深入理解SVG feDisplacementMap滤镜及实际应用

这篇文章发布于 2017年12月14日，星期四，21:58，归类于 SVG相关。 阅读 8795 次, 今日 8 次 7 条评论

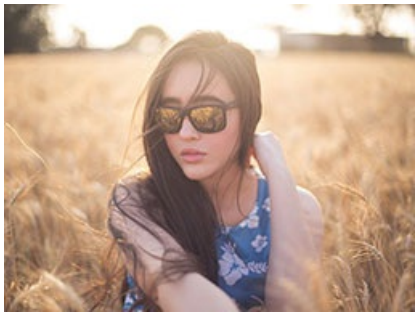
补充于2018-05-07

应该就是最近一两个版本的Chrome吧，居然feDisplacementMap滤镜实时效果都没了，不支持了，匪夷所思，Firefox还是有的。

一、先看一个SVG feDisplacementMap滤镜实现的效果

在Chrome或者Firefox浏览器下点击下面的按钮或者图片，就会看到，从点击位置开始，有水波荡漾的特效：

点击我



上面的按钮和图片就是普通的 `<button>` 和 `<img>` 元素，也就意味着基本上这个效果可以作用于几乎任何的HTML元素，适用场景非常广泛。

如何应用上面波纹效果呢?

本文后面会介绍。

//zxx :如果对效果感兴趣，对 `feDisplacementMap` 不感兴趣，可以直接点击上面文字链接定位到下面。

二、SVG feDisplacementMap滤镜简介

CSS3中有很多基本的滤镜，例如模糊，下阴影，反相，灰度，亮度控制等等，使用非常方便，以至于Chrome和Firefox等浏览器（目前Safari 11并未支持）的 `canvas` 也增加了同语法的 `filter` API，例如：

```
context.filter = "blur(15px)";
```

很棒！然而，对于SVG滤镜，总要一分为二得看待。

首先，CSS3支持的所有这些滤镜SVG都是可以实现的，注意自己的措辞，“可以实现”。但是如果要问容不容易实现呢，那和CSS比，那就是雅思跟四级的区别，除了高斯模糊等少数几个滤镜，其他滤镜效果只要自己重新进行组合定制。

这并不是说SVG滤镜不好，而是本身定位和策略的差异。

SVG滤镜提供的是更基础，更底层的控制手段；而CSS的这些滤镜是看成是经过封装处理后暴露的高度定制的API。你可以看成是原生JavaScript和jQuery的区别。

例如SVG滤镜可以对特定通道颜色进行细致的处理，比方说把图片里面所有红色全部去掉。但是在CSS中，目前却没有这样的功能。

所以SVG滤镜要更加强大，更加灵活，虽然学习成本有点高，但是一旦熟练掌握，深入了解，我们就能自己创造出很多非常精湛的效果。尤其现在现代浏览器下普通HTML元素也能直接使用CSS `filter` 属性应用SVG滤镜，这是说我们的web交互动效可以更上一层楼。可见好好学习SVG还是很有价值和前景的。

不如就先从本文的 `feDisplacementMap` 滤镜开刀。

SVG本质是XML，和HTML元素同宗，因此，其所有的滤镜都是使用标签和属性实现的。所以，这里的 `feDisplacementMap` 滤镜实际上指的是 `<feDisplacementMap>` 元素。

在SVG中，所以的滤镜元素标签都是以字母 `fe` 打头，例如高斯模糊滤镜 `<feGaussianBlur>` 等。

配合一些SVG元素属性，就能实现我们想要的滤镜效果了。

实际上，学习SVG滤镜难的并不是这些属性的掌握和了解，而是并不知道滤镜本身是个什么东西，作用是什么？原理是什么？

例如这里的 `feDisplacementMap` ，看名称这么长，吓都吓死了，谁还敢学？

feDisplacementMap滤镜简介

`feDisplacementMap` 实际上是一个位置替换滤镜，就是改变元素和图形的像素位置的。

`map` 含义和ES5中数组的 `map` 方法是一样的，遍历原图形的所有像素点，使用 `feDisplacementMap` 重新映射替换一个新的位置，形成一个新的图形。

