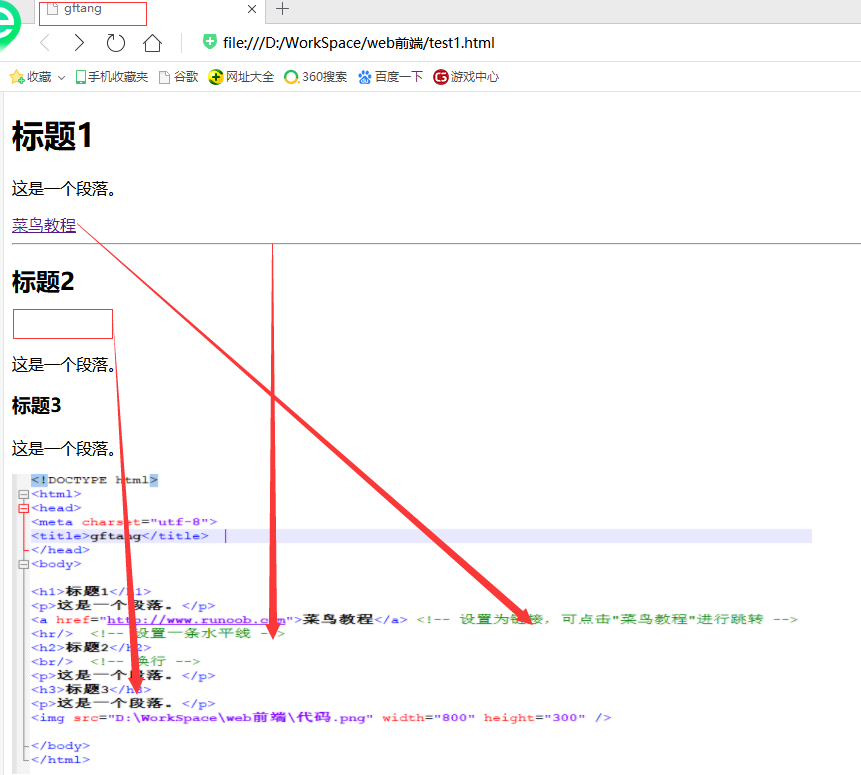
# html基础 h,p,a,img,ul,il,div,span



注1：**空元素：**即没有内容的 HTML 元素，需要在开始标签中关闭，比如换行**<br/>**

注2：标签小写，且所有标签都要闭合

## 标签和属性

HTML 链接由 <a> 标签定义。链接的地址在 **href 属性**中指定：

<a href="http://www.runoob.com">这是一个链接</a>

1. 属性值一般在双引号中，如果属性值本身就含有双引号，那么您必须使用单引号，例如：name='John "ShotGun" Nelson'
2. 属性小写
3. 适用于大多数HTML元素的属性有：
4. class：为html元素定义一个或多个**类名**（classname）(类名从样式文件引入)，class=" " （引号里面可以填入多个class属性）
5. id：定义元素的唯一ID，id=" "（只能填写一个，多个无效）
6. style：规定元素的行内样式
7. title：描述了元素的额外信息（作为工具条使用）

