

移动端

一、 网页的常用布局方式：

(一) 响应式布局

概念：借助 CSS3 媒体查询技术，响应式设计的目标是确保一个页面在所有终端上（各种尺寸的 PC、手机、手表、冰箱的 Web 浏览器等等）都能显示出令人满意的效果，通常是结合了流式布局+弹性布局，再搭配媒体查询技术使用。——分别为不同的屏幕分辨率定义布局，同时，在每个布局中，应用流式布局的理念，即页面元素宽度随着窗口调整而自动适配。即：创建多个流体式布局，分别对应一个屏幕分辨率范围。可以把响应式布局看作是流式布局和自适应布局设计理念的融合。

媒介查询的使用：

媒介类型： screen（用于电脑屏幕，平板电脑，智能手机等）

媒体功能：

device-height：定义输出设备的屏幕可见高度

device-width：定义输出设备的屏幕可见宽度

height：定义输出设备中的页面可见区域高度

max-device-height：定义输出设备的屏幕可见的最大高度

max-device-width：定义输出设备的屏幕最大可见宽度

min-device-width：定义输出设备的屏幕最小可见宽度

max-height：定义输出设备中的页面最大可见区域高度

max-width：定义输出设备中的页面最大可见区域宽度

min-height：定义输出设备中的页面最小可见区域高度。

```
@media screen and (max-width: 1000px) {  
    body {  
        background-color: lightblue;  
    }  
}
```

```
<link rel="stylesheet" media="screen and (max-width: 1000px)" href="/test.css">
```

(二) 弹性布局

概念：使用 em 或者 rem 进行网页单位的书写，rem,em 都是顺应不同网页字体大小展现而产生的。其中，em 是相对其父元素，在实际应用中相对而言会带来很多不便；而 rem 是始终相对于 html 大小，即页面根元素。相比流式布局的%，这样的布局方式，会使得字体也可以随同屏幕宽度的变化，而改变；

移动端的 meta：

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">
```

width	设置 layout viewport 的宽度, 为一个正整数, 或字符串"width-device"
initial-scale	设置页面的初始缩放值, 为一个数字, 可以带小数
minimum-scale	允许用户的最小缩放值, 为一个数字, 可以带小数
maximum-scale	允许用户的最大缩放值, 为一个数字, 可以带小数
height	设置 layout viewport 的高度, 这个属性对我们并不重要, 很少使用
user-scalable	是否允许用户进行缩放, 值为"no"或"yes", no 代表不允许, yes 代表允许

devicePixelRatio:

获取: **window.devicePixelRatio**

机型	iPhone 3G/3GS	iPhone 4/4S	iPhone 5/5C/5S/SE	iPhone 6/6S	iPhone 6S Plus	iPhone 7	iPhone 7 Plus	iPhone 8	iPhone 8 Plus	iPhone X
屏幕尺寸	3.5 英寸	3.5 英寸	4 英寸	4.7 英寸	5.5 英寸	4.7 英寸	5.5 英寸	4.7 英寸	5.5 英寸	5.8 英寸
独立像素 (CSS 像素)	480*320	480*320	568*320	667*375	736*414	667*375	736*414	667*375	736*414	812*375
物理像素 (分辨率)	480*320	960*640	1136*640	1334*750	1920*1080(2208x1242)	1334*750	1920*1080	1334*750	1920*1080	2436*1125
ppi/dpi (像素密度)	163	326	326	326	401	326	401	326	401	458
dpr(倍数)	1	2	2	2	3(2.5)	3	3	3	3	3(2.9)

移动端 1px 细线解决方案总结:

1、

```
.border { border: 1px solid #999 }

@media screen and (-webkit-min-device-pixel-ratio: 2) {
    .border { border: 0.5px solid #999 }
}

@media screen and (-webkit-min-device-pixel-ratio: 3) {
    .border { border: 0.333333px solid #999 }
}
```

2、

`transform(scale(0.5));`

计算:

1、 使用百分比设置 rem:

浏览器的默认字体高度一般为 16px, 即 1em:16px, 但是 1:16 的比例不方便计算, 为了使单位 em/rem 更直观, CSS 编写者常常将页面跟节点字体设为 62.5%, 比如选择用 rem 控制字体时, 先需要设置根节点 html 的字体大小, 因为浏览器默认字体大小 16px*62.5%=10px。这样 1rem 便是 10px, 方便了计算。

2、 使用 js 进行 rem 的运算:

当前设备的视口宽度/设计图的宽度*100, 最后将计算出的单位值赋给根元素, 那么后期再开发过程中, 设计图上的 100px 的内容宽, 我们就可以设置为 1rem;

```
function Rem() {  
    if (document.documentElement.clientWidth >= 750) {  
  
        document.documentElement.style.width = 750 + "px";  
  
    } else if (document.documentElement.clientWidth <= 320) {  
        document.documentElement.style.width = 320 + "px";  
  
    } else {  
        var rem = document.documentElement.clientWidth / 7.5;  
        document.documentElement.style.fontSize = rem + 'px';  
    }  
}  
Rem()  
window.onresize = function() {  
  
    Rem();  
};
```

3、 使用 flexible

优点: 理想状态是所有屏幕的高宽比和最初的设计高宽比一样, 或者相差不多, 完美适应。

缺点: 这种 rem+js 只不过是宽度自适应, 高度没有做到自适应, 一些对高度, 或者元素间距要求比较高的设计, 则这种布局没有太大的意义。如果只是宽度自适应, 更推荐响应式设计。

Demo:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport"  
content="width=device-width,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no" />  
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

```

<title>Document</title>
<script src="../libs/flexible.js"></script>
<style>
    * {
        margin: 0;
        padding: 0;
    }

    .titleTest {
        font-size: calc(100rem/75);
        width: calc(750rem/75);
        background: linen
    }

    [data-dpr="2"] div {
        font-size: 16px;
    }

    [data-dpr="3"] div {
        font-size: 24px;
    }

    [data-dpr="2"] p {
        width: calc(750rem/75);
        height: calc(500rem/75);
        background: url("../img/@2pic.jpg") no-repeat;
        background-size: calc(750rem/75) calc(500rem/75)
    }

    [data-dpr="3"] p {
        width: calc(750rem/75);
        height: calc(500rem/75);
        background: url("../img/@3pic.jpg");
        background-size: calc(750rem/75) calc(500rem/75)
    }
</style>
</head>

<body>
    <div class="titleTest">天气很好</div>
    <p></p>
</body>

</html>

```

4、 弹性布局之 vh, vw 单位使用, 结合 rem 实现移动端布局

视区宽度是 100vm, 则 1vm 是视区宽度的 1/100, 也就是 1%, 类似于 width: 1%;

注：这里视口是指的网页视区宽度，指为浏览器内部的可视区域大小，即 `window.innerWidth/window.innerHeight` 大小

应用实例：

弹出层，根据屏幕的大小进行设置值（js 实现的弊端）

(三) 静态布局

即传统 Web 设计，网页上的所有元素的尺寸一律使用 `px` 作为单位。

1、布局特点：不管浏览器尺寸具体是多少，网页布局始终按照最初写代码时的布局来显示。常规的 pc 的网站都是静态（定宽度）布局的，也就是设置了 `min-width`，这样的话，如果小于这个宽度就会出现滚动条，如果大于这个宽度则内容居中外加背景，这种设计常见与 pc 端。

2、设计方法：

PC：居中布局，所有样式使用绝对宽度/高度(px)，设计一个 Layout，在屏幕宽高有调整时，使用横向和竖向的滚动条来查阅被遮掩部分；

移动设备：另外建立移动网站，单独设计一个布局，使用不同的域名如 `wap.`或 `m.`。

● 自适应布局

自适应布局的特点是分别为不同的屏幕分辨率定义布局，即创建多个静态布局，每个静态布局对应一个屏幕分辨率范围。改变屏幕分辨率可以切换不同的静态局部（页面元素位置发生改变），但在每个静态布局中，页面元素不随窗口大小的调整发生变化。可以把自适应布局看作是静态布局的一个系列。

1、布局特点：屏幕分辨率变化时，页面里面元素的位置会变化而大小不会变化。

2、设计方法：使用 `@media` 媒体查询给不同尺寸和介质的设备切换不同的样式。在优秀的响应范围设计下可以给适配范围内的设备最好的体验，在同一个设备下实际还是固定的布局

(四) 流式布局

概念：流式布局就是百分比布局，通过盒子的宽度设置成百分比来根据屏幕的宽度来进行伸缩，不受固定像素的限制，内容向两侧填充，同时会设定最小宽度和最大宽度，适用于图片比较多的首页、门户、电商等。

特点：

屏幕分辨率变化时，页面里元素的大小会变化而但布局不变。【这就导致如果屏幕太大或者太小都会导致元素无法正常显示】

使用%百分比定义宽度，高度大都是用 `px` 来固定住，可以根据可视区域（viewport）和父元素的实时尺寸进行调整，尽可能的适应各种分辨率。往往配合 `max-width/min-width` 等属性控制尺寸流动范围以免过大或者过小影响阅读

优点：

这种布局方式在 Web 前端开发的早期历史上，用来应对不同尺寸的 PC 屏幕（那时屏幕尺寸的差异不会太大），在当今的移动端开发也是常用布局方式

缺点：主要的问题是如果屏幕尺度跨度太大，那么在相对其原始设计而言过小或过大的屏幕上不能正常显示。因为宽度使用%百分比定义，但是高度和文字大小等大都是用 `px` 来固定，所以在大屏幕的手机下显示效果会变成有些页面元素宽度被拉的很长，但是高度、文字大小还是和原来一样（即，这些东西无法变得“流式”），显示非常不协调

(五) 栅格布局

栅格系统（CSS Grids）是一种运用固定的格子设计版面布局，在报刊杂志上尤为常见。

如今响应式设计大行其道，对于前端开发，栅格系统可以：

提高生产力，通过在网格的划分，元素更容易堆放而且在跨浏览器上面具有一致性，使我们可以专心的注意布局而不是兼容上。

具有灵活性，无论是什么样的布局，都可以拆分到粒度为一个网格的大小。

支持响应式设计，栅格系统本身能很好的和响应式设计结合在一起，或者说，我们的栅格系统是基于响应式设计的。

栅格系统的组成：

container: 包裹整个栅格系统的容器

rows: 行

columns: 列

gutters: 各列的间的空隙

(六) 弹性盒布局

Flex 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。

任何一个容器都可以指定为 Flex 布局。

A. 容器的属性

a) flex-direction 属性

flex-direction 属性决定主轴的方向（即项目的排列方向）。

```
.box {  
  flex-direction: row | row-reverse | column | column-reverse;  
}
```

它可能有 4 个值。

row（默认值）：主轴为水平方向，起点在左端。

row-reverse：主轴为水平方向，起点在右端。

column：主轴为垂直方向，起点在上沿。

column-reverse：主轴为垂直方向，起点在下沿。

b) flex-wrap 属性

默认情况下，项目都排在一条线（又称“轴线”）上。flex-wrap 属性定义，如果一条轴线排不下，如何换行。

```
.box {  
  flex-wrap: nowrap | wrap | wrap-reverse;  
}
```

它可能取三个值。

(1) nowrap（默认）：不换行。

(2) wrap：换行，第一行在上方。

(3) wrap-reverse：换行，第一行在下方。

c) flex-flow

flex-flow 属性是 flex-direction 属性和 flex-wrap 属性的简写形式，默认值为 row nowrap。

```
.box {  
  flex-flow: <flex-direction> || <flex-wrap>;  
}
```

d) justify-content 属性

justify-content 属性定义了项目在主轴上的对齐方式。

```
.box {  
  justify-content: flex-start | flex-end | center | space-between | space-around;  
}
```

它可能取 5 个值，具体对齐方式与轴的方向有关。下面假设主轴为从左到右。

flex-start (默认值)：左对齐

flex-end：右对齐

center：居中

space-between：两端对齐，项目之间的间隔都相等。

space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

e) align-items 属性

align-items 属性定义项目在交叉轴上如何对齐。

```
.box {  
  align-items: flex-start | flex-end | center | baseline | stretch;  
}
```

它可能取 5 个值。具体的对齐方式与交叉轴的方向有关，下面假设交叉轴从上到下。

flex-start：交叉轴的起点对齐。

flex-end：交叉轴的终点对齐。

center：交叉轴的中点对齐。

baseline：项目的第一行文字的基线对齐。

stretch (默认值)：如果项目未设置高度或设为 auto，将占满整个容器的高度。

f) align-content 属性

align-content 属性定义多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

```
.box {  
  align-content: flex-start | flex-end | center | space-between | space-around | stretch;  
}
```

该属性可能取 6 个值。

flex-start：与交叉轴的起点对齐。

flex-end：与交叉轴的终点对齐。

center：与交叉轴的中点对齐。

space-between：与交叉轴两端对齐，轴线之间的间隔平均分布。

space-around：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。

stretch (默认值)：轴线占满整个交叉轴。

B. 项目的属性

a) order 属性

order 属性定义项目的排列顺序。数值越小，排列越靠前，默认为 0。

```
.item {  
  order: <integer>;  
}
```

b) flex-grow 属性

flex-grow 属性定义项目的放大比例，默认为 0，即如果存在剩余空间，也不放大。

```
.item {  
  flex-grow: <number>; /* default 0 */  
}
```

如果所有项目的 flex-grow 属性都为 1，则它们将等分剩余空间（如果有的话）。如果一个项目的 flex-grow 属性为 2，其他项目都为 1，则前者占据的剩余空间将比其他项多一倍。

c) flex-shrink 属性

flex-shrink 属性定义了项目的缩小比例，默认为 1，即如果空间不足，该项目将缩小。

```
.item {  
  flex-shrink: <number>; /* default 1 */  
}
```

如果所有项目的 flex-shrink 属性都为 1，当空间不足时，都将等比例缩小。如果一个项目的 flex-shrink 属性为 0，其他项目都为 1，则空间不足时，前者不缩小。

负值对该属性无效。

d) flex-basis 属性

flex-basis 属性定义了再分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为 auto，即项目的本来大小。

```
.item {  
  flex-basis: <length> | auto; /* default auto */  
}
```

它可以设为跟 width 或 height 属性一样的值（比如 350px），则项目将占据固定空间。

e) flex 属性

flex 属性是 flex-grow, flex-shrink 和 flex-basis 的简写，默认值为 0 1 auto。后两个属性可选。

```
.item {  
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]  
}
```

该属性有两个快捷值：auto (1 1 auto) 和 none (0 0 auto)。

建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值。

f) align-self 属性

align-self 属性允许单个项目有与其他项目不一样的对齐方式，可覆盖 align-items 属性。默认值为 auto，表示继承父元素的 align-items 属性，如果没有父元素，则等同于 stretch。

