一.1.transition(过渡)基本介绍

CSS中最简单的动画叫做 transition(转变)。通常，当一个元素的样式属性值发生变化时，
我们会立即看到页面元素发生变化，也就是页面元素从旧的属性值立即变成新的属性值的效果。
Transition(转变)

让页面元素不是立即的、而是慢慢的从一种状态变成另外一种状态，从而表现出一种动画过程

transition-property – 什么属性将用动画表现，例如，opacity。
transition-duration – 转变过程持续时间。
transition-timing-function – 转变时使用的调速函数(比如，linear、 ease-in 或自定义的 cubic bezier 函数)

实例：transition-三种属性的合体简写

```
div{
  opacity: 1;
  transition: opacity 1s linear;
}
div:hover {
  opacity: 0;
}
```

1. 扩展：

【opacity属性】
也是一个css3属性，该属性用于设置元素的不透明度级别，所有的浏览器都支持这个属性。
语法：opacity: value|inherit;
参数说明：
value ：规定不透明度。从 0.0 （完全透明）到 1.0（完全不透明）

2. 实例1：

```
div {
  transition-property: opacity, left;
  transition-duration: 2s, 4s;
}
//上面的演示规则中，opacity的变化过程将会持续2秒，而left值的变化过程将会持续4秒。
```

3. transition-property 属性用法

实例：
请把鼠标指针移动到蓝色的 div 元素上，就可以看到过渡效果。

```
<!DOCTYPE html>
<html>
<head>
<style>
div
{
```

```
    width:100px;
    height:100px;
    background:blue;
    transition-property: width;
    transition-duration: 2s;

    -moz-transition-property: width; /* Firefox 4 */
    -moz-transition-duration: 2s; /* Firefox 4 */

    -webkit-transition-property: width; /* Safari and Chrome */
    -webkit-transition-duration: 2s; /* Safari and Chrome */

    -o-transition-property: width; /* Opera */
    -o-transition-duration: 2s; /* Opera */
}
div:hover
{
    width:300px;
}
</style>
</head>
<body>
<div></div>
<p>请把鼠标指针移动到蓝色的 div 元素上，就可以看到过渡效果。</p>
<p><b>注释：</b>本例在 Internet Explorer 中无效。</p>
</body>
</html>
```

一.2.transform(转化)

定义和用法：transform 属性向元素应用 2D 或 3D 转换。该属性允许我们对元素进行旋转、缩放、移动或倾斜
文档：https://www.w3school.com.cn/css3/css3_2dtransform.asp

```
<!-- 将div旋转角度 -->
<!DOCTYPE html>
<html>
<head>
<style>
div
{
margin:30px;
width:200px;
height:100px;
background-color:yellow;
/* Rotate div */
transform:rotate(9deg);
-ms-transform:rotate(9deg); /* Internet Explorer */
-moz-transform:rotate(9deg); /* Firefox */
-webkit-transform:rotate(9deg); /* Safari 和 Chrome */
-o-transform:rotate(9deg); /* Opera */
}
</style>
</head>
<body>
<div>Hello World</div>
</body>
```

```
</html>
```

translate()
rotate()
scale()
skew()
matrix()

- translate() 方法
  通过 translate() 方法，元素从其当前位置移动，根据给定的 left（x 坐标） 和 top（y 坐标） 位置参数：

```
/*值 translate(50px,100px) 把元素从左侧移动 50 像素，从顶端移动 100 像素。*/
div
{
    transform: translate(50px,100px);
}
```

- 通过 rotate() 方法，元素顺时针旋转给定的角度。允许负值，元素将逆时针旋转。

```
/*值 rotate(30deg) 把元素顺时针旋转 30 度。*/
div
{
    transform: rotate(30deg);
}
```

- 通过 scale() 方法，元素的尺寸会增加或减少，根据给定的宽度（X 轴）和高度（Y 轴）参数：

```
/*值 scale(2,4) 把宽度转换为原始尺寸的 2 倍，把高度转换为原始高度的 4 倍。*/
div
{
    transform: scale(2,4);
}
```

- 通过 skew() 方法，元素翻转给定的角度，根据给定的水平线（X 轴）和垂直线（Y 轴）参数：
- matrix() 方法把所有 2D 转换方法组合在一起。
  matrix() 方法需要六个参数，包含数学函数，允许您：旋转、缩放、移动以及倾斜元素。

一. 3.animation(动画)

为页面设置动画时，往往会用到transition还有animation以及transfrom属性或者用到js。
其实通常情况下，对于使用js我们更加倾向于使用css来设置动画。

> 极端条件下，animation占用的资源相应的比transition多，所以如果能用transition实现，就尽量用transition来实现，如果追求复杂更自由的动画，就可以用animation。

