Chapter 3 CSS + SASS

1. px：

在CSS中，像素（px）单位通常表示一个固定的点，而不受设备或屏幕尺寸的影响。这意味着当你将元素的尺寸或字体大小设置为特定的像素值时，它们将保持相对恒定，无论用户使用的是大屏幕显示器、手机、平板电脑还是其他设备。

1. %：

在CSS中，百分比（%）单位表示相对于父元素的特定属性的百分比。这可以用于调整元素的尺寸、间距、边距等，使其相对于父元素的特定属性按比例进行调整。

1. 调整元素尺寸：

/\* 父元素宽度为400px，子元素宽度为父元素宽度的50%（200px） \*/

.parent {

width: 400px;

}

.child {

width: 50%;

}

1. 设置行高：

/\* 设置文本行高为字体大小的150% \*/

.text {

font-size: 16px;

line-height: 150%;

}

1. 垂直居中元素：

/\* 垂直居中子元素 \*/

.parent {

display: flex;

justify-content: center; /\* 水平居中 \*/

align-items: center; /\* 垂直居中 \*/

height: 200px;

}

.child {

width: 50%;

}

1. 设置边距：

/\* 父元素宽度为300px，子元素上下边距为父元素高度的10%（30px） \*/

.parent {

width: 300px;

height: 200px;

}

.child {

margin-top: 10%;

margin-bottom: 10%;

}

1. vh，vw

vh 和 vw：视窗单位

vh：视窗高度的百分比单位，1vh 等于视窗高度的1%。

vw：视窗宽度的百分比单位，1vw 等于视窗宽度的1%。

这些单位可以用来创建响应式设计，根据视窗大小调整元素的尺寸。

<style>

/\* vh vw 的用法：视窗最为参照值 \*/

.div-vh-vw{

width: 10vw;

height: 10vh;

background-color: red;

}

/\* %百分比的用法：父元素的宽高作为参照值 \*/

.div-vh-vw-child {

width: 50%;

height: 50%;

background-color: aliceblue;

}

</style>

1. em：

相对于父元素字体尺寸的单位。如果父元素字体大小为16px，1em 等于16px。

<style>

/\* 子元素的字体大小的em是相对于父元素的字体大小.

元素的width/height/padding/margin用em的话是相对于该元素的font-size \*/

.parent{

/\* 400px 400px \*/

font-size: 40px;

width: 10em;

height: 10em;

border: solid 1px blue;

}

.child {

/\* 20px 200px 200px \*/

font-size: 0.5em;

width: 10em;

height: 10em;

border: solid 1px red;

}

.grandchild {

/\* 10 100 100 \*/

font-size: 0.5em;

width: 10em;

height: 10em;

border: solid 1px black;

}

</style>

1. rem：

相对于根元素（通常是 <html>）字体尺寸的单位。可以用来解决嵌套元素字体大小累积的问题。

<style>

/\* 我们可以在并且只能在html标签（因为html节点是根节点）里面修改font-size : 10px， 从而1rem = 10px \*/

/\* px 一般用于边框或者定位 \*/

html {

font-size: 10px;

}

.parent{

/\* 40px 300px 300px \*/

font-size: 4rem;

width: 30rem;

height: 30rem;

border: solid 1px black;

}

.grandchild {

font-size: 1.5rem;

width: 10rem;

height: 10rem;

border: solid 1px blue;

}

</style>

1. bem：

* BEM 是一种命名约定，用于创建可维护且可扩展的CSS代码。
* 块（Block）：独立的组件或模块，如 .header。
* 元素（Element）：块内的子元素，如 .header\_\_logo。
* 修饰符（Modifier）：对块或元素进行修饰，如 .header--dark。

