

前端面试之CSS篇

一、CSS基础

1.CSS选择器及其优先级

选择器	格式	优先级权重
id选择器	#id	100
类选择器	.classname	10
属性选择器	a[ref="eee"]	10
伪类选择器	li:last-child	10
标签选择器	div	1
伪元素选择器	li:after	1
相邻兄弟选择器	h1+p	0
子选择器	ul>li	0
后代选择器	li a	0
通配符选择器	*	0

对于选择器的优先级：

- 标签选择器、伪元素选择器：1；
- 类选择器、伪类选择器、属性选择器：10；
- id 选择器：100；
- 内联样式：1000；

注意事项：

- `!important` 声明的样式的优先级最高(`!important` 用于向属性/值添加比正常值更重要的内容，将覆盖该元素上特定属性所有以前的样式设置规则。更改一个 `!important` 规则样式的唯一方法是包括另一个在源代码中具有相同（或更高）特定性的声明的 `!important` 规则。)
- 如果优先级相同，则最后出现的样式生效；
- 继承得到的样式的优先级最低；
- 通用选择器（*）、子选择器（>）和相邻同胞选择器（+）并不在这四个等级中，所以它们的权值都为 0；
- 样式表的来源不同时，优先级顺序为：**内联样式 > 内部样式 > 外部样式 > 浏览器用户自定义样式 > 浏览器默认样式。**

2.伪元素和伪类

1.伪元素：在内容元素的前后插入额外的元素或样式，但是这些元素实际上并不在文档中生成。它们只在外部显示可见，但不会在文档的源代码中找到它们，因此，称为“伪”元素。例如：

```
p::before {content: "第一章: ";}
p::after {content: "Hot!";}
p::first-line {background: red;}
p::first-letter {font-size: 30px;}
```

2. 伪类：将特殊的效果添加到特定选择器上。它是已有元素上添加类别的，不会产生新的元素。例如：

```
a:hover {color: #FF00FF}
p:first-child {color: red}
```

3.CSS中可继承与不可继承的属性有哪些

1. 无继承性的属性

1. display：规定元素应该生成的框的类型
2. 文本属性：
 - vertical-align：垂直文本对齐
 - text-decoration：规定添加到文本的装饰
 - text-shadow：文本阴影效果
 - white-space：空白符的处理
 - unicode-bidi：设置文本的方向
3. 盒子模型的属性：width、height、margin、border、padding
4. 背景属性：background、background-color、background-image、background-repeat、background-position、background-attachment
5. 定位属性：float、clear、position、top、right、bottom、left、min-width、min-height、max-width、max-height、overflow、clip、z-index
6. 生成内容属性：content、counter-reset、counter-increment
7. 轮廓样式属性：outline-style、outline-width、outline-color、outline
8. 页面样式属性：size、page-break-before、page-break-after
9. 声音样式属性：pause-before、pause-after、pause、cue-before、cue-after、cue、play-during

2. 有继承性的属性

1. 字体系列属性

- font-family：字体系列
- font-weight：字体的粗细
- font-size：字体的大小
- font-style：字体的风格

2. 文本系列属性

- text-indent：文本缩进
- text-align：文本水平对齐
- line-height：行高
- word-spacing：单词之间的间距
- letter-spacing：中文或者字母之间的间距
- text-transform：控制文本大小写（就是uppercase、lowercase、capitalize这三个）
- color：文本颜色

3. 元素可见性

- visibility: 控制元素显示隐藏列表布局属性
- list-style: 列表风格, 包括list-style-type、list-style-image等

5. 光标属性

- cursor: 光标显示为何种形态

3.display的属性值及其作用

属性值	作用
none	元素不显示, 并且会从文档流中移除。
block	块类型。默认宽度为父元素宽度, 可设置宽高, 换行显示。
inline	行内元素类型。默认宽度为内容宽度, 不可设置宽高, 同行显示。
inline-block	同一行内的块内元素。默认宽度为内容宽度, 可以设置宽高, 同行显示。
list-item	像块类型元素一样显示, 并添加样式列表标记。
table	此元素会作为块级表格来显示。
inherit	规定应该从父元素继承display属性的值。

4.display中block、inline和inline-block的区别

1.block:

- 上下左右的padding、margin、width、height都可以设置
- 独占一行, 并会自动换行
- 多个块状, 默认排列从上到下

2.inline:

- 只能设置水平方向的padding、margin, 无法设置垂直方向的padding、margin
- 无法设置padding、margin
- 不独占一行, 不会自动换行

3.inline-block:

- 与inline相比, 可以在元素上设置宽高, 内外边距
- 将对象设置为inline对象, 但是对象的内容作为block呈现, 之后的内联元素会被排列在同一行内 (用于水平而不是垂直的设置列表项, 可实现导航栏)

5.隐藏元素的方法

- display:none: dom树不渲染该对象, 因此该元素不在页面中占位, 也不会响应的监听事件
- visibility:hidden: 元素在页面中仍占据空间, 但不响应绑定的事件
- opacity:0: 设置元素的透明度为0, 元素在页面中占据空间, 响应绑定的监听事件
- transform:scale(0,0): 将元素缩放为0, 元素仍在页面中占据位置, 但是不会响应绑定的监听事件。
- position: absolute: 使用绝对定位将元素移除可视区域
- z-index: 负值: 来使其他元素遮盖住该元素, 以此来实现隐藏。
- clip/clip-path: 使用元素裁剪的方法来实现元素的隐藏, 这种方法下, 元素仍在页面中占据位置, 但是不会响应绑定的监听事件

6.display:none与visibility:hidden的区别

这两个属性都是让元素隐藏，不可见。两者区别如下：

1.在渲染树中

- display:none会让元素完全从渲染树中消失，渲染时不会占据任何空间；
- visibility:hidden不会让元素从渲染树中消失，渲染的元素还会占据相应的空间，只是内容不可见。

2.是否是继承属性

- display:none是非继承属性，子孙节点会随着父节点从渲染树消失，通过修改子孙节点的属性也无法显示；
- visibility:hidden是继承属性，子孙节点消失是由于继承了hidden，通过设置visibility:visible可以让子孙节点显示；
- 修改常规文档流中元素的 **display** 通常会造成文档的重排，但是修改visibility属性只会造成本元素的重绘；
- 如果使用读屏器，设置为display:none的内容不会被读取，设置为visibility:hidden的内容会被读取。

7.link和@import的区别

1.link是在页面加载时同时载入，@import需要页面完成载入后加载

2.link是XHTML标签，除了CSS外，还可以定义RSS等其他事务；@import属于CSS范畴，只能加载CSS。

RSS:网页内容聚合器，对那些频繁更新内容的网站是很有帮助的，比如：

- 新闻站点 - 列出新闻的标题、日期以及描述
- 企业 - 列出新闻和新产品
- 日程表 - 列出即将来临的安排和重要日期
- 站点更新 - 列出更新过的页面或新的页面

3.link支持使用javascript控制DOM去改变样式；而@import不支持。

4.link是XHTML标签，无兼容问题；@import是在CSS2.1提出的，低版本的浏览器不支持

8.src和href的区别

src 用于替换当前元素，href 用于在当前文档和引用资源之间确立联系。

1.1 src

src 是 source 的缩写，指向外部资源的位置，指向的内容将会**嵌入**到文档中当前标签所在位置；在请求src资源时会将其指向的资源下载并应用到文档内，例如js脚本，img图片和frame等元素。

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

当浏览器解析到该元素时，会**暂停其他资源的下载和处理**，直到将该资源加载、编译、执行完毕。图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将js脚本放在底部而不是头部。

1.2 href

href 是 Hypertext Reference 的缩写，指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，如果在文档中添加

```
<link href="common.css" rel="stylesheet"/>
//禁用href跳转页面或定位链接
e.preventDefault();
href="javascript:void(0);
```

那么浏览器会识别该文档为 css 文件，就会**并行下载资源并且不会停止对当前文档的处理**。这也是为什么建议使用 link 方式来加载 css，而不是使用@import 方式。

9.transition和animation的区别

1.transition

过渡属性，强调过度，它的实现需要触发一个事件（比如鼠标移动上去，焦点，点击等）才执行动画。它类似于flash的补间动画，设置一个开始关键帧，一个结束关键帧，然后补充中间帧

```
#test{
  width: 200px;
  height: 200px;
  position: center;
  background-color: antiquewhite;
  transition:width 10s linear ;    //参数分别表示动画对象属性、过渡的时间、过渡的方式
}
#test:hover{
  width: 300px;
  background-color: #20da64;
}
```

transition的特点：

- transition需要一个事件来触发，比如hover，所以没法在网页加载时自动发生
- transition是一次性的，不能重复发生，除非一再触发。
- transition只能定义开始状态和结束状态，不能定义中间状态，也就是说只有两个状态。
- 一条transition规则，只能定义一个属性的变化，不能涉及多个属性。

2.animation

动画属性，它的实现不需要触发事件，设定好时间之后可以自己执行，且可以循环一个动画。它也类似于flash的补间动画，但是它可以设置多个关键帧（用@keyframe定义）完成动画，由@keyframes来描述每一帧的样式

```
//参数表示：动画名称 动画时长 速度曲线 延迟时间 重复次数 动画方向 执行完毕状态；
#box {
  width: 200px;
  height: 100px;
  background-color: pink;
  animation: change 1s;    //调用已经定义好的关键帧
}

@keyframes change {
```

```
0% {  
    width: 200px;  
}  
50% {  
    width: 500px;  
    height: 300px;  
    background-color: antiquewhite;  
}  
100% {  
    width: 800px;  
    height: 500px;  
    background-color: aquamarine;  
}  
}
```

animation的特点:

- animation不需要触发
- animation可设置循环次数
- animation可设置每一帧的样式和时间，transition只能设置头尾

10.对requestAnimationFrame的理解

setTimeout执行动画的缺点：它通过设定间隔时间来不断改变图像位置，达到动画效果。但是容易出现卡顿、抖动的现象；原因是：

- setTimeout任务被放入异步队列，只有当主线程任务执行完后才会执行队列中的任务，因此实际执行时间总是比设定时间要晚；
- setTimeout的固定时间间隔不一定与屏幕刷新闻隔时间相同，会引起丢帧。

