# 一、新增选择器

## 1、属性选择器

### （1）存在的意义

属性选择器可以根据元素的特定属性来选择元素，这样就不用借助与类或者id选择器了。

### （2）新增选择器描述

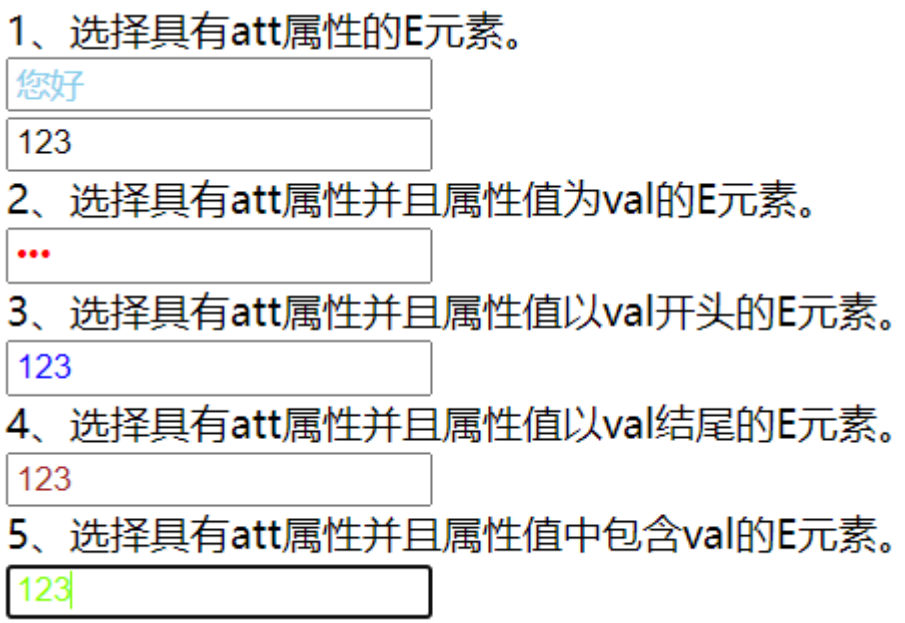
| 选择符 | 说明 |
| --- | --- |
| E[att] | 选择具有att属性的E元素。例:input[value] |
| E[att="val"] | 选择具有att属性并且属性值为val的E元素。例：input[text=password] |
| E[att^="val"] | 选择具有att属性并且属性值以val开头的E元素。例：input[text=p] |
| E[att$="val"] | 选择具有att属性并且属性值以val结尾的E元素。例：input[text=d] |
| E[att*="val"] | 选择具有att属性并且属性值中包含val的E元素。例：input[text=s] |

**案例**

```
<head>
    <style>
        /* 选择具有value属性的input元素。 */
        input[value]{
            color: skyblue;
        }
        /* 选择具有name属性并且属性值为psd的E元素。 */
        input[name=psd]{
            color:red;
        }
        /* 选择具有name属性并且属性值以em开头的input元素。 */
        input[name^=em]{
            color: blue;
        }
        /* 选择具有name属性并且属性值以ch结尾的input元素。 */
        input[name$=ch]{
            color: brown;
        }
        /* 选择具有id属性并且属性值中包含d的input元素。 */
        input[id*=d]{
            color: chartreuse;
        }
    </style>
</head>
```

```
<body>
    <form action="">
        1、选择具有att属性的E元素。<br>
        <input type="text" value="您好"><br>
        <input type="text" name="name" ><br>
        2、选择具有att属性并且属性值为val的E元素。<br>
        <input type="password" name="psd" id=""><br>
        3、选择具有att属性并且属性值以val开头的E元素。<br>
        <input type="email" name="email"><br>
        4、选择具有att属性并且属性值以val结尾的E元素。<br>
        <input type="search" name="search" id=""><br>
        5、选择具有att属性并且属性值中包含val的E元素。<br>
        <input type="text" name="" id="id">
    </form>
</body>
</html>
```

**演示**




# 2、结构伪类选择器

## （1）新增结构伪类选择器描述

| 选择符 | 描述 |
| --- | --- |
| father E: first-child | 匹配父元素中的第一个子元素E |
| father E:last-child | 匹配父元素中的最后一个子元素E |
| **father E:nth-child**（注意） | 匹配父元素中的第n个子元素E |
| E:first-of-type | 指定类型E的第一个 |
| E:last-of-type | 指定类型E的最后一个 |
| E:nth-of-type(n) | 指定类型E的第n个 |

## （2）选择符细讲

### <1>father E: first-child

- 匹配父元素中的第一个子元素、最后一个子元素

  案例

```
<head>
    <style>
        /* 匹配ul中的第一个子元素 */
        ul li:first-child{
            background-color: skyblue;
        }
        /* 匹配ul中的最后一个子元素 */
        ul li:last-child{
            background-color: rgb(142, 245, 214);
        }
    </style>
</head>
<body>

    <ul>
        <li>第一个</li>
        <li>第二个</li>
        <li>第三个</li>
        <li>第四个</li>
        <li>第五个</li>
        <li>第六个</li>
    </ul>
</body>
</html>
```

- 第一个
- 第二个
- 第三个
- 第四个
- 第五个
- 第六个

### <2>father E:nth-child

- 匹配父元素中的第n个子元素（n有三个取值：数字，关键字，公式n）
  - 数字：父元素中的第几个
  - 关键字：偶数even 奇数odd
  - 公式：nth-child(n) 这里的字母必须是n，不能是其他字母。并且n是从0开始，每次加1，但是0和超出子元素个数的部分不作显示，所以当括号里只写一个n的时候选择的是第一个子元素到最后一个子元素。

  案例

```
<head>
    <style>
        /* 将第5行改成红色 */
        ol li:nth-child(5){
            background-color: red;
        }

        /* 使用公式 */
        ol p:nth-child(n){
            background-color: rgb(240, 242, 158);
        }
        /* 使用关键字实现隔行变色 */
        ul li:nth-child(even){
            background-color: rgb(252, 218, 223);
        }
        ul li:nth-child(odd){
            background-color: rgb(200, 233, 247);
        }
    </style>
</head>
<body>
    <ol>
        <li>第一个</li>
        <li>第二个</li>
        <li>第三个</li>
        <li>第四个</li>
        <li>第五个</li>
        <li>第六个</li>
        <p>第一行文本</p>
        <p>第二行文本</p>
        <p>第三行文本</p>
        <p>第四行文本</p>
    </ol>
    <ul>
        <li>第一个</li>
        <li>第二个</li>
        <li>第三个</li>
        <li>第四个</li>
        <li>第五个</li>
        <li>第六个</li>
    </ul>

</body>
```

```
</html>
```

演示

1. 第一个
2. 第二个
3. 第三个
4. 第四个
5. 第五个
6. 第六个

第一行文本

第二行文本

第三行文本

第四行文本

- 第一个
- 第二个
- 第三个
- 第四个
- 第五个
- 第六个

**思考：如果公式是2n,2n+1,5n,n+5，-n+5，分别代表什么呢?**

解答：因为n是从0开始，一次加1，所以n是0,1,2,3,4,5,6,……那么2n就是2乘以n，也就是2乘0,2乘1,2乘2，2乘3,2乘4……得到的是偶数，以此类推

| 公式 | 取值（说明） |
| --- | --- |
| 2n | 偶数，等价于关键字even |
| 2n+1 | 奇数,等价于关键字odd |
| 5n | 5,10,15,20,25…… |
| n+5 | 从第五个开始（包含第五个）到最后 |
| -n+5 | 前5个（包含第五个） |

## <3>后三个选择符

与前三个使用类似，但也有区别，以E:nth-of-type为例：

- 区别：
  - nth-child对父元素里面的所有孩子排序选择（序号是固定的），先找到第N个孩子，然后看看是否与E匹配（例：ol p：nth-child（1），先找到nth-child（1）第一个孩子，然后匹配ol里面的第一个孩子是不是p，如果不是，则没有任何选择）
  - nth-of-type对父元素里面指定子元素进行排序，先去匹配E，然后根据E找到第N个孩子（例：ol p:nth-of-type(1),先将ol里面的所有p元素进行排序，然后选择p元素中的第一个）

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        /* 选择p中的第一个和第二个孩子 */
        /* 方式一：未选中 */
        ol p:nth-child(1){
            background-color: red;
        }
```

