

coderwhy前端八股文（一）

SEO相关的题目在面试被问的较多，一方面是SEO对于Web站点很重要，另一方面很多同学的简历中都会写到SEO相关的表述。

另外SEO是一个相对宽泛的话题，所以回答的好坏差别会很大。

01-什么是SEO（搜索引擎优化）？为什么SEO对于一个网站至关重要？

SEO是搜索引擎优化，它的英文全称是*Search Engine Optimization*

- 因为现在很多我们开发的网站，类似于门户网站或者功能网站（比如我之前在企业里面开发的弘源旅途、星客SC、网易云音乐），一方面我们需要通过营销宣传来提升我们产品的知名度，另一方面靠自然搜索结果获取流量也是一个非常重要的过程。
- 所以我们需要在了解一定的搜索引擎自然排名机制的基础上，对我们的网站进行内部和外部的调整优化，让用户在使用关键字搜索时我们的网站可以尽量高的提升自然排名，获取更多的流量，从而达到我们预期的销量以及品牌的知名度。

关键点：

- 关键一：回答出问题本身，比如说问到你SEO优化是什么了，你就要回答出来
- 关键二：结合真实的项目或者案例来回答，越回答对应的项目，越真实，而不是你背诵的八股文
- 关键三：可以做一定的引导，会引导到你做的项目上，我们提前针对项目准备的很多问题就能派上用场了

02-SEO有哪些关键点？你在日常开发中，都采取了哪些措施来进行SEO呢？

从重要到次要依次来说明，如果中间面试官开始根据你的回答问其他问题，那么就可以暂时停下来回答其他问题。

当然目前在国内针对百度有一个很直接的SEO方式就是给钱，其实有可以通过不给钱的方式来提升网站关键字排名的方案。

方式一：SSR服务器端渲染

因为我之前企业的项目，包括现在很多现在新的项目，都是基于现代的框架，比如Vue、React来开发的，大部分页面元素是由客户端JavaScript动态生成的。很多的搜索引擎，在爬虫时只能抓取静态的HTML源代码，而不会执行JavaScript，因此动态生成的内容无法被爬虫索引。另外很多的搜索情况不会等待一部数据加载完成后再进行抓取，也会导致我们网站的很多关键信息不能被完整的收录。

为了确保网站的SEO优化，我们之前的项目需要SEO优化的都采用了SSR技术。SSR能够在服务器上执行JavaScript并渲染出完整的HTML页面，然后将其发送到客户端。这样，爬虫在抓取网站时就能获取到完整的页面内容，从而提升SEO效果。

如果是开发初期就计算进行SEO优化的话，我们一般会直接选择一些比较成熟的SSR框架，比如对于Vue来说选择Nuxt.js，对于React来说选择Next.js。

后续就是一些开发中的细节了

方式二：准确的TDK描述

TDK就是我们常说的（面试）title、description、keywords

- Title（标题）：也就是网站显示的标题，不仅仅用户会看到，搜索引擎通常会首先检索和收录title信息，所以title至关重要。title一般不需要过长，多个关键词之间使用“|”或者“-”分割，会被搜索引擎提取和收录。
- Description（描述）：这是对网页内容的简短描述，通常在搜索引擎结果页中标题下方显示。描述应概述页面内容，包含相关关键词，并吸引用户点击。
- Keywords（关键词）：这是网页内容中重要的词汇，反映了页面的主题和内容，每个关键字都要有对应的内容匹配。虽然现代搜索引擎（如Google）对关键词标签的重视程度已经降低，但在某些情况下，合理使用关键词仍然有助于SEO。

方式三：语义化的HTML元素、图片alt、h1、h2的合理使用

语义化的HTML代码和符合W3C规范是SEO的关键要素之一。

- 语义化是指使用具有明确含义的HTML元素，搜索引擎在爬取网站时，也会更容易理解网站的内容以便进行收录，从侧面也能印证我们的网页更加的规范。而且这不仅有助于搜索引擎理解网页内容，还能提高网页的可读性和可维护性。
- 包括Header, Nav, Aside, Article, Footer元素，这些都能帮助爬虫更好的获取页面内容，理解网页。

图片要求必须加alt规范

- 我们要求每个前端在使用图片时，必须加上和图片相关的alt，一方面是图片无法显示时用户可以看到提示，另一方面也有利于SEO优化。

重要的标签h1/h2/h3等的使用

- H1、H2、H3等HTML标题标签在SEO中起着非常重要的作用。这些标签有助于搜索引擎理解网页内容的结构和层次，从而更准确地索引和评估页面的相关性。

方式四：编写合理的robots.txt文件

`robots.txt` 是一个存放在网站根目录中的文本文件，其主要作用是告诉搜索引擎爬虫哪些部分的网站可以被抓取（爬取）以及哪些部分不应该被抓取。

为什么需要使用'robots.txt'

