[网页](https://book.apeland.cn/details/489/" \l "%E6%89%80%E6%9C%89%E7%9A%84%E6%A0%87%E7%AD%BE%E9%83%BD%E6%94%AF%E6%8C%81HTML%205%E6%96%B0%E5%B1%9E%E6%80%A7)

**HTML5有趣的新特性**

* 用于媒介回放的video和audio元素
* 新的特殊内容元素：比如article,footer,header,nav,section
* 新的表单控件：比如calendar、date、time、email、url、search
* 2D/3D绘图&效果
* 支持对本地离线存储

**HTML5浏览器支持**

最新版本的五个主流浏览器都支持某些HTML5特性，IE9以上浏览器支持HTML5新特性。但是IE8以下的浏览器不支持

IE8以下(包含IE8)以下版本浏览器兼容HTML5的方法，我们必须使用htmlshiv垫片包，让其支持HTML5新特性

1. <!--[if lt IE 9]>
2. <script src="https://cdn.jsdelivr.net/npm/html5shiv@3.7.3/dist/html5shiv.min.js"></script>
3. <![endif]-->

**HTML5新标签**

**8个新语义元素和新属性**

header,section,footer,aside,nav,main,article,figure所有的这些元素都是**块级**元素

**所有的标签都支持HTML5新属性**

| **属性** | **描述** | **浏览器支持性** |
| --- | --- | --- |
| contenteditable | 规定是否可编辑元素的内容（默认为true可编辑） | All |
| data-\* | 嵌入自定义数据 | All |
| draggable | 规定元素是否可拖动。 (默认false不可拖动) | ie8以下浏览器都不支持，其它都支持 |
| hidden | 规定对元素进行隐藏。如果使用该属性，则会隐藏元素，隐藏的元素不会被显示，可以通过js来设置hidden属性为true,使得元素变得可见 | All |

**1.nav标签**

<nav>标签定义导航链接的部分。

**2.header标签**

header标签定义文档的头部区域，它是作为网页的头部介绍内容或者是导航链接栏的容器。在一个文档中，你可以定义多个header*注意：header标签不能被放在footer、address或者另一个*header*元素内部*

**3.main标签**

标签规定文档的主要内容。一个文档中只有一个main元素。

**4.aside标签**

aside标签定义aside标签外的内容。aside的内容应该与附近的内容相关，比如像稀土掘金网站上

**5.section标签**

section标签定义了文档的某个区域。比如章节、头部、顶部或者文档的其他区域

**6.article标签**

定义页面独立的内容。必须是独立于文档的其余部分。

article通常都应用在：

* 论坛帖子
* 博客文档
* 新闻故事
* 评论

**7.figure标签**

figure标签规定独立的流内容(图像、图标、照片、代码等)

figure元素的内容应该与主内容有关，同时元素的位置相对于主内容是独立的。如果被删除，则不应对文档流产生影响

**8.footer标签**

定义文档或者文档一部分区域的页脚。该元素会包含文档闯作业的姓名、文档的版权信息、使用条款的链接、联系信息等等。在一个文档中，可以定义多个footer。比如小猿圈的底部就是用footer

**其它的新语义化标签**

**mark**

高亮文本标签

1. <p><mark> 元素用于 <mark>高亮</mark> 文本</p>

**progress**

用来显示一项任务的完成进度。

属性：

* max 最大值
* value 当前进度

1. <progress value**=**"70" max**=**"100">70 %</progress>

**address**

个人或者某个组织的联系信息等等

1. <address>
2. <a href**=**"mailto:jim@rock.com">mjj67890@163.com</a><br>
3. <a href**=**"tel:+13115552368">(311) 555-2368</a>
4. </address>

以上标签都是我们在网页中常见到的。还有很少可以在网页中见到的HTML5新标签。在这里我就不一一赘述了。大家可以参考这个链接去查阅相关资料：[新标签](https://developer.mozilla.org/zh-CN/docs/Web/Guide/HTML/HTML5/HTML5_element_list)

**新的表单特性**

HTML5新增了新的表单元素

* datalist
* keygen
* output

**datalist**

该元素规定输入域的选项列表。规定了form或input应该拥有自动完成功能，都能够用户在自动完成域中开始输入是，浏览器应该在该域中显示填写的选项。*注意：input中的list跟datalist中的id必须关联。*

**kegen(了解)**

是提供一种验证用户的可靠方法，当提交表达时，会生成两个键，一个是私钥，一个是公钥。

私钥存储于客户端，公钥则被发送给服务器。公钥可用于之后验证用户的客户端证书

1. <form action**=**"hello.asp" method**=**"get">
2. 用户名: <input type**=**"text" name**=**"usr\_name">
3. 加密: <keygen name**=**"security">
4. <input type**=**"submit">
5. </form>

**output**

用于不同类型的输出，比如计算或脚本输出

1. <form oninput**=**"x**.**value**=**parseInt**(**a**.**value**)+**parseInt**(**b**.**value**)**">0
2. <input type**=**"range" id**=**"a" value**=**"50">100 +
3. <input type**=**"number" id**=**"b" value**=**"50">=
4. <output name**=**"x" for**=**"a b"></output>
5. </form>

**HTML5新的表单属性**

**form新属性**

**autocomplete属性**

此属性规定form或input应该拥有自动完成功能

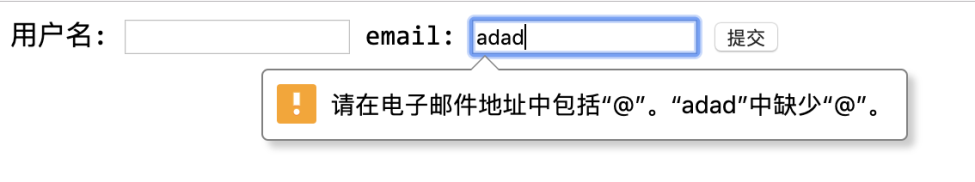
当用户在自动完成域中开始输入时，浏览器应该在该域中显示填写的选项

1. <form action**=**"" autocomplete**=**"on">
2. 用户名: <input type**=**"text" name**=**"usr\_name" >
3. <input type**=**"submit">
4. </form>

**novalidate属性**

是一个布尔值，默认为ture,当为true时，表示规定在提交表单时，不应该验证form或input域

如果给input的type改成email。则我们在输入邮箱时通常自动验证。如图所示

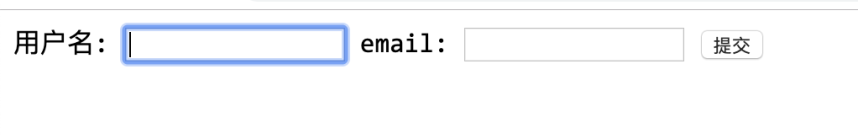


**input新属性**

**autofocus属性**

在页面加载时，自动获得焦点

1. 用户名: <input type**=**"text" name**=**"usr\_name" autofocus>



**formaction属性**

该属性用于描述表单提交的URL地址。会覆盖form元素中的action属性。该属性用于type='submit'。

1. <form action**=**"" autocomplete**=**"on" novalidate**=**"">
2. 用户名: <input type**=**"text" name**=**"usr\_name" autofocus>
3. email: <input type**=**"email">
4. <input type**=**"submit" value**=**'提交到当前服务器'>
5. <input type**=**"submit" formaction**=**"http://www.baidu.com" value**=**'提交到百度服务器'>
6. </form>

**formenctype属性**

该属性描述了表单提交到服务器的数据编码(只对form表单中 method=’post‘表单)

第一个提交按钮已默认编码发送表单数据，第二个提交按钮以 “multipart/form-data” 编码格式发送表单数据:

1. <form action**=**"" autocomplete**=**"on" novalidate**=**"" method**=**'post'>
2. 用户名: <input type**=**"text" name**=**"usr\_name" autofocus>
3. email: <input type**=**"email">
4. <input type**=**"submit" value**=**'提交到当前服务器'>
5. <input type**=**"submit" formenctype**=**"multipart/form-data" value**=**"以 Multipart/form-data 提交">
6. </form>

**formmethod属性**

formmethod 属性定义了表单提交的方式。

formmethod 属性覆盖了 <form>元素的 method 属性。

**注意:** 该属性可以与 type=”submit” 和 type=”image” 配合使用。

1. <form action**=**"" autocomplete**=**"on" novalidate**=**"" method**=**'get'>
2. 用户名: <input type**=**"text" name**=**"usr\_name" autofocus>
3. email: <input type**=**"email">
4. <input type**=**"submit" value**=**'get提交'>
5. <input type**=**"submit" method**=** 'post' formenctype**=**"multipart/form-data" value**=**"post提交">
6. </form>

**formnovalidate属性**

novalidate 属性是一个 boolean 属性.

novalidate属性描述了 <input> 元素在表单提交时无需被验证。

formnovalidate 属性会覆盖 <form> 元素的novalidate属性.

**注意:** formnovalidate 属性与type=”submit一起使用

1. <form action**=**"">
2. E-mail: <input type**=**"email" name**=**"userid">
3. <input type**=**"submit" value**=**"提交">
4. <input type**=**"submit" formnovalidate value**=**"不验证提交">
5. </form>

**formtarget属性**

formtarget属性指定一个名称或一个关键字来指明表单提交数据接收后的展示。

1. <form action**=**"">
2. 用户名: <input type**=**"text">
3. 密码: <input type**=**"password">
4. <input type**=**"submit" formtarget**=**"\_blank" value**=**"提交到一个新的页面上">
5. </form>

**height和width属性**

定义一个图像提交按钮，使用width和height属性

1. <input type**=**"image" src**=**"img\_submit.gif" width**=**"30" height**=**"30">



**list属性**

规定输入域的datalist。datalist是输入域的选项列表。前面有介绍

**multiple属性**

规定input元素可以选择多个值。适用于像input标签：file

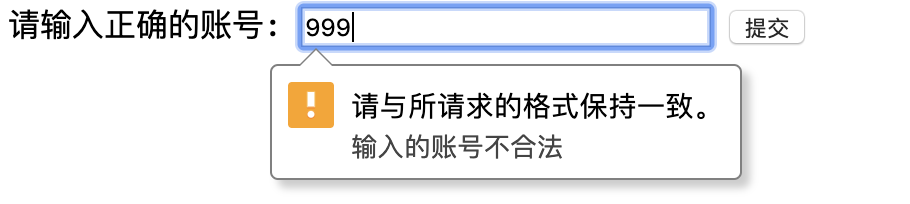
1. 上传多个文件:
2. 选择图片:<input type**=**'file' name**=** 'img' multiple>

**pattern属性**

描述了一个正则表达式用于验证input元素的值

注意：适用于以下类型input标签:text,search,url,tel,email,passworld

1. <form action**=**"">
2. 请输入正确的账号: <input type**=**"text" style**=**"width**:** 200px**;**" placeholder**=**"由字母,数字,下划线 组成,以字母开头 4-16位" pattern**=**"^[a-zA-Z]\w{3,15}$"
3. title**=**"输入的账号不合法">
4. <input type**=**"submit" value**=**"提交" />
5. </form>