```
        /* 方式二：选中 */
        ol p:nth-of-type(2){
            background-color: red;
        }
    </style>
</head>
<body>
    <ol>
        <li>第1个li孩子</li>
        <li>第2个li孩子</li>
        <p>第1个P孩子</p>
        <li>第3个li孩子</li>
        <li>第4个li孩子</li>
        <p>第2个P孩子</p>
        <p>第3个P孩子</p>
        <p>第4个P孩子</p>
        <li>第5个li孩子</li>
    </ol>
</body>
</html>
```

1. 第1个li孩子
2. 第2个li孩子

   第1个P孩子

3. 第3个li孩子
4. 第4个li孩子

   第2个P孩子

   第3个P孩子

   第4个P孩子

5. 第5个li孩子

# 3、伪元素选择器

## （1）选择器描述

伪元素选择器可以帮助我们利用CSS创建新标签元素，而不需要添加额外的HTML标签，从而达到简化代码的目的

| 选择符 | 描述 |
| --- | --- |
| ::after | 在元素内部的前面插入内容 |
| ::before | 在元素内部的后面插入内容 |

**注意：**

- after 和 before创建一个元素，但是属于**行内元素**(直接设置宽高无效)
- 新创建的这个元素在文档树中是找不到的，也就是不是一个html标签，所以称为**伪元素**（比如：有一张图片，当鼠标放到图片上时，会出现灰色遮罩层，用平常写法遮罩层)是用一个盒子，大小与图片一样，当鼠标放上去时，进行覆盖。但是使用伪元素的话，可以直接用hover::before,然后设置样式即可，而不需要创建div

- after是在**父元素内容**的前面创建元素（也就是创建了一个最大的儿子，而不是创建在了父元素的前面（父元素的兄弟））。before是在父元素内容的后面创建元素（也就是创建了一个幺儿）
- 伪元素与标签选择器一样，权重为1

## （2）语法

```
element::before{
    content:‘’;
}
```

**注意：**

before和after必须要有content属性，需要添加的内容充当属性值。

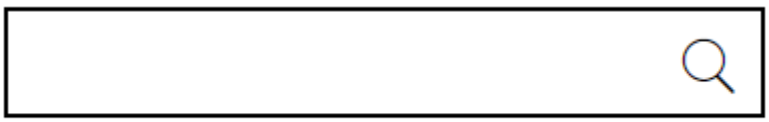
### 案例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div{
            width: 100px;
            height: 100px;
            background-color: skyblue;
        }
        div::before{
            /* 在父元素div内容的前面添加“我” */
            content: '我';
            /* 添加的是行内元素 直接添加宽高无效*/
            width: 30px;
            height: 30px;
            background-color: pink;
            /* 转换成行内块元素 */
            display: inline-block;
        }
        div::after{
            /* 在父元素div内容的后面添加“喜羊羊” */
            content: '喜羊羊';
        }
    </style>
</head>
<body>
    <div>是</div>
</body>
</html>
```

### 演示


## (3) 案例

- 通过伪元素选择器结合字体图标完成以下搜索框



代码:

```
<head>
<style>
        /* 声明字体 */
        @font-face {
            font-family: 'icomoon';
            src: url('fonts/icomoon.eot?xfgrp8');
            src: url('fonts/icomoon.eot?xfgrp8#iefix') format('embedded-opentype'),
                url('fonts/icomoon.ttf?xfgrp8') format('truetype'),
                url('fonts/icomoon.woff?xfgrp8') format('woff'),
                url('fonts/icomoon.svg?xfgrp8#icomoon') format('svg');
            font-weight: normal;
            font-style: normal;
            font-display: block;
        }

        div {
            width: 300px;
            height: 40px;
            border: 2px solid;
            /* 子绝父相 */
            position: relative;
        }

        /* 因为搜索框的图标是一直显示在上面的，所以使用after */
        div::after {
            /* 添加在内容前面，并去声明字体 */
            content: '';
            /* 调用上面的字体 */
            font-family: 'icomoon';

            /* 调整字体图标大小 */
            font-size: 32px;
```

```
            /* 使用绝对定位将图标放到右边    子绝父相（否则会跑到页面的右边*/
            position: absolute;
            top: 5px;
            right: 5px;

        }
    </style>
</head>
<body>
    <!-- 使用伪元素完成搜索框 -->
    <div></div>
</body>
</html>
```

- 补充：通过伪元素选择器实现清除浮动（额外标签法的升级和优化）

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        div{
            width: 1000px;
            margin: 10px auto;
        }
        img{
            width: 300px;
            height: 250px;
            float: left;
            margin-right: 10px;
        }
        /* 方式一：使用伪元素清除浮动 */
       /* img::after{ */
        /* content: ""; */
        /* 额外标签法，插入的必须是块级元素 */
        /* display: block; */
        /* 将插入的元素不显示 高度为0 ，隐藏*/
        /* height: 0; */
        /* visibility: hidden; */
        /* 清除浮动 */
        /* clear: both; */
       /* } */

       /* 方式二，使用双重伪元素清除浮动，使前后都进行包裹 */
       img::after, img::before{
        content: "";
        /*display: inline-block;将其转为行内块元素，直接转成块级元素，创建的第二个会到第二行
```

```
            或者写display:table    转换为表格形式 */
            display: inline-block;
        }
        img::after{
            clear: both;
        }
    </style>
</head>
<body>
    <div>
        <img src="../图片/06.png" alt="">
        <img src="../图片/06.png" alt="">
        <img src="../图片/06.png" alt="">

        <!-- 额外标签法清除浮动  -->
        <!-- <div style="clear: both;"></div> -->
    </div>
    <p>1111111111111111111111111111111111111111111111111111</p>
    <p>1111111111111111111111111111111111111111111111111111</p>
</body>
</html>
```


1111111111111111111111111111111111111111111111111111

1111111111111111111111111111111111111111111111111111

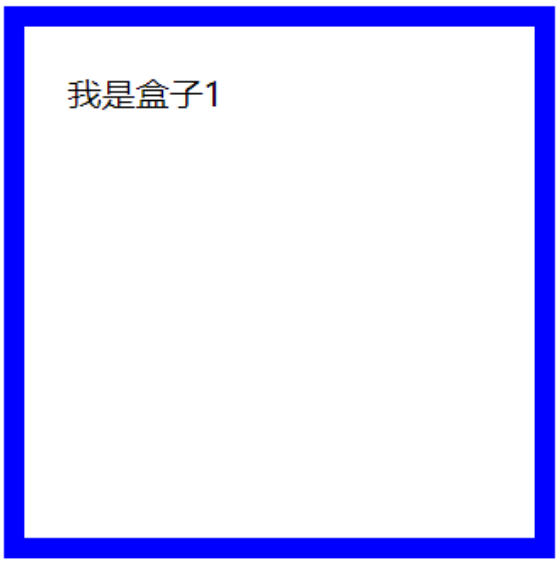
# 二、新增盒子模型属性

## （1）box-sizing:border-box

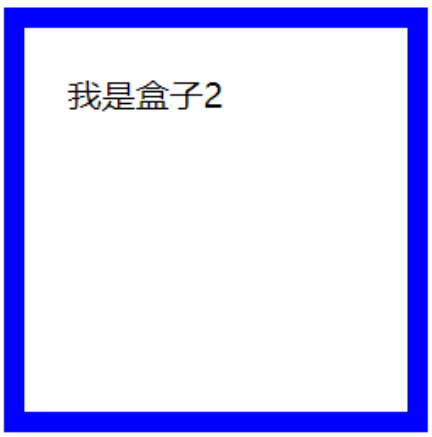
在以前，我们写完一个盒子，假设宽度为200，但是加上边框，内边距之后，盒子的总宽度会发生变化，对实际布局造成影响。如果盒子模型被改为了box-sizing:border-box，即使添加了边框，内边距，盒子的总大小也不会发生改变（前提是padding和border不会超过width宽度）

```
<!DOCTYPE html>
<html lang="en">
<head>
        <style>
        .box1{
            /* 这种情况就是默认的 box-sizing: content-box;*/
            width: 200px;
            height: 200px;
            border: 10px solid blue;
```

```
            padding: 20px;
        }
        .box2{
            width: 200px;
            height: 200px;
            border: 10px solid blue;
            padding: 20px;
            box-sizing: border-box;
        }
    </style>
</head>
<body>
    <div class="box1">我是盒子1</div>
    <br>
    <div class="box2">我是盒子2</div>
</body>
</html>
```




**没有添加box-sizing:border-box属性的盒子，会被边框，内边距撑大**




# 三、滤镜filter

将模糊或颜色便宜等图形效果应用于元素

## 1、语法

