**有趣的鼠标悬浮模糊效果**

### 主要是用下面几个CSS属性实现的：

1. background-image
2. -webkit-background-clip
3. -webkit-text-fill-color
4. background-size
5. animation

### 具体实现

#### 绘制渐变背景图

background-image: -webkit-linear-gradient(left, blue, red 25%, blue 50%, red 75%, blue 100%);  
使用CSS3的渐变绘制图像，从左到右。  
需要注意的是颜色是 0到49%的颜色组 = 50%到99%的颜色组，且最后100%的颜色要和开头0的颜色相等  
这是为了能无缝衔接流光效果, 之后有说到

#### 裁剪背景图

-webkit-background-clip: text;  
使用文字作为裁剪区域向外裁剪，此时文字颜色仍覆盖背景图

#### 设置字体颜色

-webkit-text-fill-color: transparent; or color: transparent;  
将字体颜色设置成透明，这样就能将背景图显示出来了

#### 设置背景图长度

background-size: 200% 100%;  
将背景图宽度拉长至两倍，之前设置background-image的两份相同的颜色组，就是为了能在此拉长后只显示一份颜色组，另外超出的半截颜色组用来实现流光效果

#### 开始动画

animation: streamer 5s infinite linear;

@**keyframes** streamer {

0% {

background-position: 0 0;

}

100% {

background-position: -100% 0;

}

}

最后通过动画改变背景图的位置实现流光效果

[我的实现0w0](https://nightcatsama.github.io/ife2017/views/mouseOverhang.html)

### 概述

这是IFE2017的任务，也是我选择的第一个任务，练练手。  
主要实现的是CSS3动画的编写，伪元素的使用，背景图片模糊以及文字流光效果，知识点大概就是这三个。

先看一下实现的效果：  


### 边框从两边散开

这个主要是使用了伪元素::before，::after来实现，在父容器上加上伪元素，里面填写的CSS分别的上下，左右的边框，并定位到相应的位置。  
在这里父容器使用相对定位，伪元素使用绝对定位，先做好位置。

那么怎么实现边框从两边散开？  
在hover的时候，分别吧对应的left 和top值变为0，之前设置了相对父容器的50%。

说了这么多，看代码吧

.wrap .caption .container::before {

content: '';

position: absolute;

left: 0;

top: 50%;

width: 100%;

height: 0;

border: 3px solid ghostwhite;

border-width: 0 3px;

overflow: hidden;

visibility: hidden;

transition: all .8s;

box-sizing: border-box;

}

.wrap .caption .container::after {

content: '';

position: absolute;

left: 50%;

top: 0;

width: 0;

height: 100%;

border: 3px solid ghostwhite;

border-width: 3px 0;

overflow: hidden;

visibility: hidden;

transition: all .8s;

box-sizing: border-box;

}

.wrap:hover .container {

opacity: 1

}

.wrap:hover .container::before {

height: 100%;

visibility: visible;

top: 0;

}

.wrap:hover .container::after {

width: 100%;

visibility: visible;

left: 0;

}

### 文字流光效果

这个在之前并不知道怎么实现，Google了一下，看了张鑫旭大神的博客，也能理解了。  
主要是用背景的属性来实现。  
用到的CSS属性

background-image -webkit-text-fill-color -webkit-background-clip -webkit-background-size

代码：

.wrap .caption .container .ribbon {

*/\*渐变背景，此处为能无缝拼接的渐变 即0~100%\*/*

background-image: -webkit-linear-gradient(left, #3498db, #f47920 10%, #d71345 20%, #f7acbc 30%, #ffd400 40%, #3498db 50%, #f47920 60%, #d71345 70%, #f7acbc 80%, #ffd400 90%, #3498db);

color: transparent;

*/\*文字填充色为透明\*/*

-webkit-text-fill-color: transparent;

*/\*背景剪裁为文字，相当于用背景填充文字 CSS3新属性\*/*

-webkit-background-clip: text;

*/\*背景图片向水平方向扩大一倍，这样background-position才有移动与变化的空间\*/*

-webkit-background-size: 200% 100%;

background-size: 200% 100%;

*/\* 动画 \*/*

-webkit-animation: masked-animation 4s infinite linear;

}

@**keyframes** masked-animation {

0% {

background-position: 0 0;

}

100% {

background-position: -100% 0;

}

}

@-**webkit**-**keyframes** masked-animation {

0% {

background-position: 0 0;

}

100% {

background-position: -100% 0;

}

}

### 总结

多写多总结，多查API。

#### 参考文章

[小tip: CSS3与文字渐变光影流动动画效果实现](http://www.zhangxinxu.com/wordpress/2014/02/css3%E4%B8%8E%E6%96%87%E5%AD%97%E6%B8%90%E5%8F%98%E6%B0%B4%E7%BA%B9%E6%B3%A2%E5%8A%A8%E6%95%88%E6%9E%9C/)  
[MDN CSS3](https://developer.mozilla.org/zh-CN/docs/Web/CSS/CSS3)

看到贴吧里面有同学觉得课程对初学者不是很友好，我就把自己以前学 CSS 动画时候的一篇笔记贴出来吧。。可能对于本题不一定能直接解决问题，但是应该会比较有启发性。由于这个 markdown 不能直接贴 jsfiddle，就劳烦大家一一点开吧。。

原文地址：[CSS 动画学习笔记](https://wuchanzhe.cn/2017/02/08/css-dong-hua-xue-xi-bi-ji/)

一直以来，我对于 CSS 的好感度是要略小于 Javascript 的，它给我的感觉是对网页的作用只有布局上，似乎用途只有老生常谈的 N 栏布局之类，不像 Javascript 负责交互、后端开发负责逻辑。我之前阅读的一本书《精通 CSS》，书虽然是好书，但解决问题的思路停留在 CSS2 时代，涉及的多是网页布局方面，几乎没有对CSS3中动画的介绍。这些看法在我接触过 CSS 动画之后有所改变，甚至开始爱上 CSS 动画了❤。

通过CSS动画，我们可以做出一些有趣的特效。至少有以下几点让我们可以优先考虑使用CSS动画：

* CSS动画的语法相比与 Javascript 动画、JQuery 动画的语法要容易的多；
* CSS动画表现效果较 Javascript 更好。CSS 动画甚至可以在低性能的设备上流畅运行，渲染引擎会使用一些技术使动画尽可能流畅运行，而 Javascript 动画表现通常不佳；
* 由浏览器控制动画序列，浏览器可以优化性能；
* 在一些情况下，我们甚至可以用 CSS 动画来代替图片和 Flash，大大减少了文件大小，减少了服务器压力；
* 支持 Chrome、IE 10、Firefox、Opera 等现代浏览器，随着 Windows XP 份额的减少，我们已经几乎不用考虑兼容性问题；即便是不支持该属性，动画也只是突变发生，不会对显示结果带来影响（特别为了动画效果而使用 CSS 动画的除外）。

## Transition

### 简介

一般来说，当 CSS 有元素的属性发生变化时，这个变化会立刻体现在元素上，也就是“突变”。transition 可以控制属性变化的动画的速率，让变化过程从“突变”改变为持续一段时间的过程。

例如，在不加 transition 时，我们把鼠标移动到 wrapper上时是“突变”的：

<https://jsfiddle.net/wmc54321/qfyf2ghs/embedded/result,css,html/>

当我们添加了 transition 属性，为动画添加一个时间，变化就圆润多了：

<https://jsfiddle.net/wmc54321/a8k4no9p/embedded/result,css,html/>

我们甚至可以为wrapper的边框和阴影添加动画：

<https://jsfiddle.net/wmc54321/r49pn31f/embedded/result,css,html/>

### 指定属性

当然，仅仅这样是不够用的。我们还可以指定动画只在指定的属性上发生。例如，我们给时间加上属性width、height，那么动画就只在width和height上发生，其他的属性仍然是“突变”：

<https://jsfiddle.net/wmc54321/m2L6ky0u/embedded/result,css,html/>

### 时间

我们还可以给动画加上延时。我们在原有的时间后面加上我们希望它们在被触发后开始运行的时间，就可以看到动画依次发生在height、width、border、border-radius、box-shadow上：

<https://jsfiddle.net/wmc54321/rbvk5avs/embedded/result,css,html/>

我们发现，当我们把鼠标移走的时候，动画仍然会按照我们所设定顺序的发生，这样看起来似乎比较奇怪。我们可以把鼠标移入的动画写在 .wrapper:hover 里面，由于其在文档中位置靠后，因此当 :hover 发生时该属性发生作用；我们把鼠标移出的动画写在 .wrapper 里面，这样当鼠标移出时 :hover 中的属性不起作用，该属性起作用。

<https://jsfiddle.net/wmc54321/buch8a1z/embedded/result,css,html/>

### 调速函数

我们可以为transition设定速度变化函数。默认的情况是ease，是逐渐放慢。除此之外，还有linear（线性）、ease-in（加速）、ease-out（减速）等。例如：

<https://jsfiddle.net/wmc54321/h3muzmhy/embedded/result,css,html/>

我们发现 wrapper 的宽度变化变为了匀速改变。

### 特殊情况

transition 能够应用在我们所能见到的大多数属性上，具体可以查看[CSS动画属性表](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties)。不过，在实际应用中，我们常常会发现有一种情况下transition失灵了：

<https://jsfiddle.net/wmc54321/bthg9rb8/embedded/result,css,html/>

我们明明给 wrapper 加了 transition，元素却没有像想象中那样“渐变”，而是“突变”，transition 失灵了！

查阅资料，我们发现，当我们为 width、height 属性设置 transition 时，如果起始状态（或结束状态）是 auto，那么 transition 是无效的。解决方法是，为元素设置一个合适的 max-height：

<https://jsfiddle.net/wmc54321/t2e6bbux/embedded/result,css,html/>

### Transition的各个子属性

最后，我们再来仔细学习一下各个属性的用法。

* transition-property – 触发动画的属性。

允许值：none | all | <属性名>

初始值：all

none表示无属性。all表示所有可以触发动画的属性。或者，可以指定属性触发动画效果。

* transition-duration – 动画的持续时间。

允许值：<时间>

初始值：0

* 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)。

允许值: ease | linear | ease-in | ease-out | ease-in-out | cubic-bezier(x1, y1, x2, y2)

初始值: ease

linear – 线性函数，返回值一个输入值一样的结果（匀速）。

ease – 减缓函数, 是缺省值, 等同于 cubic-bezier(0.25, 0.1, 0.25, 1.0)（逐渐放慢）。

ease-in – 等同于 cubic-bezier(0.42, 0, 1.0, 1.0)（加速）。

ease-out – 等同于 cubic-bezier(0, 0, 0.58, 1.0)（减速）。

ease-in-out – 等同于 cubic-bezier(0.42, 0, 0.58, 1.0)

cubic-bezier - 自定义[贝塞尔曲线](http://cubic-bezier.com/)。

* transition – 三种属性的合体简写。

### 局限

transition 的动画效果已经让我们感到惊叹。但是，transition 仍然有相当的局限性。

* transition 是有触发条件的（有属性发生改变）；
* transition 只和开始时属性状态和结束时属性状态有关，不能定义中间状态；
* 动作是一次性的，动画执行一次就没有了。

有没有更灵活的动画呢？

## Animation

### 简介

使用 animation，需要用到 @keyframes 规则。使用 @keyframes 规则时，需要把它绑定到选择器上。绑定的方式是，至少要定义动画的持续时间和动画的名称。

例如，如下面的例子，我们定义一个动画时间2s，定义了一个动画jumping，并让该动画在 :hover 时发生。我们用 @keyframes 定义动画在各个时间节点的属性（关键帧）。

<https://jsfiddle.net/wmc54321/5mg0zgc2/embedded/result,css,html/>

加上 infinite 关键字可以让动画无限循环（或者是动画播放次数）：

<https://jsfiddle.net/wmc54321/5o3xtyo5/embedded/result,css,html/>

### Animation的各个子属性

我们再来介绍一下Animation的各个属性，下文中我们就不详细介绍了。

* animation-delay - 设置延时，也即从元素加载完成之后到动画序列开始执行的时间。
* animation-direction - 设置动画在每次运行完后是反向运行还是重新回到开始位置重复运行。

允许值：normal, alternate, reverse, alternate-reverse

默认值：normal

normal：动画循环播放时，从结束状态回到起始状态

浏览器对其他值的支持情况不佳，应该慎用。

* animation-duration - 设置动画一个周期的时长。
* animation-iteration-count - 设置动画重复次数（可以指定infinite无限次重复动画），默认为一次
* animation-name 指定动画名称，即由 @keyframes 描述的关键帧名称。
* animation-play-state - 允许暂停和恢复动画。

允许值：paused, running

running：动画停止后停止动画，动画开始时重新开始

paused：当动画突然停止时，保持暂停状态，当动画开始时继续播放动画

* animation-timing-function - 设置动画速度， 即通过建立加速度曲线，设置动画在关键帧之间是如何变化。

允许值：ease, ease-out, ease-in, ease-in-out, linear, cubic-bezier(x1, y1, x2, y2)

* animation-fill-mode - 指定动画执行前后如何为目标元素应用样式。

允许值：forwards, backwards, both, none

默认值：none

none：回到动画还未开始前的状态

backwards：动画回到第一帧的状态

forwards：动画停留在结束时的状态

both: 根据 animation-direction 轮流应用 forwards 和 backwards 规则。

### 分层动画——曲线运动

我们使用 transition 和 animation 可以非常容易地做出元素从 A 到 B 的直线运动，当然运动轨迹也是直线。如果我们想要做出曲线运动的效果，似乎就无从下手了。即便是上文中提到的贝塞尔曲线也只是负责时间曲线，而不是空间曲线。