**required属性**

规定必须在提交之前输入框不能为空。

1. 用户名: <input type**=**"text" name**=**"usrname" required>

**新的input类型**

HTML5拥有多个表单的输入类型。这些新特性提供了更好的输入控制和验证

新的输入类型

1. color **:** **取色**
2. date **:** **日期选择器**
3. datetime **：选择**UTC**时间**
4. datetime**-**local**:** **选择一个日期和时间(无时区)**
5. email**：提交表单时。自动验证**email**的值是否有效**
6. month**：选择月份**
7. number**:输入数值**
8. range**：包含一定范围内数字值的输入域**
9. search**:搜索域**
10. tel**:输入电话号码字段**
11. time**:选择一个时间**
12. url**:输入包含**URL**地址**
13. week**:选择周和年**

**HTM5中的API**

**新的操作方法**

**1.获取元素的方法**

获取单个元素,参数可以是我们任意的选择器。

1. document**.**querySelect**(**'选择器'**);**

获取多个元素，参数是任意的选择器

1. document**.**querySelectAll**(**'选择器'**);**

**2.类的操作**

**添加类**

1. oDiv**.**classList**.**add**(**'类名'**);**

**移除类**

1. oDiv**.**classList**.**remove**(**'类名'**);**

**检测类**

1. oDiv**.**classList**.**contains**(**'类名'**);**

**切换类**

1. oDiv**.**classList**.**toggle**(**'类名'**);**//有则删除，无则添加

**3.自定义属性**

我们可以通过data-自定义属性名来给元素添加自定义的属性名。一旦添加完成之后。通过JS可以获取以及设置自定义属性。

比如定义一个data-test属性名

**获取自定义的属性名**

1. oDiv**.**dataset**.**test

**设置自定义属性**

1. oDiv**.**dataset**.自定义属性名** **=** **值**

**文件读取**

读取文件，首先先得将文件上传，可以通过input的type为file的表单控件实现

1. <input type**=**'file' name**=**''>

其次，通过FileReader读取文件。读取完文件之后，会将结果存储在result属性中，而不是直接返回

**FileReader常用方法**

1. 1.readAsBinaryString**:** **将文件读取为二进制编码**
2. 2.readAsDataURL**:** **将文件读取为**DataURL
3. 3.readAsText**:将文件读取为文本**

**FileReader提供的事件**

1. 1.onabort
2. **读取文件中断时触发**
3. 2.onerror
4. **读取文件出错时触发**
5. 3.onload
6. **读取文件完成时触发，只在读取成功后触发**
7. 4.onloadend
8. **读取文件完成时触发，无论读取成功或失败都会触发**
9. 5.onloadstart
10. **读取文件开始时触发**
11. 6.onprogress
12. **正在读取文件**

**读取文件实例**

1. <!DOCTYPE html>
2. <html lang**=**"en">
3. <head>
4. <meta charset**=**"UTF-8">
5. <title>read file</title>
6. </head>
7. <body>
8. <input type**=**"file" name**=**"">
9. <script type**=**"text/javascript">
10. **var** input **=** document**.**querySelector**(**"input"**);**
11. input**.**onchange **=** **function()** **{**
12. // 获取到文件
13. **var** file **=** **this.**files**[**0**];**
14. // 创建读取器
15. **var** reader **=** **new** FileReader**();**
16. // 开始读取
17. reader**.**readAsText**(**file**);**
18. // 文件读取完成后，获取读取的结果
19. reader**.**onload **=** **function()** **{**
20. console**.**log**(**reader**.**result**);**
21. **}**
22. **}**
23. </script>
24. </body>
25. </html>

**获取网络状态**

window.navigator.onLine返回浏览器的网络状态。联网状态返回true,断网状态时返回false。

**实例代码**

1. <!DOCTYPE html>
2. <html lang**=**"en">
3. <head>
4. <meta charset**=**"UTF-8">
5. <title>network status</title>
6. </head>
7. <body>
8. <script type**=**"text/javascript">
9. // 获取当前网络状态
10. **var** state **=** window**.**navigator**.**onLine**;**
11. **if** **(**state**)** **{**
12. alert**(**"联网状态"**);**
13. **}**
14. **else** **{**
15. alert**(**"断网状态"**);**
16. **}**
17. </script>
18. </body>
19. </html>

**地理位置定位**

地理位置api允许用户向web应用程序提供他们的位置。处于隐私考虑，报告地理位置前会先请求用户许可

**geolocation对象**

地理位置API通过navigator.geolocation提供

1. **if(**navigator**.**geolocation**){**
2. /\*地理位置服务可用\*/
3. **}else{**
4. /\*地理位置服务不可用\*/
5. **}**

**获取当前定位**

1. **if(**navigator**.**geolocation**){**
2. /\*地理位置服务可用\*/
3. navigator**.**geolocation**.**getCurrentPosition**(function(**postion**){**
4. position**.**coords**.**latitude**;**//经度
5. position**.**coords**.**longitude**;**//纬度
6. **})**
7. **}else{**
8. /\*地理位置服务不可用\*/
9. **}**

据我最近测试，目前原生获取经纬的度的方法已不起作用，多次测试，猜测可能导致的问题是因为国内对浏览器定位有限制。

**百度地图定位**

所以，为了实现定位，我们还是用第三方的好啊。比如 [百度地图开发平台](http://lbsyun.baidu.com/index.php?title=jspopular3.0)，真的！！这个平台上想要什么都有。

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta http-equiv**=**"Content-Type" content**=**"text/html; charset=utf-8" />
5. <meta name**=**"viewport" content**=**"initial-scale=1.0, user-scalable=no" />
6. <style type**=**"text/css">
7. body**,** html**,#**allmap **{**width**:** 100**%;**height**:** 100**%;**overflow**:** hidden**;**margin**:**0**;**font**-**family**:**"微软雅黑"**;}**
8. </style>
9. <script type**=**"text/javascript" src**=**"http://api.map.baidu.com/api?v=2.0&ak=您的密钥"></script>
10. <title>地图展示</title>
11. </head>
12. <body>
13. <div id**=**"allmap"></div>
14. </body>
15. </html>
16. <script type**=**"text/javascript">
17. // 百度地图API功能
18. **var** map **=** **new** BMap**.**Map**(**"allmap"**);** // 创建Map实例
19. map**.**centerAndZoom**(new** BMap**.**Point**(**116.404**,** 39.915**),** 11**);** // 初始化地图,设置中心点坐标和地图级别
20. //添加地图类型控件
21. map**.**addControl**(new** BMap**.**MapTypeControl**({**
22. mapTypes**:[**
23. BMAP\_NORMAL\_MAP**,**
24. BMAP\_HYBRID\_MAP
25. **]}));**
26. map**.**setCurrentCity**(**"北京"**);** // 设置地图显示的城市 此项是必须设置的
27. map**.**enableScrollWheelZoom**(true);** //开启鼠标滚轮缩放
28. </script>

效果显示:

*注意：一定要去申请自己的密钥。*

注册该网站完成之后，去这[申请密钥](http://lbsyun.baidu.com/apiconsole/key/create)

在这提供了大量的[Web服务API](http://lbsyun.baidu.com/index.php?title=webapi)

百度地图Web服务API为开发者提供http/https接口，即开发者通过http/https形式发起检索请求，获取返回json或xml格式的检索数据。用户可以基于此开发JavaScript、C#、C++、Java等语言的地图应用。

不过下面的服务都是一些接口。目前我们没有学习Ajax技术。大家可以关注一下这个技术。非常重要的。等咱们学到Ajax技术之后我们就可以使用此些服务。

**本地存储**

HTML5 web存储，一个比cookie更好的本地存储方式。

随着互联网的快速发展，基于网页的应用越来越普遍，同时也变得越来越复杂。为了满足各种各样的需求，会经常在本地存储大量的数据，传统方式我们会以document.cookie来进行存储，但是由于其存储大小只有4k左右，并且解析也相当的复杂，给开发带来诸多不便，HTML5规范提出解决方案，使用sessionStorage和localStorage存储数据

**localStorage**

特点：

1. 永久存储
2. 多窗口共享
3. 容量大约为20M

**常用方法**

1. window**.**localStorage**.**setItem**(**key**,**value**);** //设置存储的内容
2. window**.**localStorage**.**getItem**(**key**);** //获取内容
3. window**.**localStorage**.**removeItem**(**key**);**//删除内容
4. window**.**localStorage**.**clear**();** //清空内容

**sessionStorage**

1. 生命周期为关闭当前浏览器窗口
2. 可以在同一个窗口下访问
3. 数据大小为5M左右
4. window**.**sessionStorage**.**setItem**(**key**,**value**);** //设置存储的内容
5. window**.**sessionStorage**.**getItem**(**key**);** //获取内容
6. window**.**sessionStorage**.**removeItem**(**key**);**//删除内容

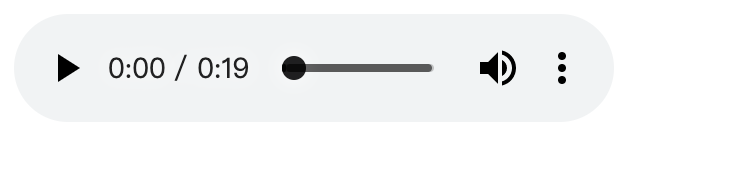
**音频audio**

HTML5提供了播放音频文件的标准。直到现在，仍然不存在在网页上播放音频的标准。今天，大多数音频都是通过插件(比如Flash)来播放的。然而，并非所有浏览器都拥有同样的插件。

**音频audio**

**基本使用**

1. <audio controls>
2. <source src**=**"my.mp3" type**=**"audio/mp3">
3. 您的刘看齐不支持audio元素
4. </audio>



常用属性

* controls属性添加音频的控件，播放、暂停和音量控件
* autoplay:使音频自动播放
* loop:使音频自动重复播放
* muted:是否静音
* volume:音量（0~1）
* paused:是否停止播放
* duration:当前音频的总时间
* currentTime:当前音频的时间

事件

1.loadstart:当浏览器开始查找音视频触发

2.progress: 当浏览器正在下载音频/视频时触发。

3.durationchange:当音视频的时长已更改时触发

4.loadedmetadata:当浏览器已加载音视频的元数据触发

5.Loadeddata:当浏览器已加载音频/视频的当前帧时触发。

6.canplay:当浏览器可以开始播放音频/视频时触发。

7.canplaythrough:.当浏览器可在不因缓冲而停顿的情况下进行播放时触发。

8.play:播放

9.pause:暂停

10.timeupdate:目前播放位置发生改变

11.ended:音视频播放结束

12.volumechange:音量发生改变

在<audio>与</audio>之间插入浏览器不支持的提示文本。audio元素允许使用多个source标签，**浏览器将使用第一个支持的音频文件。如果audio中有src,source中的链接不生效。**

在HTML5中，video标签是不支持使用绝对路径的 （未知）

**浏览器支持**

目前，此标签支持三种音视频格式文件：MP3/Wav和Ogg;

| **浏览器** | **MP3** | **Wav** | **Ogg** |
| --- | --- | --- | --- |
| Internet Explorer 9+ | YES | NO | NO |
| Chrome 6+ | YES | YES | YES |
| Firefox 3.6+ | YES | YES | YES |
| Safari 5+ | YES | YES | NO |
| Opera 10+ | YES | YES | YES |

同样，audio可以配合JS来实现自己的音乐播放器

大家可以参考MDNvideo和audio标签的相关事件：[媒体对象相关事件](https://developer.mozilla.org/zh-CN/docs/Web/Guide/Events/Media_events),[DOM相关事件](http://www.w3school.com.cn/tags/html_ref_audio_video_dom.asp)

