盒模型：

1. 从内到外分别是：content、padding、border、margin.
2. 两种模型：标准模型：width只包含content；IE模型：width包含content、padding、border；可以用box-sizing设置content-box（标准模型），border-box（IE模型）

链接：<https://www.cnblogs.com/chengzp/p/cssbox.html>

BFC (块格式化上下文):

渲染规则：默认根元素（html 元素）会创建一个 BFC，其块级盒子元素将会按照如下规则进行渲染：

● 块级盒会在垂直方向，一个接一个地放置，每个盒子水平占满整个容器空间

● 块级盒的垂直方向距离由上下 margin 决定，**同属于一个 BFC 中的两个或以上块级盒的相接的 margin 会发生重叠**

● BFC 就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此

● 计算 BFC 的高度时，浮动元素也参与计算

# 如何解决外边距叠加的问题？

根据 BFC 的定义，两个元素同属于一个 BFC 中的两个或以上块级盒的相接的 margin 会发生重叠。

● 1.如果不想重叠，将某一个设置成inline-block，让某一个不是BFC即可（不好）

● 2.把这两个 <div> 分别放进两个建立了 BFC 的父级容器里（用上面的BFC设置都行）

链接：<https://www.zhihu.com/question/19823139>

链接：<https://developer.mozilla.org/zh-CN/docs/Web/Guide/CSS/Block_formatting_context>

清除浮动：

1. clear:both;

<div class=“container clearfix”>

<div class=“box” style=“float:left;”></div>

</div>

.clearfix::after{

content:“”;

display:table;

clear:both;

}

1. overflow:hidden;(创建BFC)

<div class=“container”>

<div class=“box” style=“float:left;”></div>

</div>

.container{

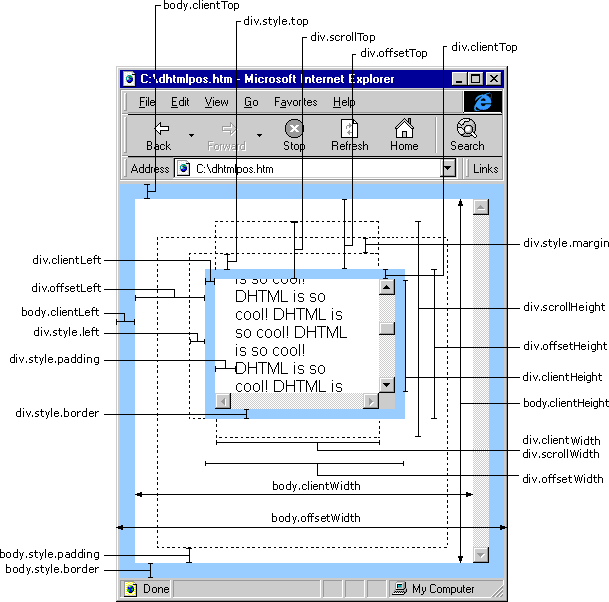
overflow:hidden;

}

圣杯布局&双飞翼布局：

链接：<https://blog.csdn.net/wangchengiii/article/details/77926868>

[clientHeight , scrollHeight , offsetHeight之间的区别](http://www.cnblogs.com/nanshanlaoyao/p/5964730.html)&Element.getBoundingClientRect()：



以下属性或方法会刷新渲染队列

* offsetTop、offsetLeft、offsetWidth、offsetHeight
* clientTop、clientLeft、clientWidth、clientHeight
* scrollTop、scrollLeft、scrollWidth、scrollHeight
* getComputedStyle()（IE中currentStyle）

我们在修改样式过程中，要尽量避免使用上面的属性

● InnerHeight/innerWidth与上面3者的区别：

只读属性，声明了窗口的文档显示区的高度和宽度，以像素为单位。这里的宽度和高度不包括菜单栏、工具栏以及滚动条等的高度。只有window有这个属性，用window.innerHeight/innerWidth获取。

● outerHeight/outerWidth同理：

outerHeight属性设置或返回一个窗口的外部高度，包括所有界面元素（如工具栏/滚动条）。outerWidth属性设置或返回窗口的外部宽度，包括所有的界面元素（如工具栏/滚动）。

