Web开发实战

TG 著

集合了大量开发案例，直观易懂

包含CSS实战篇、JavaScript实战篇、Canvas实战篇和移动实战篇

目　录

概述

概述

《Web开发实战》是作者的第二本技术书籍，集合了大量的前端开发案例，目前主要选择日常开发中会用到的加入本书，分为四部分：CSS实战篇、JavaScript实战篇、Canvas实战篇和移动实战篇。

第一本书的地址：JavaScript半知半解

在《Web开发实战》一书中，每一节都会添加上详细的代码和讲解，也会提供Demo代码下载。

为了苦逼的生活，望谅解本书收费！

已完成的功能如下：

1、CSS实战篇

- 白光划过效果
- 水中倒影
- 按钮
- 工具提示

- 单选复选Switch
- Loading加载
- 图片滤镜
- 阴影
- 自定义滚动条
- 3D立方体
- 溢出省略号
- 自定义选择文本样式
- 心跳灯和呼吸灯
- 文本效果
- 进度条
- 遮罩条
- 切角
- 表单
- 动态的边框
- 导航菜单
- 文件上传组件美化
- 竖着排的文字
- 多形状图像
- 面包屑导航
- 多彩的渐变

2、JavaScript实战篇

- 点击水波纹效果
- 手风琴布局
- 滑块
- 下拉菜单
- 全屏滚动
- 富文本编辑器
- 带表情输入的评论框
- 图片懒加载
- 开启全屏
- 返回顶部
- 选项卡

- 上传图片预览
- 走马灯
- 万年历
- 树形菜单
- 旋转加载
- 瀑布流
- 圆形水波纹加载进度
- 检测是否移动端
- 搜索过滤
- 弹幕
- 收缩菜单
- 幻灯片
- 固定头的表格
- 自定义滚动条
- 城市联动选择器
- 滚动监听
- 边栏悬浮菜单

3、Canvas实战篇

- 粒子动画
- 刮刮卡
- 截图下载
- 图片放大器
- 手势密码
- 雪花纷飞
- 粒子聚合

4、移动实战篇

- 移动联动选择器
- 列表滑动删除

也许你浏览完上面的目录，会发现上面的各大功能在网上你都可以搜到相应的插件，那为什么还要写这本书呢？

网上的插件虽多，但大多数并没有深究到原理，故而产生了写一本前端实例的书籍。主要是

为了想深入学习的伙伴们，书上的内容也许不是最优秀的，但会一步步讲解，会告诉你每一步的原理，让你学习后也可以自己造轮子！

在完成大部分功能后，会利用这些功能开发出一套完整的模板出来。

模板预览

此书永久更新，会不断地添加新的功能！

由于作者的水平有限，书中内容有限，也难免会出现一些错误或者不准确的地方，恳求读者批评指正。为此，我特意创建了一个在线支持站点（书籍评论站点）。

CSS实战篇

CSS实战篇

CSS实战篇主要讲解使用CSS就可以实现的功能。

其中包括：

- 白光划过效果
- 水中倒影
- 按钮
- 工具提示
- 单选复选Switch
- Loading加载
- 图片滤镜
- 阴影
- 自定义滚动条
- 3D立方体
- 溢出省略号
- 自定义选择文本样式
- 心跳灯和呼吸灯
- 文本效果
- 进度条
- 遮罩条
- 切角
- 表单
- 动态的边框
- 导航菜单
- 文件上传组件美化
- 竖着排的文字
- 多形状图像
- 面包屑导航
- 多彩的渐变

白光划过效果

白光划过效果

这一节来介绍白光划过效果！

当鼠标移动到图片上时，有一道貌似白光的东东从图片上划过去。

1、创建模板

我们要放置一张图片，用一个div包裹起来：

```
<div class="highlight-box">
        <img src="images/example.jpg" alt="" />
</div>
```

2、设置CSS样式

定义初始样式（高宽可调整）：

```
.highlight-box {
        width: 300px;
        height: 120px;
        overflow: hidden;
        position: relative;
}

.highlight-box img {
        width: 100%;
        height: 100%;
}
```

注意：包裹图片的div（也就是类名为`.higlight-box`的div）一定要加上定位属性position，不加的结果是神马呢？你试试吧。

静态图片已经有了，接着让我们来制作白光，我们不需多余的元素，只需使用`:before`选择器：

```
.highlight-box:before {
        display: block;
```

```
        /*注意这里top和left，让白光移动到图片左上角，
        后续的划过动画也是依靠这两个属性*/
        top: -200%;
        left: -100%;
        /*定义白光的宽高*/
        width: 50%;
        height: 300%;
        /*旋转角度，你也可以调整*/
        -webkit-transform: rotate(45deg);
        transform: rotate(45deg);
        /*使用渐变来实现白光*/
        background: -webkit-linear-gradient(left, rgba(255, 255, 255, .05
) 20%,
            rgba(255, 255, 255, .6) 65%, rgba(255, 255, 255, .05) 100%);
        background: linear-gradient(left, rgba(255, 255, 255, .05) 20%,
            rgba(255, 255, 255, .6) 65%, rgba(255, 255, 255, .05) 100%);
        content: '';
        z-index: 2;
        position: absolute;
}
```

我们使用渐变（linear-gradient）来实现白光效果，同时为了斜向划过，使用`transform: rotate(45deg)`将其旋转45度。

上面的height、width、top和left，你也可以使用具体的像素值，不过建议采用百分比，这样可以重复使用，而不需手动改变太多值。

图片有了，白光有了，接下来就是让白光动起来：

```
.highlight-box:hover:before {
    //这里省略了私有前缀代码
        animation: crossed .5s linear;
}

@keyframes crossed {
        0% {
                top: -200%;
                left: -100%;
        }
        100% {
                top: -50px;
                left: 100%;
        }
}
```

我们需要改变的只是top和left的值，也就是让白光从左上角向右下角移动。

到这里，白光划过效果已经实现了。

源码路径： `WebDemo/CSS/highlight`

源码下载地址在目录的最后。

3D立方体

3D立方体

随着CSS3的出现，实现3D效果已经不是难事，这一节就来看看3D立方体是如何实现的。

1、创建模板

首先来放置一个父div `.cude`，然后在其里面放置6个div，分别表示立方体的6个面。

```
<div class="cude">
        <div class="front surface">
                正面
        </div>
        <div class="surface left">
                左面
        </div>
        <div class="surface right">
                右面
        </div>
        <div class="surface bottom">
                底面
        </div>
        <div class="surface top">
                顶面
        </div>
        <div class="surface back">
                背面
        </div>
</div>
```

2、设置CSS样式

```
.cude {
        width:300px;
        height:300px;
        position:relative;
        margin:100px auto;
        transform-style:preserve-3d;
        -webkit-transform-style:preserve-3d;
}
```

`transform-style` 属性规定如何在 3D 空间中呈现被嵌套的元素，默认是其子元素不呈现3D效果，所以我们需要添加 `preserve-3d` 值，让其子元素保留其 3D 效果。

接下来，我们制作立方体的每一个面：

```
.surface {
        position:absolute;
        top:0;
        left:0;
        width:300px;
        height:300px;
        background:#666;
        opacity:0.8;
        font-size:60px;
        text-align:center;
        line-height:300px;
        font-weight:bold;
        color:#fff;
        border:1px solid #fff;
        -webkit-transition:all .3s;
        transition:all .3s;
}
.surface img {
        width:100%;
}
.front {
        transform:rotateY(0) translateZ(150px);
}
.back {
        transform:translateZ(-150px) rotateY(180deg);
}
.left {
        transform:rotateY(-90deg) translateZ(150px);
}
.right {
        transform:rotateY(90deg) translateZ(150px);
}
.top {
        transform:rotateX(90deg) translateZ(150px);
}
.bottom {
        transform:rotateX(90deg) translateZ(-150px);
}
```

在上面的代码中，我们使用`transform`的`rotate()`和`translateZ()`来转换每一面，平移都是宽度的一半。

要注意`rotate()`和`translateZ()`的顺序，顺序不同，转换的结果也会不一样，比如左侧.left这一面，在这里，我们是先绕着Y轴顺时针旋转90，然后再在Z轴的正方向平移150，如果你先平移后旋转，结果就不一样了，你可以试试。

最后，我们还给`div.cude`加上动画：

```
@-webkit-keyframes rotate {
    from {
        transform: rotateX(0deg) rotateY(0deg);
    }
    to {
        transform: rotateX(360deg) rotateY(360deg);
    }
}
```

源码路径：`WebDemo/CSS/cude`

水中倒影

水中倒影

先来看看何为水中倒影：

要实现倒影，有两种方式：`box-reflect方式`和`transform方式`。

1、box-reflect方式

1.1 属性介绍

要实现倒影，最简单的就是使用`box-reflect`属性。

box-reflect属性简单介绍：

```
box-reflect：<direction> <offset>? <mask-box-image>?
```

参数说明：

- 表示倒影的方向，可能值：above、below、left、right（上下左右）；
- 表示倒影与元素之间的间隔；
- <mask-bo-image>表示遮罩图像，可为url地址、渐变

1.2 创建模板

只需简单的放置一张图片：

```
<img src="images/example.jpg" alt="" />
```

1.3 设置样式

在上面我们已经介绍过`box-reflect`的使用方法：

```
img {
    -webkit-box-reflect: below 0 -webkit-linear-gradient(top, rgba(250, 2
50, 250, 0), rgba(250, 250, 250, .0) 20%, rgba(250, 250, 250, .3));
}
```

在这里，我们使用渐变来充当遮罩图像。

不过目前只有webkit引擎的浏览器才支持。

2、transform方式

这种方式其实就是复制图片，然后翻转，最后同样是使用渐变来充当遮罩层，覆盖在翻转的图片之上。

2.1 创建模板

在这里，我们用一个div包裹住图片：

```
<div class="box-reflect">
        <img src="images/example.jpg" alt="" />
</div>
```

2.2 设置样式

我们需要将图片克隆一份，然后翻转过来：

```
.box-reflect {
    position: relative;
}
.box, .box-reflect {
    width: 150px;
    float: left;
    margin-right: 40px;
}
.box-reflect img {
    width: 100%;
    height: 100%;
}
.box-reflect:before {
    background: url(images/example.jpg) no-repeat;
    background-size: 100% 100%;
    -webkit-transform: scaleY(-1);
    -moz-transform: scaleY(-1);
    -ms-transform: scaleY(-1);
    -o-transform: scaleY(-1);
    transform: scaleY(-1);
    /*添加半透明是为了更真实*/
    opacity: 0.5;
    filter: alpha(opacity='50');
}
.box-reflect:before, .box-reflect:after {
    position: absolute;
    width: 100%;
```

```
    height: 100%;
    top: 100%;
    left: 0;
    content: "";
}
```

注意：这里的`transform:scaleY(-1)`相当于`transform:rotateX(-180deg)`，可联想一下`transform:scaleX(-1)`的效果。

然后，添加一层渐变，让其覆盖在倒影的上面：

```
.box-reflect:after {
    background-image: -moz-linear-gradient(bottom, rgb(0,0,0) 20%, rgba(0
,0,0,0) 90%);
    background-image: -o-linear-gradient(rgba(0,0,0,0) 10%, rgb(0,0,0) 30
%);
    background-image: -webkit-linear-gradient(bottom,rgb(0,0,0) 20%,rgba(
0,0,0,0) 90%);
    filter: progid:DXImageTransform.Microsoft.Gradient(gradientType=0,sta
rtColor=0, EndColorStr=#000000);
}
```

渐变的值你可以改变，最后的filter是为了兼容IE。

源码路径：`WebDemo/CSS/boxReflect`

工具提示（tooltip）

工具提示

最简单的提示就是使用`title`属性，不过它的样式比较单一，如果要实现下面的工具提示，我们怎么做呢？

用JS？这当然是可以的，不过，有强大的CSS3，使用JS就显得有点多余了。

属性介绍

```
content: attr(attribute)
```

上面的意思是插入标签属性值。

1、创建模板

先放置一个按钮，留意里面的两个`data-*`自定义属性，这是关键。

```
<button class="btn btn-primary tooltip"
    data-tooltip="在下方" data-direction="down">
    在下方
</button>
```

2、设置样式

看看`:after`选择器里面的`content: attr(data-tooltip)`，是不是和上面按钮的`data-tooltip`属性名称一致。

```
.tooltip
{
    position: relative;
}
.tooltip:before
{
    position: absolute;
    content: '';
    border: 5px solid transparent;
}
.tooltip:after
{
```

```
    font-size: 14px;
    line-height: normal;
    position: absolute;
    padding: 5px 10px;
    // 看这里！看这里！
    content: attr(data-tooltip);
    white-space: nowrap;
    color: #fff;
    border-radius: 3px;
    background: #383838;
}
```

注意：attr()里面的属性名称上不要加引号！

添加`:hover`让其显示：

```
.tooltip:before,
.tooltip:after
{
    z-index: 1000000;
    visibility: hidden;
    -webkit-transition: .3s ease;
       -moz-transition: .3s ease;
            transition: .3s ease;
    -webkit-transition-delay: 0ms;
       -moz-transition-delay: 0ms;
            transition-delay: 0ms;
    pointer-events: none;
    opacity: 0;
}
.tooltip:hover:before,
.tooltip:hover:after
{
    visibility: visible;
    opacity: 1;
}
```

添加`pointer-events: none;`属性是为了不让提示也被`hover`。

为了更好看，我们还可以添加提示动画：

```
[data-direction='down']:before
{
    top: -webkit-calc(100% - 5px);
    top:    -moz-calc(100% - 5px);
    top:         calc(100% - 5px);

    border-bottom-color: #383838;
}
[data-direction='down']:after
```

```
{
    top: -webkit-calc(100% + 5px);
    top:    -moz-calc(100% + 5px);
    top:         calc(100% + 5px);
}
[data-direction='down']:before,
[data-direction='down']:after
{
    left: 50%;
    -webkit-transform: translate3d(-50%,0,0);
    -moz-transform: translate3d(-50%,0,0);
    transform: translate3d(-50%,0,0);
}
```

在上面，我们用到了属性选择器，这是为了定义提示在不同位置，你可以看看完整源码，目前我已经提供了四个方向。

源码路径：`WebDemo/CSS/tooltip`

单选复选

单选复选Switch

我们都知道，默认的单选复选框都很丑，而且在移动端还不统一，所以，美化单选复选框是有必要的。

我们使用CSS3就可以美化，而且不难，真的不难！

1、单选

1.1 创建模板

简单的布局：

```
<label for="radio3" class="tg-icheck-radio tg-icheck-flat-radio">
        <input type="radio" id="radio3" name="sex1" checked>
        <div class="tg-icheck-media"></div>
        <div class="tg-icheck-inner">
                <div class="tg-icheck-title">男</div>
        </div>
</label>
```

上面的有两点要注意：

- `label` 和 `input` 的关系，通过 `label` 的 `for` 属性和 `input` 的 `id` 属性将两者绑定在一起，虽然将input放置在label里会默认绑定（隐式绑定），不过还是建议加上id。
- 你的自定义UI（也就是上面的 `tg-icheck-media` ），一定要放到input后面，最好紧随其后

当然，布局是多样化的，你也可以自由设计。

设置CSS样式

如何响应单选框是否选中而改变UI样式，这才是美化的重点。

既然要自定义，当然得先把原始input隐藏起来，用opacity？当然不是，因为它还占据空间，我们这样隐藏：

```
.tg-icheck-radio > input
{
    display: none;
}
```

估计你会想，完全隐藏了，如何选择表单呢？这就是前面提到的注意点，`label` 要绑定 `input`。

`<label>` 标签为 input 元素定义标注（标记）。它的作用是，当用户选择该标签时，浏览器就会自动将焦点转到和标签相关的表单控件上。

接下来，我们通过`:checked`伪类选择器（匹配处于选中状态的元素）和关系选择器`+`（选择紧随其后的兄弟元素）。

```
 input:checked + .tg-icheck-media
{
    border-color: #009a61;
    background:#fff;
}
 input:checked + .tg-icheck-media:after
{
   -webkit-transform:scale(1);
   -moz-transform:scale(1);
   transform: scale(1);
}
```

到这里，美化单选框已经完成了。

复选框和Switch组件的原理是一样，你可以先动手试试，如果搞不定，可以参考我给出的例子。

复选

```
<label  for="checkbox4" class="checkbox checkbox-inline">
             <input type="checkbox" value="option1"  id="checkbox4>
             <i class="input-helper"></i>
             checkbox
 </label>
```

Switch

```
<label class="tg-switch tg-switch-3" for="switch1">
        <input type="checkbox" id="switch1">
```

```
        <div class="tg-switch-media">
                <span class="switch-label"></span>
        </div>
        <div class="tg-switch-inner">switch1</div>
</label>
```

源码路径：`WebDemo/CSS/input`

Loading

Loading加载

一个页面总少不了页面预加载Loading特效，特别是在提交数据更需要。

我们以示例图中的第一个loading来讲解！

1、创建模板

我们需要一个div，里面放置子元素。

```
<div class="loader circle-line small">
        <span></span>
        <span></span>
        <span></span>
        <span></span>
        <span></span>
        <span></span>
        <span></span>
        <span></span>
</div>
```

2、设置CSS样式

先来看看静态的loading是怎么产生的，效果如下：

其实，是通过转换不同span之间的位置和角度来实现：

```
.loader {
        position: relative;
        width: 5rem;
        height: 5rem;
}
.loader.circle-line span {
        position: absolute;
        display: inline-block;
        width: 1.5rem;
        height: .5rem;
        border-top-left-radius: .25rem;
        border-bottom-left-radius: .25rem;
        background: #1ABC9C;
```

```
}
.loader.circle-line span:nth-child(1) {
        top: 50%;
        left: 0;
        margin-top: -.25rem;
}

.loader.circle-line span:nth-child(2) {
        top: 1rem;
        left: .5rem;
        -webkit-transform: rotate(45deg);
        transform: rotate(45deg);
}

.loader.circle-line span:nth-child(3) {
        left: 50%;
        top: .5rem;
        margin-left: -.75rem;
        -webkit-transform: rotate(90deg);
        transform: rotate(90deg);
}

.loader.circle-line span:nth-child(4) {
        right: .5rem;
        top: 1rem;
        -webkit-transform: rotate(145deg);
        transform: rotate(145deg);
}

.loader.circle-line span:nth-child(5) {
        left: 3.5rem;
        top: 50%;
        margin-top: -.25rem;
        -webkit-transform: rotate(180deg);
        transform: rotate(180deg);
}

.loader.circle-line span:nth-child(6) {
        bottom: 1rem;
        right: .5rem;
        -webkit-transform: rotate(-145deg);
        transform: rotate(-145deg);
}

.loader.circle-line span:nth-child(7) {
        left: 50%;
        bottom: .5rem;
        margin-left: -15px;
        -webkit-transform: rotate(-90deg);
        transform: rotate(-90deg);
}

.loader.circle-line span:nth-child(8) {
        bottom: 1rem;
        left: .5rem;
        -webkit-transform: rotate(-45deg);
```

```
        transform: rotate(-45deg);
}
```

当然，这些位置数据都是需要耐心的去调整。

静态的loading已经有了，现在我们需要添加动画让其真实地动起来：

创建一个名为circle-line的动画（改变透明度）：

```
@keyframes circle-line {
        0% {
                opacity: .05;
        }
        100% {
                opacity: .7;
        }
}

@-webkit-keyframes circle-line {
        0% {
                opacity: .05;
        }
        100% {
                opacity: .7;
        }
}
```

然后给每一个线条添加circle-line动画：

```
.loader.circle-line span {
        position: absolute;
        display: inline-block;
        width: 1.5rem;
        height: .5rem;
        border-top-left-radius: .25rem;
        border-bottom-left-radius: .25rem;
        background: #1ABC9C;
        opacity: .05;
        -webkit-animation: circle-line 1s ease infinite;
        animation: circle-line 1s ease infinite;
}
```

要实现不同线条的间隔动画，我们只需添加 `animation-delay` 延迟时间：

```
.loader.circle-line span:nth-child(1) {
        top: 50%;
        left: 0;
        margin-top: -.25rem;
```

```
        -webkit-animation-delay: .13s;
        animation-delay: .13s;
}

.loader.circle-line span:nth-child(2) {
        top: 1rem;
        left: .5rem;
        -webkit-transform: rotate(45deg);
        transform: rotate(45deg);
        -webkit-animation-delay: .26s;
        animation-delay: .26s;
}

.loader.circle-line span:nth-child(3) {
        left: 50%;
        top: .5rem;
        margin-left: -.75rem;
        -webkit-transform: rotate(90deg);
        transform: rotate(90deg);
        -webkit-animation-delay: .39s;
        animation-delay: .39s;
}

.loader.circle-line span:nth-child(4) {
        right: .5rem;
        top: 1rem;
        -webkit-transform: rotate(145deg);
        transform: rotate(145deg);
        -webkit-animation-delay: .52s;
        animation-delay: .52s;
}

.loader.circle-line span:nth-child(5) {
        left: 3.5rem;
        top: 50%;
        margin-top: -.25rem;
        -webkit-transform: rotate(180deg);
        transform: rotate(180deg);
        -webkit-animation-delay: .65s;
        animation-delay: .65s;
}

.loader.circle-line span:nth-child(6) {
        bottom: 1rem;
        right: .5rem;
        -webkit-transform: rotate(-145deg);
        transform: rotate(-145deg);
        -webkit-animation-delay: .78s;
        animation-delay: .78s;
}

.loader.circle-line span:nth-child(7) {
        left: 50%;
        bottom: .5rem;
        margin-left: -15px;
        -webkit-transform: rotate(-90deg);
```

```
        transform: rotate(-90deg);
        -webkit-animation-delay: .91s;
        animation-delay: .91s;
}

.loader.circle-line span:nth-child(8) {
        bottom: 1rem;
        left: .5rem;
        -webkit-transform: rotate(-45deg);
        transform: rotate(-45deg);
        -webkit-animation-delay: 1.04s;
        animation-delay: 1.04s;
}
```

注意时间的把控，在上面，`circle-line`动画的周期时间正好是最后一个span的延迟时间。

其他加载Loading也是同样的原理，先是定义好静态布局，然后实现一个动画（透明度变化、高度变化、大小变化等），最后将动画添加给给每个子项，再添加动画延迟时间即可。

你可以动手试试，如果实现不了，再参考我给出的例子！

源码路径：`WebDemo/CSS/loading`

图片滤镜

图片滤镜

看看下面的示例图，这些效果就像是photoshop整出来的一样，其实就是如此，有很多效果都是类似于photoshop中的特效

这一节就让我们一起来了解一下。

属性说明：

```
  filter: none | <filter-function > [ <filter-function> ]
```

`filter`属性就是专门用来实现滤镜效果。

`filter-function`具有以下值：

- grayscale灰度
- sepia褐色
- saturate饱和度
- hue-rotate色相旋转
- invert反色
- opacity透明度
- brightness亮度
- contrast对比度
- blur模糊
- drop-shadow阴影

注意：可以同时添加一个以空格分隔的列表。

1、创建模板

只需一个`<img>`图片：

```
<img src="../../common/Images/girl.jpg" class="blur" alt="">
```

2、设置CSS样式

实现滤镜效果很简单，只需添加一个属性`filter`：

```
// 高斯模糊
.blur {
        -webkit-filter:blur(2px);
        filter:blur(2px);
}
// 灰度
.grayscale {
        -webkit-filter:grayscale(100%);
        filter:grayscale(100%);
}
// 褐色
.sepia {
        -webkit-filter:sepia(100%);
        filter:sepia(100%);
}
// 饱和度
.saturate {
        -webkit-filter:saturate(5);
        filter:saturate(5)
}
// 色相旋转
.hue-rotate {
        -webkit-filter:hue-rotate(180deg);
        filter:hue-rotate(180deg)
}
// 反色
.invert {
        -webkit-filter:invert(1);
        filter:invert(1);
}
// 透明度
.opacity {
        -webkit-filter:opacity(.5);
        filter:opacity(.5)
}
// 亮度
.brightness {
        -webkit-filter:brightness(3);
        filter:brightness(3)
}
// 对比度
.contrast {
        -webkit-filter:contrast(.5);
        filter:contrast(.5)
}
// 阴影
.drop-shadow {
        -webkit-filter:drop-shadow(10px 10px 5px #000);
        filter:drop-shadow(10px 10px 5px #000);
}
```

源码路径：`WebDemo/CSS/filter`

阴影（box-shadow）

阴影

属性说明

要实现实现阴影，我们先来了解一下box-shadow属性：

```
box-shadow:[inset] x-offset y-offset blur-radius spread-radius color;
```

参数说明：

- `阴影类型`：可选；如省略，默认是外阴影；它有且只有一个值“inset”，表示为内阴影；
- `x-offset`：阴影水平偏移量，它可以是正负值。如为正值，则阴影在元素的右边；如其值为负值，则阴影在元素的左边；
- `y-offset`：阴影垂直偏移量，它可以是正负值。如为正值，则阴影在元素的底部；如其值为负值时，则阴影在元素的顶部；
- `blur-radius`：阴影模糊半径，可选，它只能是正值。如值为0，则阴影不具有模糊效果；它的值越大，阴影的边缘就越模糊；
- `spread-radius`：阴影扩展半径，可选，它可以是正负值。如为正值，则扩大阴影的尺寸；如为负值，则缩小阴影的尺寸；（记住这个属性值，制作单边阴影的关键）
- `color`：阴影颜色，可选，如不设定颜色，浏览器会取默认色，但各浏览器默认取色不一致。（经测试，在Safari上是半透明的，在chrome、firefox、ie上都是黑色的）。不推荐省略颜色值。