```
.item {  
  align-self: auto | flex-start | flex-end | center | baseline | stretch;
```


}

扩展：

<https://www.cnblogs.com/cench/p/5314044.html>

移动端，多屏幕尺寸高清屏 retina 屏适配的解决方

物理像素(physical pixel)

设备独立像素（实际像素）

二、 字体图标

(一) 为什么使用字体图标？

1.可以通过 css 的样式改变其颜色(最霸气的理由)

2.相对于图片来说,具有更高的分辨率

3.更小的存储

缺点:浏览器兼容性不够普及,所幸目前大部分主流浏览器都支持

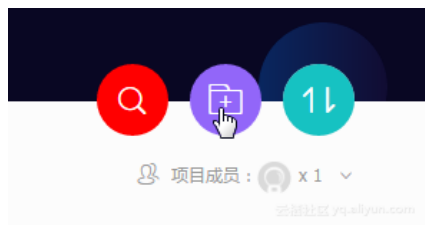
(二) Iconfont 图标（阿里巴巴 iconfont）下载

1.登录:<http://www.iconfont.cn/>

2.找到图标管理->我的项目->然后新建项目



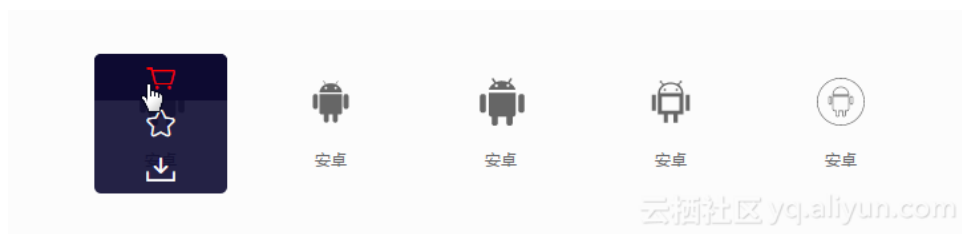
点击新建项目,用于保存自己常用的图标



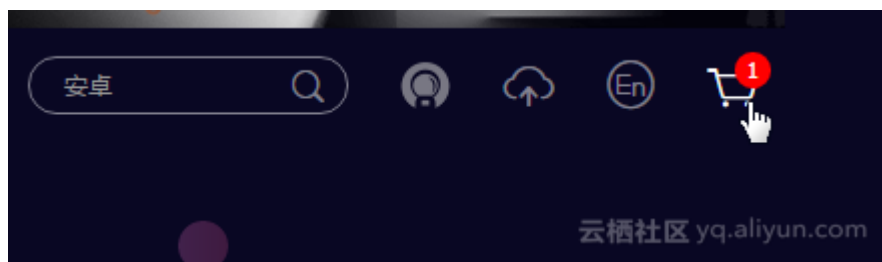
3.项目新建完成后,往项目里添加我们想使用的图标



找到图标库,搜索一个想要的图标,然后添加到购物车



4. 添加到购物车完成后,购物车徽章数字应该显示 1 了,点击右上角的购物车图标,选择添加至项目,选择我们刚刚创建的项目,确定



自动跳转到对应的项目里了



5. 接下来一部比较关键,将打包好的字体文件下载到本地添加到你的项目中,在项目中引用文件中的 iconfont.css 文件



下载后解压文件

demo.css	2017/3/29 15:16	层叠样式表文档	7 KB
demo_fontclass.html	2017/3/29 15:16	Chrome HTML D...	3 KB
demo_symbol.html	2017/3/29 15:16	Chrome HTML D...	4 KB
demo_unicode.html	2017/3/29 15:16	Chrome HTML D...	4 KB
iconfont.css	2017/3/29 15:16	层叠样式表文档	1 KB
iconfont.eot	2017/3/29 15:16	EOT 文件	6 KB
iconfont.js	2017/3/29 15:16	JScript Script 文件	5 KB
iconfont.svg	2017/3/29 15:16	SVG 文件	3 KB
iconfont.ttf	2017/3/29 15:16	TrueType 字体文件	6 KB
iconfont.woff	2017/3/29 15:16	WOFF 文件	4 KB

(三) Iconfont 图标 (阿里巴巴 iconfont) 应用到代码

1. 强调: 把上面下载的文件都放在一个文件夹内, 然后放在项目目录中, 再在项目中引入 iconfont.css 文件, 或将 iconfont.css 文件里的内容复制粘贴到你当前项目的 css 内容里



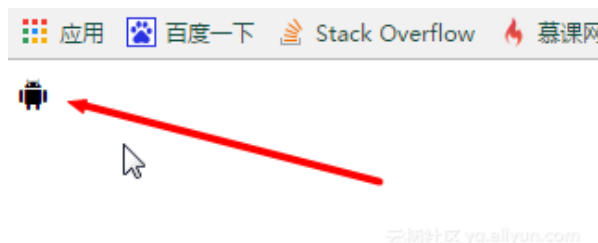
2. 如何在项目中使用字体图标呢, 其实很简单, 创建一个 i 标签或 span 标签, 添加两个类名, 一个固定的是 iconfont, 另一个是你想要的那个图标对应的类名



具体代码如下:



现在刷新页面,图标就出来了



三、 Css3 动画

(一)过渡(transition)

CSS3 的过渡功能就像是一种黄油,可以让 CSS 的一些变化变得平滑。因为原生的 CSS 过渡在客户端需要处理的资源要比用 JavaScript 和 Flash 少的多,所以才会更平滑。

过渡相关属性:

transition-property:要过渡的 CSS 属性名称(比如 background-color、 text-shadow 或者 all, 使用 all 则过渡会被应用到每一个可能的 CSS 属性上);

transition-duration:定义过渡效果持续的时间(时间单位为秒, 比如.3s、2s 或 1.5s);

transition-timing-function:定义过渡期间速度如何变化(比如 ease、linear、 ease-in、ease-out、ease-in-out 或 cubic-bezier);

值	描述
linear	规定以相同速度开始至结束的过渡效果 (等于 cubic-bezier(0,0,1,1))。
ease	规定慢速开始, 然后变快, 然后慢速结束的过渡效果 (cubic-bezier(0.25,0.1,0.25,1))。
ease-in	规定以慢速开始的过渡效果 (等于 cubic-bezier(0.42,0,1,1))。
ease-out	规定以慢速结束的过渡效果 (等于 cubic-bezier(0,0,0.58,1))。
ease-in-out	规定以慢速开始和结束的过渡效果 (等于 cubic-bezier(0.42,0,0.58,1))。
cubic-bezier(<i>n,n,n,n</i>)	在 cubic-bezier 函数中定义自己的值。可能的值是 0 至 1 之间的数值。

transition-delay:可选, 用于定义过渡开始前的延迟时间。相反, 将该值设置为一个负数, 可以让过渡效果立即开始, 但过渡旅程则会从半路开始。

特点:

- (1) transition 需要事件触发, 所以没法在网页加载时自动发生。
- (2) transition 是一次性的, 不能重复发生, 除非一再触发。
- (3) transition 只能定义开始状态和结束状态, 不能定义中间状态, 也就是说只有两个状态。
- (4) 一条 transition 规则, 只能定义一个属性的变化, 不能涉及多个属性。

Css:

```
.testcss{
    color:red;
    transition: all 3s ease 0s;
```

```
}  
.testcss:hover{  
    color:yellow;  
}
```

(二)变形(transform)

在 CSS3 中，可以利用 transform 功能来实现文字或图像的旋转、缩放、倾斜、移动这四种类型的变形处理；

旋转 rotate

用法：transform: rotate(45deg);

共一个参数“角度”，单位 deg 为度的意思，正数为顺时针旋转，负数为逆时针旋转，上述代码作用是顺时针旋转 45 度。

缩放 scale

用法：transform: scale(0.5) 或者 transform: scale(0.5, 2);

参数表示缩放倍数；

一个参数时：表示水平和垂直同时缩放该倍率

两个参数时：第一个参数指定水平方向的缩放倍率，第二个参数指定垂直方向的缩放倍率。

倾斜 skew

用法：transform: skew(30deg) 或者 transform: skew(30deg, 30deg);

参数表示倾斜角度，单位 deg

一个参数时：表示水平方向的倾斜角度；

两个参数时：第一个参数表示水平方向的倾斜角度，第二个参数表示垂直方向的倾斜角度。

关于 skew 倾斜角度的计算方式表面上看并不是那么直观

移动 translate

用法：transform: translate(45px) 或者 transform: skew(45px, 150px);

参数表示移动距离，单位 px，

一个参数时：表示水平方向的移动距离；

两个参数时：第一个参数表示水平方向的移动距离，第二个参数表示垂直方向的移动距离。

基准点 transform-origin

在使用 transform 方法进行文字或图像的变形时，是以元素的中心点为基准点进行的。使用 transform-origin 属性，可以改变变形的基准点。

用法：transform-origin: 10px 10px;

共两个参数，表示相对左上角原点的距离，单位 px，第一个参数表示相对左上角原点水平方向的距离，第二个参数表示相对左上角原点垂直方向的距离；

两个参数除了可以设置为具体的像素值，其中第一个参数可以指定为 left、center、right，第二个参数可以指定为 top、center、bottom。

多方法组合变形

上面我们介绍了使用 transform 对元素进行旋转、缩放、倾斜、移动的方法，这里讲介绍综合使用这几个方法来对一个元素进行多重变形。

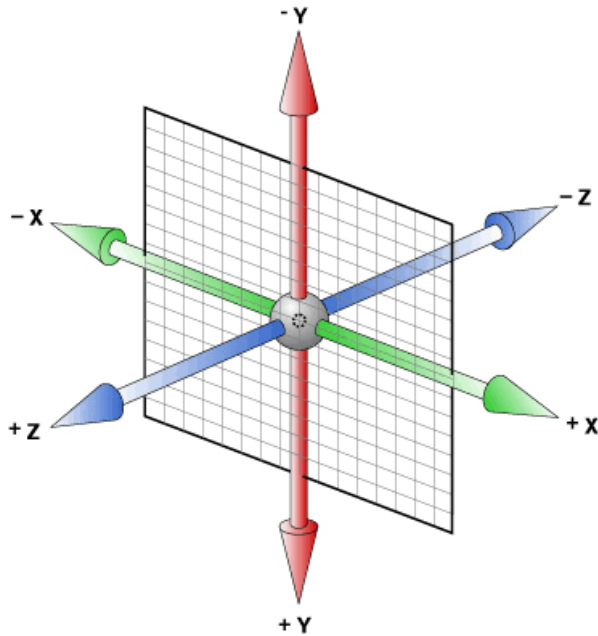
用法：transform: rotate(45deg) scale(0.5) skew(30deg, 30deg) translate(100px, 100px);

这四种变形方法顺序可以随意，但不同的顺序导致变形结果不同，原因是变形的顺序是从左到右依次进行，

这个用法中的执行顺序为 1.rotate 2.scale 3.skew 4.translate

(三)Transform (3d)

1. 熟悉 x 轴, y 轴, z 轴的概念



2. 相比 transform2d 的基础增加如下几个属性:

- transform-style: preserve-3d; 3d 空间
- perspective: 800px; 它被成为视距或者紧身. (眼睛距离物体的距离)
- perspective-origin 属性定义 3D 元素所基于的 X 轴和 Y 轴. 该属性允许您改变 3D 元素的底部位置, 眼睛的位置

3. 深化的 transform2d 属性

rotateX/rotateY/rotateZ 可以帮助理解三维坐标

translateZ 则可以帮你理解透视位置。

transform-origin 我们成为基点 在水平方向改变观看 div 的位置

scale 缩放 rotate 旋转 translate 移动 skew 倾斜 通过这些来进行 3d 效果

(四)动画(animation)

A. 关键帧:

关键帧(keyframes) - 定义动画在不同阶段的状态。

动画属性(properties) - 决定动画的播放时长, 播放次数, 以及用何种函数式去播放动画等。(可以类比音视

频播放器)

css 属性 - 就是 css 元素不同关键帧下的状态。

如：

```
@keyframes dropdown {
  0% { margin-top: 0px;}
  /** 暂停效果 */
  10% { margin-top: 0px;}
  50% { margin-top: -100px;}
  60% { margin-top: -100px;}
  90% { margin-top: -200px;}
  100% { margin-top: -200px;}
}
```

B. 动画属性

动画属性可以理解为播放器的相关功能，一个最基本的播放器应该具有：播放/暂停、播放时长、播放顺序(逆序/正序/交替播放)、循环次数等。

主要也分为两大点：

指定播放的元素

定义播放信息的配置

相应的配置有：

animation-name 规定需要绑定到选择器的 keyframe 名称

animation-duration 规定完成动画所花费的时间，以秒或毫秒计。

animation-timing-function 规定动画的速度曲线。如：

linear 动画从头到尾的速度是相同的。

ease 默认。动画以低速开始，然后加快，在结束前变慢。

ease-in 动画以低速开始。

ease-out 动画以低速结束。

ease-in-out 动画以低速开始和结束。

animation-delay 规定在动画开始之前的延迟。

animation-iteration-count 规定动画应该播放的次数。

animation-direction 规定是否应该轮流反向播放动画。

normal 默认值。动画应该正常播放。

alternate 动画应该轮流反向播放。

四、浏览器内核和前缀

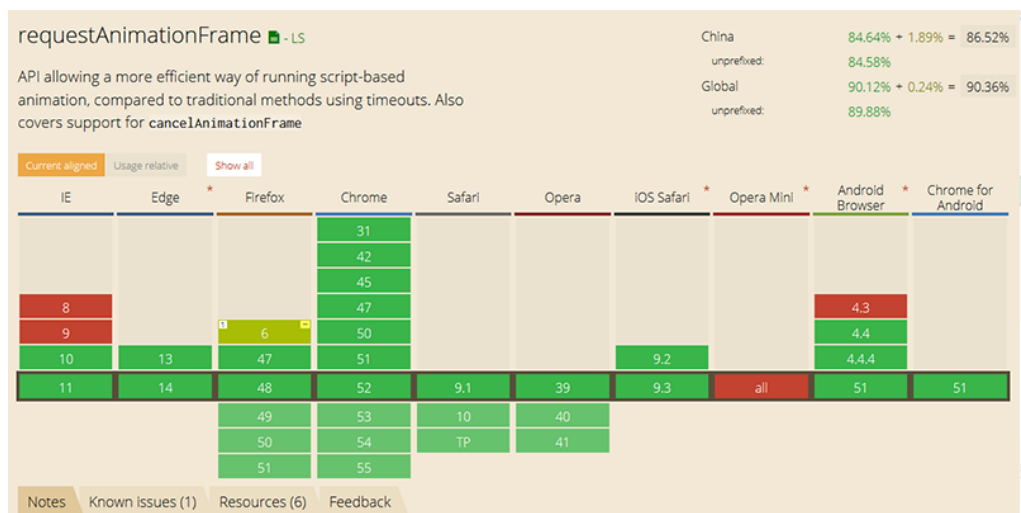
Chrome(谷歌浏览器)：	WebKit 内核	-webkit-
Safari(苹果浏览器)：	WebKit 内核	-webkit-
Firefox(火狐浏览器)：	Gecko 内核	-moz-
IE(IE 浏览器)：	Trident 内核	-ms-
Opera(欧朋浏览器)：	Presto 内核	-o-

五、Js 的异步运行机制：

详见文档：js 运行机制

六、 Window.requestAnimationFrame()(异步事件)

支持度：



`window.requestAnimationFrame()`这个 API 是浏览器提供的 js 全局方法，针对动画效果。

使用

用法 1:

```
function animate() {  
  //done();  
  requestAnimationFrame(animate);  
}  
requestAnimationFrame(animate);
```

注意函数里的 `requestAnimationFrame(animate)`

有了这句话，就形成了递归调用，设置应为此函数多用在持续的动画中，可以自由处理要不要这句话。

用法 2:

```
var globalID;  
function animate() {  
  // done(); 一直运行  
  globalID=requestAnimationFrame(animate);  
  // Do something animate  
}  
globalID=requestAnimationFrame(animate);//开始  
cancelAnimationFrame(globalID);//结束
```

setInterval 的缺点:

setInterval() 方法, 不考虑浏览器中发生的事情, 如果你正在浏览其他页面, 这个函数仍然会每隔几毫秒就会被调用一次, 除此之外, setInterval() 方法并没有跟显示器的重画同步, 这可能会导致较高的 CPU 使用, 降低系统效率。

requestAnimationFrame 优点:

1. requestAnimationFrame 会把每一帧中的所有 DOM 操作集中起来, 在一次重绘或回流中就完成, 并且重绘或回流的时间间隔紧紧跟随浏览器的刷新频率, 一般来说, 这个频率为每秒 60 帧。
2. 在隐藏或不可见的元素中, requestAnimationFrame 将不会进行重绘或回流, 这当然就意味着更少的 CPU, GPU 和内存使用量。
3. 窗口没激活时, 动画将停止, 省计算资源;

使用场景:

可以调节重新渲染, 大幅提高网页性能。其中最重要的, 它可以将某些代码放到下一次重新渲染时执行。避免短时间内触发大量 reflow。

```
function doubleHeight(element) {  
    var currentHeight = element.clientHeight;  
    window.requestAnimationFrame(function () {  
        element.style.height = (currentHeight * 2) + 'px';  
    });  
}  
elements.forEach(doubleHeight);
```

页面滚动事件 (scroll) 的监听函数, 就很适合用这个 api, 推迟到下一次重新渲染。

```
$(window).on('scroll', function() {  
    window.requestAnimationFrame(scrollHandler);  
});
```

最佳的应用场景还是在帧动画里, 可以大幅优化性能;

七、 移动端事件

A. pc 端事件在移动端的问题

移动设备主要特点是不配备鼠标, 键盘也只是在需要输入的地方才会激活虚拟键盘。所以以前的 pc 端事件在移动端使用起来就没有以前那么爽了, 虽然部分仍然可以使用。

click 事件的 300ms 延迟问题。

2007 年第一代 iPhone 发布, 由于那个年代所有的网页都是针对大屏的 PC 端设计的, iPhone 的 Safari 浏览器为了让用户浏览网页的时候可以浏览到整个网页, 把 viewport 设置为 960px (参考前面的文章), 好是好, 但是由于缩放了整个页面, 导致内容变得非常小, 视力 6.0 的都不一定看得清楚。

所以 Safari 浏览器自带了一个当时看起来相当酷的一个功能: 双击缩放。你双击页面的时候, 浏览器会智能的缩放当前页面到原始大小。

双击缩放的原理就是，当你 click 一次之后，会经过 300ms 之后检测是否再有一次 click，如果有的话，就会缩放页面。否则的话就是一个 click 事件。

所以，当你想执行 click 操作的时候，就感觉到了”卡顿”。如果点击之后 100ms 之后没有反应，基本就有卡顿的感觉。

dblclick 事件失效

由于双击缩放的存在，pc 端的 dblclick 事件也失效了。

B. 移动端 web 新增 touch 事件

随着触屏设备的普及，w3c 为移动端 web 新增了 touch 事件。

最基本的 touch 事件包括 4 个事件：

touchstart

当在屏幕上按下手指时触发

touchmove

当在屏幕上移动手指时触发

touchend

当在屏幕上抬起手指时触发

touchcancel

当一些更高级别的事件发生的时候(如电话接入或者弹出信息)会取消当前的 touch 操作,即触发 touchcancel。一般会在 touchcancel 时暂停游戏、存档等操作。

C. TouchEvent 详解

为了区别触摸相关的状态改变，存在多种类型的触摸事件。可以通过检查触摸事件的 TouchEvent.type 属性来确定当前事件属于哪种类型。

注意:在很多情况下，触摸事件和鼠标事件会同时被触发（目的是让没有对触摸设备优化的代码仍然可以在触摸设备上正常工作）。如果你使用了触摸事件，可以调用 event.preventDefault()来阻止鼠标事件被触发。

1. Touchstart

当用户手指触摸到的触摸屏的时候触发。事件对象的 target 就是 touch 发生位置的那个元素。

2. touchmove

当用户在触摸屏上移动触点(手指)的时候，触发这个事件。一定是先要触发 touchstart 事件，再有可能触发 touchmove 事件。

touchmove 事件的 target 与最先触发的 touchstart 的 target 保持一致。touchmove 事件和鼠标的 mousemove 事件一样都会多次重复调用，所以，事件处理时不能有太多耗时操作。不同的设备，移动同样的距离 touchmove 事件的触发频率是不同的。

有一点需要注意：即使手指移出了 原来的 target 元素，则 touchmove 仍然会被一直触发，而且 target 仍然是原来的 target 元素。

3. touchend

当用户的手指抬起的时候，会触发 touchend 事件。当用户的手指从触屏设备的边缘移出了触屏设备，也会触发 touchend 事件。

touchend 事件的 target 也是与 touchstart 的 target 一致，即使已经移出了元素。

4. touchcancel

当触点由于某些原因被中断时触发。有几种可能的原因如下(具体的原因根据不同的设备和浏览器有所不同)：

由于某个事件取消了触摸：例如触摸过程被一个模态的弹出框打断。

触点离开了文档窗口，而进入了浏览器的界面元素、插件或者其他外部内容区域。
当用户产生的触点个数超过了设备支持的个数，从而导致 TouchList 中最早的 Touch 对象被取消

D. Touch 详解

表示用户和触摸设备之间接触时单独的交互点(a single point of contact)。这个交互点通常是一个手指或者触摸笔。触摸设备通常是触摸屏或者触摸板。js 对象提供了多个属性来描述 Touch 对象。

基本属性: (只读属性)

identifier:

表示每 1 个 Touch 对象 的独一无二的 identifier。有了这个 identifier 可以确保你总能追踪到这个 Touch 对象。

screenX:

触摸点相对于屏幕左边缘的 x 坐标。

screenY:

触摸点相对于屏幕上边缘的 y 坐标。

clientX:

触摸点相对于浏览器的 viewport 左边缘的 x 坐标。不会包括左边的滚动距离。

clientY:

触摸点相对于浏览器的 viewport 上边缘的 y 坐标。不会包括上边的滚动距离。

pageX:

触摸点相对于 document 的左边缘的 x 坐标。与 clientX 不同的是，他包括左边滚动的距离，如果有的话。

pageY:

触摸点相对于 document 的左边缘的 y 坐标。与 clientY 不同的是，他包括上边滚动的距离，如果有的话。

target:

总是表示 手指最开始放在触摸设备上的触发点所在位置的 element。即使已经移出了元素甚至移出了 document, 他表示的 element 仍然不变

E. 封装移动端事件

前面的是最基本的事件，直接使用相对比较麻烦，所以有必要对一些常用事件进行封装。

比如：单击事件、双击事件、滑动方向等

封装:

```
(function (window){ //传入 window, 提高变量的查找效率
    function myQuery(selector){ //这个函数就是对外提供的接口。
        //调用这个函数的原型对象上的_init 方法, 并返回
        return myQuery.prototype._init(selector);
    }
    myQuery.prototype = {
        /*初始化方法, 获取当前 query 对象的方法*/
        _init: function (selector){
            if (typeof selector == "string"){
                //把查找到的元素存入到这个原型对象上。
                this.ele = window.document.querySelector(selector);
                //返回值其实就是原型对象。
                return this;
            }
        }
    },
},
```

```

/*单击事件:
 * 为了规避 click 的 300ms 的延迟, 自定义一个单击事件
 * 触发时间:
 *   当抬起手指的时候触发
 *   需要判断手指落下和手指抬起的事件间隔, 如果小于 500ms 表示单击时间。
 *
 *   如果是大于等于 500ms, 算是长按时间
 * */
tap: function (handler){
    this.ele.addEventListener("touchstart", touchFn);
    this.ele.addEventListener("touchend", touchFn);

    var startTime,
        endTime;

    function touchFn(e){
        e.preventDefault()
        switch (e.type){
            case "touchstart":
                startTime = new Date().getTime();
                break;
            case "touchend":
                endTime = new Date().getTime();
                if (endTime - startTime < 500){
                    handler.call(this, e);
                }
                break;
        }
    }
},
/**
 * 长按
 * @param handler
 */
longTap: function (handler){
    this.ele.addEventListener("touchstart", touchFn);
    this.ele.addEventListener("touchmove", touchFn);
    this.ele.addEventListener("touchend", touchFn);
    var timerId;

    function touchFn(e){
        switch (e.type){
            case "touchstart" : //500ms 之后执行
                timerId = setTimeout(function (){
                    handler.call(this, e);
                }, 500)
                break;
            case "touchmove" :
                //如果中间有移动也清除定时器

```

```

        clearTimeout(timerId)
        break;
    case "touchend" :
        //如果在 500ms 之内抬起了手指, 则需要定时器
        clearTimeout(timerId);
        break;
    }
}
},
/**
 * 左侧滑动。
 * 记录手指按下的左边, 在离开的时候计算 deltaX 是否满足左滑的条件
 */
slideLeft: function (handler){
    this.ele.addEventListener("touchstart", touchFn);
    this.ele.addEventListener("touchend", touchFn);
    var startX, startY, endX, endY;

    function touchFn(e){
        e.preventDefault();
        var firstTouch = e.changedTouches[0];
        switch (e.type){
            case "touchstart":
                startX = firstTouch.pageX;
                startY = firstTouch.pageY;
                break;
            case "touchend":
                endX = firstTouch.pageX;
                endY = firstTouch.pageY;
                //x 方向移动大于 y 方向的移动, 并且 x 方向的移动大于 25 个像素, 表示在向左侧滑动
                if (Math.abs(endX - startX) >= Math.abs(endY - startY) && startX
- endX >= 25){
                    handler.call(this, e);
                }
                break;
            }
        }
    },
    /**
     * 右侧滑动。
     */
    rightLeft: function (e){
        //TODO:
    }
}
window.$ = window.myQuery = myQuery;
})(window);

```

使用：

```
$("#div").tap(function (e){
    console.log("单击事件")
})

$("#div").longTap(function (){
    console.log("长按事件");
})

$("#div").slideLeft(function (e){
    console.log(this);
    this.innerHTML = "左侧滑动了....."
})
```

八、 Html5 新增 Api

(一)前端存储

客户端在没有数据库的情况，做为记录状态的方式之一，可以为当前平台，当前网站进行客户端数据的保存，以及多页面间的数据交互通信；是浏览器的一种状态记录方式；

● Cookie

Cookie：存在于 http 请求的头部，始终在客户端和服务端通讯的过程，进行携带；具有一定性能的浪费；当网页要发 http 请求时，浏览器会先检查是否有相应的 cookie，有则自动添加在 request header 中的 cookie 字段中。这些是浏览器自动帮我们做的，而且每一次 http 请求浏览器都会自动帮我们做。

特征

1. cookie 的存储是以域名形式进行区分的，不同的域下存储的 cookie 是独立的。
2. 我们可以设置 cookie 生效的域(当前设置 cookie 所在域的子域),也就是说,我们能够操作的 cookie 是当前域以及当前域下的所有子域
3. 一个域名下存放的 cookie 的个数是有限制的，不同的浏览器存放的个数不一样,一般为 20 个。
4. 每个 cookie 存放的内容大小也是有限制的，不同的浏览器存放大小不一样，一般为 4KB。
5. cookie 也可以设置过期的时间，默认是会话结束的时候，当时间到期自动销毁
- 6.

设置

1、 客户端设置

```
document.cookie = '名字=值';
```

```
document.cookie = 'username=cfangxu;domain=baike.baidu.com'//并且设置了生效域
```

注意：客户端可以设置 cookie 的下列选项：expires、domain、path、secure（有条件：只有在 https 协议的网页中，客户端设置 secure 类型的 cookie 才能成功），但无法设置 HttpOnly 选项。

2、 服务器端设置

不管你是请求一个资源文件 (如 html/js/css/图片), 还是发送一个 ajax 请求, 服务端都会返回 response。而 response header 中有一项叫 set-cookie, 是服务端专门用来设置 cookie 的。

Set-Cookie 消息头是一个字符串, 其格式如下 (中括号中的部分是可选的):

Set-Cookie: value[; expires=date][; domain=domain][; path=path][; secure]

注意: 一个 set-Cookie 字段只能设置一个 cookie, 当你要想设置多个 cookie, 需要添加同样多的 set-Cookie 字段。

服务端可以设置 cookie 的所有选项: expires、domain、path、secure、HttpOnly

通过 Set-Cookie 指定的这些可选项只会在浏览器端使用, 而不会被发送至服务器端。

读取

document.cookie: 例如 username=chenfangxu;job=coding

修改 cookie

要想修改一个 cookie, 只需要重新赋值就行, 旧的值会被新的值覆盖。但要注意一点, 在设置新 cookie 时, path/domain 这几个选项一定要旧 cookie 保持一致。否则不会修改旧值, 而是添加了一个新的 cookie。

删除

把要删除的 cookie 的过期时间设置成已过去的时间,path/domain/这几个选项一定要旧 cookie 保持一致。

cookie 的属性 (可选项)

1. 过期时间

一个设置 cookie 时效性的例子:

```
function setCookie(c_name, value, expiredays){
    var exdate=new Date();
    exdate.setDate(exdate.getDate() + expiredays);
    document.cookie=c_name+ "=" + escape(value) + ((expiredays==null) ? "" : ";expires="+exdate.toGMTString())
}
```

使用方法;

