01-HTML-CSS-常见面试题

1.HTML 标签有哪些行内元素

- a
- img
- picture
- span
- input
- textarea
- select
- label

2.说说你对元素语义化的理解

元素语义化就是用正确的元素做正确的事情。虽然在理论上，所以的html元素都可以通过css样式实现相同的事情，但是这么做会使事情复杂化，所以我们需要元素语义化来降低复杂度。

元素语义化在我们实际的开发中有很多好处，比如：

- 提高代码的阅读性和可维护性;
- 减少coder之间的沟通成本;
- 能让语音合成工具正确识别网页元素的用途，以便做出正确的反应
- 有利于SEO(Search Engine Optimization)

3.HTML 中有哪些语义化标签

- header
- footer
- main
- aside
- article
- section
- address
- summary/details
- menu
- h1/h2/h3/h4/h5/h6
- img
- p
- strong/italic

4.什么是 URL 编码 (URL Encode)

encodeURI 用来编码**URI**，其不会编码保留字符。

encodeURIComponent 用来编码 URI**参数**，除了字符：A-Z a-z 0-9 - _ . ! ~ * ' ( )，都将会转义。

5.说说你对SEO的理解

SEO就是搜索引擎优化(Search Engine Optimization)，SEO通过了解搜索引擎的运行规则来调整网站，以提高网站的曝光度,以及网站的排名。

Google 搜索引擎的工作流程主要分为三个阶段：

抓取：Google 会使用名为“抓取工具”的自动程序搜索网络，以查找新网页或更新后的网页。Google 会将这些网页的地址（即网址）存储在一个大型列表中，以便日后查看。我们会通过许多不同的方法查找网页，但主要方法是跟踪我们已知的网页中的链接。

编入索引：Google 会访问它通过抓取得知的网页，并会尝试分析每个网页的主题。Google 会分析网页中的内容、图片和视频文件，尝试了解网页的主题。这些信息存储在 Google 索引中，而 Google 索引是一个存储在海量计算机中的巨大数据库。

呈现搜索结果：当用户在 Google 上进行搜索时，Google 会尝试确定最优质的搜索结果。“最佳”结果取决于许多因素，包括用户的位置、语言、设备（桌面设备或手机）以及先前用过的搜索查询。例如，在用户搜索“自行车维修店”后，Google 向巴黎用户显示的答案与向香港用户显示的答案有所不同。支付费用不能提高网页在 Google 搜索结果中的排名，网页排名是完全依靠算法完成的。

6.'+' 与 '~' 选择器有什么不同

`~` 是匹配元素之后的选择器

`+` 是匹配相邻元素选择器

```
<template>
    <div>我是div</div>
    <p>我是p</p>
    <p>我是p</p>
    <div>我是div</div>
    <p>我是p</p>
    <div>
      <p>我是div下面的p</p>
      <p>我是div下面的p</p>
    </div>
    <span>我是span</span>
</template>
<style>
    div+p {
        color: red;
    }
    /* 第一个p标签变红色了 */
```

```
    div~p{
        color:red;
    }
    /* div后面的p标签都变成红色了 */
</style>
```

7.说明text-align居中的条件

- `text-align`: 直接翻译过来设置文本的水平对齐方式 (是继承属性)(是继承属性)
- `text-align` 并不控制块元素自己的对齐，只控制它的行内内容的对齐
- MDN解释: 定义行内内容（例如文字）如何相对它的块父元素对齐(可以设置图片居中)
- W3C官方文档解释: 设置行内(inline-level)元素(没有填满父元素)在快级父元素的对齐方式

8. line-height为什么可以让文字垂直居中?

- line-height :两行文字基线(baseline)之间的间距 基线(baseline `):与小写字母x最底部对齐的线
- 一行文本 等于 line-height
- 行高 - 文本高度 = 行距
- 属性值:
  - `normal`:取决于用户端。桌面浏览器（包括Firefox）使用默认值，约为1.2，这取决于元素的`font-family`
  - `<数字>`:该属性的应用值是这个无单位数字<数字>乘以该元素的字体大小 这是设置`line-height`的推荐方法，不会在继承时产生不确定的结果
  - `<长度>`:指定<长度>用于计算 line box 的高度 以 em 为单位的值可能会产生不确定的结果
  - `<百分比>`:与元素自身的字体大小有关。计算值是给定的百分比值乘以元素计算出的字体大小。百分比值可能会带来不确定的结果
- `height`:元素的整体高度 `line-height`:元素中每一行文字所占据的高度
- 假设div中只有一行文字，如何让这行文字在div内部垂直居中 让`line-height`等同于`height`

9.说说盒子模型包含哪些内容?