```
filter:blur函数();
/* 例如：filter:blur(5px)*       blur 是一个函数，后面js部分会讲解    括号里数值越大，越模糊/
```

## 2、示范

### 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        img{
            width: 200px;
            height: 200px;
        }
        .p1{
            filter: blur(5px);
        }
    </style>
</head>
<body>
    <img src="../图片/06.png" alt="">  
    <img src="../图片/06.png" alt="" class="p1">
</body>
</html>
```

### 演示


# 四、过渡

# 1、介绍

过渡是CSS3中一个很重要的内容，可以在不适用flash动画或JS的情况下，当元素从一种样式变换成另外一种样式时为元素添加效果。（低版本浏览器不支持，比如：ie9以下，但不影响布局）

经常与：hover搭配一起使用

# 2、语法

**谁需要过渡给谁用！！！**

```
/* transition:需要过渡的属性 花费的时间 运动曲线 何时开始 */
transition: width 0.5s ease 1s;
```

**注意：**

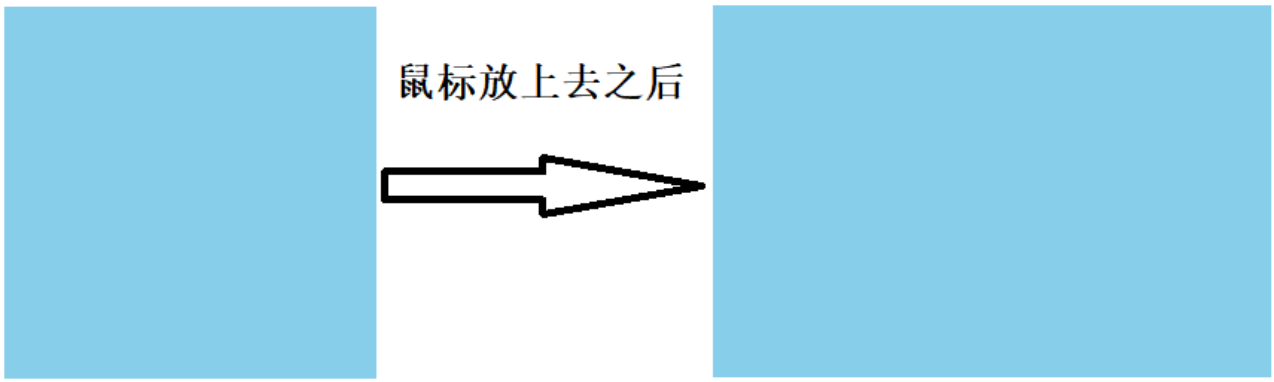
- 属性：需要变化的CSS属性，宽高，背景颜色，内外边距……，需要多个属性值变化，可以利用逗号进行分割，如果需要所有的属性都变化 则可以直接写all
- 花费时间：单位是秒（必须要写单位）
- 运动曲线（可以省略）：默认是ease(逐渐变慢)，linear（匀速），ease-in(加速)，ease-out(减速)，ease-in-out(先快后慢)
- 何时开始：单位是秒（必须要写单位），默认是0秒

# 3、示范

## 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
<style>
        div{
            width: 200px;
            height: 200px;
            background-color: skyblue;
            /* transition:需要过渡的属性 花费的时间 运动曲线 何时开始 */
             /* 谁需要过渡给谁用，在这里是div的样式发生变化，所以给div用*/
            transition: width 0.5s ease 0.1s;
             /* transition: width 0.5s,height 0.5s;*/
        }
        div:hover{
            width: 300px;
             /* height:300px*/
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

**演示**




# 4、练习

使用过渡属性，完成进度条案例


代码

```
<head>
<style>
        .father{
            width: 150px;
            height: 10px;
            border: 1px solid red;
            border-radius: 5px;
            /* box-sizing: border-box; */
            padding: 0.5px;
        }
        .son{
            width: 50%;
            height: 100%;
            background-color: red;
            /* 给一个边框，好设置圆角  但是不给线的形状，就不会显示出来 */
            border: 1px;
            border-radius: 5px;
            /* 谁发生变化，给谁设置过渡 */
            transition: all .7s;

        }
        /* 当鼠标放到父盒子上，子盒子发生变化 */
        .father:hover .son{
            width: 100%;
        }
    </style>
</head>
<body>
   <div class="father">
```

```
        <div class="son"></div>
    </div>

</body>
</html>
```

# 五、2D转换

## 1、2D转换之移动translate

**转换（transform)**是CSS3中具有颠覆性的特征之一，可以实现元素的位移、旋转、缩放等效果。转换可以简单理解为“变形”

### （1）思考

移动可以用哪些方法实现?

定位，外边距，translate

### （2）语法

```
transform:translate (x,y)    /*x表示x轴的移动距离，y表示y轴的移动距离     注意，y轴向下数值越来越大*/

/*或者可以分开写   适用于只对一个方向进行移动，但是，如果两个方向都需要移动，就不要分开写，因为下面一个transform会覆盖上面的一个*/
transform:translateX(n);
transform:translateY(n);
```

### （3）示范

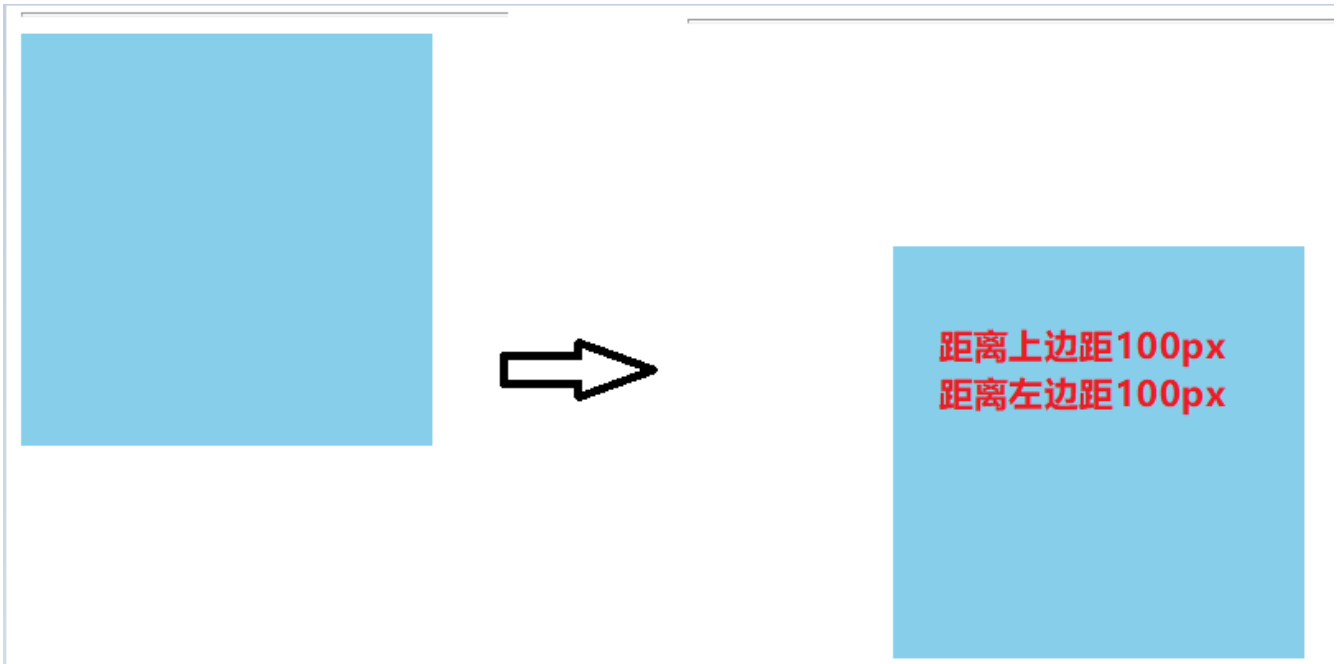
#### 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
         <title>移动</title>
    <!--移动盒子，使其距离上边距100px,距离左边距100px -->
    <style>
        div{
            width: 200px;
            height: 200px;
            background-color: skyblue;
            /* 使用移动translate进行移动，向右向下移动100px */
            transform: translate(100px,100px);
             /* 还可以单独设置 */
            /* transform: translateX(100px); */
            /* transform: translateY(100px); */
        }

