# 1.流式布局与响应式布局

2020年9月3日 14:09

## 移动端开发主要内容

1. 移动端布局；
2. css3新增属性；
3. html5新增功能性标签；
4. touch事件；
5. 两个插件---better-scroll;swiper;

移动端网页对于显示设备的尺寸比较敏感，主要原因是因为移动端
设备基础尺寸太小；任何微小尺寸都会引发显示效果的大变化
在移动端我们要想实现网页，在移动平台的正确显示，必须重点考虑尺寸单位的设置
在pc端，我们采用大部分是px值，在移动端上我们就需要注意固定取值已经没办法适应
移动端平台众多的屏幕尺寸

## 移动端布局：

1. 流式布局
    a. 给所有的元素采用百分比设置宽度
2. 响应式布局
    a. 为了保证页面内容输出样式基本一致，我们通过判断显示设备的尺寸来改变网页元素的一行布局
3. 弹性盒子布局
    a. 既不会像流式布局一样拉的很长，又不会像响应式一样非常复杂
4. rem布局
    a. 手机型号太多，不同的型号对应不同的屏幕，尺寸不一样
    b. 手机操作系统的区别
    c. 相对值

## 流式布局

流式布局的特征：

宽度自适应，高度写死，并不是百分百还原设计图。

缺点

对于大屏幕来说,用户体验并不是特别好,有些布局元素会显得很长

1. 左侧固定,右侧自适应
2. 右侧固定，左侧自适应
3. 圣杯布局，两侧固定，中间自适应

## 响应式布局

为了保证页面内容输出样式基本一致，我们通过判断显示设备的尺寸来改变网页元素的一行布

局，这样一种布局方式我们叫做响应式布局；
媒体查询：@media
媒体设备类型：screen---显示设备；printer---打印设备；kindles---阅读器
超小设备（手机，小于 768px）lg

小型设备（平板电脑，768px 起） md
@media screen and (min-width:768px) { ... }

中型设备（台式电脑，992px 起） sm
@media screen and (min-width:992px){ ... }

大型设备（大台式电脑，1200px 起） xm
@media screen and (min-width:1200px){ ... }

```
@media screen and （max-width:XXXpx）{
    .box{width:100%;}
}
```

响应式布局的弱点：
应对功能复杂，结构复杂的网站时；这个时候我们的响应式实现方式就太复杂了；

## 弹性盒子布局

Flex 是 Flexible Box的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。
特点：
直接改变原来写页面采用px等固定值来取值的方式；
改为使用关键字取值，等比例实现；

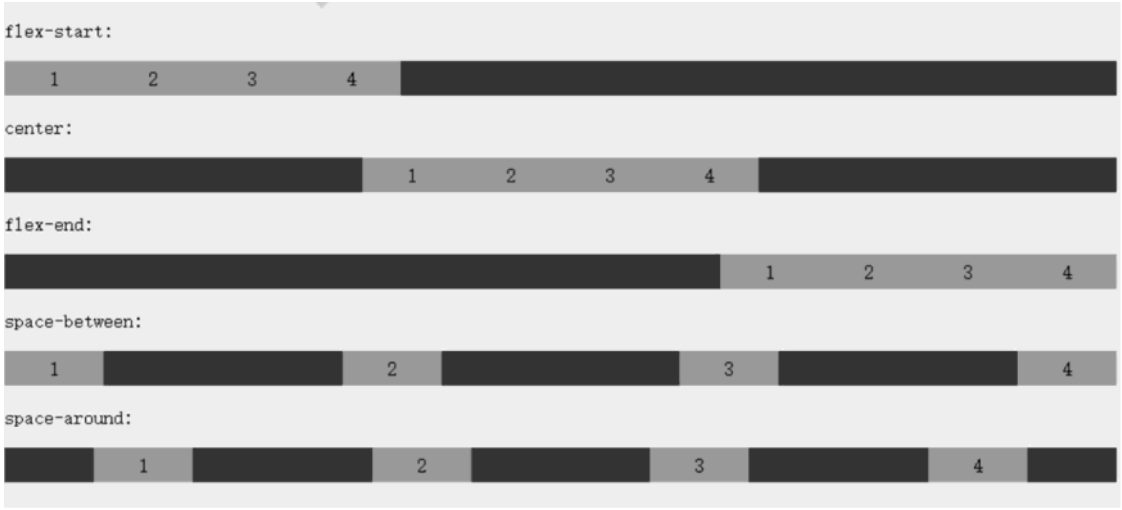
# 写在父级控制子级

1. 任何一个容器都可以指定为 Flex 布局。

```
.box{
   display: flex;
}
```

2. 弹性盒子的排列方向：flex-direction
   - row：横向从左到右排列（左对齐），默认的排列方式。
   - row-reverse：反转横向排列（右对齐，从后往前排，最后一项排在最前面。
   - column：纵向排列。
   - column-reverse：反转纵向排列，从后往前排，最后一项排在最上面。
3. 水平对齐方式 justify-content

| | |
|---|---|
| space-around | 子级项目平均分布在一行，两侧子级留有一半的间隔空间 |
| space-between | 子级项目平均分布在一行，两侧子级与边线对齐 |
| Center | 子级项目紧挨着居中填充 |
| flex-start | 子级项目向行头紧挨着填充 |
| flex-end | 子级项目向行尾紧挨着填充 |




4. 垂直对齐方式align-items
   a. centert 垂直居中
   b. flex-start 顶上边
   c. flex-end 顶着下边
5. 换行 flex-wrap
   a. nowrap - 默认， 弹性容器为单行。该情况下弹性子项可能会溢出容器。
   b. wrap - 弹性容器为多行。
   c. wrap-reverse -反转 wrap 排列。
6. align-content 属性
   a. stretch - 默认。各行将会伸展以占用剩余的空间。
   b. flex-start - 各行向弹性盒容器的起始位置堆叠。
   c. flex-end - 各行向弹性盒容器的结束位置堆叠。
   d. center -各行向弹性盒容器的中间位置堆叠。
   e. space-between -各行在弹性盒容器中平均分布。
   f. space-around - 各行在弹性盒容器中平均分布，两端保留子元素与子元素之间间距大小的一半。