注意：颜色（color）也可以放在最前面的，inset值也可以放在最后面；阴影并不会占据空间，也就是说它不会阴影布局。

还可以使用多阴影，每个阴影之间用“,”逗号隔开。

注意：设置多个阴影时，覆盖顺序是由前至后，定义越前面的阴影有越高的层级，会覆盖在后面定义的阴影之上

1、创建模板

只需一个div：

```
<div class="box shadow">四周阴影</div>
```

2、设置CSS样式

在上面，我们已经介绍过`box-shadow`属性的用法，下面来真正地实践一下：

```
// 四周阴影
.shadow {
        box-shadow:0 0 25px 0px rgba(0,0,0,0.22);
}
// 左上邻边阴影
.shadow-top-left {
        box-shadow:-10px -10px 25px 0px rgba(0,0,0,0.22);
}
// 右上邻边阴影
.shadow-top-right {
        box-shadow:10px -10px 25px 0px rgba(0,0,0,0.22);
}
// 右下邻边阴影
.shadow-bottom-right {
        box-shadow:10px 10px 25px 0px rgba(0,0,0,0.22);
}
// 左下邻边阴影
.shadow-bottom-left {
        box-shadow:-10px 10px 25px 0px rgba(0,0,0,0.22);
}
// 顶部单边阴影
.shadow-top {
        box-shadow:0 -15px 10px -10px rgba(0,0,0,.22);
}
// 右侧单边阴影
.shadow-right {
        box-shadow:15px 0 10px -10px rgba(0,0,0,.22);
}
// 底部单边阴影
.shadow-bottom {
        box-shadow:0 15px 10px -10px rgba(0,0,0,.22);
}
// 左侧单边阴影
.shadow-left {
        box-shadow:-15px 0 10px -10px rgba(0,0,0,.22);
}
```

不知道你注意到没有，当实现单边阴影时，第四个数值都是负数，而且水平偏移量（第一个数值）或垂直偏移量（第二个数值）的绝对值都会比第四个数值的绝对值大。

源码路径：`WebDemo/CSS/boxShadow`

3D按钮

按钮

每个网站都或多或少的拥有按钮，而默认的按钮样式很丑，而且在移动端千奇百怪，所以美化按钮实在有必要。

这一节我们就来看看如何美化按钮且实现3D按钮。

1、普通按钮

1.1 创建模板

要在页面放置一个按钮，其实很简单，只需添加如下代码：

```
<button class="btn btn-default">default</button>
```

1.2 设置CSS样式

对于这类普通按钮美化，我们只需修改按钮的背景色、字体颜色、边框色即可，当然，我们还可以使用阴影来增加一些仿3D效果。

```
.btn-default {
    color: #333;
    background-color: #fff;
    border-color: #ccc;
}
.btn-default:focus {
  color: #333;
  background-color: #e6e6e6;
  border-color: #8c8c8c;
}
.btn-default:hover,
.btn-default:active {
  color: #333;
  background-color: #e6e6e6;
  border-color: #adadad;
}
.btn-default:active:hover,
.btn-default:active:focus {
  color: #333;
  background-color: #d4d4d4;
  border-color: #8c8c8c;
```

```
}
```

上面的按钮效果并没有太多可讲的，最主要的是耐心地去实现。

2、3D按钮

2.1 创建模板

同样，放置一个要设置成3D按钮的元素，这里使用a标签：

```
<a href="#" class="button">Button</a>
```

2.2 设置CSS样式

3D效果主要是依靠阴影（`box-shadow`）来实现，如果你不熟悉`box-shadow`，可翻看上一节或者网上搜索一下。

我们还有必要了解一下新的颜色属性：

```
HSL(H,S,L)
```

- Hue(色调)。0(或360)表示红色，120表示绿色，240表示蓝色，也可取其他数值来指定颜色。取值为：0 - 360
  S：
- Saturation(饱和度)。取值为：0.0% - 100.0%
- L：
  Lightness(亮度)。取值为：0.0% - 100.0%

要获取一个十六进制的颜色的hsl，我们只需使用开chrome浏览器随便打开一个页面，然后打控制台，将颜色赋值给任意的元素，然后点击颜色那个正方形（你会看到一个颜色选择器），然后点击那里的上下箭头，你就会看到不同类型的颜色值，如下图所示：

接下来，我们就可以来看看3D按钮的实现：

```
box-shadow:
    inset rgba(255, 254, 255, 0.6) 0 0.3em .3em,
    inset rgba(0, 0, 0, 0.15) 0 -0.1em .3em,
    hsl(0, 0%, 60%) 0 .1em 3px, hsl(0, 0%, 45%) 0 .3em 1px,
```

```
        rgba(0, 0, 0, 0.2) 0 .5em 5px;
```

在这里，我们使用了5层阴影，1和2层是内阴影，这是为了实现上下两边光滑效果，你也可以去除这两个；第5层只是普通的阴影；第3层是为了更真实，可有可无；关键是第4层，实现类似于border的实投影。

我们还需要实现点击效果：

```
.button:active {
        box-shadow: inset rgba(255, 255, 255, 0.6) 0 0.3em .3em, inset rg
ba(0, 0, 0, 0.2) 0 -0.1em .3em,
        rgba(0, 0, 0, 0.4) 0 .1em 1px,
        rgba(0, 0, 0, 0.2) 0 .2em 6px;
        transform: translateY(.2em);
}
```

对于点击效果，简单地说，就是让实阴影消失。

你还可能想知道上面的图中那些不同的形状是如何实现的？

其实，只是利用了`border-radius`而已，需要提一点的是：这个属性其实是可以这样设置的：

```
border-radius: 10px / 10px;
```

神马意思呢？指的是独立设置水平半径和垂直半径。

比如上面那个橙色按钮：

```
border-radius: .4em .4em 2em 2em / .4em .4em 3em 3em;
```

前面四个表示水平半径，后面四个表示垂直半径，顺序是左上角、右上角、右下角、左下角。

源码路径：`WebDemo/CSS/button`

自定义滚动条

自定义滚动条

浏览器自带的滚动条总是千篇一律，我们需要个性化。

仔细看下面每一条样式，都有相应的注释说明：

```
/*定义滚动条高宽及背景 高宽分别对应横竖滚动条的尺寸*/
::-webkit-scrollbar {
        width: 10px;
        height: 10px;
}
/*定义滚动条轨道（track）和滑块（thumb）*/
::-webkit-scrollbar-track,
::-webkit-scrollbar-thumb {
        border-right: 1px solid transparent;
        border-left: 1px solid transparent;
}

/*定义滚动条顶部（左侧）的按钮块*/
::-webkit-scrollbar-button:start {
        width: 10px;
        height: 12px;
        background: transparent url(../images/arrow.png) no-repeat 0 0;
}
/*定义滚动条底部（右侧）的按钮块*/
::-webkit-scrollbar-button:end {
        width: 10px;
        height: 12px;
        background: transparent url(../images/arrow.png) no-repeat -50px
0;
}

::-webkit-scrollbar-button:start:hover {
        background-color: #eee;
}

::-webkit-scrollbar-button:end {
        background-color: #eee;
}

::-webkit-scrollbar-thumb {
        -webkit-border-radius: 8px;
        border-radius: 8px;
        background-color: rgba(0, 0, 0, 0.2);
}
/*定义水平和垂直滚动条两者的交界处（拐角）*/
::-webkit-scrollbar-corner {
```

```
        display: block;
}

::-webkit-scrollbar-track:hover {
        background-color: rgba(0, 0, 0, 0.15);
}

::-webkit-scrollbar-thumb:hover {
        -webkit-border-radius: 8px;
        border-radius: 8px;
        background-color: rgba(0, 0, 0, 0.5);
}
```

注意：目前只有webkit内核的浏览器（比如chrome，Safari）支持。如果要实现兼容的自定义滚动条，可参考《JavaScript实战篇：自定义滚动框》。

源码路径：`WebDemo/CSS/scrollbar`

文字效果

文字效果

看看下面的文字，是不是很酷炫呢！

如果你想知道如何实现，阅读这一节绝对不会错。

1、属性介绍

在讲代码之前，先来了解四个CSS属性：

- text-shadow

text-shadow定义文本阴影效果：

```
text-shadow: h-shadow v-shadow blur color;
```

h-shadow： 必需。水平阴影的位置。允许负值。

v-shadow： 必需。垂直阴影的位置。允许负值。

blur：可选。模糊的距离。

color：可选。阴影的颜色。如果省略，则使用当前字体颜色。

注意：`text-shadow` 属性向文本添加一个或多个阴影。该属性是逗号分隔的阴影列表，每个阴影有两个或三个长度值和一个可选的颜色值进行规定。省略的长度是 0，而且它并没有像 `box-shadow` 那样的第四个数值。

- background-clip

`background-clip` 可以让图片（或渐变色）填充文本

```
background-clip:text
```

`background-clip` 需配合其私有属性 `-webkit-text-fill-color: transparent`

。

- text-fill-color

  ```
  text-fill-color：<color>
  ```

  text-fill-color会覆盖color所定义的字体颜色。

- mask-image

```
-webkit-mask-image: url() | gradient;
```

mask-image实现图片或渐变遮罩效果，它从图片遮罩里读出图片的透明信息，然后应用到html元素上。

虽然mask-image的支持度不是灰常高，但你可以大胆使用，因为对于文本来说，本身只是装饰用的，如果不支持，只会显示纯色，并不会影响什么。

1、简单的文字投影

示例图的第一个文字。

```
.textShadow {
        color:#fff;
        text-shadow:0.1em 0.1em 0 rgba(0,0,0,0.5);
}
```

2、镂空文字

只需添加如下样式：

```
.fill {
        font-size:3em;
        -webkit-text-fill-color:transparent;
        -webkit-text-stroke:1px #fff;
}
```

如果不明白，就回滚到上面看看属性介绍。

3、文字外发光、3D文字

这两种文字效果，都是通过添加多阴影来实现的。

```
/*文字外发光*/
.highlight {
        color:#ffc;
        text-shadow:0 0 .2em #fff,
                0 0 .3em #fff;
}
/*3D文字*/
.extruded {
        color:white;
        text-shadow:0 1px hsl(0,0%,85%),
                0 2px hsl(0,0%,80%),
                0 3px hsl(0,0%,75%),0 4px hsl(0,0%,70%),
                0 5px hsl(0,0%,65%),
                0 5px 10px black;
}
```

4、自定义投影

自定义投影也不难，只需借助`mask-image`遮罩属性即可：

```
-webkit-mask-image:url(images/mask.png);
mask-image:url(images/mask.png);
```

完整代码：

```
.mask {
        font-size:3em;
}
.mask:after {
        content:attr(data-text);
        color:rgba(0,0,0,.35);
        text-shadow:none;
        position:absolute;
        left:.0875em;
        top:.0875em;
        z-index:-1;
        -webkit-mask-image:url(images/mask.png);
        mask-image:url(images/mask.png);
}
```

注意：使用mask-image时，需要加上前缀。

5、渐变文字

如果你只是为了实现渐变文字，也可以使用mask-image属性来设置，但如果你要实现动态

渐变，就需要使用背景渐变了。

先来说说如何让渐变填充文字：

```
-webkit-background-clip:text;
-webkit-text-fill-color:transparent;
```

这两行代码的意思是将字体颜色设置为透明，使用图片或渐变色来填充文本。

完整代码：

```
.bgText {
        font-size:5em;
        font-weight:bolder;
        background-color:#d32c46;
        background-image:-webkit-linear-gradient(-45deg,rgba(255,255,255,
.15) 25%,transparent 25%,transparent 50%,rgba(255,255,255,.15) 50%,rgba(2
55,255,255,.15) 75%,transparent 75%,transparent);
        background-image:linear-gradient(-45deg,rgba(255,255,255,.15) 25%
,transparent 25%,transparent 50%,rgba(255,255,255,.15) 50%,rgba(255,255,2
55,.15) 75%,transparent 75%,transparent);
        background-size:30px 30px;
        /*下面一行让背景色填充问自己*/
        -webkit-background-clip:text;
        /*将文字填充为透明色*/
        -webkit-text-fill-color:transparent;
        display:inline-block;
        /*动画*/
        -webkit-animation:tg-progress-bar-stripes 2s linear infinite;
        animation:tg-progress-bar-stripes 2s linear infinite;
}

@keyframes tg-progress-bar-stripes {
        0% {
          background-position:0 0;
    }
  100% {
        background-position:30px 0;
  }
}
```

源码路径：`WebDemo/CSS/textShadow`

多彩的渐变

条纹网格

CSS3中的渐变是无所不能的，为了方便后续功能的讲解，这里特意将渐变这一节提前到这里。

1、属性介绍

先来了解一下线性渐变 `linear-gradient` ：

```
linear-gradient([ <angle> | to <side-or-corner> ,]? <color-stop> [, <colo
r-stop>]+ )
```

- `<side-or-corner>`
  描述渐变线的起始点位置。它包含两个关键词：第一个指出垂直位置 `left or right` ，第二个指出水平位置 `top or bottom` 。关键词的先后顺序无影响，且都是可选的。`to top` , `to bottom` , `to left` 和 `to right` 这些值会被转换成角度 `0度、180度、270度和90度` 。其余值会被转换为一个以向顶部中央方向为起点顺时针旋转的角度。渐变线的结束点与其起点中心对称。
- `<angle>`
  用角度值指定渐变的方向（或角度），可负值。
- `<color-stop>`
  由一个 `<color>` 值组成，并且跟随着一个可选的终点位置（可选，可以是一个百分比值或者是沿着渐变轴的）。

看看下图，可以帮助你更好的理解渐变角度：

理解了线性渐变的原理后，你就可以随心所欲的使用酷炫效果：

我们看看示例图中的桌面网格的实现：

1、创建模板

```
<div class="box linearGradient">水平条纹</div>
```

2、设置CSS样式

```
.grid {
        background:white;
        background-image:linear-gradient(90deg,rgba(200,0,0,.5) 50%,trans
parent 0),
        linear-gradient( rgba(200,0,0,.5) 50%,transparent 0);
        background-size:30px 30px;
}
```

要记得背景在默认情况下是会平铺的。

源码路径：`WebDemo/CSS/linearGradient`

进度条

进度条

这一节我们来讲解使用纯CSS3美化进度条。

1、创建模板

```
<div class="tg-progress">
        <div class="tg-progress-bar" style="width:40%;
        ">40%</div>
</div>
```

在上面的代码：

- `tg-progress`表示一个进度条
- `tg-progress-bar`表示已填充部分。

这里的width表示的是已加载的长度

2、设置CSS样式

```
.tg-progress {
        box-sizing:border-box;
        height:20px;
        margin-bottom:15px;
        background:#f7f7f7;
        overflow:hidden;
        line-height:20px;
        box-shadow:inset 0 0 0 1px rgba(0,0,0,.07),inset 0 2px 2px rgba(0
```

```
,0,0,.07);
        border-radius:4px;
}
.tg-progress-bar {
        width:0;
        height:100%;
        background:#009dd8;
        float:left;
        -webkit-transition:width .6s ease;
        transition:width .6s ease;
        font-size:12px;
        color:#fff;
        text-align:center;
        background-image:-webkit-linear-gradient(top,#00b4f5,#008dc5);
        background-image:linear-gradient(to bottom,#00b4f5,#008dc5);
        box-shadow:inset 0 -1px 0 rgba(0,0,0,.2),inset 0 0 0 1px rgba(0,0
,0,.1);
        text-shadow:0 -1px 0 rgba(0,0,0,.2);
}
```

在上面的代码中，我们使用`bos-shadow`内阴影来产生凹槽效果，使用渐变来让填充部分有立体感。

现在来看看第二个，如何实现这种多条纹呢？

答案还是渐变（如果不了解渐变，可到多彩的渐变那一节去了解）。

布局并没有变化：

```
<div class="tg-progress tg-progress-striped">
        <div class="tg-progress-bar" style="width: 65%;"></div>
</div>
```

上面只是多了一个`tg-progress-striped`样式。

```
.tg-progress-striped .tg-progress-bar {
        background-image:-webkit-linear-gradient(-45deg,rgba(255,255,255,
.15) 25%,transparent 25%,transparent 50%,rgba(255,255,255,.15) 50%,rgba(2
55,255,255,.15) 75%,transparent 75%,transparent);
        background-image:linear-gradient(-45deg,rgba(255,255,255,.15) 25%
,transparent 25%,transparent 50%,rgba(255,255,255,.15) 50%,rgba(255,255,2
55,.15) 75%,transparent 75%,transparent);
        background-size:30px 30px;
}
```

注意渐变第一个参数的角度。

接下来，我们实现第三个，让进度条动起来，改变背景色位置：

```
.tg-progress-striped.tg-active .tg-progress-bar {
        -webkit-animation:tg-progress-bar-stripes 2s linear infinite;
        animation:tg-progress-bar-stripes 2s linear infinite;
}
@keyframes tg-progress-bar-stripes {
        0% {
            background-position:0 0;
    }
    100% {
            background-position:30px 0;
    }
}
```

使用：

```
<div class="tg-progress tg-progress-striped tg-active">
        <div class="tg-progress-bar" style="width: 85%;"></div>
</div>
```

当然，我们不仅仅局限于当前的颜色，要使用其他颜色，只需这样修改：

```
.tg-progress-danger .tg-progress-bar {
    background-color: #d32c46;
}
```

源码路径：`WebDemo/CSS/progress`

遮罩条

遮罩条

有些时候，我们总是需要一些遮罩条提示，比如下图效果：

通过纯CSS能否实现呢？答案是肯定的。

1、创建模板

先来看看div布局：

```
<div class="mask" data-title="遮罩条">
        <img src="images/girl.jpg" alt="">
</div>
```

在上面的代码中，相信你也留意到 `data-title` 自定义属性，这就是我们的提示语，如何用CSS获取到呢？

我们只需这样：

```
content: attr(data - title);
```

在tooltip一节中也已经用到。

2、设置CSS样式

效果图中实现第一个的完整样式：

```
.mask {
        position:relative;
        width:150px;
        height:150px;
        overflow:hidden;
}
.mask:after {
        content:attr(data-title);
        position:absolute;
        width:100%;
        top:100%; // 这里的100%很重要
        left:0;
```

```
        padding:.7em 0;
        text-align:center;
        color:#fff;
        background:rgba(0,0,0,.5);
        -webkit-transition:all .3s ease-in-out;
        transition:all .3s ease-in-out;
        opacity:0;
}
// 鼠标移动上去时
.mask:hover:after {
        opacity:1;
        -webkit-transform:translate(0,-100%);
        transform:translate(0,-100%);
}
```

注意：包裹图片的div一定要设置position定位（不为static），不然遮罩条会飞到外太空去。

第二个类似关门效果也不难，同样是利用 `:before` 和 `:after` 伪元素选择器，同时将两个的高度设为50%。

```
.mask-door {
        position:relative;
        width:150px;
        height:150px;
        overflow:hidden;
}

.mask-door:before,
.mask-door:after {
        position:absolute;
        left:0;
        width:100%;
        opacity:0;
        color:#fff;
        height:50%;
        text-align:center;
        background:rgba(0,0,0,.5);
        -webkit-transition:all .3s ease-in-out;
        transition:all .3s ease-in-out;
}
.mask-door:before {
        content:'';
        top:0;
        -webkit-transform:translate(0,-100%);
        transform:translate(0,-100%);
}
.mask-door:after {
        content:attr(data-title);
        top:50%;
        padding:.7em 0;
        -webkit-transform:translate(0,100%);
```

```
        transform:translate(0,100%);
}
.mask-door:hover:before,
.mask-door:hover:after {
        opacity:1;
        -webkit-transform:translate(0,0);
        transform:translate(0,0);
}
```

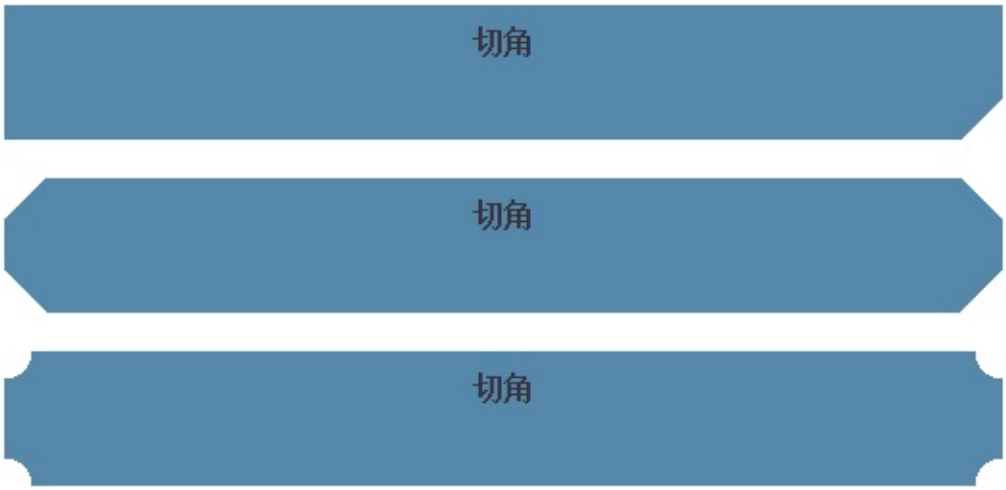
遮罩条的变化还有N多种，这里无法一一列出，你可以根据自己的需要去调整改变，比如改变高度、加圆角、改变鼠标移动上遮罩条出现的动画等等。

源码路径：`WebDemo/CSS/overlay`

切角

切角

切角效果是一种非常流行的设计风格，最常见的形态就是把元素的一个或多个角切成45度的缺口。



其实，实现切角并不会很难，主要是借助CSS3的渐变。如果你不了解渐变，可看《多彩的渐变》一节。

单个切角

```
.corner {
        background: linear-gradient(-45deg, transparent 15px, #58a 0);
}
```

在上面的代码中，我们设置渐变的角度是-45（相当于315）。

上面0的作用相当于15px。

原因：如果某个色标的位置值比整个列表中在它之前的色标的位置值都要小，则该色标的位置值会被设置为它前面所有色标位置值的最大值。

多个切角

```
.corner2 {
        background: linear-gradient(135deg, transparent 15px, #58a 0) top
```

```
 left,
        linear-gradient(-135deg, transparent 15px, #58a 0) top right,
        linear-gradient(-45deg, transparent 15px, #58a 0) bottom right,
        linear-gradient(45deg, transparent 15px, #58a 0) bottom left;
        background-size: 50% 50%;
        background-repeat: no-repeat;
}
```

实现多个切角的时候，我们需要将每一个渐变的范围都缩小到四分之一，不然渐变之间会重叠。

弧形切角

利用径向渐变实现弧形切角：

```
.corner3 {
        background: radial-gradient(circle at top left, transparent 15px,
 #58a 0) top left,
        radial-gradient(circle at top right, transparent 15px, #58a 0) to
p right,
        radial-gradient(circle at bottom right, transparent 15px, #58a 0)
 bottom right,
        radial-gradient(circle at bottom left, transparent 15px, #58a 0)
bottom left;
        background-size: 50% 50%;
        background-repeat: no-repeat;
}
```

源码路径：`WebDemo/CSS/corner`

表单

表单

大家都知道默认的表单样式一点也不好看，这一节将讲解如何美化表单。

1、创建模板

简单的div，包裹着input，还有一个span。

```
<div class="input-field">
        <input id="last_name" type="text" placeholder="账号">
        <span></span>
</div>
```

在上面的代码中，span标签是用来设置下划线的。

有些小伙伴可能要问了，不是可以用 `:after` 伪元素选择器实现下划线吗，为什么还要多一个span元素。

这就涉及到input的 `:hover` 了，你无法在表单获取到焦点时去修改 `:after` 的属性值，但是你可以 `input:hover+span` 来修改span的属性值。

2、设置CSS样式

来看看样式是怎样设置的：

```
.input-field,
.input-field *{
        -webkit-box-sizing: border-box;
        box-sizing:border-box;
}
.input-field {
        position: relative;
        width: 200px;
        margin: 20px 50px;
}

.input-field input {
        background-color: transparent;
        border: none;
        border-radius: 0;
        height: 35px;
```

```
        width: 100%;
        padding: 0;
        box-shadow: none;
        outline: none;
        -webkit-transition: all 0.3s;
        transition: all 0.3s;
}

.input-field input+span {
        position: absolute;
        top: 100%;
        left: 0;
        display: inline-block;
        max-width: 100%;
        z-index: 0;
        width: 100%;
        height: 1px;
        border-bottom: 1px solid #d9d9d9;
}

.input-field span:after {
        content: "";
        position: absolute;
        bottom: 0;
        left: 0;
        width: 100%;
        height: 2px;
        background: #2196f3;
        -webkit-transform: scale(0);
        transform: scale(0);
        -webkit-transition: all .2s ease-out;
        transition: all .2s ease-out;
}
```

在上面的代码中，你可以看到，我们是使用`transform:scale()`来实现下划线缩放动画，因为`transform`默认是以中心点为缩放点的，所以你会看到下划线向两边延伸的效果。

获取焦点时：

```
.input-field input:focus+span:after {
        -webkit-transform: scale(1);
        transform: scale(1);
}
```

3、表单校验

这里只提供三种表单校验样式：校验成功、校验提醒和校验失败。

```
// 校验成功
<div class="input-field success input-field-icon">
        <input id="last_name" type="text" placeholder="账号">
        <span></span>
        <i class="ion-android-done">
        </i>
</div>

// 校验提醒
<div class="input-field warning input-field-icon">
        <input id="last_name" type="text" placeholder="账号">
        <span></span>
        <i class="ion-alert-circled">
        </i>
</div>

// 校验失败
<div class="input-field error input-field-icon">
        <input id="last_name" type="text" placeholder="账号">
        <span></span>
        <i class="ion-android-close">
        </i>
</div>
```

这里的校验图标使用的ionicons图标，你也可以使用其他字体图标，只需替换i标签的类名即可。

样式：

```
.input-field.input-field-icon i{
        position:absolute;
        right:0;
        top:50%;
        padding:0 5px;
        font-size:1.5em;
        -webkit-transform: translate(0, -50%);
        transform: translate(0, -50%);
}
.input-field.input-field-icon input{
        padding-right:30px;
}
```

在上面的代码中有这么一段：

```
top:50%;
transform: translate(0, -50%);
```

这是干嘛的呢？其实这是让图标垂直居中的，`top`是根据父元素的百分比，而`translate()`则是根据元素自身百分比。

下划线的颜色：

```
.input-field.success span:after {
        background:#4caf50;
}
```

只需像上面一样更改，你可以设置任何颜色，甚至渐变，如下图：

变化是多样的，只要你了解了其中的原理。

源码路径：`WebDemo/CSS/form`

溢出省略号

溢出省略号

有些时候，我们需要设置宽度文字文本显示不完超出溢出的内容以省略号排版。

在以前，我们只能使用脚本去截取，然后添加省略号，可是现在，使用纯CSS，我们就可以实现。

1、单行

1.1 创建模板

```
<div class="single-line box shadow">我是单行文本溢出显示省略号略号</div>
```

1.2 设置CSS样式

```
.box{
        width:200px;  // 别漏了定义宽度
}
.single-line {
        overflow:hidden;
        white-space:nowrap;
        text-overflow:ellipsis;
}
```