## 标题

标题（Heading）是通过 <h1> - <h6> 标签进行定义的。1-6依次变小

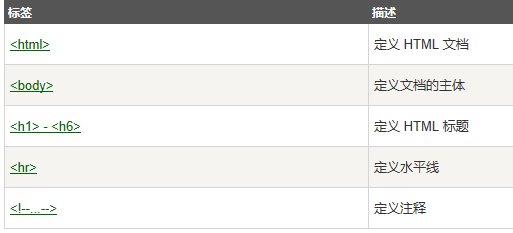
注：浏览器会自动地在标题的前后添加空行。

请确保将 HTML 标题 标签**只用于标题**。不要仅仅是为了生成**粗体**或**大号**的文本而使用标题。

搜索引擎使用标题为您的网页的结构和内容编制索引。

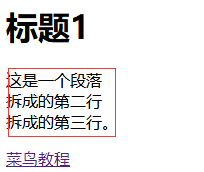
因为用户可以通过标题来快速浏览您的网页，所以用标题来呈现文档结构是很重要的。

应该将 h1 用作主标题（最重要的），其后是 h2（次重要的），再其次是 h3，以此类推。



## 段落

<p>这是一个段落<br>被拆成的第二行<br>被拆成的第三行。</p>



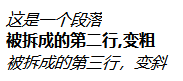
1. 我们无法确定 HTML 被显示的确切效果。屏幕的大小，以及对窗口的调整都可能导致不同的结果。

2．对于 HTML，无法通过在 HTML 代码中添加额外的空格或换行来改变输出的效果。

3. 当显示页面时，浏览器会移除源代码中多余的空格和空行。所有连续的空格或空行都会被算作一个空格。需要注意的是，HTML 代码中的所有连续的空行（换行）也被显示为一个空格。

## 文本格式化

<p><em>这是一个段落,着重</em><br><b>被拆成的第二行,变粗</b><br><i>被拆成的第三行，变斜</i></p>



<b>和**<strong>** 加粗

<i>和**<em>** 斜体

<big>和<small> 字体变大变小



<sub>和<sup> 生成下标和上标

<del>和<ins> 删除和插入 例<p>My favorite color is <del>blue</del> <ins>red</ins>!</p>

## 链接 <a href=>

1. <a href="<http://www.runoob.com>" target="\_blank">菜鸟教程</a>

点击 菜鸟教程 可跳转该页面，加入target属性，会另外打开一个窗口进行跳转

1. 用图片做链接：

<a href="//www.runoob.com/html/html-tutorial.html">

**<img border="10" src="smiley.gif" alt="HTML 教程" width="32" height="32">**</a></p>

1. 在当前页面跳转

首先<h3><a id="C3">标题3</a></h3> <!—定义一个id属性 -->

然后<p><a href="**#**C3">点击跳转到标题3</a></p>

1. 跳出框架之外，跳到其他网站？

使用属性target="\_top"

1. 发送电子邮件

<a href="mailto:982541258@qq.com?Subject=Hello%20world%20%20！" target="\_top">

发送内容是Subject=之后的内容，空格用%20

<a href="mailto:someone@example.com**?**cc=someoneelse@example.com**&**bcc=andsomeoneelse@example.com**&**subject=Summer%20Party**&**body=You%20are%20invited%20to%20a%20big%20summer%20party!" target="\_top">发送邮件!</a>

注：cc=”抄送人”；dcc=”密件抄送人”；subject=”标题”；body=”正文”

## 头部

<head> 元素包含了所有的头部标签元素。在 <head>元素中你可以插入脚本（scripts）, 样式文件（CSS），及各种meta信息。

可以添加在头部区域的元素标签为: <title>, <style>, <meta>, <link>, <script>, <noscript>, and <base>.



1. <title>：定义不同文档的标题，必须要的

<title>元素： 定义了浏览器工具栏的标题

当网页添加到收藏夹时，显示在收藏夹中的标题

显示在搜索引擎结果页面的标题

1. <base>: 标签描述了基本的链接地址/链 接目标，该标签作为HTML文档中所有的链接标签的默认链接: <base href="http://www.runoob.com/images/" target="\_blank">
2. <link> 标签定义了文档与外部资源之间的关系。常用于链表到样式表

<link rel="stylesheet" type="text/css" href="mystyle.css">

1. <style> 标签定义了HTML文档的样式文件引用地址，可以直接添加样式渲染文档

<style type="text/css"> body {background-color:yellow} p {color:blue} </style> <!--渲染body的背景颜色为黄色，正文字体为蓝色-->

