# CSS

## CSS权重及其引入方式

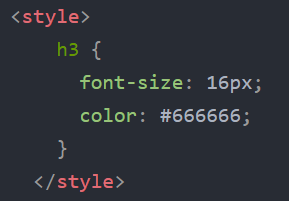
**1.行间样式（行内样式？内联样式？）**

直接在html的style属性中书写：



**2.内部样式**

<head>标签内，用<style>标签定义



**3.外部样式**

<head>标签内通过<link>引入外部样式



**4.权重比较**

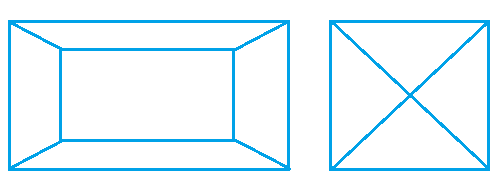
 \*

注：尽管10个id大不过行间样式，但是多一层选择器会令权重变大

1000+100+1 > 1000+1

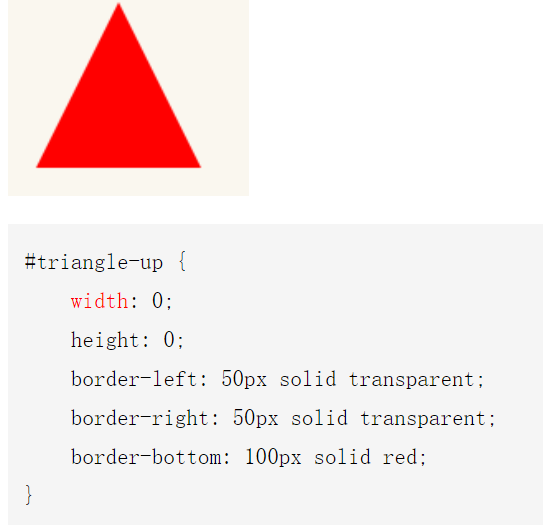
## 用CSS画三角形

随着实体面积减小，最终四个边框会组成一个矩形，取其中一个边框，隐藏其他边框后就可得到三角形。



例如上三角：

只保留下边框，其余透明，去掉top会向上移。



例如斜三角：



想自定义度数：



## 盒子模型及其理解

渲染引擎对HTML元素的布局方式



**height、width：**元素的height、width是content内定义的

IE的盒模型height、width会加上padding和border

**margin合并：**

**（1）兄弟：**两个元素的margin相邻时，都使用两者间最大的margin

**（2）父子：**当父元素没有内容或内容在子元素的后面且没有内边距或没有边框时，子元素的上外边距将和父元素的上外边距合并为一个上外边距，选取两者最大值（要防止父、子元素的上外边距合并，只需在子元素前面设置父元素内容或保持父元素内容不变的情况下添加内边距或添加边框)

## 定位方式及其区别（文档流）

**1. relative（相对定位）**

相对定位的偏移参考元素是元素本身，不会使元素脱离文档流。元素的初始位置占据的空间会被保留。相对定位元素常常作为绝对定位元素的父元素。并且定位元素经常与z-index属性进行层次分级

**2. absolute（绝对定位）**

绝对定位元素以父辈元素中最近的非static定位元素为参考坐标，如果绝对定位元素的父辈元素中没有采用定位的，那么此绝对定位元素的参考对象是html,元素会脱离文档流。就好像文档流中被删除了一样。并且定位元素经常与z-index属性进行层次分级