```
setcookie('username','zxy',30);
```

注:

expires 是 http/1.0 协议中的选项

max-age: 1.1 中的 max-age 用秒来设置 cookie 的生存期,

0;//不记录 cookie

-1;//会话级 cookie, 关闭浏览器失效

3600;//过期时间为 1 小时 (从文档第一次加载开始计算时间)

2. path 属性

它指定与 cookie 关联在一起的网页。在默认的情况下 cookie 会与创建它的网页，该网页处于同一目录下的网页以及与这个网页所在目录下的子目录下的网页关联。

3. domain 属性

domain 属性可以使多个 web 服务器共享 cookie。domain 属性的默认值是创建 cookie 的网页所在服务器的主机名。不能将一个 cookie 的域设置成服务器所在的域之外的域。

例如让位于 order.example.com 的服务器能够读取 catalog.example.com 设置的 cookie 值。如果 catalog.example.com 的页面创建的 cookie 把自己的 path 属性设置为“/”，把 domain 属性设置成“.example.com”，那么所有位于 catalog.example.com 的网页和所有位于 orlders.example.com 的网页，以及位于 example.com 域的其他服务器上的网页都可以访问这个 coolie。

4. secure 属性

它是一个布尔值，指定在网络上如何传输 cookie，默认是不安全的，通过一个普通的 http 连接传输

服务端设置

httpOnly

这个选项用来设置 cookie 是否能够通过 js 去访问。默认情况下，cookie 不会带 httpOnly 选项(即为空)，所以默认情况下，客户端是可以通过 js 代码去访问（包括读取、修改、删除等）这个 cookie 的。当 cookie 带 httpOnly 选项时，客户端则无法通过 js 代码去访问（包括读取、修改、删除等）这个 cookie。

在客户端是不能通过 js 代码去设置一个 httpOnly 类型的 cookie 的，这种类型的 cookie 只能通过服务端来设置。

cookie 的编码

cookie 其实是个字符串,但这个字符串中等号、分号、空格被当做了特殊符号。所以当 cookie 的 key 和 value 中含有这 3 个特殊字符时，需要对其进行额外编码，一般会用 escape 进行编码，读取时用 unescape 进行解码；当然也可以用 encodeURIComponent/decodeURIComponent 或者 encodeURI/decodeURI，查看关于编码的介绍

● localStorage, sessionStorage

localStorage 对象存储的数据没有时间限制。第二天、第二周或下一年之后，数据依然可用。

不管是 localStorage，还是 sessionStorage，可使用的 API 都相同，常用的有如下几个（以 localStorage 为例）：

保存数据：localStorage.setItem(key,value);
读取数据：localStorage.getItem(key);
删除单个数据：localStorage.removeItem(key);
删除所有数据：localStorage.clear();
得到某个索引的 key：localStorage.key(index);

sessionStorage 方法针对一个 session 进行数据存储。当用户关闭浏览器窗口后，数据会被删除。

使用方法：

保存数据：sessionStorage.setItem("website", "W3Cfuns.com");
读取数据：sessionStorage.getItem("website");

● websql

Web SQL Database 规范中定义三个核心方法：

1. openDatabase：这个方法使用现有数据库或新建数据库来创建数据库对象
2. transaction：这个方法允许我们根据情况控制事务提交或回滚
3. executeSql：这个方法用于执行 SQL 查询。
4. openDatabase

我们可以使用这样简单的一条语句，创建或打开一个本地的数据库对象

```
var db = openDatabase('testDB', '1.0', 'Test DB', 2 * 1024 * 1024);
```

openDatabase 接收五个参数：

```
var config = {  
    name: 'testDB', //数据库名字  
    version: '1.0', //数据库版本  
    desc: '测试前端数据库', //数据库描述  
    size: 2 * 1024 * 1024 //数据库大小  
};  
var db = window.openDatabase(config.name, config.version, config.desc, config.size);
```

transaction 和 executeSql 整体使用：

```
db.transaction(function(tx) {  
    tx.executeSql(sql, null, function(tx, rs) {  
        console.log('执行 sql 成功');  
    }, errorCallback);  
});
```

由于 Web SQL Database 规范已经被废弃，原因说的很清楚，当前的 SQL 规范采用 SQLite 的 SQL 方言，而作为一个标准，这是不可接受的，每个浏览器都有自己的实现这还搞毛的标准。这样浏览器兼容性就不重要了，估计慢慢会被遗忘。

● indexedDB——浏览器里边的内置数据库

IndexedDB 是 HTML5 规范里新出现的浏览器里内置的数据库。对于在浏览器里存储数据，你可以使用 cookies 或 local storage，但它们都是比较简单的技术，而 IndexedDB 提供了类似数据库风格的数据存储和使用方式。存储在 IndexedDB 里的数据是永久保存，不像 cookies 那样只是临时的。IndexedDB 里提供了查询数据的功能，在 online 和 offline 模式下都能使用。你可以用 IndexedDB 存储大型数据。

IndexedDB 里数据以对象的形式存储，每个对象都有一个 key 值索引。IndexedDB 里的操作都是事务性的。一种对象存储在一个 objectStore 里，objectStore 就相当于关系数据库里的表。IndexedDB 可以有很多 objectStore，objectStore 里可以有很多对象。每个对象可以用 key 值获取。

1、indexedDB VS LocalStorage

IndexedDB 和 LocalStorage 都是用来在浏览器里存储数据，但它们使用不同的技术，有不同的用途，你需要根据自己的情况适当的选择使用哪种。LocalStorage 是用 key-value 键值模式存储数据，但跟 IndexedDB 不一样的是，它的数据并不是按对象形式存储。它存储的数据都是字符串形式。如果你想让 LocalStorage 存储对象，你需要借助 JSON.stringify() 能将对象变成字符串形式，再用 JSON.parse() 将字符串还原成对象。但如果要存储大量的复杂的数据，这并不是一种很好的方案。毕竟，localStorage 就是专门为小数量数据设计的，它的 api 是同步的。

IndexedDB 很适合存储大量数据，它的 API 是异步调用的。IndexedDB 使用索引存储数据，各种数据库操作放在事务中执行。IndexedDB 甚至还支持简单的数据类型。IndexedDB 比 localStorage 强大得多，但它的 API 也相对复杂。

对于简单的数据，你应该继续使用 localStorage，但当你希望存储大量数据时，IndexedDB 会明显的更适合，IndexedDB 能提供你更为复杂的查询数据的方式。

2、IndexedDB vs Web SQL

WebSQL 也是一种在浏览器里存储数据的技术，跟 IndexedDB 不同的是，IndexedDB 更像是一个 NoSQL 数据库，而 WebSQL 更像是关系型数据库，使用 SQL 查询数据。W3C 已经不再支持这种技术。因为不再支持上面也大致分析了其用法，也就不再赘述。

3、IndexedDB vs Cookies

Cookies(小甜点)听起来很好吃，但实际上并不是。每次 HTTP 接受和发送都会传递 Cookies 数据，它会占用额外的流量。例如，如果你有一个 10KB 的 Cookies 数据，发送 10 次请求，那么，总计就会有 100KB 的数据在网络上传输。Cookies 只能是字符串。浏览器里存储 Cookies 的空间有限，很多用户禁止浏览器使用 Cookies。所以，Cookies 只能用来存储小量的非关键的数据。

4、IndexedDB 的用法

想要理解这个 API，最好的方法是创建一个简单的 web 应用：比如把你们班的学生的学号和姓名存储在 IndexedDB 里。IndexedDB 里提供了简单的增、删、改、查接口。

(1)、浏览器是否支持：

```
window.indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB ||  
window.msIndexedDB;  
if(!window.indexedDB){  
    console.log("你的浏览器不支持 IndexedDB");  
}
```

(2)、创建

一旦你的浏览器支持 IndexedDB，我们就可以打开它。但不能直接打开 IndexedDB 数据库。IndexedDB 需要你创建一个请求来打开它。

```
// 第一个参数，数据库的名字，第二个参数为数据库的版本  
var request = window.indexedDB.open("testDB", 2);  
var db;  
// 打开数据库失败监听
```

```

request.onerror = function(event) {
    console.log("打开 DB 失败", event);
}
// 页面第一此请求时, 或版本更新时的事件监听, 可以创建你初始化的存储数据
request.onupgradeneeded = function(event) {
    console.log("Upgrading");
    db = event.target.result;
    // var objectStore = db.createObjectStore("students", {
    //     keyPath: "rollNo"
    // });
};
// 数据库打开成功时的事件监听
request.onsuccess = function(event) {
    console.log("成功打开 DB");
    db = event.target.result;
}

```

(3)增——往 ObjectStore 里新增对象

为了往数据库里新增数据, 我们首先需要创建一个事务, 并要求具有读写权限。在 indexedDB 里任何的存取对象的操作都需要放在事务里执行。

```

//添加数据
function add() {
    db.transaction(["students"], "readwrite")
        .objectStore("students").add({
            students: "1",
            name: "zhangsan"
        })
}

```

(4)删——ObjectStore 里删除对象

删除跟新增一样, 需要创建事务, 然后调用删除接口, 通过 key 删除对象。

```

function deleteHandle() {
    var request = db.transaction(["students"], "readwrite")
        .objectStore("students").delete("1")
}

```

(5)查——通过 key 取出对象, 即往 get()方法里传入对象的 key 值, 取出相应的对象。

```

//读取数据
function select() {
    var request = db.transaction(["students"], "readwrite")
        .objectStore("students").get("1")

    request.onsuccess = function(data) {

```

```
        console.log(request.result)
    }
}
```

(6)改——为了更新一个对象，首先要把它取出来，修改，然后再放回去。

```
var transaction = db.transaction(["students"], "readwrite");
var objectStore = transaction.objectStore("students");
var request = objectStore.get(rollNo);
request.onsuccess = function(event){
    console.log("Updating : "+request.result.name + " to " + name);
    request.result.name = name;
    objectStore.put(request.result);
};
```

九、 拖放

拖放是一种常见的特性，即抓取对象以后拖到另一个位置。在 HTML5 中，拖放是标准的一部分，任何元素都能够拖放。

拖拽：Drag

释放：Drop

拖拽指的是鼠标点击源对象后一直移动对象不松手，一旦松手即释放了

(一) 源对象&目标对象

源对象：指的是我们鼠标点击的一个事物，这里可以是一张图片，一个 DIV，一段文本等等。

目标对象：指的是我们拖动源对象后移动到一块区域，源对象可以进入这个区域，可以在这个区域上方悬停(未松手)，可以松手释放将源对象放置此处(已松手)，也可以悬停后离开该区域。

(二) 事件

被拖动的源对象可以触发的事件：

- (1)ondragstart：源对象开始被拖动
- (2)ondrag：源对象被拖动过程中(鼠标可能在移动也可能未移动)
- (3)ondragend：源对象被拖动结束

目标对象可以触发的事件：

- (1)ondragenter：目标对象被源对象拖动进入
- (2)ondragover：目标对象被源对象拖动悬停在上方
- (3)ondragleave：源对象拖动离开了目标对象
- (4)ondrop：源对象拖动在目标对象上方释放/松手

(三) 事件对象

HTML5 为所有的拖动相关事件提供了一个新的属性：

e.dataTransfer { } //数据传递对象
功能：用于在源对象和目标对象的事件间传递数据

源对象上的事件处理中保存数据：

e.dataTransfer.setData(k, v); //k-v 必须都是 string 类型

目标对象上的事件处理中读取数据：

var v = e.dataTransfer.getData(k);

注：

ondragover 有一个默认行为!!! 那就是当 ondragover 触发时, ondrop 会失效!!!! 这个可能是浏览器的版本问题, 需要以后浏览器不断更新可能才会解决!!

如何阻止?

```
ondragover= function(e){ //源对象在悬停在目标对象上时
    e.preventDefault(); //阻止默认行为, 使得 drop 可以触发
}
ondrop= function(e){ //源对象松手释放在了目标对象中
    .
}
```

案例：垃圾桶

十、 FileReader

在 html5 中新增的文件操作：

File：代表一个文件对象

FileList：代表一个文件列表对象，类数组

FileReader：用于从文件中读取数据

FileWriter：用于向文件中写出数据

调用 fileReader 对象的方法

FileReader 实例拥有四个方法，其中三个是用来读取文件，另一个是用来中断读取的。需要注意的是，无论读取成功或是失败，方法并不会返回读取结果，这一结果(储存在 result 属性中)要用 FileReader 处理事件去获取：

方法名	参数	描述
abort	none	中断读取
readAsBinaryString	file	将文件转化为二进制码
readAsDataURL	file	将文件读取为 DataURL
readAsText	file,[encoding]	将文件读取为文本

readAsText：该方法有两个参数，其中第二个参数是文本的编码方式，默认值为 UTF-8。这个方法非常容易理解，将文件以文本方式读取，读取的结果即是这个文本文件中的内容。

readAsBinaryString：该方法将文件读取为二进制字符串，通常我们将其传送到后端，后端可以通过这段字符串存储文件。

readAsDataURL：这是例子程序中用到的方法，该方法将文件读取为一段以 data: 开头的字符串，这段字符串的实质就是 Data URL，Data URL 是一种将小文件直接嵌入文档的方案。这里的小文件通常是指图像与 html 等格式的文件。

事件	描述
onabort	中断时触发
onerror	出错时触发
onload	文件读取成功完成时触发
onloadend	读取完成时触发，无论读取成功或失败
onloadstart	读取开始时触发
onprogress	读取中

事件 监听函数

FileReader 包含了一整套完成的事件模型，用于捕获读取文件时的状态,下面这个表格归纳了这些事件。

案例：图片，文件拖拽至浏览器，进行展示

十一、 地理定位

常见的定位方式有基站定位，WiFi 定位，IP 定位，GPS 定位等

基站定位

基站是能进行信号交换的站点。手机能接受信号，打电话就是通过手机基站。这些基站是由国家移动通信运营商建的，比如中国移动，中国联通，中国电信。原理如下，通过手机接受不同几个基站的信号强度来判断二者之间距离，当然基站的位置信息本来是已知的，所以手机的位置就知道了。前提是手机必须处于 SIM 卡注册状态下。由于手机信号会受干扰，基站定位精度较低。而且精度也受基站的密度影响，密度越大越精准。

WiFi 定位

WiFi 是无线上网的一种技术。平时手机不连上 WiFi 的功能就能定位。大致的原理是，WiFi 信号被设备检测到，数据库记录这个 WiFi 信号和设备对应的位置。当它被越来越多的设备检测到，它的位置就可以利用这些数据通过某种算法来得出。由于信号随着距离的增加而减少，根据设备获取到的 WiFi 信号强度就可以计算出两者之间的距离。知道了周围几个点，以及与这些点之间的距离，待定位的设备位置就不难计算出来了。WiFi 定位是由谷歌提出的，主要解决了室内定位的问题。缺点是，当某个 WiFi 搬家的时候，数据库没有及时更新，就会出现定位不准的问题。

IP 定位

每个能联网的设备都被分配了一个 ip，通过查到数据库，可以粗略地知道这个 ip 所在的地理位置。你可以点这里试试 IP 定位。

GPS 定位

GPS，全称 Global Position System，全球定位系统，简单的说，就是天上有很多卫星(24 颗)，通过 4 颗卫星的位置以及卫星与待定位设备之间的距离，计算出该设备的位置。GPS 精度高，但费电。还有在室内的时候，讯号就会被阻挡，所以室内 GPS 定位不准。

AGPS 定位

AGPS 是 Assisted GPS 的意思。由于 GPS 定位最初使用都有一个冷启动时间 (2-3 分钟)，在此之前先借助其他的定位方式进行粗略地定位，然后可以较快地根据 GPS 进行精确定位。一般借用的辅助定位方式为基站定位

定位技术比较

以上的定位方式各有优缺点，实际开发中一般同时采用多个定位方式进行定位。

定位方式	应用场景	优点	缺点
基站定位	能通电话的手机	快速，耗能小	受基站的密度影响，信号也会被干扰
WiFi 定位	有 WiFi 的地方	精度尚可，解决室内定位问题	WiFi 数据库更新不及时
IP 定位	能上网的设备		精度依赖数据库

定位方式	应用场景	优点	缺点
GPS 定位	室外	精确度高	不能用于室内，首次定位较慢
AGPS 定位	同上	同上，解决了首次定位慢的问题	不能用于室内

Html5 地理定位：

浏览器支持

Internet Explorer 9、Firefox、Chrome、Safari 以及 Opera 支持地理定位。

getCurrentPosition() 方法 - 返回数据

若成功, 则 getCurrentPosition() 方法返回对象。始终会返回 latitude、longitude 以及 accuracy 属性。如果可用, 则会返回其他下面的属性。

属性	描述
coords.latitude	十进制数的纬度
coords.longitude	十进制数的经度
coords.accuracy	位置精度
coords.altitude	海拔，海平面以上以米计
coords.altitudeAccuracy	位置的海拔精度
coords.heading	方向，从正北开始以度计
coords.speed	速度，以米/每秒计
timestamp	响应的日期/时间

Geolocation 对象 - 其他有趣的方法

watchPosition() - 返回用户的当前位置，并继续返回用户移动时的更新位置（就像汽车上的 GPS）。

clearWatch() - 停止 watchPosition() 方法

下面的例子展示 watchPosition() 方法。您需要一台精确的 GPS 设备来测试该例（比如 iPhone）：

