作者：苏梦苓  
链接：https://zhuanlan.zhihu.com/p/260177225  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

引言  
目前HTML5和CSS3在各大浏览器上的普及率已经比较高，日常开发中也是经常使用，但对于这两者的知识点一直没有做一个比较详细的整理，这里我总结了HTML5和CSS3新增的特性详解和使用问题。

本篇文章预计占用三十分钟阅读，如果你已经熟练于HTML5和CSS3，阅读本文将带你回顾其中知识点遗漏补缺。如果你还没有对HTML5和CSS3有全面的了解，本文将扩充完善你的知识体系。且面试当中问到HTML5和CSS3你都将能轻松拿下。  
如有错误,欢迎指出,将在第一时间修改,欢迎提出修改意见和建议

**HTML5  
1.了解HTML5**

HTML5属于上一代HTML的新迭代语言，设计HTML5最初目的是为了在移动设备上支持多媒体。  
例如： video 标签和 audio 及 canvas 标签  
声明必须位于HTML5文档中的第一行：

<!DOCTYPE html>

**特性：**

1. 新语义标签的引入，取消了过时的显示效果标记 <font></font> 和 <center></center>...
2. HTML多媒体元素引入（video、audio）
3. 新表单控件引入（date、time...）
4. canvas标签（图形设计）
5. 本地数据库（本地存储）
6. 对本地离线存储有更好的支持
7. 一些API（文件读取、地址位置、网络信息...）

好处：

1. **跨平台**

* 比如你开发了一款HTML5的游戏，你可以很轻易地移植到UC的开放平台、Opera的游戏中心、Facebook应用平台，甚至可以通过封装的技术发放到App Store或Google Play上，所以它的跨平台性非常强大，这也是大多数人对HTML5有兴趣的主要原因。

1. 可以很好的替代Flash
2. 可以给站点带来更多的多媒体元素(视频和音频)
3. 当涉及到网站的抓取和索引时，对于SEO很友好
4. 可移植性好

**缺点：**

1. pc端浏览器支持不是特别友好，造成用户体验不佳

2. 新语义标签  
语义标签可以清楚地向浏览器和开发者描述其意义。

<header></header> // 页眉

<nav></nav> // 导航

<section></section> // 文档的节

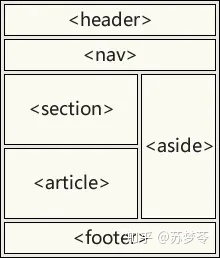
<article></article> // 文章

<aside></aside> // 侧边栏

<main></main> // 主要内容

<footer></footer> // 页脚

....



**3. 多媒体标签  
3.1 vedio**  
HTML5 规定了一种通过 video 元素来包含视频的标准方法。  
目前vedio支持三种视频格式：Ogg、MPEG 4、WebM。

**标签属性有：**

* **src**： 视频URL
* **width**： 宽
* **height**： 高
* **autoplay**： 是否自动播放
* **controls**： 是否展示控件
* **loop**： 是否循环播放
* **preload**： 是否在页面加载后载入视频
  + auto： 当页面加载后载入整个视频
  + meta： 当页面加载后只载入元数据
  + none： 当页面加载后不载入视频

☞ 多媒体标签在网页中的兼容效果方式：

<video>

<source src="trailer.mp4">

<source src="trailer.ogg">

<source src="trailer.WebM">

Your browser does not support the video tag.

</video>

3.2 audio  
audio提供了播放音频的标准。  
目前audio提供了三种音频格式：Ogg、MP3、Wav。  
标签属性有：

* **src**： 音频URL
* **autoplay**： 是否自动播放
* **loop**： 是否循环播放
* **preload**： 是否在页面加载后载入音频。
  + auto： 当页面加载后载入整个音频
  + meta： 当页面加载后只载入元数据
  + none： 当页面加载后不载入音频

音频格式兼容处理：

<audio controls="controls">

<source src="song.ogg" type="audio/ogg">

<source src="song.mp3" type="audio/mpeg">

Your browser does not support the audio tag.