transition 是可以为一个或者多个用数值表示的CSS属性发生变化时添加动画效果。

> transition的属性有transition-property（css属性name，transition效果）、transition-duration（动画持续多少秒）、transition-timing-function（动画的变化过程速度曲线）、transition-delay（动画开始的时候，也就是延迟多少秒）

```
<!-- 效果：div向右平移 -->
```

```
<!DOCTYPE html>
<html>
<head>
<style>
div
{
    width:100px;
    height:100px;
    background:red;
    position:relative;
    animation:mymove 5s infinite;
}
@keyframes mymove
{
    from {left:0px;}
    to {left:200px;}
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

> 同时还可以这样想，transition是从：hover延伸出来的，不管是动态设置的还是非动态设置的过渡效果，只要过渡效果指定的属性值发生了变化就会触发过渡效果。

> 而animation是从flash延伸出来的，使用关键帧的概念，如果是非动态设置的，那么页面加载完后就会触发动画效果；如果是动态设置的，那么设置完后（假设没有设置 delay）就会触发动画效果。后面再改变属性值也不会触发动画效果了，除了一种情况（这种情况不会触发transition定义的过渡效果），就是元素从 display:none 状态变成非 display:none 状态时，也会触发动画效果。

总结区别

- 1、transition 需要去触发比如：点击事件、鼠标移入事件；而 animation 可以配合 @keyframe 可以不触发事件就触发这个动画
- 2、transition 触发一次播放一次；而 animation 则是可以设置很多的属性，比如循环次数，动画结束的状态等等；
- 3、transition关注的是样式属性的变化，属性值和时间的关系是一个三次贝塞尔曲线；而animation作用于元素本身而不是样式属性，可以使用关键帧的概念，可以实现更自由的动画效果；
- 4、在性能方面：浏览器有一个主线程和排版线程；主线程一般是对 js 运行的、页面布局、生成位图等等，然后把生成好的位图传递给排版线程，而排版线程会通过 GPU 将位图绘制到页面上，也会向主线程请求位图等等；我们在用使用 aniamtion 的时候这样就可以改变很多属性，像我们改变了 width、height、postion 等等这些改变文档流的属性的时候就会引起，页面的回流和重绘，对性能影响就比较大，但是我们用 transition 的时候一般会结合 tansfrom 来进行旋转和缩放等不会生成新的位图，当然也就不会引起页面的重排了。

二.svg

1.一种使用XML描述的2D图形的语言
2.SVG基于XML意味着，SVG DOM中的每个元素都是可用的，可以为某个元素附加Javascript事件处理器。
3.在 SVG 中，每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化，那么浏览器能够自动重现图形。

三.Canvas

1.通过Javascript来绘制2D图形。
2.是逐像素进行渲染的。
3.其位置发生改变，会重新进行绘制。

svg和Canvas的区别

Canvas
1.依赖分辨率
2.不支持事件处理器
3.弱的文本渲染能力
4.能够以 .png 或 .jpg 格式保存结果图像
5.最适合图像密集型的游戏，其中的许多对象会被频繁重绘

svg
1.不依赖分辨率
2.支持事件处理器
3.最适合带有大型渲染区域的应用程序（比如谷歌地图）
4.复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）
5.不适合游戏应用

四.webGL

WebGL是一种3D绘图标准，这种绘图技术标准允许把JavaScript和OpenGL ES 2.0结合在一起，通过增加OpenGL ES 2.0的一个JavaScript绑定，WebGL可以为HTML5 Canvas提供硬件3D加速渲染，这样Web开发人员就可以借助系统显卡来在浏览器里更流畅地展示3D场景和模型了，还能创建复杂的导航和数据视觉化。显然，WebGL技术标准免去了开发网页专用渲染插件的麻烦，可被用于创建具有复杂3D结构的网站页面，甚至可以用来设计3D网页游戏等等。

WebGL和three.js

通过webgl可以直接使用显卡的计算资源，创建高性能的二维和三维计算机图形，然后在JavaScript里直接使用webGL编程，创建三维场景并生成动画，这个过程非常复杂，而且容易出错。three.js库可以简化这个过程。
three.js是以webgl为基础的库，封装了一些3D渲染需求中重要的工具方法与渲染循环。

window.requestAnimationFrame()

> 告诉浏览器——你希望执行一个动画，并且要求浏览器在下次重绘之前调用指定的回调函数更新动画。该方法需要传入一个回调函数作为参数，该回调函数会在浏览器下一次重绘之前执行

HTML5/CSS3时代，我们要在web里做动画选择其实已经很多了:
你可以用CSS3的animattion+keyframes;
你也可以用css3的transition;

当然最原始的你还可以使用window.setTimout()或者window.setInterval()通过不断更新元素的状态位置等来实现动画，前提是画面的更新频率要达到每秒60次才能让肉眼看到流畅的动画效果。

现在又多了一种实现动画的方案，那就是还在草案当中的window.requestAnimationFrame()方法。
初识requestAnimationFrame:
https://www.cnblogs.com/Wayou/p/requestAnimationFrame.html

```
setCurrentIndex(Element_offsetWidth, Element_offsetLeft) {
  const nav = this.$refs.nav;
  const nav_offestwidth = nav.offsetWidth;
  const to =
    Element_offsetLeft - (nav_offestwidth - Element_offsetWidth) / 2;
  //解析：动画，这二者的效果是一样的
  // this.$refs.nav.scrollLeft = to
  this.scrollLeftTo(nav, to, 0.3);
},
scrollLeftTo(scroller, to, duration) {
  let count = 0;
  let from = scroller.scrollLeft;
  let frames = duration === 0 ? 1 : Math.round((duration * 1000) / 16);
  let _this = this;
  function animate() {
    scroller.scrollLeft += (to - from) / frames;
    if (++count < frames) {
      _this.scrollLeftRafId = _this.raf(animate);
    }
  }
  animate();
},
raf(fn) {
  return window.requestAnimationFrame.call(window, fn);
},
```