CSS动画的两大组成部分：transition和animation。

第一部分：CSS Transition

1.1 基本用法

在CSS 3引入Transition（过渡）这个概念之前，CSS是没有时间轴的。也就是说，所有的状态变化，都是即时完成。

如：当鼠标放置于缩略图之上，缩略图会迅速变大。注意，缩略图的变大是瞬间实现的。下面是代码，相当简单。

```
1  img{
2   height:15px;
3   width:15px;
4  }
5
6  img:hover{
7   height: 450px;
8   width: 450px;
9  }
10
11
```

transition的作用在于，指定状态变化所需要的时间。

```
1  img{
2   transition: 1s;
3  }
```

上面代码指定，图片放大的过程需要1秒，效果如下。

我们还可以指定transition适用的属性，比如只适用于height。

```
1  img{
2   transition: 1s height;
3  }
```

这样一来，只有height的变化需要1秒实现，其他变化（主要是width）依然瞬间实现。

1.2 transition-delay

在同一行transition语句中，可以分别指定多个属性。

```
img{
 transition: 1s height, 1s width;
}
```

但是，这样一来，height和width的变化是同时进行的，跟不指定它们没有差别。

我们希望，让height先发生变化，等结束以后，再让width发生变化。实现这一点很容易，就是为width指定一个delay参数。

```
img{
 transition: 1s height, 1s 1s width;
}
```

width在1秒之后，再开始变化，也就是延迟（delay）1秒，可看效果。

delay的真正意义在于，它指定了动画发生的顺序，使得多个不同的transition可以连在一起，形成复杂效果。

1.3 transition-timing-function

transition的状态变化速度（又称timing function），默认不是匀速的，而是逐渐放慢，这叫做ease。

```
img{
 transition: 1s ease;
}
```

除了easel以外，其他模式还包括:

```
linear：匀速
ease-in：加速
ease-out：减速
cubic-bezier函数：自定义速度模式
```

最后那个cubic-bezier，可以使用工具网站来定制。

```
img{
 transition: 1s height cubic-bezier(.83,.97,.05,1.44);
}
```

上面的代码会产生一个最后阶段放大过度、然后回缩的效果。

1.4 transition的各项属性

transition的完整写法如下。

```
img{
 transition: 1s 1s height ease;
}
```

这其实是一个简写形式，可以单独定义成各个属性。

```
img{
 transition-property: height;
 transition-duration: 1s;
 transition-delay: 1s;
 transition-timing-function: ease;
}
```

1.5 transition的使用注意

（1）目前，各大浏览器（包括IE 10）都已经支持无前缀的transition，所以transition已经可以很安全地不加浏览器前缀。
（2）不是所有的CSS属性都支持transition，完整的列表查看这里，以及具体的效果。
（3）transition需要明确知道，开始状态和结束状态的具体数值，才能计算出中间状态。比如，height从0px变化到100px，transition可以算出中间状态。但是，transition没法算出0px到auto的中间状态，也就是说，如果开始或结束的设置是height: auto，那么就不会产生动画效果。类似的情况还有，display: none到block，background: url(foo.jpg)到url(bar.jpg)等等。

1.6 transition的局限

transition的优点在于简单易用，但是它有几个很大的局限。
（1）transition需要事件触发，所以没法在网页加载时自动发生。
（2）transition是一次性的，不能重复发生，除非一再触发。
（3）transition只能定义开始状态和结束状态，不能定义中间状态，也就是说只有两个状态。
（4）一条transition规则，只能定义一个属性的变化，不能涉及多个属性。
CSS Animation就是为了解决这些问题而提出的。

第二部分：CSS Animation

2.1 基本用法

首先，CSS Animation需要指定动画一个周期持续的时间，以及动画效果的名称。

```
div:hover {
 animation: 1s rainbow;
}
```

上面代码表示，当鼠标悬停在div元素上时，会产生名为rainbow的动画效果，持续时间为1秒。为此，我们还需要用keyframes关键字，定义rainbow效果。

```
@keyframes rainbow {
 0% { background: #c00; }
 50% { background: orange; }
 100% { background: yellowgreen; }
}
```

上面代码表示，rainbow效果一共有三个状态，分别为起始（0%）、中点（50%）和结束（100%）。如果有需要，完全可以插入更多状态。效果如下。

默认情况下，动画只播放一次。加入infinite关键字，可以让动画无限次播放。

```
div:hover {
 animation: 1s rainbow infinite;
}
```

也可以指定动画具体播放的次数，比如3次。

```
div:hover {
 animation: 1s rainbow 3;
}
```

2.2 animation-fill-mode

动画结束以后，会立即从结束状态跳回到起始状态。如果想让动画保持在结束状态，需要使用animation-fill-mode属性。

```
div:hover {
 animation: 1s rainbow forwards;
}
```

forwards表示让动画停留在结束状态，效果如下。

animation-fill-mode还可以使用下列值。

```
（1）none：默认值，回到动画没开始时的状态。
（2）backwards：让动画回到第一帧的状态。
（3）both：根据animation-direction（见后）轮流应用forwards和backwards规则。
```