- 通过指示搜索引擎忽略不重要的文件或目录，可以让搜索引擎更专注于重要内容的抓取和索引。
- 当然也可以避免一些敏感或私有内容被无意中索引。

所以如果网站不编写robots.txt，可能会降低网站的SEO效率，因为搜索引擎花费更多时间和资源在不重要的页面上。

举几个例子：

- [知乎](#)
- [爱彼迎](#)

方式五：HTTPS

自2014年以来，Google已将HTTPS作为其搜索排名的信号之一。这意味着，使用HTTPS的网站在搜索结果中可能会获得比非HTTPS网站更好的排名。

而且HTTPS也有利于用户的安全，在用户使用网站时也会增加信任度。

方式六：内部链接和外部链接

内部链接是指从一个页面到同一网站内另一个页面的链接。它可以提高提高网站导航、增强网站的权重、提升网站的索引。

外部链接是指从一个网站指向另一个网站的链接。在网页中放合适的外部链接，也有利用于提升网站的权重指数，容易被搜索引擎收录。

其他方式

当然还有一些其他的细节，比如sitemap文件、网站导航、响应式的处理，都在某种程度上能提高网站的权重。另外有一些企业还会专门请一些SEO的专员来进行SEO优化的操作，每个企业情况不太一样。

03-defer和async属性在script标签中分别有什么作用？

浏览器在解析HTML的过程中，遇到了script元素是不能继续构建DOM树的

- 它会停止继续构建，首先下载JavaScript代码，并且执行JavaScript的脚本；
- 只有等到JavaScript脚本执行结束后，才会继续解析HTML，构建DOM树；

为什么要这样做呢？

- 这是因为JavaScript的作用之一就是操作DOM，并且可以修改DOM；
- 如果我们等到DOM树构建完成并且渲染再执行JavaScript，会造成严重的回流和重绘，影响页面的性能；
- 所以会在遇到script元素时，优先下载和执行JavaScript代码，再继续构建DOM树；

但是这个也往往会带来新的问题，特别是现代页面开发中：

- 在目前的开发模式中（比如Vue、React），脚本往往比HTML页面更“重”，处理时间需要更长；
- 所以会造成页面的解析阻塞，在脚本下载、执行完成之前，用户在界面上什么都看不到；

为了解决这个问题，script元素给我们提供了两个属性（attribute）：defer和async。

defer的作用

defer 属性告诉浏览器不要等待脚本下载，而继续解析HTML，构建DOM Tree。

- 脚本会由浏览器来进行下载，但是不会阻塞DOM Tree的构建过程；
- 如果脚本提前下载好了，它会等待DOM Tree构建完成，在DOMContentLoaded事件之前先执行defer中的代码；

所以DOMContentLoaded总是会等待defer中的代码先执行完成。

```
<script defer src="./js/defer-demo.js"></script>
<script>
  window.addEventListener("DOMContentLoaded", () => {
    console.log("DOMContentLoaded")
  })
</script>
```

另外多个带defer的脚本是可以保持正确的顺序执行的。

从某种角度来说，defer可以提高页面的性能，并且推荐放到head元素中；

async的作用

async 特性与 defer 有些类似，它也能够让脚本不阻塞页面。

async是让一个脚本的下载和执行是独立的：

- 浏览器不会因 async 脚本的下载而阻塞（与 defer 类似）；
- async脚本会在下载好后立即执行，不能保证在DOMContentLoaded之前或者之后执行（执行时会阻塞DOM Tree的构建）；
- async脚本不能保证顺序，它是独立下载、独立运行，不会等待其他脚本；

```
<script>
  window.addEventListener("DOMContentLoaded", () => {
    console.log("DOMContentLoaded")
  })
</script>
<script async src="./js/async-demo.js"></script>
```

具体的执行时机



面试回答

`<script>` 标签的 `defer` 和 `async` 属性用来控制外部脚本文件的加载和执行方式，它们对于改善页面加载速度非常有帮助。

但是它们的机制并不相同：

- `defer` 的下载不会阻止 DOM 的构建，但是在 DOM Tree 构建完成后，在 DOMContentLoaded 事件之前，先执行脚本的内容，并且 `defer` 脚本的执行是有顺序的。
- `async` 的下载也不会阻止 DOM 的加载，而且不会保证在 DOMContentLoaded 之前或者之后执行，也不能保证顺序，它的每个脚本是独立进行的。

所以它们的应用场景是这样的：

- `defer` 通常用于需要在文档解析后操作 DOM 的 JavaScript 代码，并且对多个 `script` 文件有顺序要求的；
- `async` 通常用于独立的脚本，对其他脚本，甚至对 DOM 没有依赖的脚本；

在现代化框架开发过程中，往往不需要我们自己来配置 `async` 或者 `defer`，在使用脚手架或者自己搭建的 `webpack` 或者 `vite` 项目进行打包时，它会根据需要帮我们加上 `defer` 属性，某些情况下我们想要进行性能优化时，也可以手动的加上 `async` 属性（例如一些第三方的分析工具或者广告追踪脚本）。