**3. fixed  (固定定位)**

位移的参考坐标是可视窗口，使用fixed的元素脱离文档流。并且定位元素经常与z-index属性进行层次分级

**4. static （静态定位）**

默认值，元素框正常生成的，top left bottom right这几个偏移属性不会影响其静态定位的正常显示

**5.sticky（粘性定位）**

针对元素top/left等属性，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。

**6.inherited（继承父元素position）**

## margin合并

**1.定义**

当两个元素的margin在垂直方向邻接时，会选取其中一个margin作为公共的margin；正负取和，负取绝对值大，正取大；

父子元素，子元素会和父元素公用一个外边距，相对定位是父元素的父元素。

**2.发生条件**

（1）常规文档流，同一BFC

（2）不被分开，无padding、border分割（父子间）

（3）垂直方向，父子，兄弟

**3.解决**

将兄弟元素外边距距离分别计算，最终距离为两者的和，而不是合并

将父子元素外边距距离分别计算，各有各的距离

**兄弟：（1）两者分开：**中间加个1px高度的元素

**（2）BFC：**给其中一个设置为行内块元素（不能两个都设置，不然就会按行内元素水平排列）

给其中一个外面套一个div，触发其BFC

全部脱离文档流

**（3）父元素：**设置grid布局

**父子：（1）父元素：** 添加padding或：者border（子元素加Border没用）

**（2）BFC：**父或子或同时设置 float（非none就行);

display：inline-block（父子都可），flex（父），grid（父），table（父）;

position（父子都可）: absolute、fixed

overflow（父）: hidden、auto、scroll

## margin塌陷

**1.定义**

在文档流中，父元素的高度默认是被子元素撑开的

但是当子元素设置浮动之后，子元素会完全脱离文档流

此时将会导致子元素无法撑开父元素的高度，导致父元素高度塌陷

**2.解决**

**（1）变为同一个BFC**

触发**父元素**的BFC，具有bfc的元素我们可以抽象的理解成为隔离了的独立容器

bfc的三个特性：

a.阻止外边距折叠

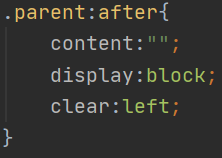
b.可以包含浮动的元素

c.可以阻止元素被浮动元素覆盖

**（2）间接撑**

**1）手动添加：**浮动元素后面加入一个div/br/hr空标签，设置其clear: both（两边都不允许出现浮动元素），就会令空标签移动到浮动元素下方，也就间接地撑开了父元素

**2）伪元素添加：**

****

在父类元素最后面添加（即浮动元素后面），同理；

## IFC与BFC

**1.BFC**

**定义：**BFC（Block Formatting Context）叫做“块级格式化上下文”。BFC的布局规则如下：

**性质：**

1. 对齐：内部的盒子会在垂直方向，一个个地放置；每个元素向左接触对齐
2. 外边距重叠：同一个BFC内才会发生外边距重叠，不同BFC不会发生外边距重叠
3. 隔绝影响：BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素，反之也如此；不会与float元素重叠；

哪些元素会**产生BFC：**

1. 根元素；
2. float的属性不为none；
3. position为absolute或fixed；
4. display为inline-block，table-cell，table-caption，flex；
5. overflow不为visible

**2.IFC**

**定义：**IFC(Inline Formatting Contexts)直译为"内联格式化上下文"。

**性质：**

1. 在一个IFC中，从父级元素的顶部开始，盒子一个接一个横向排列
2. 一个line box总是足够高对于包含在它内的所有盒子。然后，它也许比包含在它内最高的盒子高
3. 当盒子的高度比包含它的line box的高度低，在line box内的盒子的垂直对齐线通过'vertical align'属性决定
4. 当在一行中行内级盒子的总宽度比包含他们的line box的宽度小，他们的在line box中的水平放置位置由'text align'属性决定
5. 当一个行内盒子超过了line box的宽度，则它被分割成几个盒子并且这些盒子被分配成几个横穿过的line boxs

**作用：**

水平居中：当一个块要在环境中水平居中时候，设置其为inline-block则会在外层产生IFC，通过text-align:center则可以使其水平居中。

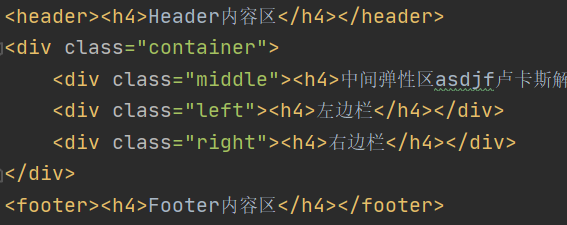
垂直居中：创建一个IFC，用其中一个元素撑开父元素的高度，然后设置其vertical-align:middle,其他行内元素则可以在此父元素下垂直居中。