2.3 animation-direction

动画循环播放时，每次都是从结束状态跳回到起始状态，再开始播放。animation-direction属性，可以改变这种行为。

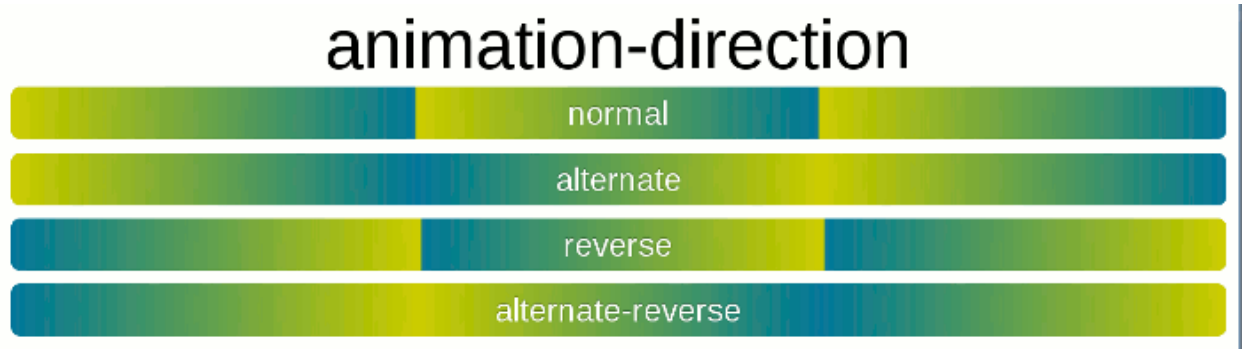
下面看一个例子，来说明如何使用animation-direction。假定有一个动画是这样定义的。

```
@keyframes rainbow {
 0% { background-color: yellow; }
 100% { background: blue; }
}
```

默认情况是，animation-direction等于normal。

```
div:hover {
 animation: 1s rainbow 3 normal;
}
```

此外，还可以等于取alternate、reverse、alternate-reverse等值。它们的含义见下图（假定动画连续播放三次）。



简单说，animation-direction指定了动画播放的方向，最常用的值是normal和reverse。浏览器对其他值的支持情况不佳，应该慎用。

2.4 animation的各项属性

同transition一样，animation也是一个简写形式。

```
div:hover {
 animation: 1s 1s rainbow linear 3 forwards normal;
}
```

这是一个简写形式，可以分解成各个单独的属性。

```
div:hover {
 animation-name: rainbow; // 动画名称
 animation-duration: 1s; // 动画执行时间
 animation-timing-function: linear; // transition的状态变化速度
 animation-delay: 1s; // 动画延迟1秒开始执行
 animation-fill-mode:forwards; // 动画结束以后停留的状态
 animation-direction: normal;
 animation-iteration-count: 3; // 动画循环播放次数
}
```

2.5 keyframes的写法

keyframes关键字用来定义动画的各个状态，它的写法相当自由。

```
@keyframes rainbow {
 0% { background: #c00 }
 50% { background: orange }
 100% { background: yellowgreen }
}
```

0%可以用from代表，100%可以用to代表，因此上面的代码等同于下面的形式。

```
@keyframes rainbow {
 from { background: #c00 }
 50% { background: orange }
 to { background: yellowgreen }
}
```

如果省略某个状态，浏览器会自动推算中间状态，所以下面都是合法的写法。

```
@keyframes rainbow {
 50% { background: orange }
 to { background: yellowgreen }
}

@keyframes rainbow {
 to { background: yellowgreen }
}
```

甚至，可以把多个状态写在一行。

```
@keyframes pound {
 from, to { transform: none; }
 50% { transform: scale(1.2); }
}
```

另外一点需要注意的是，浏览器从一个状态向另一个状态过渡，是平滑过渡。steps函数可以实现分步过渡。

```
div:hover {
 animation: 1s rainbow infinite steps(10);
}
```

2.6 animation-play-state

有时，动画播放过程中，会突然停止。这时，默认行为是跳回到动画的开始状态。

如最开始最默认的例子看，如果鼠标移走，色块立刻回到动画开始状态。

如果想让动画保持突然终止时的状态，就要使用animation-play-state属性。

```
div {
 animation: spin 1s linear infinite;
 animation-play-state: paused;
```

```
}

div:hover {
 animation-play-state: running;
}
```

上面的代码指定，没有鼠标没有悬停时，动画状态是暂停；一旦悬停，动画状态改为继续播放。效果如下。

2.7 浏览器前缀

目前，IE 10和Firefox（>= 16）支持没有前缀的animation，而chrome不支持，所以必须使用webkit前缀。

也就是说，实际运用中，代码必须写成下面的样子。

```
div:hover {
 -webkit-animation: 1s rainbow;
 animation: 1s rainbow;
}

@-webkit-keyframes rainbow {
 0% { background: #c00; }
 50% { background: orange; }
 100% { background: yellowgreen; }
 }

 @keyframes rainbow {
  0% { background: #c00; }
  50% { background: orange; }
  100% { background: yellowgreen; }
 }
```