```
var x = document.getElementById("demo");

function getLocation() {
    if (navigator.geolocation) {
        console.log(111)
```



```

navigator.geolocation.getCurrentPosition(showPosition, showError, {
    // 指示浏览器获取高精度的位置，默认为 false
    enableHighAccuracy: true,
    // 指定获取地理位置的超时时间，默认不限时，单位为毫秒
    timeout: 5000,
    // 最长有效期，在重复获取地理位置时，此参数指定多久再次获取位置。
    maximumAge: 3000
});
} else {
    console.log("222")
    x.innerHTML = "Geolocation is not supported by this browser.";
}
}
getLocation()

function showPosition(position, a) {
    console.log(position, a)
    x.innerHTML = "Latitude: " + position.coords.latitude +
        "<br />Longitude: " + position.coords.longitude;
}

function showError(error) {
    console.log(error)
    switch (error.code) {
        case error.PERMISSION_DENIED:
            x.innerHTML = "User denied the request for Geolocation."
            break;
        case error.POSITION_UNAVAILABLE:
            x.innerHTML = "Location information is unavailable."
            break;
        case error.TIMEOUT:
            x.innerHTML = "The request to get user location timed out."
            break;
        case error.UNKNOWN_ERROR:
            x.innerHTML = "An unknown error occurred."
            break;
    }
}
}

```

注：上述原生方案只有在 ie 中进行测试成功，其他浏览器都不可以；因为服务器被强的原因

百度地图 sdk

开发实战：

微信 sdk, 百度地图 sdk, 高德 sdk

十二、 Video & radio

(一) Video:

直到现在，仍然不存在一项旨在网页上显示视频的标准。

今天，大多数视频是通过插件（比如 Flash）来显示的。然而，并非所有浏览器都拥有同样的插件。

HTML5 规定了一种通过 video 元素来包含视频的标准方法。

<video> 元素提供了 播放、暂停和音量控件来控制视频。

同时 <video> 元素也提供了 width 和 height 属性控制视频的尺寸.如果设置的高度和宽度，所需的视频空间会在页面加载时保留。如果没有设置这些属性，浏览器不知道大小的视频，浏览器就不能再加载时保留特定的空间，页面就会根据原始视频的大小而改变。

<video> 与</video> 标签之间插入的内容是提供给不支持 video 元素的浏览器显示的。

<video> 元素支持多个 <source> 元素. <source> 元素可以链接不同的视频文件。浏览器将使用第一个可识别的格式：

支持的格式：

MP4 = 带有 H.264 视频编码和 AAC 音频编码的 MPEG 4 文件

WebM = 带有 VP8 视频编码和 Vorbis 音频编码的 WebM 文件

Ogg = 带有 Theora 视频编码和 Vorbis 音频编码的 Ogg 文件

常用属性

属性	描述
audioTracks	返回表示可用音频轨道的 AudioTrackList 对象。
autoplay	设置或返回是否在就绪（加载完成）后随即播放视频。
buffered	返回表示视频已缓冲部分的 TimeRanges 对象。
controller	返回表示视频当前媒体控制器的 MediaController 对象。
controls	设置或返回视频是否应该显示控件（比如播放/暂停等）。
crossOrigin	设置或返回视频的 CORS 设置。
currentSrc	返回当前视频的 URL。

currentTime	设置或返回视频中的当前播放位置（以秒计）。
defaultMuted	设置或返回视频默认是否静音。
defaultPlaybackRate	设置或返回视频的默认播放速度。
duration	返回视频的长度（以秒计）。
ended	返回视频的播放是否已结束。
error	返回表示视频错误状态的 <code>MediaError</code> 对象。
height	设置或返回视频的 <code>height</code> 属性的值。
loop	设置或返回视频是否应在结束时再次播放。
mediaGroup	设置或返回视频所属媒介组合的名称。
muted	设置或返回是否关闭声音。
networkState	返回视频的当前网络状态。
paused	设置或返回视频是否暂停。
playbackRate	设置或返回视频播放的速度。
played	返回表示视频已播放部分的 <code>TimeRanges</code> 对象。
poster	设置或返回视频的 <code>poster</code> 属性的值。
preload	设置或返回视频的 <code>preload</code> 属性的值。
readyState	返回视频当前的就绪状态。
seekable	返回表示视频可寻址部分的 <code>TimeRanges</code> 对象。
seeking	返回用户当前是否正在视频中进行查找。
src	设置或返回视频的 <code>src</code> 属性的值。
startDate	返回表示当前时间偏移的 <code>Date</code> 对象。
textTracks	返回表示可用文本轨道的 <code>TextTrackList</code> 对象。
videoTracks	返回表示可用视频轨道的 <code>VideoTrackList</code> 对象。
volume	设置或返回视频的音量。
width	设置或返回视频的 <code>width</code> 属性的值。

常用事件：

`onabort`

`script` 当发生中止事件时运行脚本

<i>oncanplay</i>	<i>script</i> 当媒介能够开始播放但可能因缓冲而需要停止时运行脚本	对象方法
<i>oncanplaythrough</i>	<i>script</i> 当媒介能够无需因缓冲而停止即可播放至结尾时运行脚本	
<i>ondurationchange</i>	<i>script</i> 当媒介长度改变时运行脚本	
<i>onemptied</i>	<i>script</i> 当媒介资源元素突然为空时（网络错误、加载错误等）运行脚本	
<i>onended</i>	<i>script</i> 当媒介已抵达结尾时运行脚本	
<i>onerror</i>	<i>script</i> 当在元素加载期间发生错误时运行脚本	
<i>onloadeddata</i>	<i>script</i> 当加载媒介数据时运行脚本	
<i>onloadedmetadata</i>	<i>script</i> 当媒介元素的持续时间以及其他媒介数据已加载时运行脚本	
<i>onloadstart</i>	<i>script</i> 当浏览器开始加载媒介数据时运行脚本	
<i>onpause</i>	<i>script</i> 当媒介数据暂停时运行脚本	
<i>onplay</i>	<i>script</i> 当媒介数据将要开始播放时运行脚本	
<i>onplaying</i>	<i>script</i> 当媒介数据已开始播放时运行脚本	
<i>onprogress</i>	<i>script</i> 当浏览器正在取媒介数据时运行脚本	
<i>onratechange</i>	<i>script</i> 当媒介数据的播放速率改变时运行脚本	
<i>onreadystatechange</i>	<i>script</i> 当就绪状态（ready-state）改变时运行脚本	
<i>onseeked</i>	<i>script</i> 当媒介元素的定位属性 [1] 不再为真且定位已结束时运行脚本	
<i>onseeking</i>	<i>script</i> 当媒介元素的定位属性为真且定位已开始时运行脚本	
<i>onstalled</i>	<i>script</i> 当取回媒介数据过程中（延迟）存在错误时运行脚本	
<i>onsuspend</i>	<i>script</i> 当浏览器已在取媒介数据但在取回整个媒介文件之前停止时运行脚本	
<i>ontimeupdate</i>	<i>script</i> 当媒介改变其播放位置时运行脚本	
<i>onvolumechange</i>	<i>script</i> 当媒介改变音量亦或当音量被设置为静音时运行脚本	
<i>onwaiting</i>	<i>script</i> 当媒介已停止播放但打算继续播放时运行脚本	

方法	描述
addTextTrack()	向视频添加新的文本轨道。
canPlayType()	检查浏览器是否能够播放指定的视频类型。
load()	重新加载视频元素。
play()	开始播放视频。
pause()	暂停当前播放的视频。

实现字幕：

1. 创建字母文件.vtt:

WEBVTT

Cue-1

00:00:15.000 --> 00:00:18.000

At the left we can see...

2. 潜入文件：

```
<video controls>
  <source src="/vedio/b.mp4" type="video/mp4">

  <track label="English subtitles" kind="subtitles" srclang="en"
    src="/vedio/vtt.vtt" default>
  <track label="Deutsche Untertitel" kind="subtitles" srclang="de"
    src="/vedio/vtt.vtt">
  <track label="English chapters" kind="chapters" srclang="en"
    src="/vedio/vtt.vtt">
</video>
```

(二)Audio 对象方法

方法	描述
addTextTrack()	向音频添加新的文本轨道。
canPlayType()	检查浏览器是否能够播放指定的音频类型。
fastSeek()	在音频播放器中指定播放时间。
getStartDate()	返回新的 Date 对象，表示当前时间线偏移量。
load()	重新加载音频元素。
play()	开始播放音频。
pause()	暂停当前播放的音频。

注：其他方法跟 vedio 一样

案例：

实现音乐播放器

十三、 Swiper

十四、 Better-scroll

better-scroll 是一款重点解决移动端 (已支持 PC) 各种滚动场景需求的插件。它的核心是借鉴的 iscroll 的实现, 它的 API 设计基本兼容 iscroll, 在 iscroll 的基础上又扩展了一些 feature 以及做了一些性能优化。

better-scroll 是基于原生 JS 实现的, 不依赖任何框架。它编译后的代码大小是 63kb, 压缩后是 35kb, gzip 后仅有 9kb, 是一款非常轻量的 JS lib。

(一) 下载:

Npm 下载:

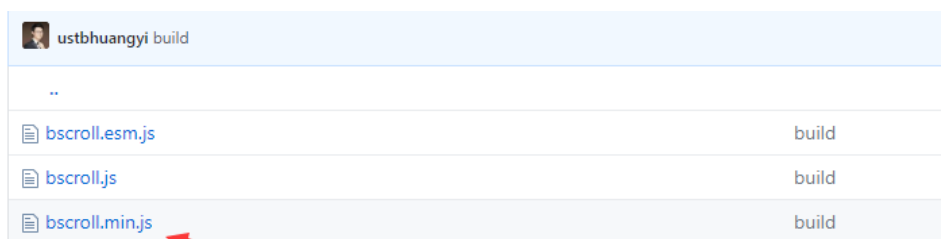
```
npm install better-scroll --save
```

js 下载:

<https://unpkg.com/better-scroll/dist/bscroll.min.js>

或者:

打开地址: <https://github.com/ustbhuangyi/better-scroll/tree/master/dist>



注: 在使用容器内部滚动时, 必须将容器进行定位, 否则会浏览器的全局滚动想冲突

(二) 使用:

Demo:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
```

```
<style>
  * {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  html {
    font-size: calc(100/750*100vw);
  }

  li {
    height: 3rem;
  }

  li:nth-child(1) {
    background: red;
  }

  li:nth-child(2) {
    background: green;
  }

  li:nth-child(3) {
    background: lightblue;
  }

  li:nth-child(4) {
    background: lightpink;
  }

  li:nth-child(5) {
    background: lightslategray;
  }
  /* 纵向滚动 */

  .vertical {
    height: 3rem;
    width: 7.5rem;
    position: relative;
    overflow: hidden;
  }
  /* 测试横向滚动 */

  .vertical li {
    width: 20rem;
  }

  .vertical ul {
```

```

        width: 20rem;
    }

</style>
</head>

<body>
    <div class="vertical">
        <ul>
            <li>1</li>
            <li>2</li>
            <li>3</li>
            <li>4</li>
            <li>5</li>
        </ul>
    </div>
    <script src="./tool/bscroll.min.js"></script>
    <script>
        let scrollIV = new BScroll('.vertical', {
            scrollbar: true, //默认为 false。当设置为 true 或者是一个 Object 的时候，都会开
            启滚动条
            scrollY: true, //垂直滚动
            scrollX: true, //水平滚动

        })

        // 事件监听相关
        let $li = document.querySelector(".vertical").children;

        [...$li].forEach((dom) => {
            dom.addEventListener('tap', function() {
                console.log(22)
            }, false)
        })

        scrollIV.on("scroll", function() {
            console.log(22)
        })
    </script>
</body>

</html>

```

(三)常用 api:

实例化滚动条

new BScroll(加滚动条的父元素,{配置项})

配置项:

```
{
  scrollbar: true, //默认为 false。当设置为 true 或者是一个 Object 的时候, 都会开启滚动条
  scrollY: true, //垂直滚动
  scrollX: true, //水平滚动
  // eventPassthrough: "horizontal", //开启此属性, 横向的滚动效果失效
  // freeScroll: true, //开启后可以实现横向和纵向同时滚动
  // click: true, //点击事件
  // dblclick: { //双击
  //   delay: 300
  // },
  // tap: true
  // bounce: { // //临界点的动画
  //   top: false,
  //   bottom: true,
  //   left: true,
  //   right: true
  // }
  // probeType: 2, //0|1|2|3 0时不会派发滚动事件 1时滚动停止之后触发一次 2 滚动过程中触发 3 实时触发
  // preventDefault: true //阻止默认事件
}
```

方法:

控制滚动条滚动到当前元素

实例对象.scrollToElement(dom 节点,time,offsetx,offsety)

time 滚动条运动到 dom 节点的时间

offsetx 滚动条距 dom 节点 x 轴的距离 true 默认在父元素的中间

offsety 滚动条距 dom 节点 y 轴的距离 true 默认在父元素的中间

控制滚动位置

实例对象.scrollTo(x,y,time)

x: x 轴的距离

y: y 轴的距离

time:时间

刷新滚动条 当子元素的 dom 结构发生变化时重新计算滚动距离

实例对象.refresh();