<body>

<!-- - 中划线 ：仅作为连字符使用，表示某个块或者某个子元素的多单词之间的连接记号。 -->

<!-- \_\_ 双下划线：双下划线用来连接块和块的子元素 -->

<!-- \_ 单下划线：单下划线用来描述一个块或者块的子元素的一种状态 -->

<!-- 在vue，react中，在较小组件中可以不用。对于公共的，全局的模块，还是推荐使用bem -->

<div class="artical">

<div class="artical\_body">

<button class="artical\_button--primary"></button>

<button class="artical\_button--seccess"></button>

</div>

</div>

<div>

<form action="" class="search-form">

<div class="search-form\_\_content"></div>

<button class="search-form\_\_btn--save"></button>

<button class="search-form\_\_btn--edit"></button>

</form>

</div>

</body>

1. media 媒体查询（响应式布局）：

！！！一定在head中定义：

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

Query syntax(媒体查询):

媒体查询使用`@media`作为关键字，后面紧跟一个条件用于指定样式的条件。例如：

`@media (max-width: 768px) {

/\* 在屏幕宽度小于等于768px时应用的样式 \*/

}`

Condition expression(条件表达式)：

媒体查询的条件表达式使用媒体特性和值来指定应用样式的条件。例如：

`width` `height` `oientation` `resolution`

`@media (min-width: 768px) and (orientation: landscape) {`

`/\* 在屏幕宽度大于等于768px且为横向方向时应用的样式 \*/`

`}`

示例：

@media screen and (max-width: 1200px){

.container{

width: 100%;

}

.container ul li {

width: 33.33%;

background-color: purple;

}

}

@media screen and (max-width: 600px){

.container{

width: 100%;

}

.container ul li {

width: 33.33%;

background-color: yellow;

}

}

1. Position：

Default: static

1. relative: -> `元素仍然占有文档流空间`

相对定位是相对于元素自身原始位置进行定位的。使用相对定位时，元素仍然占据原始文档流中的位置，但可以通过调整top、right、bottom和left属性来移动元素。相对定位不会影响其他元素的布局。

.box1 {

position: relative;

top: 100px;

left: 100px;

width: 200px;

height: 200px;

background-color: pink;

}

.box2 {

width: 200px;

height: 200px;

background-color: deeppink;

}

1. absolute: -> `元素脱离了文档流`

绝对定位是相对于最近的已定位祖先元素或父元素进行定位的。如果没有已定位的祖先元素，元素会相对于文档的初始包含块进行定位。绝对定位的元素脱离了文档流，不再占据空间，因此其他元素可能会占据它原来的位置。

示例1：

.father{

width: 500px;

height: 500px;

background-color: skyblue;

position: relative;

top: 50px;

left: 50px;

}

.son{

width: 200px;

height: 200px;

background-color: brown;

position: absolute;

top: 100px;

left: 100px;

}

示例2：

/\* 使用绝对定位的元素以它最近的一个“已经定位为非static”的“祖先元素”为基准进行偏移。

如果没有已经定位的祖先元素，那么会以浏览器窗口为基准进行定位。 \*/

.grandfather {

position: relative;

width: 800px;

height: 800px;

background-color: hotpink;

padding: 50px;

}

.father {

position: relative;

width: 500px;

height: 500px;

background-color: skyblue;

}

.son {

position: absolute;

width: 300px;

height: 300px;

background-color: yellow;

left: 30px;

bottom: 10px;

}

1. fixed: -> `元素脱离了文档流`

固定定位是相对于视口（浏览器窗口）进行定位的，无论页面滚动与否，固定定位的元素始终保持在相同的位置。固定定位的元素也脱离了文档流。

.box {

width: 800px;

height: 1000px;

background-color: skyblue;

margin: 0 auto;

}

.fixed {

width: 50px;

height: 150px;

background-color: purple;

position: fixed;

left: 50%;

margin-left: 405px;

}

1. sticky: -> 元素会在指定的偏移阈值内保持固定，超过阈值后会恢复正常文档流。

粘性定位可以让元素在滚动过程中表现得像相对定位和固定定位的结合体。相对父元素进行定位，`top` `left` `right` `bottom` 等偏移量也是相对于父元素的边界， 必须添加偏移量的其中一个才会生效。

/\* 参考https://www.jd.com/ \*/

body{

height: 3000px;