- 内容
  - 通过宽度和高度设置
- 内边距
  - 通过padding设置
  - padding: padding-top padding-right padding-bottom padding-left;
- 边框
  - 通过border设置

    - border: border-width border-style border-color
- 外边距
    - 通过margin设置
    - margin: margin-top margin-right margin-bottom margin-left

10.说说你对margin的传递和折叠的理解

margin的传递一般是父子块元素之间,有margin-top传递,margin-bottom传递.

- margin-top传递: 当块级元素的顶部线和父元素的顶部线重叠，那么这个块级元素的margin-top值会传递给父元素
- margin-bottom传递:当块级元素的底部线和父元素的底部线重叠，那么这个块级元素的margin-bottom值会传递给父元素

折叠: 指的是 垂直方向上相邻的2个margin（margin-top、margin-bottom）有可能会合并为1个margin.

它有两个兄弟块级元素之间的上下margin的折叠,也有父子块元素之间的margin折叠

11.CSS 隐藏页面中某个元素的几种方法

- display: none
    - 通过 CSS 操控 display，移出文档流
- opacity: 0
    - 透明度为 0，仍在文档流中，当作用于其上的事件(如点击)仍有效
- visibility: hidden
    - 透明度为 0，仍在文档流中，**但作用于其上的事件(如点击)无效**，这也是 `visibility:hidden` 与 `opacity: 0` 的区别
- content-visibility
    - 移出文档流，但是再次显示时消耗性能低
- 绝对定位于当前页面的不可见位置

```
position: absolute;
top: -9000px;
left: -9000px;
```

12.box-sizing有什么作用？content-box和border-box的区别

box-sizing用来设置盒子模型中宽高的计算方式：

- content-box: padding、border都布置在width、height外边
- border-box: padding、border都布置在width、height里边

13.为什么会发生样式抖动

- 因为没有指定元素具体高度和宽度,比如数据还没有加载进来时元素高度是 100px(假设这里是100px)
- 数据加载进来后,因为有了数据,然后元素被撑大,所有出现了抖动

14.说说浮动常见的规则?

- 元素一旦浮动后, 脱离标准流
  - 朝着向左或向右方向移动，直到自己的边界紧贴着包含块（一般是父元素）或者其他浮动元素的边界为止
  - 定位元素会层叠在浮动元素上面
- 如果元素是向左（右）浮动，浮动元素的左（右）边界不能超出包含块的左（右）边界
- 浮动元素之间不能层叠
  - 如果一个元素浮动，另一个浮动元素已经在那个位置了，后浮动的元素将紧贴着前一个浮动元素（左浮找左浮，右浮找右浮）
  - 如果水平方向剩余的空间不够显示浮动元素，浮动元素将向下移动，直到有充足的空间为止
- 浮动元素不能与行内级内容层叠，行内级内容将会被浮动元素推出
  - 比如行内级元素、inline-block元素、块级元素的文字内容
  - 行内级元素、inline-block元素浮动后，其顶部将与所在行的顶部对齐

15.为什么需要清除浮动？清除浮动有几种方法?

为什么需要清除浮动：

1）由于浮动元素脱离了标准流，变成了浮动元素，不再向父元素汇报高度。所以父元素在计算高度时并没有将浮动元素的高度计算进来，因此就造成了高度塌陷的问题 。解决高度塌陷的问题就叫做清除浮动（3分）

2）清除浮动的目的：是为了让父元素在计算高度的时候把浮动子元素的高度计算进去

清除浮动有几种方法：

- 给父元素设置固定高度，扩展性不好，不推荐
- 在父元素的最后增加一个空的块级子元素，并设置让他clear:both , 但是增加了无意义的空标签，违反了结构与样式分离的原则
- 给父元素添加一个伪元素(推荐)

```
.clear_fix::after {
    content: "";
    display: block;
    clear:both;
    visibility: hidden; /* 浏览器兼容性 */
    height: 0; /* 浏览器兼容性 */

}
.clear_fix {
    *zoom: 1; /* IE6/7兼容性 */
}
```

- overflow:auto触发BFC来清除浮动（前提高度为auto）

16.伪类与伪元素有什么区别?