上面代码的意思是让文本不换行，同时溢出方式为当内联内容溢出块容器时，将溢出部分替换为（...）

2、多行

2.1 创建模板

```
<div class="multiple-line box">
我是多行文本溢出显示省略号我是多行文本溢出显示省略号
</div>
```

2.2 设置CSS样式

```
.box{
        width:200px;  // 别漏了定义宽度
}
.multiple-line {
        display:-webkit-box;
        -webkit-box-orient:vertical;
        -webkit-line-clamp:2;
        overflow:hidden;
}
```

注意：多行截取的方法只适合webkit引擎的浏览器（chrome、Safari等）

源码路径：`WebDemo/CSS/ellipsis`

自定义选择文本样式

自定义选择文本样式

在浏览器上，默认情况下，我们拖动鼠标选中文字，会看到那段被选中的文字会是蓝色背景、白色字体，我们能否改变呢？答案是肯定的。

我们只需加上下面的代码：

```
::selection {
        background: #009a61;
        color: #fff;
}

::-moz-selection {
        background: #009a61;
        color: #fff;
}

::-webkit-selection {
        background: #009a61;
        color: #fff;
}
```

使用`::selection`伪类选择器，我们可以实现与众不同的选择文本样式。

注意：只能设置背景色和文字颜色，同时不支持渐变色。

源码路径：`WebDemo/CSS/selection`

表格

表格

虽然现在有flexbox布局，Grid布局（还未普及）等等响应式布局，但表格的作用还是不容忽视的，特别是一些数据报表之类的展示。

在这一节中，我将介绍条纹表格、鼠标悬停效果。

效果如下图所示：

1、条纹表格

条纹表格类似斑马线，实现也很简单，只需要借助伪类选择器`:nth-child()`：

```
tr:nth-child(2n+1){
        background:#ddd;
}
```

2、鼠标悬停

```
tr:hover{
        background:#ddd;
}
```

固定头的表格会在《JavaScript实战篇：固定头的表格》中讲解！

源码路径：`WebDemo/CSS/table`

导航菜单

导航菜单

一个网站总是缺少不了导航菜单，如果是以前，我们只能使用毫无生机的静态菜单，可是现在有了CSS3，我们可以实现很多动态菜单，比如下图所示：

1、属性介绍

在讲实现之前，我们先来了解三个属性：

- perspective()

`perspective()`属性设置镜头到元素平面的距离（激活3D空间）。所有元素默认都是放置在z=0的平面上（也就是你的屏幕）。比如perspective(100px)表示，镜头距离元素表面的位置是100像素。

`perspective()`取值越小，3D效果就越明显，也就是镜头越靠近真3D。

注意：`perspective()`属性是定义在子元素上的。

- rotateX()
- rotateX()定义元素绕着X轴旋转
- transition-delay
- transition-delay定义过渡延迟时间

1、折叠菜单

1.1 创建模板

```
<nav class="collapse-navbar">
        <a class="menu_btn icon-menu">
                菜单
        </a>
        <ul class="item navbar-menu">
                <li class="item4">
                        <a>子菜单</a>
                </li>
                <li class="item3">
```

```
                    <a>子菜单</a>
                </li>
                <li class="item2">
                        <a>子菜单</a>
                </li>
                <li class="item1">
                        <a>子菜单</a>
                </li>
                <li class="item">
                        <a>子菜单</a>
                </li>
        </ul>
</nav>
```

在上面的代码：

- `collapse-navbar` 表示一个折叠菜单
  - `menu_btn` 表示菜单按钮
  - `item` 表示一个子项

1.2 设置CSS样式

父元素样式：

```
.collapse-navbar .navbar-menu {
        z-index: 10000;
        position: absolute;
        width: 200px;
}
```

子菜单样式：

```
.collapse-navbar .navbar-menu> li {
        background-color: #c8161e;
        /*初始高度为0*/
        height: 0;
        opacity: 0;
        overflow: hidden;
        /*设置透视，且旋转-90度*/
    -webkit-transform: perspective(100px) rotateX(-90deg);
        -ms-transform: perspective(100px) rotateX(-90deg);
        transform: perspective(100px) rotateX(-90deg);
        -webkit-transform-origin: center top 0;
        -ms-transform-origin: center top 0;
        transform-origin: center top 0;
}
```

菜单关闭情况下，子菜单的高度设为0，透视设为100px，且绕着X轴旋转-90，当前的旋转点为li的上边中心处（你可以试试去掉`perspective()`，你会发现看不到旋转效果）。

鼠标移动上去时

```
.collapse-navbar:hover .navbar-menu> li {
        height: 50px;
        opacity: 1;
        -webkit-transform: perspective(100px) rotateX(0deg);
        -ms-transform: perspective(100px) rotateX(0deg);
        transform: perspective(100px) rotateX(0deg);
}
```

可能你也注意到了，子菜单的出现是有顺序的，这是因为加上了transition-delay过渡延迟时间

```
.collapse-navbar:hover .navbar-menu> li.item4,
.item {
        -webkit-transition: all .4s ease;
        transition: all .4s ease;
}

.collapse-navbar:hover .navbar-menu> li.item3,
.item1 {
        -webkit-transition: all .4s ease .1s;
        transition: all .4s ease .1s;
}

.item2 {
        -webkit-transition: all .4s ease .2s;
        transition: all .4s ease .2s;
}

.collapse-navbar:hover .navbar-menu> li.item1,
.item3 {
        -webkit-transition: all .4s ease .3s;
        transition: all .4s ease .3s;
}

.collapse-navbar:hover .navbar-menu> li.item,
.item4 {
        -webkit-transition: all .4s ease .4s;
        transition: all .4s ease .4s;
}
```

上面代码中的第四个时间值就是延迟时间，开启时，过渡延迟是从上往下递增；同时还需注意，当关闭菜单时，子菜单关闭的顺序是相反的，也就是过渡延迟时间从下往上递增。

2、具有子菜单的导航

2.1 创建模板

```
<nav class="shrink-navbar shrink-center">
        <ul class="item navbar-menu">
                <li class="dropdown">
                        <a>子菜单</a>
                        <ul class="dropdown-menu">
                                <li>
                                        子子菜单
                                </li>
                        </ul>
                </li>
                <li class="dropdown dropdown-arrow">
                        <a>子菜单</a>
                        <ul class="dropdown-menu">
                                <li>
                                        子子菜单
                                </li>
                        </ul>
                </li>
                <li>
                        <a>子菜单</a>
                </li>
        </ul>
</nav>
```

在上面的代码，li多了一个dropdown类，表示此菜单下有子菜单（也就是ul.dropdown-menu）。

2.2 设置CSS样式

下面来实现显示和隐藏：

```
.dropdown {
        position:relative;
}
.dropdown .dropdown-menu {
        position:absolute;
        top:100%;
        left:0;
        min-width:100%;
        opacity:0;
        height:0;
        border:1px solid #d9d99d;
        box-shadow:3px 3px 5px rgba(0,0,0,.5);
        -webkit-transform:translate(0,30px);
        transform:translate(0,30px);
        -webkit-transition:all .3s;
```

```
        transition:all .3s;
}
.dropdown:hover .dropdown-menu {
        opacity:1;
        height:auto;
        -webkit-transform:translate(0,0);
        transform:translate(0,0);
}
```

在上面的代码中，通过设置 .dropdown-menu的高度和透明度分别为0，这样可以起到隐藏效果；当鼠标移动上去时，我们只需设置透明度为1，高度为auto即可。

在这里添加了translate()并不是必需的，只是为了多点动态效果。

只要明白了原理，你就可以实现独特的组件。

源码路径：`WebDemo/CSS/nav`

动态的边框

动态的边框

要酷炫，总是需要费点心思的，看看下面的动态边框。

还是那句话，当你理解了原理后，一点都不难。

1、创建模板

```
<div class="dynamic-border dynamic-border-1">
        <span>
        </span>
        <img src="images/girl.jpg" alt="">
</div>
```

在上面的代码中，只有div和div里的第一个span是必需的。

2、设置CSS样式

我们通过 `div.dynamic-border` 和 `span` 的 `:before` 和 `:after` 伪元素来实现边框。

```
.dynamic-border {
        position:relative;
        width:200px;
        height:200px;
        background:gray;
}
.dynamic-border:before,
.dynamic-border:after,
.dynamic-border span:first-child:before,
.dynamic-border span:first-child:after {
        content:"";
        position:absolute;
        background:red;
        -webkit-transition:all .2s ease;
        transition:all .2s ease;
}
/*上边边框*/
.dynamic-border:before {
        width:0;  // 初始宽度
        top:-2px;
        right:0;
        height:2px;
}
```

```
/*右边边框*/
.dynamic-border:after {
        width:2px;
        height:0;  // 初始高度
        right:-2px;
        bottom:0;
}
/*下边边框*/
.dynamic-border span:first-child:before {
        width:0;  // 初始宽度
        height:2px;
        bottom:-2px;
        left:0;
}
/*左边边框*/
.dynamic-border span:first-child:after {
        width:2px;
        height:0;  // 初始高度
        top:0;
        left:-2px;
}
```

在上面的代码中，我并没有采取border，而是使用了div里的`:before`和`:after`，还有span里的`:before`和`:after`制作了边框。

鼠标移动上去的动画，实际就是改变宽高。

```
/*鼠标移动上去时*/
.dynamic-border:hover:before,
.dynamic-border:hover span:first-child:before {
        width:calc(100% + 2px);
}
.dynamic-border:hover:after,
.dynamic-border:hover span:first-child:after {
        height:calc(100% + 2px);
}
/*添加过渡延迟时间*/
.dynamic-border-1:hover:before,
.dynamic-border-1:hover span:first-child:before {
        -webkit-transition-delay:.2s;
        transition-delay:.2s;
}
```

第二种动画

布局依旧：

```
<div class="dynamic-border dynamic-border-2">
        <span>
```

```
        </span>
        <img src="images/girl.jpg" alt="">
</div>
```

然后改变一下各边的过渡延迟时间：

```
.dynamic-border-2:hover span:first-child:before {
        -webkit-transition-delay:.2s;
        transition-delay:.2s;
}
.dynamic-border-2:hover:after {
        -webkit-transition-delay:.4s;
        transition-delay:.4s;
}
.dynamic-border-2:hover:before {
        -webkit-transition-delay:.6s;
        transition-delay:.6s;
}
```

源码路径：`WebDemo/CSS/border`

文件上传组件美化

文件上传组件美化

默认的上传组件看起来很丑，有些人就会很纠结地找有没有方法去改变input[type="file"]的默认样式。

其实完全没必要，我们可以自己DIY。

先看看美化后的组件：

我们来分析最后一个。

1、创建模板

先看布局：

```
<div class="file file-input">
        <div class="file-inner">
                选择文件
                <button class="btn btn-primary file-inner-btn">
                        文件上传
                        <i class="ion-ios-cloud-upload-outline">
                        </i>
                </button>
        </div>
        <input type="file" />
</div>
```

在上面的代码：

- `file`表示一个上传组件
  - `file-inner`表示自定义的UI

2、设置CSS样式

```
.file {
        position:relative;
}
.file input {
        position:absolute;
        top:0;
        left:0;
```

```
        opacity:0;
        width:100%;
        height:100%;
}
.file-inner {
        position:relative;
        width:250px;
        height:35px;
        border:1px solid #d9d9d9;
        border-radius:5px;
        padding-left:10px;
        line-height:35px;
}
.file-inner .file-inner-btn {
        position:absolute;
        right:0;
        top:0;
        height:100%;
        box-shadow:none;
}
```

不知道你有没有从上面的代码中看出了猫腻。

首先，上面的组件，都是由其他元素组成的。

对于input的样式呢？我将input的透明度 `opacity` 设为了 `0`（这样你就看不到默认的上传样式，而是看到自定义的），而且设置定位 `positon:absolute` ，也就是将其input放置在最上层。

源码路径： `WebDemo/CSS/fileupload`

打字机效果

打字机效果

1、属性介绍

`alternate`是animation-direction的值之一，表示动画应该轮流反向播放。

这里还使用了animation的`steps()`函数，它是一个阶跃函数，用于把整个操作领域切分为相同大小的间隔，每个间隔都是相等的。

语法：

```
steps(number[, end | start])
```

参数说明：

- number参数指定了时间函数中的间隔数量（必须是正整数）
- 第二个参数是可选的，可设值：start和end，表示在每个间隔的起点或是终点发生阶跃变化，如果忽略，默认是end。

注意：第二个参数还有两个内置值，step-start等同于steps(1,start)，动画分成1步，动画执行时以左侧端点为开始；step-end等同于steps(1,end)：动画分成1步，动画执行时以结尾端点为开始。

2、创建模板

先来放置你要打印的文字：

```
<h1>Hello World!</h1>
```

3、设置CSS样式

定义打字效果：

```
  @keyframes typing {
         from {
```

```
            width:0;
    }
}
@keyframes blink-caret {
        50% {
            border-color:transparent;
    }
}
h1 {
        font:bold 200% Consolas,Monaco,monospace;
        border-right:.1em solid;
        width:6.6em;
        margin:2em 1em;
        white-space:nowrap;
        overflow:hidden;
        animation:typing 20s steps(12,end),blink-caret .5s step-end infin
ite alternate;
}
```

源码路径：`WebDemo/CSS/print`

多形状图像

多形状图像

1、属性介绍

要实现上面这些多形状图像，其实并不难，先来了解一个属性：`border-radius`。

`border-radius`相信大家都不陌生，用来定义圆角边框。

先来看看一个特别的用法：

```
border-radius: 10px / 20px;
```

这是啥意思呢？其实这是单独指定水平半径和垂直半径。

还是用图说话：

定义顺序：`top right bottom left / top right bottom left`

我们还可以设置单边：

```
border-top-left-radius: 10px 20px;
```

分别表示水平半径和垂直半径。

了解完`border-radius`属性了，现在来看看上面的多形状图像是怎样实现的？

圆形（效果图的第一个）很简单：

```
border-radius: 50%;
```

注意：取百分比值时，是相对于当前元素的高宽的。

半椭圆（效果图中的最后一个）：

```
border-radius: 50% 50% 0 0 / 100% 100% 0 0;
```

四分之一椭圆（倒数第二个）：

```
border-radius: 100% 0 0;
```

只要有耐心，可以实现更多效果。

源码路径：`WebDemo/CSS/shape`

心跳灯和呼吸灯

心跳灯和呼吸灯

心跳灯

```
<div class="heartbeat">心跳灯</div>
```

动画样式：

```
.heartbeat {
    -webkit-animation-name: heartbeat;
        -webkit-animation-duration: 0.83s;
        -webkit-animation-timing-function: ease-in-out;
        -webkit-animation-iteration-count: infinite;
}
@keyframes heartbeat {
        from {
            opacity:0.1;
    }
    50% {
            opacity:1;
    }
    to {
            opacity:0.1;
    }
}
```

呼吸灯

```
<div class="breath">心跳灯</div>
```

动画样式：

```
.breath {
    -webkit-animation-name: breath;
        -webkit-animation-duration: 6s;
        -webkit-animation-timing-function: ease-in-out;
        -webkit-animation-iteration-count: infinite;
}
```

```
@keyframes breath {
        from {
            opacity:0.1;
    }
    50% {
            opacity:1;
    }
    to {
            opacity:0.1;
    }
}
```

可能你也注意到了，心跳灯和呼吸灯的区别只是周期时间（animation-duration)不同。

源码路径：`WebDemo/CSS/breath`

竖着排的文字

竖着排的文字

这一节并没有什么技巧可言，只是添加一个偶尔需要的功能。

1、属性介绍

先来了解一个属性 `writing-mode` ，此属性用来设置元素的内容块固有的书写方向。

```
writing-mode: horizontal-tb | vertical-lr | vertical-rl;
```

- `horizontal-tb` 表示水平方向自上而下的书写方式。
- `vertical-rl` 表示垂直方向自右向左的书写方式
- `vertical-lr` 表示垂直方向自左向右的书写方式

注意：使用时需要加上webkit私有前缀。

IE中：

```
writing-mode: lr-tb | tb-rl;
```

- `lr-tb` 水平方向自左向右的书写方式
- `tb-rl` 垂直方向自上而下的书写方式。

2、创建模板

```
<div class="test lr-tb">
        ABCDE本段文字将按照水平从左到右的书写方向进行流动。
</div>
<div class="test tb-rl">
        ABCDE本段文字将按照垂直从右到左的书写方向进行流动。
</div>
<div class="test tb-lr">
        ABCDE本段文字将按照垂直从左到右的书写方向进行流动。
</div>
```

3、设置CSS样式

```
.lr-tb {
        -webkit-writing-mode: horizontal-tb;
        writing-mode: lr-tb;
        writing-mode: horizontal-tb;
}

.tb-rl {
        -webkit-writing-mode: vertical-rl;
        writing-mode: tb-rl;
        writing-mode: vertical-rl;
}

.tb-lr {
        -webkit-writing-mode: vertical-lr;
        writing-mode: tb-rl;
        writing-mode: vertical-lr;
}
```

源码路径：`WebDemo/CSS/vertical`

面包屑导航

面包屑导航

面包屑导航实现其实并没有什么难度，关键是耐心。

Thema Data Table Basic Table

Home / Library

属性介绍

三角形是利用border属性来设置，来看看border实质的产生：
一个简单的div：

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

样式如下：

```
.box {
        width: 50px;
        height: 50px;
        border-top: 10px solid red;
        border-right: 10px solid blue;
        border-bottom: 10px solid purple;
        border-left: 10px solid green;
        background: black;
}
```

在上面的代码中，给不同的边框设置不同的颜色，而背景色设置为黑色。

效果如下：

从效果图来看，你会发现其实border并不是一条线，当你将width和height都设置为0时，你会发现每条边框是三角形的。

这也是实现三角形的原理。

1、创建模板

```
<ul class="breadcrumbs">
        <li>
                <a href="#">Thema</a>
        </li>
        <li>
                <a href="#">Data Table</a>
        </li>
        <li class="active">
                <a href="#">Basic Table</a>
        </li>
</ul>
```

在上面的代码：

- `breadcrumbs`表示一个面包屑导航
  - `li`表示一个子项

2、设置CSS样式

```
ul.breadcrumbs {
        margin: 0;
        padding: 0;
        list-style: none;
        overflow: hidden;
        width: 100%;
}

ul.breadcrumbs a {
        background: #F5F5F5;
        padding: .3em 1em;
        float: left;
        color: #666;
        text-decoration: none;
        position: relative;
}

ul.breadcrumbs a:after {
        content: "";
        position: absolute;
        top: 50%;
        margin-top: -1.5em;
        border-top: 1.5em solid transparent;
        border-bottom: 1.5em solid transparent;
        border-left: 1em solid #F5F5F5;
        right: -1em;
        margin-right: 1px;
}
```

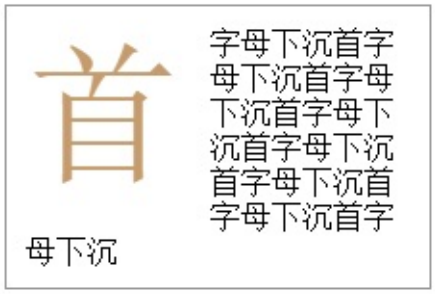
```
ul.breadcrumbs li {
        float: left;
        margin: 0 .2em 0 1em;
}

ul.breadcrumbs a:hover {
        background: #BDBDBD;
}

ul.breadcrumbs a:hover:after {
        border-left-color: #BDBDBD;
}
```

源码路径：`WebDemo/CSS/breadcrumb`

首字母下沉

首字母下沉

在杂志排版中，常常能看到首字下沉的效果。



这一节就来介绍如何用CSS实现首字下沉效果！

1、属性说明

`::first-letter`伪元素选择器用于选取指定选择器的首字母。

2、创建及模板

很简单的布局：

```
<p>首字母下沉首字母下沉首字母下沉首字母下沉首字母下沉首字母下沉首字母下沉首字母下沉</p
>
```

也可以换成其他元素。

3、设置CSS样式

其实现原理非常的简单，就是把首字font-size值设置较大，然后通过float来实现，只需如下设置：

```
p::first-letter {
        color:#c69c6d;
        float:left;
        font-size:5em;
        margin:0 .2em 0 0;
}
```

就是这么简单就可以实现首字母下沉效果，当然，你也可以添加更多的样式美化。

源码路径：`20160116/CSS/letter`

美化有序列表

美化有序列表

我们都知道默认的有序列表很简单，只是简单的1、2、3，我们无法去修改其样式。

那如果要实现下面这种有序列表，我们如何实现呢？当然，我们说的是只用CSS实现。

下面就来看看如何美化有序列表！

我们就以第一种为例来讲解。

1、属性介绍

`counter-reset` 属性必须用于选择器，主要用来标识该作用域，其值可以自定义

```
counter-reset：none | [<identifier> <integer>]
```

- none:阻止计数器增加
- ： identifier定义一个或多个将被重置的selector，id，或者class
- ： 定义被重置的数值，可以为负值，默认值是0

注意：`identifier` 和其后的数字之间要用空格隔开。

`counter-increment` 就是“计数器-递增”的意思。值为counter-reset的1个或多个关键字。后面可以跟随数字，表示每次计数的变化值。如果缺省，则使用默认变化值1。

```
counter-increment：none | [<identifier> <integer>]
```

- none:阻止计数器增加
- ： identifier定义一个或多个将被增加的selector，id，或者class
- ： 定义计算器每次增加的数值，可以为负值，默认值是1

`counter()` 用来设置插入计数器的值。一般配合content使用。

```
content: counter(name)
```

name为 `counter-reset` 的名称

2、创建模板

```
<div class="roundedList">
        <ol>
                <li>列表</li>
                <li>列表</li>
                <li>列表</li>
                <ol>
                        <li>列表</li>
                        <li>列表</li>
                </ol>
        </ol>
</div>
```

2、设置CSS样式

基本样式

```
.roundedList,
.roundedList *{
        margin: 0;
        padding: 0;
        -webkit-box-sizing: border-box;
        -moz-box-sizing: border-box;
        box-sizing: border-box;
}

.roundedList ol {
        width: 200px;
        /*注意这里！！*/
        counter-reset: li;
        list-style: none;
        font-size: 15px;
        text-shadow: 0 1px 0 rgba(255, 255, 255, .5);
}
```

```
.roundedList> ol {
        margin-left: .7em;
}

.roundedList ol ol{
        margin: 0 0 0 2em;
}
```

留意上面的 `counter-reset: li`，表示此计数器的作用域是li选择器。

编码

```
.roundedList ol li {
        position: relative;
        padding: 0.4em 0.4em 0.4em 2em;
        margin: 0.5em 0;
        background: #ddd;
        color: #444;
        text-decoration: none;
        border-radius: 0.3em;
}

.roundedList ol li:before {
        content: counter(li);
        counter-increment: li;
        position: absolute;
        left: -1.3em;
        top: 50%;
        margin-top: -1.3em;
        background: #f60;
        height: 2em;
        width: 2em;
        line-height: 2em;
        border: 0.3em solid #fff;
        text-align: center;
        color: #fff;
        font-weight: bold;
        border-radius: 2em;
}
```

在上面的代码中，我们以上面定义的`count-reset`的`li`为计数名，分别用`counter(li)`作为插入内容和`counter-increment: li`作为递增。

当然，li后面可以加以空格隔开的数字，比如：

```
counter-increment: li 2;
```

这样，不管你添加多少个li，它都会自动计数，而不用脚本去计数。

源码路径：`20160116/CSS/orderedList`

缎带效果

缎带效果

上面这种缎带效果，我们以前只能用图片来实现，不过，现在我们只用CSS就能实现了。

这一节将教你如何实现缎带效果。

1、编辑模板

```
<div class="bubble">
        <div class="rectangle">
                <h2>缎带</h2>
        </div>
        <div class="info">
                <h2>缎带效果</h2>
        </div>
</div>
```

2、设置CSS样式

基本样式

```
.bubble {
        margin: 0 auto;
        width: 350px;
        border-radius: 10px;
        box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
        position: relative;
        z-index: 90;
}
```

制作矩形

```
.rectangle {
        background: #f60;
        height: 50px;
        width: 380px;
        box-shadow: 0px 0px 4px rgba(0, 0, 0, 0.55);
        z-index: 100;
        position: relative;
        left: -15px;
        top: 30px;
}
```

制作左侧三角形

```
.bubble:after {
        content: "";
        border: 15px solid #c55205;
        border-color: transparent #c55205 transparent transparent;
        height: 0px;
        width: 0px;
        position: absolute;
        left: -30px;
        top: 65px;
        z-index: -1;
}
```

制作右侧三角形

```
.bubble:before {
        content: "";
        border: 15px solid #c55205;
        border-color: transparent transparent transparent #c55205;
        height: 0;
        width: 0;
        position: absolute;
        right: -30px;
        top: 65px;
        z-index: -1;
}
```

到这里，我们就实现了缎带效果。

当然，我们还可以实现更多，比如：

你可以尝试一下，如果实现不了，我的Demo里有例子。

源码路径：`20160116/CSS/ribbon`

JavaScript实战篇

JavaScript实战篇

功能目录：

- 点击水波纹效果
- 手风琴布局
- 滑块
- 下拉菜单
- 全屏滚动
- 富文本编辑器
- 带表情输入的评论框
- 图片懒加载
- 开启全屏
- 返回顶部
- 选项卡
- 上传图片预览
- 走马灯
- 万年历
- 树形菜单
- 旋转加载
- 瀑布流
- 圆形水波纹加载进度
- 检测是否移动端
- 搜索过滤
- 弹幕
- 收缩菜单
- 幻灯片
- 固定头的表格
- 自定义滚动条
- 城市联动选择器
- 滚动监听
- 边栏悬浮菜单

点击水波效果

点击水波纹

点击水波纹是一个使用非常广泛的特效，它可以根据你在元素上的点击位置来展开波纹效果。

效果如下：

在这一节中，我们将介绍如何实现点击水波纹。

1、创建模板

一般来说，我们都是给一个按钮添加波纹效果，所以，我们需要一个简单的按钮：

```
<button class="btn btn-default ripple-effect">点击波纹</button>
```

类名`ripple-effect`表示此按钮需要添加点击水波纹效果。

2、设置CSS样式

```
.ripple-effect {
        position: relative;
        overflow: hidden;
}
.btn {
        width:150px;
        height:40px;
        border:1px solid #d9d9d9;
        outline:none;
        background:#008DC5;
        color:#fff;
}
```

其实只有`ripple-effect`里的样式是必需的，下面的.btn是为了让按钮更好看。

3、添加波纹元素

接下来添加一个放置波纹动画的元素（类名为`ripple`）的样式：

```
.ripple {
        border-radius: 50%;
        background: rgba(0, 0, 0, .2);
        -webkit-transform: scale(0);
        transform: scale(0);
        position: absolute;
        opacity: 1;
}
```

你可以看到，我们给元素添加了`transform:scale(0)`、`border-radius:50%`和`opacity:1`，表示缩放比例为0，设为圆形和透明度为1.