</audio>

**4.新表单元素及属性  
4.1 新的输入类型**  
input标签通过**type**可以对输入类型进行限制，增加了type类型：

* email: 输入合法的邮箱地址
* url： 输入合法的网址
* number： 只能输入数字
* range： 滑块
* color： 拾色器
* date： 显示日期
* month： 显示月份
* week ： 显示第几周
* time： 显示时间

**4.2 新的表单属性**  
新的**form**属性：

* autocomplete：on｜off 自动完成输入
* novalidate： true｜false 是否关闭校验，目前支持程度非常低。

新的**input**属性：

* **autofocus**： 自动获取焦点
* **multiple**： 实现多选效果
* **required**： 必填项
* **placeholder**： 输入框内的提示
* **pattern**: 进行验证，正则表达式。
* form： 规定所属的表单，引用所属表单的id。
* list： 属性规定输入域的 datalist。

<input type="text" list="abc"/>

<datalist id="abc">

<option value="123">12312</option>

<option value="123">12312</option>

<option value="123">12312</option>

<option value="123">12312</option>

</datalist>

* min、max、step： 最小值、最大值、合法间隔

**5. 画布（canvas）**  
HTML5 的 canvas 元素使用 JavaScript 在网页上绘制图像，canvas本身是没有绘图能力的，所有的绘制工作必须在JS内部完成

**注意点：**

1. 创建画布大小使用属性方式设置（**不能通过CSS样式设置，会产生失真问题！**）
2. **解决画布重绘问题：**
   1. 设置一次描边
   2. 开启新的图层

**5.1 绘图基本方法**  
方法：

* **ctx.moveTo(x,y)**： 落笔
* **ctx.lineTo(x,y)**： 连线
* **ctx.stroke()**： 描边
* **ctx.beginPath()**： 开启新的图层
* **ctx.closePath()**： 闭合路径
* **ctx.fill()**： 填充

属性：

* **strokeStyle**： 描边颜色
* **lineWidth**： 线宽
* **lineJoin**： 线连接方式 round | bevel | miter (默认)
* **lineCap**：线帽（线两端结束的样式）butt(默认值) | round | square
* **fillStyle**： 填充颜色

画布上的X、Y轴用于对绘画进行定位：



实现一个简单的三角形：

<!--html-->

<canvas width="600" height="400"></canvas>

<!--js-->

//获取画布对象

var canvas = document.querySelector('canvas')

//获取绘图上下文

var ctx = canvas.getContext('2d')

<!--落笔-->

ctx.moveTo(100,100);

<!--连线-->

ctx.lineTo(100,200);

<!--连线-->

ctx.lineTo(200,200);

//闭合路径

ctx.closePath();

<!--描边-->

ctx.stroke();

**5.2 渐变方案**

**线性渐变**：

**ctx.createLinearGradient(x0,y0,x1,y1)**

* x0-->渐变开始的x坐标
* y0-->渐变开始的y坐标
* x1-->渐变结束的x坐标
* y1-->渐变结束的y坐标

实现一个简单的线性渐变：

grd.addColorStop(0,"black"); 设置渐变的开始颜色

grd.addColorStop(0.1,"yellow"); 设置渐变的中间颜色

grd.addColorStop(1,"red"); 设置渐变的结束颜色

ctx.strokeStyle=grd;

ctx.stroke();

addColorStop(offse,color)中渐变的开始位置和结束位置介于0-1之间，0代表开始，1代表结束。中间可以设置任何小数  
**径向渐变**：  
**ctx.createRadialGradient(x0,y0,r0,x1,y1,r1)**

* (x0,y0)：渐变的开始圆的 x,y 坐标
* r0：开始圆的半径
* (x1,y1)：渐变的结束圆的 x,y 坐标
* r1：结束圆的半径

**5.3 绘制虚线**  
**ctx.setLineDash(array)**  
原理：设置虚线其实就是设置实线与空白部分直接的距离,利用数组描述其中的关系。  
例如：

* [10,10] 实线部分10px 空白部分10px
* [10,5] 实线部分10px 空白部分5px
* [10,5,20] 实线部分10px 空白5px 实线20px 空白部分10px 实线5px 空白20px....
* [] 空数组就是实现