04-你能描述一下CSS3引入的一些主要新特性吗？

比较好的回答是：其实并不存在真正意义上的 CSS3，因为我有阅读 W3C 的文档。从 CSS3 并不是一个单一的规范，而是一系列独立模块的集合，这些模块扩展了 CSS 的功能。

这种模块化的方法允许不同的特性以不同的速度发展，可以更快的标准化一些特性，而不必等待整个规范的完成。

比如说：

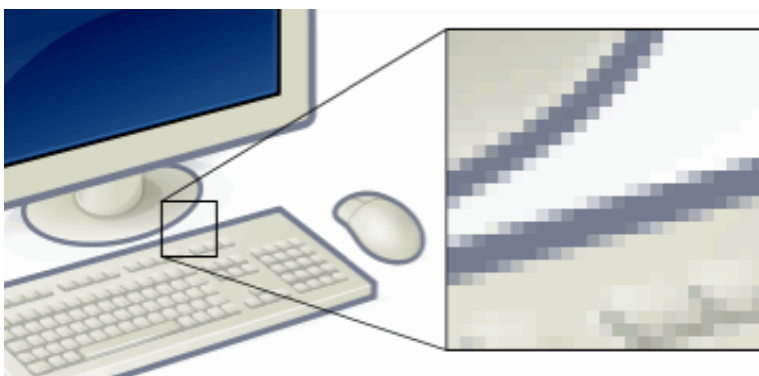
- 选择器 (Selectors) :
 - 新的属性选择器，如 `[attr^=value]`（属性值以特定字符串开始）；
 - 结构性伪类，如 `:nth-child`、`:nth-last-child`、`:first-of-type`；
- 背景和边框 (Backgrounds and Borders) :
 - 边框圆角 (`border-radius`)，简化了创建圆角效果的过程。
 - 边框图片 (`border-image`)，允许使用图片来创建边框。
 - 多重背景，支持在单个元素上使用多个背景图片。

- 文本效果（Text Effects）：
 - 文本阴影（`text-shadow`），可以在文字后面添加阴影效果。
 - 文本溢出（`text-overflow`），控制文本溢出容器时的显示方式。
- 转换和动画（CSS Transforms Module, CSS Animations）：
 - 2D 和 3D 转换（`transform`），包括旋转（`rotate`）、缩放（`scale`）、倾斜（`skew`）和平移（`translate`）。
 - CSS 动画（`animation`），允许定义关键帧动画，控制动画序列。
- 等等

05-物理像素-逻辑像素-CSS像素-像素密度-DPR-PPI-DPI

当我们聊pixel时，到底在聊些什么？

- 像素是影响显示的基本单位。（比如屏幕上看到的画面、一幅图片）；
- pix是英语单词picture的常用简写，加上英语单词“元素”element，就得到pixel；
- “像素”表示“画像素”之意，有时亦被称为pel（picture element）；



但是我们又听说过各种像素的名称，物理像素、逻辑像素、CSS像素，它们分别是什么？又有什么关系呢？

我们这里先区分物理像素和逻辑像素

物理像素（Physical Pixel）也称为设备像素，是显示屏幕的最小物理单位。

- 每个物理像素可以发光并显示特定的颜色。
- 物理像素的大小是固定的，由设备的硬件决定。
- 比如iPhone X的分辨率 1125x2436，指的就是物理像素；

物理像素的密度（像素每英寸，即PPI，英语：Pixels Per Inch，缩写：PPI）PPI越高，屏幕显示的内容就越细腻。

- 1英寸=2.54厘米，在工业领域被广泛应用；

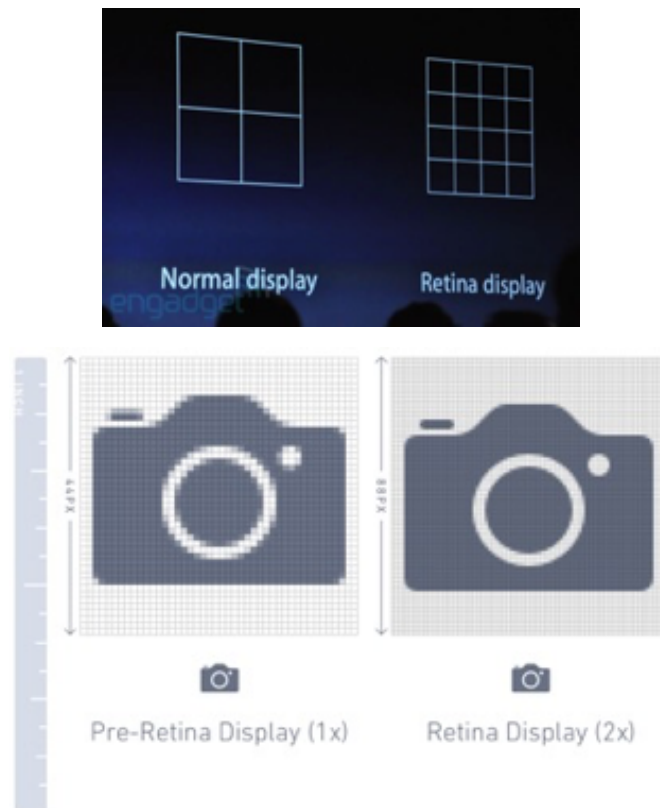
逻辑像素（Logical Pixel），有时也被称为设备独立像素（Device Independent Pixel，简称DIP）

