一、基本概念

1、流
2、块级元素和内联（行内）元素
3、width：auto和height：auto
4、盒模型
5、外在盒子和内在盒子
6、css中可继承和不可继承属性
不可继承属性
可继承属性
7、替换元素
8、对css sprites的理解
9、对媒体查询的理解

二、css基础

1、CSS选择器和权重
2、隐藏元素的方法有哪些?
3、display:none 和 visibility:hidden的区别
4、伪元素和伪类的区别和作用
5、display的block、inline、inline-block的区别
6、link和@import的区别
7、transition和animation的区别
8、li和li之间看不见的空白间隔是什么原因引起的？如何解决?
9、为什么有时候用translate来改变位置而不是使用定位position?
10、margin和padding的使用场景
11、对line-height的理解
12、单行、多行文本溢出隐藏
13、z-index在什么情况下会失效

三、css进阶

1、css优化和提高性能的方法有哪些?
2、css预处理器/后处理器是什么？为什么要使用他们?
3、对css工程化的理解
4、如何判断元素是否到达可视区域

四、页面布局

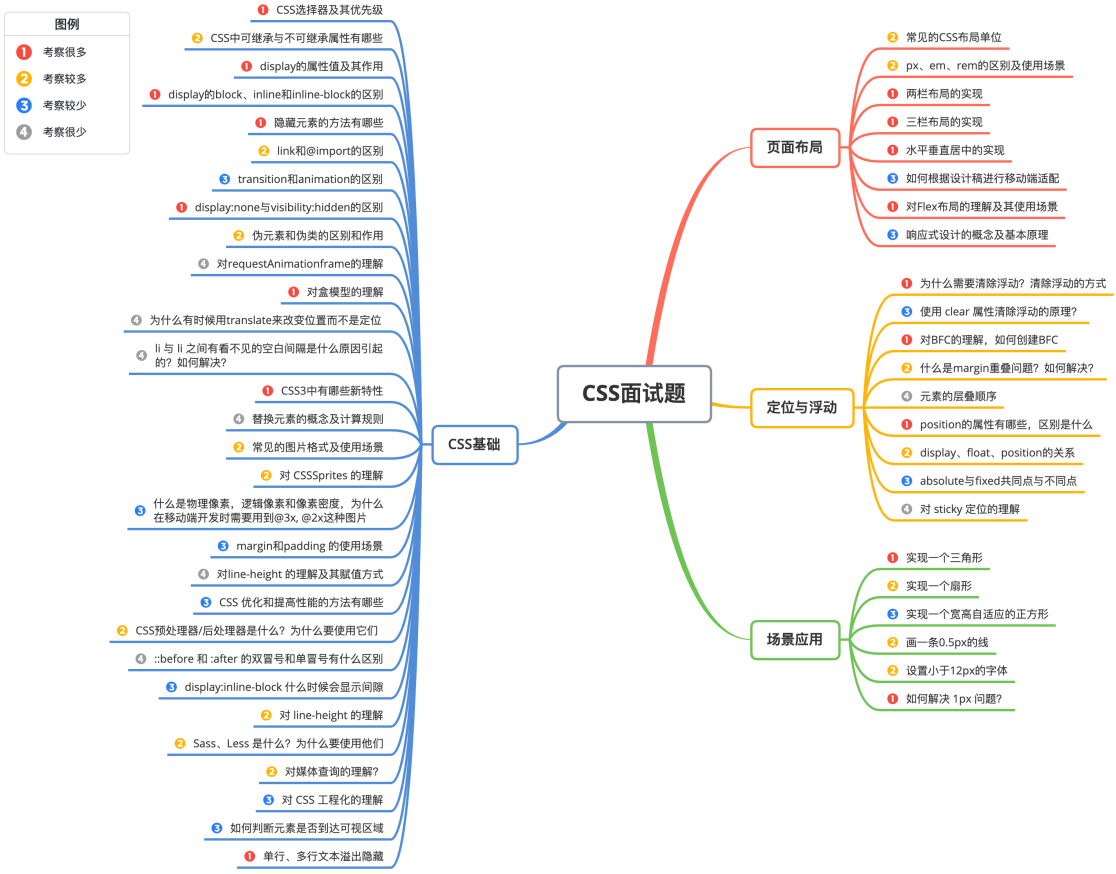
1、常见的页面布局单位
2、两栏布局的实现
3、三栏布局的实现
4、实现水平垂直居中
5、对于flex布局的理解及其使用场景
6、用九宫格浅谈布局
（1）flex实现
（2）grid布局
（3）float布局
（4）inline-block实现
（5）table布局

五、定位与浮动

1、为什么要清除浮动？清除浮动的方式
清除浮动的方式:
2、使用clear属性清除浮动的原理
3、对BFC的理解，如何创建BFC
4. 什么是margin重叠问题？如何解决?
5、元素的层叠顺序
6、position的属性有哪些，区别是什么
7. display、float、position的关系
8、对 sticky 定位的理解
9、 absolute与fixed共同点与不同点

六、场景应用

1、画一个三角形
2、实现一个扇形
3、画一条0.5px的线
4、画一个梯形
5、画一个半圆



一、基本概念

1、流

"流"又叫文档流，是css的一种基本定位和布局机制。

流是HTML的一种抽象概念，暗喻这种排列布局方式好像水流一样自然流动。"流体布局"是html默认的布局机制，如果写的HTML不用css，默认自上而下（块级元素），从左到右（内联元素）堆砌的布局方式。

2、块级元素和内联（行内）元素

块级元素是指单独撑满一行的元素，如 `div、ul、li、table、p、h1` 等。这些元素的默认值是 `block、table、list-item` 等。

行内元素是指只占它对应标签的边框所包含的空间的元素，这些元素如果父元素宽度足够则排列在一行显示，如 `span、a、em、i、img、td` 等。这些元素的默认值是 `inline、inline-block、inline-table、table-cell` 等。

在实际开发中，我们常常把 display计算值为 inline、inline-block、inline-table、table-cell的元素叫做内联元素，而把display计算值为block的元素称为块级元素。