4、添加波纹动画

再添加动画效果，在后面给类名为`ripple`元素动态添加下面的动画：

```
.rippleEffect {
        -webkit-animation: rippleEffect 2s cubic-bezier(0.23, 1, 0.32, 1)
;
        animation: rippleEffect 2s cubic-bezier(0.23, 1, 0.32, 1);
}

@keyframes rippleEffect {
        100% {
            -webkit-transform: scale(2);
            transform: scale(2);
                opacity: 0;
        }
}
```

透明度在动画中变为0，而缩放比例变为2，也就是说，圆形由小变大，且透明度逐渐变为0。

6、添加脚本

我们添加一个名为ripple的函数，其中包括了获取鼠标位置、获取波纹元素的高宽和left/top、添加`rippleEffect`动画，还有监听动画结束。

```
function ripple(event, $this) {
        event = event || window.event;
    //  获取鼠标位置
        var x = event.pageX || document.documentElement.scrollLeft + docu
ment.body.scrollLeft + event.clientX;
        var y = event.pageY || document.documentElement.scrollTop + docum
ent.body.scrollTop + event.clientY;
        var wx = $this.offsetWidth;
```

```
        x = x - $this.offsetLeft - wx / 2;
        y = y - $this.offsetTop - wx / 2;
        // 添加.ripple元素
        var ripple = document.createElement('span');
        ripple.className = 'ripple';
        var firstChild = $this.firstChild;
        if (firstChild) {
                $this.insertBefore(ripple, firstChild);
        } else {
                $this.appendChild(ripple);
        };
        ripple.style.cssText = 'width: ' + wx + 'px;height: ' + wx + 'px;
top: ' + y + 'px;left: ' + x + 'px';
        ripple.classList.add('rippleEffect');
        //  监听动画结束，删除波纹元素
        animationEnd(ripple,
        function() {
                this.parentNode.removeChild(ripple);
        });
};
```

我们来看看几段重要的代码：

（1）获取鼠标的位置

```
var x = event.pageX || document.documentElement.scrollLeft + document.bod
y.scrollLeft + event.clientX;
var y = event.pageY || document.documentElement.scrollTop + document.body
.scrollTop + event.clientY;
```

`pageX/Y`获取到的是触发点相对文档区域左上角距离，会随着页面滚动而改变，`||`后面的代码是为了兼容IE。

（2）计算相对位置

上面的x和y获取的是相对于文档的鼠标位置，而`transform`默认是以元素的中心点来转换的，所以我们需要获取相对位置，也就是波纹元素中心点相对于按钮中心点的位置。

```
        var wx = $this.offsetWidth;
        x = x - $this.offsetLeft - wx / 2;
        y = y - $this.offsetTop - wx / 2;
        ripple.style.cssText = 'width: ' + wx + 'px;height: ' + wx + 'px;
top: ' + y + 'px;left: ' + x + 'px';
```

本身波纹元素`ripple`的中心点是在元素的中心，所以我们根据鼠标位置和元素的位置，计

算应该偏移的位置。

7、添加点击事件

我们给包含类名`ripple-effect`的元素需要添加点击事件：

```
var btn = document.querySelectorAll('.ripple-effect');
for (var i = 0; i < btn.length; i++) {
        addEvent(btn[i], 'click',
        function(e) {
                ripple(e, this);
        });
}
```

如果你需要改变波纹的颜色值，只需这样设置：

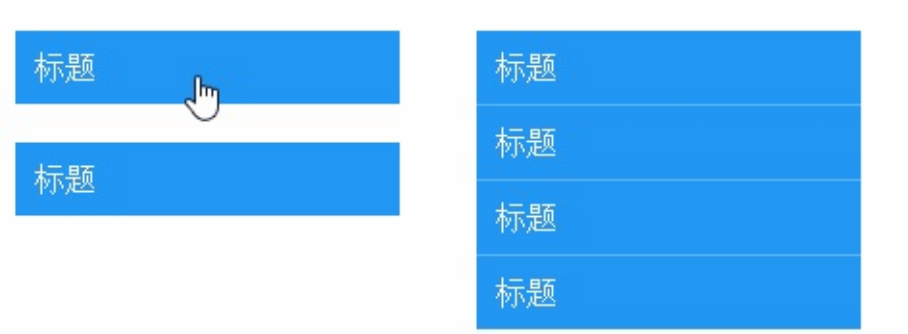
```
.btn-primary .ripple {
        background:rgba(255,255,255,.4);
}
```

源码路径：`WebDemo/JavaScript/ripple`

手风琴布局

手风琴特效

手风琴特效（折叠特效），当然也可以叫做侧栏菜单导航。



我们先来看单个手风琴元素如何实现？

1、创建模板

手风琴布局可以这样：

```
<div class="accordion-list">
        <div class="accordion-item accordion-item-expand">
                <div class="accordion-item-toggle">
                        标题
                </div>
                <div class="accordion-item-content">
                        <div>
                                手风琴特效，当然也可以叫做侧栏菜单导航，集中了当今比较流行的竖向菜单导航，  既可以控制只展开一个，亦可以点击展开或者折叠
                        </div>
                </div>
        </div>
</div>
```

在上面的代码

- `accordion-list`：多个手风琴的列表，可选
  - `.accordion-item`：单个手风琴
    - `accordion-item-toggle`，用来展开 / 折叠手风琴元素内容的开关（必选）
    - `accordion-item-content`，手风琴元素的内容（必选）。
  - `accordion-item-expand`：展开的手风琴元素

如果你希望独立的可折叠元素，可以这样：

```
<div class="accordion-item accordion-item-expand">
        <div class="accordion-item-toggle">
                标题
        </div>
        <div class="accordion-item-content">
                <div>
                        手风琴特效，当然也可以叫做侧栏菜单导航，集中了当今比较流行的竖向菜单导航， 既可以控制只展开一个，亦可以点击展开或者折叠
                </div>
        </div>
</div>
```

2、设置CSS样式

`accordion-item-content` 的样式：

在合并状态 `height` 设为0，`overflow` 设为 `hidden`

```
.accordion-item-content {
        position: relative;
        overflow: hidden;
        height: 0;
        -webkit-transition-duration: 300ms;
        transition-duration: 300ms;
        -webkit-transform: translate3d(0, 0, 0);
        transform: translate3d(0, 0, 0);
}
```

留意一下这里的过渡时间，后续的展开和关闭也是依赖CSS3的过渡。

展开状态的样式：

```
.accordion-item-expand>.accordion-item-content{
        height:auto;
}
```

3、展开/关闭

首先给所有`accordion-item-toggle`添加点击事件：

```
var collapse = document.querySelectorAll('.accordion-item');

for (var i = 0; i < collapse.length; i++) {
        addEvent(collapse[i].querySelector('.accordion-item-toggle'), 'cl
ick',
        function() {
                var con = this.nextElementSibling;
                if (!con) return;
                if (this.parentNode.classList.contains('accordion-item-ex
pand')) {
                        accordionCallapse.close(this.parentNode);
                } else {
                        accordionCallapse.open(this.parentNode);
                }
        },
        false);
}
```

在上面的代码中，`accordionCallapse.close()`用来关闭手风琴，`accordionCallapse.open()`用来展开手风琴，两个方法都需要传入一个`accordion-item`的元素作为参数。

accordionCallapse.open()

```
var content = item.querySelector('.accordion-item-content');
content.style.height = content.scrollHeight + 'px';

deleteTransitionEnd(content, accordionCallapse.transitionEnd);
transitionEnd(content, accordionCallapse.transitionEnd);

item.classList.add('accordion-item-expand');
```

在open()方法中，我们通过`scrollHeight`来获取内容的真实高度，然后赋值给`accordion-item-content`（初始高度为0，所以展开过渡是高度`0->scrollHeight`）。

`deleteTransitionEnd()`和`transitionEnd()`两个方法分别是移除过渡结束的事件

监听和绑定过渡结束的事件监听。

transitionEnd()

```
transitionEnd: function() {
        var content = this;
        var item = this.parentNode;
        if (item.classList.contains('accordion-item-expand')) {
                setTransitionDuration(content, 0);
                content.style.height = 'auto';
                var clientLeft = content.clientLeft;
                setTransitionDuration(content, '');
        } else {
                content.style.height = '';
        }
}
```

`setTransitionDuration()` 方法是设置CSS3 的过渡时间。

这里的`var clientLeft = content.clientLeft;`是为了延迟过渡，你可以试试删除这一行看看。

注意：当过渡时间为0时，我们肉眼是很难看得到过渡之间的变化的。

accordionCallapse.close()

```
var content = item.querySelector('.accordion-item-content');
item.classList.remove('accordion-item-expand');

setTransitionDuration(content, 0);
content.style.height = content.scrollHeight + 'px';
var clientLeft = content.clientLeft;
setTransitionDuration(content, '');
content.style.height = '';

deleteTransitionEnd(content, accordionCallapse.transitionEnd);
transitionEnd(content, accordionCallapse.transitionEnd);
```

当需要关闭时，我们依旧需要将内容高度设为`scrollHeight`，也就是过渡是`scrollHeight -> 0`。

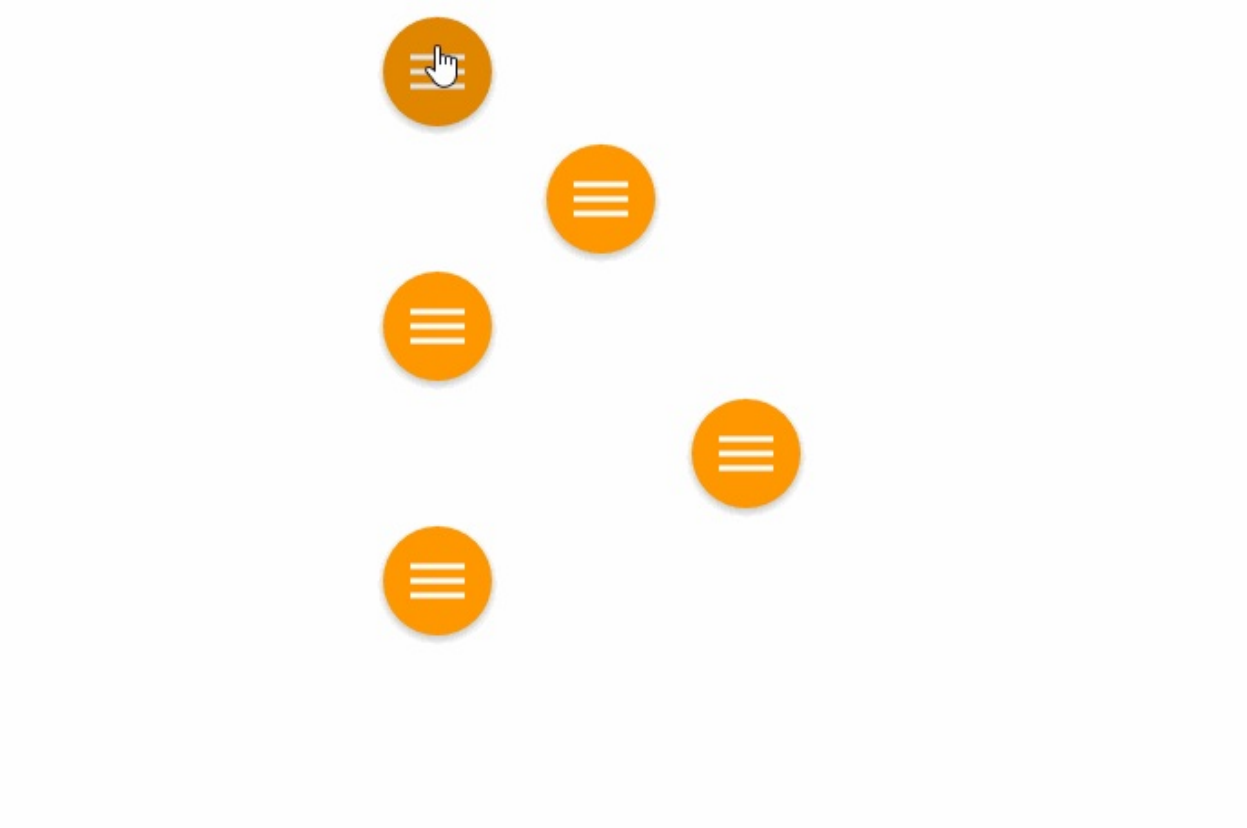
手风琴特效就是这么简单的实现了。

源码路径：`WebDemo/JavaScript/collpase`

收缩菜单

收缩菜单

收缩菜单是指点击一个指定图标后，在图标的周围会展开很多个相关菜单图标选项，再次点击后即可收缩。



1、创建模板

```
<div class="suspend suspend-horizontal">
        <span class="suspend-btn burge burge-line">
                <span></span>
                <span></span>
                <span></span>
        </span>
        <span class="suspend-item">
                <i class="ion-home">
                </i>
        </span>
</div>
```

在上面的代码：

- `suspend` 是一个收缩菜单列表
- `suspend-btn` 是指定显示按钮
- `suspend-item` 是一个相关菜单选项

2、设置CSS样式

```
.suspend {
        position: relative;
        font-size: 20px;
        width: 3em;
        height: 3em;
        z-index:99;
}

.suspend-btn{
        z-index: 99;
}

.suspend-btn,
.suspend-item {
        display: block;
        width: 100%;
        height: 100%;
        border-radius: 100%;
        color: #fff;
        line-height: 3em;
        background-color: #ff9800;
        border-color: #e68900;
        text-align: center;
        cursor: pointer;
        overflow: hidden;
        box-shadow: 0 2px 5px rgba(0, 0, 0, 0.16), 0 2px 10px rgba(0, 0,
0, 0.12);
}

.suspend-item,
.suspend-btn {
        position: absolute;
        top: 0;
        left: 0;
}

.suspend-item {
        opacity: 0;
        z-index: 1;
}

.suspend-item,
.suspend-item:after,
.suspend-btn:after,
```

```
.suspend-btn span{
        -webkit-transition: all .3s ease-out;
        -moz-transition: all .3s ease-out;
        -ms-transition: all .3s ease-out;
        transition: all .3s ease-out;
}
```

在上面的代码中，`suspend`必须加上定位属性（position非static），`suspend-btn`和`suspend-item`都是采取绝对定位，同时还要注意设置`suspend-btn`和`suspend-item`之间的层叠关系（也就是`z-index`的值）。

留意最后面的`transition`属性。

3、JavaScript设置

要展开相关菜单，其实只需计算并设置相关菜单的位置即可，比如：右边展开的菜单:

```
for (var i = 0; i < n.items.length; i++) {
        var x = 'translate(' + op + ((n.itemWidth + n.params.distance) *
(i + 1)) + 'px, 0)';
        setTransform(n.items[i], x);
        n.items[i].style.top = '0px';
        n.items[i].style.opacity = 1;
};
```

在上面的代码中，`n.items`是所有相关菜单的列表，`n.itemWidth`是菜单的宽度，`n.params.distance`是每个菜单之间的距离，`setTransform()`方法是我封装的`transform`：

```
var setTransform = function(element, animation) {
        element.style.webkitTransform = animation;
        element.style.mozTransform = animation;
        element.style.oTransform = animation;
        element.style.msTransform = animation;
        element.style.transform = animation;
};
```

那如何实现圆形菜单呢？其实只需记住一点，那就是半径不变原则。

计算半径后，这时就需要通过勾股定理来计算位置值了：

```
var x = Math.sin(angle) * r;
var y = Math.cos(angle) * r;
```

```
var xy = 'translate(' + x + 'px,' + y + 'px)';
setTransform(n.items[i], xy);
```

在上面的代码中，`r` 是圆形半径，`angle` 是弧度制。

这个插件的参数说明：

```
var suspend = new LazySuspend('.suspend-circle', {
        type: 'circle',
        distance: 50,
        delay: 100,
        direction: 'rb',
        angle: 45
});
```

- 第一个参数是收缩菜单的类名
- 第二个参数是一个配置选项，可选。
    - `type` 是指类型，可选值：`horizontal`（右侧，默认）、`horizontal-reverse`（左侧）、`vertical`（顶部）、`vertical-reverse`（底部）、`circle`（圆形）。
    - `distance` 是每个菜单的间距
    - `delay` 是菜单出现的延迟时间，只有 `type` 为 `circle` 时起作用
    - `direction` 指菜单位置，只有 `type` 为 `circle` 时起作用，可选值：`lt`、`lb`、`rt`、`rb`（左上、左下、右上、右下）。
    - `angle` 角度，，只有 `type` 为 `circle` 时起作用，默认45

源码路径：`WebDemo/JavaScript/shrink`

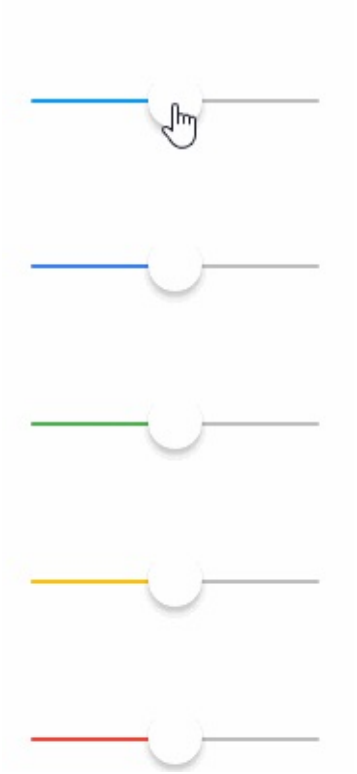
滑块

滑块

滑块(Slider) 拖动手柄来选择一个数值。

在HTML5中，input新增了一个`range`类型，不过默认样式很丑。

你是否希望拥有如下的`range`滑块：



在这一节，我将教你如何实现？

其实，有两种方式：

1、使用浏览器的私有属性设置

不过，Chrome、Firefox、IE上各不相同，下面来列出相应的属性：

Chrome

```
.range-slider input[type=range]::-webkit-slider-thumb {
        /*滑块*/
}
.range-slider input[type=range]::-webkit-slider-runnable-track {
        /*轨道*/
}
```

Firefox

```
.range-slider input[type=range]::-moz-range-thumb {
        /*滑块*/
}

.range-slider input[type=range]::-moz-range-track {
        /*轨道*/
}

.range-slider input[type=range]::-moz-range-progress {
        /*已填充的部分*/
}
```

IE+9

```
.range-slider input[type=range]::-ms-track {
        /*轨道*/
}

.range-slider input[type=range]::-ms-thumb {
        /*滑块*/
}

.range-slider input[type=range]::-ms-fill-lower {
        /*进度条已填充的部分*/
}

.range-slider input[type=range]::-ms-fill-upper {
        /*进度条未填充的部分*/
}
```

通过修改`range`的样式虽然可以美化，不过兼容性不少，接下来用自定义`range`的方法来实现。

2、自定义滑块

2.1 创建模板

```
<div class="range-slider">
        <div class="range-bar">
        </div>
        <div class="range-bar range-bar-active">
        </div>
        <div class="range-knob-handle">
                <div class="range-knob">
                </div>
        </div>
        <span class="range-slider-tooltip">
        </span>
</div>
```

在上面的代码：

- `range-slider`：一个滑块
  - `range-bar`：整体轨道
  - `range-bar-active`：已填充的轨道
  - `range-knob-handle`：可拖动的滑块
  - `range-knob`：圆形滑块
  - `range-slider-tooltip`：显示当前数值的工具块

2.2 设置CSS样式

来看看对应的样式（你也可以自己定义）：

```
.range-bar {
    position: absolute;
    top: 21px;
    left: 0;
    width: 100%;
    height: 2px;
    border-radius: 1px;
    background: #bdbdbd;
    pointer-events: none;
}
.range-bar-active {
    bottom: 0;
    width: auto;
    background: #387ef5;
}
.range-knob-handle {
    position: absolute;
    top: 21px;
    left: 0%;
    margin-top: -21px;
    margin-left: -21px;
    width: 42px;
```

```
    height: 42px;
    text-align: center;
}
.range-knob {
    position: absolute;
    top: 7px;
    left: 7px;
    width: 28px;
    height: 28px;
    border-radius: 50%;
    background: #fff;
    box-shadow: 0 3px 1px rgba(0, 0, 0, 0.1), 0 4px 8px rgba(0, 0, 0, 0.1
3), 0 0 0 1px rgba(0, 0, 0, 0.02);
    pointer-events: none;
}

.range-slider .range-slider-tooltip {
        font-size: 18px;
        position: absolute;
        z-index: 1;
        top: -34px;
        padding: 5px 10px;
        text-align: center;
        opacity: .8;
        color: #333;
        border: 1px solid #ddd;
        border-radius: 6px;
        background-color: #fff;
        text-shadow: 0 1px 0 #f3f3f3;
        display: none;
}
```

如需要改变已填充部分的颜色，只需这样：

```
.range-slider.warning .range-bar-active{
        background:#ffc107;
}
```

填充部分其实就是设置 `range-bar-active` 的 `right` 属性。

3、脚本控制

由于滑块是可以点击选择，所以我们需要将鼠标事件绑定到 `range-slider` 上，而不是滑块 `range-knob-handle` 上。

实现滑块的重要一点就是获取相对于 `range-slider` 的鼠标位置：

```
function getPoint(element, event) {
        /*将当前的触摸点坐标值减去元素的偏移位置，返回触摸点相对于element的坐标值*/
```

```
        event = event || window.event; //e.originalEvent.targetTouches[0]
        var touchEvent = isTouch ? event.changedTouches[0] : event;
        var x = (touchEvent.pageX || touchEvent.clientX + document.body.s
crollLeft + document.documentElement.scrollLeft);
        x -= element.offsetLeft;
        var y = (touchEvent.pageY || touchEvent.clientY + document.body.s
crollTop + document.documentElement.scrollTop);
        y -= element.offsetTop;
        return {
                x: x,
                y: y
        };
};
```

在上面的代码中，`isTouch`是为了判断是否移动端。

获取到相对值，就只需计算相对百分比即可：

```
var percent = value / n.range.offsetWidth * maxPercent;
```

`value`是鼠标相对`range`的位置，`range.offsetWidth`的宽度，`maxPercent`是100.