事件:

实例对象.on('scroll',function(){ 滚动事件 默认不会触发

this 指向实例对象

})

实例对象.on('scrollEnd',function(){ 滚动条停止滚动 回到初始状态

this 指向实例对象

})

实例对象.on('touchEnd',function(){})) 手指松开

属性:

实例对象.x 当前 x 轴的距离

实例对象.y 当前 y 轴的距离

实例对象.maxScrollX x 轴最大滚动距离

实例对象.maxScrollY y 轴最大滚动距离

只找父元素下第一个子元素,子元素的宽度和高度必须超过父元素才能撑开滚动条

父元素添加 position: relative; overflow:hidden

横向滚动条子元素的宽度不会被撑开所以要动态计算子元素的宽度

十五、 Zepto

简介: Zepto 是一个轻量级的针对现代高级浏览器的 JavaScript 库,它与 jquery 有着类似的 api。如

果你会用 jquery，那么你也会用 zepto。需要注意的是 Zepto 的一些可选功能是专门针对移动端浏览器的；因为它的最初目标在移动端提供一个精简的类似 jquery 的 js 库。

下载：http://www.css88.com/doc/zeptojs_api/#download

下载 Zepto

zepto.js v1.2.0 (开发版) – 57.3k 未压缩, 包含注释

zepto.min.js v1.2.0 (生产版) – 9.6k gzip压缩

与 jquery 的区别：

zepto 主要用在移动设备上，只支持较新的浏览器，好处是代码量比较小，性能也较好。

jquery 主要是兼容性好，可以跑在各种 pc，移动上，好处是兼容各种浏览器，缺点是代码量大，同时考虑兼容，性能也不够好。

Demo：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <script src="./zepto.js"></script>
</head>

<body>
  <div class="delete">删除</div>
  <script>
    $(''.delete').on("click", function() {
      console.log($(this))
    })
  </script>
</body>

</html>
```

十六、 FastClick

移动设备上的浏览器默认会在用户点击屏幕大约延迟 300 毫秒后才会触发点击事件，这是为了检查用户是否在做双击。为了能够立即响应用户的点击事件，才有了 FastClick。

Demo:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <!-- <meta name="viewport" content="width=device-width, initial-scale=1.0" -->
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <style>
    .topSlide {
      position: relative;
      background-color: blue;
      margin: auto;
      margin-top: 30px;
    }

    .topSlide p {
      text-align: center;
      font-size: 20px;
      line-height: 36px;
      color: white;
    }

    .result {
      position: relative;
      background-color: green;
      margin: auto;
      margin-top: 30px;
      color: white;
    }
  </style>
</head>

<body>
  <div id="t1" class="topSlide">
    <p>click me</p>
  </div>

  <div id="t2" class="topSlide">
    <p>click me(by FastClick)</p>
  </div>

  <div id="result1" class="result">
```

```

        <h3>touchStart</h3>
        <p id="result1p">haha</p>
    </div>

    <div id="result2" class="result">
        <h3>click(ms)</h3>
        <p id="result2p">haha</p>
    </div>

    <div id="result3" class="result">
        <h3>touchEnd(ms)</h3>
        <p id="result3p">haha</p>
    </div>
</script>
<script src="./fastclick.js"></script>
<script>
    if ('addEventListener' in document) {
        document.addEventListener('DOMContentLoaded', function() {
            // FastClick.attach(document.body);

            var startTime;
            var log = function(msg) {
                var t = (new Date().getTime() - startTime);
                if (msg == 'touchStart')
                    document.querySelector("#result1p").innerHTML = t;
                else if (msg == 'touchEnd')
                    document.querySelector("#result3p").innerHTML = t;
                else if (msg == 'mouseClick')
                    document.querySelector("#result2p").innerHTML = t;
                console.log(msg);
            };
            var touchStart = function() {
                startTime = new Date().getTime();
                log('touchStart');
            };
            var mouseClick = function() {
                log('mouseClick');
            };
            var touchEnd = function() {
                log('touchEnd');
            };
            var me = document.querySelector("#t2")
            FastClick.attach(me);
            document.querySelector("#t1").addEventListener("click", mouseClick);
            document.querySelector("#t2").addEventListener('click', mouseClick);
            document.addEventListener('touchstart', touchStart);
            document.addEventListener('touchend', touchEnd);
        }, false);
    }
</script>

```

</body>

</html>

不需要使用 fastclick 的情况

以下这几种情况是不需要使用 fastclick：

1、FastClick 是不会对 PC 浏览器添加监听事件

2、Android 版 Chrome 32+浏览器，如果设置 viewport meta 的值为 width=device-width，这种情况下浏览器会马上出发点击事件，不会延迟 300 毫秒。

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

3、所有版本的 Android Chrome 浏览器，如果设置 viewport meta 的值有 user-scalable=no，浏览器也是会马上出发点击事件。

4、IE11+浏览器设置了 css 的属性 touch-action: manipulation，它会在某些标签（a，button 等）禁止双击事件，IE10 的为-ms-touch-action: manipulation

使用 needsclick 过滤特定的元素

如果页面上有一些特定的元素不需要使用 fastclick 来立刻触发点击事件，可以在元素的 class 上添加 needsclick:

```
<a class="needsclick">Ignored by FastClick</a>
```

十七、Svg

SVG 指可伸缩矢量图形 (Scalable Vector Graphics)

SVG 用来定义用于网络的基于矢量的图形

SVG 使用 XML 格式定义图形

SVG 图像在放大或改变尺寸的情况下其图形质量不会有所损失

SVG 是万维网联盟的标准

SVG 与诸如 DOM 和 XSL 之类的 W3C 标准是一个整体

使用：

SVG 文件可通过以下标签嵌入 HTML 文档：<embed>、<object> 或者 <iframe>。

SVG 的代码可以直接嵌入到 HTML 页面中，或您可以直接链接到 SVG 文件。

```
<embed src="a.svg" type="image/svg+xml" />
<object data="a.svg" type="image/svg+xml"></object>
<iframe src="a.svg"></iframe>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="red" />
</svg>
```

Api 地址：<http://www.runoob.com/svg/svg-inhtml.html>

十八、 Webwork

JavaScript 语言采用的是单线程模型，也就是说，所有任务只能在一个线程上完成，一次只能做一件事。前面的任务没做完，后面的任务只能等着。随着电脑计算能力的增强，尤其是多核 CPU 的出现，单线程带来很大的不便，无法充分发挥计算机的计算能力。

Web Worker 的作用，就是为 JavaScript 创造多线程环境，允许主线程创建 Worker 线程，将一些任务分配给后者运行。在主线程运行的同时，Worker 线程在后台运行，两者互不干扰。等到 Worker 线程完成计算任务，再把结果返回给主线程。这样的好处是，一些计算密集型或高延迟的任务，被 Worker 线程负担了，主线程（通常负责 UI 交互）就会很流畅，不会被阻塞或拖慢。

Worker 线程一旦新建成功，就会始终运行，不会被主线程上的活动（比如用户点击按钮、提交表单）打断。这样有利于随时响应主线程的通信。但是，这也造成了 Worker 比较耗费资源，不应该过度使用，而且一旦使用完毕，就应该关闭。

Web Worker 有以下几个使用注意点。

(1) 同源限制

分配给 Worker 线程运行的脚本文件，必须与主线程的脚本文件同源。

(2) DOM 限制

Worker 线程所在的全局对象，与主线程不一样，无法读取主线程所在网页的 DOM 对象，也无法使用 document、window、parent 这些对象。但是，Worker 线程可以 navigator 对象和 location 对象。

(3) 通信联系

Worker 线程和主线程不在同一个上下文环境，它们不能直接通信，必须通过消息完成。

(4) 脚本限制

Worker 线程不能执行 alert()方法和 confirm()方法, 但可以使用 XMLHttpRequest 对象发出 AJAX 请求。

(5) 文件限制

Worker 线程无法读取本地文件, 即不能打开本机的文件系统 (file://), 它所加载的脚本, 必须来自网络。

使用:

主线程:

```
var worker = new Worker('work.js');
worker.postMessage('Hello World');

worker.onmessage = function(event) {
    console.log('主线程: ' + event.data);
    // 执行任务
    if (event.data !== "子线程结束") {
        worker.postMessage('stop');
    } else {
        worker.terminate();
    }
}
```

子线程:

```
//this.addEventListener("message", () => {})
// addEventListener("message", () => {})
self.addEventListener('message', function(e) {
    console.log(e.data)
    var data = e.data;
    switch (data) {
        case 'start':
            self.postMessage("子线程开始");
            break;
        case 'stop':
            self.postMessage('子线程结束');
            self.close(); // Terminates the worker.
            break;
        default:
            self.postMessage('Unknown command: ' + data.msg);
    }
}, false);
```

扩展 api:

1. Worker 加载脚本

Worker 内部如果要加载其他脚本, 有一个专门的方法 importScripts()。


```
importScripts('script1.js');
```

```
importScripts('script1.js', 'script2.js');
```

2. 错误处理

主线程可以监听 Worker 是否发生错误。如果发生错误，Worker 会触发主线程的 error 事件。

```
worker.onerror(function (event) {});
```

// 或者

```
worker.addEventListener('error', function (event) {});
```

3. 关闭 Worker

使用完毕，为了节省系统资源，必须关闭 Worker。

// 主线程

```
worker.terminate();
```

// Worker 线程

```
self.close();
```

4. 同页面的 Web Worker

通常情况下，Worker 载入的是一个单独的 JavaScript 脚本文件，但是也可以载入与主线程在同一个网页的代码。

```
<!DOCTYPE html>
```

```
<body>
```

```
<script id="worker" type="app/worker">
```

```
  addEventListener('message', function () {
```

```
    postMessage('some message');
```

```
  }, false);
```

```
</script>
```

```
</body>
```

```
</html>
```

上面是一段嵌入网页的脚本，注意必须指定<script>标签的 type 属性是一个浏览器不认识的值，上例是 application/javascript。

然后，读取这一段嵌入页面的脚本，用 Worker 来处理。

```
var blob = new Blob([document.querySelector('#worker').textContent]);
```

```
var url = window.URL.createObjectURL(blob);
```

```
var worker = new Worker(url);
```

```
worker.onmessage = function (e) {
```

```
    // e.data === 'some message'
```

```
};
```

上面代码中，先将嵌入网页的脚本代码，转成一个二进制对象，然后为这个二进制对象生成 URL，再让 Worker 加载这个 URL。这样就做到了，主线程和 Worker 的代码都在同一个网页上面。

十九、Canvas

<canvas> 是 HTML5 新增的元素，可用于通过使用 JavaScript 中的脚本来绘制图形。例如，它可以用于绘制图形，制作照片，创建动画，甚至可以进行实时视频处理或渲染。

Demo:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>

<body>
  <canvas id="canvas"></canvas>
  <script>
    var canvas = document.getElementById('canvas');
    var ctx = canvas.getContext('2d');

    ctx.fillStyle = 'green';
```

```
        ctx.fillRect(10, 10, 100, 100);
    </script>
</body>

</html>
```

基本 api 使用：

(一)颜色、样式和阴影：

1. fillStyle：设置或返回用于填充绘画的颜色、渐变或模式

属性值

<i>color</i>	指示绘图填充色的 <u>CSS 颜色值</u> 。默认值是 #000000。
<i>gradient</i>	用于填充绘图的渐变对象 (<u>线性</u> 或 <u>放射性</u>)
<i>pattern</i>	用于填充绘图的 pattern 对象

如：

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
var my_gradient = ctx.createLinearGradient(0, 0, 0, 170);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
ctx.fillStyle = my_gradient;
ctx.fillRect(20, 20, 150, 100);
```

2. strokeStyle 属性设置或返回用于笔触的颜色、渐变或模式。

属性如 fillStyle

Demo:

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

ctx.font = "30px Verdana";
// 创建渐变
var gradient = ctx.createLinearGradient(0, 0, c.width, 0);
gradient.addColorStop("0", "magenta");
gradient.addColorStop("0.5", "blue");
gradient.addColorStop("1.0", "red");
// 用渐变进行填充
```

```
ctx.strokeStyle = gradient;  
ctx.strokeText("Big smile!", 10, 50);
```

3. **shadowColor** 属性设置或返回用于阴影的颜色。

```
ctx.shadowColor="black";
```

4. **shadowBlur** 属性设置或返回阴影的模糊级数。

```
ctx.shadowBlur=20;
```

5. **shadowOffsetX** 属性设置或返回形状与阴影的水平距离。

shadowOffsetX=0 指示阴影位于形状的正下方。

shadowOffsetX=20 指示阴影位于形状 left 位置右侧的 20 像素处。

shadowOffsetX=-20 指示阴影位于形状 left 位置左侧的 20 像素处。

6. **shadowOffsetY** 属性设置或返回形状与阴影的垂直距离。

shadowOffsetY=0 指示阴影位于形状的正下方。

shadowOffsetY=20 指示阴影位于形状 top 位置下方的 20 像素处。

shadowOffsetY=-20 指示阴影位于形状 top 位置上方的 20 像素处。

7. **createLinearGradient()** 方法创建线性的渐变对象。

```
context.createLinearGradient(x0,y0,x1,y1);
```

参数	描述
<i>x0</i>	渐变开始点的 x 坐标
<i>y0</i>	渐变开始点的 y 坐标
<i>x1</i>	渐变结束点的 x 坐标
<i>y1</i>	渐变结束点的 y 坐标

8. **createPattern()** 方法在指定的方向内重复指定的元素。

```
context.createPattern(image,"repeat|repeat-x|repeat-y|no-repeat");
```

9. **createRadialGradient()** 方法创建放射状/圆形渐变对象

```
context.createRadialGradient(x0,y0,r0,x1,y1,r1);
```

10. **addColorStop()** 方法规定 gradient 对象中的颜色和位置。

```
var grd=ctx.createLinearGradient(0,0,170,0);  
grd.addColorStop(0,"black");
```

(二)线条样式

1. lineCap 属性设置或返回线条末端线帽的样式。

值	描述
butt	默认。向线条的每个末端添加平直的边缘。
round	向线条的每个末端添加圆形线帽。
square	向线条的每个末端添加正方形线帽。

2. lineJoin 属性设置或返回所创建边角的类型，当两条线交汇时。

Bevel: 创建斜角

Round: 创建圆角。

Miter: 默认。创建尖角。

3. lineWidth 属性设置或返回当前线条的宽度，以像素计。

4. miterLimit 属性设置或返回最大斜接长度。

只有当 lineJoin 属性为 "miter" 时，miterLimit 才有效。

边角的角度越小，斜接长度就会越大。

Demo:

```
ctx.lineWidth = 10;
ctx.lineJoin = "miter";
ctx.miterLimit = 5;
ctx.moveTo(20, 20);
ctx.lineTo(50, 27);
ctx.lineTo(20, 34);
ctx.stroke();
```

(三)矩形

1. rect() 方法创建矩形。