因此， `feDisplacementMap` 滤镜在业界的主流应用是对图形进行形变，扭曲，液化。

例如本文实现的波纹效果，实际上就是将原始图形按照水波的波纹形状进行扭曲实现。

而这个水波效果的SVG滤镜实际上就一小撮代码，你往页面上任意位置一放，CSS就这么一应用，效果就出来了。SVG和CSS代码如下：

```
<svg style="position:absolute;height:0;clip:rect(0 0 0 0);">
    <defs>
        <filter id="filter-ripple">
            <feImage xlink:href="./map.png" x="0" y="0" width="512" height="512" result="r
ipple"></feImage>
            <feDisplacementMap xChannelSelector="G" yChannelSelector="R" color-interpolati
on-filters="sRGB" in="SourceGraphic" in2="ripple" scale="80"></feDisplacementMap>
            <feComposite operator="in" in2="ripple"></feComposite>
        </filter>
    </defs>
</svg>

.element {
  filter: url(#filter-ripple);
}
```

哈哈，是不是有种看天书的感觉，不要紧张，其实都是纸老虎。

三、SVG feDisplacementMap滤镜深入了解

`feDisplacementMap` 对图形进行位置隐射，那一定有一个映射公式，只要我们理解这个映射公式，那基本上你掌握就不远了。

公式如下（摘自MDN文档）：

```
P'(x,y) ← P( x + scale * (XC(x,y) - 0.5), y + scale * (YC(x,y) - 0.5))
```

解释下：

· `P'(x,y)` 指的是转换之后的 `x, y` 坐标。
· `x + scale * (XC(x,y) - 0.5), y + scale * (YC(x,y) - 0.5)` 指的是具体的转换规则。
· `XC(x,y)` 表示当前 `x,y` 坐标像素点其X轴方向上设置的对应通道的计算值，范围是0~1。
· `YC(x,y)` 表示当前 `x,y` 坐标像素点其Y轴方向上设置的对应通道的计算值，范围是0~1。
· `-0.5` 是偏移值，因此 `XC(x,y) - 0.5` 范围是 `-0.5~0.5` ， `YC(x,y) - 0.5` 范围也是 `-0.5~0.5` 。
· `scale` 表示计算后的偏移值相乘的比例， `scale` 越大，则偏移越大。

再用一句话解释就是，根据设定的通道颜色对原图的 `x, y` 坐标进行偏移。

下面，我们回到SVG `<feDisplacementMap>` 滤镜代码部分：

```
<feDisplacementMap xChannelSelector="G" yChannelSelector="R" color-interpolation-filters="
sRGB" in="SourceGraphic" in2="ripple" scale="80"></feDisplacementMap>
```

此时，这段SVG代码就容易理解多了，一个一个来：

- `xChannelSelector` 对应 `XC(x,y)` ，表示X轴坐标使用的是哪个颜色通道进行位置偏移。我们应该都知道，颜色有RGBA四个通道，R表示red红色，G表示green绿色，B表示blue蓝色，A表示alpha可以理解为透明度。因此， `xChannelSelector` 属性值可以是R、G、B、A中的任意一个，默认是A，基于透明度进行位置偏移。

  假设 `xChannelSelector="A"` ，请问，如果用来map映射图片像素点是完全不透明，也就是A的值是1，请问，最终映射后的图片是如何位置偏移的？

  是原地不动吗？不是的！会看到原始图片所有的像素点都往左移动了。因为此时的 `XC(x,y)` 是 `1` ，套用公式， `x + scale * (XC(x,y) - 0.5) = x + scale * 0.5` ，此时 `scale` 为 `80` 。也就是原来的 `0,0` 坐标变成了 `40,0` 坐标，于是，原始图片所有点都往左移动了 `40` 像素。

  注意这里的位移逻辑，偏移为正，是反向的位移。

  什么时候图片原地不动呢，那就是图片透明度是 `0.5` 的时候，也就是 `50%` 透明。此时 `scale * (XC(x,y) - 0.5)` 的计算值是 `0` .

- `yChannelSelector` 和 `xChannelSelector` 类似，只是一个是x轴（横轴）方向，一个是y轴（纵轴）方向，其它都类似，不赘述。
- `color-interpolation-filters` 表示滤镜对颜色进行计算时候采用的颜色模式类型。分为 `linearRGB` （默认值）和 `sRGB` ， `sRGB` 是我们平常用的RGB颜色，因此，这里设置为 `sRGB` 方便我们理解；

翌日更新
测试发现目前最新版本Safari 11并不支持 `sRGB`，只能以 `linearRGB` 渲染。

加上目前Safari 11仅支持SVG元素滤镜。因此，若想使Safari浏览器下有完全一致的效果，需要：1. 使用SVG元素；2. `linearRGB` 通道值转换成对应的 `sRGB` 值。