移动端适配：rem、vw/vh/vmin/vmax、viewport、media query

viewport：<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">

一般写为<meta name="viewport" content="width=device-width, initial-scale=1">即可

media query：@media screen and (max-width:720px) and (min-width:320px)

模块化：防止全局污染、去耦合、方便维护。解决命名冲突和模块间依赖管理的问题。

工程化：因为开发环境需要模块化、可能使用Sass或TypeScript编程，因此会导致文件数过多，影响HTTP请求；而且，Sass和TypeScript浏览器也无法解析，需要将其转化为CSS和JavaScript。所以，需要进行转化、压缩、打包后将文件放在线上环境中进行，而自动处理这个过程的行为，称为模块化。

CSS优先级：（CSS2特异性）

● 第一优先级：无条件优先的属性只需要在属性后面使用！important。它会覆盖页面内任何位置定义的元素样式。ie6不支持该属性。

● 第二优先级：在html中给元素标签加style，即内联样式。该方法会造成css难以管理，所以不推荐使用。

● 第三优先级：由一个或多个id选择器来定义。例如，#id{margin:0;}会覆盖.classname{margin:3pxl}

● 第四优先级：由一个或多个类选择器、属性选择器、伪类选择器定义。如.classname{margin:3px}会覆盖div{margin:6px;}

● 第五优先级：由一个或多个元素选择器、伪元素选择器定义。如div{marigin:6px;}会覆盖\*{margin:10px；}

● 第六优先级：通配选择器，如\*{marigin:6px;}

HTTP2：

1. 二进制分帧：比起像HTTP/1.x这样的文本协议，二进制协议解析起来更高效、“线上”更紧凑，更重要的是错误更少。因为它们对如空白字符的处理、大小写、行尾、空链接等的处理很有帮助。

（在应用层(HTTP/2)和传输层(TCP or UDP)之间增加一个二进制分帧层在二进制分帧层中， HTTP/2 会将所有传输的信息分割为更小的消息和帧（frame）,并对它们采用二进制格式的编码，，其中 HTTP1.x 的首部信息会被封装到 HEADER frame，而相应的 Request Body 则封装到 DATA frame 里面。）

1. 多路复用：允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息（在 HTTP/1.1 协议中 「浏览器客户端在同一时间，针对同一域名下的请求有一定数量限制。超过限制数目的请求会被阻塞」）
2. 一个域只需要一个TCP连接：H1虽然有Connection:keep-alive保证持续连接，但也有max-connections数，（TCP 慢启动）所以有可能一次请求需要多次TCP连接；H2实现一个TCP请求资源无上限。（HTTP/2 通过让所有数据流共用同一个连接，可以更有效地使用 TCP 连接，让高带宽也能真正的服务于 HTTP 的性能提升。）
3. 头部压缩：使用HPACK进行头部压缩（因为HTTP头部更替不是很频繁，所以正在client和server端做个缓存，http2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小。）
4. 服务端推送（Server Push）：H2允许服务器，预先将网页所需要的资源PUSH到浏览器的内存中（如请求page.html，server会push给我们script.js和style.css，省去了requset的开销）

链接：<https://www.zhihu.com/question/34074946>

链接：<https://blog.csdn.net/jiayanhui2877/article/details/44957105>

Vue生命周期

beforecreated：el 和 data 并未初始化   
created:完成了 data 数据的初始化，el没有  
beforeMount：完成了 el 和 data 初始化   
mounted ：完成挂载

链接：<https://segmentfault.com/a/1190000008010666>

LocalStorage:



链接：<https://developer.mozilla.org/zh-CN/docs/Web/API/Window/localStorage>

设计模式：

单例模式：用一个命名空间将自己的属性与方法包起来，防止与他人的属性和方法冲突；并且，可以用来管理私有变量；

同源策略：是一种约定。

● 同源政策的目的：是为了保证用户信息的安全，防止恶意的网站窃取数据。

● 同源策略应该是浏览器在接收加载资源之前对其来源进行了检查，然后限制加载。

● 目前，如果非同源，共有三种行为受到限制：

（1） Cookie、LocalStorage 和 IndexDB 无法读取。

（2） DOM 无法获得。

（3） AJAX 请求不能发送。

● AJAX的同源策略限制：可能限制发出（保护用户的cookie）或者限制接收（隔绝不信任的服务器返回的数据）【根据MDN：https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access\_control\_CORS，不同浏览器实现方式不同，看浏览器产商自己怎么做。一般是限制发出】

