HTML：

Doctype的作用?

> 文档声明，告诉浏览器以什么标准去渲染解析html

> **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；
>
> **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

注意：js引擎和渲染引擎是互斥的，也就是说当前js引擎在运行的时候，渲染引擎不会工作，反之亦然，这也是为什么建议把通过script标签引入的脚本放到body后面，而不是body前面的原因。

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,会在浏览器和服务器中来回传递,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 ['ɑrtɪkl]

本地离线存储： localStorage[ˈloʊkl] 和sessionStory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: webwoker / websocket (säkit)/ Geolocation(ˌjēōlōˈkāSHən)

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['spkɪt]/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loacalStorage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,需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

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	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

伪类:

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

伪元素:

::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样式优先级计算法:

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

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	决定项目在主轴的对齐方式 * space-between：两端对齐，项目之间的间隔都相等。 * 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。
align-items	flex-start/flex-end/center/baseline/stretch	定义项目在交叉轴上如何对齐 * baseline: 项目的第一行文字的基线对齐。 * 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。
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

oreder	.item{order:1}	定义项目的排列顺序。数值越小，排列越靠前，默认为0。
flex-grow	.item{flex-grow:<number>}	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	.item{flex-shrink:<number>/*default 1*/}	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	.item{flex-basis:length/*default auto*/}	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	.item{flex:none}	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	.item { align-self: auto | flex-start | flex-end | center | baseline | stretch; }	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来的?

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'læps]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

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吃匹配,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是分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是设置杭行高的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**?**

visibilty：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-chang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 : -wedkit-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:

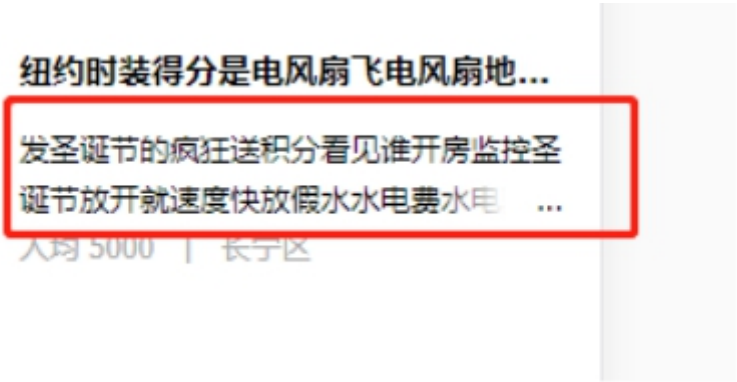
-webkit-line-clamp用来限制在一个块元素显示的文本的行数,为了实现该效果,它需要组合其它的webkit属性。

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，设置或减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



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（clipboardata.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。注意：根节点的字体大小需要动态设置，一般是屏幕的宽度/经验值（你自己定义）

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-demo{
  width:100px;
  height:100px;
  background-color:#06c;
  margin: auto;
  position:absolute;
```

```
  top: 0;
  left: 0;
  bottom: 0;
  right: 0;
 }
```

· 水平垂直居中 —— **方式二**

```
.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%);
 }
```

· 水平垂直居中 —— **方式三**，（新旧伸缩盒兼容）

```
  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;
  }
```

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，你可能会看到：

auto	浏览器计算内边距。
length	规定以具体单位计的内边距值，比如像素、厘米等。默认值是 0px。
%	规定基于父元素的宽度的百分比的内边距。
inherit	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
```

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：**

方法一:

html,body{ margin:0;width:100**%**; }

h3{ height: 100**px**; margin: 20**px** 0 0; }

#left,*#right*{ width:200**px**;height: 200**px**;background: **#****000**; position: absolute;top: 120**px**;}

#left{left:0**px**;}

#right{right: 0**px**;}

#center{margin:2**px** 210**px** ;background-color: **#****eee**;height: 200**px**; }

方法二:

div{

height: 300**px**;

}

#left,*#right*{

width: 200**px**;

background: **#****000**;

}

#left{

float:left;

}

#right{

float:right;

}

#center{

margin-left:200**px**;

margin-right:200**px**;

}

实现三列宽度自适应布局

左边右边中间

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，并且会有一点小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从新显示文档，造成页面闪烁

解决方法：把样式表放到文档的head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*1000ms = 16.7ms