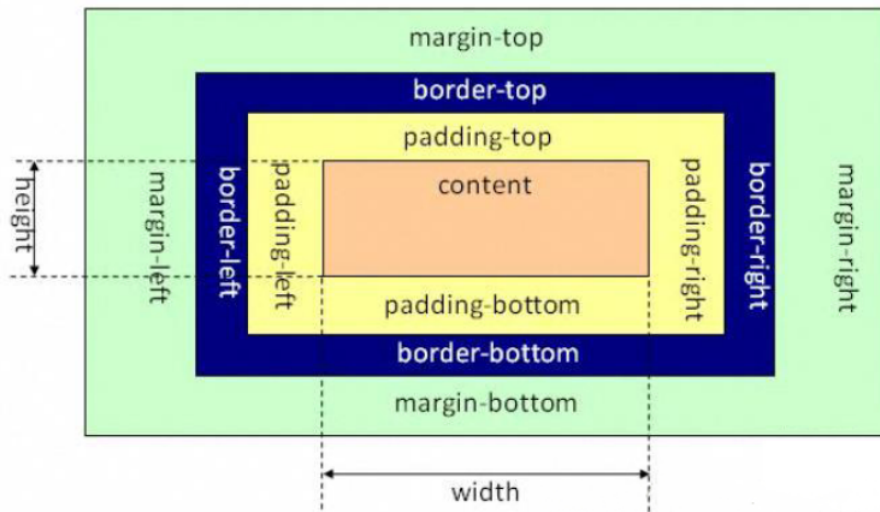
实现动画效果的方法比较多，Javascript 中可以通过定时器 setTimeout 来实现，CSS3 中可以使用 transition 和 animation 来实现，HTML5 中的 canvas 也可以实现。除此之外，HTML5 提供一个专门用于请求动画的API，那就是 requestAnimationFrame，顾名思义就是请求动画帧。

语法： window.requestAnimationFrame(callback); 其中，callback是下一次重绘之前更新动画帧所调用的函数(即上面所说的回调函数)。该回调函数会被传入DOMHighResTimeStamp参数，它表示 requestAnimationFrame() 开始去执行回调函数的时刻。该方法属于宏任务，所以会在执行完微任务之后再执行。

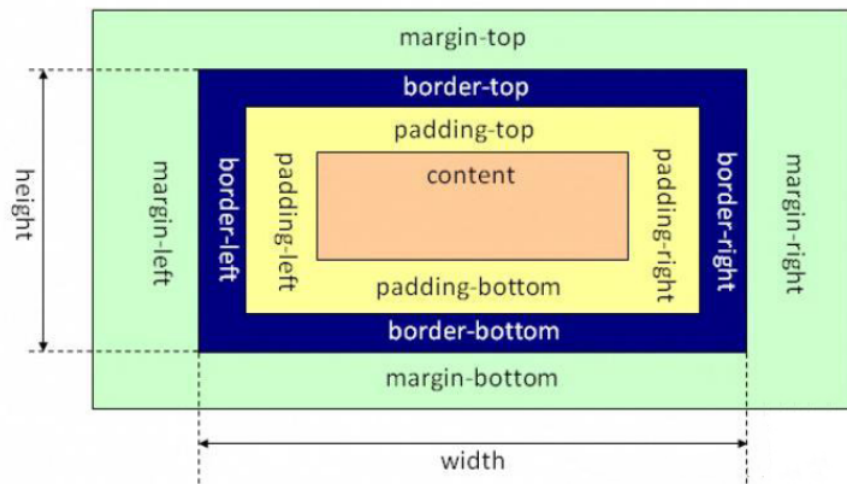
11.对盒模型的理解

CSS3中的盒模型有以下两种：标准盒子模型、IE盒子模型

■ 标准盒子模型



■ IE盒子模型



两个主要的区别在于height和width指代的内容不一致：

- 标准盒模型的width和height属性的范围只包含了content，
- IE盒模型的width和height属性的范围包含了border、padding和content。

可以通过修改元素的box-sizing属性来改变元素的盒模型：

- box-sizing: content-box表示标准盒模型（默认值）
- box-sizing: border-box表示IE盒模型（怪异盒模型）

12.为什么有时候用translate来改变位置而不是定位？

translate 是 transform 属性的一个值。改变transform或opacity不会触发浏览器重新布局（reflow）或重绘（repaint），只会触发复合（compositions）。而改变绝对定位会触发重新布局，进而触发重绘和复合。**transform使浏览器为元素创建一个 GPU 图层，而改变绝对定位会使用到 CPU。** 因此 translate()更高效，可以缩短平滑动画的绘制时间。而translate改变位置时，元素依然会占据其原始空间，绝对定位就不会发生这种情况。

13.li 与 li 之间有看不见的空白间隔是什么原因引起的？如何解决？

浏览器会把inline内联元素间的空白字符（空格、换行、Tab等）渲染成一个空格。为了美观，通常是一个

- 放在一行，这导致
- 换行后产生换行字符，它变成一个空格，占用了一个字符的宽度。

解决方法：

(1) 为

- 设置float:left。不足：有些容器是不能设置浮动，如左右切换的焦点图等。

(2) 将所有

- 写在同一行。不足：代码不美观。

(3) 将

内的字符尺寸直接设为0，即font-size:0。不足：

中的其他字符尺寸也被设为0，需要额外重新设定其他字符尺寸，且在Safari浏览器依然会出现空白间隔。

(4) 消除

的字符间隔letter-spacing:-8px，不足：这也设置了

- 内的字符间隔，因此需要将
- 内的字符间隔设为默认letter-spacing:normal。

14.CSS3中的新特性：

- 新增各种CSS选择器（:not(input)：所有 class 不是“input”的节点）
- 圆角（border-radius:8px）
- 多列布局（multi-column layout）

多列布局：我们有时想布局成报纸、杂志那样的多列方式（至少两列，一般三列以上），但早期 CSS 提供的布局方式都有着极大的限制。如果是固体布局，那么使用浮动或定位布局都可以完成。但对于流体的多列，比如三列以上，那只能使用浮动布局进行，而它又有极大的限制。因为它是区块性的，对于动态的内容无法伸缩自适应，基本无能力。CSS3提供columns属性，可以控制分列数和列宽等，见后续的补充

- 阴影和反射（Shadoweflect）
- 文字特效（text-shadow）
- 文字渲染（Text-decoration）
- 线性渐变（gradient）
- 旋转（transform）
- 增加了旋转,缩放,定位,倾斜,动画,多背景

15.替换元素的概念及计算规则（没仔细看）

通过修改某个属性值呈现的内容就可以被替换的元素就称为“替换元素”。

替换元素除了内容可替换这一特性以外，还有以下特性：

- 内容的外观不受页面上的CSS的影响：用专业的话讲就是在样式表现在CSS作用域之外。如何更改替换元素本身的外观需要类似appearance属性，或者浏览器自身暴露的一些样式接口。
- 有自己的尺寸：在Web中，很多替换元素在没有明确尺寸设定的情况下，其默认的尺寸（不包括边框）是300像素×150像素，如
- 在很多CSS属性上有自己的一套表现规则：比较具有代表性的就是vertical-align属性，对于替换元素和非替换元素，vertical-align属性值的解释是不一样的。比方说vertical-align的默认值的baseline，很简单的属性值，基线之意，被定义为字符x的下边缘，而替换元素的基线却被硬生生定义成了元素的下边缘。
- 所有的替换元素都是内联水平元素：也就是替换元素和替换元素、替换元素和文字都是可以在一行显示的。但是，替换元素默认的display值却是不一样的，有的是

inline, 有的是inline-block。

替换元素的尺寸从内而外分为三类：

- 固有尺寸：指的是替换内容原本的尺寸。例如，图片、视频作为一个独立文件存在的时候，都是有着自己的宽度和高度的。
- HTML尺寸：只能通过HTML原生属性改变，这些HTML原生属性包括的width和height属性、的size属性。
- CSS尺寸：特指可以通过CSS的width和height或者max-width/min-width和max-height/min-height设置的尺寸，对应盒尺寸中的content box。

这三层结构的计算规则具体如下：

- (1) 如果没有CSS尺寸和HTML尺寸，则使用固有尺寸作为最终的宽高。
- (2) 如果没有CSS尺寸，则使用HTML尺寸作为最终的宽高。
- (3) 如果有CSS尺寸，则最终尺寸由CSS属性决定。
- (4) 如果“固有尺寸”含有固有的宽高比例，同时仅设置了宽度或仅设置了高度，则元素依然按照固有的宽高比例显示。
- (5) 如果上面的条件都不符合，则最终宽度表现为300像素，高度为150像素。
- (6) 内联替换元素和块级替换元素使用上面同一套尺寸计算规则。

16.常见的图片格式及使用场景

(1) BMP，是无损的、既支持索引色也支持直接色的点阵图。这种图片格式几乎没有对数据进行压缩，所以BMP格式的图片通常是较大的文件。

(2) GIF是无损的、采用索引色的点阵图。采用LZW压缩算法进行编码。文件小，是GIF格式的优点，同时，GIF格式还具有支持动画以及透明的优点。但是GIF格式仅支持8bit的索引色，所以GIF格式适用于对色彩要求不高同时需要文件体积较小的场景。

(3) JPEG是有损的、采用直接色的点阵图。JPEG的图片的优点是采用了直接色，得益于更丰富的色彩，JPEG非常适合用来存储照片，与GIF相比，JPEG不适合用来存储企业Logo、线框类的图。因为有损压缩会导致图片模糊，而直接色的选用，又会导致图片文件较GIF更大。

(4) PNG-8是无损的、使用索引色的点阵图。PNG是一种比较新的图片格式，PNG-8是非常好的GIF格式替代者，在可能的情况下，应该尽可能的使用PNG-8而不是GIF，因为在相同的图片效果下，PNG-8具有更小的文件体积。除此之外，PNG-8还支持透明度的调节，而GIF并不支持。除非需要动画的支持，否则没有理由使用GIF而不是PNG-8。

(5) PNG-24是无损的、使用直接色的点阵图。PNG-24的优点在于它压缩了图片的数据，使得同样效果的图片，PNG-24格式的文件大小要比BMP小得多。当然，PNG24的图片还是要比JPEG、GIF、PNG-8大得多。

(6) SVG是无损的矢量图。SVG是矢量图意味着SVG图片由直线和曲线以及绘制它们的方法组成。当放大SVG图片时，看到的还是线和曲线，而不会出现像素点。SVG图片在放大时，不会失真，所以它适合用来绘制Logo、Icon等。