- 是一个抽象的单位，用于在编程中统一不同设备的显示标准。
- 逻辑像素是用来衡量在不同设备上如何统一显示内容的尺寸单位。
- 例如，在高分辨率设备上，可能有多个物理像素组成一个逻辑像素。

- 这样，无论设备的物理像素密度如何，使用逻辑像素单位开发的界面都能保持相对一致的大小和视觉效果。

DPR: device pixel ratio

- 2010年，iPhone4问世，不仅仅带来了移动互联网，还带来了Retina屏幕；
- Retina屏幕翻译为视网膜显示屏，可以为用户带来更好的显示；
- 在Retina屏幕中，一个逻辑像素在长度上对应两个物理像素，这个比例称之为设备像素比（device pixel ratio）；
- 我们可以通过window.devicePixelRatio获取到当前屏幕上的DPR值；



CSS像素（CSS Pixel），CSS像素可以被看作是逻辑像素的一种形式，用在Web端的。

- 它们被设计用来简化Web开发者的工作，使网页在不同显示设备上都能保持设计的一致性。
- 随着设备屏幕密度的增加，浏览器会自动处理CSS像素与物理像素之间的比例关系，确保网页元素在视觉上的大小保持一致。

DPI（Dots Per Inch）:每英寸的打印点数

- DPI主要用于描述打印机输出的精细度。
- 例如，一个高DPI值的打印机可以打印出更细致、更少见瑕疵的图像。

PPI和DPI的区别是什么？

- DPI主要用于打印领域，而PPI则主要用于屏幕显示领域。
- DPI衡量的是墨水点的数量，PPI衡量的是像素的数量。

面试如何回答呢？

- 物理像素也称为设备像素，是显示屏幕的最小物理单位，也就是说它是实际的物理存在的单位，是由设备的硬件决定。
- 但是我们知道现在的显示器也好，手机也好，它们的分辨率也就是物理像素差别非常大，我们开发者如果面向物理像素开发就需要先考虑每个设备的真实分辨率，开发的难度就会大大提升。
- 所以操作系统和浏览器就抽象出另外一种像素，我们称之为逻辑像素，也被称之为设备独立像素。
- 逻辑像素是一个抽象的单位，用于在编程中统一不同设备的显示标准。这样，无论设备的物理像素密度如何，使用逻辑像素单位开发的界面都能保持相对一致的大小和视觉效果。
- 当然这个过程中还衍生出来很多不同的概念，比如PPI（物理像素的密度，每英寸的物理像素数量），DPR（设备像素比，也就是一个逻辑像素对应的物理像素数量），DPI（每英寸打印点数，它主要应用于打印领域）。