注意，由于最小值`min`有可能是负值，所以当前数值并不一定等于上面的百分比`percent`。

需要这样：

```
var t = n.range.max - n.range.min;
var cp = Math.ceil(t * percent / 100) + n.range.min;
```

在上面的代码中，首先计算实际数值`t`，然后将`t`平均分为100段，算出当前填充的段数，最后，别忘了加上最小值。

插件的参数说明：

```
var range2 = new LYRange('.custom-range', {
                                type: 2,  // 类型，2为自定义，1为input
                                min: 10,  // 最小值
                                max:100, // 最大值
                                defaultValue: 20,  // 当前值
                                onChange: function(range) {  // 监听滑块变
化
                                        console.log(range.value);  // 获取
滑块当前值
```

```
                                    }
});
```

源码路径：`WebDemo/JavaScript/range`

瀑布流

瀑布流

瀑布流是我们常见的功能，比如我们经常见到很多网站都有瀑布流功能，如淘宝、京东这些商品展示等等。

瀑布流其实就是指将很多图片大小不一，有顺序的排列在一起，随着页面滚动条向下滚动而不断加载。

优点：

- 操作简单（滚动鼠标）
- 节省空间
- 用户体验好

这一节主要讲解如何实现瀑布流？

要实现瀑布流，需要考虑几点：

- 排列方式
- 如何实现排列

1、创建模板

```
<div class="fallbox">
        <div class="fall-wrapper">
                <div class="fall-item">
```

```
                        <div class="fall-inner">
                                <div class="fall-media">
                                        <img src="../images/a.jpg" alt=""
 />
                                </div>
                                <div class="fall-intro">
                                        瀑布流
                                </div>
                        </div>
                </div>
        </div>
</div>
```

在上面的代码中：

- `fallbox` 表示一个瀑布流
- `fall-wrapper` 用来放置列表项
  - `fall-item` 表示一个列表项

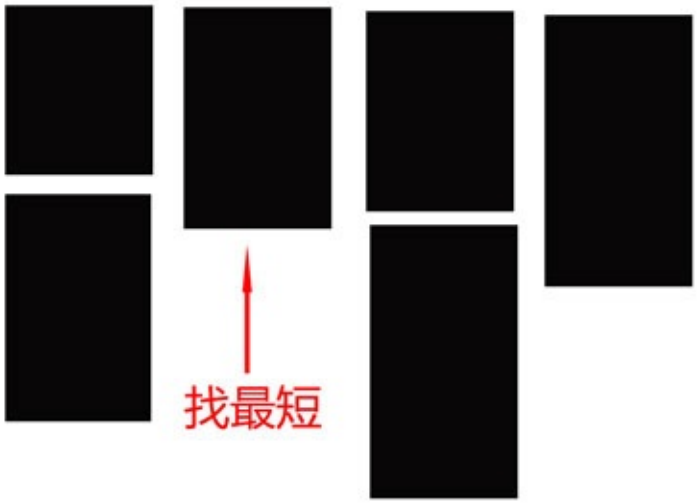
2、设置CSS样式

```
.fall-wrapper{
        position:relative;
}
.fall-item {
        position: absolute;
        opacity: 0;
}
```

这里的核心就是列表项采取 `绝对定位` ，其父元素采取绝对定位（当然，你也可以不设置父元素定位，那样会以body为主）

3、如何实现排列

瀑布流一般都是宽度一样，高度参差不齐，所以，实现瀑布流的关键点就是如何找到最短的一列，然后添加其他列表项（如果你的图片全都是一样高宽的，那你就可以省略这一步，直接一个个append就行了）。



瀑布流原理的实质就是不断获取最短的一列，然后改变新项的top和left，然后放置进去。

在这里，我们用一个数组来保存每一列当前的整体高度：

```
var iHeight = [];
```

初始化时：

```
for (var i = 0; i < n.params.cols; i++) {
        var col = i % n.params.cols;
        items[i].style.width = n.params.itemWidth + 'px';
        items[i].style.left = col * n.params.itemWidth + 'px';
        items[i].style.top = '0px';
        iHeight[i] = items[i].offsetHeight;
        items[i].classList.add('moveIn');
};
```

默认是分为5列，初始化时要获取开头5个的高度，然后保存到 `iHeight` 中。

每次加载新的数据时，都要重新计算一下每一列的整体高度：

```
n.refresh = function() {
        var items = n.container.querySelectorAll('.fall-item');
        for (var i = n.params.cols; i < items.length; i++) {
                if (items[i].classList.contains('news')) {
                        n.countHeight(items, i);
                        items[i].classList.remove('news');
                }
        };
};
```

```
n.countHeight = function(items, i) {
    //  判断是哪一列
        var col = i % n.params.cols;
        var minHeight = n.getMin(iHeight);
        var min = iHeight.indexOf(minHeight);
        items[i].style.width = n.params.itemWidth + 'px';
        items[i].style.left = min * n.params.itemWidth + 'px';
        items[i].style.top = minHeight + 'px';
        iHeight[min] = n.getMin(iHeight) + items[i].offsetHeight;
}
```

在上面的代码中，`getMax()`和`getMin()`方法分别用来计算一个数组中的最大值和最小值。

```
n.getMax = function(arr) {
        return Math.max.apply(Math, arr);
};
n.getMin = function(arr) {
        return Math.min.apply(Math, arr);
};
```

最后，当然少不了监听滚动：

```
window.addEventListener('scroll',
function() {
        var top = document.body.scrollTop;
        if (!n.isLoading && (top + window.innerHeight) >= n.getMin(iHeigh
t)) {
                n.params.onLoad && n.params.onLoad(n);
                n.refresh();
        };
},
false);
```

当`iHeight`数组中的最小值小于内容区的整体高度时，触发加载新数据方法。

这就是完整的瀑布流实现原理。

插件参数说明：

```
var fall = new WallFall('.fallbox', {
                                cols: 3,  // 分成几列
                                onLoad: function(fall) {  // 动态加载事件
                                        var data = [{
                                                title: '瀑布流',
                                                url : '../../common/Image
s/e.jpg'
```

```
                                        },{
                                                title: '瀑布流',
                                                url : '../../common/Image
s/f.jpg'
                                        },{
                                                title: '瀑布流',
                                                url : '../../common/Image
s/g.jpg'
                                        },{
                                                title: '瀑布流',
                                                url : '../../common/Image
s/h.jpg'
                                        }]
                                        fall.addItem(data); // 添加子项
                                }
                        });
```

源码路径：`WebDemo/JavaScript/fallwall`

下拉菜单

下拉菜单

下拉菜单是一个实现简单但实用的功能，不仅节省了占据空间，而且变化也是多样的。

1、创建模板

```
<div class="dropdown">
        <button class="btn btn-primary btn-circle">
                <i class="ion-android-more-vertical">
                </i>
        </button>
        <ul class="dropdown-menu pull-left">
                <li>左侧菜单</li>
                <li>左侧菜单</li>
                <li>左侧菜单</li>
        </ul>
</div>
```

类名`dropdown`的div是整个下拉菜单的外层框，包裹着一个初始显示内容和一个包裹着子菜单的容器（.dropdown-menu）。

2、设置CSS样式

```
.dropdown {
        position:relative;
        display:inline-block;
}
// 子菜单容器
.dropdown .dropdown-menu {
        position: absolute;
        top: 100%;
        background: #fff;
        list-style: none;
        color: #333;
        margin: 0;
        padding: 5px 0;
        z-index: 10000;
        border: 1px solid transparent;
    border-radius: 2px;
    -webkit-box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
        -webkit-transition: all .25s;
        -o-transition: all .25s;
```

```
        transition: all .25s;
        margin-top: 1px;
}
// 关闭时
.dropdown .dropdown-menu {
        -webkit-transform: scale(0);
        -ms-transform: scale(0);
        -o-transform: scale(0);
        transform: scale(0);
        opacity: 0;
        filter: alpha(opacity=0);
        display: block;
}
// 展开时
.dropdown.open .dropdown-menu {
        -webkit-transform: scale(1);
        -ms-transform: scale(1);
        -o-transform: scale(1);
        transform: scale(1);
        opacity: 1;
        filter: alpha(opacity=100);
}
```

在上面的代码中，我们使用类名`open`来展开和关闭子菜单，当包含类名dropdown的元素同时包含`open`时，展开子菜单，否则关闭。

当然，展开和关闭的效果也可以采取最简单的`display: none`和`display:block`互相切换。

3、脚本

我们需要通过点击事件来展开和关闭子菜单。

先创建一个`toggleDropdown()`方法，用来展开或关闭菜单：

```
function toggleDropdown() {
        var d = this.parentNode;
        for (var i = 0; i < dropdown.length; i++) {
                dropdown[i].classList.remove('open');
        };
        var classList = d.classList;
        if (classList.contains('open')) {
                classList.remove('open');
        } else {
                classList.add('open');
        }
};
```

在上面的代码中，我们只需简单的判断是否存在`open`就可以展开或关闭菜单。

上面的for循环是为了一次只有一个下拉菜单展开。

这里虽然实现了下拉菜单，但有些时候，你可能需要点击其他位置关闭下拉菜单的功能。

如何实现呢？这就需要用到`事件代理`了。

由于事件的冒泡，我们可以将子元素的点击事件添加到父元素中（或直接添加到document上）。

```
function closeDropdown(e) {
        var target = e.target;
        while (target) {
                if (target && target.nodeName == '#document') { [].forEac
h.call(dropdown,
                        function(d) {
                                d.classList.remove('open');
                        });
                } else if (target.classList.contains('dropdown')) {
                        break;
                }
                target = target.parentNode;
        }
}

document.addEventListener('click',closeDropdown,false);
```

target 事件属性可返回事件的目标节点（注意：并不一定是添加事件的那个元素）

在上面的代码中，我们通过不断地获取目标节点的父节点，也就是从目标节点向上遍历父节点、祖先节点等，如果有一个父节点包含你指定的类名（在这里是dropdown），表示点击的是下拉菜单；如果遍历到document，则表示你点击的位置并不在dropdown内，则关闭下拉菜单。

在这里，我还给下拉菜单添加了不同的展开位置：

```
.dropdown-menu.pull-right {
        right: 0;
        left: auto;
}

.dropdown-menu.pull-left {
        left: 0;
        right: auto;
}

.dropdown .dropdown-menu.pull-right {
        -webkit-transform-origin: top right;
```

```
        -moz-transform-origin: top right;
        -ms-transform-origin: top right;
        transform-origin: top right;
}

.dropdown .dropdown-menu.pull-left {
        -webkit-transform-origin: top left;
        -moz-transform-origin: top left;
        -ms-transform-origin: top left;
        transform-origin: top left;
}
```

`transform-origin`属性定义转换基点位置。

你也可以添加其他的展开或关闭动画。

源码路径：`WebDemo/JavaScript/dropdown`

幻灯片

幻灯片

本节来讲解一下幻灯片的实现。

实例效果：实例预览

`slide3D`是一款酷炫幻灯片插件，支持左右箭头和圆点按钮播放控制,支持多种不同的3D动态切换特效。

1、创建模板

```
<div class="container3D slide">
        <div class="wrapper3D">
                <div class="slide3D">
                        <img src="a.jpg" alt="" />
                </div>
        </div>
        <div class="slide3D-pagination">
        </div><div class="slide3D-prev-button"></div>
        <div class="slide3D-next-button"></div>
</div>
```

在上面的代码：

- `container3D`是一个幻灯片
- `wrapper3D`是幻灯片列表
  - `slide3D`是一个幻灯片选项
- `slide3D-pagination`是分页器容器
- `slide3D-prev-button`和`slide3D-next-button`是左右箭头。

2、设置CSS样式

```
.container3D {
        position: relative;
        width: 100%;
        height: 100%;
        z-index: 1;
        overflow: hidden;
}
.wrapper3D,
```

```
.slide3D,
.container3D .slide3D-pagination {
        display: -webkit-box;
        display: -moz-box;
        display: -ms-flexbox;
        display: -webkit-flex;
        display: flex;
}
```

在这里，我采取Flexbox布局，每个幻灯片选项自动向右排列（在以前，多数都是采取浮动float）

3、JavaScript

实现幻灯片的切换，实质是设置`wrapper3D`的`transform`，每次移动的距离都是一个幻灯片选项的宽度：

```
s.setTransform(s.wrapper, 'translate3d(-' + (s.params.width * index) + 'p
x, 0, 0)');
```

注意初始值是0，所以后面的移动是负值。

那如何实现循环切换呢？

其实也不难，实际就是将第一个幻灯片选项和最后一个幻灯片选项都复制一份，然后作为最后一个幻灯片选项和第一个幻灯片选项放置进`wrapper3D`里。

这样就实现了第一个幻灯片选项向左切换时，切换到最后一个；最后一个幻灯片选项向右切换时，切换到第一个。

当然，这并没有结束，我们还需要这样：

```
s.setTransitionDuration(s.wrapper, 0);
```

将过渡时间设置为0，然后将`wrapper3D`的`transform`值改变一下，也就是当切换由第一个切换到最后一个时，过渡结束后，要将`wrapper3D`的`transform`值设置为选项长度 * 选项宽度；当最后一个切换到第一个时，将`wrapper3D`的`transform`值设置为`-width`。

这是为什么呢？这是因为当过渡时间（transition-duration）为0时，切换会很快，我们肉眼是很难看到切换过程的。

4、切片幻灯片

对于切片幻灯片，其实就是将一张图片分成了多个切片，然后分别对它们进行动画处理。

选项卡

选项卡

这一节介绍一款简单实用的tab标签选项卡切换特效。

1、创建模板

```
<div class="sg-tabs-block sg-tabs-line">
        <div class="sg-tabs-nav">
                <div class="sg-button-tabs">
                        <a class="sg-button-tab">
                                热门文章
                        </a>
                        <a class="sg-button-tab">
                                最新文章
                        </a>
                        <a class="sg-button-tab">
                                一周热门
                        </a>
                </div>
        </div>
        <div class="sg-tabs-body">
                <div class="sg-tabs">
                        <div class="sg-tab active">
                                热门文章
                        </div>
                        <div class="sg-tab">
                                最新文章
                        </div>
                        <div class="sg-tab">
                                一周热门
```

```
                        </div>
                </div>
        </div>
</div>
```

在上面的代码中，`.sg-tabs-block` 的div用来包裹选项卡的tab按钮（`.sg-tabs-nav`）和内容区（`.sg-tabs-body`）。

2、设置CSS样式

```
// 使用flexbox布局

.sg-button-tabs {
        position: relative;
        display: -webkit-box;
        display: -moz-box;
        display: -ms-flexbox;
        display: -webkit-flex;
        display: flex;
        -webkit-box-pack: center;
        -webkit-align-items: center;
        align-items: center;
        border-bottom: 1px solid #d9d9d9;
}

// 内容区简单的显示与隐藏

.sg-tabs .sg-tab {
        display: none;
}

.sg-tabs .sg-tab.active {
        display: block;
}
```

3、动画

第一个选项卡的下划线效果其实动态添加了一个类名为 `sg-tabs-indicator` 的div，然后根据切换tab来改变其位置：


```
indicator.style.left = w * index + 'px';
```

第二个选项卡只是简单的通过样式变化：

```
.tab-scale .sg-button-tab:after {
        content: "";
        position: absolute;
        bottom: 0;
        left: 0;
        width: 100%;
        height: 2px;
        background: #1ABC9C;
        -webkit-transform: scale(0);
        transform: scale(0);
        -webkit-transition: transform .3s;
        transition: transform .3s;
}

.tab-scale .sg-button-tab.active:after {
        -webkit-transform: scale(1);
        transform: scale(1);
}
```

在上面的代码中，我们通过 `scale()` 改变伪元素的缩放比例来实现切换动画。

源码路径：`WebDemo/JavaScript/tab`

全屏滚动

全屏滚动

我们经常能看到全屏网站，这些网站用几幅很大的图片或背景色做背景，然后添加一些简单的内容，显得格外的高端大气上档次。比如：百度百科

这种全屏效果，我们不仅能通过点击分页器跳转，而且可以通过鼠标滚轮来切换页面。

这一节我们就来看看全屏滚动的实现。

1、创建模板

```
<div class="fullscreen-container">
        <div class="fullscreen-wrapper">
                <div class="fullscreen-slide" style="background:yellowgre
en">
                </div>
                <div class="fullscreen-slide" style="background:blue">
                </div>
        </div>
</div>
```

`.fullscreen-container` 是整个滚动的外框，`.fullscreen-wrapper` 用来放置所有子页面，且用它来进行页面切换，每个 `.fullscreen-slide` 都表示一个页面。

2、设置CSS样式

```
html,body{
        height:100%;
        overflow: hidden;
}
```

这个很重要，如果省略 `height:100%` ，有可能无法全屏；如果忽略 `overflow:hidden` ，则会出现滚动条。

```
.fullscreen-container,
.fullscreen-wrapper,
.fullscreen-slide{
        position: relative;
        width: 100%;
        height: 100%;
        overflow: hidden;
```

```
}
```

3、添加滚轮事件

IE6, Opera9+, Safari2+以及Firefox1+均支持“onmousewheel”事件，在FF 3.x中，与之相当的是“DOMMouseScroll”事件。“onmousewheel”作为事件名，不为其识别。所以，为了保证能在每个浏览器中都能运行，就需要针对不同的浏览器来绑定不同的事件。

```
var type = (/Firefox/i.test(navigator.userAgent)) ? "DOMMouseScroll": "mo
usewheel"
if (document.attachEvent) {
        document.attachEvent("on" + type, nf.scrollFunc);
} else if (document.addEventListener) {
        document.addEventListener(type, nf.scrollFunc, false);
}
```

上面的代码给document绑定了鼠标滚轮事件，但是，鼠标每向上或向下移动一次，滚动了多少？

当该事件触发时，在非Firefox浏览器中，记录其距离的是“wheelDelta”，它返回的总是120的倍数（120表明mouse向上滚动，-120表明鼠标向下滚动）。在FireFox中，记录其滚动距离的是“detail”属性，它返回的是3的倍数（3表明mouse向下滚动，-3表明mouse向上滚动）。

```
nf.scrollFunc = function(e) {
        var direct = 0;
        e = e || window.event;
        if (!nf.isScroll) {
                if (e.wheelDelta) { //判断浏览器IE，谷歌滑轮事件
                        if (e.wheelDelta > 0) { //当滑轮向上滚动时
                                nf.scrollPrev();
                        } else if (e.wheelDelta < 0) { //当滑轮向下滚动时
                                nf.scrollNext();
                        };
                } else if (e.detail) { //Firefox滑轮事件
                        if (e.detail < 0) { //当滑轮向上滚动时
                                nf.scrollPrev();
                        } else if (e.detail > 0) { //当滑轮向下滚动时
                                nf.scrollNext();
                        };
                };
        };
};
```

`nf.isScroll`用来锁定，防止当前页面还没滚动结束，又进入另一个页面切换。

上面的 `nf.scrollPrev()` 和 `nf.scrollNext()` 两个方法用来页面切换，分别表示切换到上一个页面和切换到下一个页面。

其实页面之间的切换，实质就是改变 `.fullscreen-wrapper` 的 `transform: translate3d(0px, -1294px, 0px);` 中第二个参数的值，也就是y轴，每次页面的切换，就是加一个屏幕高或减一个屏幕高。

全屏滚动就是这么简单。

插件参数说明：

```
var nf = new NFullscreen(".fullscreen-container", {
                                    pagination: true,  // 是否显示分页器
                                    pageChangeEnd: function(fullscreen) {  //
 每一页切换完成时

                                    },
                                    paginationClickable: true  // 分页器是否可
点击
                            });
```

源码路径：`WebDemo/JavaScript/fullpage`

富文本编辑器

富文本编辑器

富文本编辑器是一种所见即所得的文本编辑器，我们不仅可以输入简单的文字，而且拥有可以修改格式，输入表情等多种功能。

富文本编辑器的编写看似很神秘很复杂，可是当你了解其开发原理时，却会发现并不难，在这一节，我们就来看看富文本编辑器的实现是怎样的？

1、创建模板

`contenteditable`属性的出现，让我们可以将任何元素设置成可编辑状态。

```
<div id="editor" contenteditable="true"></div>
```

当然，你也可以采取iframe方式，将其`designMode`属性设置为`on`。

2、获取和恢复焦点位置

开发富文本编辑器的一个重点就是如何获取到焦点的位置。

这个看起来很复杂，其实并不难。

先来了解Range对象：

Range 对象表示文档的连续范围区域，简单的说就是高亮选区。一个Range的开始点和结束点位置任意，开始点和结束点也可以是一样的（空Range）。最常见的就是用户在浏览器窗口中用鼠标拖动选中的区域。

不过，不同的浏览器，Range对象是不一样的，在Chrome、Mozilla、Safari等主流浏览器上，Range属于selection对象（表示range范围），而在IE下，Range属于textRange对象（表示文本范围）。

获取当前焦点位置的代码如下：

```
getCurrentRange: function() {
        //获取当前range
```

```
        if (window.getSelection) {
                //使用 window.getSelection() 方法获取鼠标划取部分的起始位置和结束位置
                var sel = window.getSelection();
                if (sel.rangeCount > 0) {
                        //通过selection对象的getRangeAt方法来获取selection对象的某个Range对象
                        return sel.getRangeAt(0);
                }

        } else if (document.selection) {
                var sel = document.selection;
                return sel.createRange();
        }
        return null;
}
```

保存焦点位置（为了后续恢复）：

```
selectedRange = r.selections.getCurrentRange();
```

当你点击了工具条时，当前焦点也就改变了，所以我们需要恢复前一个焦点位置：

```
restoreSelection: function() { //重置为上个range
        var selection = window.getSelection();
        if (selectedRange) {
                try {
                        selection.removeAllRanges();
                } catch(ex) {
                        document.body.createTextRange().select();
                        document.selection.empty();
                };
                selection.addRange(selectedRange);
        }
}
```

当然，我们需要在鼠标点击或键盘输入时，不断的保存新位置，所以我们要给编辑框绑定 `keyup` 和 `mouseup` 事件：

```
addEvent(et, "keyup",
function(e) {
        r.selections.saveSelection();
},
false);
addEvent(et, "mouseup",
function(e) {
        r.selections.saveSelection();
},
```

```
false);
```

3、添加格式化工具

当document对象被转换为设计模式的时候（选中，设置contentEditable等），document对象就会提供一个execCommand方法，通过给这个方法传递参数命令可以操作可编辑区域的内容。这个方法的命令大多数是对document选中区域的操作 (如bold, italics等), 也可以插入一个元素(如增加一个a链接或图片) 或者修改一个完整行 (如缩进).。当元素被设置了contentEditable，通过执行execCommand() 方法可以对当前活动元素进行很多操作。

语法：

```
bool = document.execCommand(aCommandName, aShowDefaultUI, aValueArgument)
```

参数说明：

- aCommandName：String类型，命令名称
- aShowDefaultUI：Boolean类型，默认为false，是否展示用户界面，一般设为false。Mozilla没有实现。
- aValueArgument：String类型，一些命令需要额外的参数值（比如insertHTML需要提供HTML内容），默认为null，一般不需要参数时都使

比如我们要改变背景色，可以这样：

```
document.execCommand('backColor',false, 'red');
```

字体加粗，可以这样设置：

```
document.execCommand('Bold','false',null);
```

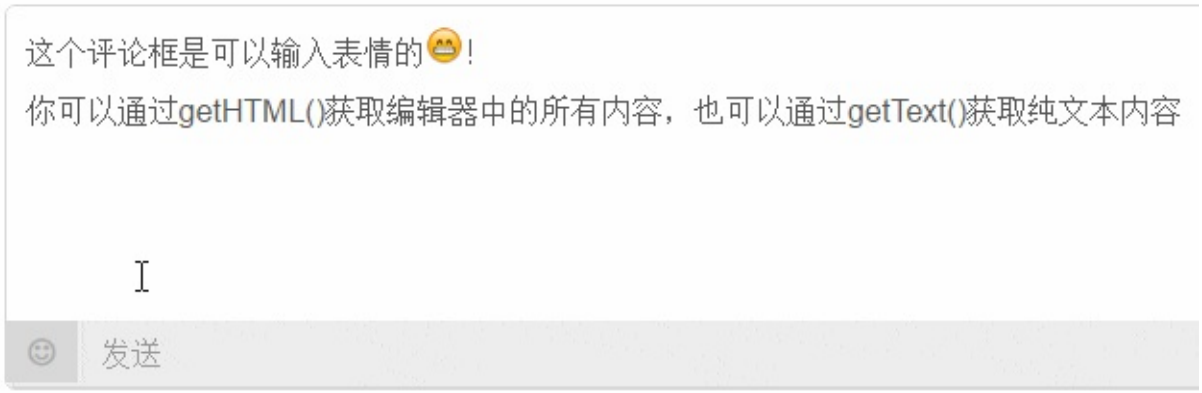
当然，execCommand()的命令不止这几个，更多命令：聊聊开发富文本编辑器：execCommand方法

源码路径：`WebDemo/JavaScript/editor`

带表情输入的评论框

带表情输入的评论框

现在的网站在评论框中都添加了表情输入，其实这相当于一个简单的富文本编辑器。



如果你不明白富文本编辑器的原理，可以看上一章节。

先来看看一段代码：

```
document.execCommand('insertHTML', true, img);
```

`execCommand()` 方法的 `insertHTML` 命令表示往焦点位置插入一段HTML，在这里的img就是一个img元素（ `<img src=""/>` ）。

源码路径： `WebDemo/JavaScript/comment`

图片懒加载

图片懒加载

目前，网络上各大论坛，尤其是一些图片类型的网站上，在图片加载时均采用了一种名为懒加载的方式，具体表现为，当页面被请求时，只加载可视区域的图片，其它部分的图片则不加载，只有这些图片出现在可视区域时才会动态加载这些图片，从而节约了网络带宽和提高了初次加载的速度，具体实现的技术并不复杂。

1、放置懒加载图片

```
<img data-src="../images/example.jpg"  class="lazyload" alt="" />
```

在上面的代码中，具有`lazyload`类名的img表示需要懒加载，我们并不会直接给img元素添加src属性，也就是让图片并不会立即加载。

当然，你也可以添加一个默认图片：

```
<img data-src="../images/example.jpg" src="../images/loading5.gif" class=
"lazyload" alt="" />
```

2、添加滚动事件

我们需要绑定滚动事件：

```
window.addEventListener('scroll',_loadImage);
```

在上面的_loadImage方法就是获取页面中所有需要懒加载的图片：

```
function _loadImage() {
        var imgList = document.querySelectorAll(".lazyload");
        for (var i = 0; i < imgList.length; i++) {
                var el = imgList[i];
                if (_isToShow(el)) {
                        var imgUrl = el.getAttribute("data-src");
                        el.setAttribute("src", imgUrl);
                        el.className = el.className.replace("lazyload", "
loaded");
                }
        };
```

```
};
```

那如何判断元素是否在可视区呢？或许你已经发现了，在`_loadImage()`方法中，有一个`_isToShow()`方法，传递给它的参数是每一个懒加载图片。

```
function _isToShow(el) {
        var coords = el.getBoundingClientRect();
        var wHeight = window.innerHeight || document.documentElement.clie
ntHeight;
        return (coords.top >= 0 && coords.left >= 0 && coords.top <= wHei
ght);
};
```

`getBoundingClientRect()`这个方法返回一个矩形对象，包含四个属性：left、top、right和bottom。分别表示元素各边与页面上边和左边的距离。

获取到了元素与页面上边和左边的距离，后续就很简单了，我们只需判断:
`coords.top >= 0 && coords.left >= 0 && coords.top <= wHeight`

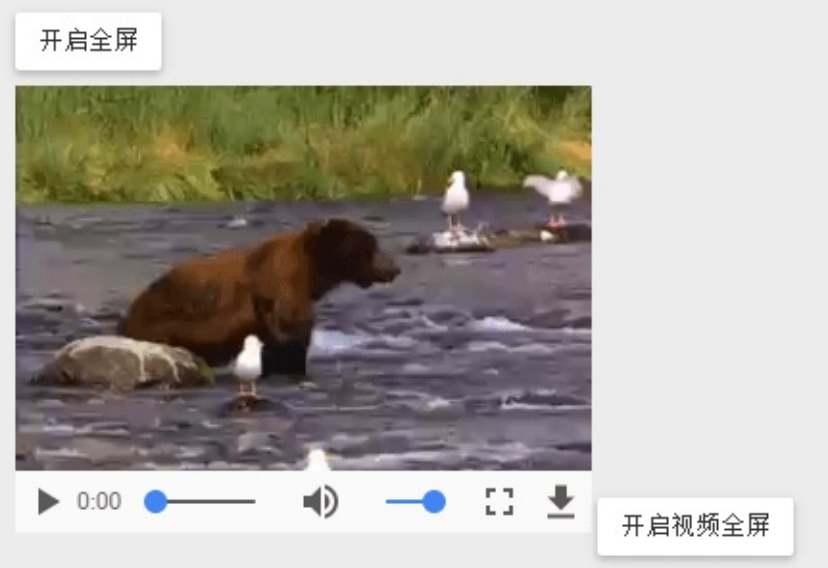
注意一点的是：我们需要将已经加载的图片去掉`lazyload`类名：
`el.className = el.className.replace("lazyload", "loaded");`
虽然留着也不会出错，但影响效率。