- 伪类使用单冒号，而伪元素使用双冒号。如 `:hover` 是伪类， `::before` 是伪元素
- 伪元素会在文档流生成一个新的元素，并且可以使用 `content` 属性设置内容

17.结构伪类nth-child(n)和nth-of-type(n)的区别?

:nth-child

- 是结构伪类选择器，选中父元素的第几个子元素，计数时与元素的类型无关。

:nth-of-type

- 是结构伪类选择器和nth-child类似，但是计数时只计算同种类型的元素。

18.元素或文本水平居中实现方案有哪几种?

1) text-align: center

- 指定该属性的元素可以让其内部：行内元素，行内块级元素和文本水平居中。

2) margin: 0 auto

- 该属性可以让具有宽度的块级元素水平居中。

3) 定位

position: relative;

left:50%

transfrom:translate(-50%,0); (行内元素无效)

或者

width: 200px

height:200px

position: absolute;（需要设置宽）

left: 0;

right: 0;

margin: 0 auto;

或者

position: absolute;

left:50%

width: 200px;

margin-left: -100px（需要居中的元素使用）

3) flex布局

display: flex;（1分）

justify-content: center (flex item居中)

19.元素或文本垂直居中实现方案有哪几种?

1）line-height（2分）

- 可以让块级和行内元素（行内非替换元素，行内替换元素，行内块级元素）的文本垂直居中。

2）定位

position: relative;

top:50%

transfrom:translate(0,-50%)(行内元素无效)

或者

width: 200px

height:200px

position: absolute;（需要设置高）

left: 0;

right: 0;

margin: 0 auto;

或者

position: absolute;

top:50%

height: 200px;

margin-top: -100px （需要居中的元素使用）

3）flex布局实现

display: flex;

align-items: center (flex item居中)

20.rem、em、vw、vh 单位是什么意思?

- `rem`: 单位是根据根元素(即 `html`)的 `font-size` 大小来计算
- `em`: 单位是根据**自身元素**的 `font-size` 大小来计算
- `vw`: viewport width，即视口的宽
- `vh`: viewport height，即视口的高

21.什么是视口(viewport)?