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

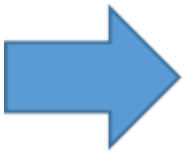
作用：程序安全，内存的释放

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，然后用函数吧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****

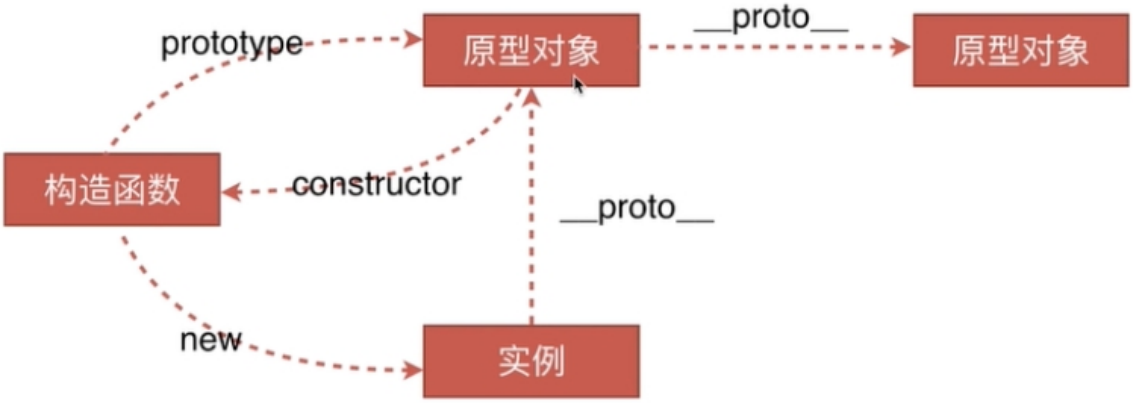
原型：函数都要prototype(显示原型)属性，而prototype会自动初始化一个空对象，这个对象就是原型对象

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 instanseof 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,补鞲呢放在构造器中

\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。

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

动态创建DOM方式：**

```
<script type="text/javascript">
   function downloadJSAtOnload() {
       varelement = document.createElement("script");
      element.src = "defer.js";
      document.body.appendChild(element);
   }
   if (window.addEventListener)
      window.addEventListener("load",downloadJSAtOnload, false);
   else if (window.attachEvent)
      window.attachEvent("onload",downloadJSAtOnload);
   else window.onload =downloadJSAtOnload;
</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：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 ['junɪˌkod]码排序的

可以在sort()中添加一个比较函数函数:

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

Math.rendom()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 windHeight = $(window).width()/$(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 :

```
// 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'));
        // 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();
```

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

```
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
```

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

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", "iPad", "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 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?

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

clientWidth、clientHieight返回值值包含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

字符串操作方法。

splite（）：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（）：可返回某个自妇女换在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菲酵犯缌□?-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
}
```

· 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<southSu.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.innerElement

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。

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为扩展插件提拱了两个方法

· $ .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targt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

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.

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

基于标准化的被广发支持的技术，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

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');
    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)

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

2>子窗口(http://child.domain.com/b.html)

```
document.domain = 'domain.com';
// 获取父窗口中变量
alert('get js data from parent ---> ' + window.parent.user);
```

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) 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会在处理玩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localstorge、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

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生命的变量不能通过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。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() => {};
```

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。

```
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}
```

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}

async函数的基本用法:

asyn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awai他命令后面，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Peomise对象，这比Generator函数的返回值是Iterator对象方便对了，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;

· 理解了这个顺序，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

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

```
//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();
```

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

```
//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}`);
```

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. 可以在对象子里面量里面定义原型

\2. 定义方法可以不用function关键词

\3. 直接调用父类方法

```
var human = {
  breathe(){
```

```
    console.log('breathing...');
  }
};
var worker = {
  __proto__:human, //设置此对象的原型为human,想觉羽继承human
  company:'freelancer',
  work(){
    console.log("working..")
  }
}
human.breathe();
//调用继承来的breathe方法
worker.breathe();
```

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。

```
var [x,y] = getVal(), //函数返回值解析
  [name, ,age] = ["wayou","male","secrect"]; //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
```

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。

```
//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
```

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。这只是一种语法糖，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。

```
//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
```

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）

```
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
```

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。

```
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
```

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