源码路径：`WebDemo/JavaScript/lazyloader`

开启全屏

开启全屏

开启视频全屏，这是我们日常看电影最常用的，而有些管理系统一般也会添加全屏功能，实现起来也不难，不过兼容性问题要考虑不少。



这一节，我们就来看看如何开启全屏和关闭全屏。

首先来了解几个属性和方法：

1、开启全屏

Element节点的`requestFullscreen`方法，可以使得这个节点全屏

我们可以这样开启全屏：

```
var docElm = elem || document.documentElement;
if (docElm.requestFullscreen) {
        docElm.requestFullscreen();
} else if (docElm.mozRequestFullScreen) {
        docElm.mozRequestFullScreen();
} else if (docElm.webkitRequestFullScreen) {
        docElm.webkitRequestFullScreen();
} else if (docElm.msRequestFullscreen) {
        docElm.msRequestFullscreen();
};
```

运行到这里，Gecko 与 WebKit 两个实现中出现了一个值得注意的区别：
Gecko 会为元素自动添加 CSS 使其伸展以便铺满屏幕： "width: 100%; height: 100%"。
WebKit 则不会这么做；它会让全屏的元素以原始尺寸居中到屏幕中央，其余部分变为黑色。
所以为了在 WebKit 下也达到与 Gecko 同样的全屏效果，你需要手动为元素增加 CSS 规则"width: 100%; height: 100%;"：

```
/* html */
:-webkit-full-screen {}

:-moz-fullscreen {}

:fullscreen {}

:-webkit-full-screen video {
  width: 100%;
  height: 100%;
}
```

2、关闭全屏

Document对象的`exitFullscreen`方法用于取消全屏。

```
if (document.exitFullscreen) {
        document.exitFullscreen();
} else if (document.mozCancelFullScreen) {
        document.mozCancelFullScreen();
} else if (document.webkitCancelFullScreen) {
        document.webkitCancelFullScreen();
} else if (document.msExitFullscreen) {
        document.msExitFullscreen();
}
```

用户手动按下ESC键或F11键，也可以退出全屏键。此外，加载新的页面，或者切换tab，或者从浏览器转向其他应用（按下Alt-Tab），也会导致退出全屏状态。

3、判断目前是否处于全屏状态

document.fullscreenElement: 当前处于全屏状态的元素

我们可以将开启全屏和关闭全屏的方法合并：

```
function toggleFullScreen(elem) {
        if (!document.fullscreenElement && !document.mozFullScreenElement
```

```
  && !document.webkitFullscreenElement) {
                var docElm = elem || document.documentElement;
                if (docElm.requestFullscreen) {
                        docElm.requestFullscreen();
                } else if (docElm.mozRequestFullScreen) {
                        docElm.mozRequestFullScreen();
                } else if (docElm.webkitRequestFullScreen) {
                        docElm.webkitRequestFullScreen();
                } else if (docElm.msRequestFullscreen) {
                        docElm.msRequestFullscreen();
                };
        } else {
                if (document.exitFullscreen) {
                        document.exitFullscreen();
                } else if (document.mozCancelFullScreen) {
                        document.mozCancelFullScreen();
                } else if (document.webkitCancelFullScreen) {
                        document.webkitCancelFullScreen();
                } else if (document.msExitFullscreen) {
                        document.msExitFullscreen();
                }
        };
};
```

比如你要给一个视频添加全屏：

```
toggleFullScreen(document.getElementById('video'));
```

4、全屏事件

`fullscreenchange` 事件：浏览器进入或离开全屏时触发。
fullscreenerror事件：浏览器无法进入全屏时触发，可能是技术原因，也可能是用户拒绝。

```
document.addEventListener('fullscreenchange',function(){
  if(document.fullscreenElement){
    //进入全屏
  }else{
    //退出全屏
  }
},false);
```

由于兼容性，使用全屏事件时，我们需要加上不同的前缀：

```
function getWebType() {
        var type = ['webkit', 'moz', 'o', 'ms'];
        var cur = '';
        type.forEach(function(t) {
                var mo = t + 'Transform';
```

```
                if (mo in document.createElement('div').style) {
                        cur = t;
                }
        });
        return cur;
};
var isOpen = false;
var prefix = getWebType();
addEvent(document, prefix + 'fullscreenchange',
function() {
        if (!isOpen) {
                document.getElementById('open').innerHTML = '关闭全屏';
                isOpen = true;
        } else {
                document.getElementById('open').innerHTML = '开启全屏';
                isOpen = false;
        }
},
false);
```

在上面的代码中，采取如下方式判断添加哪一个前缀：

```
var type = ['webkit', 'moz', 'o', 'ms'];
var mo = t + 'Transform';
if (mo in document.createElement('div').style) {
        cur = t;
}
```

源码路径：`WebDemo/JavaScript/fullscreen`

返回顶部

返回顶部

有些时候，当网页过长时，为了方便用户，我们就需要提供一个返回顶部的功能。

实现起来难吗？不难。

1、放置返回顶部按钮

```
<button type="button" class="btn btn-danger btn-circle scrollTop">
返回顶部
</button>
```

按钮样式可以自定义。

2、添加点击事件

先来看看一个属性和一个方法：

```
document.body.scrollTop || document.documentElement.scrollTop
```

在Firefox或Chrome浏览器的控制台可以查看document.body 对应于页面中 `<body></body>` 部分的元素，而document.documentElement则相当于整个HTML，说明浏览器在解释渲染后的页面位置范围是存在不同的。

FF、Opera和IE浏览器认为在客户端浏览器展示的页面的内容对应于整个HTML，所以使用document.documentElement来代表，相应的滚动距离则通过 `document.documentElement.scrollLeft` 和 `document.documentElement.scrollTop` 来获取，而Safari和Chrome浏览器则认为页面开始于body部分，从而相应的滚动距离用 `document.body.scrollLeft` 和 `document.body.scrollTop` 来获取。

window.scrollTo(x, y)

`scrollTo()` 方法可把内容滚动到指定的坐标。
x：必需。要在窗口文档显示区左上角显示的文档的 x 坐标。
y：必需。要在窗口文档显示区左上角显示的文档的 y 坐标。

下面添加点击滚动回顶部的代码：

```
var scrollTop = document.querySelector('.scrollTop');
scrollTop.addEventListener('click',
function() {
        var top = document.body.scrollTop || document.documentElement.scr
ollTop;
        var ax = 0;
        if (top > 0) {
                scrollTop.timer = setInterval(function() {
                        if (top <= 0) {
                                clearInterval(scrollTop.timer);
                        } else {
                                ax += 20;
                                top -= ax;
                                window.scrollTo(0, top);
                        };
                },
                100);
        }
});
```

源码路径：`WebDemo/JavaScript/toTop`

上传图片预览

上传图片预览

在以前，我们只能将图片上传到服务器，然后服务器给我们传回一个地址，然后我们才能看到图片的预览效果。但是现在，我们能使用纯JS来直接预览，而无需上传到服务器。

这一节将来讲解如何实现?

1、file

首先我们需要一个上传图片的input file：

```
<input type="file" id="file" accept="image/*" />
```

`accept`属性用来设置上传格式，你也可以加上`multiple`属性，让其可以多选。

上传图片预览美化的方面你可以《CSS开发实战》中的上传图片组件美化一节。

2、获取选择图片

我们需要绑定`change`事件来监听input的变化，也就是用户选择图片的变化：

```
document.getElementById('file').addEventListener('change', handleFiles);
```

`handleFiles`方法是监听函数：

```
function handleFiles(e) {
        var files = e.target.files;
        for (var i = 0; i < files.length; i++) {
                var file = files[i];
                var imageType = /^image\//;
                if (!imageType.test(file.type)) continue;
                var review = document.createElement('div');
                review.className = 'review';
                var img = document.createElement('img');
                review.appendChild(img);
                document.querySelector('.box').appendChild(review);
                try {
                        var URL = window.URL || window.webkitURL;
                        var imgURL = URL.createObjectURL(file);
                        img.src = imgURL;
```

```
                } catch(e) {
                        var reader = new FileReader();
                        reader.onload = function(e) {
                                img.src = e.target.result;
                        };
                        reader.readAsDataURL(file);
                }

        }
}
```

在上面的代码中，`e.target.files`是一个FileList对象。

FileList对象都是一组文件对象的集合，而每个文件对象则拥有下列的属性：
name – 文件名（不包含路径）
type – 文件的MIME类型（小写）
size – 文件的尺寸（单位为字节）
lastModifiedDate 为上传文件的最后修改时间

`URL.createObjectURL()`方法会根据传入的参数创建一个指向该参数对象的URL. 这个URL的生命仅存在于它被创建的这个文档里. 新的对象URL指向执行的File对象或者是Blob对象.

`FileReader`对象的`readAsDataURL`方法可以将读取到的文件编码成Data URL。

源码路径：`WebDemo/JavaScript/uploadReview`

走马灯

跑马灯

这是一个简单的跑马灯效果。该跑马灯特效使文本从右向左不停循环,当鼠标放到跑马灯上的文字时,跑马灯会暂停运动。

```
item1    item2    item3    item4    item5          i
```

1、跑马灯布局

```
<div class="marquee-box">
        <div class="marquee-inner">
          // 内容区
        </div>
</div>
```

比如我的例子：

```
<div class="marquee-box">
        <div class="marquee-inner">
                <ul>
                        <li>item1</li>
                        <li>item2</li>
                        <li>item3</li>
                        <li>item4</li>
                        <li>item5</li>
                </ul>
        </div>
</div>
```

2、设置样式

样式也很简单：

```
.marquee-box {
        position: relative;
        width: 400px;
        height: 40px;
        background: #eee;
        margin: 10px;
        padding: 0 10px;
```

```
        line-height: 40px;
        overflow: hidden;
}

.marquee-inner {
        position: absolute;
        top: 0;
        left: 0;
}
```

`marquee-box`的宽高你可以自定义，但是一定要设置宽度，同时设置`position:relative`和`overflow:hidden`。

3、JavaScript

我们使用JavaScript让跑马灯动起来。

```
// 这是为了兼容transform
var setTransform = function(element, animation) {
        element.style.webkitTransform = animation;
        element.style.mozTransform = animation;
        element.style.oTransform = animation;
        element.style.msTransform = animation;
        element.style.transform = animation;
};

var marquee = document.querySelectorAll('.marquee-box');
for (var i = 0; i < marquee.length; i++) {
        var m = marquee[i];
        var inner = m.querySelector('.marquee-inner');
        var width = inner.offsetWidth;
        var bWidth = m.offsetWidth;
        if (bWidth < width * 2) {
                var clone = inner.cloneNode(true);
                m.appendChild(clone);
                // 跑动速度
                var ax = 3;
                setTransform(clone, 'translate(' + width + 'px,0)');
                inner.translateX = 0;
                clone.translateX = width;
                anim(m, inner, clone, ax, width);

        }
}
```

在上面的代码中，为了实现不间断地跑，我们使用DOM节点的`cloneNode(true)`方法将`marquee-inner`里的内容克隆一份，然后添加进`marquee-box`，同时让其的位置`translate`到其宽度处。

上面的`anim()`方法就是动画方法：

```
function anim(m, a, b, s, w) {
        m.timer = setInterval(function() {
                a.translateX -= s;
                b.translateX -= s;
                if (a.translateX <= -w) {
                        a.translateX = w;
                };
                if (b.translateX <= -w) {
                        b.translateX = w;
                };
                setTransform(a, 'translate(' + a.translateX + 'px,0)');
                setTransform(b, 'translate(' + b.translateX + 'px,0)');
        },
        80);
};
```

五个参数，第一个是当前的`marquee-box`；第二个是第一个`marquee-inner`；第三个是第二个也就是克隆的`marquee-inner`；第四个是跑动速度；第五个是内容宽度。

在定时器中，我们让两个`marquee-inner`同时跑动，同时用参数`translateX`分别来记录当前跑动的值。

实质跑动是：第一个（0 -> -width -> width)；第二个（ width -> 0 -> -width -> width ）

我们还可以添加鼠标事件，当鼠标移到跑马灯上时停止，移开后继续跑：

```
m.addEventListener('mouseover',
function() {
        clearInterval(this.timer);
});
m.addEventListener('mouseout',
function() {
        anim(m, inner, clone, ax, width);
});
```

一个简单的跑马灯就这样诞生了！

源码路径：`WebDemo/JavaScript/marquee`

万年历

万年历

要实现万年历，就有必要来了解一下Date对象的一些属性和方法。

Date()

```
var curDate=new Date()
```

Date实例的方法

- getFullYear()：返回年
- getMonth()：返回月（1-12）
- getDate()：返回一个月中的某一天（1-31）
- getDay()：返回一周中的某一天（0-6，0是星期天）

- setFullYear()：设置年
- setMonth()：设置月
- setDate()：设置天

了解了上面的方法，再来说说几个技巧：

获取当月有多少天

```
var curDate = new Date();
curDate.setDate(0);
var dd = curDate.getDate();
```

当`setDate()`的参数为0时，表示为上一个月的最后一天，所以你可以获取一个月的天数。

比如你要获取7月的天数，你只需将月份设置为8月份，然后`setDate(0)`，就可以获取7月份的天数了。

获取前一天或后一天

```
function getSomeDay(day, add) {
        var now = new Date(day);
        now.setDate(now.getDate() - add);
}
```

`getSomeDay()`方法接受两个参数，第一个是日期字符串，第二个是当前日期的前几天或后几天的数字。

了解上面的方法和技巧后，相信你可以获取得到天数、一个月的第一天是星期几和前几天或后几天的日期浏览。

现在来讲讲万年历的绘制。

从上面的示例图中，我们分为了七列，为什么呢？因为一周有七天。

现在最重要的是如何知道第一天在哪个格呢？

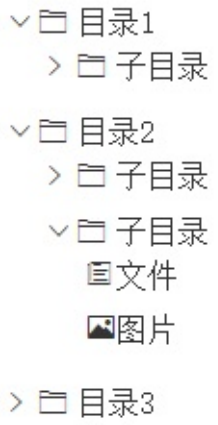
其实很简单，你只需获取到当月的第一天是星期几，然后空出前面几格，接着排列下去就行了。比如第一天是星期3，如果你是以星期天为一周的第一天，就需要空出3格，从第四格开始排列。

就是这么简单！

源码路径：`WebDemo/JavaScript/calendar`

树形菜单

树形菜单



这一节讲解简单的树形菜单。

1、创建模板

```
<div class="tree">
        <ul>
                <li class="collapsable open">
                        <span>目录2</span>
                        <ul>
                                <li class="collapsable">
                                        <span>子目录</span>
                                        <ul>
                                                <li class="last">
                                                        <span>文件</span>
                                                </li>
                                        </ul>
                                </li>
                                <li class="collapsable open">
                                        <span>子目录</span>
                                        <ul>
                                                <li class="last">
                                                        <span>文件</span>
                                                </li>
                                                <li class="last">
                                                        <span class="imag
e">图片</span>
                                                </li>
                                        </ul>
                                </li>
                        </ul>
                </li>
```

```
        </ul>
</div>
```

在上面的代码中：

- `tree` 表示一个树形菜单列表
  - `li.collapsable` 表示一个树形菜单
  - `li.coollapsable` 里面的 `ul` 表示树形菜单中的子菜单
  - `open` 表示展开
  - `last` 表示最底层的菜单
  - `image` 表示图片

2、设置CSS样式

```
.tree ul li.collapsable>span {
    position: relative;
    padding-left: 35px;
    margin: 5px 0;
}
.tree ul li.collapsable>span:before{
        content: "\f3d3";
        position:absolute;
        top:0;
        left:0;
}
.tree ul li.collapsable>span:after{
        content:"\f434";
        position:absolute;
        top:0;
        left:15px;
}
.tree ul li.collapsable.open>span:before{
        content:"\f3d0";
}
.tree ul li.collapsable.open{
        height:auto;
}
```

在上面的代码中，我们通过伪元素 `:before` 和 `:after` 来放置图标，`padding-left: 35px` 就是空出图标位置。

3、展开与关闭

```
+function(){
        var trees = document.querySelectorAll('.tree');
        for(var i = 0; i < trees.length; i++){
```

```
                var tree = trees[i];
                var folder = tree.querySelectorAll('.collapsable>span');
                for(var j = 0; j < folder.length; j++) {
                        var f = folder[j];
                        addEvent(f, 'click', toggle, false);
                }
        };
        function toggle(e){
                e.stopPropagation();
                var cl = this.parentNode.classList;
                cl.toggle('open');
        }
}();
```

在这里，我们通过简单的添加 `open` 来展开或关闭树形菜单。

DOM节点的 `classList` 中的 `toggle()` 方法，用来在元素中切换类名（和jQuer中的 `toggleClass()` 作用一样）。

源码路径： `WebDemo/JavaScript/tree`

旋转加载

圆形旋转加载

先来了解一个CSS 属性：`clip`

通过对元素进行剪切来控制元素的可显示区域，默认情况下，元素是不进行任何剪切的，但是也有可能剪切区域也显式的设置了clip属性

`clip` 支持图形方法，当前只支持 rect () 属性。

rect有四个值，顺序和margin、padding以及border具有一样的标准。

```
clip:rect (top, right, bottom, left)
```

注意：rect()属性的top和bottom指定的偏移量是从元素盒子顶部边缘算起的；left和right指定的偏移量是从元素盒子左边边缘算起的

这一节中的圆形旋转加载，就是利用`clip`来实现的。

1、布局

```
<div class="circleloading">
        <div class="left"></div>
        <div class="right"></div>
        <img src="../images/t_balloon_logo.png" alt="" />
</div>
```

2、设置样式

```
.circleloading {
        position: relative;
        width: 100px;
```

```
        height: 100px;
        text-align: center;
        overflow: hidden;
        margin-top: -60px;
}

.circleloading img {
        position: absolute;
        top: 10px;
        left: 10px;
        width: 80px;
        height: 80px;
        z - index: 10;
}

.circleloading,
.circleloading img,
.left,
.right {
        border-radius: 50% ;
}

.left,
.right {
        position: absolute;
        top: 0;
        left: 0;
        width: 100% ;
        height: 100% ;
        clip: rect(0, 50px, 100px, 0);
        background: #fff;
}

.left {
        z - index: 7;
        background: #afe4dd;
}

.right {
        z - index: 8;
}
```

3、旋转动画

```
var left = document.querySelector('.left');
var right = document.querySelector('.right');
var range = 10;
var timer = null;
timer = setInterval(function() {
        range += 10;
        if (range <= 180) {
                left.style.webkitTransform = 'rotate(' + range + 'deg)';
                left.style.transform = 'rotate(' + range + 'deg)';
```

```
        } else {
                right.style.background = '#afe4dd';
                if (range <= 360) {
                        right.style.webkitTransform = 'rotate(' + range +
 'deg)';
                        right.style.transform = 'rotate(' + range + 'deg)
';
                } else {
                        clearInterval(timer);

                };
        };
},
20);
```

源码路径：`WebDemo/JavaScript/circleLoading`

固定头的表格

固定头的表格

1、固定头表格

我们来重点介绍一下这个。

由于tbody里的td会受thead里的td的宽度影响，所以我们看到的表格都是对齐的。

但有些时候，我们需要固定thead，只让tbody里的区域滚动。

你可能想到的是将thead和tbody加上下面的定位：

```
thead,tbody{
    position:absolute;
}
```

然后设置tbody:

```
tbody{
    top: 40px;  /*假如thead高为40px*/
    overflow:hidden;
    overflow-y: auto;
    height:200px;
}
```

结果会是怎样呢？看下图：

确实头固定了，tbody区域也可以滚动了，可是杯具的是，头部的表格和tbody里的表格不对齐了。

为神马呢？这是因为绝对定位破坏了thead和tbody之间的关联性。

那该如何实现固定头表格？

```
<div class="table-fixed">
        <div class="table-fixed-box">
                <table cellpadding="0" cellspacing="0" data-minHeight="10
0">
                        <thead>
                                <tr>
```

```
                                        <th>    标题标题标题
                                        </th>
                                </tr>
                        </thead>
                        <tbody>
                                <tr>
                                        <td>
                                                表格
                                        </td>
                                </tr>
                        </tbody>
                </table>
        </div>
</div>
```

类名为table-fixed-box的div是为了实现滚动条：

```
.table-fixed-box{
        overflow:hidden;
        overflow-y:auto;
}
```

接下来，我们需要借助JavaScript来计算thead里的每个th的宽度。

看看生成的html布局是怎样的：

```
<div class="table-fixed" style="padding-top: 39px;">
        <div class="fixed-table-header">
        </div>
        <div class="table-fixed-box" style="height: 100px;">
                <table cellpadding="0" cellspacing="0" data-minheight="10
0">
                        <thead>
                                <tr>
                                        <th>
                                                <div class="th-inner" sty
le="width: 113px;">
                                                        标题标题标题
                                                </div>
                                                <div style="width: 113px;
">标题标题标题</div>
                                        </th>

                                </tr>
                        </thead>
                        <tbody>
                                <tr>
                                        <td>
                                                表格
                                        </td>
                                </tr>
```

```
                        </tbody>
                </table>
        </div>
</div>
```

在上面的代码中，多了`<div class="fixed-table-header"></div>`，这是由于多了滚动条，放置这个是为了方便获取最终整个表格的宽度，样式如下：

```
.fixed-table-header {
    height: 38px;
    position: absolute;
    top: 0;
    right: 0;
    left: 0;
    border-bottom: 1px solid #dddddd;
    border-radius: 4px 4px 0 0;
    -webkit-border-radius: 4px 4px 0 0;
    -moz-border-radius: 4px 4px 0 0;
}
```

同时，你会发现thead里的th内容也变了：

```
<th>标题标题标题</th>
```

替换成了这样：

```
<th>
        <div class="th-inner" style="width: 113px;">
                标题标题标题
        </div>
        <div style="width: 113px;">标题标题标题</div>  // 为了保留thead里每个
th的高度而创建
</th>
```

看看样式先：

```
.table-fixed thead th .th-inner {
    position: absolute;
    top: 0;
    padding: 8px;
    line-height: 22px;
    vertical-align: top;
    border-top:1px solid #d9d9d9;
    border-right:1px solid #d9d9d9;
}
```

这段样式的意思就是让类名为.th-inner的元素定位到头部，由于这里，我们只给.table-fixed的div加了position定位，所以前者是根据这个来定位的，也就跑到了最顶部。

最后，我们还需要将thead隐藏起来，注意，不是使用display:none，而是这样：

```
.table-fixed thead th{
        height:0;
        padding:0 !important;
}
```

这样我们隐藏了thead，但宽度依然在。

固定头表格的逻辑就是这样的，当然，方法还有很多，比如使用div布局等等。

源码路径：`WebDemo/JavaScript/table`

圆形水波纹加载进度

圆形水波纹加载进度

要实现这种水波纹加载，我们可以选择用canvas。

当然，这里并不是使用canvas，而是使用简单的CSS和JS来实现。

1、布局

```
<div class="wave">
        <div class="turn">
        </div>
</div>
```

在上面的代码：

- `wave` 表示一个加载元素
- `turn` 用来实现水波纹

2、样式设置

```
.wave {
        width: 100px;
        height: 100px;
        border: 5px solid#000;
        border-radius: 50 % ;
        position: absolute;
        top: 100px;
        left: 100px;
        overflow: hidden;
        z-index: 1;
        background: rgba(0, 0, 0, .6);
}

.turn {
        width: 200% ;
```

```
        height: 200% ;
        position: absolute;
        background: rgba(200, 100, 30, 1);
        border-radius: 40% ;
        top: 100% ;
        left: -50% ;
        animation: turn 5s linear infinite;
}
```

在上面的代码中，`wave` 实现外层的圆形；我们给 `turn` 添加了 `border-radius: 40%` ，这是实现水波纹的关键。

如果你把 `turn` 单独拿出来：

水波纹动画实质就是不断旋转 `turn`：

```
@keyframes turn {
        from {
                transform: rotate(0deg);
        }
        to {
                transform: rotate(360deg)
        }
}
```

3、脚本设置

经过上面的布局和样式，你会看到一个水波纹，但是并没有加载，所以我们需要使用js来控制：

```
var topNum = 100;
var timer = setInterval(function() {
        document.querySelector('.turn').style.top = topNum + '%';
        topNum -= 0.05;
        document.querySelector('.tips').textContent = text;
        if (topNum <= 0) {
                document.querySelector('.turn').style.top = '0%';
                clearInterval(timer);
        }
},
```

```
  1);
```

你也可以调整上面的`0.05`，值越大，加载的越快。

源码路径：`WebDemo/JavaScript/rippleLoading`

检测是否移动端

检测是否移动端

利用`navigator.userAgent`，我们就可以检测是移动端还是PC端。

```
window.addEventListener("DOMContentLoaded",
function() {
        if (/AppleWebKit.*Mobile/i.test(navigator.userAgent) || (/MIDP|Sy
mbianOS|NOKIA|SAMSUNG|LG|NEC|TCL|Alcatel|BIRD|DBTEL|Dopod|PHILIPS|HAIER|L
ENOVO|MOT-|Nokia|SonyEricsson|SIE-|Amoi|ZTE/.test(navigator.userAgent)))
{
                try {
                        if (/Android|Windows Phone|webOS|iPhone|iPod|Blac
kBerry/i.test(navigator.userAgent)) {
                            // 移动端
                                userAngent = "mobile";
                        } else if (/iPad/i.test(navigator.userAgent)) {
                            // ipad
                                userAngent = "ipad"
                        }
                        isMobile = true;
                } catch(e) {}
        } else {
                // PC端
                userAngent = "window";
        };
});
```

当然，我们还可以判断出是哪种浏览器：

```
var browserName = navigator.userAgent.toLowerCase();
if (/msie/i.test(browserName) && !/opera/.test(browserName)) {
        alert("IE");
        return;
} else if (/firefox/i.test(browserName)) {
        alert("Firefox");
        return;
} else if (/chrome/i.test(browserName) && /webkit/i.test(browserName) &&
/mozilla/i.test(browserName)) {
        alert("Chrome");
        return;
} else if (/opera/i.test(browserName)) {
        alert("Opera");
        return;
} else if (/webkit/i.test(browserName) && !(/chrome/i.test(browserName) &
& /webkit/i.test(browserName) && /mozilla/i.test(browserName))) {
        alert("Safari");
        return;
```

```
} else {
        alert("unKnow");
}
```