**纯js实现古风音乐播放器**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset**=**"UTF-8">
5. <title>音乐播放器</title>
6. <link type**=**"text/css" href**=**"css/style.css" rel**=**"stylesheet" />
7. </head>
8. <body>
9. <div class**=**"btns-bg">
10. <div class**=**"PlayEy"></div>
11. <div class**=**"Btn"></div>
12. <div class**=**"Play">
13. <audio id**=**"audios" src**=**"http://music.163.com/song/media/outer/url?id=504924216.mp3"></audio>
14. </div>
15. </div>
16. <script type**=**"text/javascript" src**=**"js/script.js"></script>
17. </body>
18. </html>

style.css

1. body **{**
2. margin**:**0**;**
3. background**-**repeat**:no-**repeat**;**
4. background**-**position**:**50**%;**
5. background**-**image**:**url**(../**img**/**page**-**bg**.**png**);**
6. background**-**size**:**100**%** **auto;**
7. background**-**color**:#**efebcb
8. **}**
9. **.**PlayEy **{**
10. display**:**flex**;**
11. justify**-**content**:**center**;**
12. align**-**items**:**center**;**
13. width**:**653px**;**
14. height**:**653px**;**
15. background**:**url**(../**img**/**bg\_circle**.**png**),** url**(../**img**/**bg\_center**.**png**)** **no-**repeat center**;**
16. background**-**size**:**100**%** **%** **auto**
17. **}**
18. **.**Btn **{**
19. position**:**absolute**;**
20. display**:**flex**;**
21. justify**-**content**:**center**;**
22. align**-**items**:**center**;**
23. width**:**95px**;**
24. height**:**95px**;**
25. background**-**color**:#**ff0**;**
26. background**:**url**(../**img**/**btn**-**bg**.**png**)** **no-**repeat**;**
27. animation**:**Btn**-**bg 3s linear infinite
28. **}**
29. **.**Play **{**
30. position**:**absolute**;**
31. width**:**29px**;**
32. height**:**36px**;**
33. background**:**red**;**
34. background**:**url**(../**img**/**pause**.**png**)** **no-**repeat**;**
35. transition**:.**5s
36. **}**
37. **.**btns**-**bg **{**
38. display**:**flex**;**
39. justify**-**content**:**center**;**
40. align**-**items**:**center**;**
41. margin**:**30px **auto;**
42. width**:**653px**;**
43. height**:**653px
44. **}**
45. @keyframes Btn**-**bg **{**
46. **from** **{**
47. **}**
48. to **{**
49. transform**:**rotate**(**360deg**)**
50. **}**
51. **}**

script.js

1. **var** i **=** 0**;**
2. **var** oPlayEy **=** document**.**getElementsByClassName**(**"PlayEy"**)[**0**];**
3. **var** oPlay **=** document**.**getElementsByClassName**(**"Play"**)[**0**];**
4. **var** audios **=** document**.**getElementById**(**'audios'**);**
5. oPlay**.**onclick **=** **function** **()** **{**
6. **var** seii **=** setInterval**(function** **()** **{**
7. **(**i **==** 360**)** **?** i **=** 0**:** i**++;**
8. oPlayEy**.**style**.**transform **=** "rotate(" **+** i **+** "deg)"**;**
9. **if** **(**audios**.**paused**)** **{**
10. clearInterval**(**seii**)**
11. **}**
12. **},** 30**);**
13. **if** **(**audios**.**paused**)** **{**
14. audios**.**play**();**
15. oPlay**.**style**.**backgroundImage **=** "url(img/play.png)"**;**
16. oPlay**.**style**.**width **=** 32 **+** "px"**;**
17. oPlay**.**style**.**height **=** 32 **+** "px"**;**
18. **}** **else** **{**
19. audios**.**pause**();**
20. oPlay**.**style**.**backgroundImage **=** "url(img/pause.png)"**;**
21. oPlay**.**style**.**width **=** 29 **+** "px"**;**
22. oPlay**.**style**.**height **=** 36 **+** "px"**;**
23. **}**
24. **}**

**视频video**

**基本使用**

1. <video width**=**"800" height**=**"" controls**=**"">
2. <source src**=**"Hero.mp4" type**=**"video/mp4"></source>
3. <source src**=**"Hero.ogv" type**=**"video/ogg"></source>
4. <source src**=**"Hero.webm" type**=**"video/webm"></source>
5. 当前浏览器不支持 video直接播放
6. </video>

<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。

**浏览器支持**

| **浏览器** | **MP4** | **WebM** | **Ogg** |
| --- | --- | --- | --- |
| Internet Explorer | YES | NO | NO |
| Chrome | YES | YES | YES |
| Firefox | YES | YES | YES |
| Safari | YES | NO | NO |
| Opera | YES (从 Opera 25 起) | YES | YES |

**简单视频的DOM操作**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset**=**"utf-8">
5. <title></title>
6. </head>
7. <body>
8. <div class**=**"box">
9. <button id**=**"playOrStop">播放/暂停</button>
10. </div>
11. <video width**=**"800" height**=**"">
12. <source src**=**"Hero.mp4" type**=**"video/mp4"></source>
13. <source src**=**"Hero.ogv" type**=**"video/ogg"></source>
14. <source src**=**"Hero.webm" type**=**"video/webm"></source>
15. 当前浏览器不支持 video直接播放
16. </video>
17. <script type**=**"text/javascript">
18. **var** playOrStop **=** document**.**getElementById**(**'playOrStop'**);**
19. **var** video **=** document**.**querySelector**(**'video'**);**
20. console**.**dir**(**video**);**
21. console**.**dir**(**playOrStop**);**
22. playOrStop**.**onclick **=** **function(){**
23. console**.**log**(**video**.**paused**);**
24. **if(**video**.**paused**){**
25. video**.**play**();**
26. **}else{**
27. video**.**pause**();**
28. **}**
29. **}**
30. </script>
31. </body>
32. </html>

**HTML5实现调用摄像头**

要想实现调用摄像头，使用了html5的getUserMedia()API

代码如下：

1. <!DOCTYPE html>
2. <html lang**=**"zh">
3. <head>
4. <meta charset**=**"UTF-8">
5. <meta name**=**"viewport" content**=**"width=device-width, initial-scale=1.0">
6. <meta http-equiv**=**"X-UA-Compatible" content**=**"ie=edge">
7. <title></title>
8. </head>
9. <body>
10. <video id**=**"video" autoplay style**=**"width**:** 480px**;**height**:** 320px**;**"></video>
11. <div>
12. <button id**=**"capture">拍照</button>
13. </div>
14. <!-- 展示拍摄的照片 -->
15. <canvas id**=**"canvas" width**=**"480" height**=**"320"></canvas>
16. <script type**=**"text/javascript">
17. window**.**onload **=** **function()** **{**
18. // 1.获取标签
19. **var** video **=** document**.**getElementById**(**'video'**);**
20. **var** capture **=** document**.**getElementById**(**'capture'**);**
21. **var** ctx **=** document**.**getElementById**(**'canvas'**).**getContext**(**'2d'**);**
22. // 调用媒体对象
23. // 参数为constraints 一个约束对象 是video还是audio
24. navigator**.**mediaDevices**.**getUserMedia**({**
25. video**:** **{**
26. width**:** 480**,**
27. height**:** 320
28. **}**
29. **}).**then**(function(**stream**)** **{**
30. // 获取到window.URL对象
31. **var** URL **=** window**.**URL **||** window**.**webkitURL**;**
32. // 创建一个video的url字符串
33. **try** **{**
34. video**.**src **=** URL**.**createObjectURL**(**stream**);**
35. **}** **catch** **(**e**)** **{**
36. video**.**srcObject **=** stream**;**
37. **}**
38. // 视频播放
39. video**.**play**();**
40. **}).catch(function(**err**)** **{**
41. console**.**log**(**err**);**
42. **})**
43. // 点击拍照按钮事件
44. capture**.**onclick **=** **function()** **{**
45. ctx**.**drawImage**(**video**,**0**,**0**,**480**,**320**);**
46. **}**
47. **}**
48. </script>
49. </body>
50. </html>

**Canvas画布**

<canvas> 标签只是一个位图，它并不提供任何已经绘制在上面的对象的信息。 canvas的内容不能像语义化的HTML一样暴露给一些协助工具。一般来说，你应该避免在交互型的网站或者App上使用canvas

canvas标签只有两个属性: width和height。默认宽300px，高：150px

**渲染上下文对象**

canvas标签创造了一个固定大小的画布，它有一个或者多个**渲染上下文对象**，用它可以绘制和处理要展示的内容。接下来我们把注意力放在2D渲染上下文中:getContext(‘2d’);

**绘制矩形常用API**

绘制一个填充的矩形

1. fillRect**(**x**,**y**,**width**,**height**);**

绘制一个矩形的边框

1. strokeRect**(**x**,** y**,** width**,** height**)**

清除指定矩形区域，让清除部分完全透明。

1. clearRect**(**x**,** y**,** width**,** height**)**

上面提供的方法之中每个都包含了相同的参数。x和y指定了canvas画布上所绘制的矩形的左上角(相对于原点)的坐标。width和height设置矩形的尺寸。

**填充颜色和描边颜色设置**

1. **var** canvas **=** document**.**getElementById**(**'myCanvas'**);**
2. **var** ctx **=** canvas**.**getContext**(**'2d'**);**
3. ctx**.**fillStyle **=** 'red'**;**//填充颜色
4. ctx**.**strokeStyle **=** 'green'**;**//描边颜色

**使用路径绘制图形**

图形的基本元素是路径。路径是通过不同颜色和宽度的线段或曲线相连形成的不同形状的点的集合。一个路径，甚至一个子路径，都是闭合的。使用路径绘制图形我们需要做以下几步。

1. **创建路径起始点**
2. **使用画布的各种方法划出路径**
3. **然后把路径封闭**
4. **一旦路径生成，你就能通过描边或填充路径区域来渲染图形**

**绘制路径常用api**

**beginPath()**

1. **新建一条路径，生成之后，图形绘制**api**被指向到路径上生成路径。无参数**

**closePath()**

1. **闭合路径之后图形绘制**api**又重新指向了上下文中**

**stroke()**

1. **通过线条来绘制图形轮廓**

**fill()**

1. **通过填充路径的内容区域生成实心的图形**

**moveTo(x,y)**

1. **:将画笔移动到指定的坐标**x**以及**y**上**

**lineTo(x,y)**

1. **:绘制直线，绘制一条从当前位置到指定**x**以及**y**位置的直线**

该方法有两个参数：x以及y ，代表坐标系中直线结束的点。开始点和之前的绘制路径有关，之前路径的结束点就是接下来的开始点，开始点也可以通过moveTo()函数改变。

**arc()**

绘制圆弧或者圆

1. arc**(**x**,**y**,**radius**,**startAngle**,**endAngle**,**anticlockwise**);**