绘制一个虚线：

ctx.moveTo(100, 100);

ctx.lineTo(300, 100);

ctx.setLineDash([2,4]);

ctx.stroke();

**5.4 绘制矩形**  
☞ ctx.**strokeRect**(x,y,width,height) ： 绘制一个描边矩形  
☞ ctx.**fillRect**(x,y,width,height)： 绘制一个填充矩形  
☞ ctx.**clearRect**(x,y,width,height)： 清除矩形区域  
**5.5 绘制文本**

* ctx.**fillText**(文本的内容,x,y)： 绘制填充文本
* ctx.**strokeText**(文本内容,x,y)： 绘制镂空文本
* ctx.**font**="20px 微软雅黑"： 设置文字大小
* ctx.**textAlign**="left | right | center"： 文字水平对齐方式
* ctx.**textBaseline**="top | middle | bottom | alphabetic(默认)"： 文字垂直对齐方式
* 文字阴影效果：
  + ctx.**shadowColor**="red"; 设置文字阴影的颜色
  + ctx.**shadowOffsetX**=值; 设置文字阴影的水平偏移量
  + ctx.**shadowOffsetY**=值; 设置文字阴影的垂直偏移量
  + ctx.**shadowBlur**=值; 设置文字阴影的模糊度

**5.6 绘制图片**

* ctx.**drawImage**(图片对象,x,y)： 将图片绘制到画布的指定位置，宽高由**图片宽高**决定
* ctx.**drawImage**(图片对象,x,y,width,height)： x,y指的是矩形区域的位置，width和height指的是**矩形区域**的大小
* ctx.drawImage(图片对象,sx,sy,swidth,sheight,dx,dy,dwidth,dheight)：
  + sx,sy 指的是要从图片哪块区域开始绘制，swidth，sheight 是值 截取**图片区域**的大小
  + dx,dy 是指**画布中矩形区域**的位置，dwidth,dheight是值**矩形区域**的大小

**5.7 绘制圆弧**  
**ctx.arc(x,y,radius,startradian,endradian[,direct])**

* x,y： 圆心坐标
* radius： 半径
* startRadian： 开始绘制的**弧度**。度数要和**Math.P**I做转化
* endRadian： 结束绘制的**弧度**
* direct： 方向，默认顺时针（false），true代表逆时针

**5.8 平移、旋转、伸缩**

* ctx.**translate**(x,y)： 可以将**原点**的位置进行平移
* ctx.**rotate**()： 坐标进行旋转，转入值为弧度
* ctx.**scale**(x,y)： 沿着x、y进行缩放，x、y为倍数，例如：0.5、1

**6. 获取页面元素和自定义属性  
6.1 获取页面元素**  
☞ **document.querySelector("选择器")**： 通过**css选择器**选中第一个符合要求的元素  
☞ **document.querySelectorAll("选择器")**：返回所有符合选择器的元素**数组**  
☞ **Dom.classList.add("类名")**：给当前dom元素添加类样式  
☞ **Dom.classList.remove("类名")**： 给当前dom元素移除类样式  
☞ **Dom.classList.contains("类名")**：检测是否包含类样式  
☞ **Dom.classList.toggle("active")**：切换类样式（有就删除，没有就添加）  
6.2 自定义属性  
☞ 自定义属性：**data-自定义属性名**

* 获取自定义属性：Dom.dataset返回的是一个对象，Dom.dataset.属性名或者Dom.dataset[属性名]
* 设置自定义属性：Dom.dataset.自定义属性名=值 或者 Dom.dataset[自定义属性名]=值 一个例子：