我们还可以判断IE的版本：

```
var browser = navigator.appName
var b_version = navigator.appVersion
var version = b_version.split(";");
var trim_Version = version[1].replace(/[ ]/g, "");
if (browser == "Microsoft Internet Explorer" && trim_Version == "MSIE6.0"
) {
        alert("IE 6.0");
} else if (browser == "Microsoft Internet Explorer" && trim_Version == "M
SIE7.0") {
        alert("IE 7.0");
} else if (browser == "Microsoft Internet Explorer" && trim_Version == "M
SIE8.0") {
        alert("IE 8.0");
} else if (browser == "Microsoft Internet Explorer" && trim_Version == "M
SIE9.0") {
        alert("IE 9.0");
}
```

搜索过滤

搜索过滤

搜索过滤就是当在输入框中输入关键词的时候,js会自动根据输入的关键词筛选底部列表,含有关键词的会保留,没有的则会被剔除。

1、搜索布局

```
<div class="filter filter-open">
        <input id="last_name" class="filter-input" type="text" placeholde
r="账号"
        autocomplete="off">

        <div class="filter-panel">
                <ul>
                        <li>
                                abc
                        </li>
                        <li>
                                dfa
                        </li>
                </ul>
        </div>
</div>
```

在上面的代码：

- `filter` 表示搜索列表
- `input` 是输入框
- `filter-panel` 表示输入时弹出的内容框
- `filter-open` 表示当前是否处于搜索状态

对于这里的例子来说，`filter-panel`是根据要搜索的内容列表来绘制的，而不是手动加入的。

2、样式设置

```
.filter {
        position: relative;
        display: inline - block;
}

.filter-panel {
        display: none;
        position: absolute;
        top: 99% ;
        left: 0;
        width: 100% ;
        z-index: 99;
        background: #fff;
        border: 1px solid#d9d9d9;
        -webkit-transition: all.3s;
        transition: all.3s;
        opacity: 0;
        -webkit-transform: translate(0, 20px);
        transform: translate(0, 20px);
        display: none;
}

.filter-open.filter-panel {
        opacity: 1;
        -webkit-transform: translate(0, 0);
        transform: translate(0, 0);
}
```

在上面的CSS样式中，`filter-panel`是用绝对定位的，将`top`设为99%，置于input非下方。

这里加`transform`是为了动态，你也可以直接的`display: block`或`display: none`。

3、过滤

对于如何过滤数据，我这里使用`indeOf()`方法。

`indexOf()`方法可返回某个指定的字符串值在字符串中首次出现的位置。
如果要检索的字符串值没有出现，则该方法返回 -1。

注意：`indexOf()`方法对大小写敏感！如果你要忽略大小写，你可以使用

`toLowerCase()` 将字符串转为小写。

当然，如果你要过滤更详细的数据，那你就要使用正则来验证了。

源码路径：`WebDemo/JavaScript/search`

弹幕

弹幕

这一节来讲解如何在网页上实现弹幕功能，类似于直播弹幕网站上的弹幕功能，可自定义背景色、字体颜色、弹幕出现时间、弹幕链接以及是否显示头像。

1、弹幕

```
<div class="sg-barrage" >
    <a>
        <img src="../images/f.jpg">
    </a>
    23
</div>
```

在上面的代码：

- `sg-barrage` 表示一条弹幕
- `<a>` 是用来添加链接
- `<img>` 是头像

2、样式设置

```
.sg-barrage{
```

```
        position:fixed;
        bottom:80px;
}
```

这里使用 `fixed` 定位。

3、弹幕动画

弹幕的动画其实很简单，只是改变每一个弹幕的 `right` 值。

我们使用一个数组来保存当前页面中存在的弹幕：

```
var barrages = [];
```

初始化弹幕位置：

```
n.barrage.right = -(n.barrage.offsetWidth + 20);
```

在上面的代码中，`n.barrage` 表示一个弹幕，`n.barrage.right` 表示弹幕当前的位置（也就是right的值）。

当一条弹幕生成时，它的 `right` 值需要位于屏幕外，也就是右侧看不到的地方。

同时，我们使用计时器来实现动画：

```
n.barrage.timeoutId = setInterval(function() {
        if (n.barrage.stoped) return;
        if (n.barrage.right <= (n.barrage.offsetWidth + width)) {
                n.barrage.right += 1;
                n.css(n.barrage, {
                        right: n.barrage.right + 'px'
                });
        } else {
                clearInterval(n.barrage.timeoutId);
                barrages.splice(n.barrage.index, 1);
                document.body.removeChild(n.barrage);
        }
},
n.params.speed);
```

`n.barrage.right <= (n.barrage.offsetWidth + width)` 用来判断弹幕是否在离开了屏幕，如果没有离开，继续运行；如果离开了，使用 `clearInterval()` 删除弹幕的计时器，同时使用 `splice()` 删除 `barrages` 数组中对应的弹幕。

在我给出的例子中，要发射一条弹幕，只需这样：

```
document.getElementById("send").addEventListener('click',
function() {
        var b = Barrage({
                text: '弹幕'
        });
});
```

当然，你还可以设置更多参数：

```
Barrage({
        text: '文本',
        color: '字体颜色',
        backgroundColor: '背景色',
        speed: 6,  // 速度
        url: '链接地址',
        img: '头像'
})
```

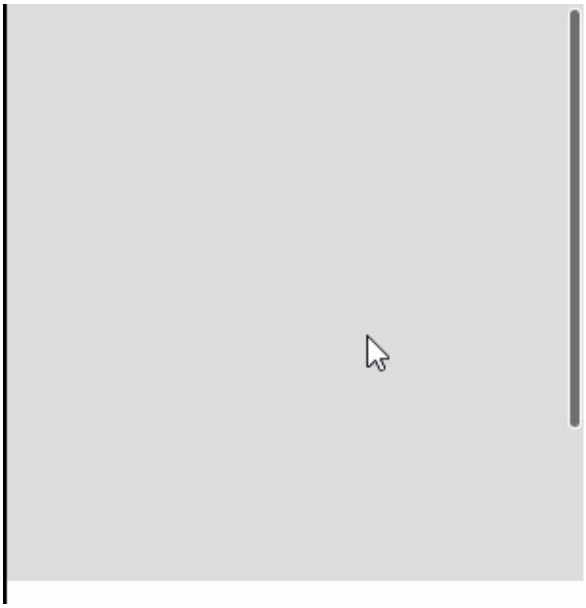
源码路径：`WebDemo/JavaScript/barrage`

自定义滚动条

自定义滚动条

在《CSS实战篇》中，我们了解了如何使用CSS美化滚动条，不过那个只支持webkit内核的浏览器。

在这一节，我们将了解如何用JS来实现自定义滚动条！



1、创建模板

```
<div class="lazy-iscroll">
        <div class="lazy-iscroll-wrapper">
        </div>
</div>
```

在上面的代码：

- `lazy-iscroll` 表示滚动框
- `lazy-iscroll-wrapper` 里面放置内容

2、设置CSS样式

```
.lazy - iscroll {
        position: relative;
```

```
        width: 300px;
        height: 300px;
}.lazy - iscroll - wrapper {
        background: #ddd;
        height: 400px;
}
```

3、JavaScript控制

首先，我们肯定要获取滚动的高度：

```
n.scrollHeight = n.wrapper.offsetHeight;
n.offsetHeight = n.container.offsetHeight;
n.scrollTop = n.scrollHeight - n.offsetHeight;
```

offsetHeight 是网页内容实际高度，也就是用 `div.lazy-iscroll-wrapper` 的实际高度减去 `div.lazy-iscroll` 的实际高度，就得到需要滚动的高度，也就是前者被后者截取的高度。

绑定滚轮事件

```
WHEEL_EV = isMoz ? 'DOMMouseScroll' : 'mousewheel';
function onWheel(e) {
        var that = this,
        wheelDeltaX, wheelDeltaY, deltaX, deltaY, deltaScale;
        if ('wheelDeltaX' in e) { // 向下滚动是负数-120, 向上滚动是正数120
                wheelDeltaX = e.wheelDeltaX / 12;
                wheelDeltaY = e.wheelDeltaY / 12;
        } else if ('wheelDelta' in e) {
                wheelDeltaX = wheelDeltaY = e.wheelDelta / 12;
        } else if ('detail' in e) { //向下滚动是正数3, 向上滚动是负数-3
                wheelDeltaX = wheelDeltaY = -e.detail * 3;
        } else {
                return;
        };
        if (!n.isLoading) {
                n.isRunning = true;
                move(wheelDeltaY);
        }
}
if (!isTouch) {
        addEvent(n.container, WHEEL_EV, onWheel, false);
};
```

在《JavaScript实战篇》的全屏滚动一节中，已经详细地介绍过滚轮事件，如果不明白，可查看那一节。

在上面的代码中，主要通过`move()`方法来滚动。

```
function move(y) {
        n.moveY += y;
        if (!n.isPressed) {
                if (Math.abs(n.moveY) > n.scrollTop) {
                        n.moveY = -n.scrollTop;
                        n.isLoading = true;
                        setTimeout(function() {
                                n.isLoading = false;
                        },
                        1000);
                        n.isRunning = false;
                } else if (y > 0 && n.moveY > 0) {
                        n.moveY = 0;
                        n.isRunning = false;
                };
                setTransitionDuration(n.wrapper, 250);
        };
        //计算滚动条滚动的高度
        var mY = 0;
        if (n.moveY >= 0) {
                mY = 0;
        } else if (Math.abs(n.moveY) >= n.scrollTop) {
                mY = -n.scrollTop;
        } else {
                mY = n.moveY;
        };
        var sv = mY / n.scrollTop * (n.iscrollbar.offsetHeight - n.scroll
BarHeight);
        sv = -Math.floor(sv);
        n.iscrollbar.style.opacity = 1;
        setTransform(n.thumb, 'translate3d(0, ' + sv + 'px, 0');
        setTransform(n.wrapper, 'translate3d(0, ' + n.moveY + 'px, 0');
}
```

其实，每次滚动，就是改变`div.lazy-iscroll-wrapper`的`transform: translate3D()`的Y轴的值。

源码路径：`WebDemo/JavaScript/scroll`

城市联动选择器

城市联动选择器

在这一节中，我将介绍带select下拉选择框美化的jQuery省市区三级联动插件，可使用默认select组件，也可以设置自定义的组件。

先来看看这个组件的参数说明：

```
var nselector2 = new NSelector('.nselect', {
        data: city,  // 城市数组
        custom: true,  // 是否采取自定义的组件，省略为默认select
        onChange: function(data) {  // 选择后的事件
                console.log(data);
        }
});
```

其实城市联动选择器的关键就是将省市区三个有一个比较好的绑定关联：

```
var city = [{
        "name": "安徽",
        "child": [{
                "name": "合肥",
                "child": [{
                        "name": "长丰"
                }, {
                        "name": "肥东"
```

```
                }, {
                        "name": "肥西"
                }, {
                        "name": "合肥市"
                }]
        }
}];
```

在上面的代码中，第一层存储的是省，省里面有一个child的属性，存储的是其对应的市，市下面有一个child属性，存储对应的区。

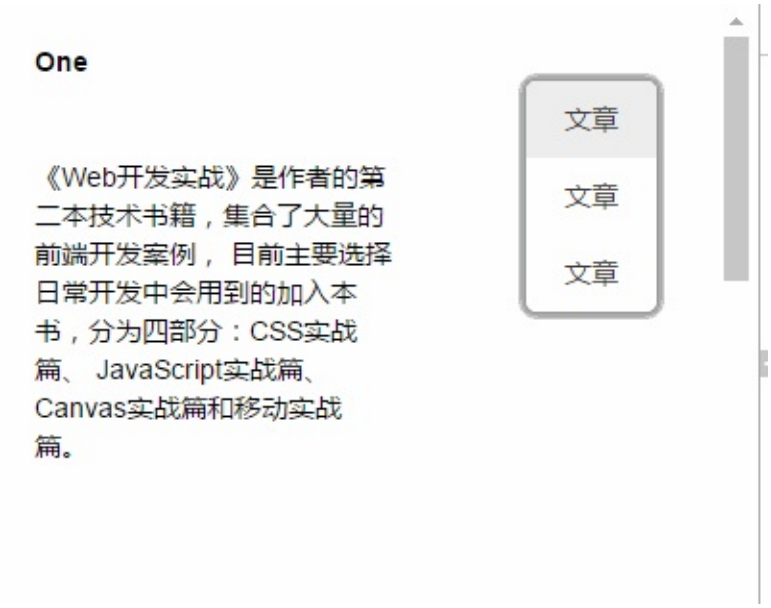
每次选择变化后，就相对应的去遍历数组的内容，重新改变select的子项数据即可。

源码路径：`WebDemo/JavaScript/citys`

滚动监听

滚动监听

滚动监听插件是用来根据滚动条所处的位置来自动更新导航项的。如下所示，点击tab或滚动下面的区域。tab菜单中的条目也会自动高亮显示。



1、创建模板

```
<div class="scrollSpy">
        <div class="scrollSpy-tabs">
                <ul>
                        <li>
                                <a href="#one" class="active">..</a>
                        </li>
                        <li>
                                <a href="#two">..</a>
                        </li>
                        <li>
                                <a href="#three">..</a>
                        </li>
                </ul>
        </div>
        <div class="scrollSpy-content">
                <h4 id="one">
                        One
                </h4>
                <p style="padding-bottom:800px">..</p>
                <h4 id="two">
                        Two
                </h4>
```

```
                <p style="padding-bottom:800px">..</p>
                <h4 id="three">
                        Three
                </h4>
                <p style="padding-bottom:800px">..</p>
        </div>
</div>
```

在上面的代码中：

- `scrollSpy` 表示滚动监听容器
- `scrollSpy-tabs` 表示滚动tab列表
  - `a` 表示滚动tab，`active` 表示当前项
- `scrollSpy-content` 表示滚动区域
  - `h4` 表示滚动目标（当然也可以用其他元素，不过一定要设置对应于tab的id ）

2、设置CSS样式

```
.scrollSpy{
        position:relative;
        overflow:hidden;
        overflow-y: auto;
        height:100%;
}
```

只有上面的样式是必须的，让滚动容器垂直方向滚动，其他美化样式可查看文件。

3、JavaScript

点击滚动到目标区域很简单，只需获取到目标的 `id` ，然后获取相对于父元素（`scrollSpy` ）的顶部距离即可：

```
var id = this.href.split('#')[1];
var con = document.getElementById(id);
var top = con.offsetTop;

el.scrollTop = top;
```

在上面的代码中，el是父元素（scrollSpy），这样就可以直接跳转到目标位置。

注意：给scrollTop赋值时不用加单位。

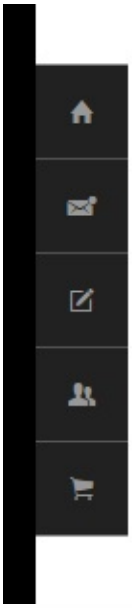
当然，你也可以像我的代码中加入缓动 `scrollSpyTo()` 方法。

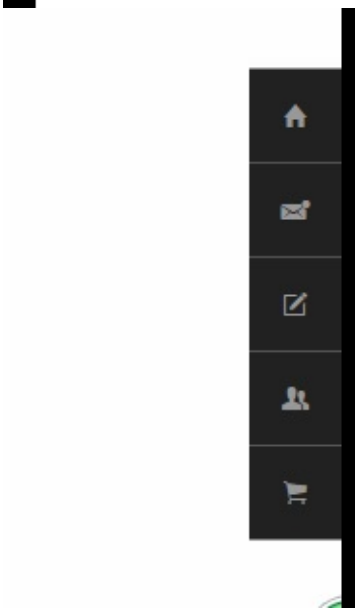
源码路径：`20170203/scrollSpy`

边栏悬浮菜单

边栏悬浮菜单

边栏悬浮菜单其实就是隐藏菜单的一部分，当鼠标移动上去时，显示隐藏的部分。如下所示：





1、创建模板

```
<div class="scroll-menu scroll-menu-left">
        <ul class="scroll-menu-tabs">
```

```
                <li>
                        <a>
                                <i class="ion-home"></i>
                                <span>首页</span>
                        </a>
                </li>
                <li>
                        <a>
                                <i class="ion-email-unread"></i>
                                <span>邮件</span>
                        </a>
                </li>
                <li>
                        <a>
                                <i class="ion-compose"></i>
                                <span>写文章</span>
                        </a>
                </li>
                <li>
                        <a>
                                <i class="ion-person-stalker"></i>
                                <span>A联系人</span>
                        </a>
                </li>
                <li>
                        <a>
                                <i class="ion-ios-cart"></i>
                                <span>购物车</span>
                        </a>
                </li>
        </ul>
</div>
```

在上面的代码：

- `scroll-menu` 表示一个悬浮菜单列表
  - `li` 表示一个菜单子项
- `scroll-menu-left` 或 `scroll-menu-right` 表示悬浮菜单位置（左或右）

2、设置CSS样式

```
.scroll-menu {
        position: fixed;
        top: 50%;
        -webkit-transform: translate(0, -50%);
        -moz-transform: translate(0, -50%);
        transform: translate(0, -50%);
}
```

在上面的代码中，使用position:fixed让菜单悬浮。

```
.scroll-menu li a,
.scroll-menu li span{
        background: #222222;
        color: #999;
    border-bottom: 1px solid rgba(0,0,0,0.1);
}

.scroll-menu ul {
        list-style:none;
}
.scroll-menu-tabs li{
        position: relative;
    -moz-transition: all 0.5s;
    -webkit-transition: all 0.5s;
    -o-transition: all 0.5s;
    transition: all 0.5s;
}
.scroll-menu li a ,
.scroll-menu li span {
    display: block;
    min-height: 3em;
    min-width: 3em;
    line-height: 3em;
    text-align: center;
    position: relative;
    cursor: pointer;
}
.scroll-menu li span{
        position:absolute;
        top:0;
         white-space: nowrap;
}

.scroll-menu.scroll-menu-left{
        left: 0;
}
.scroll-menu.scroll-menu-left li span{
        right: 3em;
}
.scroll-menu.scroll-menu-left .scroll-menu-tabs li{
        left:0;
}

.scroll-menu.scroll-menu-left .scroll-menu-tabs li span{
        padding-left: 1em;
}
```

在这里，我们让 `span` 绝对定位（ `position:absolute` ），也就是隐藏在左侧，当鼠标移动上去时，设置其父元素 `li` 的 `left` 值为 `span` 的宽度。

3、JavaScript

添加鼠标事件：

```
addEvent(tab, 'mouseover',
function() {
  //  获取span的宽度
        var w = tab.querySelector('span').offsetWidth;
    // 判断是在左侧还是右侧，设置li的left或right值。
        if (hasClass(menu, 'scroll-menu-left')) {
                this.style.left = w + 'px';
        } else {
                this.style.right = w + 'px';
        }
        addClass(this, 'active');
});
addEvent(tab, 'mouseout',
function() {
        if (hasClass(menu, 'scroll-menu-left')) {
                this.style.left = '0px';
        } else {
                this.style.right = '0px';
        }
        removeClass(this, 'active');
});
```

只是简单的改变li的定位属性left或right即可。

源码路径：`20170203/scroll_menu`

canvas实战篇

Canvas实战篇

功能目录：

- 粒子动画
- 刮刮卡
- 截图下载
- 图片放大器
- 手势密码
- 雪花纷飞
- 粒子聚合

雪花纷飞

雪花纷飞

这一节我们讲解使用HTML5 canvas制作的效果非常炫酷的3D雪花飘落特效。

1、创建模板

一个简单的div：

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

当然，你也可以直接放置 `canvas` ，我这里是通过动态绘制的，背景图也是动态绘制的。

2、JavaScript

既然是雪花纷飞，当然先来定义雪花：

```
function Item(x, y, r) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.vx = 0;
        this.vy = 5;
        this.deg = 0;
}
```

在上面的代码中，我们创建一个Item对象，用来保存每一片雪花的数据。

然后用一个数组来保存当前所有雪花：

```
var items = [];
```

雪花纷飞的难点在于如何仿雪花，只需这样：

```
var rg = ctx.createRadialGradient(body.x, body.y, Math.floor(body.r / 4),
 body.x, body.y, body.r);
rg.addColorStop(0, "rgba(255,255,255,1)");
rg.addColorStop(1, "rgba(255,255,255,0.1)");
ctx.fillStyle = rg;

ctx.beginPath();
ctx.arc(body.x, body.y, body.r, 0, 2 * Math.PI, true);
ctx.fill();
```

利用径向渐变功能来实现雪花。

让雪花动起来:

```
items[i].x += items[i].vx;
items[i].y += items[i].vy;
```

让其x和y与速度相加既可以。

最后，我们使用`requestAnimationFrame()`方法来实现循环动画（你也可以使用计时器）。

这里还需注意一点的是，为了提高效率，我们需要将每一次离开屏幕的雪花移除（也就是从items数组中移除）。

```
var i = items.length;
while (i--) {
        //判断是否离开画面
        if (x1 > width || x1 < 0 || y1 > height) {
                items.splice(i, 1);
        } else {
                items[i].x += items[i].vx;
                items[i].y += items[i].vy;
                drawItem(items[i]);
        };
};
```

这样，我们就实现了雪花纷飞的效果。

源码路径：`WebDemo/Canvas/snow`

粒子动画

粒子动画

自从canvas横空出世，各种酷炫的动画就层出不穷，而粒子动画更是流行，而且出现了各种各样的版本。其实，看似复杂的粒子动画，其他并不难实现，当然，关键是你耐心。

首先，我们需要在页面放置一个canvas：

```
<canvas id="particles"></canvas>
```

在这里，你可以直接给其设置高宽。

我将粒子动画封装成了一个类，可以这样调用：

```
var particle = new Particle('#particles', {
        effect: 'line',  // zoom | line effect表示鼠标移动上去的效果
        point: 300  // 粒子数量
        mouseDis: 200 // 连线距离
});
```

接下来，我们一步一步的分析如何实现粒子动画。

首先，我们需要获取canvas元素：

```
canvas = document.getElementById('particles');
```

创建context对象：

```
ctx = p.canvas.getContext('2d');
```

粒子动画，当然需要很多粒子，所以我定义了方法 `point()` ，专门来构建粒子：

```
//点
p.point = function() {
        this.color = new p.color();
        this.x = Math.random() * p.innerWidth;
        this.y = Math.random() * p.innerHeight;
```

```
        this.vx = p.random(10, -10) / 40;
        this.vy = p.random(10, -10) / 40;
        this.r = p.random(3, 1);
        this.scale = 1;
};
```

在这个方法中，粒子的位置（x和y）、半径（r）、运动速度（vx和vy）都是随机的，颜色（color）也是随机的：

```
//粒子颜色
p.color = function() {
        function random() {
                return Math.round(Math.random() * 255);
        };
        this.r = random();
        this.g = random();
        this.b = random();
        this.a = random(1, 0.8);
        this.rgba = 'rgba(' + this.r + ',' + this.g + ',' + this.b + ','
+ this.a + ')';
        return this;
};
```

我们还需要一个数据结构去保存页面中的粒子，这里使用一个数组：

```
p.points = [];
```

容器有了，粒子也有了，接下来就是绘制了：

```
p.initDrawPoint = function() {
        for (var i = 0; i < p.params.point; i++) {
                var point = new p.point();
                p.points.push(point);
                p.ctx.beginPath();
                p.ctx.fillStyle = point.color.rgba;
                p.ctx.arc(point.x, point.y, point.r * point.scale, 0, Mat
h.PI * 2, true);
                p.ctx.fill();
        };
        p.ctx.closePath();
};
```

p.params.point是页面粒子的数量，默认是300，你也可以更改，利用arc()方法来绘制粒子，别忘了将粒子添加进容器（p.points)。

这个时候，页面已经有粒子存在了，接着我们要让它们动起来！动起来！

我使用的是`requestAnimationFrame`方法（requestAnimationFrame 是专门为实现高性能的帧动画而设计的一个API），当然，你也可以使用setInterval或setTimeout计时器来实现动画。

```
//无限循环动画
p.animation = function() {
    //  别忘了清空canvas再绘制新画面
        p.ctx.clearRect(0, 0, p.innerWidth, p.innerHeight);
        p.drawBackground();
        for (var i = 0; i < p.params.point; i++) {
                var point = p.points[i];
                // 这里是监听粒子是否到达canvas的边界，如果到达，将速度取反
                if (point.x < 0 || point.x > p.innerWidth) {
                        point.vx = -point.vx;
                };
                if (point.y < 0 || point.y > p.innerHeight) {
                        point.vy = -point.vy;
                };
                p.ctx.beginPath();
                p.ctx.fillStyle = point.color.rgba;
                point.x += point.vx;
                point.y += point.vy;
                p.ctx.arc(point.x, point.y, point.r * point.scale, 0, Mat
h.PI * 2, true);
                p.ctx.fill();
        };
        if (p.params.effect == 'zoom') {
                p.connect();
        } else if (p.params.effect == 'line') {
                p.lineto();
        };

        // 这里是动画循环，有点像递归
        requestAnimationFrame(p.animation);
};
```

到这里，最基本的粒子动画已经有了，是不是很简单呢！

我们还可以添加一些额外的效果，比如：鼠标交互动画。

注意：由于我们无法获取到canvas上面绘制的任何图形或线等，我们只能将事件添加到canvas上。

p.lineto()方法就是鼠标移动上去的连线效果。

```
p.lineto = function() {
```

```
        function isInView(point) {
                return Math.abs(point.x - p.pageXY.pageX) < p.params.mous
eDis && Math.abs(point.y - p.pageXY.pageY) < p.params.mouseDis;
        };
        (function line() {
            //  这里使用的是线性渐变
                function lineColor(p1, p2) {
                        var linear = p.ctx.createLinearGradient(p1.x, p1.
y, p2.x, p2.y);
                        linear.addColorStop(0, p1.color.rgba);
                        linear.addColorStop(1, p2.color.rgba);
                        return linear;
                };
                for (var i = 0; i < p.params.point; i++) {
                        for (var j = 0; j < p.params.point; j++) {
                                if (i != j) {
                                        var p1 = p.points[i];
                                        var p2 = p.points[j];
                                        if (isInView(p1) && isInView(p2))
 {
                                                if (Math.abs(p2.x - p1.x)
 < p.params.minDis && Math.abs(p2.y - p1.y) < p.params.minDis) {
                                                        p.ctx.beginPath()
;
                                                        p.ctx.lineWidth =
 0.2;
                                                        p.ctx.strokeStyle
 = lineColor(p1, p2);
                                                        p.ctx.moveTo(p1.x
, p1.y);
                                                        p.ctx.lineTo(p2.x
, p2.y);
                                                        p.ctx.stroke();
                                                        p.ctx.closePath()
;
                                                };
                                        };
                                };
                        };
                };
        })(); // 立即执行函数
};
```

isInView()方法是判断粒子与鼠标的距离，默认值是200，只有粒子与鼠标的距离小于200的粒子，才会连线。

总之，只要你敢想，你就能实现很酷炫的粒子动画。

源码路径：`WebDemo/Canvas/particle`

刮刮卡

刮刮卡

类似淘宝的刮刮卡功能是很容易实现的，这一节我们就来了解如何实现?