- pc端的视口
  - 就是浏览器的可视区域
- 移动端视口
  - 布局视口
    - 会按照一个默认宽度980px,来布局一个页面盒子的内容
    - 为了可以显示完整的页面,会对整个页面进行缩小

```
<!-- width: 设置布局视口的宽度 -->
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

  - 视觉视口
    - 显示在可视区域的视口,就是视觉视口
  - 理想视口
    - 当布局视口 = 视觉视口的时候,就是理想视口
    - 怎样是这理想视口呢?

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

22.移动端适配的方案有哪些?

1）百分比布局

- 因为不同属性百分比的值，相对的可能是不同的参照物，所以百分比往很难统一。

2）视口 + (rem + 动态HTML的 font-size)

- 动态计算 HTML font-zise：
  - 用媒体查询来修改HTML font-size(缺点不能实时改变font-size的大小)
  - 自己编写JS来实现修改HTML font-size的大小(可以实时修改字体大小)
  - 是引用lib-flexiable库来实现（原理是JS动态改HTML font-size大小）
- px 转成rem：
  - 手动计算 rem
  - Less的映射来计算
  - postcss-pxtorem插件来实现（需依赖webpack或Vite）
  - cssrem VSCode插件来实现

3）视口 + vw

- px转成rem
  - 手动计算vw
  - Less的映射来计算
  - postcss-px-to-viewport的插件（需依赖webpack或Vite）
  - ccsrem VSCode插件

4）flex弹性布局

- flex container 属性
- flex item 属性

23.什么是Flexible Box布局?

- **CSS Flexible Box布局**俗称**Flexbox**，是一种一维布局模型，它具有灵活高效的布局，在项目之间分配空间来控制它们的对齐结构，即，它是一种布局模型，提供了一种简单而干净的方式来排列项目。
- CSS Flexible Box布局允许容器内的响应式元素根据屏幕或设备大小自动排列。
- Flex 布局有两根轴线，分别为主轴和交叉轴。
- 主轴由 `flex-direction` 定义，另一根轴垂直于它。

24.flex布局container 和 item的属性分别有哪些？以及其作用?

- display: flex
- flex container
  - flex-direction 决定主轴方向,默认值为row。
  - flex-wrap 决定flex container是单行还是多行,默认值为nowrap。
  - flex-flow 前面两个属性的组合写法,默认值为row。
  - justify-content 决定flex items在主轴的对齐方式,默认值为flex-start。
  - align-items 决定flex items在交叉轴的对齐方式,默认值为normal。

    - align-content 决定多行flex items在交叉轴的对齐方式,默认值为stretch。
- flex item
    - flex-grow: 决定flex items如何拉伸,默认值为0。
    - flex-shrink: 决定flex items如何收缩,默认值为1。
    - flex-basis:决定flex items的基本size大小, 默认值为auto。
    - flex: 是前面3个属性的组合写法。
    - align-self:指定flex item自身的排序方式,默认值为auto。
    - order:决定了flex items的排布顺序，越小越靠前,默认值为0。

25.常见的CSS Transform形变有哪些?

transform属性允许对某一个元素进行某些形变, 包括旋转，缩放，倾斜或平移等。transform对于行内级非替换元素（如a，span）是无效的。

- translate(x, y) :平移，用于移动元素在平面上的位置
- scale(x, y) :缩放，可改变元素的大小。
- rotate(deg) :旋转，表示旋转的角度 。
- skew(deg, deg) :倾斜，定义了一个元素在二维平面上的倾斜转换

26.说出CSS Transition和Animation动画的区别

- transition:
    - 只能定义两个状态：开始状态和结束状态，不能定义中间状态
    - 不能重复执行动画，除非一再触发动画
    - 需要在特定状态触发后才能执行，比如某属性修改了
- animation:
    - 可以用@keyframes定义动画序列（每一帧如何执行）
    - 通过设置animation-iteration-count来规定动画执行的次数
    - 不需要触发特定状态即可执行
- animation动画比transition多了:
    - animation-iteration-count， animation-direction， animation-fill-mode 和 animation-play-state属性

27.css 动画与 js 动画哪个性能更好

CSS3 的动画:

- 在性能上会稍微好一些，浏览器会对 CSS3 的动画做一些优化（比如专门新建一个图层用来跑动画）
- 代码相对简单，但在动画控制上不够灵活
- 兼容性不好
- 部分动画功能无法实现（如滚动动画，视差滚动等）

JavaScript 的动画：

- 正好弥补了css缺点，控制能力很强，可以单帧的控制、变换，同时写得好完全可以兼容 IE6，并且功能强大。

总结：

- 对于一些复杂控制的动画，使用 javascript 会比较好。而在实现一些小的交互动效的时候，可以多考虑 CSS

28.说说vertical-align的作用以及应用场景?

vertical-align影响行内级元素在一个行盒中垂直方向的位置，默认值为baseline对齐

- baseline(默认值)：基线对齐
- top：把行内级盒子的顶部跟line boxes顶部对齐
- middle：行内级盒子的中心点与父盒基线加上x-height一半的线对齐
- bottom：把行内级盒子的底部跟line box底部对齐

不同应用情景分析：

- 只有文字时：行盒包裹内容,文字的bottom-line和行盒底部对齐
- 有图片和文字时:图片的底部和文字的baseline对齐
- 有图片，有文字，有inline-block（比图片要大 : 图片的底部,行内块底部和文字的baseline对齐
- 有图片，有文字，有inline-block（比图片要大）而且设置了margin-bottom: 图片的底部,行内块margin-bottom底部和文字的baseline对齐
- 有图片、文字、 inline-block（比图片要大）而且设置了margin-bottom并且有文字 :文字的baseline和图片的底部,行内块内最后一行文字的baseline对齐

29.说说你对BFC的理解

- block format context(块级格式化上下文)
  - 是页面的一块渲染区域 并且有一套渲染规则,决定了子元素如何定位 以及与其他元素之间的排列 布局之间的关系
  - BFC是一个独立的布局环境 相当于是一个容器 在其中按照一定的规则对块级元素进行摆放 ,并且不会影响其他的布局环境中的盒子,如果一个元素触发BFC则BFC中的元素布局不受外界的影响