对于单一图片元素，简单图形，可行（如最后的鲸鱼游动效果）。复杂HTML元素，建议放弃Safari。

更多linearRGB和sRGB知识可以参见这篇文章："了解LinearRGB和sRGB以及之间的JS相互转换"。

- `in` 和 `in2` 都表示输入，支持的属性值也都是一样的，包括固定的属性值关键字 `SourceGraphic`，`SourceAlpha`，`BackgroundImage`，`BackgroundAlpha`，`FillPaint`，`StrokePaint`；以及自定义的滤镜的原始引用，例如这里的 `ripple`，引用的就是 `<feImage>` 元素输出的 `result` 值。

  那 `in` 和 `in2` 有什么区别呢?

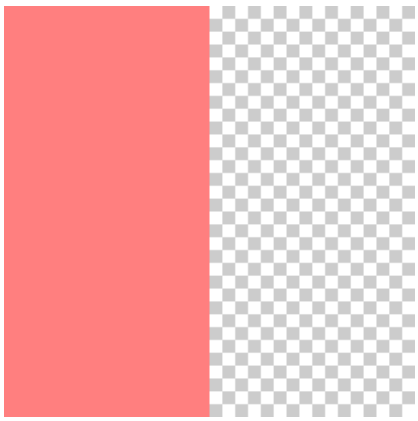
  这个需要看SVG元素的，对于 `<feDisplacementMap>` 元素，`in` 表示输入的原始图形，`in2` 表示用来映射的图形。

  至于它们的属性值，目前阶段，大家无需深入了解每个属性值的作用，记住使用 `in="SourceGraphic"` 就好了，也就是使用该 `filter` 元素的图形作为原始图像。`in2` 属性值也一定是 `<feImage>` 元素的 `result` 属性值就好了。

- `scale` 很好理解，就是公式里面的缩放比例，可正可负，默认是 `0`。通常使用正数值处理，值越大，偏移越大。

下面我们通过一个案例，加深对 `<feDisplacementMap>` 滤镜各个属性值的理解，主要是 `xChannelSelector`，`yChannelSelector` 和 `scale` 这3个属性。

假设我们的Map映射图片是下面这张图：



左边一半颜色是RGB(255,127,127)，也就是R通道值是 `1`，G和B通道计算值是 `0.5`，右半边则是完全透明。

套用下面的SVG代码：

```
<svg>
    <defs>
        <filter id="filter-ripple">
            <feImage xlink:href="./map.png" x="0" y="0" width="256" height="256" result="r
```

```
ipple"></feImage>
            <feDisplacementMap xChannelSelector="R" yChannelSelector="G" color-interpolati
on-filters="sRGB" in="SourceGraphic" in2="ripple" scale="80"></feDisplacementMap>
            <feComposite operator="in" in2="ripple"></feComposite>
        </filter>
    </defs>
</svg>
<svg width="256" height="192" style="outline:1px dotted;">
    <image xlink:href="http://image.zhangxinxu.com/image/study/s/s256/mm1.jpg" width="256"
height="192" filter="url(#filter-ripple)"></image>
</svg>
```

结果，当 `scale` 为 `0` 的时候，效果这样：

当 `scale` 为 `200` 的时候，效果这样：

效果表现为2个特点：

1. 右半区域一直都是透明的。当应用 `feDisplacementMap` 滤镜的通道不是透明度时候，映射效果会把这个透明度继承过来。例如，假设map.png右半区域颜色是 `rgba(255,127,127,.1)`，则效果会是下图这样：

2. 应用滤镜的图片只是单纯的水平移动。为什么呢？

这是因为 `xChannelSelector="R"`，`yChannelSelector="G"`，水平方向基于 `"R"` 颜色通道进行位移，垂直方向基于 `"G"` 颜色通道进行位移。

由于RGB(255,127,127)颜色的R值是 `1`，G值是 `0.5`，也就是 `XC(x,y)` 是 `1`，`YC(x,y)` 是 `0.5`，套用公式 `P( x + scale * (XC(x,y) - 0.5), y + scale * (YC(x,y) - 0.5))`，可以得到终坐标为：`P( x + scale * 0.5, y + 0)`，于是随着 `scale` 变化，最终图片仅在水平方向发生了移动。

