前言

各位同学都知道，CSS只是一门声明式的语言，主要为标记语言HTML服务。很多前端开发者都会鄙视它，不愿意深入学习，更多会抛出一个原因：现在不是有很多UI框架吗，我还编写CSS干嘛！

虽然CSS看上去弱不禁风，常用的也就是一堆静态属性声明而已。然而，这只是完全不了解CSS且还停留在编写属性声明的同学对CSS的理解而已。时至今日，随着前端技术的不断变革，也让曾经被鄙视的CSS变得越来越强大。过去只有声明式的CSS，现在也拥有了具有运算能力的函数。

CSS能做的事情可多了，JS有 变量 和 函数 ，CSS也有。本章先摸透一些常用的函数，因为函数在后面章节中各种客串出场。

函数

CSS函数指复杂类型或调用特殊处理的组件值类型。为单调的属性声明增加了更强大的点缀，让简单的CSS变得更有艺术感。其语法也很简单，编写形式为 `function(params)` ，JS里的函数调用一致。在CSS代码中，只要带有 `()` 的属性值都是函数。

有了函数后，可将一系列相关计算交给浏览器处理，可减少大量人工计算甚至无需人工计算，大大提高了CSS代码的编写效率。

分类

笔者敢相信，大部分同学常用到的函数只有 `url()` 、 `rgb()` 和 `rgba()` ，稍微深入一点的也只有 `calc()` 、 `cubic-bezier()` 和 `linear-gradient()` 。

其实不然，函数怎么会只有这几个。从W3C文档详情发现总共存在 86 个可用的函数，一点也不比属性少。按照惯例，笔者又对其进行了合理的记忆分类，以下分类均为笔者使用过的函数，暂未得到浏览器支持且在Caniuse上未收录的函数就不在分类范围内了。

> 颜色函数

- **rgb()**：RGB色彩模式
- **rgba()**：RGBA色彩模式
- **hsl()**：HSL色彩模式
- **hsla()**：HSLA色彩模式
- **color()**：色彩模式，基于当前颜色衍生出其他颜色

> 属性函数

- **attr()**：属性
- **var()**：变量

> 数学函数

- **clamp()**：区间范围值
- **counter()**：计数器
- **counters()**：嵌套计数器
- **calc()**：计算
- **max()**：最大值
- **min()**：最小值

> 背景函数

- **url()**：图像路径
- **element()**：图像映射，渲染指定元素为图像
- **image-set()**：图像集合，根据屏幕分辨率匹配合适图像
- **linear-gradient()**：线性渐变
- **radial-gradient()**：径向渐变
- **conic-gradient()**：锥形渐变
- **repeating-linear-gradient()**：重复线性渐变
- **repeating-radial-gradient()**：重复径向渐变
- **repeating-conic-gradient()**：重复锥形渐变

> 滤镜函数

- **blur()**：模糊
- **brightness()**：亮度
- **contrast()**：对比度
- **drop-shadow()**：阴影
- **grayscale()**：灰度
- **hue-rotate()**：色相旋转
- **invert()**：反相
- **opacity()**：透明度
- **saturate()**：饱和度
- **sepia()**：褐色

> 图像函数

- **circle()**：圆形
- **ellipse()**：椭圆形
- **inset()**：矩形
- **path()**：路径
- **polygon()**：多边行

> 变换函数

- **matrix()**：矩阵
- **matrix3d()**：3D矩阵
- **perspective()**：视距
- **rotate()**：旋转
- **rotate3d()**：3D旋转
- **rotateX()**：X轴旋转
- **rotateY()**：Y轴旋转
- **rotateZ()**：Z轴旋转
- **scale()**：缩放
- **scale3d()**：3D缩放
- **scaleX()**：X轴缩放
- **scaleY()**：Y轴缩放
- **scaleZ()**：Z轴缩放
- **skew()**：扭曲
- **skewX()**：X轴扭曲
- **skewY()**：Y轴扭曲
- **translate()**：位移
- **translate3d()**：3D位移
- **translateX()**：X轴位移
- **translateY()**：Y轴位移
- **translateZ()**：Z轴位移

> 缓动函数

- **cubic-bezier()**：贝塞尔曲线
- **steps()**：逐帧

颜色函数

`颜色函数` 是最常用的函数，没有之一。 `颜色函数` 可用在 `border-color` 、 `outline-color` 、 `background-color` 、 `box-shadow` 、 `color` 、 `caret-color` 等属性上使用。