- 块级元素在标准流中的布局是属于BFC的
- 创建BFC的条件:
  - 根元素: body/:root
  - float left/right
  - position absolute/fixed
  - overflow: 除visible
  - display: inline-block/table-cell/table-caption ,flex/grid...
- 特点

    - 垂直方向 自上而下排布
    - 垂直方向的间距由margin决定
    - 同一个BFC中 盒子之间的margin会折叠
    - BFC中 每个元素的左边缘紧挨着包含快的左边缘
    - 计算 BFC 的高度时，需要计算浮动元素的高度
    - BFC内部不会影响外部元素
    - BFC区域不会与浮动的元素发生重叠
- 作用
    - 解决margin折叠的问题
    - 解决高度塌陷的问题
        - 前提 :浮动的父级BFC高度为auto
    - 创建两栏布局
        - 左边浮动 右边overflow:hidden

30.介绍下 BFC、IFC、GFC 和 FFC

BFC（Block formatting contexts）： 块级格式上下文

- 页面上的一个隔离的渲染区域，那么他是如何产生的呢？可以触发BFC的元素有float、position、overflow、display：table-cell/ inline-block/table-caption ； BFC有什么作用呢？比如说实现多栏布局’

IFC（Inline formatting contexts）： 内联格式上下文

- IFC的line box（线框）高度由其包含行内元素中最高的实际高度计算而来（不受到竖直方向的padding/margin影响)IFC中的line box一般左右都贴紧整个IFC，但是会因为float元素而扰乱。float元素会位于IFC与与line box之间，使得line box宽度缩短。 同个ifc下的多个line box高度会不同 IFC中时不可能有块级元素的，当插入块级元素时（如p中插入div）会产生两个匿名块与div分隔开，即产生两个IFC，每个IFC对外表现为块级元素，与div垂直排列。 那么IFC一般有什么用呢？水平居中：当一个块要在环境中水平居中时，设置其为inline-block则会在外层产生IFC，通过text-align则可以使其水平居中。 垂直居中：创建一个IFC，用其中一个元素撑开父元素的高度，然后设置其vertical-align:middle，其他行内元素则可以在此父元素下垂直居中。

GFC（GrideLayout formatting contexts）： 网格布局格式化上下文

- 当为一个元素设置display值为grid的时候，此元素将会获得一个独立的渲染区域，我们可以通过在网格容器（grid container）上定义网格定义行（grid definition rows）和网格定义列（grid definition columns）属性各在网格项目（grid item）上定义网格行（grid row）和网格列（grid columns）为每一个网格项目（grid item）定义位置和空间。那么GFC有什么用呢，和table又有什么区别呢？首先同样是一个二维的表格，但GridLayout会有更加丰富的属性来控制行列，控制对齐以及更为精细的渲染语义和控制。

FFC（Flex formatting contexts）:自适应格式上下文

- display值为flex或者inline-flex的元素将会生成自适应容器（flex container），可惜这个牛逼的属性只有谷歌和火狐支持，不过在移动端也足够了，至少safari和chrome还是OK的，毕竟这俩动端才是王道。Flex Box 由伸缩容器和伸缩项目组成。通过设置元素的 display 属性为 flex 或

inline-flex 可以得到一个伸缩容器。设置为 flex 的容器被渲染为一个块级元素，而设置为 inline-flex 的容器则渲染为一个行内元素。伸缩容器中的每一个子元素都是一个伸缩项目。伸缩项目可以是任意数量的。伸缩容器外和伸缩项目内的一切元素都不受影响。简单地说，Flexbox 定义了伸缩容器内伸缩项目该如何布局。

31.说出不同像素之间的差异?

分为三种像素:设备像素(物理像素),设备独立像素(逻辑像素),css像素

- 设备像素(物理像素)
  - 是指显示器上真实的像素,在购买显示器或者手机的时候，提到的设备**分辨率**就是设备像素的大小
  - iPhone X的分辨率 1125 x 2436，指的就是设备像素
- 设备独立像素(逻辑像素)
  - 如果面向开发者我们使用设备像素显示一个100px的宽度，那么在不同屏幕上显示效果会是不同的
  - 开发者针对不同的屏幕很难进行较好的适配，编写程序必须了解用户的分辨率来进行开发
  - 所以在设备像素之上，操作系统为开发者进行抽象，提供了逻辑像素的概念
  - 比如你购买了一台显示器，在操作系统上是以1920x1080设置的显示分辨率，那么无论你购买的是2k、4k的显示器，对于开发者来说，都是1920x1080的大小
  - 如果物理像素很大的时候,比如2k,4k等,可以理解为一个逻辑像素里面由多个物理像素来渲染的
- css像素
  - 默认情况下就是设备独立像素(也就是逻辑像素)

32.分析比较 opacity: 0、visibility: hidden、display: none 优劣和适用场景。

结构

- display:none: 会让元素完全从渲染树中消失，渲染的时候不占据任何空间, 不能点击， visibility: hidden:不会让元素从渲染树消失，渲染元素继续占据空间，只是内容不可见，不能点击 opacity: 0: 不会让元素从渲染树消失，渲染元素继续占据空间，只是内容不可见，可以点击

继承

- display: none和opacity: 0：是非继承属性，子孙节点消失由于元素从渲染树消失造成，通过修改子孙节点属性无法显示。 visibility: hidden：是继承属性，子孙节点消失由于继承了hidden，通过设置visibility: visible;可以让子孙节点显式。

性能

- displaynone : 修改元素会造成文档回流,读屏器不会读取display: none元素内容，性能消耗较大 visibility:hidden: 修改元素只会造成本元素的重绘,性能消耗较少读屏器读取visibility: hidden元素内容 opacity: 0： 修改元素会造成重绘，性能消耗较少

联系

- 它们都能让元素不可见

33.如何解决移动端 Retina 屏 1px 像素问题

在移动端 Web 开发中，UI 设计稿中设置边框为 1 像素，前端在开发过程中如果出现 `border:1px`，测试会发现在 Retina 屏机型中，1px 会比较粗，即是较经典的移动端 1px 像素问题。

以 iphone6 为例，iphone6 的屏幕宽度为 375px，设计师做的视觉稿一般是750px，也就是 2x，这个时候设计师在视觉稿上画了 1px 的边框，于是你就写了 `border:1px`，所以1px边框问题产生了。

对设计师来说它的 `1px` 是相对于 `750px` 的（物理像素），对你来说你的 `1px` 是相对于 `375px` 的（css像素），实际上你应该是 `border:0.5px`。下面来看一下解决方案：

0.5px 实现