眼见为实，您可以狠狠地点击这里：feDisplacementMap滤镜作用原理示意demo

更多滤镜效果展示

为了方便大家更直观感受滤镜带来的扭曲效果，我制作了很多各式各样的映射图，大家可以体验下应用滤镜后的效果。

您可以狠狠地点击这里：feDisplacementMap滤镜效果走马观花demo

某效果截图：

白色RGB值最大 `255`，黑色RGB值最小 `0`,。也就是对于黑白图片而言，颜色越深，对应的图片区域越往右下方移动；颜色越浅，对应的图片区域越往左上方移动。

因此，上面人物的扭曲效果就好理解了。

什么时候图片不移动呢？

那就是我们的 `<feImage>` 颜色是 `50` 度灰的时候，RGB表示为 `rgb(127,127,127)`，16进制色值为 `#7f7f7f`。

因此，在 `feDisplacementMap` 世界里，五十度灰可以等同于“纹丝不动”的意思，例如，人被捆绑的时候是不能动的。

最终效果还与 `<feImage>` 尺寸和位置有关

假如说我们图片很小，`<feImage>` 尺寸很大，则仅原图和 `<feImage>` 对应的像素位置会发生位移。所以，通常，我们会设置 `<feImage>` 尺寸和被应用图形元素尺寸一致，当然，这并不绝对。

四、如何使用SVG feDisplacementMap滤镜实现水波特效

实现水波特效难点不在于滤镜，而在于 `<feImage>` 图片。

如果对最终的水波特效要求不是很高，我们可以直接使用SVG绘制一个嵌套环形渐变，完整滤镜代码如下：

```
<svg width="256" height="192">
  <defs>
    <radialGradient id="rg" r=".9">
      <stop offset="0%" stop-color="#f00"></stop>
      <stop offset="10%" stop-color="#000"></stop>
      <stop offset="20%" stop-color="#f00"></stop>
      <stop offset="30%" stop-color="#000"></stop>
      <stop offset="40%" stop-color="#f00"></stop>
      <stop offset="50%" stop-color="#000"></stop>
      <stop offset="60%" stop-color="#f00"></stop>
```

```
      <stop offset="70%" stop-color="#000"></stop>
      <stop offset="80%" stop-color="#f00"></stop>
      <stop offset="90%" stop-color="#000"></stop>
      <stop offset="100%" stop-color="#f00"></stop>
    </radialGradient>
    <rect id="witness" width="256" height="192" fill="url(#rg)"></rect>

     <filter id="filter-ripple">
          <feImage result="pict2" xlink:href="#witness" x="0" y="0" width="256" height="19
2"></feImage>
          <feDisplacementMap scale="10" xChannelSelector="R" yChannelSelector="R" in2="pic
t2" in="SourceGraphic">
       </feDisplacementMap>
    </filter>
  </defs>
</svg>
```

`x`轴，`y`轴均采用红色作为映射通道，绘制的渐变环效果如下：

然后10倍`scale`后的扭曲效果大概这样子：

您可以狠狠地点击这里：[简易水波扭曲demo]

然后配合JS改变渐变偏移或者`feDisplacementMap`的`scale`值，都能有水波效果。

然而上面实现的这个水波效果实际上是比较粗糙的，真正业界用来实现水波效果几乎都是用的下面这张`<feImage>`图：

中间区域使用五十度灰色，红色色带锥形交错，中间是黑色带。`x`轴，`y`轴采用红色和绿色通道来映射，最终形成的水波效果就非常接近于自然世界中的水波涟漪。

然而，业界通常解决方案就是使用一张PNG图片来解决问题，但会带来另外一个问题，那就是这张图片的尺寸实在是太大了，如果是512*512规格，则PNG图片大小由212K，这是非常惊人的，对于一个小小的水波特效，如果加载的资源非常大，那是性价比非常低的一件事情。而且由于这张图片色彩非常丰富，是无法进行PNG压缩的。导致无法在实际项目中大规模应用。

那有什么办法解决这个问题呢?