<!--html-->

<div id="myData" data-source="qwer"></div>

<!--js-->

document.querySelector('# myData').dataset.source === 'qwer' // true

7 Web存储  
HTML提供了两种在**客户端**存储数据的新方法：

1. localStorage --没有时间限制的数据存储
2. sessionStorage --有时间限制的数据存储

之前都是由**cookie**完成的，但cookie不适合大量数据的存储一般只有**4KB**的空间大小，每次http请求都会携带cookie。  
7.1 localStorage  
localStorage不同点：

* 存储空间一般为**20M**
* 仅在客户端（即浏览器）中保存，不参与和服务器的通信；不会自动把数据发送给服务器，仅在本地保存
* **永久有效**
* **多窗口共享**

方法：

* **localStorage.setItem(key, value)**： 存储值
* **localStorage.getItem(key)**： 获取值
* **localStorage.removeItem("key")**：删除指定键
* **localStorage.clear()**： 清空数据

一般用来保存长久数据。  
**7.2 sessionStorage**  
sessionStorage不同点：

* 存储空间一般为**5M**
* 仅在客户端（即浏览器）中保存，不参与和服务器的通信；不会自动把数据发送给服务器，仅在本地保存
* **生命周期为关闭当前浏览器窗口**
* 可以在同一个窗口下访问

方法：

* **sessionStorage.setItem("key", "value")**： 存储
* **sessionStorage.getItem("key")**： 读取
* **sessionStorage.removeItem("key")**： 删除指定键
* **sessionStorage.clear()**： 清空数据

一般用于**一次性登录敏感数据**的存储。  
7.3 共同点  
cookie、sessionStorage、localStorage共同点：

* 都遵循**同源协议**
* 都可以被用来在浏览器存储数据，而且都是**字符串**类型的键值对

7. 应用缓存  
使用 HTML5，通过创建 **cache manifest** 文件，可以轻松地创建 web 应用的离线版本。  
如需启用应用程序缓存，请在文档的 标签中包含 manifest 属性：

<!DOCTYPE HTML>

<html manifest="demo.appcache">

...

</html>

manifest 文件可分为三个部分：

* **CACHE MANIFEST** - 在此标题下列出的文件将在首次下载后进行缓存
* **NETWORK** - 在此标题下列出的文件需要与服务器的连接，且不会被缓存
* **FALLBACK** - 在此标题下列出的文件规定当页面无法访问时的回退页面（比如 404 页面）

一个完整的Manifest文件：

CACHE MANIFEST

/theme.css

/logo.gif

/main.js

NETWORK:

login.asp

FALLBACK:

/html5/ /404.html

注：一旦文件被缓存，则浏览器会继续展示已缓存的版本，即使您修改了服务器上的文件。为了确保浏览器更新缓存，您需要更新 manifest 文件。  
8. Web Worker  
JavaScript是单线程语言，所有任务只能在一个线程上完成。随着电脑能力的增强，尤其是CPU多核，单线程无法充分发挥计算机的能力。  
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。  
8.1 Web Worker基本知识点  
在**JavaScript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。  
WebWorker有几个注意点：

1. **同源限制**。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，它所加载的脚本，必须来自网络。

8.2 主线程  
☞ **Worker()**：用来生成Worker线程。

* 第一个参数是脚本的网址（必需）var myWorker = new Worker(jsUrl, options);
* 第二个参数是可选配置对象。其中一个作用是指定Worker的名称：
  + 主线程：var myWorker = new Worker('worker.js', { name : 'myWorker' });
  + Worker线程：self.name // myWorker

Worker构造函数返回一个Worker线程对象，属性和方法如下：

* **Worker.onerror**：指定 error 事件的监听函数。
* **Worker.onmessage**：指定message事件的监听函数，发送过来的数据在**Event.data**属性中。
* **Worker.onmessageerror**：指定messageerror事件的监听函数。发送的数据无法序列化成字符串时，会触发这个事件。
* **Worker.postMessage()**：向 Worker 线程发送消息。
* **Worker.terminate()**：立即终止 Worker 线程。