3、width：auto和height：auto

> `width`和`height`的默认值都是`auto`

对于块级元素，流体布局之下`width：auto`自适应撑满父元素的宽度。这里的撑满并不同于`width:100%`的固定宽度，而是像水一样能够根据`margin`不同而自适应父元素的宽度。

对于内联元素，`width:auto`则呈现出包裹性，即由子元素的宽度决定。

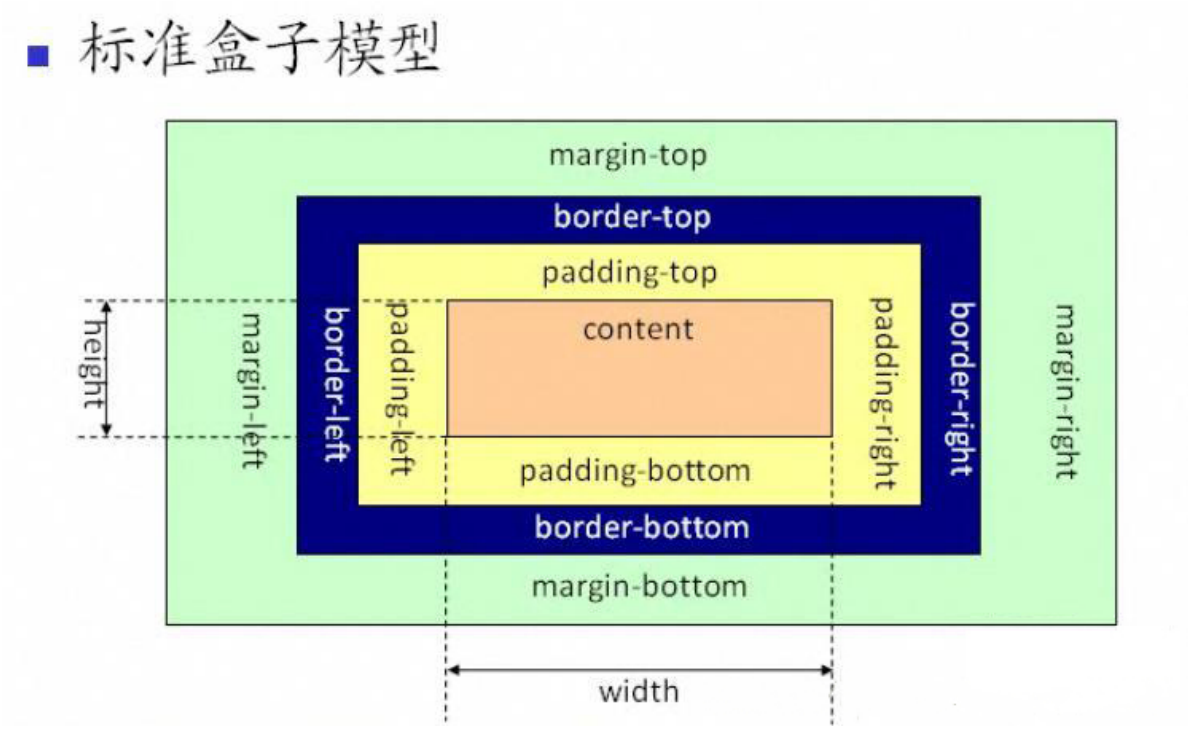
无论是内联元素还是块级元素，`height:auto`则呈现出包裹性，即高度由子级元素撑开。

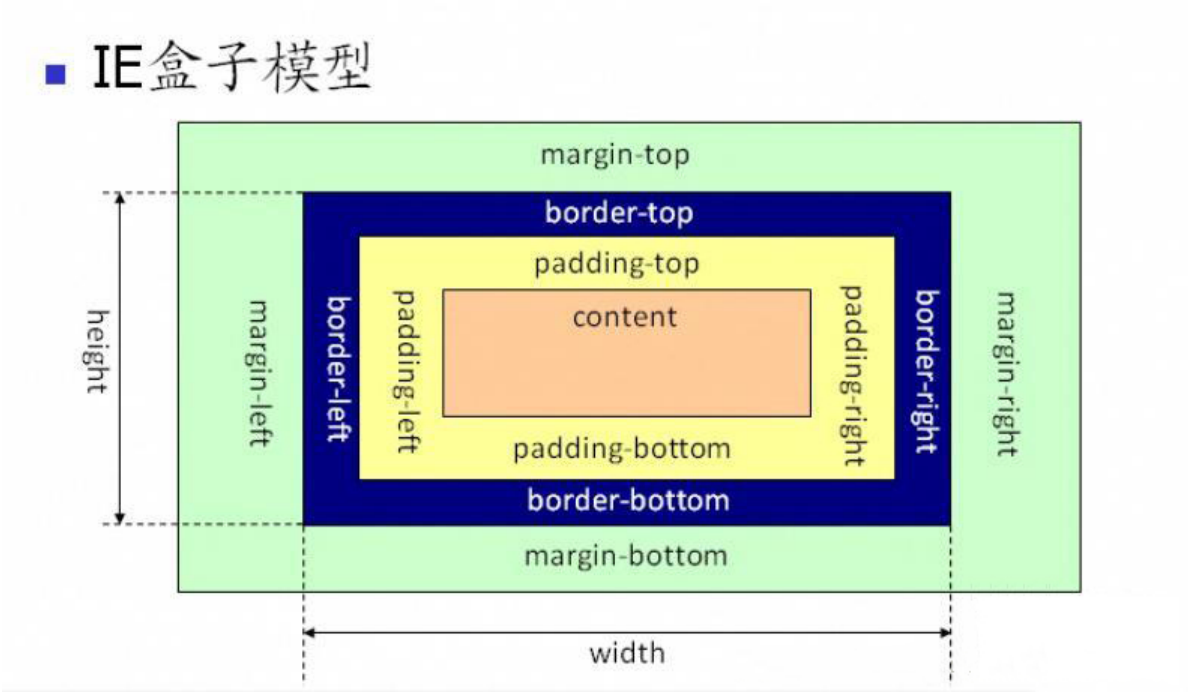
- css可真有趣，正常流下，如果块级元素的`width`是个固定值，`margin`是`auto`，则`margin`会撑满剩下的空间，这也就是水平居中的效果；
- `margin`是固定值，`width`是auto，则width会撑满剩下的空间。这就是流体布局的根本所在。