我们回忆一下高中物理中的抛物线：

* x = v t
* y = ( g t ^ 2 ) / 2

将单维度的运动转化为两个维度的运动，问题就迎刃而解了。

我们可以使用 CSS 中的一个有趣的存在——伪元素。我们在选择器中，加上 :before、:after，二者就会分别在元素内部的最前面和最后面出现。例如：

<https://jsfiddle.net/wmc54321/p5s5zk8g/embedded/result,css,html/>

因此，我们可以以元素容器作为 X 轴，容器中的 :after 伪元素作为 Y 轴，实现曲线动画。

如下面的例子，我们让左上角的白色球向右下方运动时，在 X 方向加速、Y 方向减速，就会创造出一个加速度方向斜向右上的曲线；向左上方运动时，在 X 负方向加速、Y 负方向减速，就创造出了加速度方向斜向左下的曲线。

<https://jsfiddle.net/wmc54321/ew7e6t48/embedded/result,css,html/>

### Animation实战

animation 显然要比transition要灵活得多。我们就来着手写一个加载动画吧。下面的例子展示了我写的一个加载动画，目前已经用在了我的[相册展示页面](http://minchen.wang/)上。

<https://jsfiddle.net/wmc54321/907rxt4v/embedded/result,css,html/>

思路非常简单。我们为四个颜色的球分别做一个包含 [向外发射, 向内恢复位置, 在原位置暂停一段时间] 的动画，再对四个球的父元素做一个旋转的动画 [加速转动120°, 加速转动120°, 加速转动120°]。

我们再来看一个 CSS 动画播放按钮。这个按钮已经应用到了我做的[HTML5 播放器 NyaPlayer](https://nyaplayer.wuchanzhe.cn/)中。

<https://jsfiddle.net/wmc54321/o41um2r2/embedded/result,css,html/>

这个按钮实现的原理要点如下：用 CSS 的 border 绘制伪元素 :before、:after。当显示为播放时，前者绘制为三角形，后者为宽度为0的长方形；当显示为暂停时，前者绘制为正方形，后者绘制为某宽度的白色长方形遮挡住正方形的一部分，这样看起来就像是暂停键了。

看了这么多，你是不是也想亲手试一下了呢？

## 参考文献

阮一峰 - [CSS动画简介](http://www.ruanyifeng.com/blog/2014/02/css_transition_and_animation.html)

Alon's Blog - [CSS分层动画可以让元素沿弧形路径运动](http://jinlong.github.io/2016/01/14/moving-along-a-curved-path-in-css-with-layered-animation/)

MDN - [使用 CSS transitions](https://developer.mozilla.org/zh-CN/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions)

MDN - [使用 CSS 动画](https://developer.mozilla.org/zh-CN/docs/Web/CSS/CSS_Animations/Using_CSS_animations)

赞6

## 动态数据绑定（二）

动态数据绑定是MVVM框架中最基础的的一个功能，简单描述就是：将数据和视图进行绑定，当数据发生改变时，视图随之改变，更深层次一点，数据绑定包括单向数据绑定和双向数据绑定。

本文从数据绑定中的问题出发，一步一步的来实现这个功能。

本文的所有的源代码地址： [点击此处查看源代码](https://github.com/liujianhuanzz/ife_2016_spring)

## 问题一

给定任意一个对象，如何监听其属性的读取与变化？也就是说，如何知道程序访问了对象的哪个属性，又改变了哪个属性？

举个例子：

let app = new Observer({

name: 'liujianhuan',

company: 'Qihoo 360',

**address**: 'Chaoyang, Beijing'

})

*//要实现的结果如下*

app.data.name *//你访问了name*

app.data.company *//你访问了company*

app.data.address = 'Beijing' *//你设置了address， 新的值为 Beijing*

实现这样的一个Observer并不难，在此我们暂且不考虑数组的情况，只针对传入的参数为对象。如果对ES6和ES5都熟悉的话，可以立刻想到针对上述场景，可以有两种的实现方式：

1. **采用ES6中的proxy，对目标对象的属性进行拦截处理**
2. **采用ES5中的defineProperty，为目标对象的属性添加setter和getter**

接下来首先采用ES6中的proxy方法实现上述场景，首先从阮一峰老师的《es6入门标准中》摘录：

Proxy可以理解成在目标对象前架设一“拦截”层，外界对该对象的访问都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。Proxy这个词的原意是代理，用在这里表示由它来“代理”某些操作，可以称为“代理器”。

上边的话读完后应该和没读一样，放出来也只是用来装一下的。下边直接用简单的例子来说明：

ES6原生提供Proxy构造函数，用于生成Proxy实例。

var proxy = **new** Proxy(**target**,**handler**);

var proxy = **new** Proxy(**target**,**handler**);

Proxy对象的所有用法都是上面的形式，不同的只是handler参数的写法。其中new Proxy()表示生成的一个Proxy实例，target参数表示所要拦截的目标对象，handler参数也是一个对象，用来定制拦截行为。

下面来一个拦截读取属性行为的例子：

**var** proxy = **new** Proxy({},{

**get**:**function**(target,property){

**return** 35;

}

});

proxy.time;*//35*

proxy.name;*//35*

proxy.title;*//35*

上面代码中，作为构造函数，Proxy接受两个参数。第一个参数是所要代理的目标对象（上例是一个空对象），即如果没有Proxy的介入，操作原来要访问的就是这个对象；第二个参数是一个配置对象，对于每一个被代理的操作，需要提供一个对应的处理函数，该函数将拦截对应的操作。比如，上面代码中，配置对象有一个get方法，用来拦截对目标对象属性的访问请求。get方法的两个参数分别是目标对象和所要访问的属性。可以看到，由于拦截函数总是返回35，所以访问任何属性都得到35。

##### 听话分割线出来了，以上内容摘自《ES6标准入门（第二版）》，特此声明！

看了上述的代码之后，我想也应该不用再太多的介绍了，直接上针对问题一的代码：

**function** **Observer**(data){

**return** **new** Proxy(data, {

get: **function**(target, key){

**if**(key **in** target){

console.log('你访问了' + key);

**return** target[key];

}**else**{

**throw** **new** Error('key does not exist')

}

},

set: **function**(target, key, newVal){

console.log('你设置了' + key);

console.log('新的' + key + '=' + newVal);

target[key] = newVal;

}

})

}

**let** app = **new** Observer({

name: 'liujianhuan',

company: 'Qihoo 360',

address: 'Chaoyang, Beijing'

})

测试结果如下图：



如上图结果所示，上述代码完美的实现了问题一中所提到的监听对象属性变化，但是深入思考就会发现，上述代码还是有问题的，因此，引出来问题二。

## 问题二

如果传入的参数对象是一个“比较深”的对象（也就是其属性值也可能是对象），那该怎么办？

举个例子：

let app = new Observer({

basicInfo: {

name: 'liujianhuan',

age: 25

},

company: 'Qihoo 360',

**address**: 'Chaoyang, Beijing'

})

*//要实现的结果如下*

app.data.basicInfo.name *//你访问了basicInfo，你访问了name*

首先利用问题一中的代码进行测试：



从结果可以看到并不能解决问题，到这里也许有人觉得只要在代码中加上这样的一段代码即可：

**for**(let key **in** data){

**if**(data.hasOwnProperty(key) && typeof data[key] === 'object'){

new Observer(data[key]);

}

}

事实上是这种方式是无效的，读者可自行测试。究其原因，ES6中的proxy方式是通过Proxy构造函数来new一个实例，此实例代理拦截目标对象的操作，所以对于深层递归new出来的子对象实例我们是无法操作的，所以这种方法无效。

一步一步写的现在是不是觉得人生好无趣，好不容易写了这么多却发现行不通啊。这时候先上一碗热鸡汤，**人生的每一步都是我们应该走的，因为它会给我们不同的经历，让我们更坚韧、更强大**。 此路不通，那就只能再回首走开篇提到的第二条路了，不过这时候也应该上一碗毒鸡汤，所谓，**码农之路就是，山重水复疑无路，柳暗花明又一坑**。

第二种方法是采用ES5中的defineProperty，为目标对象的属性添加setter和getter。关于defineProperty的基本知识这里不再赘述，有不清楚的地方可以自行翻阅权威书籍，比如红宝书。直接上代码，下边的代码涵盖了问题一和二。

**function** **Observer** (data) {

*//暂不考虑数组*

**this**.data = data;

**this**.makeObserver(data);

}

Observer.prototype.setterAndGetter = **function** (key, val) {

*//此为问题一的要点*

Object.defineProperty(**this**.data, key, {

enumerable: true,

configurable: true,

get: **function**(){

console.log('你访问了' + key);

**return** val;

},

set: **function**(newVal){

console.log('你设置了' + key);

console.log('新的' + key + '=' + newVal);

val = newVal;

}

})

}

Observer.prototype.makeObserver = **function** (obj) {

**let** val;

*//此为问题二的要点*

**for**(**let** key **in** obj){

**if**(obj.hasOwnProperty(key)){

val = obj[key];

*//深度遍历*

**if**(**typeof** val === 'object'){

**new** Observer(val);

}

}

**this**.setterAndGetter(key, val);

}

}

*//测试*

**let** app = **new** Observer({

basicInfo: {

name: 'liujianhuan',

age: 25

},

company: 'Qihoo 360',

address: 'Chaoyang, Beijing'

})

测试结果如下图：



看到这样的结果是不是很开心呢，同时解决了问题一和问题二，perfect。如果你这样想了，那就得回想一下毒鸡汤了，**生活中不是缺少坑，是缺少发现坑的眼睛，直到被你柳暗花明之后踩到。**。请继续看问题三。

## 问题三

如果设置新的值是一个对象的话，新设置的对象的属性是否能继续响应getter和setter呢？

举个例子：

let app = new Observer({

basicInfo: {

name: 'liujianhuan',

age: 25

},

company: 'Qihoo 360',

**address**: 'Chaoyang, Beijing'

})

*//要实现的结果如下*

app.data.basicInfo = {like: 'NBA'}*//你设置了basicInfo，新的basicInfo为{like: 'NBA'}*

app.data.basicInfo.like *//你访问了basicInfo，你访问了like*

采用问题二中的代码进行测试：



看到了吧，如果设置新的值是一个对象的话，新设置的对象的属性不能继续响应getter和setter。不过代码写到这里，这个问题应该是非常容易的就可以解决了，那就是直接在setter中添加如下代码：

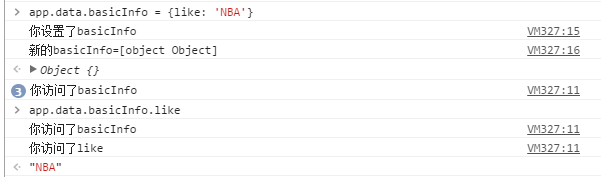
*//如果newval是对象的话*

**if**(typeof **newVal** === 'object'){

**new** **Observer**(val);

}

测试结果如下：



至此，我们已经较为完整的了实现了针对对象变化的数据监听，由于数组的操作方法比较多，所以针对数组的变化监听待后续完善，接下来我们针对上述代码继续增强完善。

## 完善点一

考虑传递回调函数。在实际应用中，当特定数据发生改变的时候，我们是希望做一些特定的事情，而不是每一次只能打印出来一些信息，所以，我们需要支持传入回调函数的功能。

举个例子：

**let** app = **new** Observer({

name: 'liujianhuan',

age: 25,

company: 'Qihoo 360',

address: 'Chaoyang, Beijing'

})

app.$watch('age', **function**(age){

console.log(`我的年龄变了，现在是：${age}岁了`)；

})

app.data.basicInfo.age = 20;*//输出：'我的年龄变了，现在已经是20岁了'*

针对上述场景，我们需要实现$watch这个API，每当年龄发生改变的时候触发相应的回调函数。这个API的实现可以很有多种方式，在此我们采用事件的方式来实现，通俗的讲就是实现一个通用的事件模型，每次$watch一个属性相当于注册了一个监听事件，当属性发生改变的则触发对应的事件，这样做的优势是可以为同一个属性通过事件模型来注册多个回调函数。

下边是一个不完整的简易事件模型：

*//实现一个事件*

**function** **Event**(){

**this**.events = {};

}

Event.prototype.on = **function**(attr, callback){

**if**(**this**.events[attr]){

**this**.events[attr].push(callback);

}**else**{

**this**.events[attr] = [callback];

}

}

Event.prototype.off = **function**(attr){

**for**(let key **in** **this**.events){

**if**(**this**.events.hasOwnProperty(key) && key === attr){

**delete** **this**.events[key];

}

}

}

Event.prototype.emit = **function**(attr, ...arg){

**this**.events[attr] && **this**.events[attr].forEach(**function**(item){

item(...arg);

})

}

有了上述事件模型后，每次new一个Observer的实例时，就new一个Event实例出来用来管理Observer实例中的所有事件；然后通过$watch API来为Observer实例注册属性的监听事件，每次当属性改变的触发相应的事件队列。

**function** **Observer** (data) {

*//暂不考虑数组*

**this**.data = data;

**this**.makeObserver(data);

**this**.eventsBus = **new** Event();

}

Observer.prototype.setterAndGetter = **function** (key, val) {

**let** \_this = **this**;

Object.defineProperty(**this**.data, key, {

enumerable: true,

configurable: true,

get: **function**(){

console.log('你访问了' + key);

**return** val;

},

set: **function**(newVal){

console.log('你设置了' + key);

console.log('新的' + key + '=' + newVal);