```
.border-1px { border: 1px solid #999 }
@media screen and (-webkit-min-device-pixel-ratio: 2) {
    .border-1px { border: 0.5px solid #999 }
}
/* dpr=2 和 dpr=3 情况下 border 相差无几，下面代码可以省略*/
@media screen and (-webkit-min-device-pixel-ratio: 3) {
    .border-1px { border: 0.333333px solid #999 }
}
```

但在 `IOS7` 及以下和 `Android` 等其他系统里，`0.5px` 将会被显示为 `0px`。所以我们需要通过 JS 检测浏览器能否处理 `0.5px` 的边框

```
if (window.devicePixelRatio && devicePixelRatio >= 2) {
  var testElem = document.createElement('div');
  testElem.style.border = '.5px solid transparent';
  document.body.appendChild(testElem);
}
if (testElem.offsetHeight == 1) {
  document.querySelector('html').classList.add('hairlines');
}
  document.body.removeChild(testElem);
}
```

- 优点：简单，没有副作用
- 缺点：支持 iOS 8+，安卓待兼容

viewport + rem 实现

通过设置缩放，让 `CSS` 像素等于真正的物理像素。

```
const scale = 1 / window.devicePixelRatio;
const viewport = document.querySelector('meta[name="viewport"]');
if (!viewport) {
    viewport = document.createElement('meta');
    viewport.setAttribute('name', 'viewport');
    window.document.head.appendChild(viewport);
}

viewport.setAttribute('content', 'width=device-width,user-scalable=no,initial-scale=' + scale + ',maximum-scale=' + scale + ',minimum-scale=' + scale);

// 设置根字体大小
var docEl = document.documentElement;
var fontsize = 10 * (docEl.clientWidth / 320) + 'px';
docEl.style.fontSize = fontsize;

// 在CSS中用rem单位就行了
```

缺点：

- 通过 JS 对文档进行修改，所以性能上有一定影响
- 会对项目中所有使用 `rem` 单位的对象进行影响。如果是老项目，则会全部更改 css 样式（不适合老项目改造）

伪元素 + transform 实现（推荐）

为什么用伪元素？ 因为伪元素 `::after` 或 `::before` 是独立于当前元素，可以单独对其缩放而不影响元素本身的缩放

基于 `media` 查询判断不同的设备像素比对线条进行缩放：