如果使用CORS，得跟服务器先协商好，（默认用户自己手动请求url的服务器可信），如果返回Access-control-allow-orign的话，说明此服务器可信，数据可接收（红宝书写：CORS定义了在必须访问跨域资源时，浏览器与服务器应该如何沟通。定义CORS是一种沟通方式）。

如果用户自己请求了某个恶意网站，这个网站返回Access-control-allow-orign:\*，就会接收恶意数据了。

（这是一种XSS+CSRF的攻击方式：诱导用户去点小广告，点完便执行请求返回恶意数据）

但不能获取用户的cookie：如果设置access-control-allow-origin为\*星号（任何域），则Access-Control-Allow-Credentials头是不能设置为true的。并且客户端也需要设置xhr.withCredentials = true;

一个面试问题：（如果不同源的话，浏览器会报错，不能获得服务器的数据。如果我是攻击者，我发送请求到后台以求获得数据，但这个行为还得征得服务器同意，那么同源策略是保护服务器还是浏览器的？）

（答：都有保护作用。主要保护用户客户端的信息不被其他域获取，服务器如果没有做过滤的话肯定也会被攻击。同源策略就像一个防火墙，保护同一域下的信息不会泄露出去。但是这个要配合很多技巧，不然也还是能获取的（各种攻击手段了解一下），只是不那么容易获取罢了）

AJAX原理：<https://www.cnblogs.com/jackson0714/p/AJAX.htm>

CORS(阮一峰)：<http://www.ruanyifeng.com/blog/2016/04/cors.html>

Git pull & git fecth：

● git fetch只会将本地库所关联的远程库的commit id更新至最新

● git pull会将本地库更新至远程库的最新状态

● git pull = git fetch + git merge



链接：<https://blog.csdn.net/a19881029/article/details/42245955>

# [Form表单点击submit按钮提交，阻止页面跳转到action地址：](https://segmentfault.com/q/1010000006779322)

方法1：监听 form 的 onsubmit 事件

$('form').on('submit', function (e) {

e.preventDefault()

// ....

}

方法2：让target指向隐藏的iframe

<https://blog.csdn.net/wobendiankun/article/details/38667379>

方法3：使用Ajax

flex布局中align-items 和align-content的区别：

1.align-items属性适用于所有的flex容器，它是用来设置每个flex元素在侧轴上的**默认对齐方式**。

2.align-content属性**只适用于多行**的flex容器，并且当侧轴上有多余空间使flex容器内的flex线对齐。

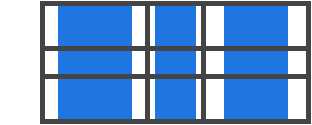
\*注：flex只有justify-content没有justify-items

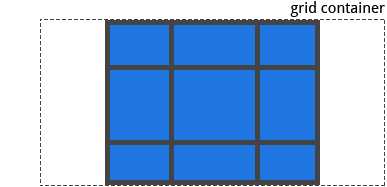
链接：<https://blog.csdn.net/sinat_27088253/article/details/51532992>

Grid布局中align-items/justify-items 和align-content/justify-content的区别：

1. 一般用align-items/justify-items，

某个item修改则用align-self/justify-self



2.align-content/justify-content：有时，网格的总大小可能小于其网格容器的大小。如果你的所有 grid items 都使用像px这样的非弹性单位来设置大小，则可能发生这种情况。此时，你可以设置网格容器内的网格的对齐方式。 此属性沿着行轴对齐网格（与之对应的是 align-content, 沿着列轴对齐）。

#### 链接：<https://segmentfault.com/a/1190000012889793#articleHeader5>

强缓存：状态码：200；相关字段：cache-control：max-age、expire

协商缓存：状态码：304(无改动)、200(有改动)；相关字段：Etag(if-None-Match)、last-modefied(If-Modified-Since)