(7) WebP是谷歌开发的一种新图片格式，WebP是同时支持有损和无损压缩的、使用直接色的点阵图。从名字就可以看出来它是为Web而生的，什么叫为Web而生呢？就是说相同质量的图片，WebP具有更小的文件体积。现在网站上充满了大量的图片，如果能够降低每一个图片的文件大小，那么将大大减少浏览器和服务器之间的数据传输量，进而降低访问延迟，提升访问体验。目前只有Chrome浏览器和Opera浏览器支持WebP格式，兼容性不太好。

- 在无损压缩的情况下，相同质量的WebP图片，文件大小要比PNG小26%；
- 在有损压缩的情况下，具有相同图片精度的WebP图片，文件大小要比JPEG小25%~34%；
- WebP图片格式支持图片透明度，一个无损压缩的WebP图片，如果要支持透明度只需要22%的额外文件大小。

17.对 CSSSprites 的理解（没仔细看）

CSSSprites（精灵图/雪碧图），**将一个页面涉及到的所有图片都包含到一张大图中**，然后利用CSS的 background-image, background-repeat, background-position属性的组合进行背景定位。

优点：

- 利用CSS Sprites能很好地减少网页的http请求，从而大大提高了页面的性能，只要一次请求一张大图就可以了，这是CSS Sprites最大的优点；
- CSS Sprites能减少图片的字节，把3张图片合并成1张图片的字节总是小于这3张图片的字节总和。
- 解决了网页设计师在图片命名上的困扰，只需对一张集合的图片上命名就可以了，不需要对每一个小元素进行命名，从而提高了网页的制作效率。
- 更换风格方便，只需要在一张或少张图片上修改图片的颜色或样式，整个网页的风格就可以改变。维护起来更加方便。

缺点：

- 在图片合并时，要把多张图片有序的、合理的合并成一张图片，还要留好足够的空间，防止板块内出现不必要的背景。在宽屏及高分辨率下的自适应页面，如果背景不够宽，很容易出现背景断裂；
- CSSSprites在开发的时候相对来说有点麻烦，需要借助photoshop或其他工具来对每个背景单元测量其准确的位置。
- 维护方面：CSS Sprites在维护的时候比较麻烦，页面背景有少许改动时，就要改这张合并的图片，无需改的地方尽量不要动，这样避免改动更多的CSS，如果在原来的地方放不下，又只能（最好）往下加图片，这样图片的字节就增加了，还要改动CSS。

18.什么是物理像素，逻辑像素和像素密度，为什么在移动端开发时需要用到@3x, @2x这种图片？（没仔细看）

19.margin 和 padding 的使用场景

- 需要在border外侧添加空白，且空白处不需要背景（色）时，使用 margin；
- 需要在border内测添加空白，且空白处需要背景（色）时，使用 padding。

20.对line-height 的理解及其赋值方式（没仔细看）

(1) line-height的概念：

- line-height 指一行文本的高度，包含了字间距，实际上是下一行基线到上一行基线距离；
- 如果一个标签没有定义 height 属性，那么其最终表现的高度由 line-height 决定；
- 一个容器没有设置高度，那么撑开容器高度的是 line-height，而不是容器内的文本内容；
- 把 line-height 值设置为 height 一样大小的值可以实现单行文字的垂直居中；
- line-height 和 height 都能撑开一个高度；

(2) line-height 的赋值方式：

- 带单位：px 是固定值，而 em 会参考父元素 font-size 值计算自身的行高
- 纯数字：会把比例传递给后代。例如，父级行高为 1.5，子元素字体为 18px，则子元素行高为 $1.5 * 18 = 27px$
- 百分比：将计算后的值传递给后代

21. CSS 优化和提高性能的方法有哪些？

1.加载性能

(1) css压缩：将写好的css进行打包压缩，可以减小文件体积。

(2) css单一样式：当需要下边距和左边距的时候，很多时候会选择使用 `margin:top 0 bottom 0`,但 `margin-bottom:bottom;margin-left:left`; 执行效率会更高。

(3) 减少使用@import，建议使用link，因为后者在页面加载时一起加载，前者是等待页面加载完成之后再加载。

2.选择器性能

(1) 关键选择器（key selector）。选择器的最后面的部分为关键选择器（即用来匹配目标元素的部分）。CSS选择符是从右到左进行匹配的。当使用后代选择器的时候，浏览器会遍历所有子元素来确定是否是指定的元素等等；

(2) 如果规则拥有ID选择器作为其关键选择器，则不要为规则增加标签。过滤掉无关的规则（这样样式系统就不会浪费时间去匹配它们了）。

(3) 避免使用通配规则，如*{}计算次数惊人，只对需要用到的元素进行选择。

(4) 尽量少的去对标签进行选择，而是用class。

(5) 尽量少的去使用后代选择器，降低选择器的权重值。后代选择器的开销是最高的，尽量将选择器的深度降到最低，最高不要超过三层，更多的使用类来关联每一个标签元素。

(6) 了解哪些属性是可以通过继承而来的，然后避免对这些属性重复指定规则。

3.渲染性能

(1) 慎重使用高性能属性：浮动、定位。尽量减少页面重排、重绘。

(3) 去除空规则：{}。空规则的产生原因一般来说是为了预留样式。去除这些空规则无疑能减少css文档体积。

(4) 属性值为0时，不加单位。

(5) 属性值为浮动小数0.**，可以省略小数点之前的0。

(6) 标准化各种浏览器前缀：带浏览器前缀的在前。标准属性在后。

(7) 不使用@import前缀，它会影响css的加载速度。

(8) 选择器优化嵌套，尽量避免层级过深。

(9) css雪碧图，同一页面相近部分的小图标，方便使用，减少页面的请求次数，但是自适应问题需要重视，使用时，优劣考虑清楚，再使用。

(10) 正确使用display的属性，由于display的作用，某些样式组合会无效，徒增样式体积的同时也影响解析性能。

(11) 不滥用web字体。对于中文网站来说WebFonts可能很陌生，国外却很流行。web fonts通常体积庞大，而且一些浏览器在下载web fonts时会阻塞页面渲染损伤性能。

4.可维护性、健壮性

(1) 将具有相同属性的样式抽离出来，整合并通过class在页面中进行使用，提高css的可维护性。

(2) 样式与内容分离：将css代码定义到外部css中。

22. CSS预处理器/后处理器是什么？为什么要使用它们？ (没仔细看)

预处理器：

目前主流的CSS预处理器包括less, sass, stylus, 用来预编译sass或者less, 增加了css代码的复用性。层级, mixin, 变量, 循环, 函数等对编写以及开发UI组件都极为方便。

后处理器：

如：postCss, 通常是在完成的样式表中根据css规范处理css, 让其更加有效。目前最常做的是给css属性添加浏览器私有前缀, 实现跨浏览器兼容性的问题。

css预处理器为css增加一些编程特性, 无需考虑浏览器的兼容问题, 可以在CSS中使用变量, 简单的逻辑程序, 函数等在编程语言中的一些基本的性能, 可以让css更加的简洁, 增加适应性以及可读性, 可维护性等。

使用原因：

- 结构清晰, 便于扩展
- 可以很方便的屏蔽浏览器私有语法的差异
- 可以轻松实现多重继承
- 完美的兼容了CSS代码, 可以应用到老项目中

23. ::before 和 :after 的双冒号和单冒号有什么区别？

(1) 冒号(:)用于CSS3伪类, 双冒号(::)用于CSS3伪元素。

(2) ::before就是以子元素的存在, 定义在元素主体内容之前的一个伪元素。并不存在于dom之中, 只存在于页面之中。

注意: :before 和 :after 这两个伪元素, 是在CSS2.1里新出现的。起初, 伪元素的前缀使用的是单冒号语法, 但随着Web的进化, 在CSS3的规范里, 伪元素的语法被修改成使用双冒号, 成为::before、::after。

24.display:inline-block 什么时候会显示间隙？

- 有空格时会有间隙, 可以删除空格解决;
- margin正值时, 可以让margin使用负值解决;
- 使用font-size时, 可通过设置font-size:0、letter-spacing、word-spacing解决;

25.文本溢出解决方法

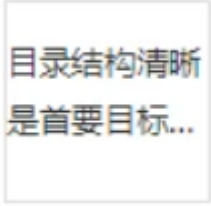
在做响应式网页的时候, 我们要想在不同尺寸的设备下保证布局不会错乱, 就需要对文字的长度进行限定。

1.单行文本溢出

目录结构清...

```
overflow: hidden;           // 溢出隐藏
text-overflow: ellipsis;    // 溢出用省略号显示
white-space: nowrap;        // 规定段落中的文本不进行换行
```

2.多行文本溢出



```
overflow: hidden;           // 溢出隐藏
text-overflow: ellipsis;    // 溢出用省略号显示
display: -webkit-box;       // 作为弹性伸缩盒子模型显示。
-webkit-box-orient: vertical; // 设置伸缩盒子的子元素排列方式：从上到下垂直排列
-webkit-line-clamp: 3;      // 显示的行数
```

注意：由于上面的三个属性都是 CSS3 的属性，没有浏览器可以兼容，所以要在前面加一个-webkit- 来兼容一部分浏览器。

26.ass、Less 是什么？为什么要使用他们？

他们都是 CSS 预处理器，是 CSS 上的一种抽象层。他们是一种特殊的语法/语言编译成 CSS。

1.less

Less 是一种动态样式语言，将 CSS 赋予了动态语言的特性，如变量，继承，运算，函数，LESS 既可以在客户端上运行 (支持 IE 6+, Webkit, Firefox)，也可以在服务端运行 (借助 Node.js)。

```
//-----变量的使用-----
@width: 10px;
@height: @width + 10px;

#header {
  width: @width;
  height: @height;
}

//编辑为:
#header {
  width: 10px;
  height: 20px;
}

//-----mixin的使用-----
//定义一组属性
.bordered {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}
```

```
#menu a {
  color: #111;
  .bordered(); //在其他规则集中使用定义好的属性
}

.post a {
  color: red;
  .bordered(); //在其他规则集中使用定义好的属性
}

//-----less中的运算-----
// 所有操作数被转换成相同的单位
@conversion-1: 5cm + 10mm; // 结果是 6cm
@conversion-2: 2 - 3cm - 5mm; // 结果是 -1.5cm

// conversion is impossible
@incompatible-units: 2 + 5px - 3cm; // 结果是 4px

// example with variables
@base: 5%;
@filler: @base * 2; // 结果是 10%
@other: @base + @filler; // 结果是 15%
```