## 圣杯布局和双飞翼布局的实现

圣杯布局其实和双飞翼布局是一回事。它们实现的都是三栏布局，两边的盒子宽度固定，中间盒子自适应

**1.圣杯布局：**

**（1）HTML：**

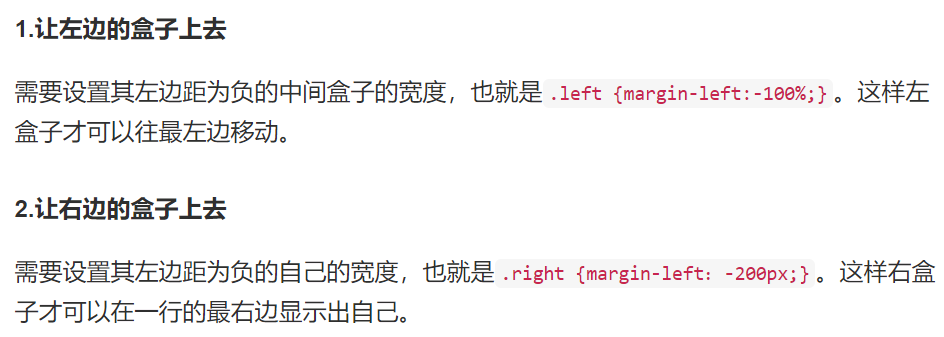
:

**（2）CSS：**

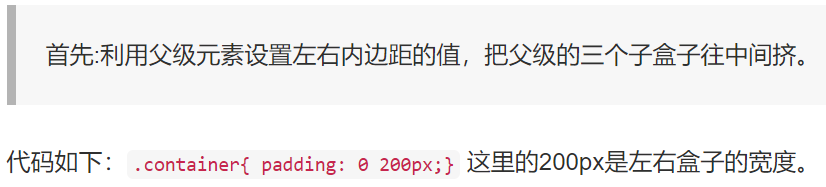
****

三个元素设置浮动，中间100%宽度

**（3）此时左右元素被挤到下方，然后**



**（4）此时中间元素被两边元素覆盖，然后**



**（5）让两边的元素向两边移动**



**2.双飞翼布局**

前面与圣杯布局都一样，只不过对双边元素覆盖中间元素问题上换了解决方式：

（1）给中间元素内部添加div

（2）给该div设置左右外边距margin

## Flex布局

### 1.定义

Flex 是 Flexible Box 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。

### 2.父元素属性

1. **flex-direction**

决定主轴的方向（即项目的排列方向）。

* row（默认值）：主轴为水平方向，起点在左端。
* row-reverse：主轴为水平方向，起点在右端。
* column：主轴为垂直方向，起点在上沿。
* column-reverse：主轴为垂直方向，起点在下沿。