> 注意：如果父元素`height:auto`会导致子元素`height:100%`百分比失效。如果想要子元素的`height:100%`生效，那么父元素需要设定显示高度；或者使用绝对定位。（绝对定位元素百分比根据padding box计算），非绝对定位元素百分比根据content box计算）

4、盒模型

> css有两种盒模型：标准盒模型和IE盒子模型





盒模型都是由四个部分组成的，分别是margin、border、padding、content

标准盒模型和IE盒模型的区别在于设置width和height时，所对应的范围不同：

- 标准盒子模型：`height=width=content`
- IE盒子模型：`height=width=content+padding+border`

可以通过修改元素的`box-sizing`属性来改变盒模型：

- `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）

5、外在盒子和内在盒子

> 在《css世界》中提出每个元素都有两个盒子（这是对元素结构和内容相关的一种形象的说法）：外在盒子和内在盒子。

外在盒子是决定元素排列方式的盒子，即决定盒子具有块级特性还是内联特性的盒子。外在盒子负责结构布局。

内在盒子是决定元素内部一些属性否生效的盒子。内在盒子负责内容显示。

如`display:inline-table`外在盒子就是inline，内在盒子就是table。外在盒子决定了元素要像内联元素一样并排在一排显示，内在盒子决定了元素可以设置宽高、垂直方向的margin等属性。

6、css中可继承和不可继承属性

不可继承属性

1. `display`：规定元素生成框的类型
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-...`
5. **定位属性**：`float、clear、position、top、right、bottom、left、min-width、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`
8. **页面样式属性**：`size、page-break-before`
9. **声音样式属性**：`pause-before、cue-after`

可继承属性

1. **字体系列属性**：`font-family、font-weight、font-size、font-style`
2. **文本系列属性**：`text-indent、text-align、line-height、word-spacing、letter-spacing、text-transform、color`
3. **元素可见性**：visibility
4. **列表布局属性**：list-style、
5. **光标属性**：cursor

7、替换元素

> 根据"外在盒子"是内联还是块级，我们把元素分为内联元素和块级元素，而根据内容是否可替换，我们把元素分为可替换元素和非替换元素。

替换元素，是指内容可替换的元素。浏览器会根据元素的标签和属性，来决定元素的具体显示内容。其内容不受css视觉格式化模型控制，css渲染模型并不考虑对此内容的渲染，且元素本身一般拥有固定尺寸（宽高，宽高比）

例子：

1. **img。**浏览器会根据img元素的src属性加载图片信息并显示，如果进查看HTML代码，只能看到引用地址，而看不到图片的实际内容。
2. 另外，`textarea、select、iframe、video`都是替换元素。这些元素往往没有实际的内容，即是一个空元素，有些元素仅在特点情况下被作为可替换元素处理，如：`option、audio、canvas、object`等

除了内容可替换外，还有以下特性：

- **内容的外观不受页面上的css的影响。**用专业的话说就是在样式表 现在css作用域之外。如何更改替换元素本身外观需要类似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情况下，其默认的尺寸（不包括边框）是300*150像素
- **在很多css属性上有自己的默认样式。**比如`vertical-align`属性，默认值是`baseline`，是基线的意思，但替换元素这个属性的默认值是`bottom`，定义为元素的下边缘。
- **所有替换元素都是内联水平元素。**也就是替换元素和替换元素、替换元素和替换文字都是可以在一行显示的。默认`display`的值是`inline`或`inline-block`。

替换元素的尺寸由内而外分为三类：

- **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。
- **HTML尺寸**：只能通过HTML原生属性改变，这些HTML原生属性包括width和height属性、size属性。、
- **css尺寸**：特指可以通过css的width和和height或者max-width/max-width和 max-height/min-height设置尺寸，对应盒尺寸中的content box。

这三层结构的计算规则：

固有尺寸》HTML尺寸》css尺寸》300*150像素

8、对css sprites的理解

css sprites（精灵图），将一个页面涉及到的所有图片都包含到一张大图中去，然后利用css 的`background-image`、`background-repeat`、`background-position`属性的组合进行背景定位。也被称为 “雪碧图”

适合用于导航的背景图、按钮小图标等。

优点：通过整合图片，减少对服务器的请求数量，减少图片的体积从而减轻服务器的负担，提高网页的加载速度。

9、对媒体查询的理解

媒体查询是由一个可选的媒体类型和零个或多个使用媒体功能的限制了样式表范围的表达式组成，例如宽度、高度和颜色。媒体查询，添加自css3，允许内容呈现针对一个特定范围的输出设备而进行裁剪，而不必改变内容本身，适合web网页应对不同型号的设备做出对应的响应适配。

媒体查询包含一个可选的媒体类型和满足css3规范的条件下，包含零个或多个表达式，这些表达式描述了媒体特征，最终会被解析为true或false。如果媒体查询中指定的媒体类型匹配展示文档所使用的设备类型，并且所有的表达式的值都是true，那么该媒体查询的结果为true，那么该媒体查询结果为true。那么媒体查询内的样式将会生效。

```
<!--link元素中的css媒体查询-->
<link rel="stylesheet" media="(max-width:800px)" href="aaa.css"/>
<!--样式表中的css媒体查询-->
<style>
    @media (max-width:600px){
        .aaa{
            display:none;
        }
    }
</style>
```

简单来说，使用@media查询，可以针对不同的媒体类型定义不同的样式。@media可以针对不同的屏幕尺寸设置不同的样式，特别是需要设计响应式的页面，@media是非常有效的。当重置浏览器大小的过程中，页面也会根据浏览器的宽度和高度重新渲染页面。

二、css基础

1、CSS选择器和权重

css选择器权重表：

权重	选择器
10000	!important（他并不是选择器，但是权重却是最高的）
1000	内联样式 `style=''`
100	id选择器 `#idname{...}`
10	类选择器 `.classname{...}` 伪类选择器 `:hover{...}` 属性选择器 `[type="text"]={...}`
1	标签选择器 `div{...}` 伪元素选择器 `:after{...}`
0	通用选择器（*） 子选择器（>） 相邻选择器（+） 兄弟选择器（~）

在css中，`!important`的权重相当高，但是由于宽高会被`max-width/min-width`覆盖，所以有时`!important`会失效。

```
width:100px !important;
min-width:200px;
```

上面代码计算之后会被引擎解析成：

```
width:200px;
```

2、隐藏元素的方法有哪些?