2.sass

sass 让人们受益的一个重要特性就是它为 css 引入了变量。你可以把反复使用的 css 属性值 定义成变量，然后通过变量名来引用它们，而无需重复书写这一属性值。

3.为什么要使用它们？

- 结构清晰，便于扩展。可以方便地屏蔽浏览器私有语法差异。封装对浏览器语法差异的重复处理，减少无意义的机械劳动。
- 可以轻松实现多重继承。完全兼容 CSS 代码，可以方便地应用到老项目中。LESS 只是在 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。

27.对媒体查询的理解

CSS2 中引入了 `@media` 规则，它让为不同媒体类型定义不同样式规则成为可能。CSS3 中的媒体查询扩展了 CSS2 媒体类型的概念：它们并不查找设备类型，而是关注设备的能力。媒体类型包括：

值	描述
all	用于所有媒体类型设备。
print	用于打印机。
screen	用于计算机屏幕、平板电脑、智能手机等等。
speech	用于大声“读出”页面的屏幕阅读器。

媒体查询可用于检查许多事情，例如：

- 视口的宽度和高度
- 设备的宽度和高度

- 方向（平板电脑/手机处于横向还是纵向模式）
- 分辨率

使用媒体查询是一种流行的技术，可以向台式机、笔记本电脑、平板电脑和手机（例如 iPhone 和 Android 手机）提供定制的样式表。如果指定的媒体类型与正在显示文档的设备类型匹配，并且媒体查询中的所有表达式均为 true，则查询结果为 true。当媒体查询为 true 时，将应用相应的样式表或样式规则，并遵循正常的级联规则。下例显示，视口宽度为 480 像素或更宽时将背景颜色更改为浅绿色（如果视口小于 480 像素，则背景颜色会是粉色）：

```
body {  
  background-color: pink;  
}  
  
@media screen and (min-width: 480px) {  
  body {  
    background-color: lightgreen;  
  }  
}
```

28.对CSS工程化的理解

CSS 工程化是为了解决以下问题：

1. 宏观设计：CSS 代码如何组织、如何拆分、模块结构怎样设计？
2. 编码优化：怎样写出更好的 CSS？
3. 构建：如何处理我的 CSS，才能让它的打包结果最优？
4. 可维护性：代码写完了，如何最小化它后续的变更成本？如何确保任何一个同事都能轻松接手？

以下三个方向都是时下比较流行的、普适性非常好的 CSS 工程化实践：

- 预处理器：Less、Sass 等；（类CSS->预处理器->CSS代码）
- 重要的工程化插件：PostCss；
- Webpack loader 等。

1.预处理器

目前主流的CSS预处理器包括less, sass, stylus, 用来预编译sass或者less, 增加了css代码的复用性。层级, mixin, 变量, 循环, 函数等对编写以及开发UI组件都极为方便。预处理器普遍会具备这样的特性：

- 嵌套代码的能力，通过嵌套来反映不同 css 属性之间的层级关系；
- 支持定义 css 变量；
- 提供计算函数；
- 允许对代码片段进行 extend 和 mixin；
- 支持循环语句的使用；
- 支持将 CSS 文件模块化，实现复用。

2.PostCss

它可以编译尚未被浏览器广泛支持的先进的 CSS 语法，还可以自动为一些需要额外兼容的语法增加前缀。更强的是，由于 PostCss 有着强大的插件机制，支持各种各样的扩展，极大地强化了 CSS 的能力。

PostCss 在业务中的使用场景非常多：

- 提高 CSS 代码的可读性：PostCss 其实可以做类似预处理器能做的工作；

- 当我们的 CSS 代码需要适配低版本浏览器时，PostCss 的 [Autoprefixer](#) 插件可以帮助我们自动增加浏览器前缀；
- 允许我们编写面向未来的 CSS：PostCss 能够帮助我们编译 CSS next 代码；

3.Webpack loader

Webpack 在裸奔的状态下，是不能处理 CSS 的，Webpack 本身是一个面向 JavaScript 且只能处理 JavaScript 代码的模块化打包工具；但是在 loader 的辅助下，它是可以处理 CSS 的。主要用到了两个模块：

- css-loader：导入 CSS 模块，对 CSS 代码进行编译处理；
- style-loader：创建style标签，把 CSS 内容写入标签。

在实际使用中，css-loader 的执行顺序一定要安排在 style-loader 的前面。因为只有完成了编译过程，才可以对 css 代码进行插入；若提前插入了未编译的代码，那么 webpack 是无法理解这坨东西的，它会无情报错。

29.如何判断元素已经到达可视区域？

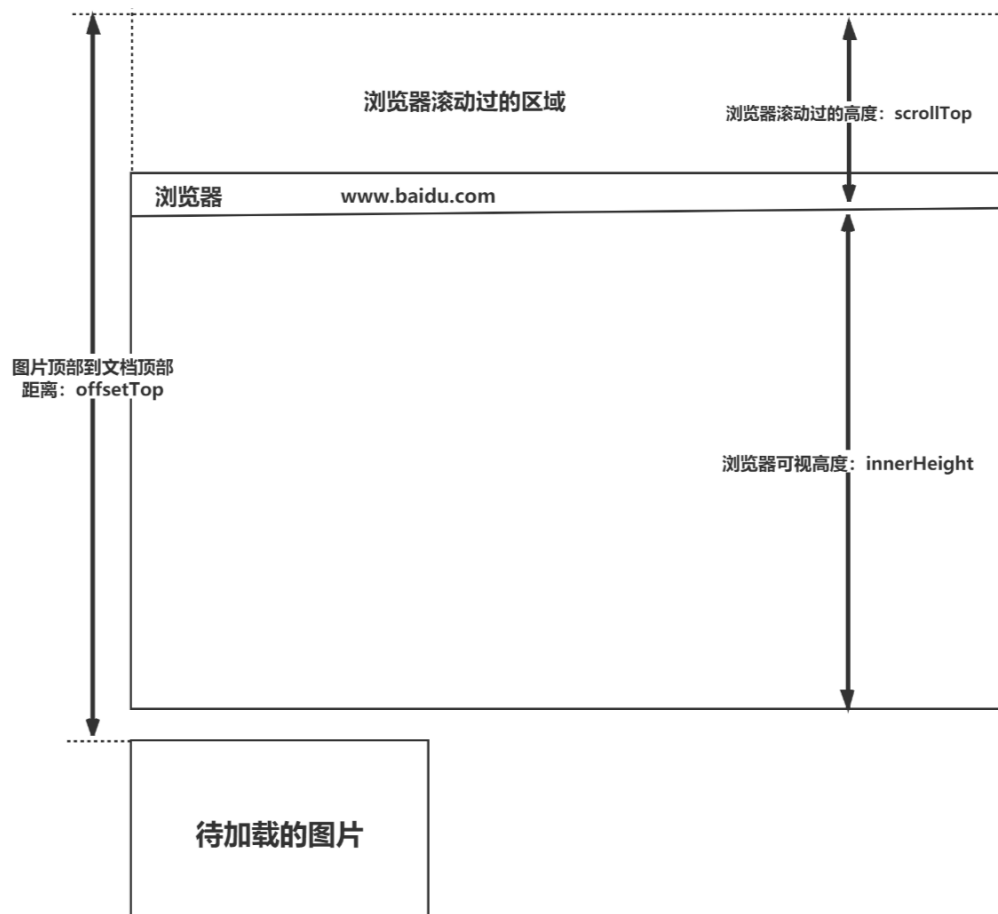
偏移量(offset dimension),元素的可见大小由其高度、宽度决定，包括所有内边距、滚动条和边框大小(注意，不包括外边距)。通过下列4个属性可以取得元素的偏移量。

偏移量	概念	公式
offsetHeight	元素在垂直方向上占用的空间大小，以像素计。包括元素的高度、(可见的)水平滚动条的高度、上边框高度和下边框高度。	offsetHeight = content + padding + border + scrollX
offsetWidth	元素在水平方向上占用的空间大小，以像素计。包括元素的宽度、(可见的)垂直滚动条的宽度、左边框宽度和右边框宽度。	offsetWidth = content + padding + border + scrollY
offsetLeft	元素的左外边框至**包含元素的左内边框之间的像素距离。	
offsetTop	元素的上外边框至包含元素的上内边框之间的像素距离。	

代码实现：

以图片显示为例：

- window.innerHeight 是浏览器可视区的高度；
- document.body.scrollTop || document.documentElement.scrollTop 是浏览器滚动的过的距离；
- imgs.offsetTop 是元素顶部距离文档顶部的高度（包括滚动条的距离）；
- 内容达到显示区域的：img.offsetTop < window.innerHeight + document.body.scrollTop;



```
function isInViewportOfOne (el) {  
    // viewportHeight 兼容所有浏览器写法  
    const viewportHeight = window.innerHeight ||  
document.documentElement.clientHeight || document.body.clientHeight  
    const offsetTop = el.offsetTop  
    const scrollTop = document.documentElement.scrollTop  
    const top = offsetTop - scrollTop  
    return top <= viewportHeight  
}
```

30.z-index属性在什么情况下会失效

通常 z-index 的使用是在有两个重叠的标签，在一定的情况下控制其中一个在另一个的上方或者下方出现。z-index值越大就越是在上层。z-index元素的position属性需要是relative, absolute或是fixed。

z-index属性在下列情况下会失效：

- 父元素position为relative时，子元素的z-index失效。
解决：父元素position改为absolute或static
- 该元素没有设置position属性为非static属性。（static：静态定位，是html的默认定位方式，遵循正常的文档流对象，该定位方式下，top、right、bottom、left、z-index等属性是无效的）
解决：设置该元素的position属性为relative, absolute或是fixed中的一种。
- 该标签在设置z-index的同时还设置了float浮动
解决：float去除，改为display: inline-block

二、页面布局

1.常见的CSS布局单位

常用的布局单位包括**像素 (px)** , **百分比 (%)** , **em** , **rem** , **vw/vh**。

(1) **像素 (px)** 是页面布局的基础, 一个像素表示终端 (电脑、手机、平板等) 屏幕所能显示的最小的区域, 像素分为两种类型: CSS像素和物理像素:

- CSS像素: 为web开发者提供, 在CSS中使用的一个抽象单位;
- 物理像素: 只与设备的硬件密度有关, 任何设备的物理像素都是固定的。

(2) **百分比 (%)** , 当浏览器的宽度或者高度发生变化时, 通过百分比单位可以使得浏览器中的组件的宽和高随着浏览器的变化而变化, 从而实现响应式的效果。一般认为子元素的百分比相对于直接父元素。

(3) **em和rem**: 相对于px更具灵活性, 它们都是相对长度单位, 它们之间的区别: em相对于父元素, rem相对于根元素。

- em: 文本相对长度单位。相对于当前对象内文本的字体尺寸。如果当前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(默认16px)。(相对父元素的字体大小倍数)。

```
font-size:16px
padding:2em    //相对单位值, 设计渲染会生成一个32px的绝对值
```

- rem: rem是CSS3新增的一个相对单位, **相对于根元素 (html元素) 的font-size的倍数**。作用: 利用rem可以实现简单的响应式布局, 可以利用html元素中字体大小与屏幕间的比值来设置font-size的值, 以此实现当屏幕分辨率变化时让元素也随之变化

(4) **vw/vh**: 是与视图窗口有关的单位, vw表示相对于视图窗口的宽度, vh表示相对于视图窗口高度, 除了vw和vh外, 还有vmin和vmax两个相关的单位。

- vw: 相对于视窗的宽度, 视窗宽度是100vw;
- vh: 相对于视窗的高度, 视窗高度是100vh;
- vmin: vw和vh中的较小值;
- vmax: vw和vh中的较大值;

vw/vh 和百分比很类似, 两者的区别:

- 百分比 (%) : 大部分相对于祖先元素, 也有相对于自身的情况比如 (border-radius、translate等)
- vw/vm: 相对于视窗的尺寸

2. px、em、rem的区别

1.三者的区别:

- px是固定的像素, 一旦设置了就无法因为适应页面大小而改变。
- em和rem相对于px更具有灵活性, 他们是相对长度单位, 其长度不是固定的, 更适用于响应式布局。
- em是相对于其父元素来设置字体大小, 这样就会存在一个问题, 进行任何元素设置, 都有可能需要知道他父元素的大小。而rem是相对于根元素, 这就意味着, 只需要在根元素确定一个参考值。

2.使用场景：

- 对于只需要适配少部分移动设备，且分辨率对页面影响不大的，使用px即可。
- 对于需要适配各种移动设备，使用rem，例如需要适配iPhone和iPad等分辨率差别比较挺大的设备。

3.flex布局的理解及应用

Flex是FlexibleBox的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。任何一个容器都可以指定为Flex布局。行内元素也可以使用Flex布局。注意，设为Flex布局以后，子元素的float、clear和vertical-align属性将失效。flex布局是CSS3新增的一种布局方式，可以通过将一个元素的display属性值设置为flex从而使它成为一个flex容器，它的所有子元素都会成为它的项目。

一个容器默认有两条轴：一个是水平的主轴，一个是与主轴垂直的交叉轴。可以使用flex-direction来指定主轴的方向。可以使用justify-content来指定元素在主轴上的排列方式，使用align-items来指定元素在交叉轴上的排列方式。还可以使用flex-wrap来规定当一行排列不下时的换行方式。对于容器中的项目，可以使用order属性来指定项目的排列顺序，还可以使用flex-grow来指定当排列空间有剩余的时候，项目的放大比例，还可以使用flex-shrink来指定当排列空间不足时，项目的缩小比例。

Flex 布局有一个隐式的坐标空间，水平方向有一条主轴(main-axis)，垂直方向上有一条交叉轴(cross-axis)：



1.常见属性：

1.justify-content:

控制**水平方向**的对齐方式，类型有 flex-start (默认对齐方式，从左到右排列)、center (居中)、flex-end (从右到左)、space-between (两端对齐)、space-evenly (均分空间)等

2.align-items:

控制**垂直方向**的对齐方式，类型有 stretch (将子元素拉伸为容器高度)、flex-start (靠上对齐，子元素不占满容器宽度)、center、flex-end、baseline (基线对齐)

3.flex-self:

子元素控制自己在交叉轴上的对齐方式，如果想要在水平方向控制子元素，没有justify-self属性，只能通过margin属性，把左右距离设置为 auto 控制

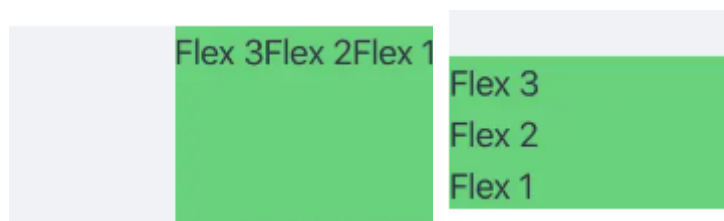
2.排列方式

flex 支持按行排布，也支持按列排布，需要设置 flex-direction: column;。按列排布时，主轴和交叉轴换了方向，但是 align-items 和 justify-content 控制的轴线不变，即 align-items 还是控制交叉轴，justify-content 控制主轴：所以在按行排序时，使用 align-items 控制水平方向的对齐，justify-content 控制垂直方向的对齐



flex也支持反向按行和列布局，相当于按容器中心线进行 180 度翻转：

```
.flex {
  display: flex;
  flex-direction: row-reverse;
}
```



3.空间占比

可以通过设置 `flex` 属性来调整空间的占比，`flex`属性是`flex-grow`，`flex-shrink`和`flex-basis`的简写，默认值为 `0 1 auto`

- 第一个参数表示: `flex-grow` 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；

- 第二个参数表示: flex-shrink 定义了项目的缩小比例, 默认为1, 即如果空间不足, 该项目将缩小;
- 第三个参数表示: flex-basis给上面两个属性分配多余空间之前, 计算项目是否有多余空间, 默认值为 auto, 即项目本身的大小。

例如让 `flex2` 在水平方向上占据其他子元素的 2 倍大小, 可以设置:

```
.flex1,
.flex3 {
  flex: 1;
}
.flex2 {
  flex: 2;
}
```

/*通过flex属性可以方便的同时设置flex-grow、flex-shrink 和 flex-basis 这三个值。*/
`flex:0 1 auto` /*即不增长, 但收缩, 收缩比例为 1, flex-basis 为 auto, 即取自用户定义的宽度或内容的宽度。

4.flex-basis:

在介绍 flex-basis 之前, 先讲一个概念 `main size`, 即主轴方向的尺寸, 那么, 在行排布模式下, 也就是水平方向的尺寸, 其实就是子元素的宽度, 而在列模式下, 它是子元素的高度, 相对应的也有 `cross size`, 即行模式下是子元素的高度, 列模式下是宽度。而 flex-basis 是用来设置 `main size` 的, 它的优先级会高于 `width`。它的默认值是 `auto`, 即在行模式下, 如果子元素设置了宽度, 它就取自这个宽度值, 没有设置的话, 就是内容的宽度。使用 `flex-basis`, 可以同时管理行模式下的宽度和列模式下的高度。

```
.flex > * {
  flex-basis: 200px;
}
```

5.缩放

1.flex-grow:

先看一下增长, `flex-grow`, 这个属性是说 flex 容器在有剩余空间的时候, 子元素占据剩余空间的占比。能够自由调整元素的空间占比

例如, 给 `flex2` 子元素设置:

```
.flex2 {
  flex-grow: 1; /*其它的元素保持默认的宽度（即内容的宽度, flex-basis 为 auto), 那么 .flex2 就会自动增长并占据整个剩余空间*/
}
```

/*如果把三个元素全部设置成 1, 那么所有元素都会自动增长, 并各自占据 1/3 的空间:

2.flex-shrink

当它们的宽度超过 flex 容器之后, 该如何进行收缩。通过 `flex-shrink` 来设置一个数值, 数值越大, 收缩程度也越大, 比如 `flex-shrink: 2` 的元素会比 `flex-shrink:1` 收缩的值大 2 倍:

6.折行

如果子元素有固定宽度，并且超出了容器的宽度，还不允许收缩的话，那么可以使用 `flex-wrap` 属性来让元素进行折行排列，使得每行的元素都不超过容器的宽度。**这里跟 `css grid` 布局的主要区别是，它无法控制单独控制行、列的占比，比如跨行、夸列，也不能自由定位元素到特定的位置。**

下边的示例新增了 2 个元素，一共 5 个，每个元素的主 size 为 300px，然后超出宽度后折行：

```
.flex {  
  flex-wrap: wrap;  
}  
  
.flex > * {  
  flex-shrink: 0;  
  flex-basis: 300px;  
}
```

4.两栏布局的实现

一般两栏布局指的是左边一栏宽度固定，右边一栏宽度自适应，两栏布局的具体实现：

1.方法一：margin-left & width

```
.outer {  
  height: 100px;  
}  
  
.left {  
  float: left;  
  width: 200px;  
  background: tomato;  
}  
  
.right {  
  margin-left: 200px;  
  width: auto;  
  background: gold;  
}
```

2.方法二：margin-left & overflow

```
.left{  
  width: 100px;  
  height: 200px;  
  background: red;  
  float: left;  
}  
  
.right{  
  height: 300px;  
  background: blue;  
  overflow: hidden;  
}
```


3.方法三：flex布局

```
.outer {
  display: flex;
  height: 100px;
}
.left {
  width: 200px;
  background: tomato;
}
.right {
  flex: 1;
  background: gold;
}
```

4.方法四：绝对定位 & margin-left

```
.outer {
  position: relative;
  height: 100px;
}
.left {
  position: absolute;
  width: 200px;
  height: 100px;
  background: tomato;
}
.right {
  margin-left: 200px;
  background: gold;
}
```

5.三栏布局的实现

一般三栏布局指页面中一共有三栏，左右两栏宽度固定，中间自适应的布局，三栏布局的具体实现：

1.绝对定位 & margin

```
/*方法一：使用绝对定位 */
.left, .right{
  position: absolute;
  width: 100px;
  background: tomato;
  background-color: darkseagreen;
}
.right{
  right: 0; /*必须设置right,top为0*/
  top:0
}
.center{
  position: relative;
  margin-left: 200px;
  margin-right: 200px;
  height: 100vh;
}
```

```
background-color: antiquewhite;
}
```