有！

我们可以使用`canvas`把这个经典的水波映射图绘制在画布上，然后转换成base64地址，赋予`<feImage>`元素的`xlink:href`属性即可！

绘制分三步：1. 50度灰色背景；2. 红绿相间锥形渐变；3. 交错暗色条纹。

具体细节不表。然后下图就是我绘制出来的最终效果：

于是`<feImage>`图片现在只需要几十行JS代码就可以得到了，实际项目中大规模应用水波特效的成本就大大降低了，基本上可以在任何项目中渐进增强使用。

项目中应用水波特效

很简单，我已经把水波特效功能写成一段JS了，ripple-min.js，直接引入就可以使用了。代码示意如下：

1. 引入JS

```
<script src="./ripple-min.js"></script>
```

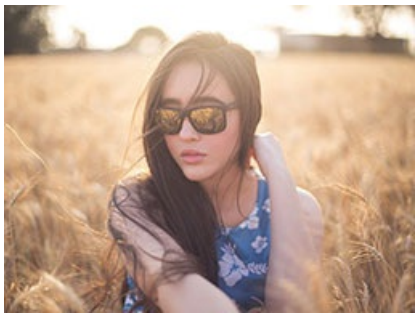
2. 调用rippleEffect方法，语法为：

```
rippleEffect(dom);
```

`dom`表示页面上希望点击出现水波的DOM元素，就这么简单。例如下面效果使用JS就是：

```
rippleEffect(document.getElementById('button'));
rippleEffect(document.getElementById('img'));
```

点击我



水波特效基本原理

插入SVG滤镜相关代码，当点击目标元素时候，根据目标元素尺寸，确定`<feDisplacementMap>`元素的合适的`scale`大小，以及`<feImage>`的偏移位置和大小，借助我在这个项目（https://github.com/zhangxinxu/Tween）中的`animation.js`和线性运动算法，实现属性值的连续变化。

于是波纹效果达成。

水波特效兼容性

PC端，Chrome和Firefox浏览器支持很好！Safari浏览器仅仅支持SVG元素应用`filter`属性的效果，对于普通元素，虽然能够识别`filter`样式，但是渲染上是有明显问题的，元素会直接看不见，也就是并不支持。PC和移动端均作了测试，版本11，均不支持。应该以后会支持。

因此特效最大的受益网站还是PC端网页项目，因为现在很多网站七八成用户都是chrome内核。对于不支持此特效的浏览器，对原有的功能和效果并无任何影响，因此可以放心大胆使用。

`ripple-min.js`大小仅几K大小，对于如今的web应用而言，大小基本上可以忽略不计。可以说是一个低成本，低影响，高收益，适用广泛的交互效果。

五、借助SVG feDisplacementMap滤镜实现其它效果

知道了`feDisplacementMap`滤镜作用原理，我们就能在这个基础上进行创造，实现其它一些我们想实现的效果。例如：

点击此链接：http://www.cssworld.cn/

可以看到中间的鲸鱼尾巴是上下游动的。

实现原理：
使用 `canvas` 绘制下图所示 `<feImage>` 图：

左侧大片区域是50度灰，也就是鲸鱼的左边大部分是位置不动的，右侧是一个渐变，从灰色到rgb(255,127,0)的渐变。

这里的渐变色值 `rgb(255,127,0)` 一看就知道不是随便设置的数值，中间的G绿色为127，由于通道设置为 `xChannelSelector="G" yChannelSelector="B"` ，也就是X轴使用绿色位移，由于正好127色值为 `0.5` ，所以鲸鱼水平方向是没有任何移动的。但垂直方向采用的通道是 `B` ，随着渐变进行，从 `127->0` ，也就是越往尾部，其上下偏移越大，于是实现视觉上的尾巴上下游动效果。

六、结束语

本文就是最近相关知识研究的一点小心得。

如何在项目中应用水波效果，这个是即插即用的，当下的就能带来收益。

至于 `feDisplacementMap` 相关知识，是留给后来人的，那些致力于在图形交互领域有所成长的小伙伴。

`feDisplacementMap` 滤镜的原理和特性在不同的语言环境中，其实都是相通的，例如WebGL中有类似滤镜，以后，很有可能CSS也会类似功能的属性。

怎么讲呢！图形图像领域的基础知识吧！

感谢阅读，行文仓促，出错难免，欢迎指出！

（本篇完） // 想要打赏？点击这里。有话要说？点击这里。

« canvas实现iPhoneX炫彩壁纸屏保外加pixi.js流体动效 小tip: 了解LinearRGB和sRGB以及使用JS相互转换 »