> RGB色彩模式：rgb()、rgba()

例如将文本声明成白色，普通的声明可用 `color:white` 和 `color:#fff` 。有了 `颜色函数` 后，可用 `rgb()` 和 `rgba()` 声明。将原来的声明改成成 `color:rgb(255,255,255)` 或 `rgba(255,255,255,1)` 。

`rgb()` 里的R表示**红色**，G表示**绿色**，B表示**蓝色**，而 `rgba()` 多出来的A表示透明度，这个A与 `opacity` 声明的透明度不同， `rgba()` 声明的透明度不会应用到子节点上，而 `opacity` 声明的透明度会应用到子节点上。

建议在声明普通颜色时使用 `HEX色彩模式` (16进制色彩模式)，若颜色存在透明度的需求，可用 `rgba()` 。但是 `rgba()` 的参数不太友好，得把 `HEX` 转换成 `RGB` 。由于本小册使用 `sass` 作为样式预处理语言，编写 `rgb()` 和 `rgba()` 时使用 `HEX` 代替 `RGB` 即可。将原来的声明改成成 `color:rgb(#fff)` 或 `rgba(#fff,1)` 。

> HSL色彩模式：hsl()、hsla()

HSL色彩模式是一种工业界的色彩标准，因为它能涵盖到人类视觉所能感知的所有颜色，所以在工业界广泛应用。

`hsl()` 和 `hsla()` 这两个颜色函数与上述两个颜色在CSS和 `sass` 上用法相似。H表示**色相**，S表示**饱和度**，L表示**亮度**，A表示**透明度**。

色相又名**色盘**，指色彩的基本属性。就是常说的颜色名称，例如红色、绿色等，此时应该想起画家那个装满不同颜料的色盘吧。色相的单位是 `deg` ，值的范围在 `0~360deg` 间，若超过 `360deg` 则相当绕N圈再计算剩余的值。 `0deg` 和 `360deg` 为红色， `120deg` 为绿色， `240deg` 为蓝色。

饱和度指色彩的纯度。越高色彩越纯，越低色彩越灰。饱和度的单位是 `%` ，值的范围在 `0~100%` 间。 `0%` 为灰色， `100%` 为全色。

亮度指色彩的发光强度。越高色彩越亮，越低色彩越暗。亮度的单位是 `%` ，值的范围在 `0~100%` 间。 `0%` 为最暗， `100%` 为最亮。若你想亮瞎别人的狗眼，把该值调整为 `100%` 即可。

饱和度和亮度的单位即使是 `0` 也得写成 `0%` ，否则整个函数都会失效。

HSL色彩模式其实是一种将 `RGB色彩模式` 中的点在圆柱坐标系中标记出来的表示法，该表示法试图做到比基于笛卡尔坐标系的几何结构RGB更直观。

属性函数

> attr()

`attr(val)` 用于返回节点属性，通常结合伪元素的 `content` 使用，是一个很优雅的函数。兼容性好不说了，还极其低调，导致很多同学以为它是一个CSS3特性。。

```
<h1 class="hello" data-name="玩转CSS的艺术之美"></h1>
```

```
h1 {
    &::before {
        content: attr(class);
    }
    &::after {
        content: attr(data-name);
    }
}
```

`::before` 通过 `attr()` 获取 `<h1 class>` 的属性值并赋值到 `content` 上， `::after` 通过 `attr()` 获取 `<h1 data-name>` 的属性值并赋值到 `content` 上，最终 `<h1>` 的 `innerText` 是 `hello玩转CSS的艺术之美` 。

`attr()` 可灵活结合 选择器 返回节点属性并赋值到伪元素的 `content` 上，通过 `attr()` 结合 `:hover` 和 `:empty` 抓取节点需显示的内容是一个很不错的技巧。