2.flex布局

```
/*方法二：使用flex */
.outer{
  display: flex;
}

.left,.right {
  width: 200px;
  height: 100vh;
  background-color: darkseagreen;
}

.center{
  flex: 1;
  background-color: antiquewhite;
}
```

3.利用浮动

```
/*利用浮动，则中间的div必须在最后*/
<div class="outer">
  <div class="left"></div>
  <div class="right"></div>
  <div class="center"></div>
</div>

/*方法三：利用浮动 */
.left {
  width: 200px;
  height: 100vh;
  float: left;
  background-color: darkseagreen;
}

.right {
  width: 200px;
  height: 100vh;
  float: right;
  background-color: darkseagreen;
}

.center {
  background-color: antiquewhite;
  height: 100vh;
}
```

4.Grid布局

建网格布局最强大和最简单的工具。实际高度会根据内容自适应，三栏高度统一。唯一的缺点是兼容性不太好

```
/* 方法五:Grid布局*/
.outer{
  display: grid;
  grid-template-columns: 200px auto 200px;
  grid-template-rows: 100vh;
  width: 100vw;
}

.left, .right{
  background-color: darkseagreen;
}

.center{
  background-color: antiquewhite ;
}
```

6.圣杯布局



步骤总结:

- 设置footer, header的高宽, middle部分要放在content的最前部分
- 设置container的左右边距, 并将其overflow设置为hidden
- 设置left模块, 设置其margin-left为-100%, 拉回来放在最左边; right模块同理
- 解决left, right覆盖未显示, 设置其left, right, 拉回来

```
<div class="header">
  <h4>header</h4>
</div>

<div class="container">
  <div class="middle">
    <h4>middle</h4>
    <p>middle-content</p>
  </div>

  <div class="left">
    <h4>left</h4>
    <p>left-content</p>
  </div>
```

```
<div class="right">
  <h4>right</h4>
  <p>right-content</p>
</div>
</div>
```

```
<div class="footer">
  <h4>footer</h4>
</div>
```

```
/*-----CSS-----
*/
```

```
.header, .footer {
  border: 1px solid #333;
  background: #ccc;
  text-align: center;
}

.container {
  padding-left: 200px;
  padding-right: 220px;
  overflow: hidden;
}

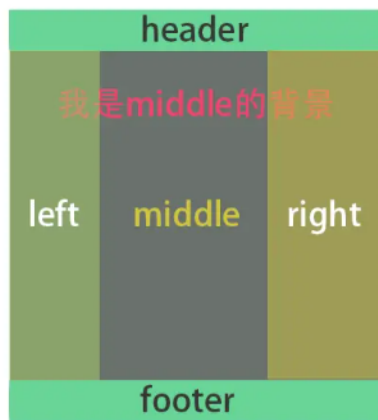
.left, .middle, .right {
  position: relative;
  float: left;
  min-height: 130px;
}

.middle {
  width: 100%;
  background: #9adbd9;
}

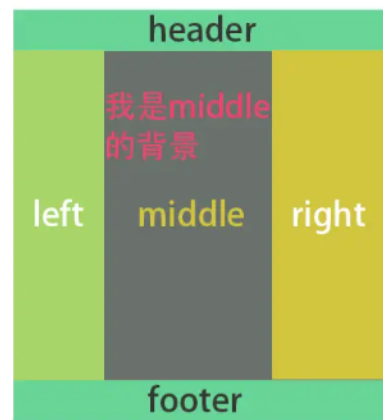
.left {
  margin-left: -100%;
  left: -200px;
  width: 200px;
  background: #eababa;
}

.right {
  margin-left: -220px;
  right: -220px;
  width: 220px;
  background: #eed3a5;
}
```

双飞翼布局



圣杯布局



7.双飞翼布局

同圣杯布局很像，不同点在于外侧的div只包裹middle部分，使用margin而不是padding设置边距

```
<div class="header">
  <h4>header</h4>
</div>

<div class="container">                                /*不同点1: 外侧div只包裹middle模块*/
  <div class="middle">
    <h4>middle</h4>
    <p>middle-
content=====
=====233333</p>
  </div>
</div>

  <div class="left">
    <h4>left</h4>
    <p>left-content</p>
  </div>

  <div class="right">
    <h4>right</h4>
    <p>right-content</p>
  </div>

<div class="footer">
  <h4>footer</h4>
</div>

.container{
  width: 100%;
  float: left;
}
.middle{
  margin: 0 200px;                                /*不同点2: 通过margin设置
左右两侧的边距*/
  background-color: #f5c1af;
  height: 60vh;
```

```

        min-width: 200px;
    }
    .middle p{
        overflow: hidden;
        text-overflow: ellipsis;
        white-space: nowrap;
    }

    .left{
        width: 200px;
        float: left;
        margin-left: -100%;
        background-color: antiquewhite;
        height: 60vh;
    }

    .right{
        width: 200px;
        float: left;
        margin-left: -200px;
        background-color: #cbd999;
        height: 60vh;
    }

    .footer, .header{
        background-color: darkseagreen;
        min-height: 60px;
    }

    .footer{
        clear: both;
    }

```

/*不同点3: 最后需要在 footer中清楚浮动*/

6.水平垂直居中

```

/*方法1: 绝对定位和transform */
.parent {
    position: relative;
}

.child {
    position: absolute;
    left: 50%;
    top: 50%;
    transform: translate(-50%, -50%);
}

/*方法2: 绝对定位和负margin*/
.parent {
    position: relative;
}

.child {
    position: absolute;
    left: 50%;
    top: 50%;
}

```

```

margin-left: -50px;    /*居中元素宽的一半*/
margin-top: -50px
}

/*方法3: flex布局*/
.parent{
  display: flex;
  justify-content: center;
  align-items: center;
}
.child{
  width: 100px;
  height: 100px;
  background-color: antiquewhite;
}

/*方法4: Grid布局*/
.parent{
  display: grid;
}
.child{
  justify-self: center;
  align-self: center
}

```

7. 响应式设计的概念及基本原理

响应式网站设计 (Responsive web design) 是一个网站能够兼容多个终端，而不是为每一个终端做一个特定的版本。

关于原理：基本原理是通过媒体查询 (@media) 查询检测不同的设备屏幕尺寸做处理。

关于兼容：页面头部必须有 meta 声明的 viewport。

```

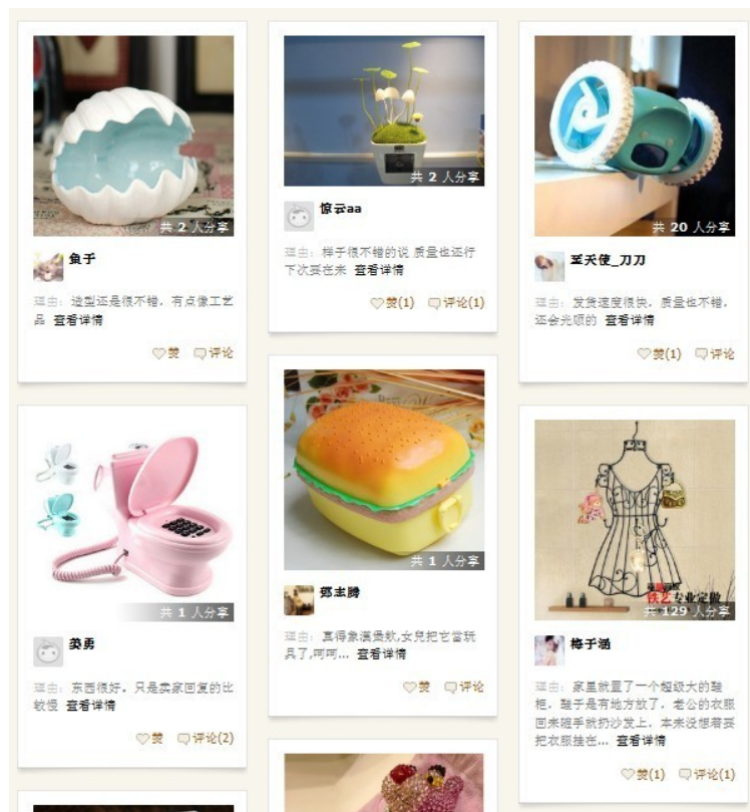
<meta name="viewport" content="width=device-width, initial-
scale="1.0" maximum-scale="1.0, user-scalable=no"/>

```

移动端适配主要有两个维度：

- 适配不同像素密度，针对不同的像素密度，使用 CSS 媒体查询，选择不同精度的图片，以保证图片不会失真；
- 适配不同屏幕大小，由于不同的屏幕有着不同的逻辑像素大小，所以如果直接使用 px 作为开发单位，会使得开发的页面在某一款手机上可以准确显示，但是在另一款手机上就会失真。为了适配不同屏幕的大小，应按照比例来还原设计稿的内容。为了能让页面的尺寸自适应，可以使用 **rem**, **em**, **vw**, **vh** 等相对单位。

8. 瀑布流布局



```
//-----html结构-----
<div class="masonry">
  <!-- 第一列 -->
  <div class="column">
    <div class="item"></div>
    <!-- more items-->
  </div>
  <!-- 第二列 -->
  <div class="column">
    <div class="item"></div>
    <!-- more items-->
  </div>
  <!-- 第三列 -->
  <div class="column">
    <div class="item"></div>
    <!-- more items-->
  </div>
</div>
```

三、定位与浮动

0.外边距塌陷问题？ 如何理解语义化

1.为什么需要清除浮动？ 清除浮动的方式

浮动的定义：非IE浏览器下，容器不设高度且子元素浮动时，容器高度不能被内容撑开。此时，内容会溢出到容器外面而影响布局。这种现象被称为浮动（溢出）。

浮动的工作原理：

- 浮动元素脱离文档流，不占据空间（引起“高度塌陷”现象）

- 浮动元素碰到包含它的边框或者其他浮动元素的边框停留

浮动元素可以左右移动，直到遇到另一个浮动元素或者遇到它外边缘的包含框。浮动框不属于文档流中的普通流，当元素浮动之后，不会影响块级元素的布局，只会影响内联元素布局。此时文档流中的普通流就会表现得该浮动框不存在一样的布局模式。**当包含框的高度小于浮动框的时候，此时就会出现“高度塌陷”。**