1. **flex-wrap**

换行方式。如果一条轴线排不下，如何换行

nowrap：不换行

wrap：下一行在下面

wrap-reverse：下一行在上面

1. **flex-flow**

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



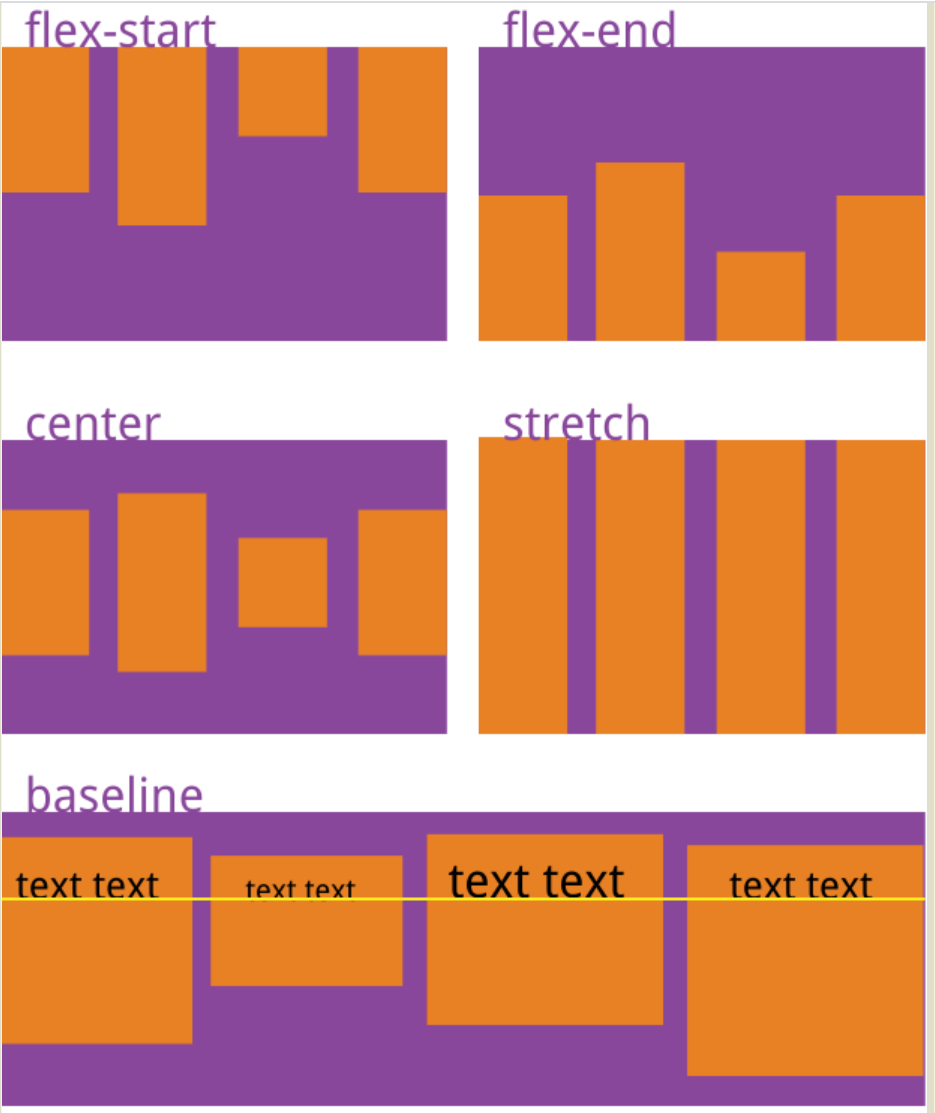
1. **justify-content**

定义了项目在主轴上的对齐方式。

* flex-start（默认值）：左对齐
* flex-end：右对齐
* center： 居中
* space-between：两端对齐，项目之间的间隔都相等。
* space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