圆心在 (x, y) 位置，半径为 radius ，根据anticlockwise （默认为顺时针）指定的方向从 startAngle 开始绘制，到 endAngle 结束。

anticlockwise:可选的，布尔值，如果为true，逆时针绘制圆弧，反之，顺时针绘制

**笑脸例子**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset**=**"utf-8">
5. <title></title>
6. </head>
7. <body>
8. <canvas id**=**"myCanvas" width**=**"300" height**=**"300" style**=**"border**:**1px solid #666;"></canvas>
9. <script type**=**"text/javascript">
10. **var** canvas **=** document**.**getElementById**(**'myCanvas'**);**
11. **var** ctx **=** canvas**.**getContext**(**'2d'**);**
12. ctx**.**beginPath**();**
13. ctx**.**arc**(**75**,** 75**,** 50**,** 0**,** Math**.**PI **\*** 2**,** **true);** // 绘制
14. ctx**.**moveTo**(**110**,** 75**);**
15. ctx**.**arc**(**75**,** 75**,** 35**,** 0**,** Math**.**PI**,** **false);** // 口(顺时针)
16. ctx**.**moveTo**(**65**,** 65**);**
17. ctx**.**arc**(**60**,** 65**,** 5**,** 0**,** Math**.**PI **\*** 2**,** **true);** // 左眼
18. ctx**.**moveTo**(**95**,** 65**);**
19. ctx**.**arc**(**90**,** 65**,** 5**,** 0**,** Math**.**PI **\*** 2**,** **true);** // 右眼
20. ctx**.**stroke**();**
21. </script>
22. </body>
23. </html>

**quadraticCurveTo(cp1x,cp1y,x,y)**

1. **绘制二次贝塞尔曲线,**cp1x**,**cp1y**为一个控制点，**x**,**y**为结束点**

**bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)**

1. **绘制三次贝塞尔曲线，**cp1x**,**cp1y**为控制点一，**cp2x**,**cp2y**为控制点二，**x**,**y**为结束点。**

参数x、y在两个方法中都是结束点坐标。cp1x,cp1y为坐标的第一个控制点(上图中的红色点)，cp2x,cp2y为坐标中的第二个控制点

**二次贝赛尔曲线绘制对话气泡**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset**=**"utf-8">
5. <title></title>
6. </head>
7. <body>
8. <canvas id**=**"myCanvas" width**=**"300" height**=**"300" style**=**"border**:**1px solid #666;"></canvas>
9. <script type**=**"text/javascript">
10. **var** canvas **=** document**.**getElementById**(**'myCanvas'**);**
11. **var** ctx **=** canvas**.**getContext**(**'2d'**);**
12. ctx**.**beginPath**();**
13. ctx**.**moveTo**(**75**,** 25**);**
14. ctx**.**quadraticCurveTo**(**25**,** 25**,** 25**,** 62.5**);**
15. ctx**.**quadraticCurveTo**(**25**,** 100**,** 50**,** 100**);**
16. ctx**.**quadraticCurveTo**(**50**,** 120**,** 30**,** 125**);**
17. ctx**.**quadraticCurveTo**(**60**,** 120**,** 65**,** 100**);**
18. ctx**.**quadraticCurveTo**(**125**,** 100**,** 125**,** 62.5**);**
19. ctx**.**quadraticCurveTo**(**125**,** 25**,** 75**,** 25**);**
20. ctx**.**stroke**();**
21. </script>
22. </body>
23. </html>

三次贝塞尔曲线绘制

1. //三次贝塞尔曲线
2. ctx**.**beginPath**();**
3. ctx**.**moveTo**(**75**,**40**);**
4. ctx**.**bezierCurveTo**(**75**,**37**,**70**,**25**,**50**,**25**);**
5. ctx**.**bezierCurveTo**(**20**,**25**,**20**,**62.5**,**20**,**62.5**);**
6. ctx**.**bezierCurveTo**(**20**,**80**,**40**,**102**,**75**,**120**);**
7. ctx**.**bezierCurveTo**(**110**,**102**,**130**,**80**,**130**,**62.5**);**
8. ctx**.**bezierCurveTo**(**130**,**62.5**,**130**,**25**,**100**,**25**);**
9. ctx**.**bezierCurveTo**(**85**,**25**,**75**,**37**,**75**,**40**);**
10. ctx**.**fill**();**

**rect(x,y,width,height)**

1. **绘制矩形**

*注意：当你调用fill()函数时，所有没有闭合的形状都会自动闭合，所以你需要调用closePath()函数。但是调用stroke()时不会自动闭包*

setLineDash

cxt.setLineDash( 数组 )

参数中这个数组，是由实线和空白组合合成，如：

[ 20, 5 ]: 20px 实线，5px空白

[ 20, 5, 10, 5]: 20px实线，5px空白, 10px实线, 5px空白

**样式和颜色**

* fillStyle = color:设置图形的填充颜色
* strokeStyle = color: 设置图形边框的颜色
* globalAlpha :设置透明度值，取值范围为0~1之间的数值
* lineWidth = value:设置线条宽度，数值无单位
* lineCap = type 设置线段末端的样式
  + type:butt 默认值，方形
  + type:round 圆形
  + type:square 以方形结束，但是增加一个宽度和线段相同，宽度是线段宽度一半的矩形区域
* lineJoin = type:设定线条和线条连接的样式
  + type:round 通过填充一个额外的，圆心在相连部分末端的扇形，绘制拐角的形状。 圆角的半径是线段的宽度。
  + type:bevel 在相连部分的末端填充一个额外的以三角形为底的区域， 每个部分都有各自独立的矩形拐角。
  + type: miter 通过延伸相连部分的外边缘，使其相交于一点，形成一个额外的菱形区域

**绘制文本**

canvas提供了两种方法来渲染文本

* filleText(text,x,y,[,maxWidth])

在指定的(x,y)位置填充指定的文本，绘制的最大宽度是可选的

* strokeText(text,x,y,[,maxWidth])

在指定的(x,y)位置绘制文本边框，绘制的最大宽度是可选的

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset**=**"utf-8">
5. <title></title>
6. </head>
7. <body>
8. <canvas id**=**"canvas" width**=**"500" height**=**"200" style**=**"border**:** 1px solid #666;"></canvas>
9. <script type**=**"text/javascript">
10. **var** canvas **=** document**.**getElementById**(**'canvas'**);**
11. **var** ctx **=** canvas**.**getContext**(**'2d'**);**
12. // 设置字体大小和字体
13. ctx**.**font **=** '30px 微软雅黑'**;**
14. ctx**.**fillText**(**'hello world'**,**50**,**50**);**
15. ctx**.**strokeText**(**'hello world'**,**50**,**80**);**
16. </script>
17. </body>
18. </html>

**有样式的文本**

* font = value

当前我们用来绘制文本的样式. 这个字符串使用和 CSS font属性相同的语法. 默认的字体是 10px sans-serif。

* textAlign = value

文本对齐选项. 可选的值包括：start, end, left, right or center. 默认值是 start。

* textBaseline = value

基线对齐选项. 可选的值包括：top, hanging, middle, alphabetic, ideographic, bottom。默认值是 alphabetic。

* direction = value

文本方向。可能的值包括：ltr, rtl, inherit。默认值是 inherit。

**使用图片**

canvas更有意思的一项特性就是图像操作能力。可以用于动态的图像合成或者作为图形的背景，以及游戏界面（Sprites）等等。浏览器支持的任意格式的外部图片都可以使用，比如PNG、GIF或者JPEG。 你甚至可以将同一个页面中其他canvas元素生成的图片作为图片源。

引入图像到canvas里需要以下两步基本操作：

1. 获得一个指向HTMLImageElement的对象或者另一个canvas元素的引用作为源，也可以通过提供一个URL的方式来使用图片
2. 使用drawImage()函数将图片绘制到画布上

**核心方法**

（1）4个参数：在画布上定位图像，并规定图像的宽度和高度：

drawImage(img,x,y,width,height);

（2）8个参数：在画布上定位被剪切的部分，并规定剪切部分的宽高，在画布上定位图像，并规定图像的宽度和高度

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

**例子：一个简单的线图**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset**=**"utf-8">
5. <title></title>
6. </head>
7. <body>
8. <canvas id**=**"canvas" width**=**"500" height**=**"500"></canvas>
9. <script type**=**"text/javascript">
10. **var** ctx **=** document**.**getElementById**(**'canvas'**).**getContext**(**'2d'**);**
11. **var** img **=** **new** Image**();**
12. img**.**onload **=** **function()** **{**
13. ctx**.**drawImage**(**img**,** 0**,** 0**);**
14. ctx**.**beginPath**();**
15. ctx**.**moveTo**(**123**,** 400**);**
16. ctx**.**lineTo**(**198**,** 350**);**
17. ctx**.**lineTo**(**300**,** 200**);**
18. ctx**.**lineTo**(**500**,** 150**);**
19. ctx**.**stroke**();**
20. **}**
21. img**.**src **=** 'image-20190703114102416.png'**;**
22. </script>
23. </body>
24. </html>

**状态的保存和恢复**

save()

保存画布的所有状态（相当于压入栈里面）

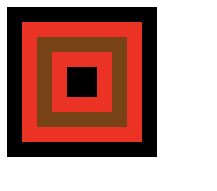
restore()

save和restore方法是用来保存和恢复canvas状态的。都没有参数。

**栗子：**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset**=**"utf-8">
5. <title></title>
6. </head>
7. <body>
8. <canvas id**=**"canvas" width**=**"200" height**=**"200"></canvas>
9. <script type**=**"text/javascript">
10. **function** draw**(){**
11. **var** ctx **=** document**.**getElementById**(**'canvas'**).**getContext**(**'2d'**);**
12. ctx**.**fillRect**(**0**,**0**,**150**,**150**);**//默认绘制一个矩形
13. ctx**.**save**();**//保存当前状态
14. ctx**.**fillStyle **=** 'red'**;** //在原有配置基础上对颜色发生改变
15. ctx**.**fillRect**(**15**,**15**,**120**,**120**);**
16. ctx**.**save**();**//保存当前状态
17. ctx**.**fillStyle **=** 'green'**;** //再次改变颜色
18. ctx**.**globalAlpha **=** 0.5**;** //改变透明度
19. ctx**.**fillRect**(**30**,**30**,**90**,**90**);**//使用新的配置绘制一个矩形
20. ctx**.**restore**();** //重载之前的颜色状态 (红色)
21. ctx**.**fillRect**(**45**,**45**,**60**,**60**);** //使用上一次的配置绘制一个矩形
22. ctx**.**restore**();** //加载默认的颜色配置
23. ctx**.**fillRect**(**60**,**60**,**30**,**30**);** //使用默认配置绘制矩形
24. **}**
25. draw**();**
26. </script>
27. </body>
28. </html>

效果展示：



**移动translate**

（移动的是原点）

translate(x,y)

translate方法接收两个参数。x是左右偏移量，y是上下偏移量。**在做变形之前先保存状态是良好的一个习惯**

**旋转**

（旋转的是已原点为旋转中心的画布）

rotate(angle)

只接受一个参数：选装的角度。顺时针方向

**基本动画**

步骤：

1. **清空canvas**

通过clearReact()来清空canvas，保证自己的画布是干净的

1. **保存canvas状态**
2. **绘制动画图形**
3. **恢复canvas**状态

**操控动画的方法**

setInterval(functuon,delay)

在设定好间隔时间后，function会定期执行

setTimeout(function,delay)

在设定好的时间之后执行函数

requestAnimationFrame(callback)

此方法一般每秒钟回到函数执行60次。告诉浏览器你希望执行一个动画，并在重绘之前，请求浏览器执行一个特定的函数来更新动画。

**SVG**

SVG 是一种基于 XML 语法的图像格式，全称是可缩放矢量图（Scalable Vector Graphics）。其他图像格式都是基于像素处理的，SVG 则是属于对图像的形状描述，所以它本质上是文本文件，体积较小，且不管放大多少倍都不会失真。

**语法以及常用标签**

**1.svg标签**

SVG代码都放在顶层标签SVG之中

1. <svg width**=**"100%" height**=**"100%">
2. <circle id**=**"mycircle" cx**=**"50" cy**=**"50" r**=**"50" />
3. </svg>