- 在 按钮1 触发悬浮状态 `:hover` 时，通过 `attr()` 获取节点的 `data-msg` 并赋值到 `::after` 的 `content` 上
- 当 按钮2 内容为空 `:empty` 时，通过 `attr()` 获取节点的 `href` 并赋值到 `::after` 的 `content` 上


```
<a class="hover-tips btn-1" href="https://www.baidu.com" data-msg="Hell
<a class="hover-tips btn-2" href="https://www.baidu.com"></a>
```

```
.hover-tips {
    position: relative;
    padding: 0 20px;
    border-radius: 10px;
    height: 40px;
    background-color: #66f;
    line-height: 40px;
    color: #fff;
    & + .hover-tips {
        margin-top: 10px;
    }
    &.btn-1 {
        &::after {
            position: absolute;
            left: 0;
            top: 0;
            border-radius: 5px;
            width: 100%;
            height: 100%;
```

```
            background-color: rgba(#000, .5);
            opacity: 0;
            text-align: center;
            font-size: 12px;
            content: attr(data-msg);
            transition: all 300ms;
        }
        &:hover::after {
            left: calc(100% + 20px);
            opacity: 1;
        }
    }
    &.btn-2:empty::after {
        content: attr(href);
    }
}
```

- [x] 在线演示：Here
- [x] 在线源码：Here

> var()

`var()` 用于引用自定义属性，是CSS变量的组成之一，在第8章**变量计算**会详细讲解 `var()` ，在此就不再讲解了。

数学函数

> counter()/counters()

`counter()` 用于返回计数器迭代值，必须结合伪元素的 `content` 使用。它以计数器名称作为参数，并作为值传递给 `content` 。 `counters()` 用于返回嵌套计数器迭代值，情况和 `counter()` 一致。

在使用 `counter()` 和 `counters()` 时，必须与 `counter-reset` 和 `counter-increment` 一起使用。

- `counter-reset` ： 重置计数器名称与初始值，编写形式为 `counter-reset:name val`
- `counter-increment` ： 对指定计数器累计其计数值，编写形式为 `counter-increment:name` ，在使用到的地方声明就会累加

对于一些迭代需求通常都会使用HTML模板，例如Vue模板、Pug模板等，所以 `counter()` 和 `counters()` 使用场景不多，笔者也很少发掘它的用处。以下就使用 `counter()` 巧妙搭配完成一个显示权重的迭代计数器。


```
<div class="iterative-counter">
    <ul>
        <li>
            <input id="angular" type="checkbox">
            <label for="angular">Angular</label>
        </li>
        <li>
            <input id="react" type="checkbox">
            <label for="react">React</label>
        </li>
        <li>
            <input id="vue" type="checkbox">
            <label for="vue">Vue</label>
        </li>
    </ul>
    <p class="count" data-unit="个">框架：</p>
    <p class="weight" data-unit="%">权重：</p>
</div>
```

```
.iterative-counter {
    ul {
        counter-reset: index 0 count 0 weight 0;
    }
    li {
        display: flex;
        position: relative;
```

```
        align-items: center;
        counter-increment: index 1;
        &::before {
            content: counter(index)"、";
        }
        & + li {
            margin-top: 10px;
        }
    }
    input {
        overflow: hidden;
        position: absolute;
        width: 0;
        height: 0;
        opacity: 0;
        &:checked + label::before {
            color: #3c9;
            content: "\2713";
        }
    }
    label {
        display: flex;
        align-items: center;
        height: 20px;
        &::before {
            margin-right: 5px;
            border: 1px solid #3c9;
            width: 20px;
            height: 20px;
            cursor: pointer;
            line-height: 20px;
            text-align: center;
            color: transparent;
            content: "";
            transition: all 300ms;
        }
    }
```

```
        p {
            margin-top: 10px;
            &.count::after {
                content: counter(count) attr(data-unit);
            }
            &.weight::after {
                content: counter(weight) attr(data-unit);
            }
        }
    }
    #angular:checked {
        counter-increment: count 1 weight 20;
    }
    #react:checked {
        counter-increment: count 1 weight 50;
    }
    #vue:checked {
        counter-increment: count 1 weight 30;
    }
```

- 在线演示：Here
- 在线源码：Here

> calc()

`calc(exp)` 用于动态计算单位， 数值 、 长度 、 角度 、 时间 和 百分比 都能作为参数。由于执行 数学表达式 后返回运算后的计算值，所以可减少大量人工计算甚至无需人工计算，是笔者认为最有用的函数，没有之一。

`calc()` 饥不择食，所有计量单位都能作为参数参加整个动态计算。

- **数值**： 整数 、 浮点数
- **长度**： `px` 、 `em` 、 `rem` 、 `vw` 、 `vh` 等(详情可回看第5章**样式计算**)
- **角度**： `deg` 、 `turn`
- **时间**： `s` 、 `ms`
- **百分比**： `%`