8.3 Worker线程  
Web Worker **有自己的全局对象**，**不是主线程的window**，而是**一个专门为 Worker 定制的全局对象**。因此定义在**window**上面的对象和方法不是全部都可以使用。  
Worker 线程有一些自己的全局属性和方法：

* **self.name**： Worker 的名字。该属性只读，由构造函数指定。
* **self.onmessage**：指定message事件的监听函数。
* **self.onmessageerror**：指定 messageerror 事件的监听函数。发送的数据无法序列化成字符串时，会触发这个事件。
* **self.close()**：关闭 Worker 线程。
* **self.postMessage()**：向产生这个 Worker 线程发送消息。
* **self.importScripts()**：加载 JS 脚本，可以同时加载多个脚本。

9. 新增API  
HTML5还新增了多种API扩充Web功能  
9.1 获取网络状态  
☞ **window.navigator.onLine**： 返回浏览器的联网状态，正常联网（在线）返回 true，不正常联网（离线）返回 false。  
对应两个事件：

1. **window.ononline**： 当浏览器在联机和脱机模式之间切换时，会在每个页面的触发**online**事件。这些事件从document.body，到document结束于window。

* 使用window，document或document.body的addEventListener方法来进行监听

1. **window.onoffline**：在navigator.onLine 属性更改并变为 **false**时，在 **body**或冒泡到body上的offline事件被触发。

9.2 文件读取  
☞ 使用 **FileReader()** 构造器去创建一个新的 FileReader。  
接口有三个用于读取文件的方法，返回结果在**result**中：

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

☞ **FileReader** 提供的事件模型：

* onabort：中断时触发
* **onerro**r：出错时触发
* **onload**：文件读取成功完成时触发
* onloadend：读取完成触发，无论成功或失败
* onloadstart：读取开始时触发
* onprogress：读取中

一个简易的读取上传图片并渲染操作：

<!--html-->

<input type="file" name="image" >

<!--js-->

const file\_input = document.querySelector('input');

file\_input.onChange = function() {

<!--获取上传图片-->

const file = this.files[0];

<!--创建新的FileReader-->

const reader = new FileReader();

<!--将文件读取为DataUrl格式-->

reader.readAsDataURL(file);

<!--读取成功回调函数-->

reader.onload = function() {

const img = document.createElement('img');

<!--将读取结果存入img标签的src属性-->

img.src = reader.result;

document.body.appendChild(img);

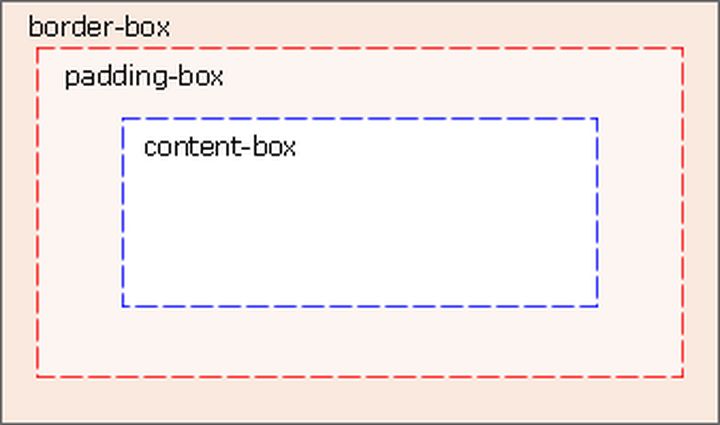
}

}

9.3 获取地理位置  
**window.navigator.geolocation.getCurrentPosition(success, error)**： 获取当前用户位置  
第一参数为获取成功时的回调函数，返回对象，包含用户地理信息：

* coords.latitude： 纬度
* coords.longitude： 经度
* ...