1、布局

只需一个最简单的div，canvas是动态画出来的。

```
<div id="box"></div>
```

2、CSS样式

一般情况下，刮刮卡的刮开的后面都是用图片：

```
position: relative;
width: 300px;
height: 200px;
background - image: url(a.jpg);
background - repeat: no - repeat;
background - size: cover;
```

当然，在这里也是用脚本去控制的，而非固定的。

3、JavaScript

初始化时，我们会看到纯灰色卡片：

```
cxt.fillStyle = r.params.foreBackgroundColor;
cxt.fillRect(0, 0, r.params.width, r.params.height);
```

颜色你可以更改。

当然，少不了鼠标或触摸事件，我们将事件绑定到canvas上。

```
overlay.addEventListener(downEvent, down);
```

将事件绑定后，我们首先要获取的是鼠标相对于canvas的坐标。

```
var move = function(e) {
        var e = e || window.event;
        var touches = r.isSupportTouch ? e.touches[0] : e;
        //获取鼠标相对于canvas的坐标
        var x = (touches.clientX + document.body.scrollLeft || touches.pa
geX) - overlay.parentNode.offsetLeft;
        var y = (touches.clientY + document.body.scrollLeft || touches.pa
geY) - overlay.parentNode.offsetTop;
        clearArc(x, y);
};
```

在上面的代码中，`clearArc()`方法是用来绘制已经刮过的部分。

```
var clearArc = function(x, y) {
        cxt.globalCompositeOperation = 'destination-out';
        cxt.beginPath();
        cxt.arc(x, y, r.params.fontSize, 0, Math.PI * 2, true);
        cxt.fill();
        checkHalf();
};
```

注意`globalCompositeOperation`属性用来设置或返回如何将一个源（新的）图像绘制到目标（已有）的图像上。

当设置为`destination-out`，表示原有内容中与新图形不重叠的部分会被保留，也就是说新绘制的会覆盖掉原先的内容。

而`checkHalf()`方法是用来判断已经刮掉的部分：

```
var checkHalf = function() {
        var cd = cxt.getImageData(0, 0, r.params.width, r.params.height);
        var j = 0;
        //每个像素点有四个值，这里判断每个像素点的一个值为0就行，也就是透明
        for (var i = 0; i < cd.data.length; i += 4) {
                if (cd.data[i] == 0) {
                        j++;
                }
        }
        if (j >= cd.data.length / 8) {
                cxt.fillRect(0, 0, r.params.width, r.params.height);
                if (r.params.onRubEnd !== 'undefined' && (typeof r.params
.onRubEnd === 'function' && !isEnd)) {
                        r.params.onRubEnd();
                }
                isEnd = true;
```

```
        }
}
```

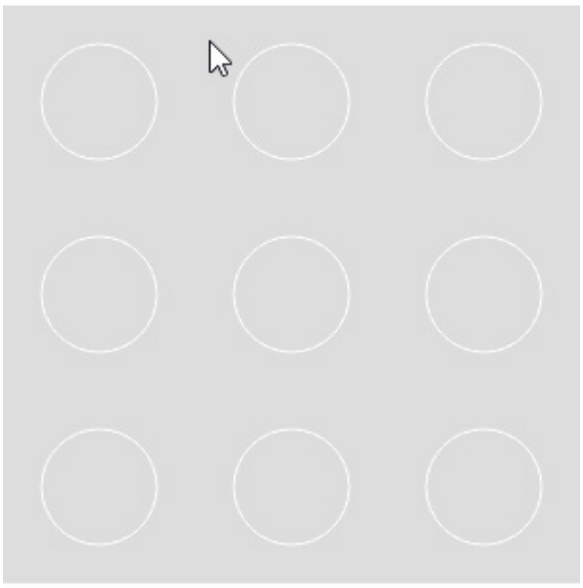
在上面的代码中，我们通过`getImageData()`方法来获取canvas上的像素值，然后判断每个像素点是否透明，如果透明点（也就是刮掉的部分）大于八分之一时，就会显示完整的背景图。

源码路径：`WebDemo/Canvas/card`

手势密码

手势密码

手势密码是仿手机九宫格密码解锁效果。



在这里，我们使用`canvas`来使用。

1、布局

依旧是一个简单的div：

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

2、JavaScript

既然是九宫格，首先就要给canvas分格：

```
var c = n.params.grid.cols;
var r = n.params.grid.rows;
var w = Math.floor(width / c);
var h = Math.floor(height / r);
n.drawBg();
for (var i = 0; i < r * c; i++) {
        var x = Math.floor(w / 2) + Math.floor(i % c) * w; // i % c表示在
第几列
        var y = Math.floor(h / 2) + Math.floor(i / c) * h; // i / c表示在
```

```
第几行
        var r = Math.floor(w / 2 * 0.6);
        var item = new Item(x, y, r);
        item.i = i;
        items.push(item);
};
```

在上面的代码中，`n.params.grid.cols`表示列数，`n.params.grid.rows`表示行数，然后计算出每一格的宽高（w和h）。

`Item`类用来保存每一格的数据：

```
function Item(x, y, r) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.theme = 0;
}
```

当然，依旧少不了获取鼠标相对canvas的位置：

```
function getPoint(event) {
        /*将当前的触摸点坐标值减去元素的偏移位置，返回触摸点相对于element的坐标值*/
        event = event || window.event; //e.originalEvent.targetTouches[0]
        var touchEvent = isTouch ? event.changedTouches[0] : event;
        var x = (touchEvent.pageX || touchEvent.clientX + document.body.s
crollLeft + document.documentElement.scrollLeft);
        x -= element.offsetLeft;
        var y = (touchEvent.pageY || touchEvent.clientY + document.body.s
crollTop + document.documentElement.scrollTop);
        y -= element.offsetTop;
        return {
                x: x,
                y: y
        };
};
```

九宫格的样式，是通过`arc()`来绘制三个同心圆，而直接的连接则是通过`moveTo()`和`lineTo()`方法。

当拖动鼠标时，我们如何判断是在哪一格中呢？

```
n.isInCircle = function(b, x, y) {
        var mx = x - b.x;
        var my = y - b.y;
        var mr = Math.sqrt(mx * mx + my * my);
```

```
        if (mr < b.r) {
                return true;
        };
        return false;
};
```

在上面的代码中，参数b表示每一格的位置（也就是每一格Item实例），然后通过判断鼠标与其的之间的距离是否大于其半径格内，否则在外面。

最后，我们需要在鼠标离开屏幕时检测密码的正确性。

我们只需在绘制连线时将选择的格子保存下来，然后对比索引即可：

```
n.check = function() {
        var isTrue = true;
        for (var i = 0; i < lines.length; i++) {
                if (lines[i].i != n.params.correct[i]) {
                        isTrue = false;
                        break;
                }
        };
        return isTrue;
};

if (n.check()) {
        n.params.onSuccess && n.params.onSuccess(n);
}
```

当然，不正确的时候，只需检测到一个不正确就行了。

源码路径：`WebDemo/Canvas/gestureLock`

截图下载

截图下载

这一节我们来讲解如何实现canvas的截图下载！

1、下载链接

我们使用 `<a>` 标签来下载：

```
<a download="截图.png" id="download">截图下载</a>
```

`download` 属性规定被下载的超链接目标。

2、JavaScript

```
url = canvas.toDataURL('image/png');
document.getElementById('download').href = url;
```

HTMLCanvasElement.toDataURL() 方法返回一个包含图片展示的 data URI 。可以使用 type 参数其类型,默认为 PNG 格式。

这样就实现了canvas元素图片的下载。

当然，这里还要附加一个有趣的功能，那就是我们可以将 `video` 元素绘制到 `canvas` 上，然后用计时器 `setInterval` 绘制，就可以实现视频播放的功能,当然，也就是说我们可以截图每一帧的图片。

源码路径：`WebDemo/Canvas/printscreen`

图片放大器

图片放大镜

图片放大镜是指当鼠标移动到图片上时，图片局部放大功能。

类似淘宝、天猫都有这种图片放大镜的功能。

这一节，我们就是来讲解放大镜的实现，这里，只会讲解两种方式。

1、canvas方式

先来了解一下 `drawImage()` 方法，它最多可传入9个参数值：

```
drawImage(img,sx,sy,swidth,sheight,x,y,width,height);
```

- img：规定要使用的图像、画布或视频。
- sx：可选。开始剪切的 x 坐标位置。
- sy：可选。开始剪切的 y 坐标位置。
- swidth：可选。被剪切图像的宽度。
- sheight：可选。被剪切图像的高度。
- x：在画布上放置图像的 x 坐标位置。
- y：在画布上放置图像的 y 坐标位置。
- width：可选。要使用的图像的宽度（伸展或缩小图像）。
- height：可选。要使用的图像的高度（伸展或缩小图像）。

现在来讲解一下原理：

首先，我们还是要获取鼠标相对于canvas的位置：

```
function getPoint(event) {
        /*将当前的触摸点坐标值减去元素的偏移位置，返回触摸点相对于element的坐标值*/
        event = event || window.event; //e.originalEvent.targetTouches[0]
        var touchEvent = isTouch ? event.changedTouches[0] : event;
        var x = (touchEvent.pageX || touchEvent.clientX + document.body.s
crollLeft + document.documentElement.scrollLeft);
        x -= element.offsetLeft;
        var y = (touchEvent.pageY || touchEvent.clientY + document.body.s
crollTop + document.documentElement.scrollTop);
```

```
        y -= element.offsetTop;
        return {
                x: x,
                y: y
        };
};
```

在上面的代码中，通过`getPoint()`方法我们能获取到鼠标相对`canvas`的坐标，那接下来就很简单了，将获取到坐标和放大的倍数传入`drawImage()`方法即可。

2、background方式

除了用`canvas`方式外，我们使用`background`方式也可以实现图片放大镜。

使用到的属性：
`background-position`：设置背景图像的起始位置

我们只需将获取到的鼠标相对源图片的位置取反（也就是负值）传给`background-position`即可。

源码路径：`WebDemo/Canvas/glass`

粒子聚合

粒子聚合

先来看看效果：

注意：如果打开后，图片没有效果，请留意图片跨域问题，请用HBuilder之类的编辑器打开即可。

1、创建模板

只需简单的canvas：

```
<canvas id="canvas"></canvas>
```

参数说明：

```
var explode = new Explode('#canvas', {
        type: 1,  // 1为文字，2为图片
        width: 500,  // canvas的宽
        height: 300,  // canvas的高
        text: '聚合文字',  // 文字
    img: 'mm.jpg'  // 图片路径
});
```

2、JavaScript

首先来认识几个方法：

`getImageData()` 方法返回 ImageData 对象，该对象拷贝了画布指定矩形的像素数据。

```
getImageData(x,y,width,height);
```

x 开始复制的左上角位置的 x 坐标（以像素计）。
y 开始复制的左上角位置的 y 坐标（以像素计）。
width 要复制的矩形区域的宽度。
height 要复制的矩形区域的高度。

putImageData() 方法将图像数据（从指定的 ImageData 对象）放回画布上。

```
putImageData(imgData,x,y,dirtyX,dirtyY,dirtyWidth,dirtyHeight);
```

imgData 规定要放回画布的 ImageData 对象。
x ImageData 对象左上角的 x 坐标，以像素计。
y ImageData 对象左上角的 y 坐标，以像素计。
dirtyX 可选。水平值（x），以像素计，在画布上放置图像的位置。
dirtyY 可选。垂直值（y），以像素计，在画布上放置图像的位置。
dirtyWidth 可选。在画布上绘制图像所使用的宽度。
dirtyHeight 可选。在画布上绘制图像所使用的高度。

现在来说说具体实现。

首先，我们需要将文字或图片绘制在一个canvas上

```
var c = document.createElement('canvas');
```

文字和图片的绘制，这里就不多说了，主要使用fillText()和drawImage()两个方法。

下面着重讲解如何分解粒子。

用一个数组来放置所有粒子：

```
var items = [];
```

首先创建一个Item构造函数，用来放置每一个粒子。

```
function Item(data, targetX, targetY, currentX, currentY) {
        this.data = data;
        this.targetX = targetX;
```

```
        this.targetY = targetY;
        this.currentX = currentX;
        this.currentY = currentY;
        this.ax = .13 - Math.random() * .06;
        this.ay = .16 - Math.random() * .08;
}
```

在上面的代码中，data表示这个粒子的ImageData 对象；targetX和targetY表示聚合的最终位置；currentX和currentY表示当前位置；ax和ay分别表示运动速度。

接下来分解粒子：

```
function drawCanvas() {
        if (n.params.type == 2) {
                picture = new Image();
                picture.onload = function() {
                        // 省略了绘制图片代码
                        draw(pw, ph);
                };
                picture.src = n.params.img;
        } else {
                // 省略了绘制文字代码
                draw(w, h);
        }

        function draw(pw, ph) {
                var w = 1;
                var h = 1;
                var cols = Math.floor(pw / w);
                var rows = Math.floor(ph / h);
                for (var i = 0; i < c.width * c.height; i++) {
                        var x = Math.floor(i % cols);
                        var y = Math.floor(i / cols);
                        var data = ct.getImageData(x * w, y * h, w, h);
                        var vx = getRandom(300, -300);
                        var vy = getRandom(500, -500);
                        var item = new Item(data, x, y, x + vx, y + vy);
                        items.push(item);
                };
                total = items.length;
                cutSlice();
        }
}
```

在上面的代码中，`draw`方法用来分解粒子，先分成`cols`列和`rows`行，每一个粒子高度都为1，然后用`getImageData()`来获取ImageData对象，然后创建新的Item实例，然后添加到items数组中。

相信你也留意到了最后的cutSlice()方法，这是用来实现粒子动画的。

我们需要循环数组items中的每个粒子，让其往最终位置运动。

```
function cutSlice() {
        ctx.clearRect(0, 0, n.container.width, n.container.height);
        for (var i = 0; i < c.width * c.height; i++) {
                var item = items[i];
                var targetX = item.targetX;
                var targetY = item.targetY;
                var currentX = item.currentX;
                var currentY = item.currentY;
                var ax = false;
                var ay = false;
                if (!item.isLock) {
                        if (Math.abs(targetX - currentX) <= .5) {
                                item.currentX = targetX;
                                ax = true;
                        } else {
                                item.currentX += (targetX - currentX) * i
tem.ax;
                        };
                        if (Math.abs(targetY - currentY) <= .5) {
                                item.currentY = targetY;
                                ay = true;
                        } else {
                                item.currentY += (targetY - currentY) * i
tem.ay;
                        };
                        if (ax && ay) {
                                total--;
                                item.isLock = true;
                        }
                };
                var ix = item.currentX;
                var iy = item.currentY;
                ctx.putImageData(item.data, ix, iy);
        };
        if (total > 0) {
                requestId = requestAnimationFrame(cutSlice);
        } else {
                cancelAnimationFrame(requestId);

        };
}
```

在这里，我们使用 `requestAnimationFrame()` 方法来实现动画，不停的改变粒子的位置，然后用 `putImageData()` 来绘制到canvas上，变量total用来记录当前完成动画的粒子个数。

注意：只有ax和ay同时到达终点时，total才会加1。

源码路径： `20170203/explode/`

移动实战篇

移动实战篇

功能目录：

- 移动联动选择器
- 列表滑动删除

列表滑动删除

列表侧滑删除

相信都用过QQ，在

1、创建模板

```
<div class="slice-list slice-non">
        <ul>
                <li class="slice-item">
                        <div class="slice-tool">
                                <span>
                                        删除
                                </span>
                                <span>
                                        删除
                                </span>
                        </div>
                        <div class="slice-content">
                                列表项
                        </div>
                </li>
        </ul>
</div>
```

在上面的代码：

- `slice-list`表示列表
- `slice-non`表示第一种风格，省略则是第二种风格（示例图2）
  - `slice-item`表示一个子项
    - `slice-tool`表示操作区域（删除）
    - `slice-content`表示内容区域
      默认情况下，操作区域是不显示的，内容区域的大小是填充整个容器，操作区域始终位于内容区域的右面。

2、设置CSS样式

```
.slice-list ul li.slice-item{
        position:relative;
}
.slice-list .slice-content{
        position:relative;
        z-index:2;
        background:#fff;
        height:40px;
        line-height:40px;
        padding:0 10px;
}

.slice-list .slice-content,
.slice-list .slice-tool{
        -webkit-transition: all .3s ease-in-out;
        transition: all .3s ease-in-out;
}

.slice-list .slice-tool{
        position:absolute;
        top:0;
        right:0;
        bottom:0;
        text-align:right;
        color:#fff;
        z-index:1;
        background:red;
}
```

在上面的代码中，我们将操作区域（`slice-tool`）绝对定位，并且设置`z-index`为1,；而内容区域的`z-index`为2，且加上定位`position:relative`（如果不设置，z-index是无效的）

当设置了样式后，我们就只看到内容区域，操作区域被遮盖了。

接下来，我们为每一个子项绑定触摸事件：

```
var items = n.container.querySelectorAll('.slice-item');
for (var j = 0; j < items.length; j++) {
        items[j].addEventListener('touchstart', util.touchstart);
}
```

来看看touchstart方法：

```
touchstart: function(e) {
        e = e.changedTouches[0];
        util.pageX = e.pageX;
        util.activeItem = this.querySelector('.slice-content');
        util.isTouch = true;
        util.tool = this.querySelector('.slice-tool');
        util.width = util.tool.offsetWidth;
        util.left = util.activeItem.style.left;
        util.left = util.left ? parseInt(util.left) : 0;
        document.body.addEventListener('touchmove', util.touchmove);
        document.body.addEventListener('touchend', util.touchend);
}
```

在上面的代码中，我们是将 `touchmove` 和 `touchend` 添加到 `document.body` 上的，这样所有子项都可以通用，而不必添加到每一个子项。

拖动距离如何计算呢？其实就是用当前的 `e.pageX` 减去上一个时间点的 `e.pageX` ：

```
e.pageX - util.pageX
```

这里的util.pageX是用来保存上一个时间点的 `e.pageX` 。

最后将拖动值设置给内容区域即可：

```
util.left += e.pageX - util.pageX;
util.activeItem.style.left = util.left + 'px';
```

在上面的left表示总拖动距离，activeItem表示当前拖动的内容区域。

源码路径： `WebDemo/Mobile/sliceList`

移动联动选择器

移动联动选择器

由于移动端的空间有限，如果使用多个select，难免占据空间，而且不好看。

这一节我们来讲解如何实现上面示例图中酷炫的选择器。

`lazyPicker` 插件是纯js无任何依赖,调用方便兼容好。

1、创建模板

首先，我们需要一个能触发选择器的元素，这里我们使用input：

```
<input type="text" class="date-picker" placeholder="选择日期" />
```

2、JavaScript控制

我们首先要做的，就是给选择器分栏，不过你不需要做，因为全是用脚本生成，所以就需要一个关联完整的数据：

```
var data = {
        "item": [{
                "id": 1,
                "name": "水果",
                "child": [{
                        "id": 101,
                        "name": "苹果",
                        "child": [{
                                "id": 3,
                                "name": "甜的"
                        }]
                }]
    }]
 };
```

在`item`数组中，第一层存储的是第一级数据，其里面有一个child的属性，存储的是其对应的二级数据，二级数据下面有一个child属性，存储对应的三级数据。

那如何实现这种拖动功能呢？

其实只需要根据拖动的方向（上下），然后判断拖动距离，再用拖动距离除以每一个子项的高度就可以实现拖动。

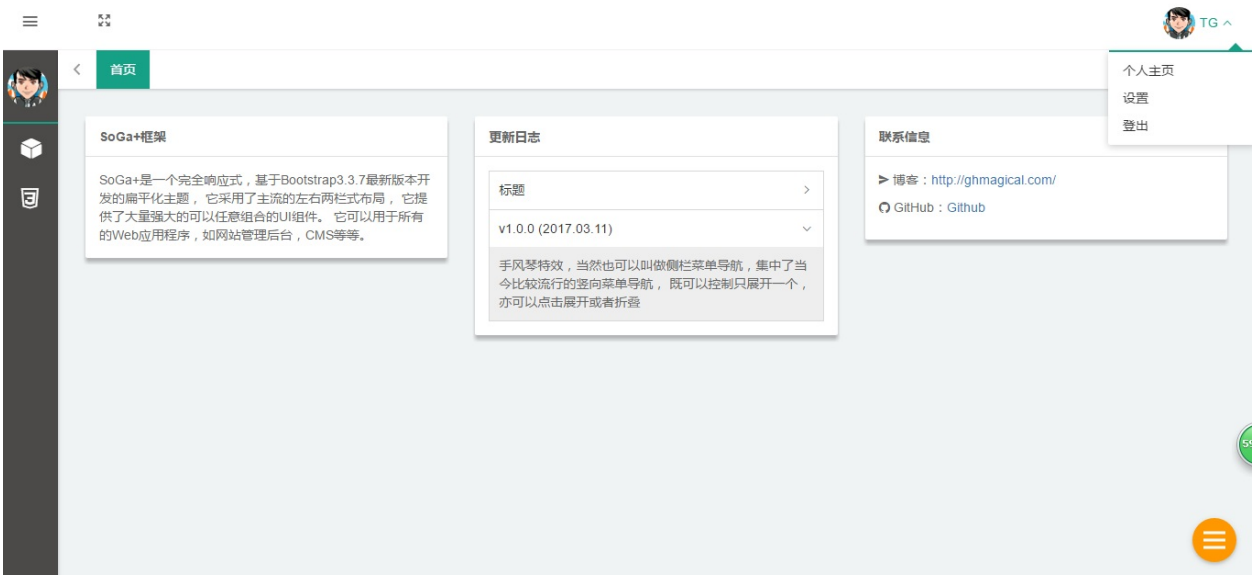
源码路径：`WebDemo/Mobile/picker`

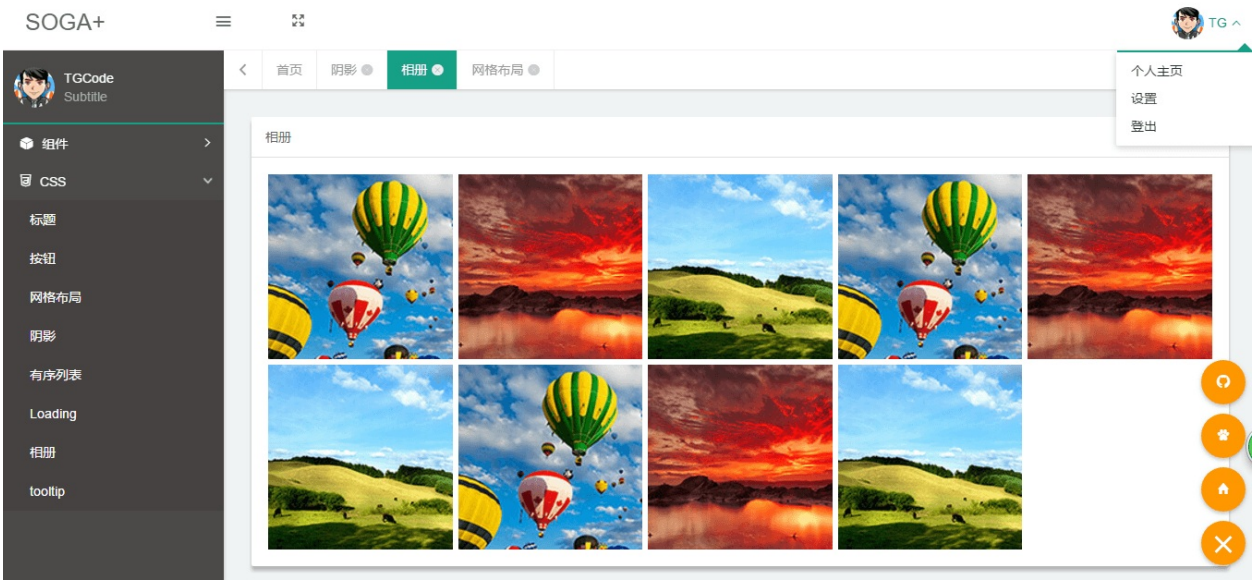
模板预览

模板预览

> 注：目前还在完善开发中，如果感兴趣，可以到源码下载里找到下载当前版本。

示例图：





线上访问地址：SoGa+主题框架

注意: 由于采取单页面形式,所以必须在服务器环境下运行

> 注：目前还在完善开发中，如果感兴趣，可以到源码下载里下载当前版本。

源码下载

源码下载

如果有任何问题或发现Demo中有错误，恳求告知！

源码下载链接: http://pan.baidu.com/s/1dEQ7d9B 密码: tav2

新增（20160116）：
链接: http://pan.baidu.com/s/1nuUuH5n 密码: 52i7

新增（20170203）
链接: http://pan.baidu.com/s/1gfJYEQJ 密码: ceda

SoGa+主题框架：链接: https://pan.baidu.com/s/1sk9sYol 密码: imk7

《JavaScript半知半解》电子版

《JavaScript半知半解》电子版

链接: http://pan.baidu.com/s/1bSFqpS 密码: 3dsa

注：未经作者允许，请不要将电子版发布到任何网站，谢谢。