## 子级控制自身的属性

7. order 属性设置弹性容器内弹性子元素的属性:
   a. order的值可以取正负值都行；数字越大，排为越靠后；数字越小排为越靠前 默认是0；
8. align-self:给子级设置单独的垂直对齐；
   a. center：居中;
   b. flex-start上边;
   c. flex-end：底边;
9. width设置 flex：
   - flex-shrink:压缩子级占据的父级比例：如果都是1;那么是等分父级；--->如何子级比较大的情况下；压缩子级；
   - flex-grow:扩展子级，占据父级的比例：如果是1；等分父级 ---->如果子级比较小的情况下扩展子级；
   - ,flex-basis:弹性盒子级的宽度：百分比；px值；

## rem布局

移动端开发逐渐成熟起来，现在大部分公司开始进行PC和移动端单独开发

痛点：

手机型号众多，不同的型号对应不同的屏幕，尺寸也不一样

操作系统的差别

rem不是固定值单位：是一种倍数取值

测量尺寸/HTML的font-size=***rem

HTML的font-size在chrome浏览器中，默认是16px，不识别低于12px的字体尺寸

实际工作中，设计人员给我们的设计图一般是以iphone6为参考基准设置的，iphone6的手机屏幕物理尺寸是375px

物理尺寸就是手机屏幕硬件的最大显示尺寸

现在的手机都是高清屏，实际上物理尺寸是375，如果需要清晰显示设计图的尺寸，我们就需要两个设计图的像素点来覆盖一个物理像素的像素点才能正常显示内容

物理像素点：设计像素点=1：2 所以我们的设计图一般是750px

那么我们按照设计图上的标准来获取rem值应该怎么算?

如果设计图是750 测量尺寸/2/html的font-size

如果是375 测量尺寸/html的font-size

```
html{font-size: 100px; }
body{font-size: 14px; }
. banner{ width:3.45rem; height: 100%; background:口skyblue;margin:0 auto ; border-radius: . 4rem;}
```

css设置font-size

Html{font-size：calc(100vw/750*100)}

如果页面尺寸是375 1rem=50px 如果是750 1rem=100px

动态改变HTML的font-size

```
var htm=document.documentElement
Htm.style.fontSize=htm.clientWidth/10+"px"
```

插件 flexible

自动计算font-size

10rem=屏幕宽度

```
<script src="http://g.tbcdn.cn/mtb/lib-flexible/0.3.2/??flexible_css.js,flexible.js"></script>
```

| 设计尺寸 | 物理尺寸 | Html font-size | Rem |
|---|---|---|---|
| 750px | 375px | 37.5 | 37.5px |
| 375px | 375px | 37.5 | 37.5px |

Iconfont 阿里巴巴字体图标库

https://www.iconfont.cn/home/index?spm=a313x.7781069.1998910419.2

1. 选择图标 加入购物车

2. 在购物车中下载代码



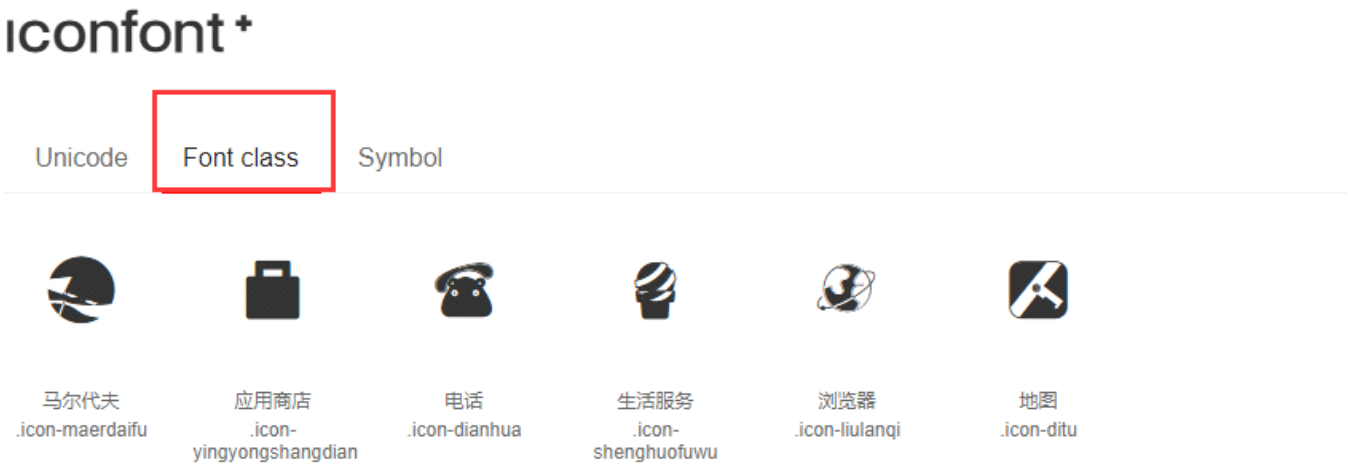
3. 下载完成后解压文件包

| 名称 | 修改日期 | 类型 | 大小 |
|---|---|---|---|
| demo.css | 2020/8/1 11:05 | 层叠样式表文档 | 9 KB |
| demo_index.html | 2020/8/1 11:05 | Chrome HTML D... | 12 KB |
| iconfont.css | 2020/8/1 11:05 | 层叠样式表文档 | 7 KB |
| iconfont.eot | 2020/8/1 11:05 | EOT 文件 | 7 KB |
| iconfont.js | 2020/8/1 11:05 | JavaScript 文件 | 30 KB |
| iconfont.json | 2020/8/1 11:05 | JSON 文件 | 2 KB |
| iconfont.svg | 2020/8/1 11:05 | SVG 文档 | 27 KB |
| iconfont.ttf | 2020/8/1 11:05 | TrueType 字体文件 | 7 KB |
| iconfont.woff | 2020/8/1 11:05 | WOFF 文件 | 5 KB |
| iconfont.woff2 | 2020/8/1 11:05 | WOFF2 文件 | 4 KB |

4. 在html中 引用iconfont.css文件

```
<link rel="stylesheet" type="text/css" href="icon/iconfont.css"/>
```