1. <meta> 元数据，不显示在页面上，但会被浏览器解析，通常用于指定网页的描述，关键词，文件的最后修改时间，作者，和其他元数据。
2. 给搜索引擎定义关键字：<meta name="keywords" content="HTML, CSS, XML, XHTML, JavaScript">
3. 为网页定义描述内容：<meta name="description" content="免费 Web & 编程 教程">
4. 定义网页作者：<meta name="author" content="Runoob">
5. 每30秒刷新当前页面：<meta http-equiv="refresh" content="30">

## 图片和图片集 img map area

### 插入图片

<img src="smiley.gif" alt="Smiley face" width="32" height="32">

<img src="smiley.gif" title="Smiley face" width="32" height="32">

注：alt属性在图片未加载出来的时候显示，title属性也行，而且在图片能加载出来时，鼠标放在图片上仍然显示

### 插入图片集

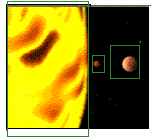
<map> & <area>



<map> 标签用于客户端图像映射。图像映射指带有可点击区域的一幅图像。

<img>中的 **usemap** 属性可引用 <map> 中的 id 或 name 属性（取决于浏览器），所以我们**应同时向 <map> 添加 id 和 name 属性**。

area 元素永远嵌套在 **map 元素内部**。area 元素可定义图像映射中的区域。



注1：coords的坐标，方形为x1,y1,x2,y2;圆形为(园心x,y,半径)

## 表格 tr th td

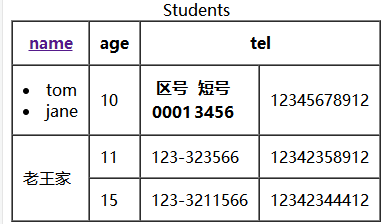




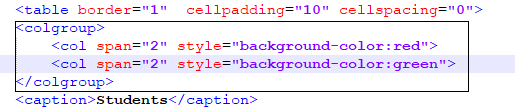
表格由 <table> 标签来定义。一行一个<tr>，一列一个<td>，标题用<th>

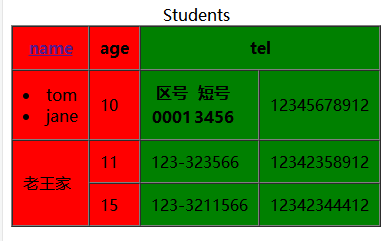
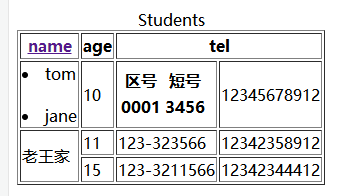
1. 无边框：去掉border=”1”
2. 表格内表头：水平标题，一个tr写th，一个tr写td；垂直标题，th，td在一个tr中
3. 表格上加标题：<caption>Students</caption>
4. 单行表格合并 <th colspan=”2”>tel</th>
5. 单列表格合并 <td rowspan="2">老王家</td>，下一个tr中不写name就行
6. 表格内可以加表格，加列表<li>，加段落，加链接等
7. 单元格边距和间距<table border="1" cellpadding="10" cellspacing="0">