1. `display:none`：渲染树不会包含该渲染对象，因此该元素不会在页面中占据位置，也不会响应绑定的对象。
2. `visibility:hidden`：元素在页面中仍占据空间，但是不会响应绑定的监听事件。
3. `opacity:0`：将元素的透明度设置为0，以此来实现元素的隐藏。元素在页面中仍然占据空间，并能够响应元素绑定的监听事件。
4. `position:absolute`：通过使用绝对定位将元素移除可视区域内，以此来实现元素的隐藏。
5. `z-index:负值`：来使其他元素遮盖该元素，以此来实现隐藏。
6. `clip/clip-path`：使用元素裁剪的方法来实现元素的隐藏。元素仍占据页面中的位置，但是不会影响绑定的监听事件。
7. `transform:scale(0,0)`：将元素缩放为0，来实现元素的隐藏。元素仍在页面中占据位置，但是不会响应绑定的监听事件。

3、display:none 和 visibility:hidden的区别

两个属性都是让元素隐藏，不可见。**两者区别如下**：

（1）在渲染树中是否渲染

- `display:none`会让元素完全从dom树中消失，渲染时不会占据任何空间。
- `visibility:hidden`不会让元素 从dom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`。设置`visibility:visible`可以让子孙节点显示。

（3）修改常规文档流中元素的`display`通常会造成文档的重排（回流），但是修改`visibility`属性只会造成本元素的重绘

（4）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。

（5）`display:none`会影响css3的`transition`过渡效果，`visibility:hidden`不会

4、伪元素和伪类的区别和作用

- **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。他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。

```
p::before{content:"第一个";}
p::after{content:"hot!";}
p::first-line{background:red;}
p::first-letter{font-size:90px;} /*第一个字的大小*/
```

- **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。

```
a:hover{color:green;}
p:first-child{color:red;}
```

> 总结：伪类是通过在元素选择器上加入伪类改变元素状态，而伪元素通过对元素的操作进行对元素的改变。

5、display的block、inline、inline-block的区别

（1）block：会独占一行，多个元素会另起一行，可以设置width、height、margin和padding属性。

（2）inline：元素不会独占一行，设置width、height属性无效。但可以设置水平方向的margin和padding属性，不能设置垂直方向的margin和padding。

（3）inline-block：将对象设置为inline，但对象的内容作为block对象呈现，之后的内联对象会被排列在同一行内。

6、link和@import的区别

两者都是外部引用css的方式，区别如下：

1. `link`是XHTML标签，除了加载css外， 还可以定义RSS（简易信息聚合）等其他事务；`@import`属于css范畴，只能加载css。
2. `link`引用css时，在页面载入时同时加载；`@import`需要页面内容完全载入以后加载。
3. `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。
4. `link`支持使用JavaScript控制dom去改变样式；而`@import`不支持。

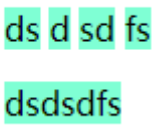
7、transition和animation的区别

- **transition是过渡属性**，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。
- **animation是动画属性**，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完成动画。

8、li和li之间看不见的空白间隔是什么原因引起的？如何解决？

我们通常会将导航栏一行放置，从而会设置li为 `inline-block`。而且为了代码美观，我们通常会一个li标签放一行。但是这导致 `li标签`产生换行字符，它就变成了一个空格，占用了一个字符的宽度。

```
<style>
    ul{
        width: 100px;
    }
    li{
        background-color: aquamarine;
        display: inline-block;
    }
</style>
<body>
    <ul>
        <li>ds</li>
        <li>d</li>
        <li>sd</li>
        <li>fs</li>
    </ul>
    <ul>
        <li>ds</li><li>d</li><li>sd</li><li>fs</li>
    </ul>
</body>
```



解决方法:

1. 为 `<li>` 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 `<li>` 写在同一行。不足：代码不美观。
3. 将 `<ul>` 内的字符尺寸直接设为0，即 `font-size:0`。不足： `<ul>` 中的其他字符尺寸也被设置为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 `<ul>` 的字符间隔 `letter-spacing:-8px`。不足：这也设置了 `<li>` 内的字符间隔。因此需要将 `<li>` 内的字符间隔设为默认 `letter-spacing:normal`

9、为什么有时候用translate来改变位置而不是使用定位position?

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（回流reflow）或重绘（repaint），只会触发复合（compositions）。而改变**绝对定位会触发回流，进而触发重绘和复合。**

transform使浏览器为元素创建一个GPU图层，但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**，绝对定位不会发生这种情况。

10、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。（上下相连的两个盒子之间的空白，需要相互抵消时）
- 需要在border内侧添加空白，且空白处需要背景（色）时，使用padding。（上下相连的两个盒子之间的空白，希望等于两者之和时）。

11、对line-height的理解

（1）line-height的概念

- line-height指一行文本的高度，包括了字间距，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

12、单行、多行文本溢出隐藏

- **单行文本溢出**：

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

- **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：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。

13、z-index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；
- 元素没有设置position属性为非static属性。解决：设置该元素的position属性为relative，absolute或是fixed中的一种；
- 元素在设置z-index的同时还设置了float浮动。解决：float去除，改为display：inline-block；

三、css进阶

1、css优化和提高性能的方法有哪些?