    </style>
```

```
</head>
<body>
    <hr>
    <div></div>
</body>
</html>
```

### 演示




## （4）注意

- translate在使用过程中，不会影响到其他元素的位置（同时有两个盒子，上下排列，把上面的盒子向下向右移动，下面的盒子依旧放在原来的位置不会发生改变）
- translate中的百分比单位，是相对自身元素的。 例盒子宽高100px， translate：（50%，50%）则向右向下都移动宽高的一半，50px
- translate对行内标签没有效果（因为行内标签，一行可以放置多个，并且设置宽高无效）

## （5）练习

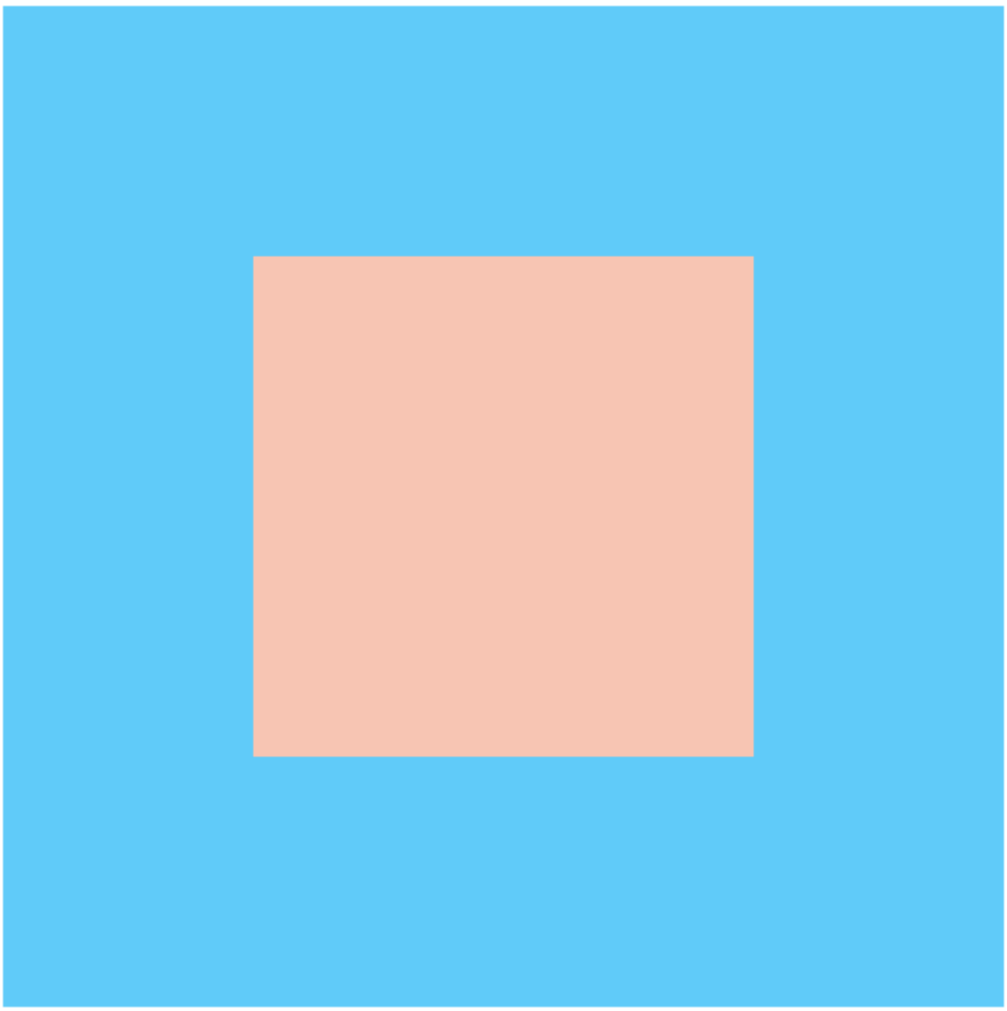
将子盒子在父盒子里面设置水平居中（定位时讲过）

### 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
    <!-- 将子盒子在父盒子内水平居中 -->
    <style>
        .f{
            position: relative;
            width: 400px;
            height: 400px;
```

```
            background-color: rgb(96, 203, 249);
        }
        .s{
            width: 200px;
            height: 200px;
            background-color: rgb(247, 197, 179);
            /* 先将盒子向右向下移动50%，然后向上向左移动盒子的一半*/
            position: absolute;
            top: 50%;
            left: 50%;
            /* 在定位中学习的方法 */
            /* margin-top: -100px; */
            /* margin-left: -100px; */
            /* 使用translate */
            transform: translate(-50%,-50%);
        }
    </style>
</head>
<body>
    <div class="f">
        <div class="s"></div>
    </div>
</body>
</html>
```

## 演示



## 2、2D转换之旋转rotate

### （1）语法

```
transform:rotate(度数)
/*度数的单位是deg      度数为正则顺时针旋转      旋转的中心就是元素的中心点*/
```

### （2）示范

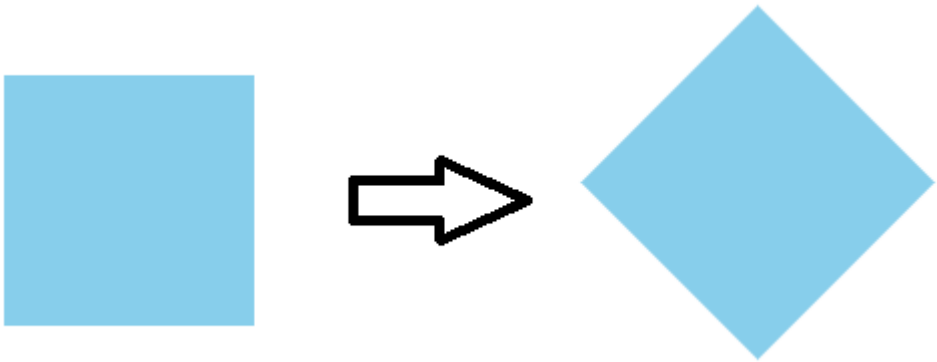
#### 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
    <style>

        div{
            width: 100px;
            height: 100px;
            background-color: skyblue;
```

```
            /* 顺时针旋转45度 */
            transform: rotate(45deg);
            /* 以元素的中心点作为中心进行旋转（斜边大于直角边）  所以需要添加外边距  才能显现完全 */
            margin: 25px;
        }
    </style>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

**演示**



## （3）练习

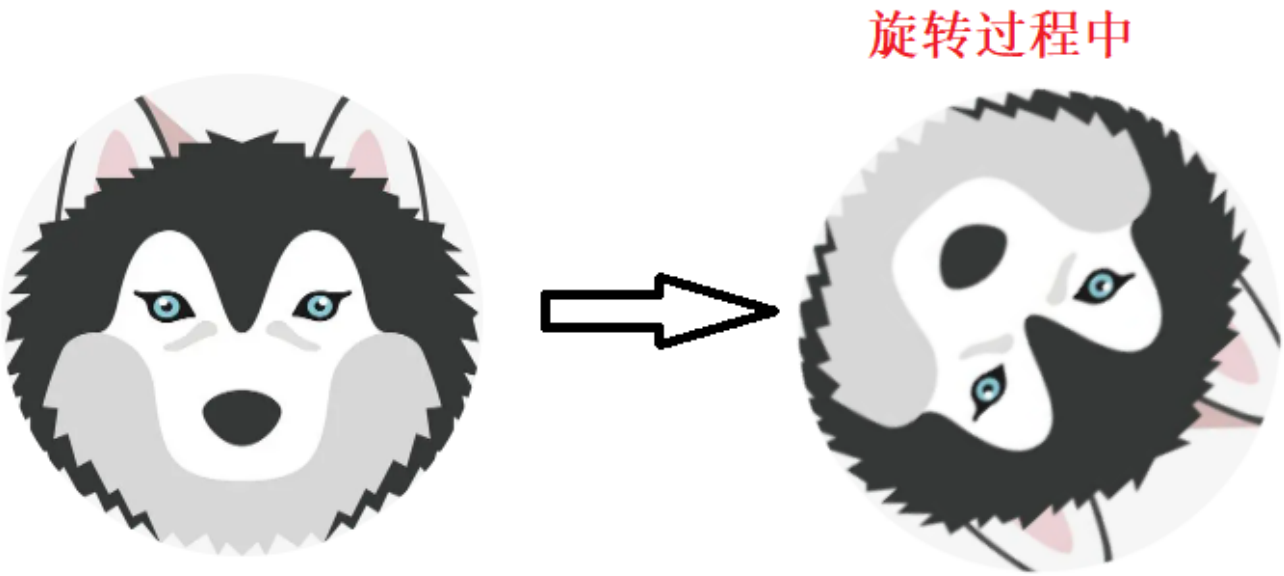
- 第一题：页面上放置一张图片，当鼠标放上图片时，图片开始360度旋转

**代码**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
    <style>
        img{
            width: 200px;
            height: 200px;
            /* 设置圆形图片 */
            border-radius: 50%;
            /* 设置过渡 */
            transition: all 0.7s;
        }
        img:hover{
            /* 旋转360度 */
            transform: rotate(360deg);
        }
    </style>
</head>
```