5. 打开demo_index.html，选择fontclass




6. 复制下方的类名，粘贴到页面标签

```
<span class="iconfont icon-maerdaifu"></span>
```

# 2.CSS3新增属性

2020年9月14日 9:27

## 选择器

### 属性选择器

input【type='checkbox'】

标签中常见的属性：href，style，src，title，name，id，class，type，target

### 构造伪类选择器

父级下的第一个元素 Parent>child:first-child

Ul > li :first-child{}

父级下的最后一个元素 parent > child:last-child

父级下任意位置 parent >child:nth-child(n)

### 同类兄弟元素选择器

某一个范围下，同类的兄弟元素

第一个 parent child：first-of-type

最后一个 parent child：last-of-type

任意位置 parent child：nth-of-type（n）

### 状态伪类选择器

鼠标悬停 ： hover

获取焦点：focus

在元素之前：before

在元素之后：after

当元素不可点击时： disabled

## css新增样式属性

### 怪异盒模型：也叫ie合模型，一般用于移动端。

box-sizing:border-box;

### 长度单位

Px %

Em 以父级font-size为准

rem 以根元素HTML的font-size为准

vw/vh 将屏幕分成了100份 vw：宽 vh:高

### 文字描边

Text-stroke：线宽 颜色

- 谷歌浏览器chrome;-webkit-
- ie：-ms-;
- 火狐：-moz-;
- Opera：-o-
- 这些私有前缀的作用是，让浏览器能够有效识别css3的样式属性；

### 文字阴影

Text-shadow: 水平偏移量 ， 垂直偏移量，模糊值 ， 颜色

```
.msg{
        text-shadow: 0px 0px 1px #0077AA ;
    }
```

盒阴影

Box-shadow：水平偏移量 ， 垂直偏移量，模糊值 ， 颜色

```
.box:hover{
            box-shadow: 0px 10px 10px red;
    }
```

透明色：

rgba（0-255，0-255，0-255，0-1）

透明度：元素透明

opacity：0-1

如果opacity设置为0，元素不显示，但是占位

如果设置为display：none，页面不会加载元素，不占位

圆角：

Border-radius

```
border-radius: 5px;
1. border-top-left-radius: 5px;
2. border-top-right-radius: 5px;
3. border-bottom-right-radius: 5px;
4. border-bottom-left-radius: 5px;
```

最大不能超过50%，超出50%，按50%计算

渐变：

Background-image：

线性渐变

Linear-gradient

1.默认从上到下 linear-gradient(red,green,yellow);

2.加入方向linear-gradient(to right, red,green,yellow);

3.对角 linear-gradient(to right bottom, red,green,yellow);

4.角度 linear-gradient(60deg, red,green,yellow);

径向渐变

radial-gradient()从内向外渐变

改变渐变形状：circle圆形，默认椭圆 ellipse

radial-gradient(circle,yellow,red,green);

背景图新增

Background-size：

1. cover：拉伸图片，铺满整个标签，缺点：会裁剪图片
2. Contain 显示全部图片，保证图片不变形，有时会铺不满元素
3. 有时候我们的背景图可以直接设置宽高尺寸

第一个值是水平方向

第二个值是垂直方向

Background-clip

1. 默认值：content-box
2. 内边距 padding-box
3. 边框 border-box

双背景：

两个背景图之间用逗号隔开

```
background: url(images/img.jpg) no-repeat left top,url(images/img2.jpg) no-repeat right bottom;
```

# 03.css3变形与过度

2020年9月15日 8:50

transform转换属性

位移：

transform：translate（水平方向，垂直方向）

transform：translateX（）水平方向移动

transform：translateY（）垂直方向移动

缩放

transform: scale(2); 放大两倍

transform: scale(0.5); 缩小到0.5倍

transform: scaleX(2); 缩放宽度

transform: scaleY(0.5); 缩放高度

transform: scale(2,0.5); 水平缩放2倍，垂直缩放0.5倍

旋转

transform: rotate(70deg); 把元素旋转70度

transform: rotateX(40deg); 元素沿着x轴旋转40度

transform: rotateY(91deg);元素沿着y轴旋转91度

倾斜

transform：skew（20deg，30deg）把元素沿水平方向倾斜30度，垂直方向倾斜20度

transform: skewX(30deg); 水平倾斜，正值向左，负值向右

transform: skewY(30deg); 垂直方向倾斜，正值向右，负值向左

**注意：一个元素transform只能写一个 但是，可以写多个值（多个效果）**

transform: skew(30deg,20deg) scale(2) rotate(80deg) translate(200px);

变形的运动中心 transform-origin

默认是center

可以取值为 left  right top  bottom

也可以取值为固定值

transform-origin：20px 50px

第一个值是距离左边，第二个值是距离顶部

## transition 过渡

transition-property ： css的属性名称 或者all

transiton-duration： 动画的持续时间

transition-timing-function： 运动曲线

一般情况liner 线性运动

ease 平滑运动

ease-in 逐渐加速，速度越来越快

ease-in-out 先加速再减速

ease-out 逐渐减速，速度越来越慢

transition-delay: 1s; 动画延迟时间

**一般情况transition不分开设置，直接写复合属性**

**transition: left 100ms linear 2s;**

# 04.3d变形与动画

2020年9月16日 14:15

## 3d变形

我们所有的变形运动，都存在3个轴 x y z

想要实现3d效果我们要给父级设置两个属性

transform-style：preserve-3d

perspective: 景深，眼睛到元素的距离，none number

## 动画：设置关键帧

```
@keyframes name {
    from {}
    to{}
}
@keyframes name {
    0% {}
    15%{}
    15.5{}
    100%{}
}
```