`calc()` 虽然好用，但是新手难免会遇到一些坑，谨记以下特点，相信就能玩转 `calc()` 了。

- 四则运算：只能使用 `+` 、 `-` 、 `*` 、 `/` 作为运算符号
- 运算顺序：遵循加减乘除运算顺序，可用 `()` 提升运算等级
- 符号连接：每个运算符号必须使用 空格 间隔起来
- 混合计算：可混合不同计量单位动态计算

第三点尤为重要，若未能遵守，浏览器直接忽略该属性。

还记得第5章**样式计算**的一行CSS代码让页面自适应吗？ `font-size:calc(100vw / 7.5)` ，其实就是根据设计图与浏览器视窗的比例动态计算 `<html>` 的 `font-size` ： `100/750 = x/100vw` 。

在SPA里有遇过因为有滚动条或没滚动条而导致页面路由在跳转过程中发生向左或向右的抖动吗？这让强迫症患者很不舒服，此时可用 `calc()` 巧妙解决该问题。

```
.elem {
    padding-right: calc(100vw - 100%);
}
```

`100vw` 是视窗宽度， `100%` 内容宽度，那么 `100vw - 100%` 就是滚动条宽度了，声明 `padding-right` 用于保留滚动条出现的位置，这样滚动条出不出现都不会让页面抖动了。

上述两个示例都是很常用的场景， `calc()` 需结合变量才好玩，后续章节都会有 `calc()` 乱入，各位同学记得注意喔。

> clamp()/max()/min()

`clamp()/max()/min()` 都和 `calc()` 类似，所有计量单位都能作为参数参加整个动态计算。这三个函数和 `calc()` 可互相嵌套使用的。

```
.elem {
    width: calc(min(1200px, 100%) / 5);
}
```

`max(...val)` 用于返回最大值， `min(...val)` 用于返回最小值，支持一个或多个值或数学表达式。虽然 `max()` 名称是最大值，但实质上是用来限制最大值的； `min()` 名称是最小值，但实质上是用来限制最小值的。

在响应式开发中，通常会声明内容宽度 `100%` 自适应且最大值不超过 `1200px` 。

```
.elem {
    width: 100%;
```

```
    max-width: 1200px;
}
```

若用 `min()` 表示，只需一行声明即可。

```
.elem {
    width: min(1200px, 100%);
}
```

`clamp(min, val, max)` 用于返回区间范围值。 `val` 在 `min~max` 间则返回 `val` ， `val` 小于 `min` 则返回 `min` ， `val` 大于 `max` 则返回 `max` ， 妥妥的响应式函数样子。

`clamp(min, val, max)` 等价于 `max(min, min(val, max))` 。 `clamp()` 可用于响应式开发中，很好地履行了响应式的义务，让组件属性在特定条件下使用特定的值。

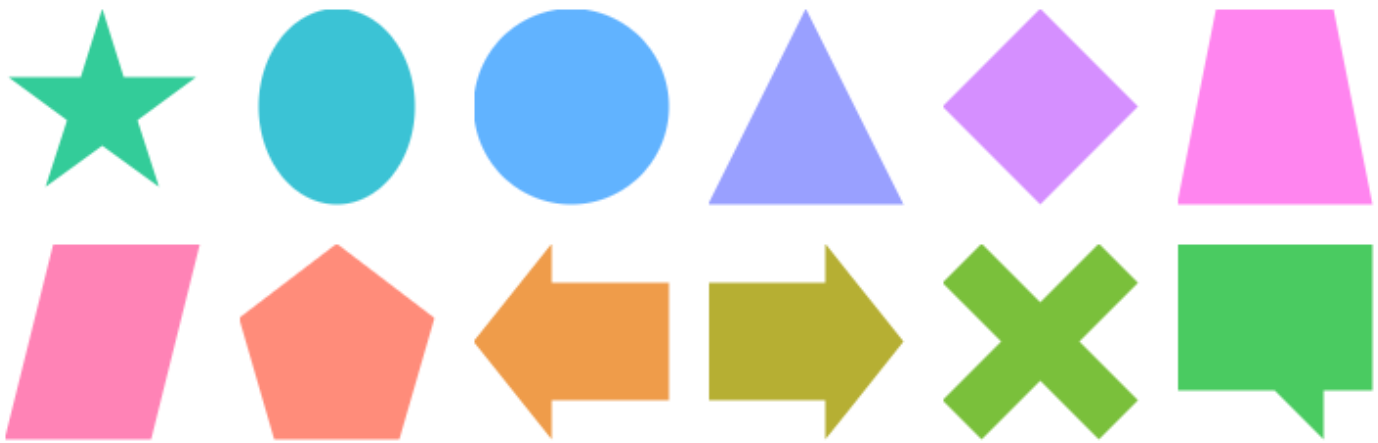
```
.elem {
    width: clamp(100px, 25vw, 300px);
}
```