猜你喜欢

- 这回试试使用CSS实现抛物线运动效果
- CSS, SVG和canvas分别实现文本文字纹理叠加效果
- 小tip: 使用CSS将图片转换成模糊(毛玻璃)效果
- 图片动态局部毛玻璃模糊效果的实现
- canvas实现iPhoneX炫彩壁纸屏保外加pixi.js流体动效
- 小tip: 使用CSS将图片转换成黑白(灰色、置灰)
- 小tips: 0学习成本实现HTML元素粘滞融合效果
- SVG图标颜色文字般继承与填充
- IE6下png背景不透明问题的综合拓展
- JavaScript实现新浪微博文字放大显示动画效果
- SVG+JS path等值变化实现CSS3兴叹的图形动画

标签： feDisplacementMap, feImage, SVG, 水波, 波纹, 渐变, 滤镜, 高斯模糊

发表评论（目前7条评论）

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

1. 集萤映雪说道：
2018年02月7日 18:11

旭神：
“当scale为0的时候”这句话后面的那两张截图示意图,我跟demo中实际操作以后的效果似乎是不太对应,感觉似乎是反了.我用的是mac上

面的chrome 版本是:63.0.3239.132。要不你看一下

回复

2. **hxf说道：**
2017年12月19日 13:21

chrome canary 65.0.3 例子失效

回复

3. 李阳说道：
2017年12月18日 16:19

我测试一下你写的那个 js，我在页面上不同的地方引用，有的可以，有的失败，失败之后点击调用浏览器直接就崩溃了，我也不知道为什么。

回复

4. 依韵说道：
2017年12月18日 15:05

http://www.cssworld.cn/ 这里的这本书什么时候上架啊？非常期待。

回复

5. 珂珂说道：
2017年12月18日 10:43

膜拜

回复

6. **sky说道：**
2017年12月18日 03:25

张老师，css世界什么时候出版啊

回复

> 张 鑫旭说道：
> 2017年12月18日 21:48
>
> 
>
> maybe就这周
>
> 回复

最新文章

» 常见的CSS图形绘制合集
» 粉丝群第1期CSS小测点评与答疑
» 分享三个纯CSS实现26个英文字母的案例
» 小tips: 纯CSS实现打字动画效果
» CSS/CSS3 box-decoration-break属性简介
» CSS :placeholder-shown伪类实现Material Design占位符交互效果
» 从天猫某活动视频不必要的3次请求说起
» CSS vector-effect与SVG stroke描边缩放
» CSS ::backdrop伪元素是干嘛用的?
» 周知：CSS -webkit-伪元素选择器不再导致整行无效

今日热门

» 常见的CSS图形绘制合集 (190)
» 未来必热：SVG Sprite技术介绍 (119)

» 粉丝群第1期CSS小测点评与答疑 (115)
» HTML5终极备忘大全（图片版+文字版） (93)
» 让所有浏览器支持HTML5 video视频标签 (86)
» Selectivizr-让IE6~8支持CSS3伪类和属性选择器 (82)
» CSS3下的147个颜色名称及对应颜色值 (79)
» 视区相关单位vw, vh..简介以及可实际应用场景 (76)
» 写给自己看的display: flex布局教程 (76)
» 小tips: 纯CSS实现打字动画效果 (76)

今年热议

» 《CSS世界》女主角诚寻靠谱一起奋斗之人 (76)
» 不借助Echarts等图形框架原生JS快速实现折线图效果 (64)
» 看，for..in和for..of在那里吵架！ (60)
» 是时候好好安利下LuLu UI框架了！ (47)
» 原来浏览器原生支持JS Base64编码解码 (35)
» 妙法攻略：渐变虚框及边框滚动动画的纯CSS实现 (33)
» 炫酷H5中序列图片视频化播放的高性能实现 (31)
» CSS scroll-behavior和JS scrollIntoView让页面滚动平滑 (30)
» windows系统下批量删除OS X系统.DS_Store文件 (26)
» 写给自己看的display: flex布局教程 (26)

猜你喜欢

- 这回试试使用CSS实现抛物线运动效果
- CSS, SVG和canvas分别实现文本文字纹理叠加效果
- 小tip: 使用CSS将图片转换成模糊(毛玻璃)效果
- 图片动态局部毛玻璃模糊效果的实现
- canvas实现iPhoneX炫彩壁纸屏保外加pixi.js流体动效
- 小tip: 使用CSS将图片转换成黑白(灰色、置灰)
- 小tips: 0学习成本实现HTML元素粘滞融合效果
- SVG图标颜色文字般继承与填充
- IE6下png背景不透明问题的综合拓展
- JavaScript实现新浪微博文字放大显示动画效果
- SVG+JS path等值变化实现CSS3兴叹的图形动画