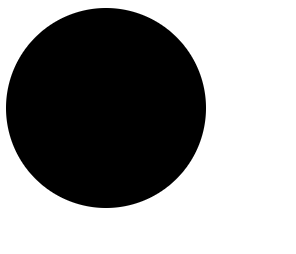
svg的width属性和height属性，指定SVG图像在HTML元素中所占据的宽度和高度。如果不指定，默认大小是300px宽，150px高

**2.circle标签**

circle标签代表圆形

1. <svg width**=**"300" height**=**"180">
2. <circle cx**=**"30" cy**=**"50" r**=**"25" />
3. <circle cx**=**"90" cy**=**"50" r**=**"25" class**=**"red" />
4. <circle cx**=**"150" cy**=**"50" r**=**"25" class**=**"fancy" />
5. </svg>

上面代码定义了三个圆。circle标签cx,cy,r属性分别为横坐标、纵坐标和半径，单位为像素。坐标都是相对于svg画布的左上角原点。

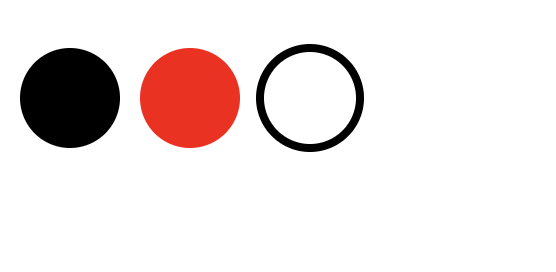


class属性用来指定对应的css类

1. **.**red **{**
2. fill**:** red**;**
3. **}**
4. **.**fancy **{**
5. fill**:** none**;**
6. stroke**:** black**;**
7. stroke**-**width**:** 3pt**;**
8. **}**

SVG 的 CSS 属性与网页元素有所不同。

* fill：填充色
* stroke：描边色
* stroke-width：边框宽度



**3.line标签**

line标签用来绘制直线

1. <line x1**=**"0" y1**=**"0" x2**=**"200" y2**=**"0" style**=**"stroke**:**rgb**(**0**,**0**,**0**);**stroke**-**width**:**5" />

* x1：线段起点的横坐标
* y1：起点纵坐标
* x2： 终点的横坐标
* y2：终点的纵坐标
* style：线段的样式

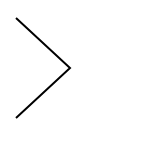


**4.polyline标签**

绘制一根折线

1. <polyline points**=**"3,3 30,28 3,53" fill**=**"none" stroke**=**"black" />

<polyline>的points属性指定了每个端点的坐标，横坐标与纵坐标之间与逗号分隔，点与点之间用空格分隔。



**5.rect标签**

绘制矩形

1. <rect x**=**"0" y**=**"0" height**=**"100" width**=**"200" style**=**"stroke**:** #70d5dd; fill: #dd524b" />

<rect>的x属性和y属性，指定了矩形左上角端点的横坐标和纵坐标；width属性和height属性指定了矩形的宽度和高度（单位像素）。

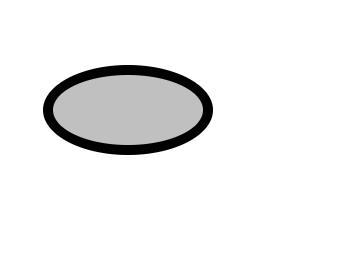


**6.ellipse标签**

绘制椭圆

1. <ellipse cx**=**"60" cy**=**"60" ry**=**"20" rx**=**"40" stroke**=**"black" stroke-width**=**"5" fill**=**"silver"/>

<ellipse>的cx属性和cy属性，指定了椭圆中心的横坐标和纵坐标（单位像素）；rx属性和ry属性，指定了椭圆横向轴和纵向轴的半径（单位像素）。



**7.polygon标签**

绘制多边形

1. <polygon fill**=**"green" stroke**=**"orange" stroke-width**=**"1" points**=**"0,0 100,0 100,100 0"/>

<polygon>的points属性指定了每个端点的坐标，横坐标与纵坐标之间与逗号分隔，点与点之间用空格分隔。



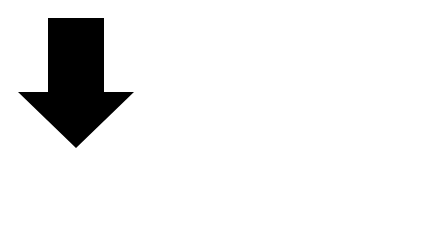
**8.path标签**

制作路径

1. <path d**=**"
2. M 18,3
3. L 46,3
4. L 46,40
5. L 61,40
6. L 32,68
7. L 3,40
8. L 18,40
9. Z
10. "></path>

<path>的d属性表示绘制顺序，它的值是一个长字符串，每个字母表示一个绘制动作，后面跟着坐标

* M：移动到（moveto）
* L：画直线到（lineto）
* Z：闭合路径

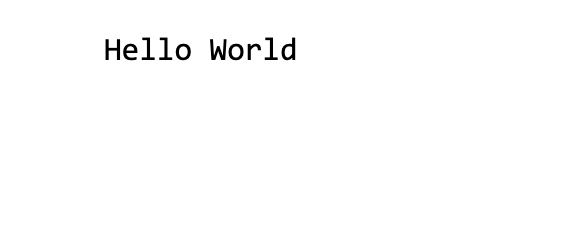


**9.text标签**

绘制文本

1. <text x**=**"50" y**=**"25">Hello World</text>

<text>的x属性和y属性，表示文本区块基线（baseline）起点的横坐标和纵坐标。文字的样式可以用class或style属性指定。

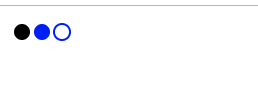


**10.use标签**

复制一个形状

1. <circle id**=**"myCircle" cx**=**"5" cy**=**"5" r**=**"4"/>
2. <use href**=**"#myCircle" x**=**"10" y**=**"0" fill**=**"blue" />
3. <use href**=**"#myCircle" x**=**"20" y**=**"0" fill**=**"white" stroke**=**"blue" />

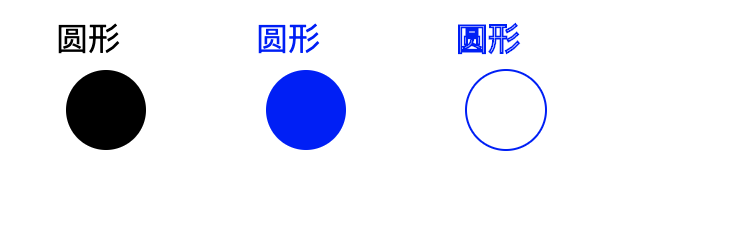
<use>的href属性指定所要复制的节点，x属性和y属性是<use>左上角的坐标。另外，还可以指定width和height坐标。



**11.g标签**

用于将多个形状组成一个组(group),方便复用

1. <g id**=**"myCircle">
2. <text x**=**"25" y**=**"20">圆形</text>
3. <circle cx**=**"50" cy**=**"50" r**=**"20" />
4. </g>
5. <use href**=**"#myCircle" x**=**"100" y**=**"0" fill**=**"blue" />
6. <use href**=**"#myCircle" x**=**"200" y**=**"0" fill**=**"white" stroke**=**"blue" />



**12.defs标签**

用于自定义形状。它内部的代码不会显示，仅供引用

**13.image标签**

用于插入图片文件

1. <image xlink:href **=** "img\_submit.gif" width**=**"20%" height**=**"20%"/>

<image>的xlink:href属性表示图像的来源。



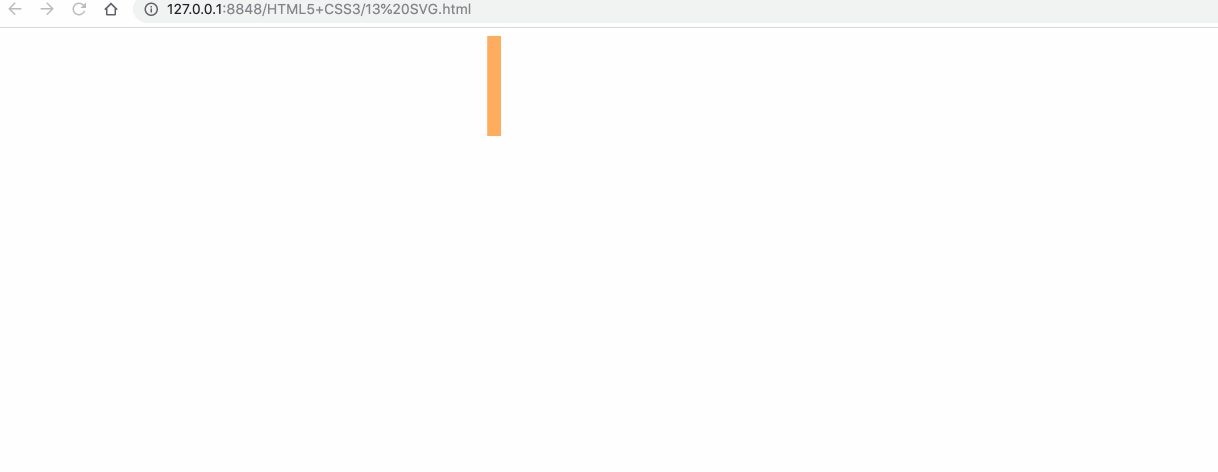
**14.animate标签**

产生动画效果

1. <rect x**=**"0" y**=**"0" width**=**"100" height**=**"100" fill**=**"#feac5e">
2. <animate attributeName**=**"x" from**=**"0" to**=**"500" dur**=**"2s" repeatCount**=**"indefinite" />
3. </rect>

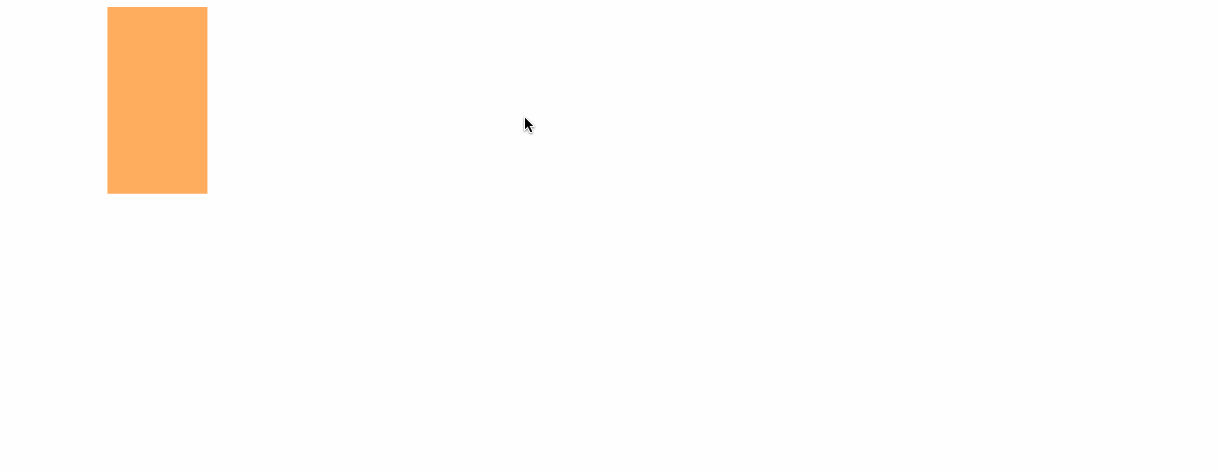
矩形会在水平反向上不断移动，产生动画效果。

* attributeName：发生动画效果的属性名。
* from：单次动画的初始值。
* to：单次动画的结束值。
* dur：单次动画的持续时间。
* repeatCount：动画的循环模式。