1. **align-items**

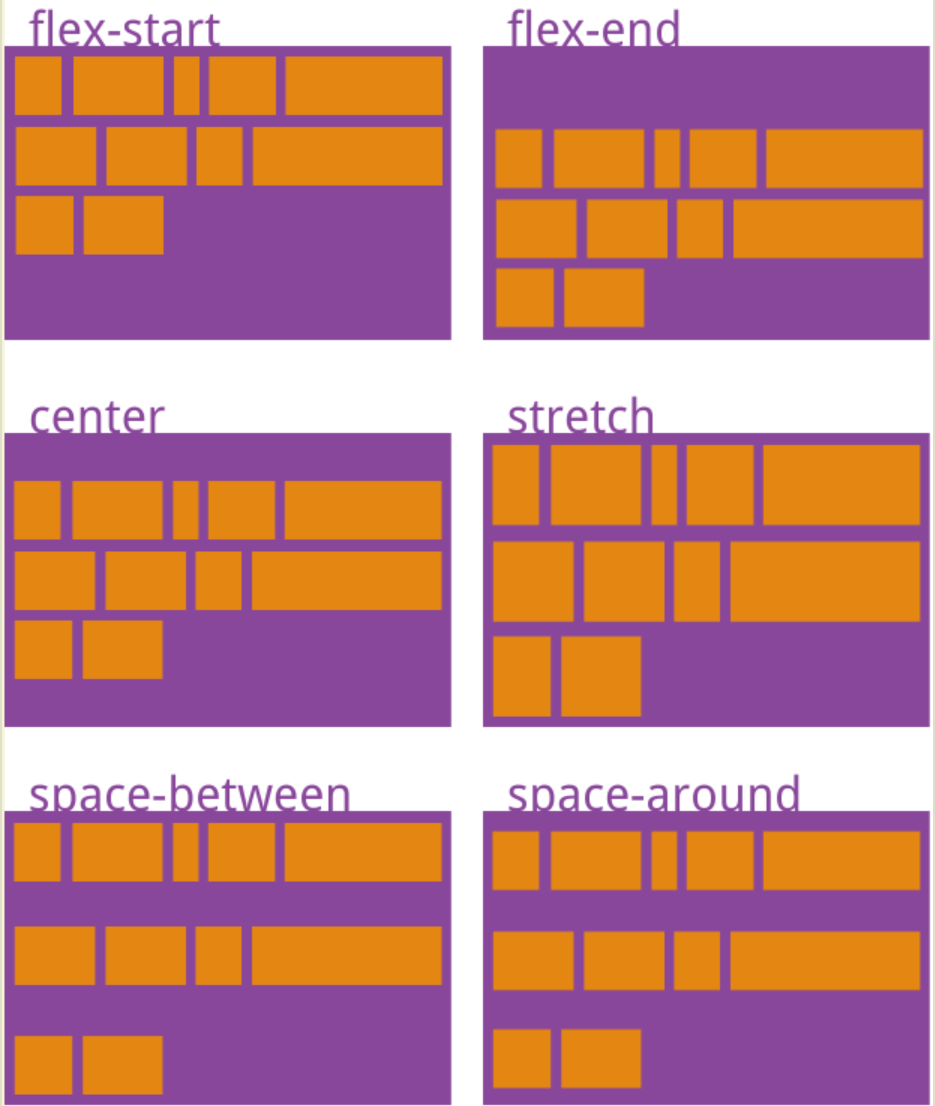
定义项目在交叉轴上如何对齐。



* flex-start：交叉轴的起点对齐。
* flex-end：交叉轴的终点对齐。
* center：交叉轴的中点对齐。
* baseline: 项目的第一行文字的基线对齐。
* stretch（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。

1. **align-content**

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。



* flex-start：与交叉轴的起点对齐。
* flex-end：与交叉轴的终点对齐。
* center：与交叉轴的中点对齐。
* space-between：与交叉轴两端对齐，轴线之间的间隔平均分布。
* space-around：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。
* stretch（默认值）：轴线占满整个交叉轴。

### 3.子元素属性

**（1）order**

定义子元素位置，数值越小越靠前

**（2）flex-grow**

定义当前行所占剩余空间的比例，每个元素与其他元素相比较，之后进行分配。默认0，不放大

**（3）flex-shrink**

定义对不足空间的缩小比列。默认1，空间不足就缩小

**（4）flex-basic**

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。默认auto，原本大小

**（5）flex**

缩写：flex-grow,、flex-shrink 、 flex-basis

默认: 0 1 auto

auto: 1 1 auto

1: 1 1 0

**（6）align-self**

允许单个项目有与其他项目不一样的对齐方式，属性同align-item

### 4.影响

flex布局会令子元素的float失效、clear和vertical-align属性失效。

## Grid布局

flex是基于轴线的，一行排完后下一行。而Grid布局一开始就将容器划分为行、列，产生单元格。是一种二维布局。

### 1、父元素属性

**（1）grid-template-c olumns、grid-template-rows**

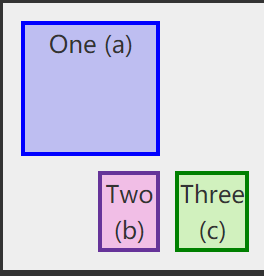
定义行列数：grid-template-columns:100px 100px 100px;

**（2）grid-xxx-gap**

定义行列间距离

**（3）grid-template-areas**

定义子元素所占区域



**（4）grid-auto-flow**

定义排列方式，按行，按列

**（5）justify-items(水平方向)/ align-items(垂直方向)**

定义单元格对齐方式

**（6）justify-content (水平方向)/ align-content(垂直方向)**

定义整体对齐方式

### 2、子元素属性

**（1）grid-(row)-start、grid-()-end**

定义该元素从哪个格开始，直到结束

**（2）grid-area**

根据grid-template-areas中的设置，直接指定元素在哪个区域

**（3）justify-self / align-self / place-self**

定义当前元素内容的位置

## 未知宽高元素水平垂直居中（方案及比较）

### 1.flex布局

**父元素**：

display: flex; //设置布局为flex

align-items: center; //容器在主轴上的对齐方式

justify-content: center; //容器在交叉轴上的对齐方式

### 2.table-cell

**父元素：**

**子元素：**

display: table-cell;