*//触发$watch函数*

\_this.eventsBus.emit(key, val, newVal);

val = newVal;

*//如果newval是对象的话*

**if**(**typeof** newVal === 'object'){

**new** Observer(val);

}

}

})

}

Observer.prototype.makeObserver = **function** (obj) {

**let** val;

**for**(**let** key **in** obj){

**if**(obj.hasOwnProperty(key)){

val = obj[key];

*//深度遍历*

**if**(**typeof** val === 'object'){

**new** Observer(val);

}

}

**this**.setterAndGetter(key, val);

}

}

Observer.prototype.$watch = **function**(attr, callback){

**this**.eventsBus.on(attr, callback);

}

**let** app = **new** Observer({

name: 'liujianhuan',

age: 25,

company: 'Qihoo 360',

address: 'Chaoyang, Beijing'

})

app.$watch('age', **function**(oldVal, newVal){

console.log(`我的年龄变了，原来是: ${oldVal}岁，现在是：${newVal}岁了`)

})

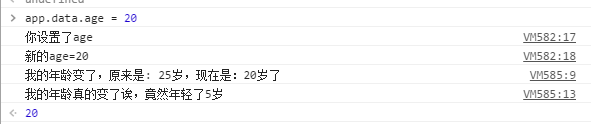
app.$watch('age', **function**(oldVal, newVal){

console.log(`我的年龄真的变了诶，竟然年轻了${oldVal - newVal}岁`)

})

app.data.basicInfo.age = 20;

测试结果如下：



测试结果显示上述代码触发了所注册的两个回调，但是上述代码也还是有着问题，比如目前只可注册监听对象的第一层的属性，对于对象的深层属性并不能有效监听，比如：

**let** app = **new** Observer({

basicInfo: {

name: 'liujianhuan',

age: 25

},

company: 'Qihoo 360',

address: 'Chaoyang, Beijing'

})

app.$watch('age', **function**(age){

console.log(`我的年龄变了，现在是：${age}岁了`)；

})

app.data.basicInfo.age = 20;

这段代码中的回调并不会触发，这个问题留下来在后续中完善补充。

总结一下本文中针对“动态数据绑定”还未解决掉的问题：

1. **当传入的参数为数组时，如何监听数组对象的变化**
2. **深层对象属性的事件回调监听，或者描述为：对象的深层属性值发生变化后如何向上传递到顶层**
3. **动态数据与视图的绑定，如何绑定，当数据变化后如何触发视图的自动刷新。**

另外附上两个最近使用Vue实现的 [Vue在SKU组合查询中应用](http://www.hhspace.cn/?p=374) 和 [基于Vue的后台管理模板](http://vueadmin.hhspace.cn/)

赞44

## 动态数据绑定（三）

# 动态数据绑定（三）

## 事件传播机制

没参考！

### 问题分析：

目前已经实现事件机制的是这样的：

注册$watch --> Event.on --> this.events中添加键值对(**注意每个Observer一个Event**)

触发(如修改属性) --> Observer.convert(setter) --> Event.emit --> this.events匹配执行

所以其实应该看得出来了，每一层的Event现在都是独立的，问题本质上就是要让他们能通信(我一开始都没理清...菜的一笔)。

### 先总结

到现在我大概知道一个可行的方法是(比如里的变量参考后面的后面的例子)：

1. $watch要能够安装参数路径(如"stu.name.firstName")去挂在该挂的地方，目前只会和data平级(控制台可以看)。

比如说我监听app.data.stu.name，那么就在name那一层(stu里面)的Event里添加stu.name: [func...]。

1. emit的时候触发的事件名与键的路径保持一致，而在冒泡的过程中，逐层剥去下层的属性名，从而使父层的监听也可以匹配到。

比如说我修改了app.data.stu.name.firstName，那么就会在firstName同层Event触发事件"stu.name.firstName"，甭管有没有相应的函数执行，接下来会开始冒泡向上层传递，并将事件"stu.name.firstName"重新解析为"stu.name"，而这是毫无疑问的，对象的存储在这点上类似树(脑补DOM...)，然后在stu.name同层Event再次触发事件"stu.name"，直到到达顶层。这样一来，就实现了事件类似树的冒泡。

而想要取到父层，这里采用的方法是给Observer构造函数里加入$parent，通过这个指针就可以访问的上层的Event了。。。这个方法可能不是很好，但目前我是这么做的。

### 目前做到什么程度

2做到了，但是没有调用那个重新解析的函数，因为1没实现调用了白搭。现在可以修改内部变量后我挂在最外层的相应事件会执行回调。1也要继续做而且肯定需要优化，但看到某个大神说过早优化是万恶之源，我决定先跑路哈哈哈！

### 思路记录

任务描述中要求的效果如下:

**let** app2 = **new** Observer({

name: {

firstName: 'shaofeng',

lastName: 'liang'

},

age: 25

});

app2.$watch('name', **function** (newName) {

console.log('我的姓名发生了变化，可能是姓氏变了，也可能是名字变了。')

});

app2.data.name.firstName = 'hahaha';

*// 输出：我的姓名发生了变化，可能是姓氏变了，也可能是名字变了。*

app2.data.name.lastName = 'blablabla';

*// 输出：我的姓名发生了变化，可能是姓氏变了，也可能是名字变了。*

也就是说在events里添加键"name"及其回调，这样我修改相应属性时就会emit相应的事件，完成事件响应。可问题就在于，"name"的值是对象，我修改firstName等触发的事件名是"firstName"啊，原有的事件管理机制是检测不到注册在"name"上的事件的。而这里需要做的就是让事件"firstName"能够被挂在"name"上的监听检测到。

**举个栗子**

如果data对象是这样的：

data: {

stu: {

name: {

firstName: 'Zhen',

lastName: 'Ge'

},

age: 24

},

...

}

我在name上监听一个:

**app**.$watch("stu.name", callback);

这样在this.events里是：

**this**.events = {

"stu.name": [...]

}

**注意，这个"stu.name"是挂在最外层的(和data同级)！写到这里我还没实现按路径去挂...**

到这里都不需要修改什么。在触发时，会调用到convert内的set，那么这个firstName怎么能知道是"stu.name.firstName"呢？

我的解决方式就是让Observer多一个参数，借助它来一层层传递path，深处也会得到拼接到的path。这样emit的时候，对应的事件名就会被修改成path。如我修改如下时：

app2.data.stu.name.firstName="zhen"

然后就会在setter里拿到path。具体可以看代码...

**我刚注意到注册位置和想象的不一样！emit的事件虽然名字和注册的可以一样，但是emit时找不到注册在外面的事件啊！简而言之深层的事件出不来啊！去他妈的**

如果想让事件传出来，要么emit时有办法向外扩散，要么能把注册的事件注册到指定位置上？

把事件放到原型上共享肯定可以了……。然看到有这么实现的。

给所有子属性都绑定也太变态了。不要啊。。。

在Observer里写了$parent，让它指向父Observer对象。

终于能拿到event了。现在就想做两件事：

1. 能指哪打哪的$watch，只在指定属性那一层的Event上加on；
2. 事件冒泡啊尼玛

终于冒泡做好了……$watch终于执行了尼玛

其实还得改很多东西：1. 冒泡的过程中需要一层层解构 不然没法触发上面的东西  
也就是stu.name.firstName要变成stu.name才行 不然传上来也监听不到，因为监听的是stu.name啊

赞2

# 作品介绍

我封装了一个module（fox-listener），其中包括了事件的委托监听 + 深浅绑定

顺便复习一下单元测试

Travis building pass & 代码覆盖率 >>>

fox-listener@1.0.2 test D:githubprogramfox-listener

istanbul cover node\_modules/mocha/bin/\_mocha test/test.js

测试 listener Pub/Sub API 能否正常使用  
√ sub 后可以执行 unsub 进行取消事件的绑定！  
√ pub 后 可以执行 sub 绑定的回调函数执行  
√ 可在一个监听类目中，添加不同的匿名空间  
√ 当执行一个监听类目后, 所有订阅命名类目都会执行 并且能够获得相同的传入的参数  
√ 发布一个只能执行一次的订阅函数  
√ 实现浅绑定  
√ 实现深绑定

7 passing (16ms)

=============================== Coverage summary ===============================  
Statements : 84.27% ( 75/89 )  
Branches : 72.92% ( 35/48 )  
Functions : 100% ( 15/15 )  
Lines : 83.91% ( 73/87 )

# 体验

npm **install** fox-listener *--save*

或者

[fox-listener](https://github.com/NumerHero/fox-listener)

# 关于实现

其他人的笔记写的很好呀，我就不多说什么了

简单写一下我的解决方案：  
对于path 的处理：

**if** (sclass.length > 1) {

sclass.reduce((b,n) => {

**if** (b.hasOwnProperty(n)) {

$p = b;

$c = n;

**return** b[n];

} **else** {

**throw** Error('Doesnt find this property! Please cleck this property name was right!');

}

}, **self**);

} **else** {

$p = **self**;

$c = sclass[0];

}

我使用 reduce 的方式迭代到指定的目录，输出 $p: 指定目录的父节点 $c：指定目录名

然后绑定getter 和 setter

Object.defineProperty(parent, key, {

**get**: **function** **bindGetter**() {

**return** getter ? getter.call(**this**, $value) : $value;

},

**set**: **function** **bindSetter**(param) {

$value = callback ? callback.call(**this**, param) : param;

}

})

深绑定的话，只要找到到指定目录后，tree 该目录下的所有孩子，然后绑getter & setter就行了

赞2

## 自定义checkbox， radio样式

### 一. label标签

#### 1. 概念：

HTML <label>元素表示用户界面中项目的标题。  
它通常关联一个控件，或者是将控件放置在label元素内，或者是用作其属性。这样的控制称作label元素的labeled control

#### 2. 用法：

* 用法1：
* <**label**>Click me <**input** **type**="text" id="User" name="Name" /></**label**>
* 用法2：
* <**label** **for**="User">Click me</**label**>
* <**input** **type**="text" id="User" name="Name" />

### 二. background-position属性

#### 1. 概念：

background-position属性用来指定背景图片的初始位置；这个初始位置相对于background-origin定义的背景位置图层来说的

#### 2. 用法:

background-position:[position-x]，[position-y]

如果只有一个值被指定，则这个值就会默认设置背景图片位置中的水平方向，与此同时垂直方向的默认值被设置成50%。在关键词、百分比和数值混合指定的情况下，关键词 left 和 right 只能被指定水平方向的位置（第一个值）同时 top 和 bottom 只能被指定指定为垂直方向的值（第二个值）。同样，负数值是允许指定的。

### 三. 伪类和伪元素

#### 1.概念：

伪类：存在的意义是为了通过选择器找到那些不存在于DOM树中的信息以及不能被常规CSS选择器获取到的信息。  
伪类由一个冒号:开头，冒号后面是伪类的名称和包含在圆括号中的可选参数。  
常用伪类:

:active 向被激活的元素添加样式。

:focus 向拥有键盘输入焦点的元素添加样式。

:hover 当鼠标悬浮在元素上方时，向元素添加样式。

:link 向未被访问的链接添加样式。

:visited 向已被访问的链接添加样式。

:first-child 向元素的第一个子元素添加样式。

:checked 向选中的控件元素添加样式

伪元素:伪元素在DOM树中创建了一些抽象元素，这些抽象元素是不存在于文档语言里的（可以理解为html源码）；  
注意:css3为了区分伪类和伪元素，规定伪类前面有一个冒号，伪元素前面有两个冒号  
常用伪元素

::**before** 作为作用元素的第一个子节点插入dom中

::**after** 作为作用元素的最后一个子节点插入dom中

#### 2.区别与联系

* 联系：都是通过选择器为元素添加样式
* 区别:伪元素会创建一个元素，但不是真正的Html元素，伪类相当于为一个元素创建一个class样式

#### 四、自定义控件的原理

隐藏原生input标签，使用label，并且给之加样式来实现自定义控件  
html：

**<!DOCTYPE html>**

<html lang="en">

<head>

<meta charset="UTF-8">

<title>自定义checkbox,radio样式</title>

<link rel="stylesheet" href="index.css">

</head>

<body>

<div id='wrap'>

<h2>伪元素实现：</h2>

<div class="checkbox">

<input type="checkbox" name="" id="checkBox" class="checkBox">

<label for="checkBox"></label>

</div>

<div class="radio">

<input type="radio" name="sex" id="radioBox" class="radios">

<label for="radioBox"></label>

<input type="radio" name="sex" id="radioBox1" class="radios">

<label for="radioBox1"></label>

</div>

<h2>雪碧图实现：</h2>

<div class="checkbox">

<input type="checkbox" name="" id="checkBox\_sprit" class="checkBox\_sprit">

<label for="checkBox\_sprit"></label>

</div>

<div class="radio">

<input type="radio" name="sex1" id="radioBox\_sprit" class=" radioBox\_sprit">

<label for="radioBox\_sprit"></label>

<input type="radio" name="sex1" id="radioBox1\_sprit" class=" radioBox\_sprit">

<label for="radioBox1\_sprit"></label>

</div>

</div>

</body>

</html>

* 伪元素方法实现  
  使用:checked伪类和::after伪元素,当控件选中时添加伪元素到label标签中：

css:

.checkbox{

margin-bottom: 30px;

font-weight: 700;

}

.checkBox+**label**,.radios+**label**{

width:16px;

height: 16px;

border:1px solid #d9d9d9;

display: -webkit-flex;

display:flex;

}

.checkBox:checked+label,.radios:checked+label{

border:1px solid #d73d32;

font-weight: 700;