节点宽度声明在 `100~300px` 间，节点随着视窗宽度变化而变化。若视窗宽度大于 `300px` 则节点宽度一直保持 `300px` ，若视窗宽度在 `100~300px` 间则节点宽度为 `25vw` 转化后的 `px值` ，若视窗宽度小于 `100px` 则节点宽度一直保持 `100px` 。

图形函数

`clip-path` 用于创建一个只有节点的部分区域可显示的剪切区域。裁剪完成后，内部区域显示，外部区域隐藏。一般应用在 `SVG` 上，但是也可当作裁剪效果用在节点上。当节点使用 `clip-path` 声明裁剪路径时，可用这5个图形函数裁剪区域了，除了 `path()` 其他4个函数的兼容性还行。

以下使用 `circle()` 、 `ellipse()` 和 `polygon()` 描绘一些常见的图像。



```html
<ul class="figure-box" style="--count: 12">
    <li class="star" style="--index: 0"></li>
    <li class="ellipse" style="--index: 1"></li>
    <li class="circle" style="--index: 2"></li>
    <li class="triangle" style="--index: 3"></li>
    <li class="rhombus" style="--index: 4"></li>
    <li class="trapezoid" style="--index: 5"></li>
    <li class="parallelogram" style="--index: 6"></li>
    <li class="pentagon" style="--index: 7"></li>
    <li class="left-arrow" style="--index: 8"></li>
    <li class="right-arrow" style="--index: 9"></li>
    <li class="close" style="--index: 10"></li>
    <li class="message" style="--index: 11"></li>
</ul>
```

```scss
.figure-box {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    max-width: 720px;
    li {
        --0: calc(var(--index) / var(--count) * 1turn);
        margin: 10px;
        width: 100px;
        height: 100px;
        background-color: #3c9;
        filter: hue-rotate(var(--0));
        &.star {
            clip-path: polygon(50% 0%, 61% 35%, 98% 35%, 68% 57%, 79% 9
        }
        &.ellipse {
            clip-path: ellipse(40% 50% at 50% 50%);
        }
        &.circle {
            clip-path: circle(50% at 50% 50%);
        }
```

```
        &.triangle {
            clip-path: polygon(50% 0%, 0% 100%, 100% 100%);
        }
        &.rhombus {
            clip-path: polygon(50% 0%, 100% 50%, 50% 100%, 0% 50%);
        }
        &.trapezoid {
            clip-path: polygon(20% 0%, 80% 0%, 100% 100%, 0% 100%);
        }
        &.parallelogram {
            clip-path: polygon(25% 0%, 100% 0%, 75% 100%, 0% 100%);
        }
        &.pentagon {
            clip-path: polygon(50% 0%, 100% 38%, 82% 100%, 18% 100%, 0%
        }
        &.left-arrow {
            clip-path: polygon(40% 0%, 40% 20%, 100% 20%, 100% 80%, 40%
        }
        &.right-arrow {
            clip-path: polygon(0% 20%, 60% 20%, 60% 0%, 100% 50%, 60% 1(
        }
        &.close {
            clip-path: polygon(20% 0%, 0% 20%, 30% 50%, 0% 80%, 20% 100
        }
        &.message {
            clip-path: polygon(0% 0%, 100% 0%, 100% 75%, 75% 75%, 75% 1(
        }
    }
}
```

整体来说很简单，在特定坐标上标记连线的点即可。推荐一个裁剪路径的网站Clippy，轻松绘制出各种由线条组成的裁剪区域。 `clip-path` 有一个明显的限制，就是只能裁剪折线形成的图形，不能裁剪曲线形成的图形。

在线演示：Here

- [x] 在线源码：Here

其他函数

由于后续章节的每一章都单独挂钩 背景函数 、 滤镜函数 、 变换函数 和 缓动函数 ，所以本节就不再讲解这四个函数了。后续章节都会对这些函数进行一些详细的讲解。