vertical-align: middle; //设置元素垂直对齐方式

text-align: center;  //设置元素水平对齐方式

（只对文本有效）

### 3.position+transform

**父元素**：

position: relative

**子元素：**

position：absolute

top：50% //相对于父元素

left:：50% //最终左上起点为中心

transform：translate(-50%, -50%) //%相对于自身长度，px相对于左上中心点

### 4.position+margin

父元素：

position: relative

子元素：

position: absolute

top: 0

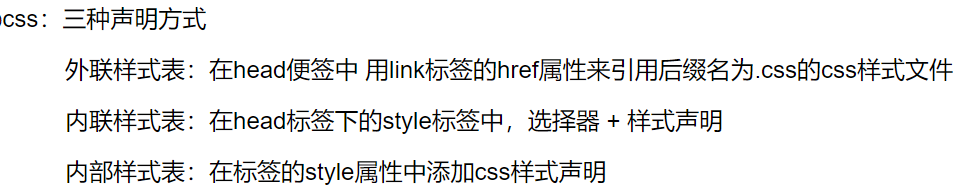
left: 0

right: 0

bottom: 0

margin: auto

## CSS声明方式



## px、em、rem的区别

**1.PX**

px像素（Pixel）。相对长度单位。像素px是相对于显示器屏幕分辨率而言的。

**PX特点**

* 1. IE无法调整那些使用px作为单位的字体大小；
* 2. 国外的大部分网站能够调整的原因在于其使用了em或rem作为字体单位；
* 3. Firefox能够调整px和em，rem，但是96%以上的中国网民使用IE浏览器(或内核)。

**2.EM**

em是相对长度单位。相对于当前对象内文本的字体尺寸。如当前对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。

**EM特点**

* 1. em的值并不是固定的；
* 2. em会继承父级元素的字体大小。

*任意浏览器的默认字体高都是16px。所有未经调整的浏览器都符合: 1em=16px。*

**3.REM**

rem是CSS3新增的一个相对单位（root em，根em），这个单位引起了广泛关注。使用rem为元素设定字体大小时，仍然是相对大小，但相对的只是HTML根元素。

通过它既可以做到只修改根元素就成比例地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。

目前，除了IE8及更早版本外，所有浏览器均已支持rem。对于不支持它的浏览器，应对方法也很简单，就是多写一个绝对单位的声明。这些浏览器会忽略用rem设定的字体大小。下面就是一个例子：

**4.px 与 rem 的选择？**

对于只需要适配少部分手机设备，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，例如只需要适配iPhone和iPad等分辨率差别比较挺大的设备。

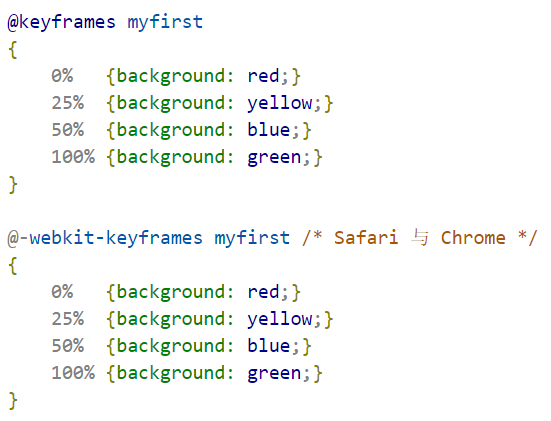
## vh与vw

vh and vw：相对于视口的高度和宽度，而不是父元素的（CSS百分比是相对于包含它的最近的父元素的高度和宽度）。1vh 等于1/100的视口高度，1vw 等于1/100的视口宽度。