```
.border-1px:before{
    content: '';
    position: absolute;
    top: 0;
    height: 1px;
    width: 100%;
    background-color: #999;
    transform-origin: 50% 0%;
}
@media only screen and (-webkit-min-device-pixel-ratio:2){
    .border-1px:before{
        transform: scaleY(0.5);
    }
}
@media only screen and (-webkit-min-device-pixel-ratio:3){
    .border-1px:before{
        transform: scaleY(0.33);
```

```
    }
}
```

- 注意：如果需要满足圆角，需要给伪类也加上 `border-radius`
- 优点：兼容性好，无副作用，推荐使用

svg 实现(推荐)

因为 `svg` 是矢量图形，它的 `1px` 对应的物理像素就是 `1px`

可以搭配 `PostCSS` 的 `postcss-write-svg` 使用：

```
@svg border-1px {
  height: 2px;
  @rect {
    fill: var(--color, black);
    width: 100%;
    height: 50%;
    }
  }
.svg {
    border: 1px solid transparent;
    border-image: svg(border_1px param(--color #00b1ff)) 2 2 stretch;
}
```

编译后：

```
.svg { border: 1px solid transparent; border-image:
url("data:image/svg+xml;charset=utf-8,%3Csvg xmlns='http://www.w3.org/2000/svg'
height='2px'%3E%3Crect fill='%2300b1ff' width='100%25'
height='50%25'/%3E%3C/svg%3E") 2 2 stretch; }
```

- 优点：实现简单，可以实现圆角，
- 缺点：需要学习 `svg` 语法

34.如何用 css 实现多行文本溢出省略效果？并考虑兼容性？

单行（推荐）：

```
p {
 overflow: hidden;
 text-overflow:ellipsis;
 white-space: nowrap;
}
```

多行（推荐）：

```
 p {
   display: -webkit-box;
   -webkit-box-orient: vertical;
   -webkit-line-clamp: 3; // 行数
   overflow: hidden;
   width: 600px;
 }
```

考虑兼容：

```
 p{
  position: relative;
  line-height: 20px;
  max-height: 40px;
  overflow: hidden;
 }
 // 使用为元素
 p::after{
  content: "...";
  position: absolute;
  bottom: 0;
  right: 0;
  padding-left: 40px;
  background: -webkit-linear-gradient(left, transparent, #fff 55%);
  background: -o-linear-gradient(right, transparent, #fff 55%);
  background: -moz-linear-gradient(right, transparent, #fff 55%);
  background: linear-gradient(to right, transparent, #fff 55%);
}
```

35.如何计算白屏时间和首屏加载时间

白屏时间:

- window.performance.timing.domLoading - window.performance.timing.navigationStart

首屏时间:

- window.performance.timing.domInteractive - window.performace.timing.navigationSta

36.如何自定义滚动条的样式

滚动条相关样式都是伪元素，以 `scrollbar` 打头，有以下伪元素，从 `-webkit` 中可见兼容性一般，不过无所谓，现在 Chrome 浏览器占大头

- `::-webkit-scrollbar` — 整个滚动条.
- `::-webkit-scrollbar-button` — 滚动条上的按钮 (上下箭头).
- `::-webkit-scrollbar-thumb` — 滚动条上的滚动滑块.
- `::-webkit-scrollbar-track` — 滚动条轨道.
- `::-webkit-scrollbar-track-piece` — 滚动条没有滑块的轨道部分.
- `::-webkit-scrollbar-corner` — 当同时有垂直滚动条和水平滚动条时交汇的部分.
- `::-webkit-resizer` — 某些元素的 corner 部分的部分样式(例:textarea 的可拖动按钮).

但其实最常用的是以下几个伪元素：**滚动条、滑块、轨道**，如下滚动条设置成功

```
::-webkit-scrollbar {
  width: 6px;
  height: 6px;
}

::-webkit-scrollbar-track {
  border-radius: 3px;
  background: rgba(0, 0, 0);
  box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.08);
}

::-webkit-scrollbar-thumb {
  border-radius: 3px;
  background: rgba(0, 0, 1);
  box-shadow: inset 0 0 10px rgba(0, 0, 0, 0.2);
}
```