}

.checkBox:checked+label::after,.radios:checked+label::after{

margin: auto;

content: '2714';

color: red;

width:8px;

height:8px;

line-height: 8px;

text-align: center;

font-size: 12px;

}

**input**{

margin:0;

display: none;

}

*/\* radio \*/*

.radio{

display: -webkit-flex;

display: flex;

width: 50px;

-webkit-justify-content: space-between;

justify-content: space-between;

}

.radios+**label**{

border-radius: 50%;

}

.radios:checked+label::after{

border-radius: 50%;

width:8px;

height:8px;

line-height: 8px;

content: ' ';

background: #d73d32;

}

[效果展示](https://elva2596.github.io/ife-tasks/task-1/index.html)  
[源码地址](https://github.com/elva2596/elva2596.github.io/tree/master/ife-tasks/task-1)

* 雪碧图实现方法:  
  给label加一个背景图片，并通过设置选中状态时的background-position属性  
  css:

.checkBox\_sprit+**label**,.radioBox\_sprit+**label**{

width: 16px;

height:16px;

border:none;

display: inline-block;

background:url('bg.png') no-repeat;

}

.checkBox\_sprit+**label**{

background-position:-25px -32px;

}

.radioBox\_sprit+**label**{

background-position: -24px -10px;

}

.checkBox\_sprit:checked+label{

background-position: -60px -32px;

}

.radioBox\_sprit:checked+label{

background-position: -59px -10px;

}

[效果展示](http://ife.baidu.com/note/detail/id/%5b%E6%95%88%E6%9E%9C%E5%B1%95%E7%A4%BA%5d(https:/elva2596.github.io/ife-tasks/task-1/index.html)  
[源码地址](https://github.com/elva2596/elva2596.github.io/tree/master/ife-tasks/task-1)

### 五. css雪碧图和css伪元素两种方法的比较

* 使用伪元素可以不用图片，这样就减少了http的收发次数，提高了网络性能。
* 使用css雪碧图，可以使还原度更加精确，代码量减少。

# 一、label标签

HTML5中的label标签为 input 元素定义标签（label），<label> 标签的 for 属性应该等于相关元素的 id 元素，以便将它们捆绑起来。  
例如：

<**input** **type**="checkbox" id="checkBox">

<**label** **for**="checkBox"></**label**>

注意：  
"for" 属性可把 label 绑定到另外一个元素。请把 "for" 属性的值设置为相关元素的 id 属性的值。

# 二、“对钩✔️” CSS3制作

首先制作一个长方形

.checkboxWrap **label**::after {

content: '';

position: absolute;

width: 9px;

height: 5px;

top: 3px;

left: 3px;

}

然后设置border值，去除两个边框的border，并进行旋转就可以得到对钩✔️

.checkboxWrap **input**[type=checkbox]:checked + **label**::after {

border: 2px solid #d73d32;

border-top: none;

border-right: none;

-webkit-transform: rotate(-45deg);

-moz-transform: rotate(-45deg);

-o-transform: rotate(-45deg);

-ms-transform: rotate(-45deg);

transform: rotate(-45deg);

}

注意：  
1.区分伪类和伪元素  
伪类：存在的意义是为了通过选择器找到那些不存在于DOM树中的信息以及不能被常规CSS选择器获取到的信息。  
伪类由一个冒号:开头，冒号后面是伪类的名称和包含在圆括号中的可选参数。  
常用伪类:

:active 向被激活的元素添加样式。

:focus 向拥有键盘输入焦点的元素添加样式。

:hover 当鼠标悬浮在元素上方时，向元素添加样式。

:link 向未被访问的链接添加样式。

:visited 向已被访问的链接添加样式。

:first-child 向元素的第一个子元素添加样式。

:checked 向选中的控件元素添加样式

伪元素:伪元素在DOM树中创建了一些抽象元素，这些抽象元素是不存在于文档语言里的（可以理解为html源码）；  
注意: css3为了区分伪类和伪元素，规定伪类前面有一个冒号，伪元素前面有两个冒号  
常用伪元素:

::**before** 为作用元素的第一个子节点插入dom中

::**after** 为作用元素的最后一个子节点插入dom中

* 同：都是通过选择器为元素添加样式
* 异：伪元素会创建一个元素，但不是真正的Html元素，伪类相当于为一个元素创建一个class样式

2."+" 选择相邻兄弟  
例如，如果要增加紧接在 h1 元素后出现的段落的上边距，可以这样写：

**h1** + **p** {margin-top:50px;}

# 三、最后

[效果展示](https://wj704.github.io/ife2017/task1/checkbox_radio.html)  
[源代码](https://github.com/wj704/wj704.github.io/tree/master/ife2017/task1)

# 自定义checkbox， radio样式

从前还没有尝试过自定义元素的样式, 趁着这一次的几乎就试一把实现自定义的按钮样式.

## 开始

实现相应的效果需要先把原来的系统效果隐藏掉. 这里找到的是-webkit-appearance这个属性, 这个属性可将任意的元素转化成相应浏览器定义的元素的外观如:

<style>

.radio-div {

-webkit-appearance: radio;

}

</style>

<div class="radio-div"></div>

上面这样就可以将一个div改造成一个单选按钮的样子, 当然只是有个样子而已.

我们这里用到的属性值为: none, 这个值可以将原来的input的的样子给隐藏掉, 从而为后面的自定义外观流出空间.

这里需要注意的是: 这个属性在现代浏览器上基本可以工作. 旧版浏览器却是并不支持. 具体的设置有很多如:

* checkbox
* radio
* button
* caret
* textarea

上面只是列举了一些属性, 这个不是关注的重点. 这里首先实现效果:

<fieldset class="form-group">

<legend>是男的吗?</legend>

<label for="pseudoRadioIsMan">是</label>

<input class="pseudo-radio" id="pseudoRadioIsMan" type="radio" name="man">

<label for="pseudoRadioNotMan">不是</label>

<input class="pseudo-radio" id="pseudoRadioNotMan" type="radio" name="man">

</fieldset>

<style>

.pseudo-radio {

position: relative;

width: 16px;

height: 16px;

border-radius: 50%;

border: solid 1px #ccc;

vertical-align: middle;

}

.pseudo-radio:checked::after {

content: '';

position: absolute;

top: 50%;

left: 50%;

width: .5rem;

height: .5rem;

margin-top: -.25rem;

margin-left: -.25rem;

border-radius: 50%;

background-color: pink;

}

.pseudo-radio:checked {

border-color: pink;

}

</style>

这里通过判断radio是否选择来在其内部添加一个after伪元素来展示需要的外观. 不过这里的实现其实并不好, 因为input元素不是一个容器. 其余的实现如checkbox等的原理也是一样.

从这里可以知道:

1. 这个实现并不能兼容旧有浏览器, 在不支持appearance的浏览器中, 原有的按钮外观还在, 甚至还会出现两个按钮同时存在的情况,
2. input并不是一个容器, 不适合在其内使用伪元素
3. rem也不要在这里用了.
4. border-radius在IE8下也用不了
5. 解决1, 我们这里就直接将input给隐藏掉display: none;,
6. 解决2, 需要用一个元素来替代input, 而且能够通过input来选择到, 这个时候就可以使用相邻元素选择, 替代的元素可以用任何喜欢的元素, 这里我倾向于使用label, **多个label可以对应一个input**, 这样也不会造成理解上的困难.
7. 解决3, 换个单位.
8. 解决4, 好像没有比较好的办法, 有看到这里而且有比较好的办法的烦请告知一下.

<fieldset class="form-group">

<legend>是男的吗?</legend>

<label for="pseudoRadioIsMan-v2">是

<input class="pseudo-radio-v2" id="pseudoRadioIsMan-v2" type="radio" name="man">

<label for="pseudoRadioIsMan-v2"></label>

</label>

<label for="pseudoRadioNotMan-v2">不是

<input class="pseudo-radio-v2" id="pseudoRadioNotMan-v2" type="radio" name="man">

<label for="pseudoRadioNotMan-v2"></label>

</label>

</fieldset>

<style>

**#pseudoFormV2** **input** {

display: none;

}

.pseudo-radio-v2 + **label** {

display: inline-block;

position: relative;

width: 16px;

height: 16px;

border-radius: 50%;

border: solid 1px #ccc;

vertical-align: middle;

}

.pseudo-radio-v2:checked + **label**::after {

content: '';

position: absolute;

top: 50%;

left: 50%;

width: 8px;

height: 8px;

margin-top: -4px;

margin-left: -4px;

border-radius: 50%;

background-color: pink;

}

.pseudo-radio-v2:checked + **label** {

border-color: pink;

}

</style>

使用雪碧图的实现也是差不多, 只是原来由伪元素显示的样式改由label本身来体现, 将背景设定到相应的位置可以.

作为对比的是, 雪碧图需要进行的设置相应会少一些, 而且兼容性比伪元素要好, 但是灵活性则较低, 伪元素修改几个地方就能够实现不同的效果, 而雪碧图则需要重新制作一个图. 不过相应地, 雪碧图的实现具有更好的兼容性, 不用考虑属性在各个浏览器上的兼容性. 个人还是倾向于使用属性的实现. 毕竟, 就算是不兼容的浏览器最多只是看到一个方形的单选按钮, 虽然会有些误导, 但影响其实不大. 毕竟选两下就知道了.

------------------------------------------------- 我是分割线 -------------------------------------------------

有同学提到代码效果实现里面出现了行内元素的对齐问题, 这个问题是因为我对所有的label标签添加了vertical-align: middle属性, 而在上面的实现中label标签是两层嵌套的形式. 由于vertical-middle的对齐特性, label内部的文字等元素的位置, 就跟外部的文字等无法正确对应. 对于vertical-align的详细解释可以看[这里](http://www.zhangxinxu.com/wordpress/2010/05/%E6%88%91%E5%AF%B9css-vertical-align%E7%9A%84%E4%B8%80%E4%BA%9B%E7%90%86%E8%A7%A3%E4%B8%8E%E8%AE%A4%E8%AF%86%EF%BC%88%E4%B8%80%EF%BC%89/)

赞0

## 自定义网页右键菜单

# 00X1 contextmenu法

<div contextmenu="mymenu" >

<div class="context">这里有一个菜单,请在firefox打开哟</div>

<menu id="mymenu" type="context">

<menuitem label="呼呼" icon="1.jpg" onclick="doSomething()"></menuitem>

<menuitem label="嘻嘻" icon="2.jpg" onclick="doSomethingElse()"></menuitem>

<menu label="哈哈">

<menuitem label="呼呼" icon="1.jpg" onclick="doSomething()"></menuitem>

<menuitem label="嘻嘻" icon="2.jpg" onclick="doSomethingElse()"></menuitem>

</menu>

</menu>

</div>

使用小TIP

* 该属性仅在firefox浏览器有效，测试请在firefox下，所以局限性比较强
* 属性contextmenu的值与menu标签id的值对应；
* 添加menu标签时，要加上**type="context"** 这个属性，要不然不灵
* 在标签里添加上icon属性，可以添加小图标
* 想要子菜单的效果，直接在menu标签中再增加menu标签，看上事例即可明白；

素不素hin简单~~~

# 00X2 JS纯手工打造法

用contextmenu只能在firefox下使用，且只能在原有的菜单上新增菜单键，所以呢，下面用js来纯手工打造，原理呢大概就是点击某个特定的区域之后，获取该点的位置，在该处添加一个菜单列表，比如用ul来做，恩！想想也是简单的呢，代码如下，该注释的都注释了。

<div class="menu context">这里还有一个菜单,呼呼</div>

<div class="menu context">这里还有一个菜单,嘻嘻</div>

<div class="menu context">这里还有一个菜单,哟哟</div>

<div id="supermenu" style="display: none;">

<ul>

<li><a href="#">menu1</a></li>

<li><a href="#">menu2</a></li>

<li><a href="#">menu3</a></li>

</ul>

</div>

</div>

这里，还有一个点要注意的事，我们可以用鼠标右键事件，也可以用HTML5里有所的contextmenu事件来写我们需要的功能。

## contextmenu事件法

$=**function**(element){

**return** document.querySelectorAll(element);

}

**var** mymenu=document.getElementById("supermenu");

*//监听contextmenu事件*

**for**(**var** i=0;i<$(".menu").length;i++){

$(".menu")[i].addEventListener("contextmenu",**function**(e){

e.preventDefault();*//阻止原有的菜单*

e.returnValue=false;*//在IE中阻止原有的菜单*

mymenu.style.cssText='display: block;'; *//先显示后才能获取menu的高宽*

**var** menuwidth=mymenu.offsetWidth;*//获取自定义菜单的高宽*

**var** menuheight=mymenu.offsetHeight;

**var** clientwidth=document.documentElement.clientWidth;*//获取当前可视窗口的高宽*

**var** clientheight=document.documentElement.clientHeight;

**if**(clientheight-e.pageY>=menuheight){ *//底下有足够空间时*

mymenu.style.top=e.pageY+"px";*//菜单在下方*

}**else**{

mymenu.style.top=(e.pageY-menuheight)+"px";

}

**if**(clientwidth-e.pageX>=menuwidth){ *//右边有足够空间*

mymenu.style.left=e.pageX+"px";*//菜单有右边*

}**else**{

mymenu.style.left=(e.pageX-menuwidth)+"px";

}

console.log(clientheight-e.pageY,clientwidth-e.pageX);

})

document.body.addEventListener("click",**function**(e){ *//点击其他地方菜单消失，可恢复默认的菜单事件*

mymenu.style.cssText='display: none;';

})

}

小TIP

在兼容DOM浏览器中，使用event.preventDefault()取消，在IE中，将event.returnValue的值设为false即可。  
其他的点差不多都注释了，就不多说了