加载性能：

1. **css压缩**：将写好的css进行压缩，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的时候一起加载，前者是等待页面加载完成之后再进行加载。

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选择器是从左到右进行匹配的。当使用后代选择器时，浏览器会遍历所有子元素来确定是否是指定的元素等。
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**标签选择器。**过滤掉无关的规则（这样样式系统就不会浪费时间去匹配他们了）
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**标签选择器**进行选择，而是使用**class选择器**
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。

渲染性能：

1. 慎重使用高性能属性：浮动，定位
2. 尽量减少页面重排、重绘
3. 去除空规则：{}。空规则产生的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.XX，可以忽略小数点前的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，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，但是同时图片本身会变大，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。

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。

2、css预处理器/后处理器是什么？为什么要使用他们?

预处理器：如 `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。

其它 `css` 预处理器语言：`Sass（Scss）`，`Less`，`Stylus`，`Turbine`，`Switch css`，`CSS Cacheer`，`DT Css`。

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

3、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;
- 重要的工程化插件： PostCss;
- Webpack loader 等 。

基于这三个方向，可以衍生出一些具有典型意义的子问题，这里我们逐个来看：

（1）预处理器：为什么要用预处理器？它的出现是为了解决什么问题?

预处理器，其实就是 CSS 世界的“轮子”。预处理器支持我们写一种类似 CSS、但实际并不是 CSS 的语言，然后把它编译成 CSS 代码：

那为什么写 CSS 代码写得好好的，偏偏要转去写"类 CSS"呢？这就和本来用 JS 也可以实现所有功能，但最后却写 React 的 jsx 或者 Vue 的模板语法一样——为了爽！要想知道有了预处理器有多爽，首先要知道的是传统 CSS 有多不爽。随着前端业务复杂度的提高，前端工程中对 CSS 提出了以下的诉求：

1. 宏观设计上：我们希望能优化 CSS 文件的目录结构，对现有的 CSS 文件实现复用；
2. 编码优化上：我们希望能写出结构清晰、简明易懂的 CSS，需要它具有一目了然的嵌套层级关系，而不是无差别的一铺到底写法；我们希望它具有变量特征、计算能力、循环能力等等更强的可编程性，这样我们可以少写一些无用的代码；
3. 可维护性上：更强的可编程性意味着更优质的代码结构，实现复用意味着更简单的目录结构和更强的拓展能力，这两点如果能做到，自然会带来更强的可维护性。

这三点是传统 CSS 所做不到的，也正是预处理器所解决掉的问题。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：PostCss 是如何工作的？我们在什么场景下会使用 PostCss?

PostCss 仍然是一个对 CSS 进行解析和处理的工具，它会对 CSS 做这样的事情：

它和预处理器的不同就在于，预处理器处理的是 类CSS，而 PostCss 处理的就是 CSS 本身。Babel 可以将高版本的 JS 代码转换为低版本的 JS 代码。PostCss 做的是类似的事情：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 能处理 CSS 吗？如何实现？

Webpack 能处理 CSS 吗：

- **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；
- Webpack 在 loader 的辅助下，是可以处理 CSS 的。

如何用 Webpack 实现对 CSS 的处理：

- Webpack 中操作 CSS 需要使用的两个关键的 loader：css-loader 和 style-loader
- 注意，答出“用什么”有时候可能还不够，面试官会怀疑你是不是在背答案，所以你还需要了解每个 loader 都做了什么事情：
- - 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。

4、如何判断元素是否到达可视区域

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;`

四、页面布局

1、常见的页面布局单位

常见的布局单位包括像素 `px`，百分比`%`，`em`，`rem`，`vh/vw`

（1）像素px是页面布局的基础，一个像素表示终端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：只与硬件设备密度有关，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，从而实现响应式效果。一般子元素的百分比相对于直接父元素。

（3）em和rem：相对于px更灵活，他们都是相对长度单位，他们之间的区别：**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。如果当前行内文字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）。（相对于父元素字体大小的倍数）
- **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元素中字体的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和vh是与视口有关的单位，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v**max**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，视窗的高度时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如（border-radius、translate）等
> - vw/vh：相对于视窗的尺寸

2、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。

（1）利用`float+margin-left`：利用浮动，将左边元素宽度设置为200px，并且设置向左浮动。将右边元素的margin-left设置为200px，宽度为auto（默认为auto，撑满整个父元素）。

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

（2）利用`float+overflow:hidden`：利用浮动，左侧元素设置固定大小，并左浮动，右侧元素设置overflow:hidden。这样右边就触发了BFC，BFC的区域不会与浮动元素发生重叠，所以两侧就不会发生重叠。

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

（3）利用flex布局，将左边元素设置固定宽度，右边元素设置为 `flex:1`

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

（4）利用绝对定位，将父级元素设置为相对定位。左边元素设置为absolute定位，并且宽度设置固定。将右边元素的margin-left设置为左边元素的宽度

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

（5）利用绝对定位，将父级元素设置为相对定位。左边元素宽度设置固定，右边元素设置为绝对定位，左边定位为左边元素宽度，其余方向定位为0

```
.outer {
  position: relative;
  height: 100px;
}
.left {
  width: 200px;
```

```
  background: tomato;
}
.right {
  position: absolute;
  top: 0;
  right: 0;
  bottom: 0;
  left: 200px;
  background: gold;
}
```

3、三栏布局的实现

> 三栏布局一般指的是页面中一共有3栏，**左右两栏宽度固定，中间自适应布局**

（1）利用绝对定位，左右两栏设置为绝对定位，中间设置对应方向大小的margin的值

```
.outer {
  position: relative;
  height: 100px;
}

.left {
  position: absolute;
  width: 100px;
  height: 100px;
  background: tomato;
}

.right {
  position: absolute;
  top: 0;
  right: 0;
  width: 200px;
  height: 100px;
  background: gold;
}

.center {
  margin-left: 100px;
  margin-right: 200px;
  height: 100px;
  background: lightgreen;
}
```