06-为什么在移动端使用@2x、@3x的图片？

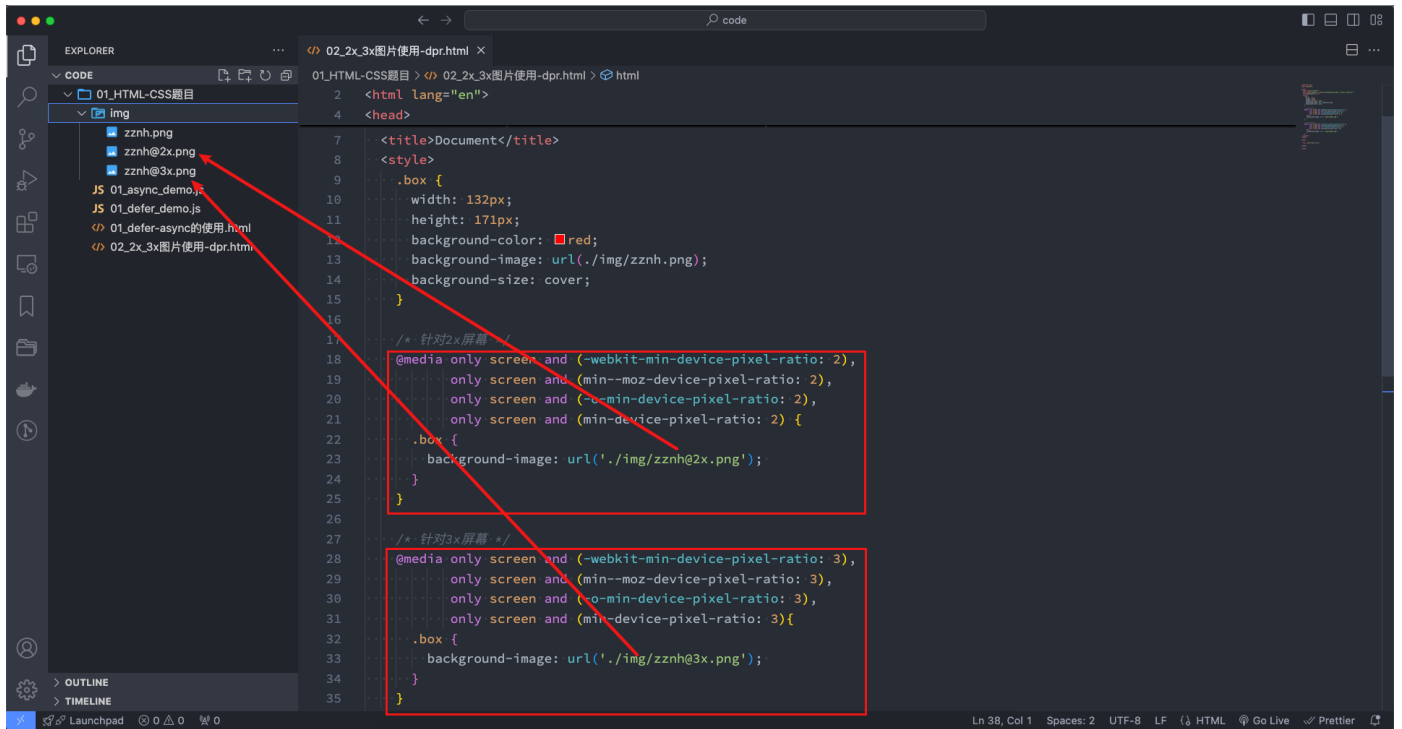
目前在移动端设备中，有非常多高分辨率的设备。为了适应不同的像素密度，UI设计师通常需要为开发者提供多个版本的图像资源。

通常标记为@1x、@2x、@3x：

- **@1x图像**：基本尺寸，适用于低分辨率设备。
- **@2x图像**：是基本图像尺寸的两倍，适用于中等分辨率设备，device-pixel-ratio为2的设备。
- **@3x图像**：是基本图像尺寸的三倍，适用于高分辨率设备，device-pixel-ratio为3的设备。

如果都使用的@1x的图片，在高分辨率下就会图像非常模糊，模糊的图像可能会使产品显得粗糙，影响用户对应用品质的整体感觉。

我们开发Web可以通过媒体查询来设置不同的图像：



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8   <style>
9     .box {
10       width: 132px;
11       height: 171px;
12       background-color: red;
13       background-image: url('./img/zznzh.png');
14       background-size: cover;
15     }
16
17     /* 针对2x屏幕 */
18     @media only screen and (min-resolution: 2dppx) {
19       .box {
20         background-image: url('./img/zznzh@2x.png');
21       }
22     }
23
24     /* 针对3x屏幕 */
25     @media only screen and (min-resolution: 3dppx) {
26       .box {
27         background-image: url('./img/zznzh@3x.png');
28       }
29     }
30 </style>
```

```
31 </head>
32
33 <body>
34
35   <div class="box"></div>
36
37 </body>
38
39 </html>
```


但是其实你在MDN上面查看会发现-webkit-min-device-pixel-ratio其实是一个非标准的特性，也就意味着不建议在生产环境使用：

 mdn web docs

ReferencesGuidesPlusCurriculumBlogTools NEW

Theme

Web 开发技术 > CSS: 层叠样式表 > @media > -webkit-device-pixel-ratio

 此页面由社区从英文翻译而来。了解更多并加入 MDN Web Docs 社区。

Filter

CSS教程CSS 基础CSS 第一步

-webkit-device-pixel-ratio

 非标准: 该特性是非标准的，请尽量不要在生产环境中使用它！

`-webkit-device-pixel-ratio` 是一个非标准的布尔类型 CSS 媒体类型，是标准 [resolution](#) (英语) 媒体类型的一个替代方案。

它推荐我们使用另外一个特性：resolution

- `resolution` 媒体特性是CSS标准中用于查询设备显示密度的推荐方式。
- 它支持多种单位，包括 `dpi`（dots per inch，每英寸点数）、`dpcm`（dots per centimeter，每厘米点数）、和 `dppx`（dots per pixel unit，每像素点数单位，相当于设备像素比）。

使用dppx即可：1dppx 相当于一个设备独立像素对应于一个屏幕物理像素。

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8   <style>
```

```

9      .box {
10         width: 132px;
11         height: 171px;
12         background-color: red;
13         background-image: url(./img/zznh.png);
14         background-size: cover;
15     }
16
17     /* 针对2x屏幕 */
18     @media only screen and (min-resolution: 2dppx) {
19         .box {
20             background-image: url('./img/zznh@2x.png');
21         }
22     }
23
24     /* 针对3x屏幕 */
25     @media only screen and (min-resolution: 3dppx) {
26         .box {
27             background-image: url('./img/zznh@3x.png');
28         }
29     }
30 </style>
31 </head>
32
33 <body>
34
35     <div class="box"></div>
36
37 </body>
38
39 </html>