拿到用户经纬度即可用于一些地图操作。  
第二参数为获取用户地理位置失败或被拒绝的处理函数。  
**window.navigator.geolocation.watchPosition(success,error)**：实时获取用户的位置信息  
**clearWatch()方法**停止watchPosition方法。  
CSS3  
CSS3是CSS（层叠样式表）技术的升级版本，于1999年开始制订，2001年5月23日W3C完成了CSS3的工作草案，主要包括盒子模型、列表模块、超链接方式、语言模块、背景和边框、文字特效、多栏布局等模块。  
1. 背景  
1.1. background-origin  
**background-origin**用来规定背景图片的定位区域。  
☞ padding-box：背景图像相对内边距定位（默认值）  
☞ border-box：背景图像相对边框定位【以边框左上角为参照进行位置设置】  
☞ content-box：背景图像相对内容区域定位【以内容区域左上角为参照进行位置设置



1.2 background-clip  
background-clip规定背景的绘制区域。  
☞ border-box：背景被裁切到边框盒子位置 【将背景图片在整个容器中显示】  
☞ padding-box：背景被裁切到内边距区域【将背景图片在内边距区域（包含内容区域）显示】  
☞ content-box：背景被裁切到内容区域【将背景图片在内容区域显示】（默认值）  
1.3 background-size  
background-size规定背景图片的尺寸。  
☞ cover：背景图片按照原来的缩放比，铺满整个容器  
☞ contain：背景图片按照原来的缩放比，完整地显示到容器中(不确定是否铺满屏幕)  
2. 边框  
2.1 box-shadow

box-shadow: x y blur spread color inset

向方框添加一个或多个阴影。

* **x**：必需。水平阴影的位置。允许负值。 测试
* **y**：必需。垂直阴影的位置。允许负值。 测试
* **blur**：可选。模糊距离。 测试
* **spread**：可选。阴影的尺寸。 测试
* **color**：可选。阴影的颜色。请参阅 CSS 颜色值。 测试
* inset：可选。将外部阴影 (outset) 改为内部阴影。

2.2 border-radius  
**border-radius**：设置所有四个border-\*-radius属性的简写属性,想边框添加圆角。  
2.3 border-image  
**border-image**向边框添加图片。  
border-image 属性是一个简写属性，用于设置以下属性：

* **border-image-source**：图片路径
* border-image-slice：图片边框向内偏移。
* border-image-width：图片边框宽度
* border-image-outset：边框图像区域超出边框的量。
* **border-image-repeat**：**repeated|rounded|stretched** 图像边框是否应平铺｜铺满｜拉伸

3. 文本  
☞ **text-shadow**：x,y,blur,color 设置文本阴影  
☞ **word-wrap**：允许长单词或 URL 地址换行到下一行。。normal|break-word

* **normal**：只在允许的断字点换行（浏览器保持默认处理）。
* **break-word**： 在长单词或 URL 地址内部进行换行。

☞ **text-overflow**：规定当文本溢出包含元素时发生的事情。clip|ellipsis|string

* **clip**：修剪文本
* **ellipsis**：显示省略符号来代表被修剪的文本
* **string**：使用给定的字符串来代表被修剪的文本

实现一个单行文本省略：

.text {

overflow: hidden;

text-overflow: ellipsis;

white-space: nowrap;

}

多行文本省略：

.text {

display: -webkit-box;

-webkit-line-clamp: 2;

-webkit-box-orient: vertical；

overflow: hidden;

text-overflow: ellipsis;

}

4. 选择器  
☞ **属性选择器**：

* [属性名] {}： 带有这个属性名的所有元素
* [属性名=值] {}：[target=\_blank]，选择 target="\_blank" 的所有元素。
* [属性名^=值] {}： 以值开头
* [属性名$=值] {}： 以值结束
* [属性名\*=值] {}： 包含

例如：为 target="\_blank" 的 <a> 元素设置样式：

a[target=\_blank]

{

background-color:yellow;

}

☞ **结构伪类选择器**：

* :first-child {}： 选中的元素为父元素第一个子元素
* :last-child {}： 选中的元素为父元素中最后一个子元素
* :nth-child(n) {}： 选中的元素为父元素中正数第n个子元素
* :nth-last-child(n) {}： 选中的元素为父元素中倒数第n个子元素
  + n取值为自然数
  + odd：选中奇数树位的元素
  + even：选中偶数位的元素
  + n可以是一个表达式：an+b的格式
* :target： 选择器可用于选取当前活动的目标元素。突出显示活动的 HTML 锚