（2）利用flex布局，左右两栏设置固定大小，中间一栏设置为flex:1

```
.outer {
  display: flex;
  height: 100px;
}

.left {
  width: 100px;
  background: tomato;
}
```

```
.right {
  width: 100px;
  background: gold;
}

.center {
  flex: 1;  //中间自适应
  background: lightgreen;
}
```

（3）利用浮动，左右两栏设置固定大小，并设置对应方向的浮动。中间一栏设置左右两个方向的margin值，注意这种方式，**中间一栏必须放到最后(在html页面中)：**

```
.outer {
  height: 100px;
}

.left {
  float: left;
  width: 100px;
  height: 100px;
  background: tomato;
}

.right {
  float: right;
  width: 200px;
  height: 100px;
  background: gold;
}

.center {
  height: 100px;
  margin-left: 100px;
  margin-right: 200px;
  background: lightgreen;
}
```

（4）圣杯布局。利用浮动和负边距来实现。父级元素设置左右的 padding，三列均设置向左浮动，中间一列放在最前面，宽度设置为父级元素的宽度，因此后面两列都被挤到了下一行，通过设置 margin 负值将其移动到上一行，再利用相对定位，定位到两边。

```
.main{
    padding: 0 100px;  /*设置左右内边距为左右部分的宽度*/
}
/*清除浮动,防止高度塌陷*/
.main::after {
    content: '';
    display: block;
    clear: both;
}
.center,.left,.right{
    height: 229px;
    float: left;
    text-align: center;
```

```
    color: aliceblue;
    background-color: cadetblue;
}
.left,.right{
    position: relative;
    width: 100px;
    background-color: rgb(41, 48, 45);
}
.left{
    left: -100px;   /*结合相对定位，偏移移原来的位置到左边，不然会覆盖中间元素的内容*/
    margin-left: -100%;
}
.center{
   width: 100%;
}
.right{
   right: -100px;
   margin-left: -100px;
}
```

（5）双飞翼布局：双飞翼布局给主面板添加了一个父标签来通过margin给子标签腾出空间。

双飞翼布局相对于圣杯布局来说，左右位置的保留是通过中间列的margin值来实现的，而不是通过父元素的padding来实现的。本质上说，也是通过浮动和外边距负值来实现的。

```
<style>
        .centerfather{
            width: 100%;
        }
        /*清除浮动,防止高度塌陷*/
        .main::after {
            content: '';
            display: block;
            clear: both;
        }
        .centerfather,.left,.right{
            height: 229px;
            float: left;
            text-align: center;
            color: aliceblue;
            background-color: cadetblue;
        }
        .left,.right{
            width: 100px;
            background-color: rgb(41, 48, 45);
        }
        .left{
            margin-left: -100%;
        }
        .center{
            margin: 0 100px;
        }
        .right{
            margin-left: -100px;
        }
</style>
</head>
```

```
<body>
    <div class="main">
        <div class="centerfather">
            <div class="center"></div>
        </div>
        <div class="left"></div>
        <div class="right"></div>
    </div>
</body>
</html>
```

4、实现水平垂直居中

（1）absolute + translate。该方法需要考虑浏览器的兼容问题。

先将元素的左上角通过top:50%和 left:50%定位到页面的中心，然后通过translate来调整元素的中心点到页面的中心。

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

（2）absolute + margin。 该方法适用于盒子**有宽高**的情况。

利用绝对定位，设置四个方向的值都为0，并将margin设置为auto，由于宽高固定，因此对应方向实现平分，可以实现水平和垂直方向上的居中。

```
.parent{
    position:relative;
}
.child{
    width:100px;
    height:100px;
    position:absolute;
    top:0;
    bottom:0;
    left:0;
    right:0;
    margin:auto;
}
```

（3）absolute + margin-top。适用于盒子宽高已知的情况。

```
.parent{
    position:relative;
}
.child{
    width:100px;
    height:100px;
    position:absolute;
    top:50%;
    left:50%;
    margin-top:-50px;  /*自身宽高的一半*/
    margin-left:-50px;
}
```

（4）flex + margin:auto

```
.parent{
    display:flex;
}
.child{
    width:100px;
    height:100px;
    margin:auto;
}
```

（5）flex。要考虑兼容问题，该方法在移动端用的较多。

```
.parent{
    display:flex;
    align-items:center;
    justify-content:center;
}
```

5、对于flex布局的理解及其使用场景

flex意味“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。

采用flex布局的元素，称为flex容器，简称“容器”。它的所有子元素自动成为容器成员，称为flex项目（flex item），简称“项目”。容器默认存在两根轴线：水平的主轴（main axis）和垂直的交叉轴（cross axis），项目默认沿水平主轴排列。

以下六个属性设置在 **容器上**：

- flex-direction属性决定主轴的方向（即项目的排列方向）。
- flex-wrap属性定义，如果一条轴线排不下，如何换行。
- flex-flow属性是flex-direction属性和flex-wrap属性的简写形式，默认值为row nowrap。
- justify-content属性定义了项目在主轴上的对齐方式。
- align-items属性定义项目在交叉轴上如何对齐。
- align-content属性定义了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。

以下6个属性设置在**项目上**：

- order属性定义项目的排列顺序。数值越小，排列越靠前，默认为0。
- flex-grow属性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。

- flex-shrink属性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。
- flex-basis属性定义了在分配多余空间之前，项目占据的主轴空间。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。
- flex属性是flex-grow，flex-shrink和flex-basis的简写，默认值为0 1 auto。
- align-self属性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。

简单来说：

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。

6、用九宫格浅谈布局

首先，定义好通用的HTML结构：

```
<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

flex设置九宫格，需要设置一个`flex-wrap:wrap`；使得盒子在该换行的时候换行。

由于给每个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，所以在这里使用类型选择器来消除他们的影响。