浮动元素引起的问题？

- 父元素的高度无法被撑开，影响与父元素同级的元素
- 与浮动元素同级的非浮动元素会跟随其后
- 若浮动的元素不是第一个元素，则该元素之前的元素也要浮动，否则会影响页面的显示结构

清除浮动的方式如下：

- 给父级div定义height属性
- 最后一个浮动元素之后添加一个空的div标签，并添加clear:both样式
- 包含浮动元素的父级标签添加overflow:hidden或者overflow:auto
- 使用 :after 伪元素。由于IE6-7不支持 :after，使用 zoom:1 触发 hasLayout

2.对BFC的理解

- Box: Box 是 CSS 布局的对象和基本单位，一个页面是由很多个 Box 组成的，这个Box就是我们所说的盒模型。
- Formatting context：块级上下文格式化，它是页面中的一块渲染区域，并且有一套渲染规则，它决定了其子元素将如何定位，以及和其他元素的关系和相互作用。

定义：

块格式化上下文（Block Formatting Context，BFC）是Web页面的可视化CSS渲染的一部分，是布局过程中生成块级盒子的区域，也是浮动元素与其他元素的交互限定区域。通俗来讲：BFC是一个独立的布局环境，可以理解为一个容器，在这个容器中按照一定规则进行物品摆放，并且不会影响其它环境中的物品。如果一个元素符合触发BFC的条件，则BFC中的元素布局不受外部影响。

创建BFC的条件：

- 根元素：body；
- 元素设置浮动：float 不是none；
- 元素设置绝对定位：position (absolute、fixed)；
- display 值为：inline-block、table-cell、table-caption、flex等；
- overflow 值不为visible的块元素；
- contain的值为layout、content或paint
- 多列容器，元素的column-count或column-width不为auto，包括column-count为1

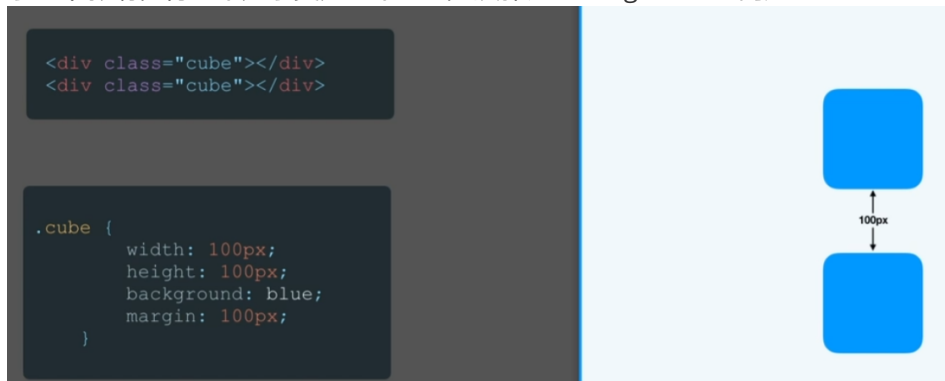
BFC的特点：

- 垂直方向上，自上而下排列，和文档流的排列方式一致。
- 在BFC中上下相邻的两个容器的margin会重叠
- 计算BFC的高度时，需要计算浮动元素的高度
- BFC区域不会与浮动的容器发生重叠
- BFC是独立的容器，容器内部元素不会影响外部元素
- 每个元素的左margin值和容器的左border相接触

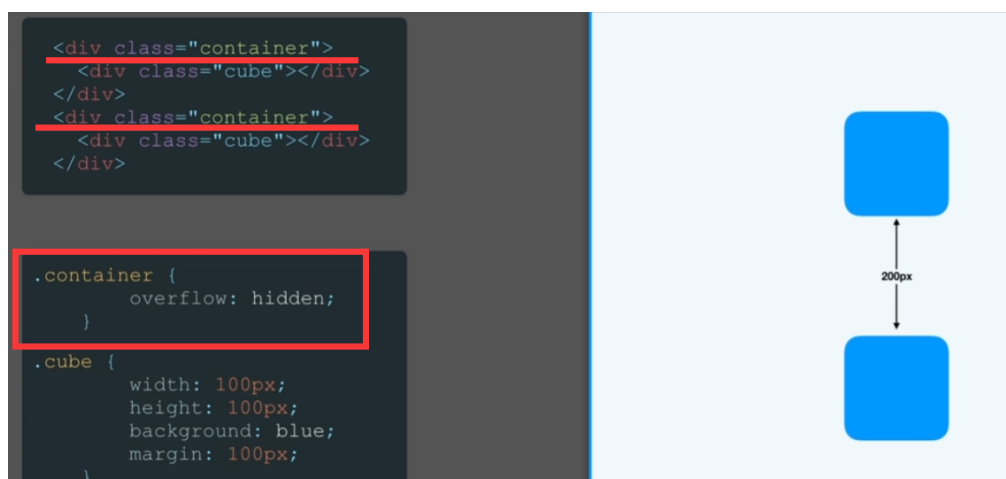
BFC的作用：

margin重叠问题：两个块级元素的上外边距和下外边距可能会合并（折叠）为一个外边距，其大小会取其中外边距值大的那个，这种行为就是外边距折叠。需要注意的是，浮动的元素和绝对定位这种脱离文档流的元素的外边距不会折叠。重叠只会出现在垂直方向

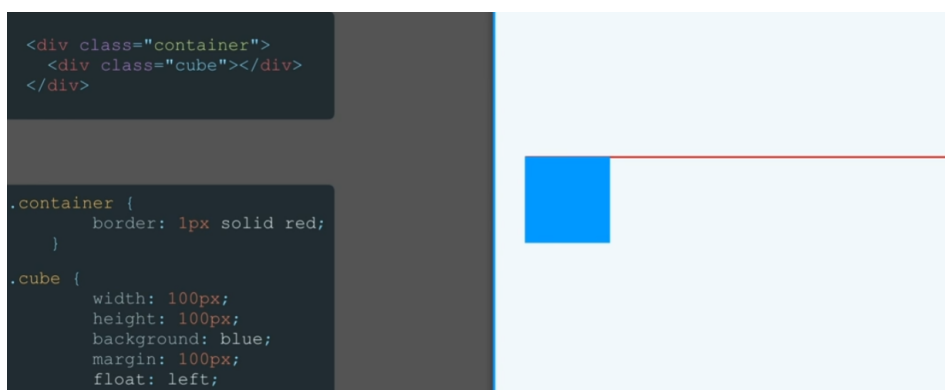
- 解决margin的边距重叠问题：由于BFC是一个独立的区域，内部的元素和外部的元素互不影响，将两个元素变为两个BFC，就解决了margin重叠的问题。



块元素的上外边距margin-top和下外边距margin-bottom会合并为单个边距，单个边距是其中的最大值，这种情况称之为**外边距重叠现象。为了解决这种问题，可以将两个块元素放置在不同的BFC中，可实现互不干扰的效果：



- 解决高度塌陷的问题：在对子元素设置浮动后，父元素会发生高度塌陷，也就是父元素的高度变为0。解决这个问题，只需要把父元素变成一个BFC。常用的办法是给父元素设置 `overflow: hidden`。如下图，`.cube`的div向左浮动，脱离了文档流，导致其父元素div中的内容为空，发生了高度塌陷



- 防止元素被浮动元素覆盖：两个div中，一个设置了浮动，则浮动的div元素将会覆盖在正常的div上。可以给正常的div元素设置BFC属性而免受影响

- 创建自适应两栏布局：可以用来创建自适应两栏布局：左边的宽度固定，右边的宽度自适应

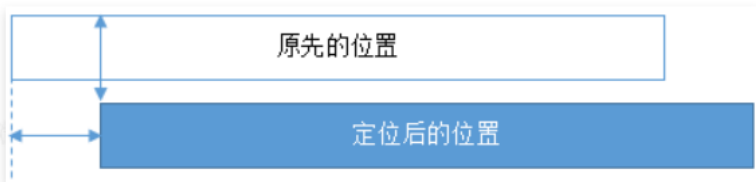
3.元素的层叠问题（没仔细看）

4.position的属性有哪些，区别是什么

属性值	概述
absolute	生成绝对定位的元素，相对于static定位以外的一个父元素进行定位。元素的位置通过left、top、right、bottom属性进行规定。
relative	生成相对定位的元素，相对于其原来的位置进行定位。元素的位置通过left、top、right、bottom属性进行规定。
fixed	生成绝对定位的元素，指定元素相对于屏幕视口（viewport）的位置来指定元素位置。元素的位置在屏幕滚动时不会改变，比如回到顶部的按钮一般都是用此定位方式。
static	默认值，没有定位，元素出现在正常的文档流中，会忽略 top, bottom, left, right 或者 z-index 声明，块级元素从上往下纵向排布，行级元素从左向右排列。
inherit	规定从父元素继承position属性的值

前三者的定位方式如下：

- relative：元素的定位永远是相对于元素自身位置的，和其他元素没关系，也不会影响其他元素。



- fixed：元素的定位是相对于 window（或者 iframe）边界的，和其他元素没有关系。但是它具有破坏性，会导致其他元素位置的变化。



- absolute：元素的定位相对于前两者要复杂许多。如果为 absolute 设置了 top、left，浏览器会根据什么去确定它的纵向和横向的偏移量呢？答案是浏览器会递归查找该元素的所有父元素，如果找到一个设置了position:relative/absolute/fixed的元素，就以该元素为基准定位，如果没找到，就以浏览器边界定位。如下两个图所示：



5.absolute与fixed共同点与不同点

共同点:

- 改变行内元素的呈现方式, 将display置为inline-block
- 使元素脱离普通文档流, 不再占据文档物理空间
- 覆盖非定位文档元素

不同点:

- absolute与fixed的根元素不同, absolute的根元素可以设置, fixed根元素是浏览器。
- 在有滚动条的页面中, absolute会跟着父元素进行移动, fixed固定在页面的具体位置。

6.display、float、position的关系

1. 绝对定位、浮动、根元素都需要调整 display
2. 可以把它看作是一个类似优先级的机制, "position:absolute"和"position:fixed"优先级最高, 有它存在的时候, 浮动不起作用, 'display'的值也需要调整;
3. 元素的'float'特性的值不是"none"的时候或者它是根元素 的时候, 调整'display'的值;