```

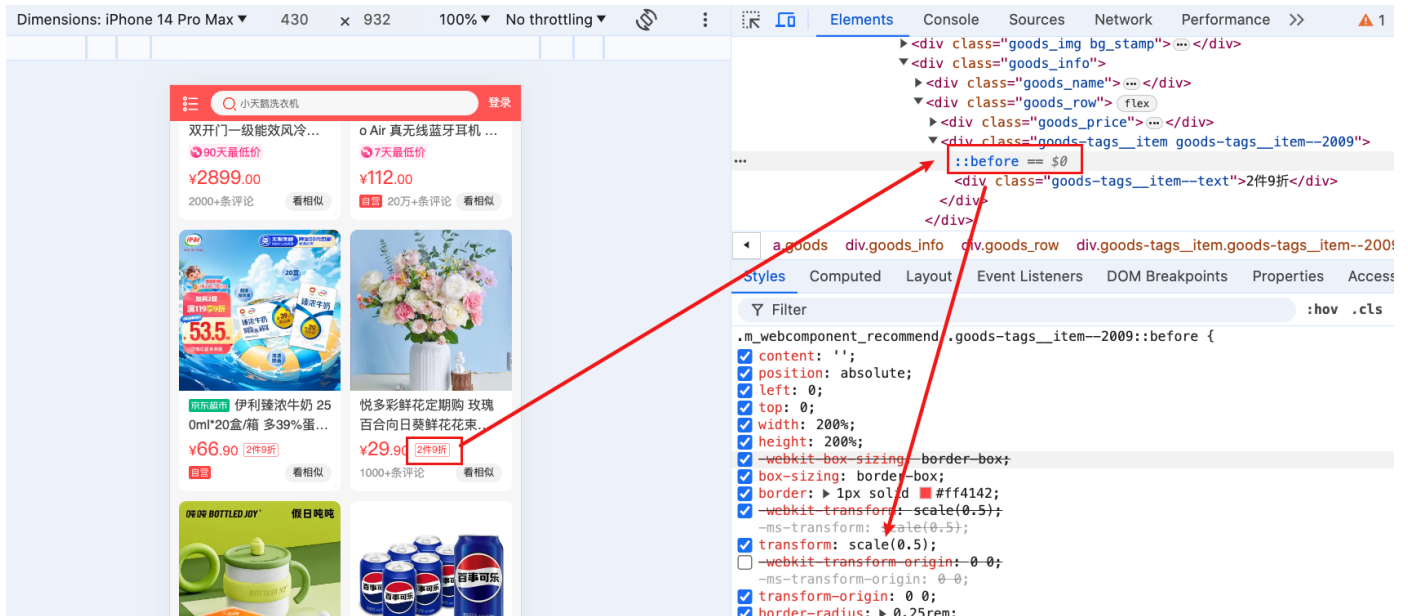
07-什么是1px问题，前端如何去解决它，如何画出0.5px边框？

我们知道在移动端的设计稿中，往往UI给的设计稿宽度为 750px，图中设计的边框宽度为 1px，在我们 375px 的设备下，我们应该将宽度写为 0.5px。

但是如果直接设置0.5的话，一些设备（特别是旧的移动设备和浏览器）并且不支持0.5px，这个就是我们常说的 1px问题以及如何画出0.5px边框的问题。

那么这种问题应该如何去处理呢？目前常见的方案有两种：

- 方案一：viewport + rem + div（淘宝，大家可以自行了解）
- 方案二：伪类 + transform（京东）



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8   <style>
9     .border-test {
10       position: relative;
11       padding: 10px;
12       margin: 20px;
13       display: inline-block;
14     }
15
16     .border-test::before {
17       content: "";
18       position: absolute;
19       left: 0;
20       top: 0;
21       width: 200%;
22       height: 200%;
23       border: 1px solid red;
24       transform-origin: 0 0;
25       transform: scale(0.5);
26     }
27   </style>
28 </head>
29
30 <body>
31   <div class="border-test">1px border</div>
```

