu.com/robots.txt 找出常见的爬虫，然后在 nginx 上判断来访问页面用户的 User-Agent 是否是爬虫，如果是爬虫就用 nginx 方向代理到我们自己用 nodejs+puppeteer 实现的爬虫服务器上，然后用你的爬虫服务器怕自己的前后端分离的前端项目页面，增加扒页面的接受延时，保证异步渲染的接口数据返回，最后得到了页面的数据，返还给来访问的爬虫即可。

移动端常见的兼容性问题：

随着手机的普及，移动端的开发也成了一个重要方向，但是由于设备的不统一，会造成一些兼容性问题。

1. 设置文字行高为字体行高，解决文字上下边留白问题
2. 给动态元素添加事件，需要使用事件委托（绑定到 document），解绑也需要用委托的方式。苹果机点击事件不能触发。需要用 touch 系列事件
3. img 标签 src 属性无值（php 渲染过的），在苹果机上显示无图片，在安卓机上显示图片裂开。可添加 alt 属性及值
4. 同一个标签多次绑定同一个事件（页面复杂情况容易出现这种情况，尽量避免这种情况），可以减少 bug 的出现，利于维护页面
5. 在 rem 自适应页面使用精灵图。会容易出现图片缺角的问题（约 1-2 像素）。解决办法：使装精灵图的盒子变大，让图片居中显示
6. 给选中的盒子增加一个标识，可以使用伪元素，减少标签的使用
7. 有横向滚动条的内容被垂直触摸，在 IOS 机上无法滚动页面
8. 当祖父元素使用 overflow 属性时，父元素采用 transform 属性会影响子元素定位 position:absolute；导致子元素超出隐藏，建议用其他属性替换 transform 属性。
9. click 事件在 IOS 系统上有时会失效，给绑定 click 事件的元素加上 cursor: pointer 解决
10. placeholder 垂直居中问题：在 IOS 和 Android 中显示不同。解决方法是：在保证 input 输入文本垂直居中的条件下，给 placehoder 设置 padding-top

如何优化 SPA(单页面 Web 应用)应用的首屏加载速度慢的问题？

- 1) 将公用的 JS 库通过 script 标签外部引入，减少 app。bundel 的大小，让浏览器并行下载资源文件，提高下载速度
- 2) 在配置路由时，页面和组件使用懒加载的方式引入，进一步缩小 App。bundel 的体积，在调用某个组件时再加载对应的 js 文件
- 3) 加一个首屏 loading 图，提升用户体验

网页从输入网址到渲染完成经历了哪些过程？

- 1) 输入网址
- 2) 发送到 DNS 服务器，并获取域名对应的 web 服务器对应的 IP 地址
- 3) 与 web 服务器建立 TCP 连接
- 4) 浏览器向 web 服务器发送 http 请求
- 5) web 服务器响应请求，并返回一定 url 的数据（或错误信息，或重定向的新 url 地址）
- 6) 浏览器下载 web 服务器返回的数据及解析 html 源文件
- 7) 生成 DOM 树，解析 css 和 js，渲染页面，直至显示完成

CSS 和 JS 的位置会影响页面效率，为什么？

CSS 在加载过程中不会影响到 DOM 树的生成，但是会影响到 Render 树的生成，进而影响到 layout，所以一般来说，style 的 link 标签需要尽量放在 head 里面，因为在解析 DOM 树的时候是自上而下的，而 css 样式

又是通过异步加载的，这样的话，解析 DOM 树下的 body 节点和加载 css

样式能尽可能的并行，加快 Render 树的生成速度

JS 脚本应该放在底部，原因在于 JS 线程与 GUI 渲染线程是互斥关系，如

果 JS 放在首部，当下载执行 JS 的时候，会影响渲染行程绘制页面，JS

的作用主要是处理交互，而交互必须得先让页面呈现才能进行，所以为了

保证用户体验，尽量让页面先绘制出来

笔试编程题：

JS 数组的顺序排序,随机排序篇

顺序排序

1. 按字符编码排序: `sort()`

```
var array = [ 30,50,20,79,40,105,15 ];  
array.sort();
```