7. 对 sticky 定位的理解

sticky 英文字面意思是粘贴, 所以可以把它称之为粘性定位。语法: position: sticky; 基于用户的滚动位置来定位。

粘性定位的元素是依赖于用户的滚动, 在 position:relative 与 position:fixed 定位之间切换。它的行为就像 position:relative; 而当页面滚动超出目标区域时, 它的表现就像 position:fixed; 它会固定在目标位置。元素定位表现为在跨越特定阈值前为相对定位, 之后为固定定位。这个特定阈值指的是 top, right, bottom 或 left 之一, 换言之, 指定 top, right, bottom 或 left 四个阈值其中之一, 才可使粘性定位生效。否则其行为与相对定位相同。(固定头部导航栏)

四、场景应用

1.实现“品”字布局

```
<div class="div1">1</div>
<div class="div2">2</div>
<div class="div3">3</div>
```

/*方法一：必须知道元素的高宽*/

```
div{
    width: 200px;
    height: 200px;
    font-size: 40px;
    color: white;
    text-align: center;
    line-height: 200px;
}
```

```
.div1{
    background-color: darkseagreen;
    margin: 0 auto;
}
```

```
.div2{
    float: left;
    background-color: #cbd999;
    margin-left: 50%;
}
```

```
.div3{
    float: left;
    background-color: cadetblue;
    margin-left: -400px; /*不能使用百分比，此时父元素为body标签，无法确定百分比的具体数值*/
}
```

/*方法2：使用inline-block*/

```
div{
    width:100px;
    height:100px;
    font-size:40px;
    line-height:100px;
    color:#fff;
    text-align:center;
}
```

```
.div1{
    background:red;
    margin:0 auto;
}
```

```
.div2{
    background: green;
    display: inline-block;
    margin-left: 50%;
}
```

```
.div3{
  background: blue;
  display: inline-block;
  margin-left: -200px;
}
```

2.实现一个三角形，扇形，半圆，圆形，梯形

平时在给盒子设置边框时，往往都设置很窄，就可能误以为边框是由矩形组成的。实际上，border属性是由三角形组成的，下面看一个例子：

```
div {
  width: 0;
  height: 0;
  border: 100px solid;
  border-color: orange blue red green;
}
```



可以根据这个特性实现三角形：

```
/*绘制倒三角形*/
#three{
  border: 100px solid transparent;
  height: 0;
  width: 0;
  border-radius: 100px;
  border-top: 50px solid red ;
}

/*绘制扇形*/
#three{
  border: 100px solid transparent;
  width: 0;
  height: 0;
  border-radius: 100px;
  border-top-color: red;
}

/*绘制半圆*/
#three{
  background-color: red;
  height: 50px;
  width: 100px;
  border-radius: 0 0 100px 100px;
}
/* 相当于两个扇形拼在一起*/

/*绘制圆形*/
#three{
  background-color: red;
```



```

        height: 200px;
        width: 200px;
        border-radius: 50%;
    }

    /*绘制直角梯形*/
    #three {
        height: 0;
        width: 100px;
        border-bottom: 100px solid red;
        border-right: 40px solid transparent;
    }

    /*绘制等腰梯形*/
    #three{
        height:0;
        width:100px;
        border-width:0 40px 100px 40px;
        border-style:none solid solid;
        border-color:transparent transparent red;
    }

```

3.画一条0.5px的线

直接缩放0.5倍

```

#three{
    background: black;
    height: 1px;
    width: 100%;
    transform: scale(0.5,0.5);    /*scaleY(0.5)*/
}

<meta name="viewport" content="width=device-width, initial-
scale=0.5, minimum-scale=0.5, maximum-scale=0.5"/>

```

这样就能缩放到原来的0.5倍，如果是1px那么就会变成0.5px。viewport只针对于移动端，只在移动端上才能看到效果

4.设置小于12px的字体

在谷歌下css设置字体大小为12px及以下时，显示都是一样大小，都是默认12px。

解决办法：

- 使用Webkit的内核的-webkit-text-size-adjust的私有CSS属性来解决，只要加了-webkit-text-size-adjust:none;字体大小就不受限制了。但是chrome更新到27版本之后就不可以用了。所以高版本chrome谷歌浏览器已经不再支持-webkit-text-size-adjust样式，所以要使用时候**慎用**。
- 使用css3的**transform缩放属性** -webkit-transform:scale(0.5); 注意 -webkit-transform:scale(0.75); 收缩的是整个元素的大小，这时候，如果是内联元素，必须要将内联元素转换成块元素，可以使用 display: block/inline-block/...;
- 使用图片：如果是内容固定不变情况下，使用将小于12px文字内容切出做图片，这样不影响兼容也不影响美观。

5.如何解决 1px 问题?

1px 问题指的是: 在一些 Retina 屏幕 的机型上, 移动端页面的 1px 会变得很粗, 呈现出不止 1px 的效果。原因很简单——CSS 中的 1px 并不能和移动设备上的 1px 划等号。它们之间的比例关系有一个专门的属性=> **设备像素比** 来描述:

`window.devicePixelRatio` = 设备的物理像素 / CSS 像素。

打开 Chrome 浏览器, 启动移动端调试模式, 在控制台去输出这个 `devicePixelRatio` 的值。这里选中 iPhone6/7/8 这系列的机型, 输出的结果就是2。

解决思路: 伪元素先放大后缩小

为了只缩放 `border` 1px 的粗细, 而保证 `border` 的大小不变。如果直接 `scale(0.5)` 的话 `border` 整体大小也会变成二分之一, 所以先放大 200% (放大的时候 `border` 的粗细是不会被放大的) 再缩放, 就能保持原大小不变了。

在目标元素的后面追加一个 `::after` 伪元素, 让这个元素布局为 `absolute` 之后、整个伸展开铺在目标元素上, 然后把它的宽和高都设置为目标元素的两倍, `border` 值设为 1px。接着借助 CSS 动画特效中的放缩能力, 把整个伪元素缩小为原来的 50%。此时, 伪元素的宽高刚好可以和原有的目标元素对齐, 而 `border` 也缩小为了 1px 的二分之一, 间接地实现了 0.5px 的效果。

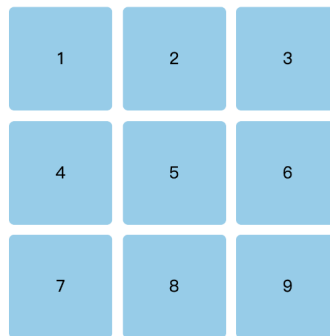
```
#container[data-device="2"] {
  position: relative;
}
#container[data-device="2"]::after{
  position: absolute;
  top: 0;
  left: 0;
  width: 200%;
  height: 200%;
  content: "";
  transform: scale(0.5);
  transform-origin: left top;
  box-sizing: border-box;
  border: 1px solid #333;
}
}
```

3.实现“九宫格”布局

```
<div class="box">
  <ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
    <li>6</li>
    <li>7</li>
    <li>8</li>
    <li>9</li>
  </ul>
</div>
```

```
//-----公共样式
ul {
    padding: 0;
}

li {
    list-style: none;
    text-align: center;
    border-radius: 5px;
    background: skyblue;
}
```



1.flex实现

```
ul {
    display: flex;
    flex-wrap: wrap;
    width: 100%;
    height: 100%;
}

li {
    width: 30%;
    height: 30%;
    margin-right: 5%;
    margin-bottom: 5%;
}

li:nth-of-type(3n){
    margin-right: 0;
    /*由于给每个li元素设置了下边距和右边距，所以最后同一列（3、6、9）的右边距和
    最后一行（7、8、9）的下边距撑大了ul，所以这里使用类型选择器来消除他们的影响。*/
}

li:nth-of-type(n+7){
    margin-bottom: 0;
}
```

2.Grid实现

```
ul {
  width: 100%;
  height: 100%;
  display: grid;
  grid-template-columns: 30% 30% 30%;
  grid-template-rows: 30% 30% 30%;
  grid-gap: 5%;
}
```

/*其中grid-template-columns属性用来设置每一行中单个元素的宽度，grid-template-rows属性用来设置每一列中单个元素的高度，grid-gap属性用来设置盒子之间的间距。*/

3.float实现

```
ul {
  width: 100%;
  height: 100%;
  overflow: hidden; /*设置BFC，消除浮动带来的影响*/
}

li {
  float: left; /*直接让所有li左浮动*/
  width: 30%;
  height: 30%;
  margin-right: 5%;
  margin-bottom: 5%;
}

li:nth-of-type(3n){
  margin-right: 0;
}

li:nth-of-type(n+7){
  margin-bottom: 0;
}
```

4.table实现

给父元素设置为table布局，然后使用border-spacing设置单元格之间的间距，最后将li设置为表格行，将div设置为表格单元格，

```
//-----html结构-----
<ul class="table">
  <li>
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </li>
  <li>
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </li>
```

```

<li>
  <div>7</div>
  <div>8</div>
  <div>9</div>
</li>
</ul>

//-----css样式-----
ul {
  width: 100%;
  height: 100%;
  display: table;           /*设置ul为table布局*/
  border-spacing: 10px;
}

li {
  display: table-row;       /*每个li标签设置为一行*/
}

div {
  width: 30%;
  height: 30%;
  display: table-cell;      /*每一个div是表格单元格*/
  text-align: center;
  border-radius: 5px;
  background: skyblue;
}

```

常见问题

扩充: less, stylus, jade

npm包, 自己封装可视化视图进去!!! 超级加分

Modern.js 现代web工程体系

rollup 专注打包js文件

webpack 打包工程性文件

九月份之前准备面试, 主要是针对自己的简历进行复习:

- 基础部分: 需要牢固掌握的
 - html、css、js
 - 浏览器相关
 - 计算机网络
 - 操作系统
 - 数据库
 - 数据结构以及常见算法题目
- 框架&项目: 经常使用, 但是没有系统总结过的内容
 - vue相关的基础知识、原理以及项目
 - react相关的基础知识、原理以及项目
 - uniapp开发微信小程序
- 实习
 - 做了什么?
 - 学到了什么?

- 加分项：之前接触过，但没有经常使用以及深入了解，后期有时间去了解一下
 - node.js, express框架, koa框架, egg.js框架
 - webpack的的核心概念以及性能优化
 - 项目优化部分

httpbin.org

flex布局实现瀑布流，js判断字符串出现字符最多

version1,2比较大小 手写emj组件，方法组件，滚动