```
32 </body>
33
34 </html>
```

08-你如何理解块级上下文（BFC），并且创建BFC的方法有哪些？

什么是FC呢？



FC的全称是Formatting Context，元素在标准流里面都是属于一个FC的；

Boxes in the normal flow belong to a formatting context, which may be block or inline, but not both simultaneously. [Block-level](#) boxes participate in a [block formatting](#) context. [Inline-level boxes](#) participate in an [inline formatting](#) context.

块级元素的布局属于Block Formatting Context（BFC）

- 也就是block level box都是在BFC中布局的；

行内级元素的布局属于Inline Formatting Context（IFC）

- 而inline level box都是在IFC中布局的；

block level box都是在BFC中布局的，那么这个BFC在哪里呢？拿出来给我看看。

9.4.1 Block formatting contexts

Floats, absolutely positioned elements, block containers (such as inline-blocks, table-cells, and table-captions) that are not block boxes, and block boxes with 'overflow' other than 'visible' (except when that value has been propagated to the viewport) establish new block formatting contexts for their contents.

MDN上有整理出在哪些具体的情况下会创建BFC：

- 根元素（`<html>`）
- 浮动元素（元素的 `float` 不是 `none`）
- 绝对定位元素（元素的 `position` 为 `absolute` 或 `fixed`）
- 行内块元素（元素的 `display` 为 `inline-block`）
- 表格单元格（元素的 `display` 为 `table-cell`，HTML表格单元格默认为该值），表格标题（元素的 `display` 为 `table-caption`）

table-caption, HTML表格标题默认为该值)

- 匿名表格单元格元素 (元素的 display 为 table、table-row、table-row-group、table-header-group、table-footer-group (分别是HTML table、row、tbody、thead、tfoot 的默认属性) 或 inline-table)
- overflow 计算值(Computed)不为 visible 的块元素
- 弹性元素 (display 为 flex 或 inline-flex 元素的直接子元素)
- 网格元素 (display 为 grid 或 inline-grid 元素的直接子元素)
- display 值为 flow-root 的元素

我们来看一下官方文档对BFC作用的描述:

In a block formatting context, boxes are laid out one after the other, vertically, beginning at the top of a containing block. The vertical distance between two sibling boxes is determined by the 'margin' properties. Vertical margins between adjacent block-level boxes in a block formatting context [collapse](#).

In a block formatting context, each box's left outer edge touches the left edge of the containing block (for right-to-left formatting, right edges touch). This is true even in the presence of floats (although a box's *line boxes* may shrink due to the floats), unless the box establishes a new block formatting context (in which case the box itself [may become narrower](#) due to the floats).

简单概况如下:

- 在BFC中, box会在垂直方向上一个挨着一个的排布;
- 垂直方向的间距由margin属性决定;
- 在同一个BFC中, 相邻两个box之间的margin会折叠 (collapse) ;
- 在BFC中, 每个元素的左边缘是紧挨着包含块的左边缘的;

问题一面试回答:

- 针对BFC网上有特别多的说法, 但是都没有解释的很清楚, 所以我是专门查过MDN文档的
- 在标准流中, 我们所有的盒子, 不管是块级盒子还是行内盒子, 它们都属于某一个FC (格式化上下文), 块级盒子属于BFC (块级格式化上下文), 行内级元素属于IFC (行内格式化上下文)。
- BFC就是用来决定块级盒子是如何排布的
- 在BFC中, 块级盒子是一个挨着一个在垂直方向排布的, 垂直方向的间距由margin属性决定, 并且在同一个BFC中, 两个相邻的box之间margin会折叠。
- 这个就是BFC的官方解释。

问题二面试回答:

- 创建BFC的方式非常多, 要根据不同的场景使用不同的方式。
- 比如HTML元素本身就是创建了一个BFC, 浮动元素、绝对定位元素、inline-block、表格元素以及很多子元素都会创建BFC。
- 在开发中我们想要在很多情况下不影响布局创建BFC, 比较常用的是将overflow设置为非visible, 比如auto就可以创建BFC。

09-开发中会使用BFC解决哪些问题? 它是如何解决的?

BFC在开发中主要解决两个问题:

- 解决margin的折叠问题;
- 解决浮动高度塌陷问题;

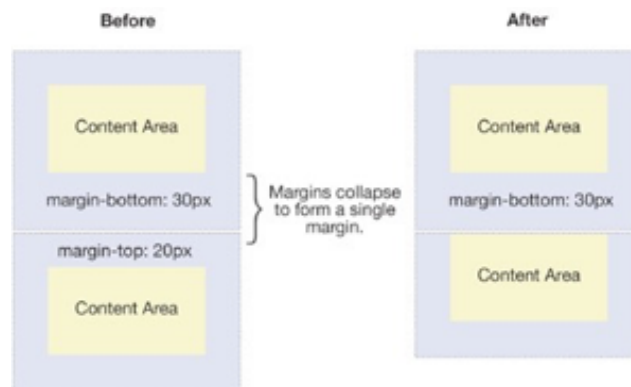
作用一：解决margin的折叠问题

从上面的内容我们可以知道，BFC可以解决BFC的margin折叠的问题，如果两个块级盒子不处于同一个BFC中，那么它们的margin就不会折叠了。

官方文档明确的有说：

- The vertical distance between two sibling boxes is determined by the 'margin' properties. Vertical margins between adjacent block-level boxes in a block formatting context collapse.

那么如果我们让两个box是不同的BFC呢？那么就可以解决折叠问题。



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8   <style>
9     .box1,
10    .box2 {
11      height: 200px;
12    }
13
14    .container {
15      overflow: auto;
16    }
17
18    .box1 {
19      background-color: #789;
20      margin-bottom: 30px;
21    }
22
23    .box2 {
24      margin-top: 60px;
25      background-color: #18f;
26    }
27  </style>
```



```
28 </head>
29
30 <body>
31
32   <div class="container">
33     <div class="box1"></div>
34   </div>
35   <div class="box2"></div>
36 </body>
37
38 </html>
```