定制多个animate

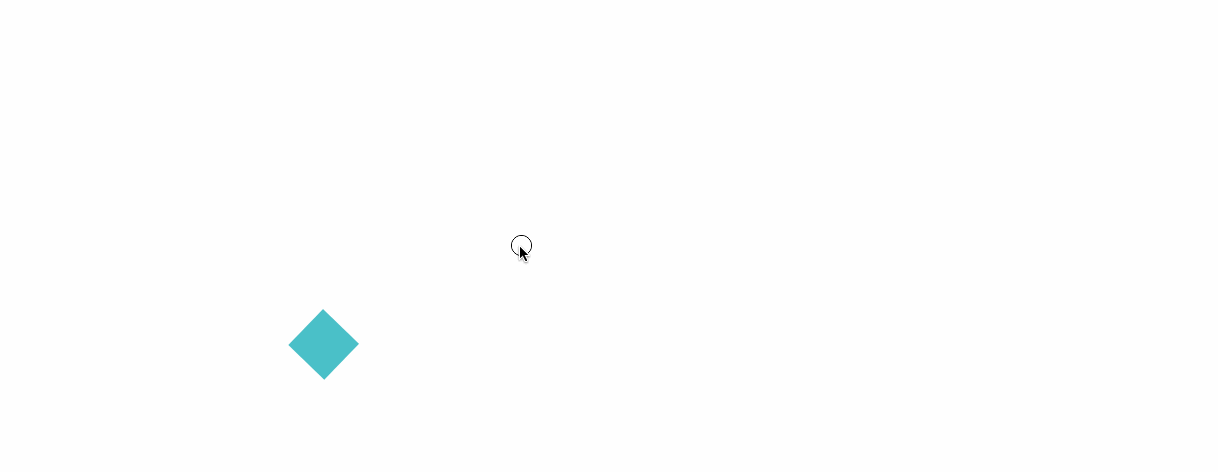
1. <animate attributeName**=**"x" from**=**"0" to**=**"500" dur**=**"2s" repeatCount**=**"indefinite" />
2. <animate attributeName**=**"height" to**=**"500" dur**=**"2s" repeatCount**=**"indefinite" />



**15.animateTransform标签**

1. <rect x**=**"250" y**=**"250" width**=**"50" height**=**"50" fill**=**"#4bc0c8">
2. <animateTransform attributeName**=**"transform" type**=**"rotate" begin**=**"0s" dur**=**"10s" from**=**"0 200 200" to**=**"360 400 400"
3. repeatCount**=**"indefinite" />
4. </rect>

<animateTransform>的效果为旋转（rotate），这时from和to属性值有三个数字，第一个数字是角度值，第二个值和第三个值是旋转中心的坐标。from="0 200 200"表示开始时，角度为0，围绕(200, 200)开始旋转；to="360 400 400"表示结束时，角度为360，围绕(400, 400)旋转。



关于SVG其实跟canvas差不了多少，都是可以做一下复杂的图形和动画效果。SVG的入门教学先写到这里。如果有同学在公司用到了SVG的高级特效，可以参考MDN[SVG](https://developer.mozilla.org/zh-CN/docs/Web/SVG)。

**拖拽/拖放**

需要取消浏览器的默认行为：dragover,dragenter,drop

拖放(drap&drop)在我们平时的工作中，经常遇到。它表示：抓取对象以后拖放到另一个位置。目前，它是HTML5标准的一部分。

**拖放的基本操作**

1. **选中=>拖动=>释放**

**选中**

在H5标准中，为了使元素克拖动，把draggable属性设置为true。文本、图片和链接式默认可以拖放的，它们的draggable属性默认被设置了true。

图片和链接按住鼠标左键选中，就可以拖放。

文本只有在被选中的情况下才能拖放。如果显示设置为恩本的draggable属性为true,按住鼠标左边也可以直接拖放。

**语法**

1. <element draggle**=**"true|false|auto"></element>

* true: 可以拖动
* false: 禁止拖动
* auto: 跟随浏览器定义是否可以拖动

**拖动**

每个可拖动的元素，在拖动过程中，都会经历三个过程：

1. **拖动开始=>拖动过程中=>拖动结束**

| **对象** | **事件名称** | **描述** |
| --- | --- | --- |
| 被拖动的元素对象 | dragstart | 在元素开始被拖动时候触发 |
|  | drag | 在元素被拖动时反复触发 |
|  | dragend | 在拖动操作完成时触发 |

| **对象** | **事件名称** | **描述** |
| --- | --- | --- |
| 目标对象 | dragenter | 当被拖动元素进入目标元素占据的屏幕空间时触发 |
|  | dragover | 当被拖动元素在目标元素内时触发 |
|  | dragleave | 当被拖动元素没有放下就离开目标元素时触发 |

*注意:dragenter和dragover事件的默认行为是拒绝接受任何被拖放的元素。因此，我们必须阻止浏览器这种默认行为。e.prevenDefault();*

**释放**

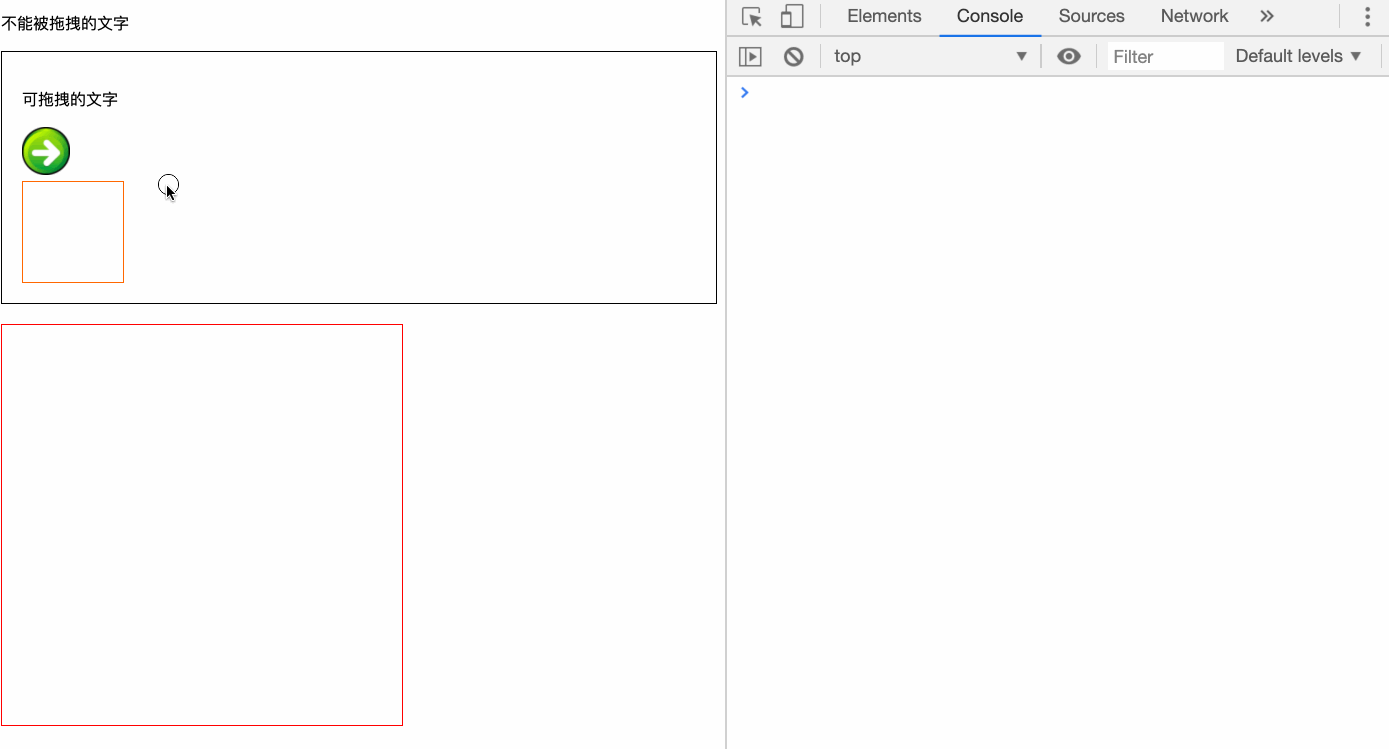
到达目的地之后，释放元素事件

| **对象** | **事件名称** | **描述** |
| --- | --- | --- |
| 目标对象 | drop | 当被拖动元素在目标元素里放下时触发，一般需要取消浏览器的默认行为。  (需要在dragover事件中使用e.preventDefault) |

看个例子演示每个API

1. <!DOCTYPE HTML>
2. <html>
3. <head>
4. <title>拖放示例-文本</title>
5. </head>
6. <style>
7. #current{
8. border**:** 1px solid #000;
9. padding**:** 20px**;**
10. margin**-**bottom**:** 20px**;**
11. **}**
12. #target{
13. border**:** 1px solid #f00;
14. width**:** 400px**;**
15. height**:** 400px**;**
16. **}**
17. **.**box**{**
18. width**:** 100px**;**
19. height**:** 100px**;**
20. border**:** 1px solid #FF6700;
21. **}**
22. </style>
23. <body>
24. <p>不能被拖拽的文字</p>
25. <div id**=**"current">
26. <div class**=**"text" id**=**"text">
27. <p draggable**=**"true">可拖拽的文字</p>
28. <img src**=**"img\_submit.gif" alt**=**"">
29. <div class**=**"box" draggable**=**"true"></div>
30. </div>
31. </div>
32. <div id**=**"target"></div>
33. <script>
34. **var** text **=** document**.**getElementById**(**'text'**);**
35. **var** target **=** document**.**getElementById**(**'target'**);**
36. text**.**ondragstart **=** **function(**event**){**
37. console**.**log**(**'元素开始被拖动'**);**
38. **}**
39. text**.**ondrag **=** **function(**e**){**
40. // console.log('一直被拖拽着');
41. **}**
42. text**.**ondragend **=** **function(**e**){**
43. console**.**log**(**'拖拽操作完成时触发'**);**
44. **}**
45. target**.**ondragenter **=** **function(**e**){**
46. console**.**log**(**'进入了所要的目标对象中'**);**
47. e**.**preventDefault**()**
48. **}**
49. target**.**ondragover **=** **function(**e**){**
50. console**.**log**(**'在目标元素内触发'**);**
51. e**.**preventDefault**();**
52. **}**
53. target**.**ondragleave **=** **function(**e**){**
54. console**.**log**(**'拖拽着元素没有放下，并离开目标对象时'**);**
55. // e.preventDefault();
56. **}**
57. target**.**ondrop **=** **function(**event**){**
58. console**.**log**(**'被拖拽元素在目标元素里放下是触发'**);**
59. event**.**preventDefault**();**
60. **}**
61. </script>
62. </body>
63. </html>

效果展示:



由上个例子我们可以看出我们确实实现了拖放的功能，猜想：能否让我们拖拽的元素放到指定的目标对象上呢？答案是可以的，如果想实现该功能，就要学一下DataTransfer对象了