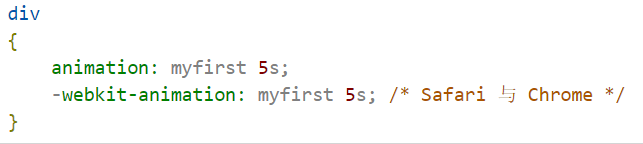
## CSS3及相关动画

**定义：**动画是使元素从一种样式逐渐变化为另一种样式的效果。

**动画效果：**

****

**绑定效果：**

****

## 如何实现响应式布局

**1.原因**

响应式布局可以让网站同时适配不同分辨率和不同的手机端，让客户有更好的体验。

**2.方法**

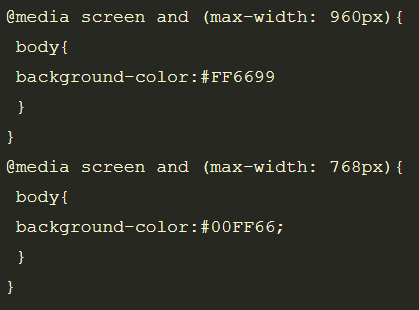
**方案一：百分比布局**

利用对属性设置百分比来适配不同屏幕，注意这里的百分比是相对于父元素； 能够设置的属性有width，、height、padding、margin，其他属性比如border、 font-size不能用百分比来设置的  
由于没办法对font-size进行百分比设置，所以用的最多就是对图片和大块布局进行百分比设置。

各个属性中如果使用百分比，相对父元素的属性并不是唯一的，造成我们使用百分比单位容易使布局问题变得复杂。

**方案二：使用媒体查询 (CSS3@media 查询)**

利用媒体查询设置不同分辨率下的css样式，来适配不同屏幕。  
媒体查询相对于百分比布局，可以对布局进行更细致的调整，但需要在每个分辨率下面都写一套css样式。  
该布局的话适用于简单的网页，可以使移动端和pc端使用一套网址。从而节约成本。也方便后期的维护，bootcss就是用了CSS3的media来实现响应的 但是相对于复杂的网页就不适合了（如：淘宝，京东）等等



**方案三.rem 响应式布局**

rem：相对于根元素(即html元素)font-size计算值的倍数。  
如 html的font-size为100px；那么2rem就为200px。  
通俗的来讲的话rem就是通过JavaScript来获取页面的宽度，从而动态的计算rem。这样就会使不同分辨率下展现出相同的效果。

**方案四.vw 响应式布局**

vm，vh相应式布局通俗来讲就是有一点类似百分比布局 将页面的宽和高分别分为100份 1vm就是为宽度的百分之一，同理1vh及为高度的百分之一。段手机端用的最多就是这个方法，能保持不同屏幕下元素显示效果一致，也不用写多套样式。

**方案五.flex 弹性布局**

flex就是利用css3中的弹性盒子向下兼容到IE10  
利用flex属性来适配不同屏幕， 该方法一般都会结合rem布局来写移动端页面

## 可继承标签

当元素的一个**继承属性 （inherited property）**没有指定值时，则取父元素的同属性的[计算值 computed](https://developer.mozilla.org/en-US/docs/Web/CSS/computed_value)

### 1.不可继承：

1、display/overflow

2、文本属性：vertical-align：

        text-decoration：

        text-shadow：

        white-space：

        unicode-bidi：

3、盒子模型的属性: 宽度：width/min-width/max-width

高度：height/min-height/max-height

内外边距：padding/margin

边框等：border

方位：left/right/top/bottom

堆叠顺序：z-index

4、背景属性: 背景图片：background-image

颜色：background-color

位置：backgournd-position

5、定位属性：浮动：float

清除浮动：clear

定位：position

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

### 2.可继承

1.所有元素可继承：元素可见：visibility/opacity

光标属性：cursor

2.内联元素可继承：字体系列属性： font：组合字体

        font-family：规定元素的字体系列

        font-weight：设置字体的粗细

        font-size：设置字体的尺寸

        font-style：定义字体的风格

        font-variant：把段落设置为小型大写字母字体：

font-stretch：允许你使文字变宽或变窄。所有主流浏览器都不支持。

        font-size-adjust：更好的控制字体大小

除text-indent、text-align之外的文本系列属性：

line-height：行高

        word-spacing：增加或减少单词间的空白（即字间隔）

        letter-spacing：增加或减少字符间的空白（字符间距）

        text-transform：控制文本大小写

        direction：规定文本的书写方向

        color：文本颜色

3.块级元素可继承：text-indent/text-align

4.其他：表格布局属性：caption-side、border-collapse、border-spacing、empty-cells、table-layout

        列表属性：list-style-type、list-style-image、list-style-position、list-style

        生成内容属性：quotes

        光标属性：cursor

        页面样式属性：page、page-break-inside、windows、orphans

        声音样式属性：speak、speak-punctuation、speak-numeral、speak-header、speech-rate、volume、voice-family、pitch、pitch-range、stress、richness、、azimuth、elevation