百分比以持续时间为单位

## 调用动画

animation-duration: 5s; 持续时间

animation-timing-function: linear; 运动曲线

animation-delay: 0s; 延时

animation-iteration-count: infinite; 循环次数无限循环 或者是 数字

animation-direction: normal; 是否循环交替反向播放

alternate:奇数次正向，偶数次反向

alternate-reverse:奇数次反向，偶数次正向

reverse：反向

animation-fill-mode: none; 动画填充模式

forwards 动画结束后，停留在最后的位置

backwards 保留动画开始的初始状态

animation-play-state: running;  动画的执行状态

默认running

暂停 paused

animation-name: rotat; 名字

# 05.touch事件

2020年9月17日 9:51

## 视窗设置：

```
<meta name="viewport" content="width=device-width,initial-scale=1,minimum-scale=1,maximum-scale=1,user-scalable=no" />
```

name="viewport" 视窗

width=device-width,可以控制视窗大小，可以是一个固定的值，device-width是设备宽

initial-scale=1, 初始缩放比

minimum-scale=1, 最小缩放比例

maximum-scale=1, 最大缩放比例

user-scalable=no 是否允许用户自主缩放

## touch事件

在pc端触发事件的鼠标

在移动端，触发事件的是手指，由于手机的灵敏度较高，手机屏幕是双层屏幕，造成了使用点击事件的时候，大约有300ms的延时

因为所有屏幕都有传感器，所有我们使用touch（触碰事件）事件，来解决延时

于是乎，我们的手机上就有了三种事件

点击　鼠标按下　触摸事件

注意：在移动端绑定事件的时候，不再使用dom0级事件，而是是dom2级事件

```
$('.box').addEventListener('click',function(){
    console.log('点击事件')
})
$('.box').addEventListener('mousedown',function(){
    console.log('鼠标按下事件')
})
$('.box').addEventListener('touchstart',function(){
    console.log('我摸到你了')
})
```

我们发现这三种事件的优先级：

Touch》鼠标按下》点击事件

## touch事件的类型

开始触摸 touchstart

触摸点移动 touchmove

手指离开 touchend

## 触碰点的获取

注意：触碰点的获取都需要在同一个事件下

直接在屏幕上的触碰点 touches

在元素上的触碰点 targetTouches

发生了状态改变的触碰点 changedTouches

我们获取到的触碰点信息，是一个列表，我们要考虑到触发touch事件的时候有几个触碰点

```
▼0: Touch
    clientX: 116
    clientY: 44
    force: 1
    identifier: 0
    pageX: 116
    pageY: 44
    radiusX: 11.5
    radiusY: 11.5
    rotationAngle: 0
    screenX: 757
    screenY: 258
  ▶target: div.box
  ▶__proto__: Touch
```

clientX: 116 触碰点到浏览器窗口的距离

clientY: 44

force: 1 压力

identifier: 0 触摸点的唯一表示 （ID）

pageX: 116 触碰点到页面的距离

pageY: 44

radiusX: 11.5 触摸点半径
radiusY: 11.5
rotationAngle: 0 旋转角度

screenX: 757 触碰点到屏幕的距离
screenY: 258

我们在同时运行三个事件后发现，在touchend事件（手指离开屏幕），e.touches和e.targetTouches是获取不到触碰点的，legth为0

```
$('.box').addEventListener('touchstart', function(e) {
    console.log(e.touches)
    console.log(e.targetTouches)
    console.log(e.changedTouches)
})

$('.box').addEventListener('touchmove', function(e) {
    // console.log(e.touches)
    // console.log(e.targetTouches)
    // console.log(e.changedTouches)
    console.log("手指移动")
})
$('.box').addEventListener('touchend', function(e) {
    console.log(e.touches)          //屏幕触碰点
    console.log(e.targetTouches)    //元素触碰点
    console.log(e.changedTouches)
})
```

```
▶ TouchList {0: Touch, length: 1}                    4.触碰点.html:51
▶ TouchList {0: Touch, length: 1}                    4.触碰点.html:52
▶ TouchList {0: Touch, length: 1}                    4.触碰点.html:53
479 手指移动                                          4.触碰点.html:60
▼ TouchList {length: 0}                              4.触碰点.html:63
    length: 0
  ▶ __proto__: TouchList
▶ TouchList {length: 0}                              4.触碰点.html:64
▶ TouchList {0: Touch, length: 1}                    4.触碰点.html:65
```

当我们在元素上开始触摸，手指移动到元素之外放开，此时，我们发现，依然是touchend事件获取不到e.touches和e.targetTouches

touchstart：e.touches,e.targetTouches,e.changedTouches获取到的值是一样的

touchmove：e.touches,e.targetTouches,e.changedTouches获取到的值是一样的

touchend：当我们手指离开屏幕的时候，e.touches,e.targetTouches,是获取不到的，也就是说，当手指离开屏幕的时候，没有touch对象

移动端滑动事件：

手指的误操作会造成touch事件的执行，为了避免这些误操作，我们需要做一些判断，来区分，是不是误操作

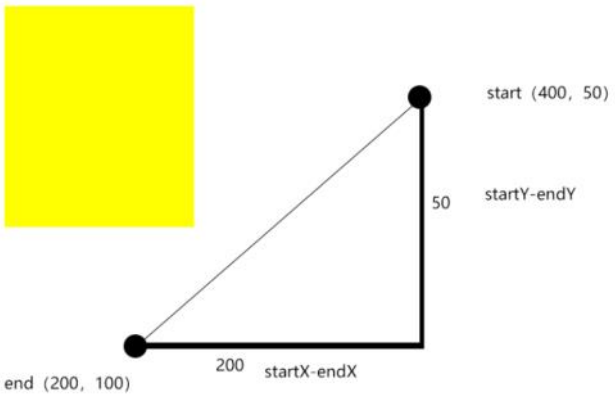
判断，是否是认为的滑动

判断依据：

判断是滑动还是点击

判断滑动的方向

1. 通过touch事件模拟
2. 计算水平移动距离和垂直移动距离
3. 如果水平移动距离大于垂直距离，说明是水平滑动
4. 水平：比较两个点的x轴方向，如果越来越小，说明是向左
5. 垂直：比较两个点的y轴方向，如果越来越小，说明是向上
6. 移动距离大于30，才能被判定为滑动




如果水平移动距离大于垂直距离，说明是水平滑动，反之

水平：
比较两个点x轴的方向

垂直：
比较两个点y轴的方向

移动端点击事件

点击事件是在150ms之内完成的

1. 手指开始触摸的时候，记录时间
2. 手机离开屏幕的时候，计算时间差
3. 创建一个全局变量，当手指在屏幕上滑动的时候，关闭开关
4. 判断：时间差在150之内，没有滑动
5. 重置开关