1. 调背景色



## 列表 <ol> <ul> <dl> <dt> <dd>

1. 无序列表：

<ul>

<li>Coffee</li>

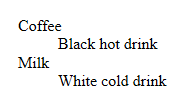
<li>Tea</li>

<li>Milk</li>

</ul>

1. 有序列表：<ol start=”10”> 可以从10开始计数
2. 自定义列表

<dl>



<dt>Coffee</dt>

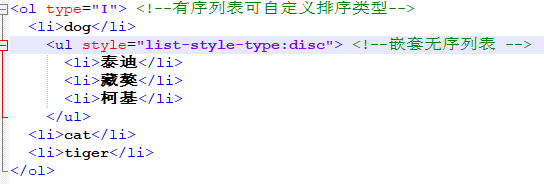
<dd>Black hot drink</dd>

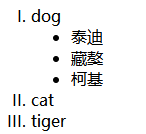
<dt>Milk</dt>

<dd>White cold drink</dd>

</dl>

1. 嵌套：往里套就完事儿了





## 区块 <div> 和 <span> 容器

<div>：定义了文档的区域，块级。 <span>：用来组合文档中的行内元素，内联元素。

囊括一个范围，可以改变其中字体的颜色

span：可以在<p>中定义文本字体颜色





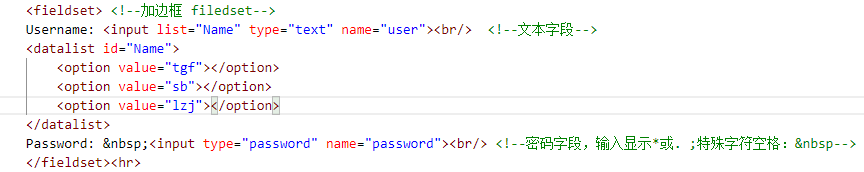
## <div> & <table>

目前都是<div> + CSS

## 重要：表单 <form> <input>

### 文本密码字段 text password fieldset (datalist option)

若要把它设置为必填项，可加<input type=”text” required> IE9之前不可用





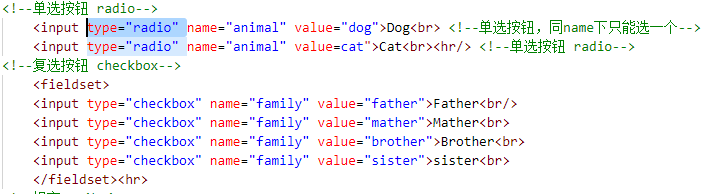
注1：在form中可以用fieldset给表单加边框

注2：可以预先指定输入参数来做选择datalist，需要在input中加入list元素



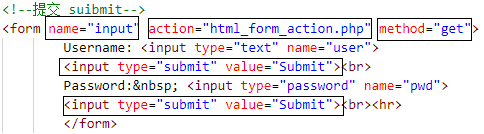
注3：为了表单框对其，加入特殊字符：空格(&nbsp)

### 单选按钮(type=”radio”)、复选按钮(type=”checkbox”)



注：单选、复选中的**value**是提交数据到服务器的值；**name**：为控件命名，以备后台程序 ASP、PHP 使用；**checked**：当设置 checked="checked" 时，该选项被默认选中

### 提交按钮 submint，重置按钮reset





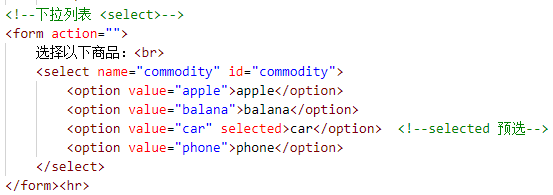
注1： 点击确认按钮，那么输入数据会传送到 "html\_form\_action.php" 的页面。该页面将显示出输入的结果。

注2： input中type=”submit”定义提交按钮，value定义按钮上的内容

注3：<input type="reset" name="button" id="button" value="重置"> 重置恢复默认值

### 下拉列表 select option





### 文本域 testarea

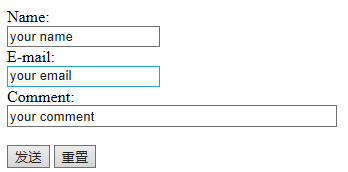




注1：可写入字数不受限制，框大小可调。resize: none;该属性可以固定文本域的大小。

### 发送邮件显示表单内容





### label标签：

label标签为input元素添加标注，label[for=]=input[id=]进行链接

作用：在点击标注字体时也可被选中。

<label for="male">Male</label>

<input type="radio" name="sex" id="male" />

### 表单重置

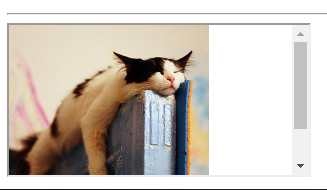
document.getElementById("Form\_id").reset()

document.getElementById("input\_id").value = ""