```
<body>
    <img src="../图片/dog.png" alt="">
</body>
</html>
```

**演示**



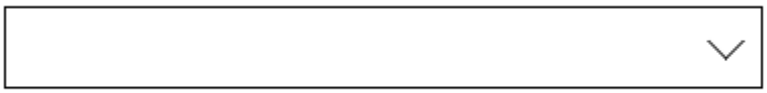

- 第二题：完成下拉列表框的显示

  **代码**

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <!-- 制作下拉框 -->
    <style>
        div{
            width: 200px;
            height: 20px;
            border: 1px solid;
            position: relative;
        }
        div::after{
            /* 添加元素  默认是行内元素 需要转换成行内块元素或者块级元素*/
            content: '';
            display: inline-block;
            /* 设置只有右边框和下边框  然后进行旋转45度  最后定位*/
            border-right: 1px solid #000;
            border-bottom: 1px solid #000;
            width: 6px;
            height: 6px;
            /* 旋转 */
            transform: rotate(45deg);
           /* 定位   子绝父相（因为该元素是后添加的   所以相当于是子元素）*/
```

```
            position: absolute;
            top: 5px;
            right: 5px;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

**演示**



## （3）思考

在默认情况下，旋转的中心点是元素的中心，但有的时候，需要更换旋转中心点，比如以元素的左上角为中心点.....该怎么调整呢?

解决方案：使用transform-origin调换中心点

### <1> transform-origin调换中心点

```
/*使用方法*/
transform-origin:x y;
```

**注意:**

- 参数x和y用空格隔开，不适用逗号（需要与translate区分开来）
- 参数x和y可以使用数值（中心点距离左上角原点的X轴和 Y轴的距离），也可以设置百分比（默认情况下中心点就是元素的中心，也就是transform-origin:50% 50%;），还可以直接使用方位名词（top,bottom,left,right,center)

### <2> 练习

设置一个盒子，当鼠标放上去时，盒子以左下角为中心旋转360度

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
    <!-- 更换旋转中心点 -->
    <!-- 鼠标放上去，盒子以左下角为中心旋转360度 -->
    <style>
        div{
            width: 200px;
            height: 200px;
            background-color: skyblue;
            margin: 80px auto;
```

```
            /* 更换中心点 */
            transform-origin: left bottom;
            /* 使用过渡，更容易看清楚旋转 */
            transition: all 0.7s;
        }
        /* 鼠标放上去开始旋转 */
        div:hover{
            transform:rotate(360deg);
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

# 3、2D转换之缩放scale

## （1）语法

```
transform:scale(x,y)
```

**注意：**

- 参数x和y是只有数字，不跟单位，是指倍数（放大（大于1）或缩小（小于1）原来宽高的倍数）

```
    <style>
        /* 将宽高为200px的盒子缩放为宽高为100px的盒子 */
        div{
            width: 200px;
            height: 200px;
            background-color: skyblue;
            margin: 100px auto;
        }
        div:hover{
            /* transform: scale(0.5,0.5); */
            /* 当宽高缩放的倍数一样时，可以只写一个参数 */
            transform: scale(0.5);
        }
    </style>
</head>
<body>
    <div>
    </div>
</body>
```

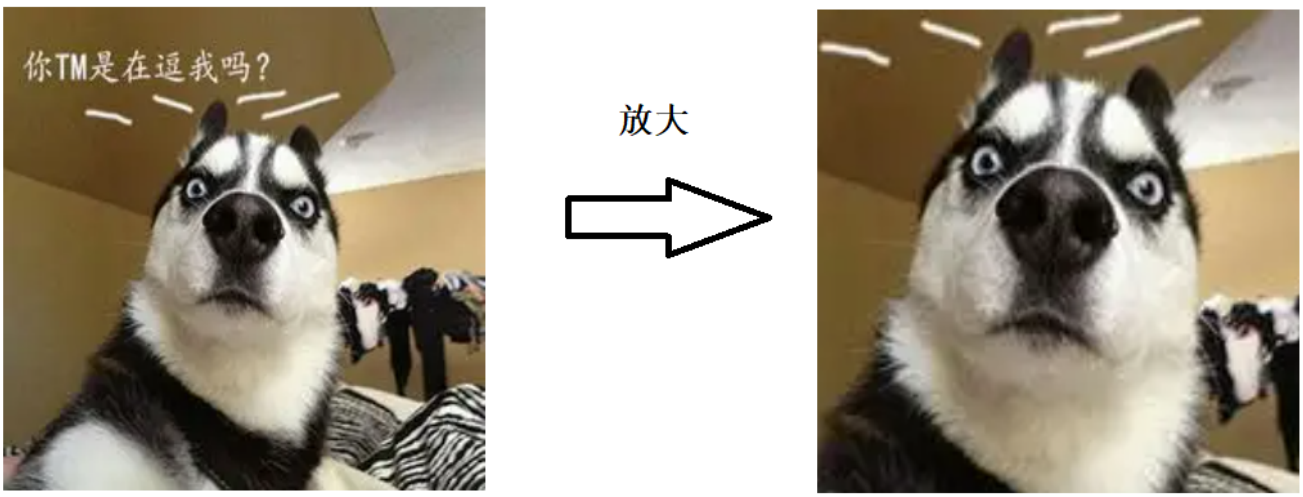
- scale最大的优点，就是不会影响其他盒子的布局
- scale缩放，默认以元素中心作为缩放中心点（一般情况下不会修改）。改变缩放中心点与旋转中心点的做法一致，详情参考“transform-origin调换中心点”，

## （2）案例

### 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div{
            width: 300px;
            height: 300px;
            float: left;
            margin-left: 30px;
            /* 图片放大后，超出部分隐藏 */
            overflow: hidden;
        }
        div img{
            /* 设置图片大小与div一致 */
            width: 300px;
            height: 300px;
            /* 给缩放过程设置过渡效果 */
            transition: all .5s;
        }
        div img:hover{
            transform: scale(1.5);
        }
    </style>
</head>
<body>
    <div><a href="#"><img src="../图片/06.png" alt=""></a></div>
    <div><a href="#"><img src="../图片/06.png" alt=""></a></div>
    <div><a href="#"><img src="../图片/06.png" alt=""></a></div>
</body>
</html>
```

### 演示



# 4、2D转换综合写法

```
transform:translate() rotate() scale()……
```

**注意：**

- 综合写法的顺序不要打乱，会影响转换的结果（先旋转会改变坐标方向）
- 当同时使用位移和其他属性的时候，一定要讲位移放在最前面

## 案例

```
<style>
        div{
            width: 200px;
            height: 200px;
            background-color: skyblue;
            /* 加上过渡 */
            transition: all .5s;

        }
        div:hover{
            /*旋转向右移动300px,向下移动150px  并缩放至原大小的一半 */
            transform: translate(300px,150px) rotate(100deg) scale(0.5);
        }
    </style>
</head>
<body>
    <div></div>
</body>
```

## 演示

# 六、动画

动画（animation)，区别于过渡，可以自动“动”起来，而过渡需要鼠标经过才能触发。

## 1、使用

步骤：

- 先定义动画
- 再调用动画

### （1）用keyframes定义动画

（类似于定义类选择器）

```
@keyframes 动画名称{
    /*0%指的是动画开始*/
        0%{
        样式
    }
    /*100%指的是动画结束      0到100的过程称为动画序列*/
    100%{
        样式
    }
}
```

**注意：**0%可以用from代替，100%可以用to代替

### （2）调用动画

**注意：**谁需要给谁加

假设是div需要加动画效果，则调用情况如下

```
div{
    width:200px;
    height:200px;
    background-color:skyblue;
    /*调用动画*/
    animation-name:动画的名称
    /*持续时间：动画从0到100需要多久完成*/
    animation-duration:持续时间;
}
```