## mousedown事件法

可以根据mousedown事件对象的button值来判断

|  |  |  |
| --- | --- | --- |
| **鼠标事件** | **button值** | **备注** |
| click | 0 | 单击鼠标左键 |
| click | 1 | 单击鼠标中键 |
| mousedown | 0 | 按下鼠标左键 |
| mousedown | 1 | 按下鼠标中键 |
| mousedown | 2 | 按下鼠标右键 |

click事件只会在单击鼠标左键和中键时触发，单击右键时不会触发

所以呢，只要把上面的代码轻轻一改就可以了。

<script type="text/javascript">

$=**function**(element){

**return** document.querySelectorAll(element);

}

**var** mymenu=document.getElementById("supermenu");

**for**(**var** i=0;i<$(".menu").length;i++){

$(".menu")[i].addEventListener("mousedown",**function**(e){

**if**(window.event) e=window.event; *//兼容性*

**if**(e.button==2){

document.oncontextmenu=**function**(e){

e.preventDefault();*//阻止原有的菜单，可以不用再加e.returnValue=false;亲测。*

}

mymenu.style.cssText='display: block;'; *//先显示后才能获取menu的高宽*

**var** menuwidth=mymenu.offsetWidth;*//获取自定义菜单的高宽*

**var** menuheight=mymenu.offsetHeight;

console.log(menuheight,menuwidth);

**var** clientwidth=document.documentElement.clientWidth;*//获取当前可视窗口的高宽*

**var** clientheight=document.documentElement.clientHeight;

**if**(clientheight-e.pageY>=menuheight){ *//底下有足够空间时*

mymenu.style.top=e.pageY+"px";*//菜单在下方*

}**else**{

mymenu.style.top=(e.pageY-menuheight)+"px";

}

**if**(clientwidth-e.pageX>=menuwidth){ *//右边有足够空间*

mymenu.style.left=e.pageX+"px";*//菜单有右边*

}**else**{

mymenu.style.left=(e.pageX-menuwidth)+"px";

}

console.log(clientheight-e.pageY,clientwidth-e.pageX);

}

})

document.body.addEventListener("click",**function**(e){ *//点击其他地方菜单消失，可恢复默认的菜单事件*

mymenu.style.cssText='display: none;';

document.oncontextmenu=**function**(e){}*//需要加上这个，否则还是不可恢复原来的默认菜单事件*

})

}

赞6

所用技术：html & css & JQuery  
效果链接：<https://justplay197.github.io/baidu-front-end/P3.html>  
1、编写移动鼠标时获取坐标值的函数，并将坐标值存到变量中

**var** mouseX;

**var** mouseY;

**function** **mouseMove**(ev){

ev = ev || window.event;

**var** mousePos = mouseCoords(ev);

mouseX = mousePos.x;

mouseY = mousePos.y;

}

**function** **mouseCoords**(ev){

**if**(ev.pageX || ev.pageY){

**return** {x:ev.pageX,y:ev.pageY};

}

**return** {

x:ev.clientX + document.body.scrollLeft - document.body.clientLeft,

y:ev.clientY + document.body.scrollTop - document.body.clientTop

}

}

2、绑定网页鼠标移动效果

document.onmousemove = mouseMove*;*

3、在div中设定右键时调用rightClick函数

<**div** id="click-me" oncontextmenu="rightClick()"></**div**>

4、函数具体内容，通过设置self.event.returnValue来屏蔽区域内部右键属性

**function** **rightClick**() {

**var** clientWidth = document.body.clientWidth;

**var** divHeight = 200;

**var** menuWidth = 150;

**var** menuHeight = 31 \* $("#menu label").length;

$("#menu").css("visibility", "visible");

$("#menu").css("left",mouseX > clientWidth - menuWidth ? mouseX - menuWidth : mouseX);

**if**(mouseY <= divHeight)$("#menu").css("top",mouseY > divHeight - menuHeight ? mouseY - menuHeight : mouseY);

self.event.returnValue=false

}

5、优化左键取消菜单栏

$("#menu label").click(**function**(){

$("#menu").css("visibility","hidden");

});

$("body").click(**function**(){

$("#menu").css("visibility","hidden");

})

赞2

### 可以全局取消浏览器默认的右键事件

window.oncontextmenu = **function**(){**return** false;}

### 鼠标指针位置：

1. screenX/Y: 参照点也就是原点是整个屏幕的左上角，不限于浏览器内。
2. clientX/Y: 参照点是浏览器内容区域的左上角（浏览器显示区域的左上角），该参照点会随之滚动条的移动而移动。
3. pageX/Y：参照点也是浏览器内容区域的左上角（整个页面的左上角），但它不会随着滚动条而变动  
   页面宽高：
4. innerWidth和innerHeight ：当前可视区域的宽度和高度，随浏览器窗口大小变化
5. scrollWidth和scrollHeight: 整个页面的宽高（整个html页面）
6. clientWidth和clientHeight：屏幕可视区域宽高，不包括边框（有内容的部分）

### 菜单显示位置实现

var winW = document.body.clientWidth,

winH = document.body.clientHeight;

var pageX = event.pageX,

pageY = event.pageY;

*// var site = GetObjPos(eventDeal.getTarget(bd));*

*// console.log(site.x,site.y)*

**if**(winW-pageX<=480 && winH-pageY<=200){

area.style.top = pageY-200 + 'px';

area.style.left = pageX-480 + 'px';

}**else** **if**(winW-pageX<=480){

area.style.left = pageX-480 + 'px';

area.style.top = pageY + 'px';

}**else** **if**(winH-pageY<=200){

area.style.top = pageY-200 + 'px';

area.style.left = pageX + 'px';

}**else**{

area.style.top = pageY + 'px';

area.style.left = pageX + 'px';

}

area.style.display = 'block';

### 阻止事件冒泡

对各个浏览器之间不同的事件处理机制进行兼容处理

**var** eventDeal = {

getEvent: **function**(**event**){

**return** **event**?**event**:window.event;

},

*// 1.IE浏览器的事件处理程序和其他的浏览器不同，在IE浏览器中，使用attachEvent()这个方法*

*// 设置事件监听，而在其他的浏览器中，使用addEventListener()进行处理*

addHandler : **function**(element,**type**,handler){

**if**(element.addEventListener){

element.addEventListener(**type**,handler,false);

}**else** **if**(element.attachEvent){

element.attachEvent('on'+**type**,handler);

}**else**{

element['on'+**type**] = handler;

}

},

*// 2. 兼容DOM的浏览器会将一个event对象传入到事件处理程序中，其中每个事件包含的属性和方法*

*// 都不同，但是有些属性和方法是所有的事件共有的：*

*// bubbles:表明事件是否可以冒泡*

*// cancelable:表示是否可以取消事件的默认行为*

*// preventDefault():取消事件默认行为，但是需要cancelable是true*

*// stopPropagation():取消事件的进一步冒泡或者捕获,需要bubbles是true*

*// stopImmediatePropagation():取消事件的进一步冒泡或者捕获，同时阻止任何事件处理程序被调用*

*// target:事件的目标，指向触发的元素*

*// IE中的事件对象有一些不同*

*// cancelBubble:默认值为false,设置为true就可以取消事件冒泡，等同stopPropagation()*

*// returnValue:默认true，设施false等同preventDefault()*

*// srcElement:等同target*

getTarget: **function**(**event**){

**return** **event**.target || **event**.srcElement;

},

preventDefault: **function**(**event**){

**if**(**event**.preventDefault){

**event**.preventDefault()

}**else**{

**event**.returnValue = false;

}

},

stopPropagation: **function**(){

**if**(**event**.stopPropagation){

**event**.stopPropagation();

}**else**{

**event**.cancelBubble = true;

}

}

}

## 正则表达式之入门

首先重复一下任务描述

1. 编写一个判断给定数字是否为手机号码的正则表达式，测试用例参照但不限于：
2. 18812011232 // 测试结果应该为 true
3. 18812312 // false
4. 12345678909 // false
5. 编写一个判断输入的字符串是否有相邻重复单词的正则表达式，测试用例可以参考但不限于：
6. foo foo bar *// true*
7. foo bar foo *// false 有重复单词但是不相邻*
8. foo barbar bar *// false*

## 关于匹配手机号码

手机号码相信大家都很轻松，最差不过\d{11};  
然后匹配第一位是1的1\d{10};  
匹配第二位是34578的1[34578]\d{9};  
匹配第二位是3时第三位是012356789的1(3[0-35-9]|[4578]\d)\d{8};  
……以此类推

## 关于匹配相邻重复单词

### 简单的单词重复匹配

首先是一个单词:[A-Za-z]+;  
然后是相邻重复的字符串:([A-Za-z]+)\s+\1;  
然后加上单词边界:\b([A-Za-z]+)\s+\1\b;

相信大多人都能做到这一步，接着我想起了一个问题，font-size算不算是一个单词呢？

### 匹配带有连字符的重复单词

先不管它算不算，匹配font-size倒是一个值得研究的问题。  
所以加上可能出现的-连字符:\b([A-Za-z]+(?:-[A-Za-z]+)\*)\s+\1\b;  
这里说明一下(?:)是**“非捕获型分组”**的意思，用下面的代码说明比较直观：

"abcabc".match*(/(a)*.\*\1/g); *// ["abca"]*

"abcabc".match*(/(?:a)(b)*.\*\1/g); *// ["abcab"]*

我们用()时就会临时保存一个分组1，使用1就可以引用，但当我们不想保存这个分组时我们就可以使用(?:)，毕竟js里我们只有9个分组可以使用。

让我们尝试一下：

**var** **reg** = /\b([A-Za-z]+(?:-[A-Za-z]+)\*)\s+\1\b/**g**;

"font-weight font-size font-size".match(**reg**); *// ["font-size font-size"]*

"font-size font-weight font-size".match(**reg**); *// null*

"font-size font-sizesize font-weight".match(**reg**); *// null*

不错，完美。

### 试试匹配各种符号乱入的单词吧

然而很快我就发现了一个严重的问题，那就是上面的正则匹配foo -foo foo或者bar bar- bar甚至aaa-aaa aaa居然能成功！

**var** **reg** = /\b([A-Za-z]+(?:-[A-Za-z]+)\*)\s+\1\b/**g**;

"foo -foo foo".match(**reg**); *// ["foo foo"]*

"bar bar- bar".match(**reg**); *// ["bar bar"]*

"aaa-aaa aaa".match(**reg**); *// ["aaa aaa"]*

这问题可就大了，然后我在控制台里测试了一下

"-abc".match(/.?\b.?/g); // ["-a", "c", ""]

"def-".match(/.?\b.?/g); // ["d", "f-"]

懂了，连字符-刚好在单词边界外。除此之外，我又意识到了一个问题，假如单词边界外是乱七八糟的符号呢？

var reg = /\b*([A-Za-z]+(?:-[A-Za-z]+)*\*)\s+\1\b/g;

"aaa@# $%^&\*aaa aaa".match*(reg)*; *// ["aaa aaa"]*

呵呵

### 弱鸡debug的辛酸结局

仔细想想，其实只需要判断单词两边是不是空白符就好了对吧，考虑到开头和结尾，还需要加上^和$的情况，所以改成(?:^|\s)\b([A-Za-z]+(?:-[A-Za-z]+)\*)\s+\1\b(?:\s|$);  
让我们来试试

var reg2 = /*(?:^|\s)*\b*([A-Za-z]+(?:-[A-Za-z]+)*\*)\s+\1\b*(?:\s|$)*/g;

"aa aa".match*(reg2)*; *// ["aa aa"]*

"aa-aa aa-aa".match*(reg2)*; *// ["aa-aa aa-aa"]*

"aa bb aa".match*(reg2)*; *// null*

"aa aa- aa".match*(reg2)*; *// null*

"aa-aa aa".match*(reg2)*; *// null*

"aa-aa aa bb-bb bb-bb".match*(reg2)*; *// [" bb-bb bb-bb"]*

不错，这次正常了，不过仔细看很快又发现了一个问题

var reg2 = /*(?:^|\s)*\b*([A-Za-z]+(?:-[A-Za-z]+)*\*)\s+\1\b*(?:\s|$)*/g;

"bb aa aa bb".match*(reg2)*; *//[" aa aa "]*

没错，我们是判断了单词两边是否是空白符，但同时也匹配到了单词两边的空白符，后面的空白符可以改成(?=\s|$)，这里解释一下，(?=)叫做**“向前正向匹配分组”**，同样，我们用代码解释

"abcd".match(/ab(?=c)/g); // ["ab"]

"abbc".match(/ab(?=c)/g); // null

当我们使用向前正向匹配时，只有之后的字符串与之匹配时才能匹配整个字符串。  
当然，同样的我们还有一个叫**“向前负向匹配分组”**的东西(?!)，跟(?=)相反，只有当之后的字符串不匹配时才能匹配整个字符串。

"abcd".match(/ab(?!c)/g); // null

"abbc".match(/ab(?!c)/g); // ["ab"]

这是我最后得到的正则表达式:  
**(?:^|\s)\b([A-Za-z]+(?:-[A-Za-z]+)\*)\s+\1\b(?=\s|$)**

然而我还是想不到有什么好的办法解决掉前面的空格，如果有大神解决了这个问题，望告知。  
最后欢迎review

[作业地址](http://ife.baidu.com/work/detail/id/2167) [代码地址](http://git.oschina.net/usaki_133/ife_2017/tree/master/regexp) [预览地址](http://tuziel.com/ife_2017/regexp/index.html)