☞ **结构伪元素选择器**：

* ::first-line：选中第一行
* ::first-letter：选中第一个字符
* ::before：在元素的内容前面插入新内容
* ::after：在元素的内容后面插入新内容
* ::selection：当被鼠标选中的时候的样式
* ::placeholder：匹配占位符的文本，只有元素设置placeholder属性是，该伪元素才能生效

**伪类和伪元素**：  
css引入伪类和伪元素概念是为了格式化文档树以外的信息。  
**伪类**用于当已有元素处于的某个状态时，为其添加对应的样式，这个状态是根据用户行为而动态变化的。  
**伪元素**用于创建一些不在文档树中的元素，并为其添加样式，比如说，我们可以通过::before来在一个元素前增加一些文本。  
两者的使用区别：

1. css3中的标准规定伪类使用单冒号: ，伪元素使用双冒号::
2. 伪类可以**叠加使用**，而伪元素在一个选择器中只能出现一次，并且只能出现在**末尾**。

.box:first-child:hover { color: #000;} //使用伪类 .box::first-letter { color: #000;} //使用伪元素 .box::first-letter:hover { color: #000;} //错误写法 复制代码

1. 伪类与类优先级相同，伪元素与标签优先级相同。

5. 颜色渐变  
☞ **线性渐变**：

background-image: linear-gradient(

to right, // 渐变方向 to + right | top | bottom | left

red, // 开始渐变的颜色

blue //

);

☞**径向渐变**：

background-image: radial-gradient(

shape, // 确定圆的类型,ellipse（默认：椭圆）｜circle（圆的经向渐变）

start-color,

...,

last-color,

);

6. 2D转换  
新的转换属性**transform**：向元素应用 2D 或 3D 转换  
☞ 位移：

* **translate(x, y)**：沿着 X 和 Y 轴移动元素。
* **translateX(n)**：沿着 X 轴移动元素。
* **translateY(n)**：沿着 Y 轴移动元素。

div {

transform: translate(50px,100px);

}

☞ 旋转：**rotate(angle)**

div {

transform: rotate(60deg); // 备注：取值为角度

}

☞ 缩放：

* **scale(x,y)**： 宽、高缩放
* **scaleX(n)**
* **scaleY(n)**

备注： 取值为倍数关系，缩小大于0小于1，放大设置大于1  
☞ 倾斜：

* **skew(x-angle,y-angle)**: 沿X、Y轴倾斜方向
* **skewX(angle)**
* **skewY(angle)**

7. 3D转换  
☞ **transform-style: preserve-3d;**：将平面图形转换为立体图形  
☞ 位移：

* tranform: translate3d(x,y,z);
* transform: translateX() translateY() translateZ();

☞ 旋转：transform: rotateX(60deg) rotateY(60deg) rotateZ(60deg);  
☞ 缩放：

* tranform: scale3d(x,y,z);
* transform: scaleX(0.5) scaleY(1) scaleZ(1);

8. 过渡  
☞ 设置哪些属性要参与到过渡动画效果中：**transition-property**: all;  
☞ 设置过渡执行时间：**transition-duration**: 1s;  
☞ 设置过渡的速度类型：**transition-timing-function**:

* **linear**：以相同速度开始至结束的过渡效果
* **ease**：慢速开始，然后变快，然后慢速结束
* ease-in：以慢速开始
* ease-out：以慢速结束
* ease-in-out：以慢速开始和结束

☞ 设置过渡延时执行时间：**transition-delay**: 1s;  
☞ 简写模式：**transition: property duration timing-function delay;**  
9. 动画  
☞ 定义动画集:

@keyframes rotate {

/\* 定义开始状态 0%\*/

from {

transform: rotateZ(0deg);