作用二：解决浮动高度塌陷问题

网上有很多说法，BFC可以解决浮动高度塌陷，可以实现清除浮动的效果。

- 但是从来没有给出过BFC可以解决高度塌陷的原理或者权威的文档说明；
- 他们也压根没有办法解释，为什么可以解决浮动高度的塌陷问题，但是不能解决绝对定位元素的高度塌陷问题呢？

事实上，BFC解决高度塌陷需要满足两个条件：

- 浮动元素的父元素触发BFC，形成独立的块级格式化上下文（Block Formatting Context）；
- 浮动元素的父元素的高度是auto的；

10.6.7 'Auto' heights for block formatting context roots

In certain cases (see, e.g., sections [10.6.4](#) and [10.6.6](#) above), the height of an element that establishes a block formatting context is computed as follows:

If it only has inline-level children, the height is the distance between the top of the topmost line box and the bottom of the bottommost line box.

If it has block-level children, the height is the distance between the top margin-edge of the topmost block-level child box and the bottom margin-edge of the bottommost block-level child box.

Absolutely positioned children are ignored, and relatively positioned boxes are considered without their offset. Note that the child box may be an [anonymous block box](#).

In addition, if the element has any floating descendants whose bottom margin edge is below the element's bottom content edge, then the height is increased to include those edges. Only floats that participate in this block formatting context are taken into account, e.g., floats inside absolutely positioned descendants or other floats are not.

BFC的高度是auto的情况下，是如下方法计算高度的

- 1.如果只有inline-level，是行高的顶部和底部的距离；
- 2.如果有block-level，是由最顶层的块上边缘和最底层块盒子的下边缘之间的距离
- 3.如果有绝对定位元素，将被忽略；
- 4.如果有浮动元素，那么会增加高度以包括这些浮动元素的下边缘

那么也就意味着我们如果让某一个高度为auto的元素，变成一个BFC，那么它会增加高度包裹浮动元素的下边缘。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7   <style>
8     .container {
9       background-color: #17f;
10      overflow: auto;
11    }
12    .item {
13      width: 300px;
14      height: 200px;
15      background-color: #f71;
16      float: left;
17      border: 1px solid slateblue;
18    }
19  </style>
20 </head>
21 <body>
22
23   <div class="container">
24     <div class="item">item1</div>
25     <div class="item">item2</div>
26     <div class="item">item3</div>
27   </div>
28
29 </body>
30 </html>
```

面试问题回答：

- BFC在开发中主要可以用来解决两个问题：解决margin的折叠问题和解决浮动元素高度塌陷问题。
- 解决margin的折叠问题是通过让两个垂直方向的盒子处于不同的BFC中，因为在同一个BFC会折叠，那么处于不同的BFC时就不会折叠了
- 解决浮动元素高度塌陷问题是源于一个盒子是BFC时，它高度的计算规则，当一个盒子是BFC并且高度设置为auto时，它会在计算高度时要求增加高度以包裹浮动元素的下边缘，那么为了包裹浮动元素的下边缘增加了高度，就不会出现高度塌陷的问题了。

10-通常会采取哪些措施来确保网站或者应用在不同的浏览器上的兼容性？

我认为可以从这几个角度来回答。

其实在现代工程化的开发架构下，大多数的浏览器兼容性问题是可以通过工程化中的配置选项来解决的。

- 1.比如browserslist可以配置目标的浏览器或者Node环境，然后在不同的工具中起作用，比如autoprefixer/babel/postcss-preset-env等，在进行了正确的配置后，开发的Vue或者React项目在进行打包时，会自动根据目标环境来添加CSS前缀、Babel代码转换等。
- 2.如果我们想要额外的适配，通常在项目中我们还会引入normal.css和polyfills来添加特定的CSS、JS的适配问题
- 3.还有一些事针对移动端的，比如移动端点击300ms的延迟、移动端1px边框的问题，都可以在特定的环境或者需求下来解决
- 4.当遇到问题时，很重要的事我们需要多查询caniuse的网站来确定某些特性的兼容性
- 5.另外如果针对特定的用户使用的事不同的浏览器和设备时，我们需要使用特定的工具，比如BrowserStack这样的工具来进行测试，遇到特定问题时，及时的解决和处理。

举一个具体的例子：

- 比如之前我们在开发中借助于transform实现动画效果，使用的是复合属性，transform: translate(10px, 20px) scale(1.5);。
- 但是这种复合属性在IE11上是有问题，因为它并不支持，所以我们就必须对它拆分属性，首先设置translate，在它的外层再包裹一个容器，用来设置scale属性。
- 如果还是不能解决，也可以通过JavaScript代码来处理，可以更加精准的根据不同的浏览器来调整CSS的动画效果。