### （3）案例

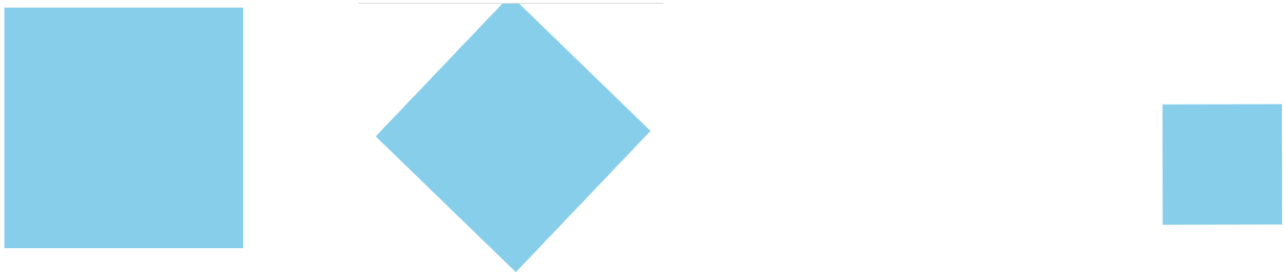
打开页面，盒子自动从左向右旋转移动 并且最后缩放至原大小的一半

#### 代码

```
<!-- 打开页面，盒子自动从左向右旋转移动  并且最后缩放至原大小的一半 -->
    <style>
        /* 第一步：先定义动画 */
        @keyframes TheOne {
            0% {
                transform: translateX(0px) rotate(0deg);
            }

            100% {
                transform: translateX(1200px) rotate(360deg) scale(0.5);
            }
        }
        div {
            width: 200px;
            height: 200px;
            background-color: skyblue;
            /* 第二步：调用动画 */
            animation-name: TheOne;
            animation-duration: 2s;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

**演示**

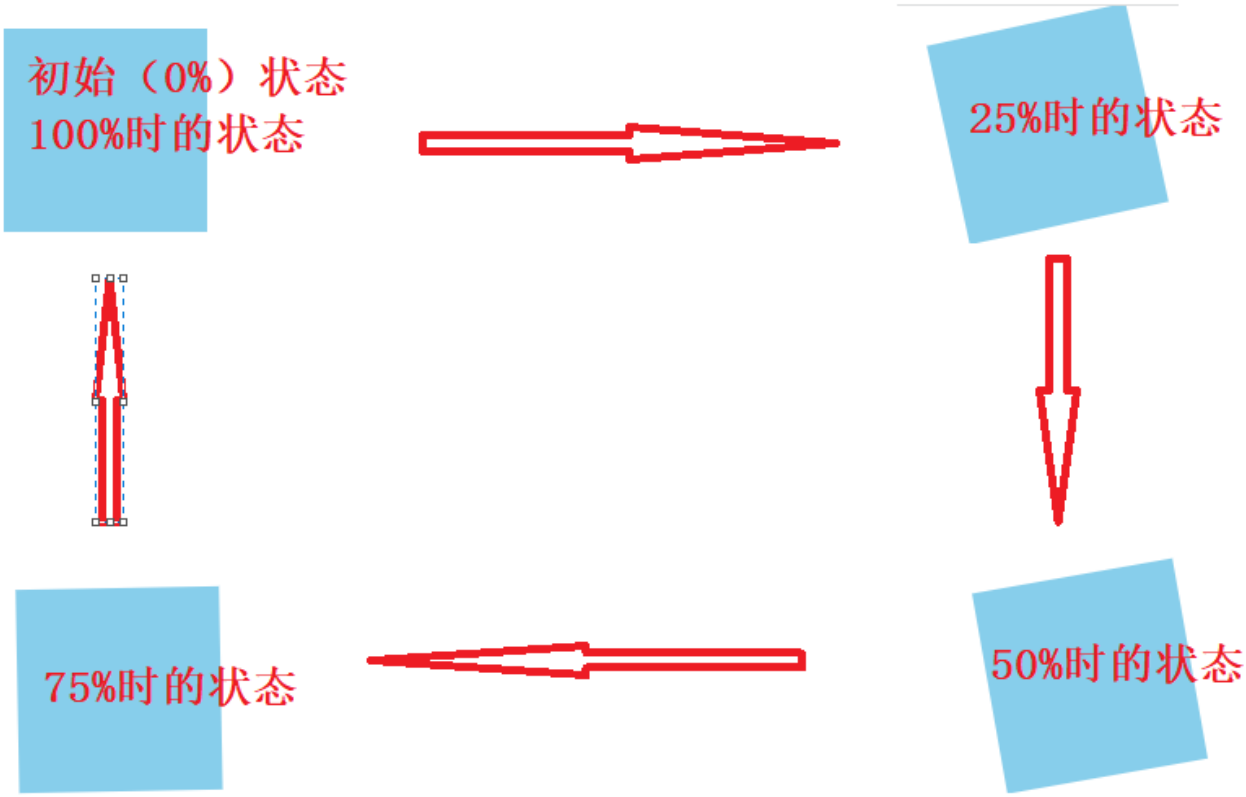


# 2、动画序列

动画是使元素从一种样式逐渐变化成另外一种样式的效果，**可以改变任意多的样式，任意多的次数**

## （1）案例

将一个盒子从左上角旋转至右上角，又从右上角旋转至右下角，接着从右下角旋转至左下角，最后回到左上角，




## 代码

```
<head>
    <style>
        /* 将一个盒子从左上角旋转至右上角，又从右上角旋转至右下角，接着从右下角旋转至左下角，最后回到左上角
*/
        @keyframes TheTwo{
            /* 0%时的状态就是初始默认状态，可以不写或者为空 */
            /* 0%{
                transform: translate(0,0) rotate(0deg);
            } */
            25%{
                transform: translate(1200px,0) rotate(360deg);
            }
            50%{
                transform: translate(1200px,500px) rotate(720deg);
            }
            75%{
                 /* 可以继续顺时针转动，就是1080px，若是要逆时针转动，就是360px */
                transform: translate(0px,500px) rotate(1080deg);
            }
            100%{
                transform: translate(0,0) rotate(1440deg);
            }
        }
        div{
            width: 100px;
            height: 100px;
            background-color: skyblue;
            /* 调用 */
```

```
            animation-name: TheTwo;
            animation-duration: 10s;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

**思考：**我们所做出来的动画，只能自动执行一次，该怎么使其循环起来呢？

# 3、动画属性

| 属性 | 说明 |
| --- | --- |
| @keyframes | 定义动画（必须写） |
| animation | 所有动画的简写属性，除了animation-play-state属性 |
| animation-name | 所调用的动画的名称（必须写） |
| animation-duration | 完成动画所需要的时间，单位是秒（必须写） |
| animation-timing-function | 规定动画的运动曲线，默认是ease(还有linear匀速，steps指定时间函数中的间隔数量（步长）) |
| animation-delay | 规定动画是从什么时候开始，默认是0 |
| animation-iteration-count | 规定动画被播放的次数，默认是1次。无限循环是infinite |
| animation-direction | 规定动画是否在下一周期逆向播放，默认是normal,逆向是alternate |
| animation-fill-mode | 规定动画结束后，是保持（forwards）结束时的状态,还是回到原始状态（backwards) |
| animation-play-state | 规定动画是否正在运行或暂停，默认是running 还有暂停pause |

## 示范

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
    <style>
        @keyframes one{
            0%{
                transform: translateX(0px) rotate(0deg);
            }
            100%{
                transform: translateX(1200px) rotate(360deg);
            }
```

```
        }
        div{
            width: 100px;
            height: 100px;
            background-color: skyblue;
            animation-name: one;
            animation-duration: 3s;
            /* 规定运动曲线为匀速 默认是ease(逐渐变慢) */
            animation-timing-function:linear;
            /* 规定动画在运行2s钟之后开始 */
            animation-delay: 2s;
            /* 规定动画循环的次数：infinite无限循环 */
            /* animation-iteration-count: infinite; */

            /* 规定动画是否在下一周期逆向（alternate）播放   */
            /* 盒子运动到右边的之后，不是嗖的一下回到左边，而是逆向运行到左边 */
            /* animation-direction: alternate; */

            /* 规定动画结束后，保持（forwards)结束是的状态，而不回到初始(backwards)状态  */
            /* 需要先注释掉循环播放和逆向播放 */
            animation-fill-mode: forwards;
        }
        /* 当鼠标放到动画上面时，停止运动 */
        div:hover{
            animation-play-state: paused;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

**补充：步长steps()使用**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        div{
            width: 200px;
            height: 26px;
            font-size: 20px;
            background-color: skyblue;
            /* steps () , 设置10步完成动画：从0px到200px */
            animation:one 4s steps(10) forwards;
            /* 强制一行显示 */
            /* white-space: inherit; */
            overflow: hidden;
```