```
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;
}

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

优点：简单，灵活，对移动端友好
缺点：浏览器的兼容性不是很好，一些浏览器不是很支持

（2）grid布局

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。

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

优点：代码量简化比较多
缺点：浏览器的兼容性不是很好，一些浏览器不是很支持

（3）float布局

这里首先需要给父元素的div设置一个宽度，宽度值为：**盒子宽 * 3 + 间距 * 2;**然后给每个盒子设置固定的宽高，为了让他换行，可以使用float来实现，由于子元素的浮动，形成了BFC，所以父元素ul使用overflow:hidden；来消除浮动带来的影响。最终的实现代码如下：

```
ul {
  width: 100%;
  height: 100%;
  overflow: hidden;
}

li {
  float: left;
```

```
  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;
}
```

优点：兼容性较好
缺点：需要清除浮动，否则页面的布局会塌陷

（4）inline-block实现

需要注意的是，设置为inline-block的元素之间可能会出现间隙，这里使用了letter-spacing属性来消除这种影响，该属性可以用来增加或减少字符间的空白（字符间距）。使用之后就正常了，出现了预期的效果。也可以给ul设置font-size: 0;来消除盒子之间的字符间距

```
ul {
  width: 100%;
  height: 100%;
  letter-spacing: -10px;
  font-size:0;
}

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

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

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

（5）table布局

HTML：

```
<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>
```

table布局也不算太难，首先给父元素设置为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，CSS样式如下：

```
.table {
  width: 100%;
  height: 100%;
  display: table;
  border-spacing: 10px;
}

li {
  display: table-row;
}

div {
  width: 30%;
  height: 30%;
  display: table-cell;
  text-align: center;
  border-radius: 5px;
  background: skyblue;
}
```

table的特性决定了其适合用来做布局，并且表格中的内容可以自动居中

1. 空间平均划分：子容器默认自动平分宽度的
2. 设置了一个table-cell为固定的宽度之后，其余的自动平分占满父容器

优点：兼容性好，布局相对简单，上手快
缺点：

1. table占用了更多的字节，下载时间延迟，占用服务器更多的资源
2. table会阻挡浏览器渲染引擎的顺序，页面生成速度延迟
3. 灵活性比较差，不易更改
4. 不利于搜索引擎抓取信息，影响网站的排名

五、定位与浮动

1、为什么要清除浮动？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时，就会出现“高度塌陷”。

浮动引起的问题：

- 父元素的高度无法撑开，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：

（1）使用带clear属性的空元素（额外标签法）

在浮动元素后使用一个空元素如 `<div class="clear"></div>` ，并在css中赋予 `.clear{clear:both}` 属性即可清除浮动。

（2）使用css的overflow属性(bfc)（给父元素添加overflow属性）

- 给浮动元素的父级（包含容器）添加 `overflow:hidden` 或者 `overflow:auto` 属性。
- 另外在IE6中还需要触发 `hasLayout` ,例如为父元素设置容器宽高或设置 `zoom:1`

（3）给浮动的元素的容器（父级）添加浮动

给浮动元素的容器也添加上浮动属性即可清除内部浮动，但是这样会使其整体浮动，影响布局，**不推荐使用。**

（4）使用邻接元素处理

什么都不做，给浮动元素后面的元素添加clear属性。

（5）使用css的:after伪元素（给父级元素添加after伪元素）

结合 `:after` 伪元素（代表一个元素之后最近的元素）和IEhack，可以完美兼容当前主流的各大浏览器，这里的IEhack指的是触发hasLayout，给浮动元素的容器增加一个clearfix的class，然后给这个class添加一个:after伪元素实现元素末尾添加一个看不见的块元素来清理浮动。

```
.clearfix:after{
    content:"\200B";
    display:table;
    height:0;
    clear:both;
}
.clearfix{
    *zoom:1;
}
```

（6）给父级元素添加高度

2、使用clear属性清除浮动的原理

使用clear清除浮动，其语法如下：

```
clear:none|left|right|both;
```

如果单看字面意思，`clear:left`是“清除左浮动”，`clear:right`是清除右浮动，实际上，这种解释是有问题的，因为浮动一直还在，并没有清除。

官方对clear的解释：“**元素盒子的边不能和前面的浮动元素相邻**”，对元素设置clear属性是为了避免浮动元素对该元素的影响，而不是清除浮动。

还需要注意clear属性指的是元素盒子的边不能和前面的浮动元素相邻，注意这里“**前面**”，也就是clear属性对“后面的”浮动元素是不闻不问的。考虑到float属性要么是left，要么是right，不可能同时存在，同时由于clear属性对“后面的”浮动元素不闻不问，因此，当`clear:left`有效时，`clear:right`必定无效，也就是此时`clear:left`等同与设置`clear:both`。同样的，`clear:right`如果有效也是等同于设置`clear:both`。由此可见，`clear:left`和`clear:right`这两个声明就没有任何使用的价值，至少在css世界中是如此，**直接使用clear:both**吧。

一般用伪元素的方式清除浮动：