}

.nav {

position: sticky;

top: 20px;

width: 800px;

height: 50px;

background-color: blue;

margin: 200px auto;

color: #fff;

display: flex;

justify-content: center;

align-items: center;

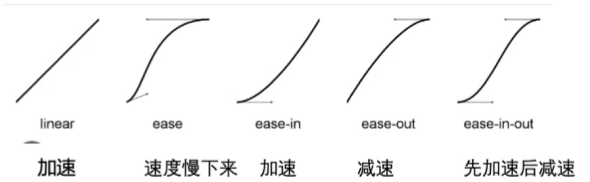
}

> Note: 在使用这些定位方式的时候，通常需要结合使用`top` `left` `right` `bottom`等属性来调整元素的位置。不同的定位方式适用于不同的布局需求。

9. 过渡（Transitions）：平滑的状态变化：

transition 属性允许您在状态改变时平滑地过渡CSS属性的值。

指定过渡的属性、持续时间和过渡函数：



示例1:

div {

width: 200px;

height: 100px;

background-color: green;

transition: all 0.5s;

}

div:hover{

width: 400px;

height: 200px;

background-color: blue;

}

示例2：

<style>

.element{

width: 100px;

height: 100px;

background-color: blue;

opacity: 0;

transition: all 0.5s;

}

.element.active{

opacity: 1;

}

.element.inactive{

opacity: 0;

}

</style>

<body>

<div class="element">

</div>

<button id="active" onclick="active()">active</button>

<button id="inactive" onclick="inactive()">inactive</button>

<script>

const active = function() {

let element=document.querySelector('.element');

element.classList.remove('inactive');

element.classList.add('active')

}

const inactive = function() {

let element=document.querySelector('.element');

element.classList.remove('active');

element.classList.add('inactive');

}

</script>

</body>

10. Box-shadow:

/\* h-offset v-offset blur spread color \*/

div {

width: 200px;

height: 200px;

background-color: pink;

margin: 100px auto;

}

div:hover {

box-shadow: 10px 10px 10px 4px rgba(255, 255, 0, 0.5);

}

11.z-index：

* z-index 控制元素在层叠上下文中的垂直堆叠顺序。
* 具有较高 z-index 值的元素将覆盖具有较低值的元素。

/\* 原先盒子没有影子,当我们鼠标经过盒子就添加阴影效果 \*/

/\* z-index仅对定位元素(不为static时)生效 \*/

.box {

position: absolute;

top: 0;

left: 0;

width: 200px;

height: 200px;

}

.box1 {

background-color: red;

z-index: 100;

}

.box2 {

background-color: green;

top: 50px;

left: 50px;

z-index: 2;

}

.box3 {

background-color: blue;

left: 100px;

top: 100px;

}

12. SASS：

SASS 是一种CSS预处理器，它通过引入更多的功能和灵活性来增强传统的CSS语法。SASS允许您使用变量、嵌套、混合（Mixins）、继承等功能，以更高效、可维护的方式编写样式。

SASS基础语法

1. 变量： 使用$来定义变量，可以存储颜色、尺寸等值。

$primary-color: #ff9900;

$font-size: 16px;

.header {

color: $primary-color;

font-size: $font-size;

}

2. 嵌套： 可以在SASS中使用嵌套结构，更好地组织和表示样式。

.nav {

ul {

list-style: none;

padding: 0;

li {

margin-bottom: 10px;

}

}

}

3. 混合（Mixins）： 混合是一种重用样式的方式。

@mixin border-radius($radius) {

border-radius: $radius;

}

.button {

@include border-radius(5px);

}

4. 继承： 可以通过@extend关键字继承一个选择器的样式。

.error-message {

color: red;

font-weight: bold;

}

.warning-message {

@extend .error-message;

color: yellow;

}

5. 导入： 使用@import来导入其他SASS文件。

// \_variables.scss

$primary-color: #ff9900;

// main.scss

@import 'variables';

.header {

color: $primary-color;

}