之前学过一遍正则，但具体运用上经常实践拿（bai）来(du)主义，这就导致了书到用时方恨少，百度知多少。。。（不能偷懒啊，正则是个美妙的东西）

我一直觉得带着问题去学习是最快的。刚好借着这个机会学习一波

实现一个正则验证手机号码

借着问题把学习拆分一下：

1. 正则写出来要能验证数字。=>需要我们学习字符类去匹配不同的类型（数字，字母等）
2. 学习完字符类，知道了/[0-9]/能匹配数字，但验证11位手机号码我们是否需要写11遍？这个时候就需要去学习量词减少重复等书写[0-9]。=> /[0-9]{11}/
3. 现在的正则已经能验证11位数字了，如果想书写更加简洁一些，需要学习元字符去简写。=>/d{11}/
4. 然而现在写的正则只要是含有11个连着的数字都能通过验证，需要去限定有其他字符的出现，此时就需要去学习锚点 去限制开头和结尾都是 => /^d{11}$/
5. 在做一些微调，比如我国手机都是1开头的，可以这么写/^1d{10}$/提高匹配精确度。还有一些更加细的规则比如 134[0-8],135,136,137,138,139,150,151,157,158,159,182,187,188 开头是移动手机号等等，完善到更详细实用吧

附上我自己的学习笔记，以及[我的实现](http://leon-z.me/2017-Baidu-IFE/baidu-rice/getting-start-regexp/)

#### 正则表达式

* 概念：
  + 描述字符串规则的表达式
* 生成方式
  + /pattern/attrs
  + new RegExp(pattern,attrs)

##### regexObject.test(str)

* 测试正则表达式与指定字符串是否匹配
* 只需要字符串包含正则所描述的规则就返回true，否则返回false

##### 锚点

* 匹配一个位置
  + ^:起始位置
    - /^http:/ 匹配http起始的字符串
    - /^139/ 匹配139起始的字符串
  + $:结尾位置
    - /.jpg$/ 匹配.jpg结尾的字符串，用于判断文件类型
  + b:表示单词边界
    - /bisb/ 匹配is 这个单词
    - /bisb/.test('this') // false
    - /bisb/.test('this is tom') // true

##### 字符类

* 匹配一类字符中的**一个**
  + /[abc]/ 匹配a或b或c
  + /[0-9]/ 匹配任意一个数字，等价于手动输入123456789
  + /[^0-9]/ **^在字符类中的含义和在锚点中含义不同，在字符类中表示‘非’**
  + /[a-z]/ 匹配任意一个字母
  + . 点号匹配除了换行意外的任意字符

##### 元字符

* 具有特殊意义的元字符
  + ^,$,b
  + d :[0-9]字符类的缩写
  + D :[^0-9]非数字，也可以写作[^d]
  + s :空白符
  + S :非空白符
  + w :[A-Za-z0-9]单词字符
  + W :非单词字符

##### 量词

* 出现的次数
  + {m,n} :m~n次
    - ：{1,}出现1到无穷次，即至少出现一次
  + ？ : {0,1}出现0或者1次。
    - : {0,}出现 0到无穷次

#### 正则的一些其他东西

##### 转义符

* 需要匹配的字符中包含元字符，因为元字符有特殊含义，如果需要匹配的是元字符作为字符串的内容，则需要加上转义符 ``
  + 匹配http开头的字符串 : /^http:/// => /http:///
  + 匹配网址中的. ：/@163.com$/ => /@163.com/

##### 多选分支

* 或 ：|
  + /thi(c|n)k/ 匹配 thick或 think 也可以用/thi[cn]k/表示
  + /.(jpg|png|gif)$/ 匹配图片文件

##### 捕获

* 保存匹配到的字符串，之后再用
  + () :捕获
  + (?:) :不捕获
* 使用
  + $1,$2,...
  + API参数或返回值

1.str.match(regexp)

* 获取匹配的字符串
* str.replace(regexp/substr,replacement)
* 替换一个子字符串

捕获练习：

1. 获取url的protocol,host,pathname,search,hash等各个部分

**let** url ='https://leon-z.me/blog/32?id=2#comment';

**let** reg=/(https?:)//([^/]+)(/[^?]\*)?(?[^#]\*)?(#.\*)?/

**let** arr=url.match(reg)

console.log(arr)

1. 替换'this price is 5.'中的价格为'5.00'的表示方法  
   ```javascript  
   let str='this price is 5';

console.log(str.replace(/(d+)/,'$1.00'))  
```

##### 更强大的检索能力：regexpObject.exec(str)

* 更详尽的结果
* 过程等状态

一路以来遇到正则表达式的地方都是直接去百度的，因为感觉大部分遇到的正则表达式的地方都是很常见的。这次顺便总结下正则表达式比较难记的部分，方便自己日后查阅吧(￣.￣)

正则表达式中的特殊字符

感觉[MDN](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide/Regular_Expressions)里面的资料还是挺不错的，特此记录下里面提到过的个人感觉比较难记的部分。

1. **?**：匹配前面一个表达式0次或者1次。等价于 {0,1}。此外，**?**还有一个比较重要的用途，即

如果?紧跟在任何量词 \*、 +、? 或 {} 的后面，将会使量词变为非贪婪的（匹配尽量少的字符），和缺省使用的贪婪模式（匹配尽可能多的字符）正好相反。例如，对 "123abc" 应用 /d+/ 将会返回 "123"，如果使用 /d+?/,那么就只会匹配到 "1"。  
还可以运用于向前断言，如 x(?=y) 和 x(?!y) 。

1. **(x)**：捕获括号。匹配 'x' 并且记住匹配项。在匹配中可以使用1、2、n表示括号中的内容，而在替换中，则使用$1、$2、$n。
2. **(:x)**: 非捕获括号。匹配 'x' 但是不记住匹配项。
3. **x(?=y)**：匹配'x'仅仅当'x'后面跟着'y'.这种叫做正向肯定查找。例如，/Jack(?=Sprat)/会匹配到'Jack'仅仅当它后面跟着'Sprat'。/Jack(?=Sprat|Frost)/匹配‘Jack’仅仅当它后面跟着'Sprat'或者是‘Frost’。但是‘Sprat’和‘Frost’都不是匹配结果的一部分。
4. **x(!=y)**：匹配'x'仅仅当'x'后面不跟着'y',这个叫做正向否定查找。例如，/d+(?!.)/匹配一个数字仅仅当这个数字后面没有跟小数点的时候。正则表达式/d+(?!.)/.exec("3.141")匹配‘141’但是不是‘3.141’

使用正则表达式的js方法

js提供了不少使用正则表达式的方法。如RegExp的exec，和String的match、replace、search和split方法。

当你想要知道在一个字符串中的一个匹配是否被找到，你可以使用test或search方法；想得到更多的信息（但是比较慢）则可以使用exec或match方法。如果你使用exec或match方法并且匹配成功了，那么这些方法将返回一个数组并且更新相关的正则表达式对象的属性和预定义的正则表达式对象（详见下）。如果匹配失败，那么exec方法返回null（也就是false）。

match与exec区别

从前往往傻傻分不清match和exec的区别，因为两者的作用都是匹配字符串，返回数组，但是在不同情况下两者所返回的内容是不同的。

* **当正则表达式包含子表达式且不包含全局标志g时，二者返回相同的数组。**

示例：

**var** str = 'cat10,bat20,kat30';

**var** patten = /w(at)d+/;

**var** arr = str.match(patten);

arr[0] <=> ['cat10']

arr[1] <=> ['at']

**var** arr = patten.exec(str);

arr[0] <=> ['cat10']

arr[1] <=> ['at']

* **当正则表达式带有全局标志g时，二者表现不一致。**  
  match会返回所有符合条件的匹配项，并以数组形式返回。数组第一项存放第一个匹配项，数组第二项存放第二个匹配项...依次类推。  
  exec则永远返回第一个匹配项。但是当连续调用exec时，则每次的返回值都是下一个匹配项。

示例1：

var str = 'cat,bat,kat';

var patten = /at/g;

str.match(patten); //['at', 'at', 'at']

patten.exec(str); //['at']

示例2：

var str = 'cat,bat,kat';

var patten = /w+/g;

str.match(patten); //['cat', 'bat', 'kat']

//第一次调用

patten.exec(str); //['cat']

//第二次调用

patten.exec(str); //['bat']

//第三次调用

patten.exec(str); //['kat']

* **当正则表达式包含子表达式时且包含全局标志g时，二者表现不一致。**  
  match会返回所有符合条件的匹配项，并以数组形式返回。这时，match不会再返回子表达式的匹配项了。数组第一项存放第一个匹配项，数组第二项存放第二个匹配项...依次类推。  
  exec会返回子表达式的匹配项。换句话说就是，数组第一项存放整个匹配项，数组第二项存放第一个子表达式匹配项，数组第三项存放第二个子表达式匹配项...依次类推。

示例：

var str = 'cat10,bat20,kat30';

var patten = /w(at)d+/g;

var arr = str.match(patten); //['cat10', 'bat20', 'kat30']

var arr = patten.exec(str);

arr[0] <=> ['cat10']

arr[1] <=> ['at']

赞1

## CSS3饼状loading效果

# CCS3饼状Loading效果

## 纯CSS实现圆形/半圆/弧形

* 将border-radius设置为正方形div的长度的一半可得到原型或者圆弧。
* 将两个半圆拼凑成一个正圆。
* 设置width和height设置为0，设置border-width和border-radius为圆的半径，通过设置每条border的颜色可以改变每个直角扇形的颜色，同理可以组合形成半圆、270°扇形。例如为一个div设置css样式如下：

**#div**{

width: 0;

height: 0;

border: 100px solid;

border-radius: 100px;

border-top-color: lightgreen;

border-right-color: deepskyblue;

border-bottom-color: dodgerblue;

border-left-color: royalblue;

}

则有如下效果：



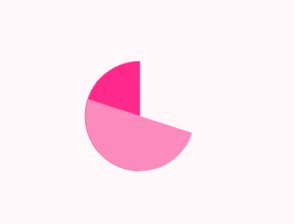
* 通过clip-path可产生圆形、以矩形的四边为直径的半圆、以矩形的四边为弦的弓形。
* 通过设置盒的非对角的border-radius为所夹边长度的一半，则可得到以矩形的四边为直径的半圆。

## Loading效果实现思路

观察效果图得知，整体效果可以分为两部分，首先是外层的四分之三弧线的旋转，以及内层的较为复杂的旋转。  
由于弧长为圆周的3/4，则可采用前部分所述的第一种方法，设置某条边的border-color为transparent即可。旋转则使用CSS3 Animation配合transform即可。  
内层较为复杂的旋转可以分为两层效果来实现，最底层为一个如图所示的正圆：



在其上有两个颜色各异的半圆在进行如图所示的旋转（单个周期）：



即可实现示意图中的效果。

* bug：在竖直方向上浅色半圆和底部完整的圆形会有一条1px的偏差无法调整，初步估计其中涉及到border的渲染机制，若关于此有任何建议，请不吝赐教。
* DEMO页：<http://codepen.io/XXXXat/pen/dvPVeR>
* Github：<https://github.com/Minxine-Yao/IFE-2017>

赞19

# 实现3 / 4圆环

我们假设要创建的圆环节点的半径为R，首先是将div变成圆环。

{

border: solid color 2px;

border-radius: R / 2 px;

}

接下来我们将圆环设为3 / 4圆环。想想一个div节点，如果我们只为其设置3边的border，那么不就是3 / 4矩形环吗，那么圆环呢？试一下。

{

border: solid color 2px;

border-radius: R / 2 px;

border-top-color: transparent;

}

果然和我们想的一样，这下圆环就完成了。

# 实现圆饼

先来看看效果图，先是从一个红的圆逐渐变为粉色的圆，然后是从粉丝的圆变成了红色的圆，以此循环下去......该怎么实现？这里我们创建两个半圆分别为红色和粉丝，和一个整圆一半红色一半粉色，接下来画圆。

{

width: 100px;

height: 100px;

border-radius: 50px;

background-color: color;

}

等等，这是一个整圆，怎么变成半圆？或者说把半边的颜色去掉？这里我们这么写，我们不设置div的宽和高，我们用 border来填充圆（毕竟我们css也就border这些分为border-left, border-right.....）

{

width: 0;

height: 0;

border: solid color 50px */\* 这里的50就是我们半圆的半径了 \*/*

border-top-color: transparent;

border-left-color: transparent;

}

好了两个半圆画完了，接下来的整圆也同理，只需要一遍的border为红另一边的border为粉就可以了。接下来我们就该考虑如何组合这三个圆来实现动画了。

# 圆饼动画

好，我们看着效果图一步一步分析。  
首先是一整个红圆，我们将两个半圆叠在一起，红色在上，粉色在下，另一边露出整圆的红色那半面。  
接下来红圆逐渐变为粉圆，这里我们将粉半圆逐渐旋转180度，这时候就一半一半了，然后呢？旋转下面的整圆让粉色的旋转出来？不不，这样子动画就得转3个圆了，还得一直改z-index，我们尽可能的让执行动画的节点变少，我们这里讲红半圆的z-index改一下，让它旋转180度到红半圆的下面，这样子他们下面的整圆的粉色部分不就逐渐露出来了吗？这下不正好实现了半个动画吗。接下来就旋转粉半圆180度，让它下面的红半圆逐渐露出来，然后是改变红半圆的z-index 让其在最上然后接着180......然后循环上面的步骤就可以了。

好了大功告成！

我的代码：<https://github.com/LNoe-lzy/baiduCourse/blob/master/course6/index.html>

《CSS3圆饼loading效果》作为糯米学院中的进阶任务，目的主要是让学员通过完成任务，掌握border-radius属性的使用以及旋转动画的设置。在已经完成鼠标悬浮效果任务之后，相对来说本任务要简单很多。以下是总结。