# Promise的队列与setTimeout的队列有何关联



原因：

因为 setTimeout 是属于 macrotask 的，而整个 script 也是属于一个 macrotask, promise.then 回调 是 microtask ,执行过程大概如下：

* 由于整个 script 也属于一个 macrotask, 由于会先执行 macrotask 中的第一个任务，再加上**promise 构造函数因为是同步的**，所以会先打印出 1和2
* 然后继续同步执行末尾的 console.log(3) 打印出3
* 此时 setTimeout 被推进到 macrotask 队列中， promise.then 回调被推进到 microtask 队列中
* 由于在第一步中已经执行完了第一个 macrotask , 所以接下来会顺序执行所有的 microtask, 也就是 promise.then 的回调函数，从而打印出5
* microtask 队列中的任务已经执行完毕，继续执行剩下的 macrotask 队列中的任务，也就是 setTimeout, 所以打印出 4

“宏任务”在下一轮Event Loop执行，“微任务”在本轮Event Loop的所有任务结束后执行

链接：<https://www.zhihu.com/question/36972010>

let & const & var :

● let 声明的变量的作用域是块级的；

● let 不能重复声明已存在的变量；

● let 会被提升，只是有暂时死区（ 所谓暂时死区，就是不能在初始化之前，使用变量 ）

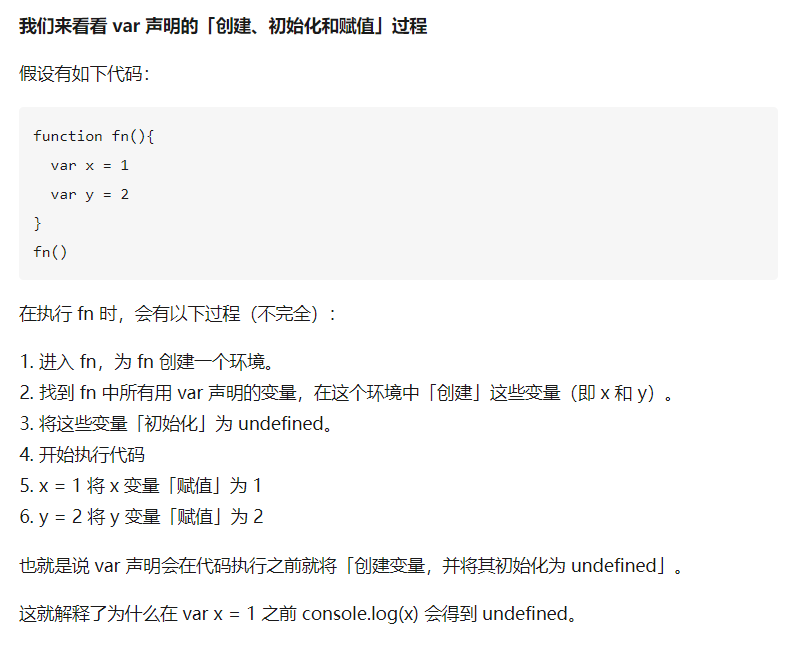
● 提升：

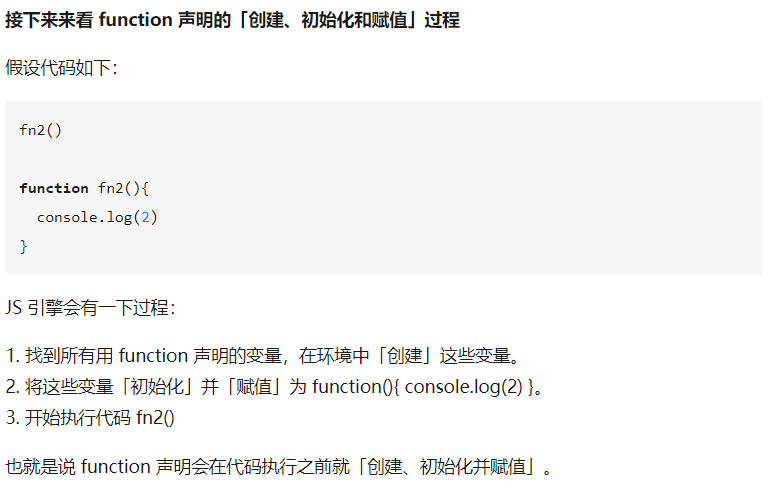
● let 的「创建」过程被提升了，但是初始化没有提升。（MDN说不会提升是错的）