2. 将数组元素倒序排列

```
var array = [ 30,50,20,79,40,105,15 ];  
array.reverse();
```

3. 从小到大排序

```
var array = [ 30,50,20,79,40,105,15 ];  
arr.sort(function(a,b){  
    return a-b;  
})
```

随机排序

1. 低效版

```
//1. 低效版  
var array = [ 30,50,20,79,40,105,15 ];  
array.sort(function(){  
    return Math.random()>0.5?-1:1;  
})
```



```
})
```

2. 高效版

2.1 洗牌算法

```
var array = [ 30,50,20,79,40,105,15 ];
if(!Array.prototype.derangedArray){
  Array.prototype.derangedArray = function(){
    for(var j,x,i = this.length;i;j = parseInt(Math.random()*i),x =
this[--i],this[i]=this[j],this[j] = x);
  }
}
```

2.2 简单明了版

```
function shuffle(arr){
  var len = arr.length;
  for(var i = 0;i<len-1;i++){
    var index = parseInt(Math.random()*(len-i));
    var temp = arr[index];
    arr[index] = arr[len-i-1];
    arr[len-i-1] = temp;
  }
  return arr;
}
var array = [ 30,50,20,79,40,105,15 ];
shuffle(array);
```

编写一个九九乘法表

```
document.write("<table border='1'>");
for(var i = 1;i<10;i++){
  document.write("<tr>");
  for(var j = 1;j<10;j++){
    if(j <= i){
      document.write("<td>");
      document.write(`${i}*${j}=${i*j}`);
      document.write("</td>");
    }else{
      document.write("<td>");
      document.write("</td>");
    }
  }
}
```

```
document.write("</tr>");
}
document.write("</table>");
```

实现一个函数，判断输入是不是回文字符串。

```
function run(input){
  if(typeof input !== 'string') return false;
  return input.split("").reverse().join("") === input;
}
```

你对重绘、重排的理解？

首先网页数次渲染生成时，这个可称为重排；

修改 DOM、样式表、用户事件或行为（鼠标悬停、页面滚动、输入框键入文字、改变窗口大小等等）这些都会导致页面重新渲染，那么重新渲染，就需要重新生成布局 and 重新绘制节点，前者叫做"重排"，后者"重绘"；

减少或集中对页面的操作，即多次操作集中在一起执行；

总之可以简单总结为：重绘不一定会重排，但重排必然为会重绘。

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

```
<div id="box">
  
</div>
<script>
  const box = document.getElementById('box');
  function isIcon(target) {
    return target.className.includes('icon');
  }

  box.onclick = function (e) {
```

```
e.stopPropagation();
const target = e.target;
if (isIcon(target)) {
  target.style.border = '1px solid red';
}
}
const doc = document;
doc.onclick = function (e) {
  const children = box.children;
  for (let i = 0; i < children.length; i++) {
    if (isIcon(children[i])) {
      children[i].style.border = 'none';
    }
  }
}
</script>
```

简单实现双向数据绑定 mvvm

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

<script>
  const data = {};
  const input = document.getElementById("input");
  Object.defineProperty(data, 'text', {
    set(value) {
      input.value = value;
      this.value = value;
      document.getElementById("box").innerHTML = value;
    }
  });
  input.onkeyup = function(e) {
    data.text = e.target.value;
  }
</script>
```

为 string 扩展一个 trim 方法,取掉字符串中的所有空格

方法一：trim（）方法-----仅能取掉字符串首尾空格

```
var str = " a b c "

console.log("trim",str.trim());

//trim 原理

function Trim(str){

    return str.replace(/(^s*)|(s*$)/g, "");

}
```

方法二：去除字符串中所有的空格

```
str.replace(/\s/g,"")
```

JS 中如何检测一个变量是 string 类型?请写出函数实现

解释一下下面代码的输出

```
//1.
console.log(0.1+0.2); //0.30000000000000004

// 2.
console.log(0.1+0.2 == 0.3); //false

//3.
console.log('1'+2+'3'+4); //1234

//4.
var arr = [];
arr[0] = 'a';
arr[1] = 'b';
arr.foo = 'c';
console.log(arr.length); //2