　　任务分解：该任务要达到的[效果](http://ife.baidu.com/course/detail/id/36)。如网页中显示情况，可将任务分解为两个小任务：

1.制作四分之三圆环并旋转；  
2.制作圆饼，圆饼内有两种颜色，分别交替旋转扩大其扇形区域（从0deg到到360deg），以达到圆饼loading效果。

　　通过完成任务，使用到如下一些重要属性：

1.制作圆环，使用伪元素，使用border(包括其transparent值)；  
2.制作圆饼过程中，使用到animation-timing-function属性的steps()曲线；  
3.圆饼及圆环的旋转，使用animation系列属性，并使用transform设置旋转角度，使用transform-origin属性设置旋转中心。

　　以下是完成本任务后的html结构：

<header>

<h1><a href="http://smallstarz.com/">小星的博客|Small Star's Blog</a></h1>

<ul>

<li><a href="#top-bottom1">点我</a></li>

<li><a href="#top-bottom2">快点我</a></li>

<li><a href="#top-bottom3">点我吧</a></li>

<li><a href="#top-bottom4">点我啦</a></li>

</ul>

</header>

<article>

<p class="round1">

<span class="bgcircle">

<span class="left"><span class="spining"></span></span>

<span class="right"><span class="spining"></span></span>

</span>

</p>

<p class="round2">

<span class="bgcircle">

<span class="inner">

<span class="spiner"></span>

<span class="filler"></span>

<span class="masker"></span>

</span>

<span class="inner2">

<span class="spiner"></span>

<span class="filler"></span>

<span class="masker"></span>

</span>

</span>

</p>

</article>

<footer>

<p>版权所有 &copy 小星</p>

</footer>

</body>

</html>

　　其中，round1为使用第一种方法制作圆饼时的loading整体结构，round2为使用第二种方法制作圆饼时的loading整体结构。可以明显发现round2方法结构复杂，实际上其实现方法也较round1方法笨拙。结构样式代码请见文末demo代码地址。页面图如下：



　　圆环及圆饼制作原理如下：

　　1.四分之三圆环的旋转：  
　　实现原理：首先将上文html结构中的类名为bgcircle的span元素设置为圆形（将其border-radius设置为50%即可），其宽高大于两个子元素span，以便圆环与圆饼分开。在.bgcircle上设置伪元素，同样设置为圆形，大小与.bgcircle相同，再设置伪元素边框，其中任一边框设置为transparent即可。最后利用animotion和transform设置旋转动画。css代码如下：

*/\*2-1-1.制作大圆背景\*/*

.bgcircle{

position:relative;

display:inline-block;

height:80px;

width:80px;

margin:50% 0 0 50%;

top:-30px;

left:-30px;

border-radius:50%;

overflow:hidden;

}

*/\*2-2.loading圆环\*/*

.bgcircle::before{

content:"";

position:absolute;

width:100%;

height:100%;

box-sizing:border-box;

border:4px solid #BD50A3;

border-top:4px solid transparent;

border-radius:50%;

-webkit-animation:circle-spining 4s infinite linear;

animation:circle-spining 4s infinite linear;

-webkit-transform-origin:50% 50% 0;

transform-origin:50% 50% 0;

}

@**keyframes** circle-spining{

0%{transform:rotate(0deg);}

50%{transform:rotate(-180deg);}

100%{transform:rotate(-360deg);}

}

@-**webkit**-**keyframes** circle-spining{

0%{-webkit-transform:rotate(0deg);}

50%{-webkit-transform:rotate(-180deg);}

100%{-webkit-transform:rotate(-360deg);}

}

　　2.圆饼旋转：  
　　（1）方法一：主要参考[codepen](http://codepen.io/)网站中[Geoffrey Crofte](http://codepen.io/CreativeJuiz/)的  
[CSS Loader](http://codepen.io/CreativeJuiz/pen/vFBIh)demo，其中也写了一些其他loading效果。总体来说，这些loading效果的主要原理都是：在.bgcircle之下，设置.left、.right为左右半圆，拼接为一个圆饼，对.left，设置其子元素.spining往右移至.right区域，并形成半圆与.right区域重叠，此时只有.spining旋转至.left区域时才能显示出来。同理，设置了.right的子元素.spining。最后，对两个.spining子元素设置不同的旋转动画已达到要求效果。css代码过长，请见文末demo代码地址，其旋转过程如下步骤表述：

　　a.初始状态下，两个.spining都没有旋转（图中深紫色圆饼）：此时.left.spining围绕圆心开始逆时针旋转，至-180deg：



　　b.“.left.spining”围绕圆心开始逆时针旋转，至-180deg（图中浅紫色半圆）：



　　c.“.left.spining”停止旋转，“.right.spining”开始旋转至-180deg，此时两个.spining都显示出来了，形成浅紫色圆饼：



　　d.“.right.spining”停止旋转，“.left.spining”又开始旋转，从-180deg旋转至-360deg，此时只剩下“.right.spining”形成的浅紫色右半圆：



　　e.“.left.spining”一周的旋转动画结束，停止旋转；“.right.spining”开始最后半周的旋转，至-360deg停止。整个一周的动画结束。  
对此动画设置无限循环次数，即可达到最终效果。



　　（2）方法二：参考[张鑫旭博客](http://www.zhangxinxu.com/)的文章[CSS3实现鸡蛋饼饼状图loading等待转转转](http://www.zhangxinxu.com/wordpress/2014/04/css3-pie-loading-waiting-animation/)，  
在张鑫旭前辈的这篇文章里，利用比喻把他的方法原理表述的很清楚，简单来说：他将动画分成了两半，如果以本文前面页面图中右边的圆饼作为例子的话，前半动画是深蓝色由12点钟位置开始逆时针旋转至-360deg，最后将橙色覆盖；后半动画刚好相反，橙色由12点钟位置开始逆时针旋转至-360deg，最后将深蓝色覆盖。由于两半部分动画的过程是相同的，所以将前半部分动画制作完成后，将该动画过程复制一份，将颜色对调，重新套入相同结构的新元素中，再将新的一套重叠在旧一套元素所占区域，设置前半段时间第一套元素显示、第二套元素透明，后半段时间第二套元素显示、第一套元素透明。这样就完成了圆饼loading效果。  
下面以前半部分动画为例表述下动画过程：

　　首先需要了解html结构（以本文开始所列html结构为例），设置类名为inner的span作为父元素，该元素背景为深蓝色，其下同列三个span作为子元素，类名分别为：spiner、filler、masker。以上四者中，按照原理，.filler、.masker的z-index应该设置得高于.spiner，但在实际设置中，张鑫旭前辈没有设置z-index值，而是利用在html结构中将需要优先显示的元素并列在后面的方式（如果在html结构中将.masker与.spiner对调，就会出现错误），规避了这点。.masker为左半圆，深蓝色；.filler为右半圆，橙色；.spiner为旋转半圆，橙色，初始位置位于左半部分。

　　a.初始状态：.masker透明，此时由.spiner和.filler分别位于左右方拼接成橙色圆饼：



　　b.此时，.spiner开始围绕圆心逆时针旋转，则作为父元素的.inner开始从12点钟方向逆时针显露（深蓝色），到.spiner旋转-180deg时，.masker由透明变为显示，.filler由显示变为透明：



　　c.之后，.spiner继续逆时针旋转，由于.filler已经透明，则.inner从6点钟方向逆时针显露，而左半圆区域由于.masker已经显示（z-index值比.spiner大），也为深蓝色，这样，当.spiner旋转一周之后，就由橙色全部变为蓝色：



　　以上为动画原理，css代码过长，请见文末demo代码地址。在实现这个方法的过程中，还需要注意，要变化透明度的元素（.inner/.inner2/.masker/.filler），其透明度变化并不是渐变，而是突变，所以需要设置animation-timing-function为steps()突变曲线，steps(n,start/end)主要有两个值，简单来说，n设置突变次数（整数），start/end设置在动画开始还是结束突变。

　　思考：

1.利用steps()曲线，可以制作一些更炫酷的css3动画。并且，在同一个方向上连续渐变的动画，由于渐变位置不同，通常需要两个以上元素实现，但利用steps()曲线突变位置，就能通过一个元素实现了；  
2.完成以上任务之后，可以优化的地方：第一个方法中的.spining子元素，可以使用伪元素替代；同理，第二个方法中的子元素，也可用伪元素替代；  
3.圆环除了定长旋转，也可根据圆饼loading旋转的方法，实现变长旋转loading效果。

　　具体参考已在文中引出，不再单列，以下为我的demo及demo代码：

[我的demo](http://smallstarz.com/baidutask-2017/nuomi-task2-loadingcircle/loadingcircle.html);  
　　[demo代码](https://github.com/smallstar92/baidutask-2017/tree/gh-pages/nuomi-task2-loadingcircle);

## 封装DOM动画类库（一）

以前做动画的时候，常常会卡顿， 猜，大致的原因可能是， 浏览器绘制页面的时候， 反应不过来，所以会出现卡顿。汗！！，然后，今天通过这个任务， 了解到了requestAnimationFrame ，通过百度才知道具体的原因。下面是我自己的理解，如果有不对的地方，大家即时指出来，方便我即时的修改。

浏览器绘制页面的时候， 主要是通过两个线程去绘制页面的。 一个是主线程， 一个排版线程。

# 主线程

通常情况下， 主线线程的负责工作是： 运行javascript 、 计算HTML元素的css样式 、 把页面的元素绘制成一个 或者 多个位图。然后， 把这些位图 移交给 排版线程

（百度了下位图，我把它理解成了用像素点组成的图片， 反正我理解的就是图像。）

# 排版线程

通常情况下， 拿到主线程 传过来的 位图以后， 排版线程主要做一下这么几个工作：

1. 通过CPU渲染位图， 在屏幕上面显示。
2. 向 主线程 请求更新位图的可见部分 或者 即将可见的部分。
3. 判断出当前页面 出于可见的部分， 判断出即将通过页面滚动而可见的部分 、 随着用户滚动页面来移动这些部分。

# CPU

排版线程通过CPU把位图绘制在页面当中。

CPU比较擅长的工作： 绘制位图到屏幕上。 在不同位置， 以不同角度，不同大小，（我感觉可以理解为同一张图，变形之类的）。  
CPU相对慢的地方 ： 将位图加载到显存里面。

讲完上面的现在浏览器就开始绘制页面了。

↓

浏览器下载完页面中所有的组件以后。（组件包括：HTML标签，javascript，css，图片.... 等）

会 生成2个内部数据，也就是我们 通常说的树， 一个是DOM树， 一个是渲染树。

DOM树 代表的是页面的结构 。 渲染树 代表的是 DOM节点如何显示。

DOM树 中的每一个需要显示的DOM节点 在 渲染树中 至少都对应着一个节点。（DOM树中隐藏的节点（display:none）在渲染树中是没有对应节点的 ）。然后，渲染树的节点，就会生成一个盒模型，没错，就是我们打开 f12 右边看到的那个盒模型。

当DOM树 和 渲染树构建完成以后， 浏览器就开始绘制页面了。（怎么绘制页面呢？就是我上面说的通过主线程和排版线程绘制页面）

# 现在重点来了

当DOM发生变化，影响了元素的几何属性（宽和高）， 那么浏览器会重新计算元素的几何属性，同样其他元素的几何属性和位置也会受到影响。这个时候，浏览器 原来的渲染树中受影响的部分就失效了，所以渲染树需要重新构造， 这个过程其实就重排。完成重排以后，浏览器会把受影响的部分重新绘制在屏幕上，这个过程就是重绘。

到这里你肯定恍然大悟了，反正我就是这样。

这些不断的重排重绘 ， 会导致 界面 卡顿。（不断重排重绘 ，其实就是 主线程 和 排版线程之间不断的请求数据（比如不断的请求宽度），如果特别频繁的话（大部分浏览器的显示频率为16.667ms），超过16.667ms肯定会卡。就比如高速公路每秒最多能通过10辆车，突然来了一大批不明车辆，那不就堵车了么，大概是这个意思 ）。

很显然，每次都重排都会导致重绘， 那些情况会导致重排呢？（提升性能的关键）

1. 增加或者删除 可见的DOM元素
2. 元素的位置发生改变
3. 元素的宽高发生改变。
4. 元素的内容发生改变（比如你 一个元素里面高度不固定，里面有一段问题，突然的你把文字替换成了一个张大图片，那不就把元素撑大了么）
5. 页面初始化渲染（ 这个避免不了 ）
6. 浏览器窗口发生变化的时候。

# 下面就是关于requestAnimationFrame

用法就不说了，网上面一大堆。说下原理。

用js 做动画，通常都是setTimeout 和 setInterval。 大部分浏览器的显示频率为16.667ms（ 1000/60 = 16.6666 ）;意思就是 每秒60帧。 如果做动画的时候我们这样写:

setInterval(**function**(){

},10)

明显要小于16.667，所以这个情况就和上面说的堵车一样，本来是16.667ms才能做的事情，非要10ms做完，这不就堵了么，所以说强扭的瓜不甜。

所以W3C就推出了 requestAnimationFrame 这么个东西，它是怎么做的呢？  
requestAnimationFrame是不需要使用者指定循环间隔时间，它会自动的根据当前浏览器的性能 自行去决定最佳的帧速率。所以我才明白为啥任务要让用requestAnimationFrame了。