### touch事件封装

既然是封装事件，我们要知道事件的三要素

事件源对象

事件类型

事件监听器

### 点击事件封装

```
tap: function(el, callback) {
    var startTime = null;
    var flag = true;
    el.addEventListener('touchstart', function() {
        startTime = new Date();
    })
    el.addEventListener('touchmove', function() {
        flag = false; //关闭开关
    })
    el.addEventListener('touchend', function(e) {
        var diffentTime = new Date() - startTime; //时间差
        if (diffentTime < 150 && flag) {
            //执行监听器
            callback(e)
        }
        flag = true;
    })
```

### 滑动事件封装

```
swipe: function(el, direction, callback) {
    var start = null;
    var end = null
    el.addEventListener('touchstart', function(e) {
        start = {
            x: e.targetTouches[0].clientX,
            y: e.targetTouches[0].clientY
        }
    })
    el.addEventListener('touchmove', function(e) {
        end = {
            x: e.targetTouches[0].clientX,
            y: e.targetTouches[0].clientY
        }
    })
    el.addEventListener('touchend', function(e) {
        //判断start和end是否有值
        if (start && end) {
            //判断是垂直还是水平
            var diffentX = start.x - end.x;
            var diffentY = start.y - end.y;
            var directionX = diffentX > 0 ? "left" : "right";
            var directionY = diffentY > 0 ? "top" : "bottom";
            if (Math.abs(diffentX) >= Math.abs(diffentY)) { //证明是水平滑动
                if (direction === directionX && Math.abs(diffentX) > 30) {
                    callback(e)
                }
            } else { //垂直
                if (direction === directionY && Math.abs(diffentY) > 30) {
                    callback(e)
                }
            }
            stat = null
            end = null
        }
    })
```

# 06.h5新增与web存储

2020年9月21日 8:31

## HTML5的发展

HTML5的发展，不得不追溯到1991年，那时候HTML1才刚刚开始研发，到1993年才发布，一直到1999年更新到第四代之后，便不疾而终，毫无发展。大概十年之后，互联网技术不断发展，人们发现原来的HTML已经不适合时代的发展，他们需要更加灵活、更加智能的HTML技术来支撑互联网的未来，于是在2008年的时候，HTML5产生了。

可惜，HTML5产生之后，并没有像人们预料的那样迅速走红，而是掉入了一个漫长无声的优化时期，直至2014年，HTML5完成了终的标准定制并全面开放，成为了互联网行业一个划时代的标志，从此HTML5死灰复燃了。

## HTML5新特性

1. 用于绘画的 canvas 元素
2. 用于媒介回放的 video 和 audio 元素
3. 对本地离线存储的更好的支持
4. 新的特殊内容元素，比如 article、footer、header、nav、section
5. 新的表单控件，比如 calendar、date、time、email、url、search

## 语义化标签更加细化

HTML5将一些结构类标签，细分出语义化结构标签

```
<header>网页的头部</header>
<footer>网页的底部</footer>
<main>用来表示文档的主体部分
    注意：一个文档，只能出现一个main标签，不能被嵌套在需要重复出现的标签中
</main>

<nav>表示导航</nav>
<aside>侧边栏</aside>
<article>表示文章</article>

<hgroup>标题管理</hgroup>
<section>章节</section>
<address>联系信息</address>
```

**注意：兼容性**

## H5新增的控件

可输入的下拉框

```
<input type="text" list="box"  />
    <datalist id="box"> <!--选项-->
        <option value ="1234567"></option>
        <option value ="12345678"></option>
```

```
        <option value ="1234567890"></option>
        <option value =""></option>
    </datalist>
```

list属性指向datalist的id

input新增类型

maxlength：输入的最大长度

- 搜索框：<input type="search" name="" id="" value="" />
- url地址栏：<input type="url" name="" id="" value="" />
- 电话：<input type="tel" name="" id="" value="" />
- email:<input type="email" name="" id="" value="" />
- color:<input type="color" name="" id="color" />
- number:<input type="number" name="" id="" value="" />
- 数字拉动条：<input type="range" name="" id="range" value="" max="50" min="10" />
  max：最大值　min：最小值
- 时间选择：<input type="time" name="" id="time" value="" />
- 日期和时间：<input type="datetime-local" name="" id="" value="" />
- 周和年：<input type="week" name="" id="" value="" />
- 年月：<input type="month" name="" id="" value="" />
- 日期：<input type="date" name="" id="" value="" />

搜索框：

url地址栏：

电话：

email:

color:

number:

数字拉动条：

时间

时间选择：

日期和时间：2020/09/17 09:07

周和年：2020 年第 38 周

年月：2020年11月

日期：2020/09/01

Webstorage

cookie

必须在服务器环境下运行

发送请求的时候，会携带cookie

cookie最大 4k

设置过期时间

localStorage

最大存储 5m

永久的，除非手动删除

一旦存储，在这个浏览器上打开的页面都能访问到

sessionStorage

临时存储

会话级：关闭浏览器自动删除

会话：一个页面跳转到另一个页面，这个过程，我们称之为发生了一次会话

API：

存：setItem(name,value)

取：getItem(name)

删：removerItem(name)

清空：clear()

```
$('.btnList').addEventListener('click', function(e) {
    var src = e.target;
    if (src.innerHTML === "添加") {
        localStorage.setItem('user', $('#user').value)
        localStorage.setItem('age', $('#age').value)
        //如果key相同，重复添加的时候会替换值
    } else if (src.innerHTML === "获取") {
        console.log(localStorage.getItem('user'))
        console.log(localStorage.getItem('age'))
    } else if (src.innerHTML === "删除") {
        localStorage.removeItem('user')
    } else if (src.innerHTML === "清空") {
        localStorage.clear()
    }
})
```