```
.clear::after{
    content:"";
    display:block;
    clear:both;
}
```

clear属性只有块级元素才有效的，而`::after`等伪元素默认都是内联水平，这就是借助伪元素清除浮动影响需要设置display属性值的原因。

3、对BFC的理解，如何创建BFC

块级格式化上下文（block Formatting Context，BFC），是Web页面可视化css渲染的一部分，是布局过程中生成块级盒子的区域，规定了内部如何布局，并且这个区域的子元素不会影响到外面的元素，

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的规则进行物品摆放，并且不会影响其他环境中的物品。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除none以外的值
- 元素设置绝对定位：position（absolute，fixed）
- display的值为：inline-block、table-cell、table-caption、flex、inline-flex等
- overflow值为：hidden、auto、scroll的元素。（也就是值不为visible的元素）

BFC的特点：

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

BFC的作用：

- **解决margin的重叠问题**：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BFC，就解决了margin重叠问题。
- **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`。

- **创建自适应两栏布局**：可以用来床架自适应两栏布局：左边宽度固定，右边宽度自适应

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

左侧设置`float:left`，右侧设置`overflow: hidden`。这样右边就触发了BFC，BFC的区域不会与浮动元素发生重叠，所以两侧就不会发生重叠，实现了自适应两栏布局。

4. 什么是margin重叠问题？如何解决？

问题描述：

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**。

计算原则：

折叠合并后外边距的计算原则如下：

- 如果两者都是正数，那么就去最大者
- 如果是一正一负，就会正值减去负值的绝对值
- 两个都是负值时，用0减去两个中绝对值大的那个

解决办法：

对于折叠的情况，主要有两种：**兄弟之间重叠**和**父子之间重叠**

（1）兄弟之间重叠

- 底部元素变为行内盒子：`display: inline-block`
- 底部元素设置浮动：`float`
- 底部元素的position的值为`absolute/fixed`

（2）父子之间重叠

- 父元素加入：`overflow: hidden`
- 父元素添加透明边框：`border:1px solid transparent`
- 子元素变为行内盒子：`display: inline-block`
- 子元素加入浮动属性或定位

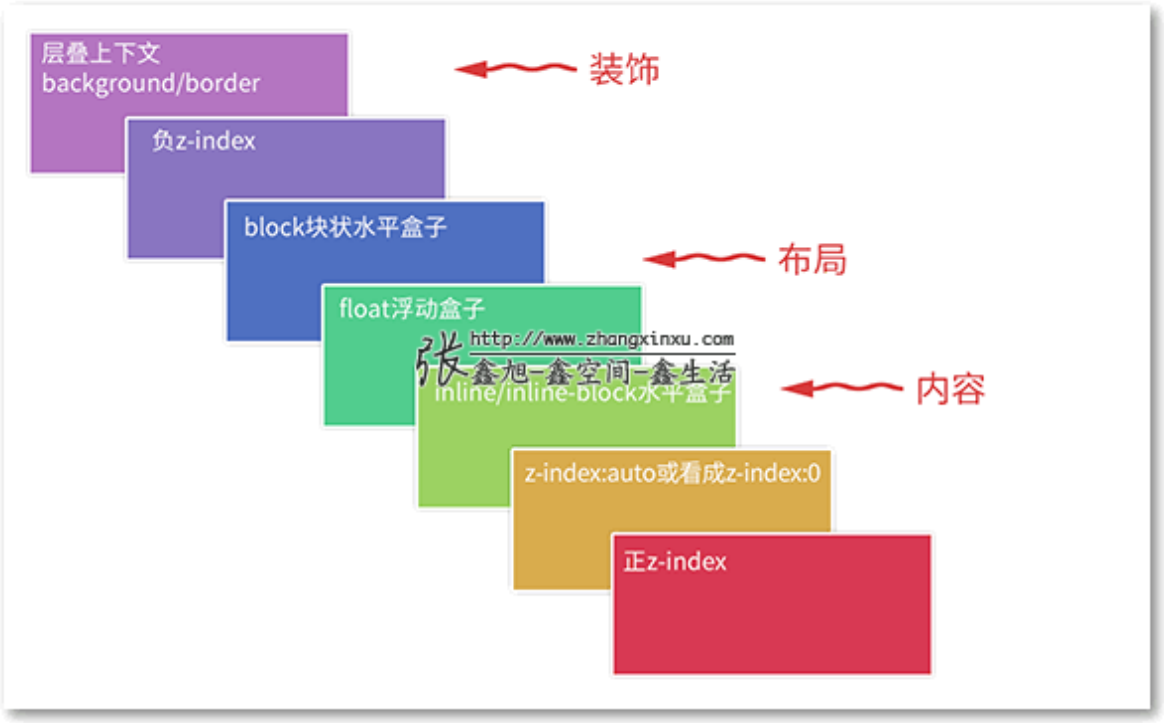
总的来说就是：

1. 利用绝对定位，子绝父相
2. 利用相对定位，相对于原来的位置
3. 利用行内块元素

4. 浮动
5. 利用BFC
6. 设置内边距或边框

5、元素的层叠顺序

层叠顺序，英文称作 stacking order，表示元素发生层叠时有着特定的垂直显示顺序。下面是盒模型的层叠规则：



注意: 当定位元素z-index:auto，生成盒在当前层叠上下文中的层级为 0，不会建立新的层叠上下文，除非是根元素。

6、position的属性有哪些，区别是什么

position有以下属性值：

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

前面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。

7. display、float、position的关系

（1）首先判断display属性是否为none，如果为none，则position和float属性的值不影响元素最后的表现。

（2）然后判断position的值是否为absolute或者fixed，如果是，则float属性失效，并且display的值应该被设置为table或者block，具体转换需要看初始转换值。

（3）如果position的值不为absolute或者fixed，则判断float属性的值是否为none，如果不是，则display的值则按上面的规则转换。注意，如果position的值为relative并且float属性的值存在，则relative相对于浮动后的最终位置定位。

（4）如果float的值为none，则判断元素是否为根元素，如果是根元素则display属性按照上面的规则转换，如果不是，则保持指定的display属性值不变。

总的来说，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；其次，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；最后，非根元素，并且非浮动元素，并且非绝对定位的元素，'display'特性值同设置值。

8、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。

9、 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。

六、场景应用

1、画一个三角形

利用`border`属性来绘制三角形，实际上，border属性是由**三角形**组成的

```
div {
    width: 0;
    height: 0;
    border-bottom: 50px solid red;
    border-right: 50px solid transparent;
    border-left: 50px solid transparent;
}
```

2、实现一个扇形

```
div{
    border: 100px solid transparent;
    width: 0;
    height: 0;
    border-radius: 100px;
    border-top-color: red;
}
```

3、画一条0.5px的线

- **采用transform:scale()的方式**，该方法用来定义元素的2D缩放转换：

```
transform:scale(0.5,0.5);
```

- **采用meta viewport的方式**

```
<meta name="viewpoint" 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、画一个梯形

```
.trapezoid {
  height:0;
  width:100px;
  border-width:0 40px 100px 40px;
  border-style:none solid solid;
  border-color:transparent transparent red;
}
```

5、画一个半圆

```
div {
  background-color: red;
  width: 100px;
  height: 50px;
  border-radius: 0px 0px 100px 100px;
}
```