## 重绘重排

浏览器运行机制

**1、构建DOM树（parse）：**渲染引擎解析HTML文档，首先将标签转换成DOM树中的DOM node（包括js生成的标签）生成内容树（Content Tree/DOM Tree）；

**2、构建渲染树（construct）：**解析对应的CSS样式文件信息（包括js生成的样式和外部css文件），而这些文件信息以及HTML中可见的指令（如<b></b>），构建渲染树（Rendering Tree/Frame Tree）；

**3、布局渲染树（reflow/layout）：**从根节点递归调用，计算每一个元素的大小、位置等，给出每个节点所应该在屏幕上出现的精确坐标；

**4、绘制渲染树（paint/repaint）：**遍历渲染树，使用UI后端层来绘制每个节点。

**重排**（重构/回流/reflow）：当渲染树中的一部分(或全部)因为元素的规模尺寸，布局，隐藏等改变而需要重新构建, 这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。

**触发：**

　 1、页面渲染初始化；(无法避免)

　　2、添加或删除可见的DOM元素；

　　3、元素位置的改变，或者使用动画；

　　4、元素尺寸的改变——大小，外边距，边框；

　　5、浏览器窗口尺寸的变化（resize事件发生时）；

　　6、填充内容的改变，比如文本的改变或图片大小改变而引起的计算值宽度和高度的改变；

　　7、读取某些元素属性：（offsetLeft/Top/Height/Width,　clientTop/Left/Width/Height,　scrollTop/Left/Width/Height,　width/height,　getComputedStyle(),　currentStyle(IE)　) 会调用 getComputedStyle()或者IE的currentStyle，浏览器为了返回最新值，会触发回流。

**重绘**（repaint或redraw）：当盒子的位置、大小以及其他属性，例如颜色、字体大小等都确定下来之后，浏览器便把这些原色都按照各自的特性绘制一遍，将内容呈现在页面上。重绘是指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。

触发重绘的条件：改变元素可见外观属性。如：color，background-color等。

**重绘和重排的关系：**在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新构造这部分渲染树，完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程称为重绘。

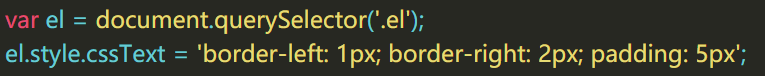
重排必定会引发重绘，但重绘不一定会引发重排。

**性能优化：**

当前浏览器用类似于MVVM的虚拟DOM，批量执行重排操作，但是实时操作无法优化：offsetTop/scrollTop/clientTop等

**方案：**

**1.使用cssText属性，一次性传入**

****

**2.提前定义样式，直接切换类名**

**3.使用文档片段createDocumentFragment进行多节点重排**

**核心思想：**

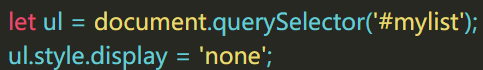
（1）让该元素脱离文档流

（2）对其进行多重改变

（3）将元素带回文档中

**方法1：隐藏元素，进行修改后，然后再显示该元素**

显示和隐藏时，产生两次重排

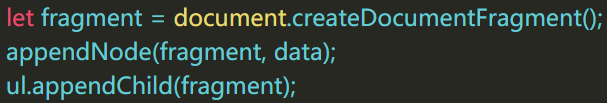
****

****

****

**方法2：使用文档片段创建一个子树，然后再拷贝到文档中**

添加片段时产生一次重排

****

**方法3：将原始元素拷贝到一个独立的节点中，操作这个节点，然后覆盖原始元素**

产生一次重排