```
        }
        @keyframes one{
            0%{
                width: 0px;
            }
            100%{
                width: 200px;
            }
        }
    </style>
</head>
<body>
    <div>今天星期五马上放假啦</div>
</body>
</html>
```


# 4、动画属性简写模式

animation：动画名称 持续时间 运动曲线 何时开始 播放次数 是否逆向播放 动画结束时状态（这个属性存在时，循环和逆向就不可以存在）

那么上述代码的简写模式就是：

```
animation:one 3s linear 2s infinite alternate;
/*非必要属性可以省略*/
```

# 5、案例

1、制作一个“信号发散模拟图”，如下所示

## 代码

```
<head>
    <style>
        .father{
            width: 200px;
            height: 200px;
            border: 1px solid;
            margin: 100px auto;
            position: relative;
        }
        /* 先定义中间不变的小圆点 */
```

```
        .circular{
            width: 2px;
            height: 2px;
            border: 1px solid rgba(88, 88, 250,0.8);
            border-radius: 50%;
            background-color: rgba(88, 88, 250,0.8);
            /* 将其放在父盒子的正中心 */
            position: absolute;
            top:50%;
            left: 50%;
            transform: translate(-50%,-50%);
        }
        .father div[class^="son"]{
            width: 3px;
            height: 3px;
            border: 2px solid rgba(88, 88, 250,0.8);
            border-radius: 50%;
            /* 将这几个全部放置父盒子的正中间，后面进行缩放的时候依旧是在正中心 */
            position: absolute;
            top:50%;
            left: 50%;
            transform: translate(-50%,-50%);
            /* 调用动画 */
            /* animation: big 1.2s linear infinite; */
            /*此时，三个子盒子是同时变大的，可以设置动画延迟使其更好看，
            但是如果后面单独给son2  son3做延迟，无法生效，因为在这个地方使用了连写（复合）模式，权重更高*/
        }
        /* 单独调用动画 */
        .son1{
            animation: big 1.2s linear infinite;
        }
        .son2{
            animation: big 1.2s 0.3s linear infinite;
        }
        .son3{
            animation: big 1.2s 0.6s linear infinite;
        }
        /* 设置动画 */
        @keyframes big{
            0%{

            }
            100%{
                width: 30px;
                height: 30px;
                border: 2px solid rgba(88, 88, 250,0);
            }
        }
    </style>
</head>
<body>
    <div class="father">
        <!-- 定义中间的小圆点 -->
```

```
        <div class="circular"></div>
        <!-- 定义缩放的几个小圆圈 -->
        <div class="son1"></div>
        <div class="son2"></div>
        <div class="son3"></div>
    </div>
</body>
</html>
```

**注意：**

- 上面的案例中，放大的小圆圈使用的是边框的形式制作的，一般情况下，使用阴影会更加好看，将border替换成“box-shadow"，设置相应的属性值即可。

  ```
  box-shadow: h-shadow v-shadow blur spread color inset;
  /*水平阴影位置（必填） 垂直阴影位置（必填） 模糊距离 阴影大小 阴影颜色 从外层的阴影改变阴影内侧阴影*/
  ```

  **示范**

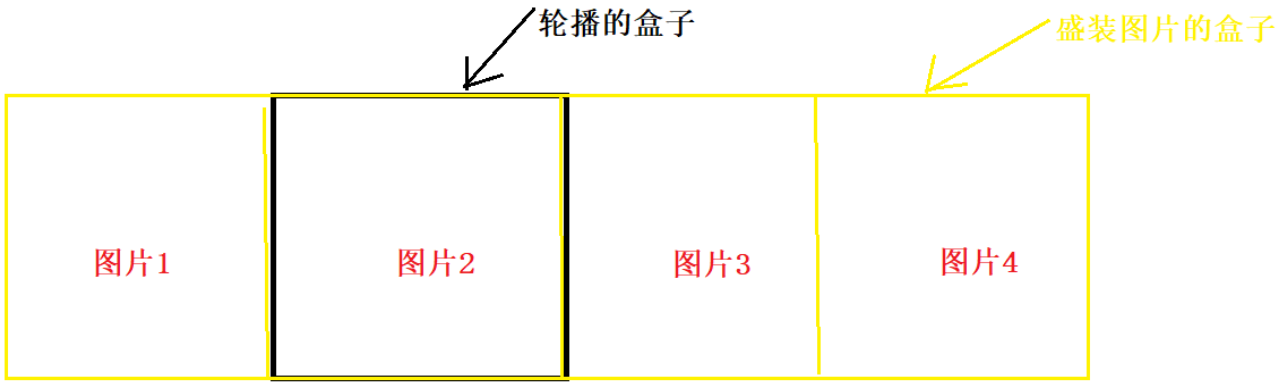
  ```
  <head>
  <title>Document</title>
      <style>
          div {
              width: 300px;
              height: 100px;
              background-color: yellow;
              box-shadow: 10px 10px 10px 5px #888888;
          }
      </style>
  </head>
  <body>
      <div></div>
  </body>
  </html>
  ```

- 放大时，在本案例中不推荐使用缩放scale，因为会将边框（阴影）一同放大。

2、制作轮播图

分析：假设有四张图片，图片在一个框框内轮播。首先，图片可以放在一个盒子里面（通过浮动，吧图片横向放在这个盒子里），最后变成盒子在框框里左移即可。（左移可以使用translate，也可以使用margin-left）



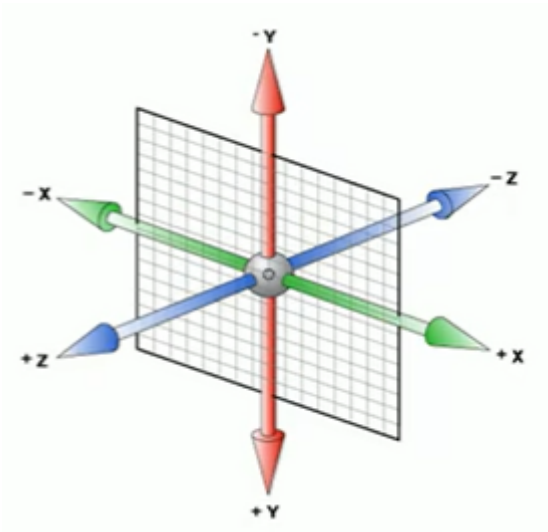

## 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
    <style>
        * {
            margin: 0px;
            padding: 0px;
        }
        .box {
            width: 300px;
            height: 300px;
            margin: 100px auto;
            overflow: hidden;
        }
        /* 盛装四张图片的，图片宽度总和就是该盒子的宽度。   轮播的时候移动该盒子即可*/
        .box1 {
            width: 1200px;
            height: 300px;
            margin: 0 auto;
            /* 通过行高去除图片空隙 */
            /* font-size: 0; */
            animation: one 10s linear 0s infinite;
        }
        img {
            width: 300px;
            height: 300px;
            float: left;
        }
        @keyframes one {
            0%,20% {
                transform: translate(0, 0);
            }
            25%,45% {
                transform: translate(-25%, 0);
            }
            50%,70% {
                transform: translate(-50%, 0);
            }
```

```
            75%,95% {
                transform: translate(-75%, 0);
            }
            100%{
                transform: translate(-100%,0);
            }
        }
    </style>
</head>
<body>
    <div class="box">
        <div class="box1">
            <img src="../图片/01.jpg" alt="">
            <img src="../图片/02.png" alt="">
            <img src="../图片/04.jpg" alt="">
            <img src="../图片/06.png" alt="">
        </div>
    </div>
</body>
</html>
```

# 七、3D转换

## 1、三维坐标系




- X轴（水平向右）：右边是正值，左边为负值
- Y轴（垂直向下）：下边是正值，上边是负值
- Z轴（垂直屏幕）：往外是正值，往内是负值

## 2、3D移动translate3d

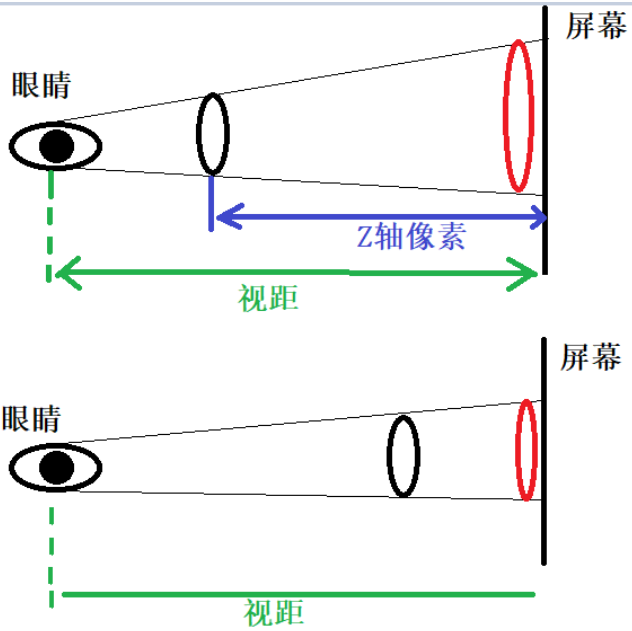
- transform:translateX(100px)，向右移动 100px
- transform:translateY(100px),向下移动100PX