# 07.drag&drop拖放

2020年9月23日 14:17

## 拖放

可以理解为，将一个元素拖动到另外一个位置，中间经过了一些元素

被拖动的元素----源对象

经过的元素---过程对象

目标区域---目标对象

## 源对象

要想让某个元素可以被拖动，必须设置draggable="true"，意思是设置当前元素可以被拖动，默认值是false

图片，a标签默认值是true

源对象有三个事件：

源对象开始拖动 dragstart

源对象拖动过程中 drag

源对象结束拖动 dragend

```
//源对象的事件
$('#img').addEventListener('dragstart', function() {
    console.log('开始拖动')
})
$('#img').addEventListener('drag', function() {
    console.log('拖动过程')
})
$('#img').addEventListener('dragend', function() {
    console.log('拖动结束')
})
```

## 过程对象

源对象（以鼠标位置为基准）开始进入过程对象 dragenter

源对象在过程对象中移动 dragover

源对象离开过程对象 dragleave

```
//过程对象事件
$('.box').addEventListener('dragenter',function(){
    console.log('他来了~')
})
$('.box').addEventListener('dragover',function(){
    console.log('别乱动')
})
$('.box').addEventListener('dragleave',function(){
    console.log('他走了...')
})
```

## 目标对象

源对象进入目标对象drop

```
$('.box2').addEventListener('drop',function(){
    console.log('留下来')
})
```

但是，我们发现一个问题，此时dorp事件是不执行的，因为页面中所有的元素都默认是过程对象，过程对象的dragover事件有一个默认行为，就是当dragover事件执行的时候，drop事件不执行，所以我们要阻止dragover的默认事件才能触发drop事件

```
//目标对象
$('.box2').addEventListener('dragover',function(e){
    //页面中所有的元素都默认是过程对象，dragover执行的时候不执行drop，
    // 我们必须先阻止dragover默认事件
    e.preventDefault()
})
```

## 拖动元素的步骤

1. 先给元素设置draggable=“true”
2. 给源对象绑定开始拖动事件，规定开始拖动的时候要发生点什么
3. 给过程对象绑定拖动经过事件，规定源对象经过过程对象的时候要发生点什么
4. 阻止目标对象的dragover事件下的默认行为
5. 给目标对象绑定drop事件，规定拖动结束要发生点什么

## dataTransfer对象

dataTransfer是拖放事件对象中的一个属性，该 属性用于源对象与目标对象之间的数据传输

设置 setData（name，value）

获取getData（name）

清空 clearData（）

setDragImage（element，x，y）用来设置拖放的图标

element：表示拖拽时鼠标下的图片（canvas）

x，y：分别表示鼠标在图片上的偏移量

# 08.上传

2020年9月24日 14:00

<input type="file" name="" id="file" multiple />

multiple：多选

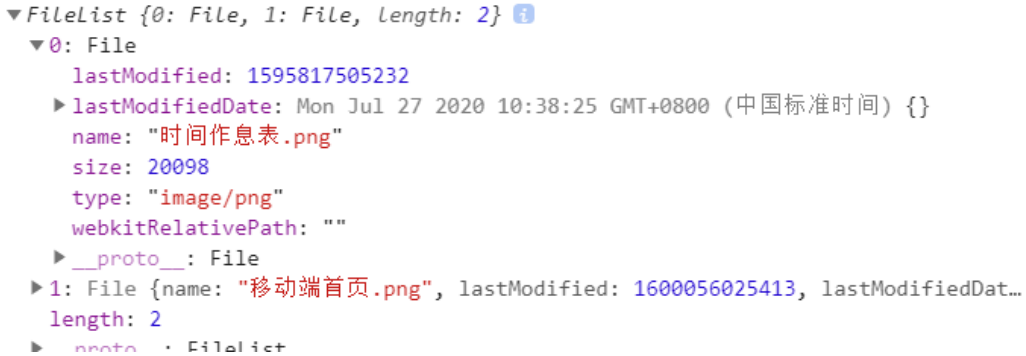
## 监听事件：

通过change事件监听上传

files属性获取上传的文件集合

```
$('#file').addEventListener('change', function() {
    var filess=this.files
```

多文件上传.html:36




name：名字

type：文件类型

size：文件大小

## fileReader（）

用来把文件读入内存，并且读取文件中的数据

1. 创建fileReader对象

```
var f=new FileReader();
```

2. 调用解析方法（参数传入file）

```
f.readAsDataURL(files)
```

f.readAsDataURL()//路径（base64格式）

f.readAsText()//读取文本文档中的文本内容

3. 取值f.result

## fileReader事件

读取中断 abort

读取出错 error

读取开始 loadstarts

正在读取 progress

读取成功 load

读取完成(不论成败) loadend

## 多文件上传

```
function createImg(str, parent) {
    var img = document.createElement('img');
    img.src = str;
    parent.appendChild(img)
}
$('#file').addEventListener('change', function() {
    var filess=this.files
    console.log(filess)
    for(var i=0;i<filess.length;i++){
        read(filess[i])
    }
})
function read(files){
    var f=new FileReader();
     f.readAsDataURL(files)
     f.onload=function(){
         createImg(f.result,$('#box'))
     }
}
```

# 09.多媒体

2020年9月25日 8:37

## 音频audio，视频video

```
<video width="800" height="">
    <source src="" type="video/mp4"></source>
    <source src="myvideo.ogv" type="video/ogg"></source>
    <source src="myvideo.webm" type="video/webm"></source>
    当前浏览器不支持 video直接播放
</video>
<audio src="./src/music.mp3" width="500" controls loop muted>
    <source src="./src/music.flac" type="audio/flac"></source>
    当前浏览器不支持audio
</audio>
```

1. audio标签用于表示音频
2. video标签用于表示视频
3. soure标签为了兼容不同浏览器
4. 如果都不支持，用一段文字表示

https://www.runoob.com/tags/ref-av-dom.html

## 标签自带的控件属性

controls:显示播放控件 （必须写的，如果不写，不显示标签）

autoplay:自动播放（被禁用）

loop:循环播放

muted:静音播放

## DOM属性

播放状态：paused true 指示音频/视频已暂停，否则为 false。

当前播放时间：currentTime

文件时间总长：duration

是否结束：ended

## 方法

播放：play()

暂停：pause()

重新加载：load()