**DataTransfer对象**

在进行拖放操作时，DataTransfer对象用来保存，通过拖放操作，拖动到浏览器的数据。它可以保存一项或多项数据、一种或者多种数据类型。

1. event**.**DataTransfer

**方法**

【1】DataTransfer.setData()

该方法用来设置拖动操作的当前数据

语法：

1. DataTransfer**.**setData**(**format**,**data**);**

* format 拖动数据的MIME类型，通常text/plain和text/uri-list
* data 要添加的数据

【2】DataTransfer.getData()

接收指定类型的拖放数据。如果拖放行为没有操作任何数据，则返回一个空字符串。返回值是字符串类型

语法:

1. dataTransfer**.**getData**(**format**);**

* format 拖动数据的MIME类型，通常text/plain和text/uri-list

【3】DataTransfer.clearData()

删除给定类型拖动操作的数据。

【4】DataTransfer.setDragImage()

可以使用该方法来拖拽图片

语法：

1. DataTransfer**.**setDragImage**(**img**,**xOffset**,**yOffset**)**

* img:拖拽图像的当前元素
* xOffset :图片的横向偏移量
* yOffset: 图片的纵向偏移量

**定义拖动效果**

dropEffect属性可以定义完成具体的效果

我们可以定义三种效果：

1. copy 表示拖动的数据将从其当前位置复制到放置位置。
2. move 表示拖动的数据将从其当前位置移动到放置位置。
3. link 表示将在源位置和放置位置之间创建某种形式的关系或连接。

**例子:实现复制和移动元素**

1. <!DOCTYPE html>
2. <html lang**=**"zh">
3. <head>
4. <meta charset**=**"UTF-8">
5. <meta name**=**"viewport" content**=**"width=device-width, initial-scale=1.0">
6. <meta http-equiv**=**"X-UA-Compatible" content**=**"ie=edge">
7. <title></title>
8. <style>
9. #copy,#move{
10. border**:** 1px solid #000;
11. width**:** 300px**;**
12. height**:** 200px**;**
13. **}**
14. #copyTarget,#moveTarget{
15. width**:** 300px**;**
16. height**:** 200px**;**
17. border**:** 1px solid #FF0000;
18. **}**
19. #newId{
20. width**:** 200px**;**
21. height**:** 50px**;**
22. border**:** 1px solid darkcyan**;**
23. **}**
24. </style>
25. </head>
26. <body>
27. <h2>使用拖拽实现移动和复制功能</h2>
28. <div draggable**=**"true" id**=**"copy">要复制的元素</div>
29. <div id**=**"copyTarget"></div>
30. <div draggable**=**"true" id**=**"move">要移动的元素</div>
31. <div id**=**"moveTarget"></div>
32. <script type**=**"text/javascript">
33. window**.**onload **=** **function()** **{**
34. // 复制
35. $**(**'copy'**).**ondragstart **=** handler\_dragstart**;**
36. $**(**'copy'**).**ondragend **=** handler\_dragend**;**
37. $**(**'copyTarget'**).**ondragover **=** hander\_dragover**;**
38. $**(**'copyTarget'**).**ondragleave **=** handler\_dragLeave
39. $**(**'copyTarget'**).**ondrop **=** handler\_drop**;**
40. // 移动
41. $**(**'move'**).**ondragstart **=** handler\_dragstart**;**
42. $**(**'move'**).**ondragend **=** handler\_dragend**;**
43. $**(**'moveTarget'**).**ondragover **=** hander\_dragover**;**
44. $**(**'moveTarget'**).**ondragleave **=** handler\_dragLeave
45. $**(**'moveTarget'**).**ondrop **=** handler\_drop**;**
46. **function** handler\_dragstart**(**event**)** **{**
47. console**.**log**(**'拖拽开始'**);**
48. // 设置数据
49. event**.**dataTransfer**.**setData**(**'text/plain'**,** event**.**target**.**id**);**
50. // 设置拖动效果 设置既复制又移动
51. event**.**effectAllowed **=** 'copyMove'**;**
52. **}**
53. **function** handler\_dragend**(**event**)** **{**
54. // 拖动操作完成时 清空设置的数据
55. // event.target.style.borderColor = 'black';
56. event**.**dataTransfer**.**clearData**();**
57. **}**
58. // 当被拖动元素在目标元素内时触发
59. **function** hander\_dragover**(**event**)** **{**
60. event**.**target**.**style**.**background **=** 'lightblue'**;**
61. event**.**preventDefault**();**
62. **}**
63. **function** handler\_drop**(**event**)** **{**
64. event**.**preventDefault**();**
65. // 获取设置的数据
66. **var** id **=** event**.**dataTransfer**.**getData**(**'text/plain'**);**
67. **if** **(**id **===** 'copy' **&&** event**.**target**.**id **==** 'copyTarget'**)** **{**
68. **var** nodeCopy **=** document**.**getElementById**(**id**).**cloneNode**(true);**
69. nodeCopy**.**id **=** 'newId'**;**
70. event**.**target**.**appendChild**(**nodeCopy**);**
71. **}**
72. **if(**id **===** 'move' **&&** event**.**target**.**id **==** 'moveTarget'**){**
73. event**.**target**.**appendChild**(**document**.**getElementById**(**id**));**
74. console**.**log**(**event**.**target**.**children**);**
75. **}**
76. **}**
77. **function** handler\_dragLeave**(**event**)** **{**
78. event**.**target**.**style**.**background **=** 'white'**;**
79. event**.**preventDefault**();**
80. **}**
81. **function** $**(**ele**)** **{**
82. **return** document**.**getElementById**(**ele**);**
83. **}**
84. **}**
85. </script>
86. </body>
87. </html>

效果展示：



**例子:实现简单拖拽购物车功能**

1. <html>
2. <head>
3. <meta charset**=**"utf-8" />
4. <title>使用拖放API将商品拖入购物车</title>
5. <style>
6. body **{**
7. font**-**size**:** 12px
8. **}**
9. **.**liT **{**
10. border**-**bottom**:** solid 1px #ccc;
11. background**-**color**:** #eee;
12. font**-**weight**:** bold
13. **}**
14. **.**liF **{**
15. **float:** left**;**
16. margin**-**right**:** 5px**;**
17. **}**
18. ul **{**
19. list**-**style**-**type**:** none**;**
20. padding**:** 0px**;**
21. height**:** 106px**;**
22. width**:** 330px
23. **}**
24. ul li **{**
25. overflow**:** hidden**;**
26. **}**
27. ul li img **{**
28. width**:** 68px**;**
29. height**:** 96px**;**
30. border**:** solid 1px #ccc;
31. padding**:** 3px
32. **}**
33. ul li span **{**
34. **float:** left**;**
35. width**:** 70px**;**
36. padding**:** 5px**;**
37. **}**
38. </style>
39. </head>
40. <body>
41. <ul>
42. <li class**=**"liF">
43. <img src**=**"image01.png" id**=**"img02" alt**=**"42" title**=**"2006作品" draggable**=**"true">
44. </li>
45. <li class**=**"liF">
46. <img src**=**"image02.png" id**=**"img03" alt**=**"56" title**=**"2008作品" draggable**=**"true">
47. </li>
48. <li class**=**"liF">
49. <img src**=**"image03.png" id**=**"img04" alt**=**"52" title**=**"2010作品" draggable**=**"true">
50. </li>
51. </ul>
52. <ul id**=**"ulCart">
53. <li class**=**"liT">
54. <span>书名</span>
55. <span>定价</span>
56. <span>数量</span>
57. <span>总价</span>
58. </li>
59. </ul>
60. <script type**=**"text/javascript">
61. pageload**();**
62. **function** $$**(**id**)** **{**
63. **return** document**.**getElementById**(**id**);**
64. **}**
65. //自定义页面加载时调用的函数
66. **function** pageload**()** **{**
67. //获取全部的图书商品
68. **var** Drag **=** document**.**getElementsByTagName**(**"img"**);**
69. //遍历每一个图书商品
70. **for** **(var** intI **=** 0**;** intI **<** Drag**.**length**;** intI**++)** **{**
71. console**.**log**(**Drag**[**intI**]);**
72. //为每一个商品添加被拖放元素的dragstart事件
73. Drag**[**intI**].**addEventListener**(**"dragstart"**,** **function(**e**)** **{**
74. e**.**dataTransfer**.**clearData**();**
75. **var** objDtf **=** e**.**dataTransfer**;**
76. console**.**log**(**objDtf**);**
77. objDtf**.**setData**(**"text/plain"**,** addCart**(this.**title**,** **this.**alt**,** 1**));**
78. **},true);**
79. **}**
80. **var** Cart **=** $$**(**"ulCart"**);**
81. //添加目标元素的drop事件
82. Cart**.**addEventListener**(**"drop"**,function(**e**)** **{**
83. **var** objDtf **=** e**.**dataTransfer**;**
84. **var** strHTML **=** objDtf**.**getData**(**"text/plain"**);**
85. **var** num **=** top\_**();**
86. Cart**.**innerHTML **+=** strHTML**;**
87. **var** lists **=** document**.**getElementsByClassName**(**'liC'**);**
88. **for(var** i **=** 0**;** i **<** lists**.**length**;** i**++){**
89. **var** spans **=** lists**[**i**].**children**;**
90. console**.**log**(**spans**);**
91. spans**[**2**].**innerHTML **=** num**;**
92. spans**[**3**].**innerHTML **=** num **\*** spans**[**1**].**innerHTML**;**
93. **}**
94. e**.**preventDefault**();**
95. e**.**stopPropagation**();**
96. **},false);**
97. **}**
98. //添加页面的dragover事件
99. document**.**ondragover **=** **function(**e**)** **{**
100. //阻止默认方法,取消拒绝被拖放
101. e**.**preventDefault**();**
102. **}**
103. //添加页面drop事件
104. document**.**ondrop **=** **function(**e**)** **{**
105. //阻止默认方法,取消拒绝被拖放
106. e**.**preventDefault**();**
107. **}**
108. //自定义向购物车中添加记录的函数
109. **function** addCart**(**a**,** b**,** c**)** **{**
110. **var** strHTML **=** **`<**li **class** **=** 'liC'**>**
111. **<**span**>**$**{**a**}</**span**>**
112. **<**span **class=**"price"**>**$**{**b**}</**span**>**
113. **<**span **class=**"num"**>**$**{**c**}</**span**>**
114. **<**span **class=**"sum"**>**$**{**b**\***c**}</**span**>**
115. **</**li**>`**
116. **return** strHTML**;**
117. **}**
118. //提示输入框
119. **function** top\_**()** **{**
120. **var** str **=** prompt**(**"请输入要购买的数量"**,** 1**);**
121. **return** str**;**
122. **}**
123. </script>
124. </body>
125. </html>

Css3

#### 背景属性

* + 1. background-size：指定背景图像的大小

（1）按百分比方式

（2）contain:

图片自身的宽高比不变，缩放至整个图片自身能完全显示出来，所以容器会有留白区域。

（3）cover: （实现响应式的图片）

图片宽高比不变、铺满整个容器的宽高，而图片多出的部分则会被裁掉

（1和2需要指定尺寸）

1. background-origin:指定背景图像的区域位置

默认padding-box 从padding开始显示 background-origin: padding-box;

border-box 从边框开始显示 background-origin: border-box;

content-box 从内容开始显示 background-origin: content-box;