//5.
function foo(){}
delete foo.length;
console.log(typeof foo.length); //number
```

说说以下代码运行会输出什么？

```
<script>
  function Foo(){
    getName = function(){alert(1)};
    return this;
  }
  Foo.getName = function(){alert(2)};
  Foo.prototype.getName = function(){ alert(3)};
  var getName = function() {alert(4)};
  function getName(){alert(5)};

  //请写出以下输出结果
  Foo.getName(); //2
  getName(); //4
  Foo().getName(); //1
  getName(); //1 ???
  new Foo.getName(); //2
  new Foo().getName(); //3
  new new Foo.getName(); //2
</script>
```

说说以下代码运行会输出什么？

```
<script>
  function test(a){
    var code = "600";
    setTimeout(()=>{
      this.code = a;
    });
  }
  var a = 700;
  b = {code:"900"};
  test.apply(b,[a]);
  alert(b.code+a); //900700
</script>
```

已知一对兔子每个月可以生一对小兔子，而一对兔子从出生后第

3 个月起每个月都会再生一对小兔子。假如一年内没有发生死亡现象，那么一对兔子一年内（12 个月）能繁殖多少对？

```
//1.定义三个变量，然后两个初始化值。第三个作为存储使用
var temp;
var num1 = 1;
var num2 = 1;
//2.先把之前的第二个变量存储在中间变量，然后把之后的第二个变量赋值为之前两个变量的和
for(var i=1;i<=10;i++){
    temp = num2;
    num2 = num1 + num2;
    //3.把存储的中间变量赋值给之后的第一个变量。
    num1 = temp;
    //4.执行 10 次。（因为前两项不需要计算，所以只需要执行 10 次）
}
console.log(num2);
```

`{ } == { }`、`[] == []` 的执行结果

`{ } == { }` //false

`[] == []` //false

利用 new String 和 String 传参的题

```
function showCase(val){
    console.log(val)
    switch(val){
        case 'A':
            console.log("Case A");
            break;
        case undefined:
            console.log('undefined');
            break;
        default:
            console.log("Do not Know!");
    }
}
```

//new String 返回的是一个对象

```
//String 返回的是一个字符串  
showCase(new String('A')); //Do not Know  
showCase(String('A')); //Case A
```

拓展: new String()、String ()、toString () 三者的区别

new String() 返回的是一个对象

String()与 toString() 返回的是字符串

区别: toString() 无法转换 null 和 undefined

['1' , '2' , '3'].map(parseInt)的答案以及为什么?

答案: [1 , NaN , NaN];

原因:

这道题的关键在于 map 函数中的 **parseInt**

map 函数每次处理数据的时候都传入的是三个参数，分别是当前值、下标、以及数组

```
var newArr = ['1','2','3'].map(function(currentVale,index, arr){  
  console.log(currentVale,index,arr);  
})
```

在 map 函数内部的 parseInt 是用来解析字符串的，有两个默认参数.parseInt(string , radix)，分别对应 map 每次循环的当前值和下标。

在 parseInt 中 radix 是解析字符串的进制数，，比如：二级制，十进制

parseInt (' 123456 ' , 10)--- 意思就是十进制解析字符串" 123456"

而我们的 radix 只介于 2-36 之间数组可以进行进制转换，在 radix 为 undefined 或者是 0 的情况下，要看字符串以什么开头。如果字符串以 '0x' 开头，用 16 进制；如果以 '0' 开头，用 8 或 10 进制；如果以其他值开头，则以十进制解析

在这道题中我们的 parseInt 分别进行了以下操作：

parseInt ('1' , 0) ; --- radix 为 0，用 8 或 10 进制解析 --> 1

parseInt ('2' , 1) ; --- radix 仅接受 2-36 之间的进制数，无法解析此字符串 ---> NaN

parseInt ('3' , 2) ; --- radix 这里用二进制解析此字符串，**但是**二进制是 0 和 1 拼接而成，3 不属于二进制的数据结构内容 ---> NaN

解决办法：使用 arr.map(Number)

前端技术交流群：104833704