## 事件

Ended 播放结束事件
Play 正在播放

# 10.swiper

2020年9月25日 14:55

## Swiper

1. 开源、免费、强大的触摸滑动插件（微场景）
2. 常用于移动端网站内容滑动
3. 可以实现焦点图，触屏tab切换，多图切换

1.首先加载插件，需要用到的文件有swiper-bundle.min.js和swiper-bundle.min.css文件

2.HTML内容。

```
<div class="swiper-container">
    <div class="swiper-wrapper">
        <div class="swiper-slide">Slide 1</div>
        <div class="swiper-slide">Slide 2</div>
        <div class="swiper-slide">Slide 3</div>
    </div>
    <!-- 如果需要分页器 -->
    <div class="swiper-pagination"></div>

    <!-- 如果需要导航按钮 -->
    <div class="swiper-button-prev"></div>
    <div class="swiper-button-next"></div>

    <!-- 如果需要滚动条 -->
    <div class="swiper-scrollbar"></div>
</div>
```

导航等组件可以放在container之外

3.你可能想要给Swiper定义一个大小，当然不要也行。

```
.swiper-container {
```

```
    width: 600px;
    height: 300px;
}
```

4.初始化Swiper

```
<script>
  var mySwiper = new Swiper ('.swiper-container', {
    direction: 'vertical', // 垂直切换选项
    loop: true, // 循环模式选项

    // 如果需要分页器
    pagination: {
      el: '.swiper-pagination',
    },

    // 如果需要前进后退按钮
    navigation: {
      nextEl: '.swiper-button-next',
      prevEl: '.swiper-button-prev',
    },

    // 如果需要滚动条
    scrollbar: {
      el: '.swiper-scrollbar',
    },
  })
  </script>
```

5.属性

```
      direction: 'horizontal', // 垂直切换选项  vertical垂直
      loop: true, // 循环模式选项
      initialSlide:1,//默认显示下标
      speed:300,//切换速度
```

effect:"slide",//切换效果

'slide'（普通切换、默认），

"fade"（淡入）

"cube"（方块）

"coverflow"（3d流）

"flip"（3d翻转）。

autoplay:true, 自动轮播

autoplay: {

delay: 300,

stopOnLastSlide: false,

disableOnInteraction: true,

},

### 6.注册事件

on： { //this指代swiper实例

}

# 11.canvas

2020年9月26日 13:59

<canvas>元素用于在html中图形绘制

<canvas>标签是视图容器

## 创建一个canvas画布

```
<canvas id="can" width="1000" height="600">
    您的浏览器不支持canvas
</canvas>
```

不要通过css设置canvas宽高，容易出bug

## 创建canvas对象 ctx

想要绘制图形，首先要获取元素，然后创建canvas对象，起到一个承接上下文的作用

```
var ctx=$('#can').getContext('2d')
```

使用ctx绘制图形，需要先设置样式，然后再绘制图形

## 设置颜色

ctx.strokeStyle="" //线框颜色

ctx.fillStyle="" //填充颜色

## 绘制矩形

ctx.strokeRect(x,y,width,height) 绘制线框

ctx.fillRect(x,y,width,height) 绘制填充矩形

x,y：原点坐标

canvas默认的原点坐标是左上角

ctx.clearRect(x,y,width,height) 清除指定区域矩形

## 绘制线条

起点ctx.moveTo(x,y)

终点ctx.lineTo(x,y)

线条宽度ctx.lineWidth='30';

线条断点样式ctx.lineCap="round"

butt默认值平角 round 圆形线帽 quare 正方形线帽

两条直线之间的拐角ctx.lineJoin="round"

bevel 斜角 round 圆角 miter 默认值

ctx.stroke()//绘制轮廓

ctx.fill() //填充

```
//绘制三角形
ctx.beginPath();
ctx.strokeStyle="blue";
ctx.moveTo(100,100);
ctx.lineTo(100,250);
ctx.lineTo(200,250);
// ctx.lineTo(100,100)
ctx.closePath()//闭合路径点
ctx.stroke()
```

## 绘制弧

ctx.arc(x,y,r,起始弧度,结束弧度,true/false)

x,y:圆心坐标,

弧度:Math.PI/180*角度,

true是逆时针旋转

false顺时针旋转 默认

```
ctx.arc(300,300,100,0,Math.PI,true);
ctx.stroke();
ctx.beginPath()
ctx.arc(300,300,60,260*Math.PI/180,300*Math.PI/180,false)
ctx.stroke();
```

两条切线之间的弧

Ctx.arcTo(x1,y1,x2,y2,r)

x1,y1:起点坐标

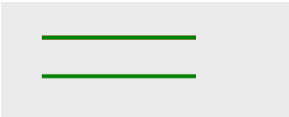
x2,y2:终点坐标

```
//两条切线之间的弧
ctx.beginPath();
ctx.moveTo(100,100);
ctx.lineTo(300,100);
ctx.arcTo(350,100,350,150,50);
ctx.lineTo(350,200)
ctx.stroke()
//扇形
ctx.beginPath()
ctx.strokeStyle="#007AFF";
ctx.fillStyle="yellow";
ctx.moveTo(200,200);
ctx.arc(200,200,100,190*Math.PI/180,300*Math.PI/180,false);
ctx.closePath()
ctx.stroke()
ctx.fill()
//圆形
ctx.beginPath();
ctx.arc(400,400,100,0,2*Math.PI)
ctx.stroke()
ctx.fill()
ctx.beginPath()
ctx.arc(400,400,80,0,2*Math.PI)
ctx.stroke()
```

## beginPath，closePath（）

beginPath（）

```
ctx.beginPath();
ctx.strokeStyle="red"
ctx.lineWidth=5;
ctx.moveTo(100,100);//---1
ctx.lineTo(300,100);//---2
ctx.stroke();
// ctx.beginPath();
ctx.strokeStyle="green"
ctx.moveTo(100,150);//---3
ctx.lineTo(300,150);//---4
ctx.stroke();
```



canvas中的绘制方法（strok，fill），都会以‘上一个beiginPath（）’之后的所有路径为基础进行绘制

第一个stroke 执行了12，颜色红色

第二个stroke 执行了1234，绿色(其中，12绿色覆盖了第一次画的红色)