1. background-clip: 指定绘图区域的位置

默认padding-box 从padding开始显示 background-clip: padding-box;

border-box 从边框开始显示 background- clip: border-box;

content-box 从内容开始显示 background- clip: content-box;

#### 边框

阴影box-shadow：x y blur spread color;

x:水平阴影的位置 y:垂直阴影的位置 blur模糊距离 spread阴影的大小 color阴影的颜色

边框图片

[border-image详解](https://blog.csdn.net/dengdai848586/article/details/51488700?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522160663578419195271629833%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=160663578419195271629833&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~rank_v28-10-51488700.pc_first_rank_v2_rank_v28&utm_term=border-image&spm=1018.2118.3001.4449)

文字过长

overflow: hidden;

white-space: nowrap; (不换行)

text-overflow: ellipsis;（多出部分省略号）

伪元素选择器

:before :after与::after ::before区别：

相同之处

两者都是css的伪元素，都可以在元素内容的前面添加内容

两者的写法是等价的，也就是 :before ==::before

不同之处

:befor是css2的写法，::before是css3的写法,出现在css3中是为了区分伪类选择器

:before的兼容性要比::before好 ，不过在H5开发中建议使用::before比较好

Css属性兼容性写法

-ms- 兼容ie

-webkit-兼容谷歌和safari

-moz- FF

-o- opera

#### 渐变

##### 线性渐变

[关于css3中linear-gradient中的百分比](https://www.jianshu.com/p/bf862535dd30)

linear-gradient 默认渐变方向 从上到下 改变方向需要兼容浏览器 或者换一种写法

background: linear-gradient(to left,#f00,#0f0,#00f);

background: -webkit-linear-gradient(left,#f00,#0f0,#00f);

对角写法

background: linear-gradient(to bottom right, #f00,#0f0,#00f);

渐变过渡区 红绿：40%-20% 绿蓝50%-40%

background: linear-gradient(#f00 20%,#0f0 40%,#00f 50%);

消除过渡区

background: linear-gradient(#f00 20%,#0f0 20%,#0f0 40%,#00f 40%,#00f 50%);

重复的线性渐变：用填充好的效果（即渐变过渡区域），重复填充 还没填充的地方

background: repeating-linear-gradient(#f00 20%,#0f0 40%,#00f 50%);

##### 径向渐变

默认情况下,渐变的中心是 center（表示在中心点），渐变的形状是 ellipse（表示椭圆形）

渐变的大小是 farthest-corner（表示到最远的角落）。

通过circle或ellipse来定义当前的形状

例：background:radial-gradient(center,形状 大小,开始的颜色,....,最后的颜色)

background: radial-gradient(#f00, #0f0);

重复径向渐变

background: repeating-radial-gradient(circle, #f00 20%, #0f0 50%);

#### transform2D

[transform-skew](https://www.cnblogs.com/mm2015/p/4812763.html)

[transform](https://blog.csdn.net/wang1989sg/article/details/7773661?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase)

[transform-origin](https://blog.csdn.net/qq_32013641/article/details/89070248?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.nonecase)

transform：移动、倾斜、缩放、扭曲（旋转）

transform-origin 指定原点的位置。默认情况下，它位于元素的中心

transform-origin属性值可以是百分比、em、px等具体的值，也可以是top、right、bottom、left、top left、top right、bottom left、bottom right和center这样的关键词

ransform-origin: top left;

transform 平移

transform: translate(30px, 30px);

transform旋转

transform: rotate(50deg);

transform 缩放scale(x,y) 宽放大x倍，高放大y倍

transform: scale(2,3);

transform 扭曲 y轴顺时针转为正，X轴逆时针转为正。

transform: skewX(45deg);

ransform: skewY(45deg);

transform: skew(45deg,30deg);

如果第二个参数为空，则默认为0

transform: skew(45deg);

#### transform3D

[transform3D](https://www.cnblogs.com/xiaohuochai/p/5351477.html)

1. 设置3d效果

[注意]当设置了overflow:非visible或clip:非auto时，transform-style:preserve-3d失效

transform-style: preserve-3d;

2. 提供一个透视图来配置3D空间

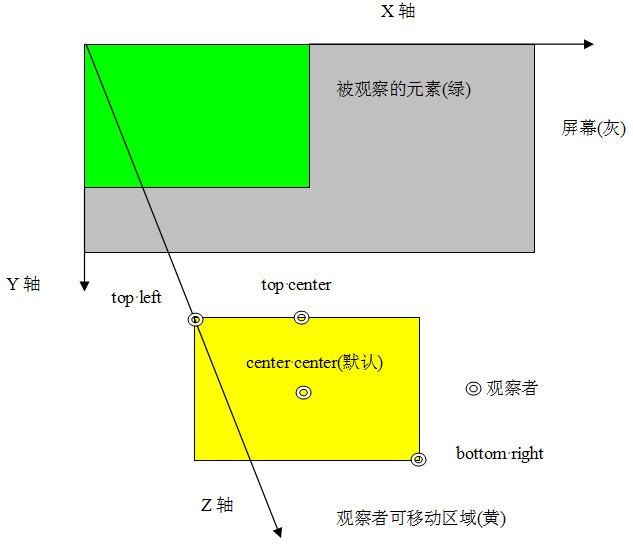
给父元素或者祖先元素设置perspective属性，此值通常在600到1000之间，能看出3D效果。perspective:1000px;

3. 改变透视来源

透视原点perspective-origin是指观察者的位置，一般地，观察者位于与屏幕平行的另一个平面上，观察者始终是与屏幕垂直的。观察者的活动区域是被观察元素的盒模型区域

透视原点 默认center center 也可以用50% 50%

perspective-origin: top left;



rotate:x为正值顺时针Y、Z为正值逆时针旋转

rotate3d(x,y,z,xdeg,ydeg,zdeg)

rotateX(45deg) 表示元素以X轴方向为轴沿顺时针旋转45角度

rotateY(45deg) 表示元素以Y轴方向为轴沿逆时针旋转45角度

rotateZ(45deg) 表示元素以Z轴方向为轴沿逆时针旋转45角度

动画：transition:过渡时间 过渡样式 过渡形式 延迟时间

[transition](https://blog.csdn.net/actionactivity/article/details/89555841?ops_request_misc=&request_id=&biz_id=102&utm_term=CSS3%20transition&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-89555841)

[Animation](https://blog.csdn.net/JingDengLai/article/details/106857823?ops_request_misc=&request_id=&biz_id=102&utm_term=html%20%E5%8A%A8%E7%94%BB&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-106857823):

动画完成次数：

Infinite:循环执行

forwards:执行一次并样式规则停留至最后一帧

数字:具体执行次数

#### transition过渡

transition-property:过渡属性

transition-duration:过渡时间

transition-timing-function:过渡函数

ease:慢-快-慢 ease-in:慢-快 ease-out:快-慢 ease-in-out: 慢-快-慢

transition-delay:过渡延时时间

transition: 过渡属性 过渡时间 过渡函数 过渡延时

#### 动画

@keyframes 设定动画

Animation:动画名 过渡时间 过渡函数 动画运行次数

移动端响应式开发

[viewport深入理解](https://www.runoob.com/w3cnote/viewport-deep-understanding.html)

##### viewport概念

通俗的讲，移动设备上的viewport就是设备的屏幕上能用来显示我们的网页的那一块区域，在具体一点，就是浏览器上(也可能是一个app中的webview)用来显示网页的那部分区域，但viewport又不局限于浏览器可视区域的大小，它可能比浏览器的可视区域要大，也可能比浏览器的可视区域要小。在默认情况下，一般来讲，移动设备上的viewport都是要大于浏览器可视区域的，这是因为考虑到移动设备的分辨率相对于桌面电脑来说都比较小，所以为了能在移动设备上正常显示那些传统的为桌面浏览器设计的网站，移动设备上的浏览器都会把自己默认的viewport设为980px或1024px（也可能是其它值，这个是由设备自己决定的），但带来的后果就是浏览器会出现横向滚动条，因为浏览器可视区域的宽度是比这个默认的viewport的宽度要小的

##### css中的1px并不等于设备的1px

在css中我们一般使用px作为单位，在桌面浏览器中css的1个像素对应着电脑屏幕的1个物理像素。

在早先的移动设备中，iphone3，分辨率为320x480，1个css像素等于一个屏幕物理像素。随着技术的发展，移动设备的屏幕像素密度越来越高，从iphone4开始，苹果公司便推出了所谓的Retina屏，分辨率提高了一倍，变成640x960，但屏幕尺寸却没变化，这就意味着同样大小的屏幕上，像素却多了一倍，这时，一个css像素是等于两个物理像素。

还有一个因素也会引起css中px的变化，那就是用户缩放。例如，当用户把页面放大一倍，那么css中1px所代表的物理像素也会增加一倍；反之把页面缩小一倍，css中1px所代表的物理像素也会减少一倍。

##### window. devicePixelRatio：

设备物理像素和设备独立像素的比例，即 devicePixelRatio = 物理像素 / 独立像素。（css中的px就可以看做是设备的独立像素）

##### PPK的关于三个viewport的理论

layout viewport：浏览器默认的viewport

layout viewport的宽度可以通过 document.documentElement.clientWidth 来获取

浏览器默认情况下把viewport设为一个较宽的值，比如980px

visual viewport：浏览器可视区域的大小

visual viewport的宽度可以通过window.innerWidth 来获取

ideal viewport：完美适配移动设备的viewport即移动设备的理想viewport。

完美适配指的是：

1. 不需要用户缩放和横向滚动条就能正常的查看网站的所有内容；
2. 显示的文字的大小是合适，比如一段14px大小的文字，不会因为在一个高密度像素的屏幕里显示得太小而无法看清，理想的情况是这段14px的文字无论是在何种密度屏幕，何种分辨率下，显示出来的大小都是差不多的。当然，不只是文字，其他元素像图片什么的也是这个道理。

##### 利用meta标签对viewport进行控制

移动设备默认的viewport是layout viewport，也就是那个比屏幕要宽的viewport，但在进行移动设备网站的开发时，我们需要的是ideal viewport。最常见的的一个动作就是把下面这个东西复制到我们的head标签中：

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

width=device-width：让当前viewport的宽度等于设备的宽度，同时不允许用户手动缩放。

initial-scale：页面的初始缩放值

minimum-scale：允许用户的最小缩放值

maximum-scale：允许用户的最大缩放值

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）

##### 把当前的viewport宽度设置为 ideal viewport 的宽度

必须把默认的layout viewport的宽度设为移动设备的屏幕宽度(两种方法)

1．width=device-width

注意：在iphone和ipad上，无论是竖屏还是横屏，宽度都是竖屏时ideal viewport的宽度

1. initial-scale=1

原因：缩放是相对于ideal viewport来进行缩放的，当对ideal viewport进行100%的缩放，也就是缩放值为1的时候，不就得到了 ideal viewport吗？

注意：windows phone 上的IE 无论是竖屏还是横屏都把宽度设为竖屏时ideal viewport的宽度。

总结：将两者都写上去，这样就 initial-scale=1 解决了 iphone、ipad的毛病，width=device-width则解决了IE的毛病

##### @media媒体类型

screen 屏幕 print 打印机

##### 媒体特性

横屏 @media all and (orientation: landscape)

竖屏 @media all and (orientation: portrait)

##### 基本尺寸

超小屏幕 手机 < 768px

小屏幕 平板 >768px

中等屏幕 桌面显示器 >992px

大屏幕 超大桌面显示器 >1200px