```
var someArray = ['a','b','c'];
for(v of someArray){
  console.log(v); //a,b,c
}
```

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。

```
//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 Ponit(0,0);
console.log(origin)
```

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。

```
//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
```

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。

```
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
```

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。

```
//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"
});
```

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

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

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: '
A custom component!
'
})
```

3) 使用组件

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HTMLElement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

为什么使用**key?****

当有相同标签名的元素切换时，需要通过key特性设置唯一的值来标记已让vue区分它们，否则vue为了效率只会替换相同标签内部的内容。

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"，简写为"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} form '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，另外一个是函数。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）、updata（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**响应式原理是什么？**

vue.js是才用数据劫持介个发布-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。

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

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**：

在执行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

路由之间的跳转:**

声明式（标签跳转）：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

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:

Vue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生命周期函数:

这个问题考察组件的声明周期：

一．初始化阶段

componentWillMount:组件即将被装载、渲染到页面上

render：组件在这里生成虚拟的DOM节点

componentDidMount：组件真正在被装载之后

二．运行中状态

componentWillReceiverProps：组件将要接收到属性的时候调用

shouldComponentUpdate：组件接受到新属性或者新状态的时候（可以返回false，接受数据后不更新，组织render调用，后面的函数不会被继续执行了）

componentWillUpdate：组件即将更新不能修改属性和状态

render：组件重新描绘

三．销毁阶段

componentWillUnmount：组件即将销毁

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嗲用次数会不确定，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才会要求响应。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propType验证以及其他的告警信息，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。

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、.unmountComponentAtNode0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(

<input type="text" ref={(input) => this.input = input}/>

Submit


)

}

}
```

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 CustomFrom({handleSubmit}){

let inputElement

return (

<form onSubmit={()=>handleSubmit(inputElement.value)}>

<input type='text' ref={(input)=>inputElement=input}/>

Submit


)

}
```