}

/\* 结束状态 100%\*/

to {

transform: rotateZ(360deg);

}

}

注意：如果设置动画集使用的是百分比，那么记住百分比是相对整个动画执行时间的。  
☞ 调用动画集：

* **animation-name**: 调用动画集名称
* **animation-duration**：完成一个周期所花费的时间
* **animation-timing-function**：动画执行的速度类型，linear|ease|ease-in|ease-out|ease-in-out
* **animation-delay**：动画延时
* **animation-iteration-count**：动画被播放的次数，**infinite**无限执行
* **animation-direction**：是否应该轮流反向播放动画
  + normal（默认值，动画正常播放）
  + **alternative**（动画应该轮流反向播放）;
* **animation-fill-mode**：设置动画填充模式，规定动画在播放之前或之后，其动画效果是否可见
  + **none**：不改变默认行为
  + **forwards**：当动画完成后，保持最后一个属性值（在最后一个关键帧中定义）
  + **backwards**：在 animation-delay 所指定的一段时间内，在动画显示之前，应用开始属性值（在第一个关键帧中定义）
  + **both**：向前和向后填充模式都被应用。
* **animation-play-state**：动画暂停paused | running;

animation简写模式：**animation: name duration timing-function delay iteration-count direction;**  
10. 弹性布局（flex）  
传统的布局方式对于特殊的布局非常不方便，比如垂直居中。**Flex**是2009年W3C提出的新方案--Flex布局，可以简便、完整、响应式地实现各种页面布局。  
10.1 容器属性  
☞ 设置父元素为伸缩盒子【直接父元素】

display: flex;

注：设为 Flex 布局以后，子元素的**float**、**clear**和**vertical-align**属性将失效。  
用于容器上的属性：

* flex-direction
* flex-wrap
* justify-content
* align-items
* align-content
* flex-flow

10.1.1 flex-direction  
☞ 设置伸缩盒子主轴方向

flex-direction: row; // 默认值，主轴为水平方向，起点在左侧

flex-direction: row-reverse; // 主轴为水平方向，起点在右侧

flex-direction: column; // 主轴为垂直方向，起点在上沿

flex-direction: column-reverse; // 主轴为垂直方向，起点在下沿

**10.1.2 flex-wrap**

☞ 设置元素是否换行显示

在伸缩盒子中所有的元素默认都会在一条线上显示

如果希望换行：

flex-wrap: wrap | nowrap | wrap-reverse;

* nowrap：默认，不换行
* wrap：换行，第一行在上方
* wrap-reverse：换行，第一行在下方

**10.1.3 justify-content**

☞ 设置元素在主轴的对齐方式

justify-content: flex-start; // 左对齐

justify-content: flex-end; // 右对齐

justify-content: center; // 居中

justify-content: space-between; // 两端对齐，项目之间的间隔相等

justify-content: space-around; // 每个项目两侧之间的间隔相等

**10.1.4 align-items**

☞ 设置元素在侧轴的对齐方式

align-items: flex-start; // 交叉轴的起点对齐。

align-items: flex-end; // 交叉轴的终点对齐。

align-items: center; // 交叉轴的中点对齐。

align-items: 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。

align-items: baseline; // 项目的第一行文字的基线对齐。

**10.1.5 align-content**

☞ 设置元素换行后的对齐方式,定义了多根轴线的对齐方式。

align-content: flex-start;

align-content: flex-end;

align-content: center;

align-content: space-around;

align-content: space-between;

/\* 换行后的默认值 \*/

align-content: stretch;

**10.1.6 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>;

}

**10.2 项目属性**

* order
* flex-grow
* flex-shrink
* flex-basis
* flex
* align-self

**10.2.1 order**

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>;

}

10.2.2 flex-grow  
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。

* 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

10.2.3 flex-shrink  
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。

* 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

10.2.4 flex-basis  
flex-basis属性定义了在分配多余空间之前，项目占据的主轴空间。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 \*/

}

**10.2.5 flex**

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

}

**10.2.6 align-self**

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;

}