不管你moveTo()把画笔移动到了哪,只要不beginpath,那么我们都是在画一条路径

路径的概念：

closePath（）

closePath不是结束路径，跟beginPath没有关系，他会试图把路径的结束点链接到开始点，让整个路径闭合，但是这不意味着之后就开始了新的路径

strokeRect和fillRect会自动闭合路径点

```
ctx.fill()
ctx.beginPath()
ctx.arc(400,400,80,0,2*Math.PI)
ctx.stroke()
ctx.fillStyle="#eaeaea";
ctx.fill()
```

绘制文本

```
ctx.beginPath();
ctx.fillStyle="green"//填充色
ctx.font="50px normal bold";//字体样式与css中一致
ctx.strokeStyle="red";//字体颜色
ctx.textAlign="left";//对齐方式
ctx.textBaseline="bottom"//基线  top  middle bottom
ctx.fillText('大吉大利',0,300);
ctx.strokeText('今晚上自习',0,400)
```

绘制图片

```
var ctx=$('#can').getContext('2d')
// ctx.drawImage(img,x,y,width,height)
    //宽高：可选
     // img:不是路径,是dom元素
    var img=document.createElement('img');
    img.src="./1.png";
    img.onload=function(){
         ctx.drawImage(img,100,100,300,300)
     }
```

绘制-变换

rotate（）旋转

scale（）缩放

translate（）重置原点

Transform() 允许缩放，旋转，移动，并倾斜当前环境

该变换只会影响设置transform之后的绘图

transform会相对于上一个变换，再次进行变换

水平缩放

水平倾斜

垂直倾斜

垂直缩放

水平移动

垂直移动

setTransform（）不会相对于其它变换来发生行为

绘制-渐变色

线性渐变

var lg=ctx.createLinearGradient(x,y,x1,y1)

lg.addColorStop(渐变位置,颜色) 渐变位置:number0-1;

ctx.fillStyle=lg

```
var lg=ctx.createLinearGradient(0,0,200,0);
lg.addColorStop(0,"red");
lg.addColorStop(0.5,"yellow");
lg.addColorStop(1,"green");
ctx.fillStyle=lg
ctx.fillRect(0,0,300,100)
```

径向渐变

var rg=ctx.createRadialGradient(x,y,半径,结束x,结束y,半径)

rg.addColorStop(渐变位置,颜色)

ctx.fillStyle=rg

```
var rg=ctx.createRadialGradient(150,100,5,150,100,300)
rg.addColorStop(0,'red');
rg.addColorStop(0.5,'yellow');
rg.addColorStop(1,"green");
ctx.fillStyle=rg
ctx.fillRect(0,0,300,200)
```

多图形组合

https://www.runoob.com/tags/canvas-globalcompositeoperation.html

ctx.globalCompositeOperation:这个属性要做的是如何将一个源(新的)图像,绘制到目标(已有的)图像上

源图像：我们要绘制的图形

目标图像：画布上已有的图形

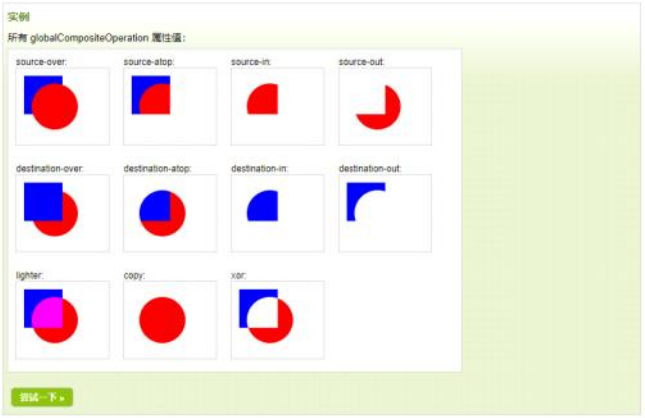
source-over:后画覆盖先画,

destination-out:后画清空先画

```
// destination-out:后画清空先画
ctx.globalCompositeOperation="destination-out"
ctx.fillRect(100,100,100,100)
//后画覆盖先画
ctx.globalCompositeOperation="source-over";
ctx.font="50px nomol"
ctx.strokeText('大吉大利',100,100)
```

**属性值**

| 值 | 描述 |
| --- | --- |
| source-over | 默认。在目标图像上显示源图像。 |
| source-atop | 在目标图像顶部显示源图像。源图像位于目标图像之外的部分是不可见的。 |
| source-in | 在目标图像中显示源图像。只有目标图像之内的源图像部分会显示，目标图像是透明的。 |
| source-out | 在目标图像之外显示源图像。只有目标图像之外的源图像部分会显示，目标图像是透明的。 |
| destination-over | 在源图像上显示目标图像。 |
| destination-atop | 在源图像顶部显示目标图像。目标图像位于源图像之外的部分是不可见的。 |
| destination-in | 在源图像中显示目标图像。只有源图像之内的目标图像部分会被显示，源图像是透明的。 |
| destination-out | 在源图像之外显示目标图像。只有源图像之外的目标图像部分会被显示，源图像是透明的。 |
| lighter | 显示源图像 + 目标图像。 |
| copy | 显示源图像。忽略目标图像。 |
| xor | 使用异或操作对源图像与目标图像进行组合。 |




## 保存画布

toDataURL（图片格式，品质）

- Image/png
- Image/jpeg
- Image/webp

如果图片格式为Image/jpeg或Image/webp，可以从0-1设置图片品质，默认0.92

返回值：base64格式

## 折线图：

1. 创建canvas对象
2. 重置原点
3. 封装画线方法
4. 画出坐标轴
5. 循环x轴
   a. 循环创建刻度
   b. 写标注
   c. 计算每个点的y轴
   d. 用线连起来
   e. 每个点上画一个圆，作为顶点
   f. 写标注
6. 循环y轴
   a. 循环创建刻度
   b. 写标注

## 柱状图：

1. 创建canvas对象
2. 重置原点
3. 封装画线方法
4. 封装画矩形方法
5. 画出坐标轴
6. 循环x轴
   a. 循环创建刻度
   b. 写标注
   c. 计算每个点的y轴
   d. 画矩形
   e. 写标注
7. 循环y轴
   a. 循环创建刻度
   b. 写标注