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(

    {this.state.todoItems.map(({task,uid})=>{

    return

- {task}

    })}

)
```

}

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

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。

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）的重要性和使用场景**

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(shadow 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-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

l 路由集中在一处；

l 布局和页面的层叠由层叠的 组件控制；

l 布局和页面组件是路由的一部分；

React Router 4

l 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。

l 不需要再在嵌套组件中使用 {props.children}

l 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browerHistory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

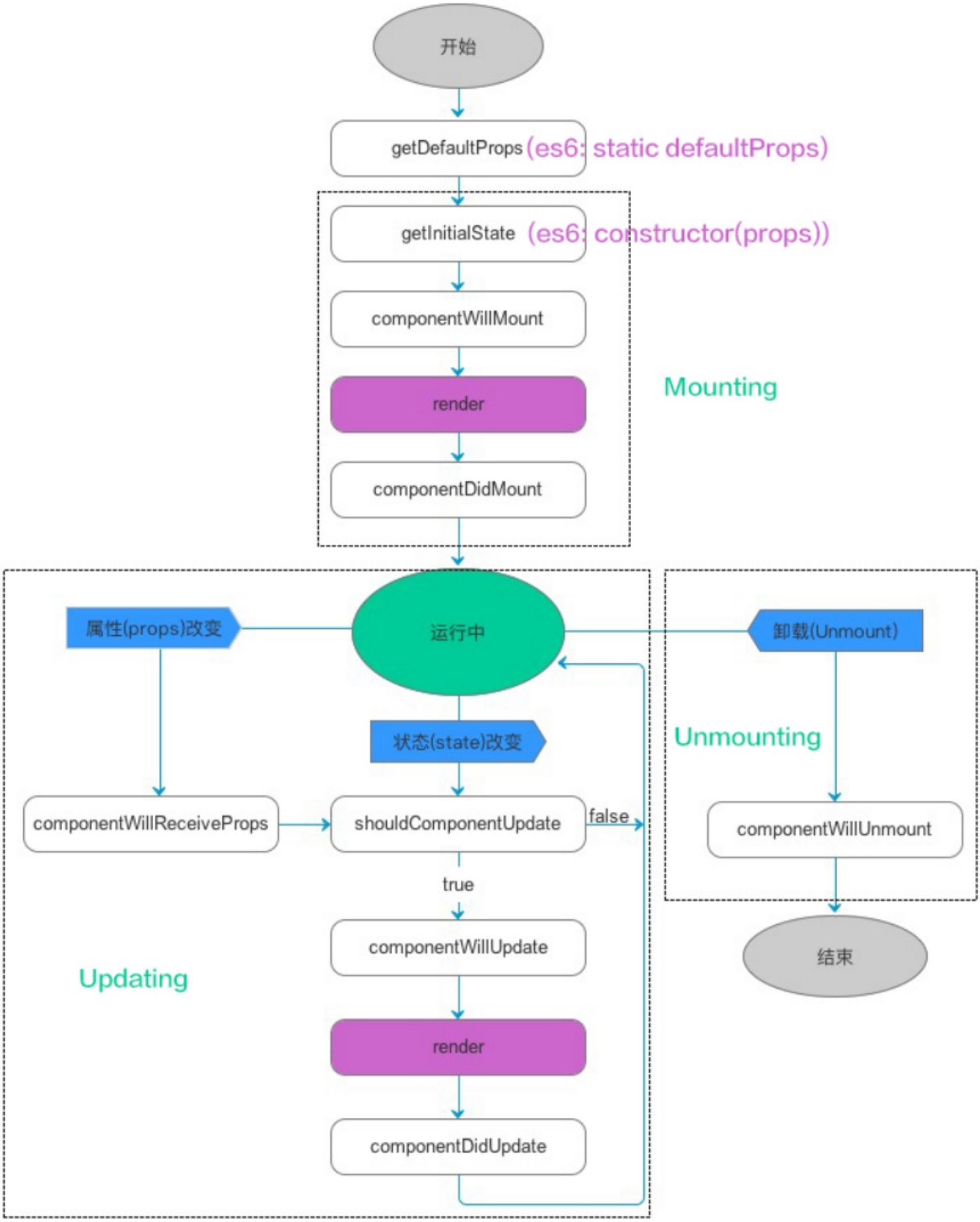
\4. 组件化开发，代码重用性高

\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：**

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成的浏览器，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 可同时迸发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

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读点击，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foucus、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devicePiexRatio =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img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;

}

var box = document.getElementById("box");

box.onclick = function(){

console.log("123");

}

防止手机中网页放大和缩小

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：

-

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 tmpl ='';
 for(var i = 1; i <= totla; i++){
   var p = padding;
   var imgSrc = 'img/' + i + '.jpg';
   if( i%4 == 1){
    p=0;
   }
 tmpl +='
```

-

```
';
 var imageObj = newImage();
 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（touchastart）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cilck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

点击拨打15677776767

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

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

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**

IOS下伪类:hover

除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

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

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

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/,prikəm'poz/" 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madia来指定不同的大小

<link rel="apple-touch-startup-image"href="start.png"/>

<link href="apple-touch-startup-image-320x460.png"media="(device-width: 320px)" rel="apple-touch-startup-image"/>

<link href="apple-touch-startup-image-640x920.png"media="(device-width: 320px) and (-webkit-device-pixel-ratio: 2)" rel="apple-touch-startup-image"/>

<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">

<link href="apple-touch-startup-image-768x1004.png"media="(device-width: 768px) and (orientation: portrait)" rel="apple-touch-startup-image"/>

<link href="apple-touch-startup-image-748x1024.png"media="(device-width: 768px) and (orientation: landscape)" rel="apple-touch-startup-image"/>

<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"/>

<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

强制竖屏

强制横屏

应用模式

//UC浏览器私有

全屏模式

强制竖屏

强制横屏

应用模式

//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

微软的老式浏览器

windows phone 点击无高光

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">

document.getElementById('testInput').addEventListener('input',function(e){

```
  varvalue = e.target.value;
 });
```

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：**

```
functioncheckTextLength(obj, length) {
  if( obj.value.length > length) {
   obj.value = obj.value.substr(0, length);
  }
}
```

问题2解决方案：**

```
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;

}

其他面试题：

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；其次仁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，需要知道其他哟范湖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/ /'riːdə/ 、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fromdata

· 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

项目类问题

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

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链接有限，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javascrip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hi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

浏览器

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

.

.

.

笔试代码题：

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和重新绘制节点，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。**

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';
}
}
}
```

简单实现双向数据绑定**mvvm**

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

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

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,'')
```

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***?****

```
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
```

说说以下代码运行会输出什么?**

```
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
```