- transform:translateZ(100px)，向眼前移动100PX

  **注意：translateZ的单位一般只用px，不用百分比。并且需要与透视perspective一起使用，否则看不出来效果**

  **简写方式：**transform:translate 3d（X,Y,Z)。**里面的X,Y,Z任意一个都不能省略，不需要移动就写0**

## （1）透视（perspective）




注意：
透视我们也称为视距：视距是指人眼到屏幕的距离。
距离眼睛越近的在屏幕上成像越大，距离越远的成像越小。

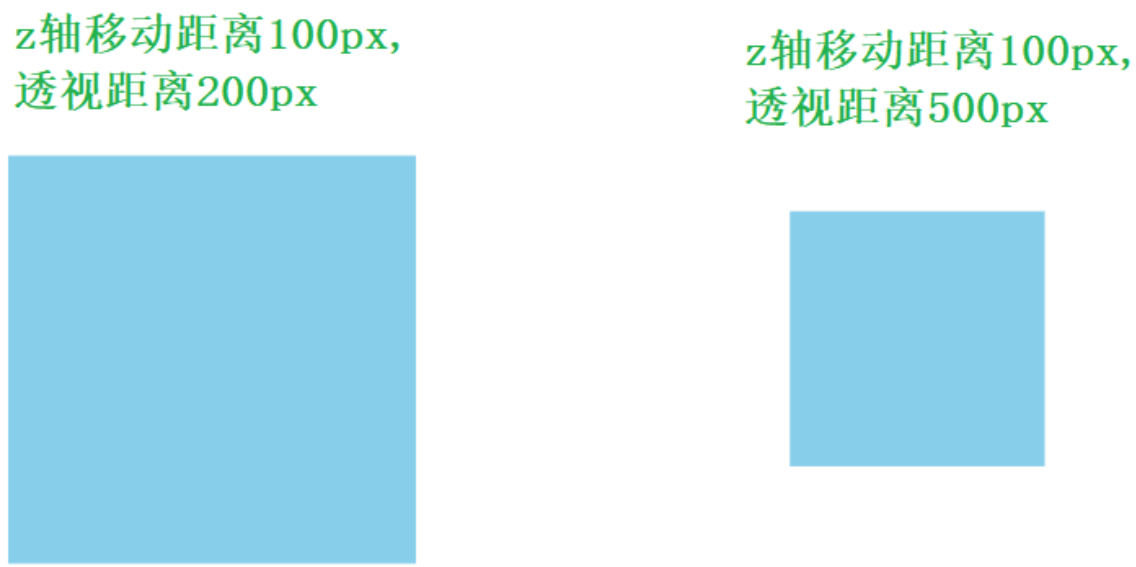
- 移动需要配合透视一起使用，**透视样式需要写到被观察元素的父盒子上面**
- 透视的单位是像素（**像素值越大，代表物体距离眼睛越远，成像越小**；像素值越小，代表物体距离眼睛越近，成像越大）（因为，当设置好z轴的移动距离之后，修改透视距离，那么改变的就是人眼到物理的距离）
- 同理，我们还可以修改z轴的像素，像素值越大，距离人眼越近，成像越大。（一般情况下做法是，固定透视距离，修改z轴移动距离）

### 示范

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
         body{
            /* perspective: 200px; */
            /* 像素值越大，也就是人眼距离屏幕越远，而z轴移动距离固定了，
            所以人眼距离物体距离就变大了，那么成像就越小 */
            perspective: 500px;
        }
        div{
            width: 100px;
            height: 100px;
            background-color: skyblue;
            /* 给当前盒子三个方向各移动100px，但是此时z轴看不出来任何效果，需要添加透视属性 */
            /* 因为透视属性需要添加在被观察元素的父盒子上面，而div的父元素是body，所以在body里面添加 */
            transform: translate3d(700px,100px,100px);
        }
    </style>
```

```
</head>
<body>
    <div></div>
</body>
</html>
```

**演示**




## 3、3D旋转rotate 3d

3D旋转可以让元素在三维平面内沿着X轴，Y轴，Z轴或者是自定义轴进行旋转

### （1）rotateX

例：transform:rotateX(45deg) 指沿着X轴正方向旋转45度。

体操运动员在单杠上旋转的时候，就是类似于沿着X轴方向旋转

度数是正数还是负数有不同的旋转方向。可以使用左手准则来判定。


**代码**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        body{
            perspective: 300px;
        }
        img{
            width: 100px;
            height: 100px;
            /* 转换成块元素，才可居中 */
            display: block;
            margin: 100px auto;

            /* 过渡一下更明显 */
            transition: all 1s;
        }
        /* 当鼠标放上去时开始旋转 */
        img:hover{
            transform: rotateX(45deg);
            /* 需要加上透视 效果更好，所以给父集body加透视*/
        }
    </style>
</head>
<body>
    <img src="../图片/01.jpg" alt="">
</body>
</html>
```

**演示**

## （2）rotateY

例：transform:rotateY(45deg) 指沿着Y轴正方向旋转45度。

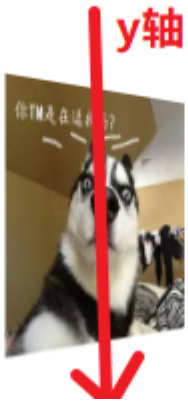
钢管舞，就是类似于沿着Y轴方向旋转

度数是正数还是负数有不同的旋转方向。也可以使用左手准则来判定（左手拇指指向Y轴的正方向，手指弯曲的方向就是度数为正时旋转的方向）


## 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
    <style>
        /* 加上透视 */
        body{
            perspective: 300px;
        }
        img{
            width: 100px;
            height: 100px;
            /* 转换成块级元素后居中 */
            display: block;
            margin: 200px auto;
            /* 加上过渡 */
            transition: all 1s;
        }
        img:hover{
            transform: rotateY(45deg);
        }
    </style>
</head>
<body>
    <img src="../图片/06.png" alt="">
</body>
</html>
```

### 演示




## （3）rotateZ

例：transform:rotateZ(45deg) 指沿着Z轴正方向旋转45度。

抽奖转盘就类似于沿着z轴旋转

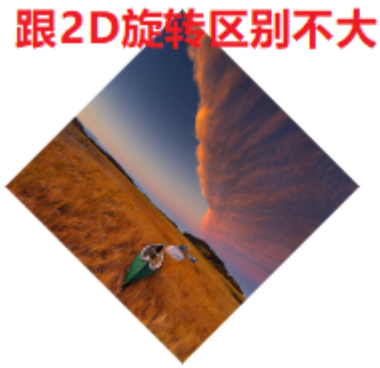
度数是正数还是负数有不同的旋转方向。也可以使用左手准则来判定（左手拇指指向Z轴的正方向，手指弯曲的方向就是度数为正时旋转的方向）

### 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        /* 加上透视 */
        body{
            perspective: 300px;
        }
        img{
            width: 100px;
            height: 100px;
            /* 转换成块级元素后居中 */
            display: block;
            margin: 200px auto;
            /* 加上过渡 */
            transition: all 1s;
        }
        img:hover{
            transform: rotateZ(45deg);
        }
    </style>
</head>
<body>
    <img src="../图片/04.jpg" alt="">
    <!-- 总结：因为Z轴是垂直与屏幕的，只能看到表面的部分，
        所以沿着z轴旋转的效果与2D旋转是一样的，所以加不加透视都无所谓 -->
```

```
</body>
</html>
```

### 演示




**总结：因为Z轴是垂直与屏幕的，只能看到表面的部分，所以沿着z轴旋转的效果与2D旋转是一样的，所以加不加透视都无所谓**

## （4）rotate 3d（了解）

例：transform：rotate3d（x,y,z,deg)：沿着自定义轴旋转，deg为角度

x,y,z,是表示旋转轴的矢量，是标示**是否（是：1；否：0）**希望沿着该轴旋转。

```
transform:rotate3d(1,0,0,45deg)沿着x轴旋转45度;
transform:rotate3d(1,1,0,45deg)x轴y轴都要旋转45度，也就是对角线方向旋转45度
```