● var 的「创建」和「初始化」都被提升了。

● function 的「创建」「初始化」和「赋值」都被提升了。

● const 和 let 只有一个区别，那就是 const 只有「创建」和「初始化」，没有「赋值」过程







链接：<https://zhuanlan.zhihu.com/p/28140450>

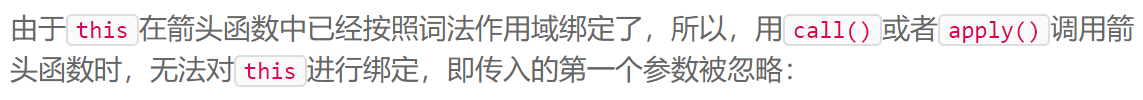
链接：https://zhuanlan.zhihu.com/p/27558914

箭头函数的this（箭头函数与普通函数的区别）：



● 箭头函数的this对象是绑定定义时时所在的作用域。

● 普通函数的this对象是绑定使用时所在的作用域。

链接：https://www.liaoxuefeng.com/wiki/001434446689867b27157e896e74d51a89c25cc8b43bdb3000/001438565969057627e5435793645b7acaee3b6869d1374000

链接：<http://es6.ruanyifeng.com/#docs/function#%E7%AE%AD%E5%A4%B4%E5%87%BD%E6%95%B0>

## **export与export default**

export跟export default 有什么区别呢？如下：

export与export default均可用于导出常量、函数、文件、模块等  
2、你可以在其它文件或模块中通过import+(常量 | 函数 | 文件 | 模块)名的方式，将其导入，以便能够对其进行使用  
3、在一个文件或模块中，export、import可以有多个，export default仅有一个  
4、通过export方式导出，在导入时要加{ }，export default则不需要

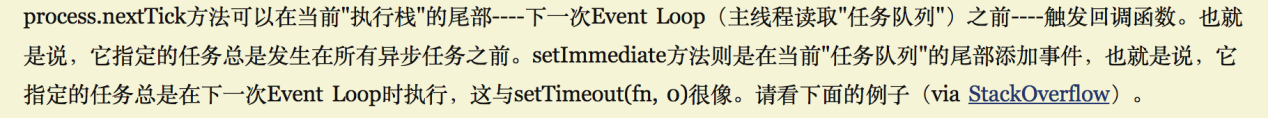
5、

链接：<https://www.cnblogs.com/xiaotanke/p/7448383.html>

链接：<https://segmentfault.com/q/1010000010126010>

链接：<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/export>

setImmediate & process.nextTick:



1.process.nextTick：添加到执行栈的尾部

2.setImmediate：添加到任务队列的尾部，浏览器端相当于setTimeout(f(){},0)，Node端是在IO callback之后

3.setImmediate&setTimeout这么写的话，顺序不确定,这得看cpu：

链接：<http://www.ruanyifeng.com/blog/2014/10/event-loop.html>

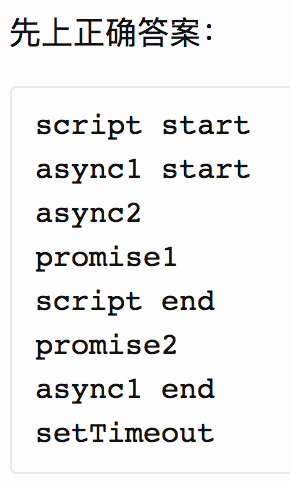
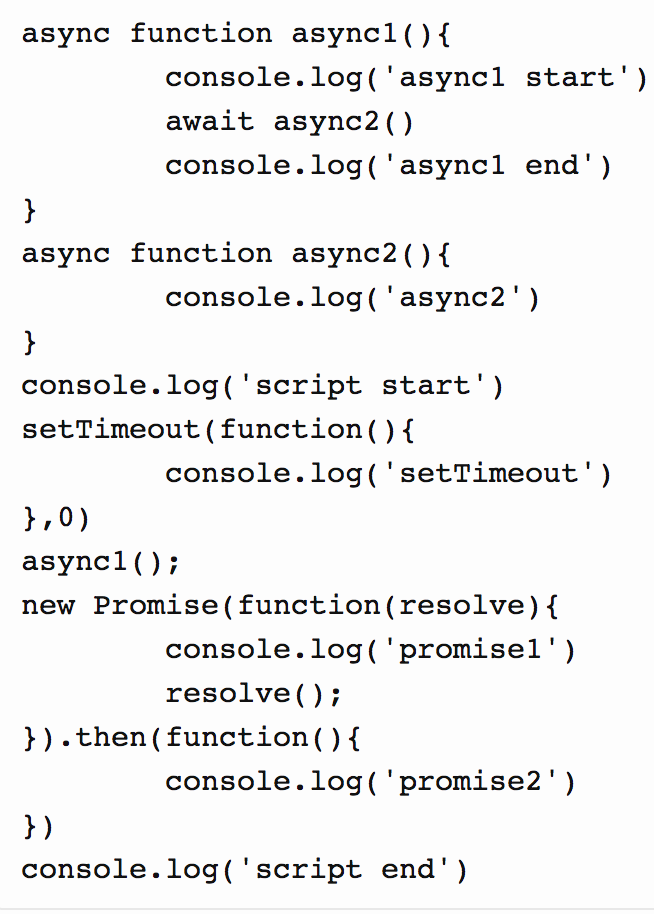
宏任务与微任务：

正常任务（task）与微任务（microtask）。它们的区别在于，“正常任务”在下一轮Event Loop执行，“微任务”在本轮Event Loop的所有任务结束后执行。

正常任务包括以下情况：setTimeout、setInterval、setImmediate、I/O、各种事件（比如鼠标单击事件）的回调函数

微任务目前主要是process.nextTick和 Promise 这两种情况

# 链接：<https://blog.csdn.net/shuidinaozhongyan/article/details/78162134>promise、async和await之执行顺序的那点事:



要点：

1.先执行调用栈的同步事件；

2.setTimeout里的函数是异步执行，放在宏任务队列（下一次event loop执行）；

3.Promise是立即执行（同步执行）；Promise里的resolve是异步执行，并且放在微任务队列（本次event loop末尾执行）；

4.async 函数返回一个 Promise 对象;

5.当函数执行的时候，一旦遇到 await 就会先返回，等到触发的异步操作完成，再接着执行函数体内后面的语句；

过程：

1. 执行一遍调用栈的同步事件，输出如上：script start-->...-->script end
2. 此时调用栈为空；微任务队列有（1）async2执行结束返回的Promise对象、（2）new Promise的resolve；宏任务队列有setTimeout里的函数。
3. 先调用“async2执行结束返回的Promise对象”，执行其Promise对象，执行到resolve之前（这是一个值为Undefined的resolve），将resolve放入微任务队列末尾；
4. 接着执行下一个微任务“new Promise的resolve”，输出‘promise2’；
5. 接着执行刚刚放进去的微任务“async2的Promise对象resolve”，无输出；
6. 接着因为await后面的函数执行结束，开始执行await下面的代码，输出’async1 end’;
7. 接着因为微任务队列为空，所以将宏任务队列的setTimeout放入的函数放入调用栈并执行，输出’setTimeout’
8. 结束

链接：<https://lvdingjin.github.io/tech/2018/05/27/async-and-await.html>

JavaScript的Object查询时间复杂度是O(1)：

V8的properties和elements都有“快速”和“慢速”模式。快速模式用密集的存储方式（FixedArray），慢速模式用哈希表（Dictionary）。对数组而言，稀疏数组一定是用慢速模式。