```
context.rect(x,y,width,height);
```

2. fillRect() 方法绘制“已填色”的矩形。默认的填充颜色是黑色，fillStyle 属性来设置用于填充绘图的颜色、渐变或模式。

```
context.fillRect(x,y,width,height);
```

3. **strokeRect()** 方法绘制矩形（不填色）。笔触的默认颜色是黑色。

```
context.strokeRect(x,y,width,height);
```

4. **clearRect()** 方法清空给定矩形内的指定像素。

```
context.clearRect(x,y,width,height);
```

(四) 路径

1. **fill()** 方法填充当前的图像（路径）。默认颜色是黑色。使用 **fillStyle** 属性来填充另一种颜色/

渐变

```
ctx.rect(20, 20, 150, 100);  
ctx.fillStyle = "green";  
ctx.fill();
```

2. **stroke()** 方法会实际地绘制出通过 **moveTo()** 和 **lineTo()** 方法定义的路径。默认颜色是黑色。

3. **beginPath()** 方法开始一条路径，或重置当前的路径。

```
var ctx=c.getContext("2d");
```

```
ctx.beginPath();  
ctx.lineWidth="5";  
ctx.strokeStyle="red"; // 红色路径  
ctx.moveTo(0,75);  
ctx.lineTo(250,75);  
ctx.stroke(); // 进行绘制
```

```
ctx.beginPath();  
ctx.strokeStyle="blue"; // 蓝色路径  
ctx.moveTo(50,0);  
ctx.lineTo(150,130);  
ctx.stroke(); // 进行绘制
```

4. **moveTo** 开始一条直线的方法

5. **closePath()** 方法创建从当前点到开始点的路径，无参数

6. **lineTo()** 方法添加一个新点，然后创建从该点到画布中最后指定点的线条（该方法并不会创建线条）。

7. setLineDash: 设置虚线

`setLineDash (a,b)` :a 为实线的长度, b 为虚线的长度

8. clip() 方法从原始画布中剪切任意形状和尺寸。

Demo:

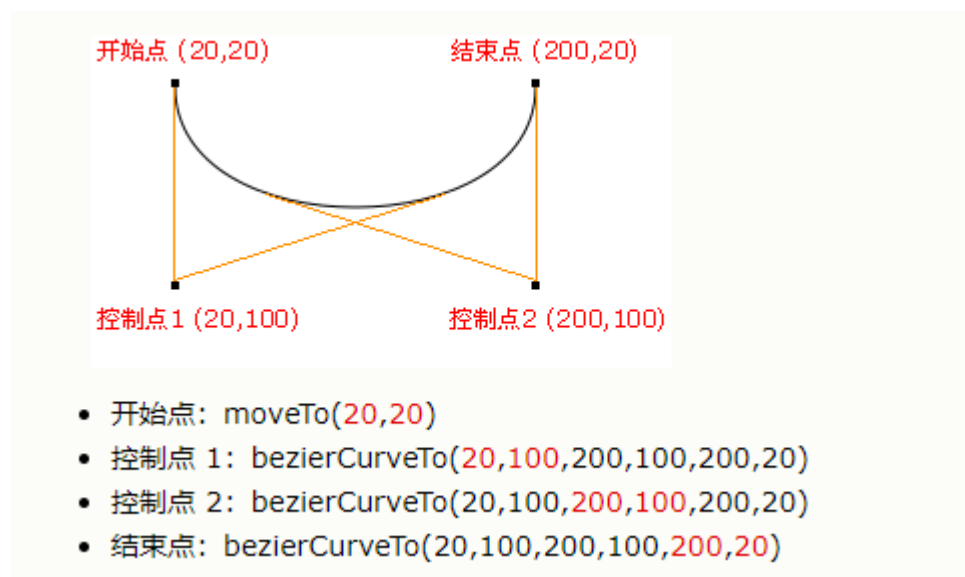
```
// 剪切矩形区域  
ctx.rect(50, 20, 200, 120);  
ctx.stroke();  
ctx.clip();
```

9. quadraticCurveTo() 方法通过使用表示二次贝塞尔曲线的指定控制点, 向当前路径添加一个点。

```
ctx.beginPath();  
ctx.moveTo(20, 20);  
ctx.quadraticCurveTo(20, 100, 200, 20);  
ctx.stroke();
```

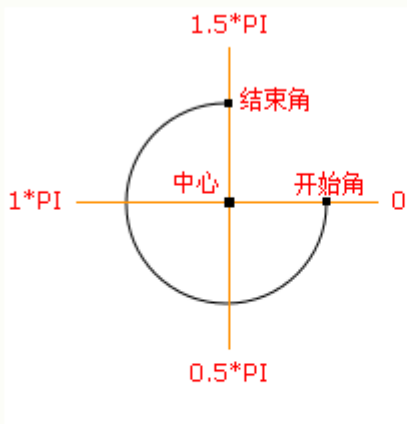
10. bezierCurveTo() 方法通过使用表示三次贝塞尔曲线的指定控制点, 向当前路径添加一个点。

```
ctx.beginPath();  
ctx.moveTo(20, 20);  
ctx.bezierCurveTo(20, 100, 200, 100, 200, 20);
```



11. arc() 方法创建弧/曲线 (用于创建圆或部分圆)。

```
ctx.beginPath();  
ctx.arc(100, 75, 50, 0, 2 * Math.PI);  
ctx.stroke();
```



- 中心: `arc(100,75,50,0*Math.PI,1.5*Math.PI)`
- 起始角: `arc(100,75,50,0,1.5*Math.PI)`
- 结束角: `arc(100,75,50,0*Math.PI,1.5*Math.PI)`

```
context.arc(x,y,r,sAngle,eAngle,counterclockwise);
```

参数	描述
<i>x</i>	圆的中心的 x 坐标。
<i>y</i>	圆的中心的 y 坐标。
<i>r</i>	圆的半径。
<i>sAngle</i>	起始角，以弧度计。（弧的圆形的三点钟位置是 0 度）。
<i>eAngle</i>	结束角，以弧度计。
<i>counterclockwise</i>	可选。规定应该逆时针还是顺时针绘图。False = 顺时针，true = 逆时针。

12. `arcTo()` 方法在画布上创建介于两个切线之间的弧/曲线。

Demo:

```
ctx.beginPath();
ctx.beginPath();
ctx.moveTo(20, 20); // 创建开始点
ctx.lineTo(100, 20); // 创建水平线
ctx.arcTo(150, 20, 150, 70, 50); // 创建弧
ctx.lineTo(150, 120); // 创建垂直线
ctx.stroke();
```

```
context.fillRect(x1,y1,x2,y2,r);
```

参数	描述
----	----

<i>x1</i>	弧的起点的 x 坐标
<i>y1</i>	弧的起点的 y 坐标
<i>x2</i>	弧的终点的 x 坐标
<i>y2</i>	弧的终点的 y 坐标
<i>r</i>	弧的半径

13. isPointInPath() 方法返回 true，如果指定的点位于当前路径中；否则返回 false。

Demo:

```
ctx.rect(20, 20, 150, 100);  
if (ctx.isPointInPath(20, 50)) {  
    ctx.stroke();  
};
```

(五)转换

1. scale() 方法缩放当前绘图，更大或更小。

Demo:

```
ctx.strokeRect(5, 5, 25, 15);  
ctx.scale(2, 2);  
ctx.strokeRect(5, 5, 25, 15);
```

2. rotate() 方法旋转当前的绘图。旋转角度

Demo:

```
ctx.rotate(20 * Math.PI / 180);  
ctx.fillRect(50, 20, 100, 50);
```

3. translate() 方法重新映射画布上的 (0,0) 位置。

```
ctx.translate(70, 70);  
ctx.fillRect(10, 10, 100, 50);
```

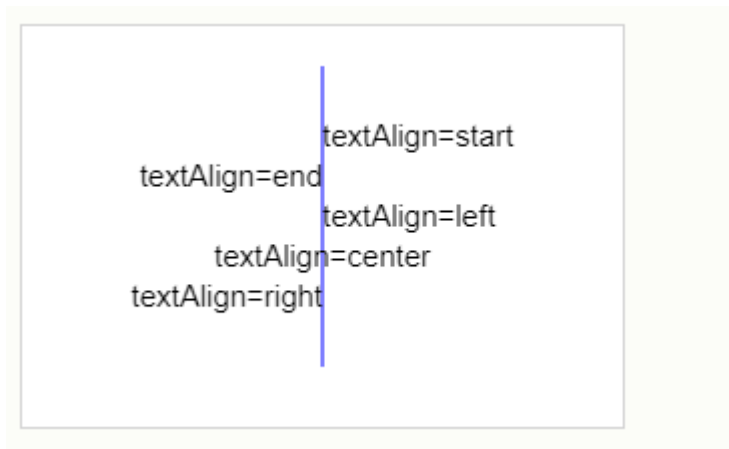
(六)文本

1. font 属性设置或返回画布上文本内容的当前字体属性。

Demo:

```
ctx.font = "40px Arial";  
ctx.fillText("Hello World", 10, 50);
```

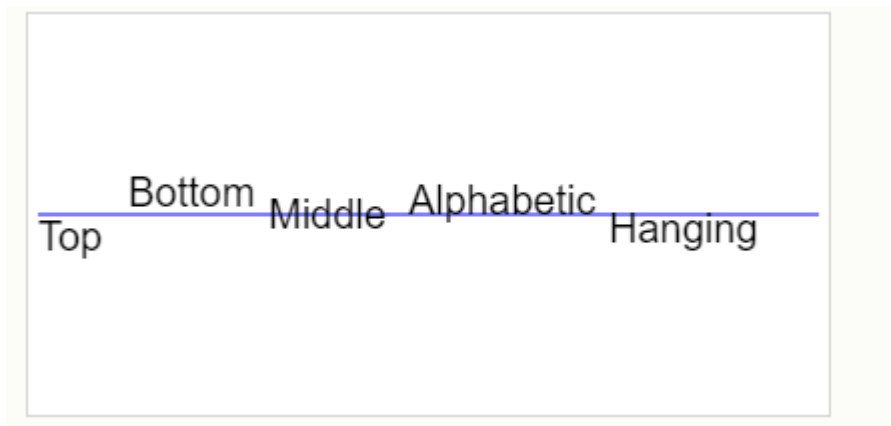
2. textAlign 属性根据锚点，设置或返回文本内容的当前对齐方式。



Demo:

```
ctx.textAlign = "end";
ctx.fillText("textAlign=end", 150, 80);
```

3. **textBaseline** 属性设置或返回在绘制文本时的当前文本基线。



```
ctx.textBaseline = "top";
ctx.fillText("Top", 5, 100);
```

4. **fillText()** 方法在画布上绘制填色的文本。文本的默认颜色是黑色。

```
ctx.font = "20px Georgia";
ctx.fillText("Hello World!", 10, 50);
```

5. **strokeText()** 方法在画布上绘制文本（没有填色）。文本的默认颜色是黑色。

Demo:

```
ctx.font = "20px Georgia";
ctx.strokeText("Hello World!", 10, 50);
```

(七)图像绘制

1. **drawImage()** 方法在画布上绘制图像、画布或视频。方法也能够绘制图像的某些部分，以及/

或者增加或减少图像的尺寸。

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

new

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

Demo1:

```
let $img = Image()
```

```
$img.src = "/img/databse.jpg";
$img.onload = function() {
    ctx.drawImage($img, 0, 0, 300, 300, 0, 0, 300, 300)
}
```

Demo2: (播放视频)

```
let v = document.querySelector("#vedios")
let lasttime
requestAnimationFrame(() => {
    if (lasttime && new Date().getTime() - lasttime > 30) {
        ctx.drawImage(v, 0, 0, 270, 135);
    }
})
```

(八)像素操作

1. createImageData() 方法创建新的空白 ImageData 对象。新对象的默认像素值 transparent black。

对于 ImageData 对象中的每个像素，都存在着四方面的信息，即 RGBA 值：

- R - 红色 (0-255)
- G - 绿色 (0-255)
- B - 蓝色 (0-255)

A - alpha 通道 (0-255; 0 是透明的, 255 是完全可见的)

因此 , transparent black 表示 (0,0,0,0)。

color/alpha 以数组形式存在, 并且既然数组包含了每个像素的四条信息, 数组的大小是 ImageData 对象的四倍。(获得数组大小有更简单的办法, 就是使用 ImageDataObject.data.length)

demo:

```
var imgData = ctx.createImageData(100, 100);
for (var i = 0; i < imgData.data.length; i += 4) {
    imgData.data[i + 0] = 255;
    imgData.data[i + 1] = 0;
    imgData.data[i + 2] = 0;
    imgData.data[i + 3] = 255;
}
ctx.putImageData(imgData, 10, 10);
```

2. 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。

对于 ImageData 对象中的每个像素, 都存在着四方面的信息, 即 RGBA 值:

R - 红色 (0-255)

G - 绿色 (0-255)

B - 蓝色 (0-255)

A - alpha 通道 (0-255; 0 是透明的, 255 是完全可见的)

color/alpha 以数组形式存在, 并存储于 ImageData 对象的 data 属性中。

Demo:

```
var imgData = ctx.getImageData(10, 10, 50, 50);
ctx.putImageData(imgData, 10, 70);
```

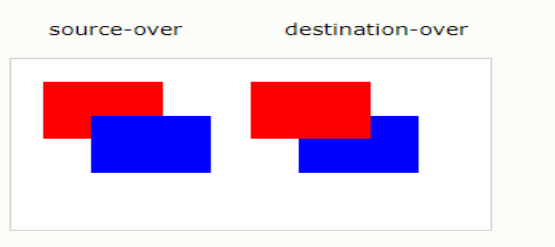
3. putImageData() 方法将图像数据 (从指定的 ImageData 对象) 放回画布上。

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

参数	描述
<i>imgData</i>	规定要放回画布的 ImageData 对象。

<i>x</i>	ImageData 对象左上角的 x 坐标，以像素计。
<i>y</i>	ImageData 对象左上角的 y 坐标，以像素计。
<i>dirtyX</i>	可选。水平值 (x)，以像素计，在画布上放置图像的位置。
<i>dirtyY</i>	可选。水平值 (y)，以像素计，在画布上放置图像的位置。
<i>dirtyWidth</i>	可选。在画布上绘制图像所使用的宽度。
<i>dirtyHeight</i>	可选。在画布上绘制图像所使用的高度。

4. **globalCompositeOperation** 属性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。



Demo:

```
ctx.fillStyle = "red";
ctx.fillRect(20, 20, 75, 50);
ctx.globalCompositeOperation = "source-over";
ctx.fillStyle = "blue";
ctx.fillRect(50, 50, 75, 50);
```

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。只会显示目标图像之外源图像部分，目标图像是透明的。
destination-over	在源图像上方显示目标图像。
destination-atop	在源图像顶部显示目标图像。源图像之外的目标图像部分不会被显示。
destination-in	在源图像中显示目标图像。只有源图像内的目标图像部分会被显示，源图像是透明的。

destination-out	在源图像外显示目标图像。只有源图像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。

(九) canvas 常用方法

1. save(): 保存当前环境的状态

2. restore(): 返回之前保存过的路径状态和属性

一般是 save 和 restore 方法相结合着使用；

Demo:

```
context.save(); //保存第一次的状态
context.rotate(30 / 180 * Math.PI);
context.beginPath();
context.moveTo(0, -180);
context.lineTo(0, -200);
context.stroke();
context.restore(); //进行第二次绘制前，将状态恢复到初始状态
```

```
context.rotate(30 / 180 * Math.PI);
context.beginPath();
context.moveTo(0, -140);
context.lineTo(0, -160);
context.stroke();
toDataURL()可以得到以 base64 编码的 dataURL
```

3. demo:

```
function getBase64(url) {
    //通过构造函数来创建的 img 实例，在赋予 src 值后就会立刻下载图片，相比
    createElement() 创建 <img> 省去了 append(), 也就避免了文档冗余和污染
    var img = new Image(),
        dataURL = "";
    img.src = url;
    img.onload = function() { //要先确保图片完整获取到，这是个异步事件
        var canvas = document.createElement("canvas"), //创建 canvas 元素
            width = img.width, //确保 canvas 的尺寸和图片一样
            height = img.height;
        canvas.width = width;
        canvas.height = height;
```

```

        canvas.getContext("2d").drawImage(img, 0, 0, width, height); //将图片绘制到 canvas 中
        dataURL = canvas.toDataURL('image/jpeg'); //转换图片为 dataURL
    };

```

(十) 弧度制和角度制：

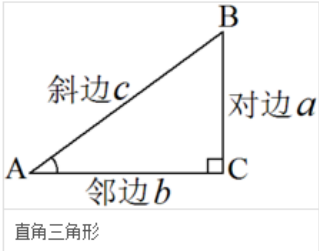
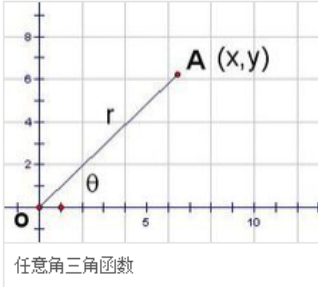
$$1^{\circ} = \pi / 180 \approx 0.01745 \text{ rad}$$

$$1\text{rad} = 180 / \pi = 57.30^{\circ}$$

角度制，就是用角的大小来度量角的大小的方法。在角度制中，把周角的 1/360 看作 1 度，那么，半周就是 180 度，一周就是 360 度。由于 1 度的大小不因为圆的大小而改变，所以角度大小是一个与圆的半径无关的量。

弧度制，顾名思义，就是用弧的长度来度量角的大小的方法。单位弧度定义为圆周上长度等于半径的圆弧与圆心构成的角。由于圆弧长短与圆半径之比，不因为圆的大小而改变，所以弧度数也是一个与圆的半径无关的量。角度以弧度给出时，通常不写弧度单位，有时记为 rad 或 R

(十一) 三角函数：

	锐角三角函数	任意角三角函数
图形	 <p>直角三角形</p>	 <p>任意角三角函数</p>
正弦 (sin)	$\sin A = \frac{a}{c}$	$\sin \theta = \frac{y}{r}$
余弦 (cos)	$\cos A = \frac{b}{c}$	$\cos \theta = \frac{x}{r}$
正切 (tan或tg)	$\tan A = \frac{a}{b}$	$\tan \theta = \frac{y}{x}$
余切 (cot或ctg)	$\cot A = \frac{b}{a}$	$\cot \theta = \frac{x}{y}$
正割 (sec)	$\sec A = \frac{c}{b}$	$\sec \theta = \frac{r}{x}$
余割 (csc)	$\csc A = \frac{c}{a}$	$\csc \theta = \frac{r}{y}$

二十、 手机设备事件：

需要用到 HTML5 的 DeviceOrientation 特性。它提供的 DeviceMotion 事件封装了设备的运动传感器时间，通过改时间可以获取设备的运动状态、加速度等数据（另还有 deviceOrientation 事件提供了设备角度、朝向等信息）。

(一) DeviceMotion

DeviceMotion 对设备运动状态的判断，则可以帮助我们在网页上就实现“摇一摇”的交互效果。

acceleration 只读

提供了设备在 X,Y,Z 轴方向上加速度的对象。加速度的单位为 m/s^2 。

accelerationIncludingGravity 只读

提供了设备在 X,Y,Z 轴方向上带重力的加速度的对象。加速度的单位为 m/s^2

rotationRate 只读

提供了设备在 alpha, beta, gamma 轴方向上旋转的速率的对象。旋转速率的单位为 $^\circ/\text{s}$ 。

interval 只读

表示从设备获取数据的频率，单位是毫秒。

Demo:

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title></title>
  <meta content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0"
name="viewport" />
  <style type="text/css">
    * {
      margin: 0;
      padding: 0;
    }

    div {
      width: 80%;
      border: solid 1px #ccc;
      padding: 5px;
      height: 300px;
      margin: 30px auto;
      line-height: 300px;
      text-align: center
    }
  </style>
</head>
```



```

<body>
  <div>摇一摇抽奖</div>
  <script type="text/javascript">
    //定义保存开始和结束的值
    var startx = starty = endx = endy = 0;
    //将多个奖项保存到数组中
    var dataA = ["一等奖", "二等奖", "三等奖", "四等奖", "谢谢参与"];
    //绑定手机摇动时的事件
    window.addEventListener("devicemotion", function(event) {
      //接收传来的数据对象
      var data = event.accelerationIncludingGravity;

      //根据对象获取开始坐标值
      startx = data.x;
      starty = data.y;

      //如果有距离
      if (startx - endx > 25 || starty - endy > 25) {
        //生成一个随机数
        var i = Math.floor(Math.random() * 5);
        //根据随机数设置奖项显示的内容
        document.querySelector("div").innerHTML = dataA[i]
      }
      endx = startx;
      endy = starty;
    })
  </script>
</body>

</html>

```

(二) DeviceOrientationEvent

要接收设备方向变化信息，你只需要注册监听 deviceorientation 事件：

根据 event 对象的三个方向的参数来确定设备的旋转角度。其中，alpha 的取值范围是 0-360,这个需要根据设

备的指南针设定情况而定，一般来说，设备指向正北方向时为 0.beta 值为设备绕 x 轴旋转的角度，取值范围为-180-180。

gamma 取值范围-90-90.

属性值

- 1.alpha 设备指示的方向，根据指南针的设定情况而定
- 2.beta 设备绕 x 轴旋转的角度
- 3.gamma 设备绕 y 轴旋转的角度

这里面 alpha 值的意义并不大，主要参考 beta 和 gamma 值。当屏幕从水平沿 y 轴向左倾斜时 gamma 值变为负值，向右倾斜变为正值。当屏幕从水平沿 x 轴向前倾斜时 beta 值变为正值，向后倾斜时变为负值。所以，如果我们设定一个阈值，当 beta 和 gamma 的绝对值大于这个阈值时，我们就认为设备发生了旋转。另外根据 beta 和 gamma 的值来判断向左倾斜还是向右倾斜，以及倾斜的程度。

两者的区别：

- 1.DeviceOrientationEvent 的值是相对于初始状态的差值，只要设备方向不变，怎么动都不会影响数值；
- 2.DeviceMotionEvent 是相对于之前的某个瞬间值的差值时间比，即变化的速度，一旦设备静止则会恢复为 0。

(三) orientationChange

此时的 event 对象不包含任何有价值的信息，

因为唯一相关信息可以通过 window.orientation 访问到

orientation 属性

它有三个值：0,90, -90

0 为竖屏模式 (portrait)，-90 意味着该设备横向旋转到右侧的横屏模式 (landscape)，而 90 表示该设备是横向旋转到左边的横屏模式 (landscape)。

还有一个是 180，表示竖屏但是是翻转过来的竖屏模式。但这种模式至今尚未得到支持。

```
<!DOCTYPE html>
```

```
<html>
```

```
<head lang="en">
```

```
<meta charset="UTF-8">
```

```
<title></title>
```

```
<meta content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0"
name="viewport" />
```

```
<style type="text/css">
```

```
* {
```

```
margin: 0;
```

```
padding: 0;
```

```
}
```

```
div {
```

```
width: 80%;
```

```
border: solid 1px #ccc;
```

```
padding: 5px;
```

```

        height: 300px;
        margin: 30px auto;
        line-height: 300px;
        text-align: center
    }
</style>
<script type="text/javascript" src="js/jquery-1.11.3.min.js">
</script>
</head>

<body>

    <div>横屏显示</div>
    <script type="text/javascript">
        $(function() {
            window.addEventListener("orientationchange", function(event) {
                //竖屏方向
                if (window.orientation == 0 || window.orientation == 180 || window.orientation ==
-180) {
                    document.querySelector("div").innerHTML = "竖屏显示"
                } else { //横屏方向
                    document.querySelector("div").innerHTML = "横屏显示"
                }
            }, false)
        })
    </script>
</body>
</html>

```

二十一、摄像头的调用

HTML 5 的 `getUserMedia` API 提供了访问媒体的能力, 基于该特性, 开发者可以不依赖任何浏览器插件下去访问视频和音频等设备. 如 `navigator.mediaDevices.getUserMedia`

不同浏览器的 api:

```

//访问用户媒体设备的兼容方法
function getUserMedia(constraints, success, error) {
    if (navigator.mediaDevices.getUserMedia) {
        //最新的标准 API
        navigator.mediaDevices.getUserMedia(constraints).then(success).catch(error);
    } else if (navigator.webkitGetUserMedia) {
        //webkit 核心浏览器
        navigator.webkitGetUserMedia(constraints, success, error)
    } else if (navigator.mozGetUserMedia) {
        //firefox 浏览器
        navigator.mozGetUserMedia(constraints, success, error);
    } else if (navigator.getUserMedia) {
        //旧版 API
    }
}

```

```

        navigator.getUserMedia(constraints, success, error);
    }
}

```

使用：

`navigator.mediaDevices.getUserMedia(constraints)` 返回一个 Promise 对象，成功后会 resolve 回调一个 `MediaStream` 对象。若用户拒绝了使用权限，或者需要的媒体源不可用，promise 会 reject 回调一个 `PermissionDeniedError` 或者 `NotFoundError`。

```

navigator.mediaDevices.getUserMedia(constraints)
    .then(function(stream) {
        /* 使用这个 stream stream */
    })
    .catch(function(err) {
        /* 处理 error */
    });

```

参数：

Constraints：

`constraints` 参数是一个包含了 `video` 和 `audio` 两个成员的 `MediaStreamConstraints` 对象，用于说明请求的媒体类型。必须至少一个类型或者两个同时可以被指定。如果浏览器无法找到指定的媒体类型或者无法满足相对应的参数要求，那么返回的 Promise 对象就会处于 rejected [失败] 状态，`NotFoundError` 作为 rejected [失败] 回调的参数。

基本用法：

```
{ audio: true, video: true }
```

使用 1280x720 的摄像头分辨率：

```

{
    audio: true,
    video: { width: 1280, height: 720 }
}

```

获取最低为 1280x720 的分辨率：

```

{
    audio: true,
    video: {
        width: { min: 1280 },
        height: { min: 720 }
    }
}

```

```
}
```

当请求包含一个 **ideal**（应用最理想的）值时，这个值有着更高的权重，意味着浏览器会先尝试找到最接近指定的理想值的设定或者摄像头（如果设备拥有不止一个摄像头）。

```
{
  audio: true,
  video: {
    width: { min: 1024, ideal: 1280, max: 1920 },
    height: { min: 776, ideal: 720, max: 1080 }
  }
}
```

或：

```
{
  audio: true,
  video: {
    width: { ideal: 1280 },
    height: { ideal: 720 }
  }
}
```

优先使用前置摄像头（如果有的话）：

```
{ audio: true, video: { facingMode: "user" } }
```

强制使用后置摄像头：

```
{ audio: true, video: { facingMode: { exact: "environment" } } }
```

二十二、 SrcObject 和 createObjectURL

SrcObject

HTMLMediaElement 接口的 srcObject 属性设定或返回一个对象，这个对象提供了一个与 HTMLMediaElement 关联的媒体源，这个对象通常是 MediaStream，但根据规范可以是 MediaSource，Blob 或者 File。

```
video.srcObject = stream;
```

createObjectURL

URL.createObjectURL() 静态方法会创建一个 DOMString，其中包含一个表示参数中给出的对象的 URL。这个 URL 的生命周期和创建它的窗口中的 document 绑定。这个新的 URL 对象表示指定的 File 对象或 Blob 对象。

```
objectURL = URL.createObjectURL(blob);
```

在每次调用 `createObjectURL()` 方法时，都会创建一个新的 URL 对象，即使你已经用相同的对象作为参数创建过。当不再需要这些 URL 对象时，每个对象必须通过调用 `URL.revokeObjectURL()` 方法来释放。浏览器会在文档退出的时候自动释放它们，但是为了获得最佳性能和内存使用状况，你应该在安全的时机主动释放掉它们。

二十三、设备信息获取

`window.navigator` 对象包含有关访问者浏览器的信息。

属性

```
navigator.appCodeName; //返回与浏览器相关的内部代码名 都为 Mozilla
navigator.appName; //返回浏览器正式名称 均为 Netscape
navigator.appVersion; //返回浏览器版本号
navigator.cookieEnabled; //返回浏览器是否启用 cookie, true 和 false
navigator.geolocation; //返回地理定位信息(h5)
navigator.javaEnabled(); //检测当前浏览器是否支持 Java, 从而知道浏览器是否能显示 Java 小程序
(IE,chrome 返回 true, firefox 返回 false)
navigator.language; //返回浏览器的首选语言
navigator.mimeTypes; //返回浏览器支持的 Mime 类型
navigator.msManipulationViewsEnabled; //仅支持 IE, true
navigator.msMaxTouchPoints; //字面意思是最大的触摸点, IE 为 0, 其他不支持
navigator.msPointerEnabled; //IE 为 true, 其他不支持
navigator.onLine; //是否连接互联网, 均返回 true(未断网)
navigator.platform; //所在平台, 返回 win32, //android,ios
navigator.plugins; //返回浏览器插件集合
navigator.preference; //允许一个已标识的脚本获取并设置特定的 Navigator 参数
navigator.product; //浏览器产品名, 返回 gecko
navigator.systemLanguage; //获取系统语言, IE 支持, 返回 zh-cn
navigator.userAgent; //判断浏览器类型
navigator.userLanguage; //返回操作系统的自然语言设置,IE 支持, 返回 zh-cn
```

方法

```
navigator.msLaunchUri; //回调函数, 未研究
navigator.taintEnabled; //回调函数
navigator.hasOwnProperty; //意思是是否支持属性, 用法如下

document.hasOwnProperty("ontouchstart"); //电脑返回 false, 手机为 true
```