# 下面就说说我怎么去一步步构建一个动画库。（ps:我基本算个菜鸟）

我是 先百度了上面的内容才开始写动画库，虽然自己以前也写过类似jq的animate,但是那个有BUG，并且正如上面所说的，setInterval和setTimeout写出来的动画，始终会有卡顿的情况，所以我果断的 参考了Velocity.js 的写法。

其实是被这两句话骗过去的。

兼容性

Velocity可以随处可见 - 回到IE8和Android 2.3。在引擎盖下，它模仿jQuery的$ .queue（），从而与jQuery的$ .animate（），$ .fade（）和$ .delay（）无缝地进行交互操作。由于Velocity的语法与$ .animate（）相同，您的代码不需要更改。

加载

Velocity在所有级别的压力下都胜过jQuery，并且优于Transit（领先的CSS动画库），从中等压力水平开始。

大体意思就是兼容IE8 比jquery动画快。 其实也很正常，jq本身就不适合做动画，大家都懂，在手机上面更是卡的不行。

然后我就吸取了上面任务的经验，（ 大家还记得vue第一个任务吗？ 布置任务的老师说他怎么解析vue源码。）然后就在github找到velocity.js， 然后从版本0.0开始解析。

不得不吐槽一下啊， 以前解析别人库源码，傻乎乎的从最新版本解析，整的一脸懵比。这里贴个 [velocity.js 0.0版本的链接](http://github.com/julianshapiro/velocity/tree/0.0.0)

我没有去研究那些 需求分析啊， 接口文档设计啊，之类的东西，因为1， 我没有参考对象，不知道要设计个什么样子的。2 ， 我感觉我就算研究了也不懂。 所以我写的时候，准备仿人家velocity.js。 可以说是盗版，嘿嘿！

在写之前我测试了下requestAnimationFrame和 setTimeout的区别，[点击document触发](http://baiduweb.github.io/nuomi/velocity/%E6%B5%8B%E8%AF%95requestAnimationFrame%E7%94%A8%E6%B3%95.html)  
很明显看出用requestAnimationFrame做出来的动画要明显平稳的多。

因为requestAnimationFrame存在兼容性，然后我就直接把velocity.js的兼容处理直接拔下来了，大家别笑。

*//因为requestAnimationFrame和setTimeout一样都是全局函数可以直接调用的。*

*//这段兼容的意思就是 如果浏览器兼容requestAnimationFrame，就使用requestAnimationFrame,否则就 用setTimeout来代替requestAnimationFrame。*

*// timeDelta 这里时间的计算最终结果为0， 这是js setTimeout 的一个小技巧。大致意思就是 普通队列执行完成以后，才顺序执行setTimeout队列*

**var** requestAnimationFrame = window.requestAnimationFrame || (**function**() {

**var** timeLast = 0;

**return** window.webkitRequestAnimationFrame || window.mozRequestAnimationFrame || **function**(callback) {

**var** timeCurrent = (**new** Date()).getTime(),

timeDelta;

*/\* Dynamically set delay on a per-tick basis to match 60fps. \*/*

*/\* Technique by Erik Moller. MIT license: https://gist.github.com/paulirish/1579671 \*/*

timeDelta = Math.max(0, 16 - (timeCurrent - timeLast) );

timeLast = timeCurrent + timeDelta;

**return** setTimeout(**function**() { callback(timeCurrent + timeDelta); }, timeDelta);

};

})();

因为我是准备这么调用的。

$("#div3").velocity({

opacity : "30"

},{

date : 1000,

**type** **:** "linear",

callback : **function**(){

console.log( "回调" )

}

});

所以开始的时候 ， 我是这么写的。

(**function**(window){

*//模拟一个jquery $。*

window.$ = **function**( obj ){

**var** objList = document.querySelectorAll( obj );

**var** len = objList.length;

**if**( len > 1 ){

**for**(**var** i=0; i<len; i++){

**return** **new** Velocity( objList[i] );

}

}**else**{

**return** **new** Velocity( objList[0] );

}

};

*//创建Velocity 构造函数*

**function** **Velocity**( obj ){

**this**.obj = obj;

};

*//Velocity添加方法及属性*

Velocity.prototype = {

constructor : Velocity,

velocity : **function**( options ){

console.log( options )

}

}

})(window);

然后就是animate的实现。动画形式利用的是贝塞尔曲线， animate实现的核心思想 就是在规定的时间内做完事情。

animate : **function**(){

var json = **this**.json;

var obj = **this**.obj;

var date = **this**.settings.date;

var fx = **this**.settings.type;

var fn = **this**.settings.callback;

var \_this = **this**;

var iCur = {} , startTime = ( **new** Date() ).getTime() ;

**for**( var attr **in** json ){

**if**( attr === 'opacity' ){

iCur[attr] = Math.round( css( obj , 'opacity') \* 100 );

}**else**{

iCur[attr] = parseInt( css( obj , attr ) );

}

};

requestAnimationFrame( animation );

**function** **animation**(){

var changeTime = ( **new** Date() ).getTime();

var t = date - Math.max( 0 , ( startTime - changeTime + date ) );

**for**(var attr **in** json){

var value = Tween[fx]( t , iCur[attr] , ( json[attr] - iCur[attr] ) , date );

**if**( attr === 'opacity' ){

obj.style.opacity = value/100;

obj.style.fliter = 'alpha(opacity='+value+')'

}**else**{

obj.style[attr] = value + 'px';

}

};

**if**( t === date){

fn && fn.call( obj );

**return**;

};

requestAnimationFrame( animation );

};

}

t 其实就是代表的当前时间段。 比如，你规定一个动画4s执行完， 动画刚开始t就为0 ，结束的时候为4 ，中间是逐步增大的。

下面案例地址：  
[dome](https://baiduweb.github.io/nuomi/animate.html)

还有好多功能没实现， 比如stop()啊， fadeIn()啊之类的，还有很多bug。不过大体思路是这样 ， 剩下的优化慢慢做。

有不足的大家多提意见。

# 当我们讨论动画时，我们在讨论什么——transform篇

基本完成了IFE的[动画库任务(一)](http://ife.baidu.com/course/detail/id/52),内心是崩溃的。JS补间动画的原理的确很简单，但在实现过程中，才意识到以前用的成熟的动画库，做了多少的工作。

## JS动画实现原理

对于多数成熟的动画库，你只需要提供动画的末状态 end 与动画时长 duration，它就能帮你绘制一段缓动动画，其原理很是单纯，大体可以拆成以下几步：

1. 根据你提供的末状态 end，获得元素对应的初状态 begin
2. 通过初末状态，获得状态的变化量 change
3. 定义一个变量来保存动画进行到的时间，如currTime
4. 将 begin，change，currTime和动画时长传入缓动函数，计算出当前时间的状态，并进行设置
5. 增加currTime，如果大于duration表明这段动画执行完毕，直接将位置定位到end；否则增加currTime，回到步骤4

但就是这简单的5步，在实现的过程中有无数的坑啊......。

**PS: 缓动函数的用法，无耻的贴上**[**如何使用Tween.js各类原生动画运动缓动算法**](http://www.zhangxinxu.com/wordpress/2016/12/how-use-tween-js-animation-easing/)**，这次的重点不是这个。**

## 如何从传入的属性值获得必要的数据

为了便于使用，传入的属性值是简单的CSS属性，如：

{

width: "12px",

height: "12px"

}

为了将其作为末状态，我们需要提取出

1. 需要改变的属性名
2. 对应属性值的数值，因为要计算中间值
3. 对应属性值的单位

我们可以用正则表达式来匹配12px中的px，将匹配结果保存即可得到单位，再将匹配到的px换成''，即可得到数值的字符串形式。这里要对transform系列的属性值做特别处理，由于transform的属性值是rotate(30deg)这种形式，所以无法直接处理，于是参考其他动画库，对于transform的属性值特别传入，如：

{

width: "12px",

height: "12px",

rotateZ: "30deg",

translateX: "20px"

}

然后对其的单位提取步骤便可以统一。

## 如何获得初始状态

初始状态的获得主要依靠getComputedStyle(el, null).getPropertyValue(propertyName)方法，由于要传入属性名，所以肯定要依据添加的动画末状态来找到对应的属性名。这里依然是要特别处理transform系列的属性值，可以用一个正则来匹配末状态传入的属性名，如果是rotateZ这样的，就将transform作为参数传入上面的方法，来获得初始状态的属性...................？结果是返回了matrix(a,b,c,d,e,f)这样的属性值，这其中缘由可见[理解CSS3 transform中的Matrix(矩阵)](http://www.zhangxinxu.com/wordpress/2012/06/css3-transform-matrix-%E7%9F%A9%E9%98%B5/),总之我们可以方便的从rotate值计算出matrix值，但是想要反过来（主要是获得rotate值，涉及到三角函数）却很不精确，暂时不考虑这种方法。事实上我们希望能像获取元素的style属性一样，直接得到rotate属性值，但对于设置在CSS样式表里的属性，的确无法直接得到。so目前的一个很坑的地方——对于一个在样式表里设置了transform属性的元素，在第一次进行动画时，无法获得其初状态......这个问题在当前的velocityJs中也仍然存在，会直接从0的值开始动画，这里的解决方法暂时还没有寻得。但至少，之后的动画我们还是有办法掌控的。

## 如何在计算完成后，设置元素的状态

非transform系的没什么好讲的，直接style = 数值 + 单位即可。但是由于transform的属性值很有可能不止一个，如果直接给style赋新值，将会覆盖所有transform属性。解决方法：定义一个transformCache对象，如果有transform属性需要设置，就在transformCache查找是否存在该属性：不存在便定义该属性与对应属性值，如果存在就更新属性值；然后遍历所有的属性，拼接出完整的transform属性。

由于transform有关的很多地方都要特殊处理，所以设计模式的运用得当在该情形下尤为重要，我自己写的似乎就有些混乱了......而且初始状态的transform获取还没有解决，待更新......

赞1

## 网页抓取分析服务系列之三（服务封装）

### 实现思路

1. 用nodejs的HTTP模块来创建一个服务，响应get请求:
2. **var** http = require("http");
3. http.createServer(**function**(request, response) {
4. console.log('request received');
5. response.writeHead(200, {"Content-Type": "text/plain"});
6. response.write("Hello World");
7. response.end();
8. }).listen(8000);
9. console.log('server started');

request.url会得到路径名开始的url，比如访问http://localhost:8000/?word=%E4%B8%AD%E5%9B%BD&device=iphone5，将会得到/?word=%E4%B8%AD%E5%9B%BD&device=iphone5。可以用URL模块来解析url。

1. 使用URL模块的url.parse()方法来解析url字符串，会得到一个urlObject，取出其中的query属性，默认得到的query是个字符串，需要将url.parse()的第二个参数设置为true，query将被解析为一个对象。访问上面的地址时就会得到如下 ：
2. var queryObj = url.parse(req.url,true).query;
3. console.log(queryObj); *// {word: '中国', device: 'iphone5'}*
4. 使用child\_process模块的child\_process.exec()方法来调用命令行。传给回调的 stdout 和 stderr 参数会包含子进程的 stdout 和 stderr 的输出。
5. var exec = require('child\_process').exec;
6. var cmdStr = 'phantomjs task.js ';
7. exec(cmdStr + queryObj.word + ' ' + queryObj.device, **function**(err, stdout, stderr){
8. **if**(err) {
9. console.error(`exec error: ${error}`);
10. } **else** {
11. *// todo*
12. }
13. 运行MongoDB服务器，从MongoDB目录的bin目录中执行mongod.exe文件,mongod.exe --dbpath c:datadb，其中c:datadb为mongodb的数据目录，需提前手动创建。
14. 使用MongoDB后台管理 Shell，从MongoDB目录的bin目录中执行mongo.exe文件,在命令行输入mongo即可。几个命令：
15. > db
16. **test**
17. > show dbs
18. admin 0.000GB
19. baidu 0.000GB
20. local 0.000GB
21. **test** 0.000GB
22. > use baidu
23. switched to db baidu
24. > db
25. baidu
26. > show collections
27. results
28. > db.results.find({}) //查找所有文档
29. > db.results.remove({}) //删除所有文档
30. 使用mongoose来操作mongodb。  
    ```  
    // 连接数据库  
    var mongoose = require('mongoose');  
    mongoose.connect('mongodb://localhost/baidu');

// 添加数据库连接失败和打开时的回调  
var db = mongoose.connection;  
db.on('error', console.error.bind(console, 'connection error:'));  
db.once('open', function (callback) {  
console.log('mongoose connected!')  
});

// 定义一个Schema  
var resultSchema = new mongoose.Schema({  
code: Number,  
msg: String,  
word: String,  
device: String,  
time: Number,  
dataList: [{  
info: String,  
link: String,  
pic: String,  
title: String  
}]  
});

// 编译定义好的Schema  
var Result = mongoose.model('Result', resultSchema);

// 新建一个文档  
var result = new Result(JSON.parse(stdout));

// 将文档保存到数据库  
result.save(function(err, result) {  
if (err) {  
console.error(err);  
} else {  
console.log(result);  
}  
});

### 注意点

1. 浏览器会默认请求favicon.ico，可以将处理逻辑放到`**if**(req.url !== '/favicon.ico'){}`里面。

2. task.js可能会输出无法转换成JSON的字符串，需要做一下处理。比如，当 device的 参数输入有误时，stdout会输出`'Usage: the device should be one of "iphone5,iphone6,ipad"'`。在返回JSON前加上判断：

**try** {

JSON.parse(stdout);

} **catch** (err) {

res.writeHead(200, {'Content-Type': 'application/json'});

return res.end(JSON.stringify({code: 0, msg: '请确认查询参数是否正确'}));

}

```