链接：<https://www.zhihu.com/question/30848981>

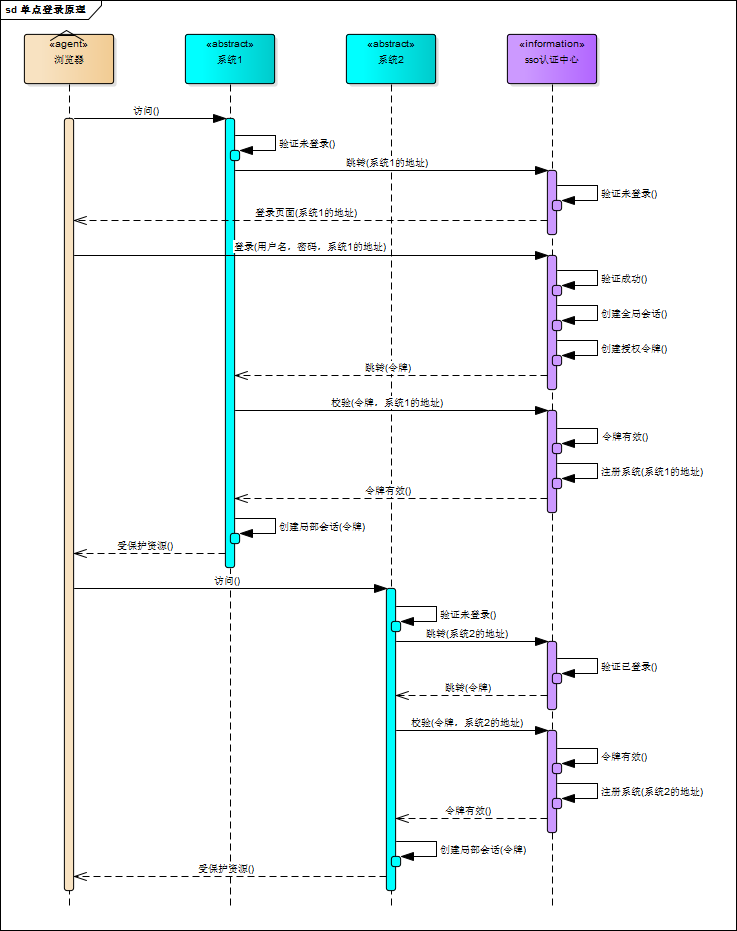
Element.getBoundingClientRect()

返回元素的大小及其相对于视口的位置

配合screen.availHeight（可用的屏幕高度），可以实现下拉刷新，监控曝光

链接：<https://developer.mozilla.org/zh-CN/docs/Web/API/Element/getBoundingClientRect>

# 单点登录之CAS原理和实现：





过程：（ssosv是sso的登陆页面）

1. 访问服务器a请求数据
2. 服务器a验证为未登陆状态，（携带服务器a的地址）请求sso服务器验证登陆状态
3. sso服务器（携带服务器a的地址）302跳转到ssosv页面
4. ssosv页面携带cookie请求ssosv网关
5. ssosv网关302后到ssosv登陆接口，并携带cookie请求
6. ssosv服务器验证通过，返回ticket并302到sso服务器
7. sso服务器验证ticket通过，200
8. 服务器a创建session回话

链接：

<https://blog.csdn.net/qq_24708791/article/details/78535565>

https://www.cnblogs.com/ywlaker/p/6113927.html

React/Vue区别：

MVC/MVVM，大型小型应用，JSX/模板，v-for/map，state状态管理/data数据

响应式开发最佳实践（viewport+rem）:

viewport只是改变document.documentElement.clientWidth或window.innerWidth

viewport的值取width与 initial-scale的最大值

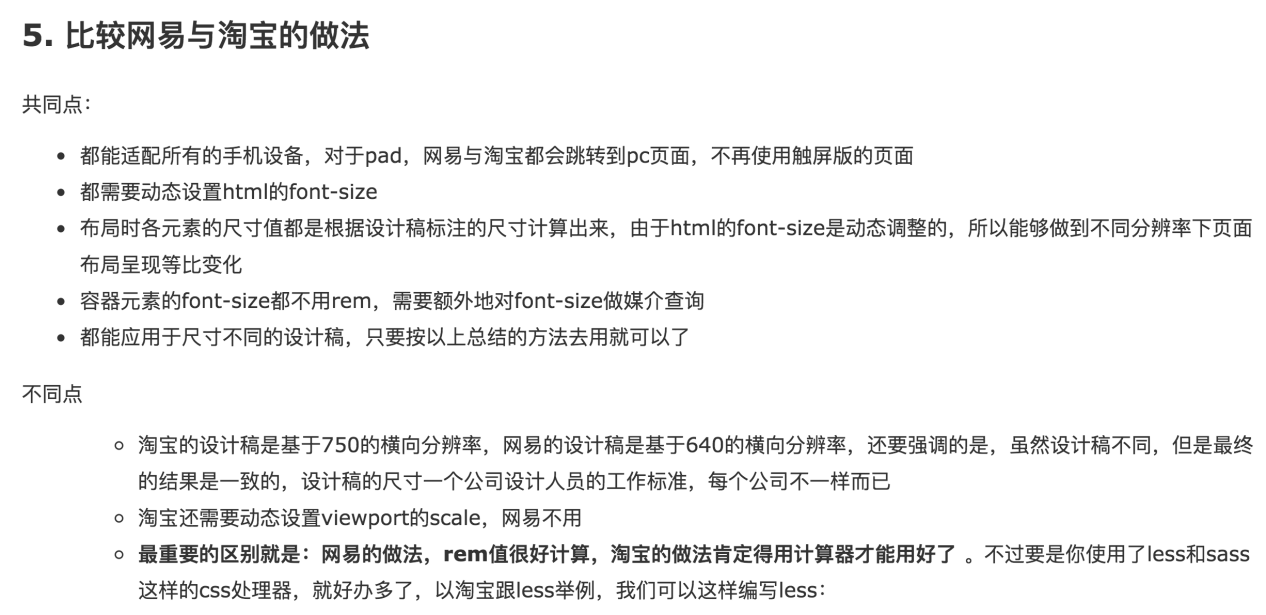
链接：https://github.com/jawil/blog/issues/21

响应式开发最佳实践: <http://www.cnblogs.com/lyzg/p/4877277.html>

**在实际项目中，把与元素尺寸有关的css，如width,height,line-height,margin,padding等都以rem作为单位，这样页面在不同设备下就能保持一致的网页布局。**

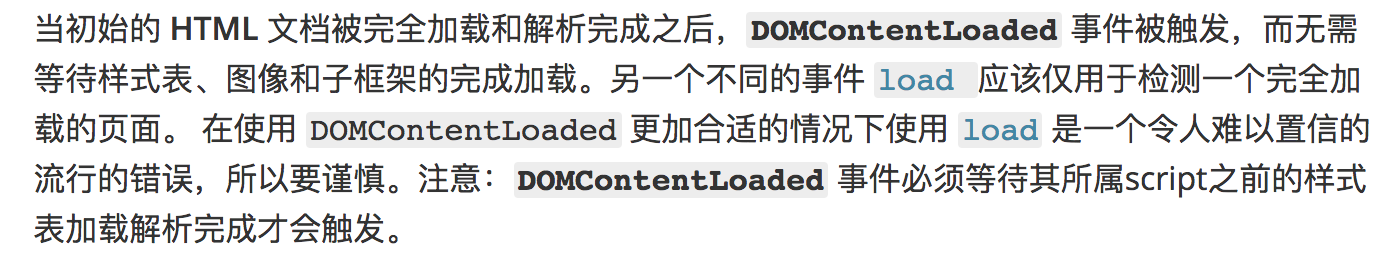
**Font-size用媒体查询改变px，不能设为rem。**





Domready: <https://www.jianshu.com/p/88b9d3874749>

Domready使用“DOMContentLoaded”事件





链接：<https://developer.mozilla.org/zh-CN/docs/Web/Events/DOMContentLoaded>

获取一个页面的节点类型数量：

```

new Set([...document.querySelectorAll('\*')].map(node => node.nodeName)).size

```

1. index原则：

一、

z-index只对设置了position(非static)的元素有用

二、

一般原则：

1. 写在后面优先级大
2. 设置了position优先级比无设置的大
3. 都设置了position，则比较z-index
4. Z-index相同，则写在后面优先级大

三、

子元素一定在父元素上面

四、

若父元素1无设置position，子元素2设置position，兄弟元素3设置了position

则3在1上面，再3与2比较z-index