## 框架内页面插入 iframe

可以在同一个浏览器中显示不止一个页面

<iframe src="D:\WorkSpace\web前端\itsmy.png" height="400xp" width="400xp"</iframe>



消除滚动条：

document.getElementById("content").style.height =

window.innerHeight + ‘px’

注1：html5只支持src

注2：一些旧浏览器不支持，不支持则会不显示

优点：

1. iframe能够原封不动的把嵌入的网页展现出来。
2. 如果有多个网页引用iframe，那么你只需要修改iframe的内容，就可以实现调用的每一个页面内容的更改，方便快捷。
3. 网页如果为了统一风格，头部和版本都是一样的，就可以写成一个页面，用iframe来嵌套，可以增加代码的可重用。
4. 如果遇到加载缓慢的第三方内容如图标和广告，这些问题可以由iframe来解决。

缺点：

搜索引擎的爬虫程序无法解读这种页面

框架结构中出现各种滚动条(可以通过修改window尺寸解决)

使用框架结构时，保证设置正确的导航链接。

iframe页面会增加服务器的http请求

会阻塞window.onload事件加载，导致页面慢等

## 插入脚本

<script>document.write("Hello World!")</script> 将打印出hello world

## 特殊字符

|  |  |  |  |
| --- | --- | --- | --- |
| 显示结果 | 描述 | 实体名称 | 实体编号 |
|  | 空格 | &nbsp; | &#160; |
| < | 小于号 | &lt; | &#60; |
| > | 大于号 | &gt; | &#62; |
| & | 和号 | &amp; | &#38; |
| “” | 引号 | &puot; | &#34; |

## 重要：html5新特性

### input新属性 type=" "

1. email 提交表单会自动验证email域的值
2. url: 输入链接
3. number规定只能输入数字，可以规定大小限制

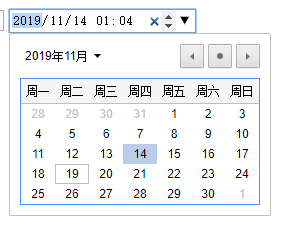
<input type="number" name="points" min="1" max="10" />

1. range：通过滑动条选定一定范围的某个数字

<input type="range" name="points" min="1" max="10" />



1. Date
   1. date: <input type="date" name="user\_date" /> 选取日月年
   2. month：选取月、年
   3. week:选取周和年
   4. time：选取时间（小时和分钟）
   5. datetime{-local}选取时间



### 表单新元素

1. datalist 配合input:url选择url



### canvas画布

<canvas> 标签通过脚本（通常是 JavaScript）来绘制图形（比如图表和其他图像）。

<canvas> 标签只是**图形容器**，必须使用脚本来绘制图形。

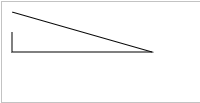




注：canvas可以自定义画布的尺寸

使用js来绘制：

1. 绘制线条

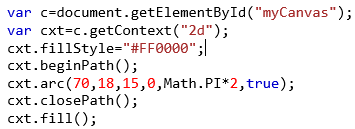




moveTo()规定起始点坐标

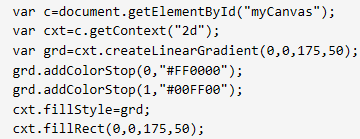
lineTo()用线连接到(150,50)，该坐标变成新起点。stroke()结束

1. 绘制圆形(规定颜色、尺寸和位置)

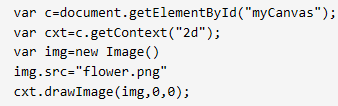


1. 渐变



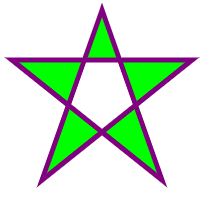


1. 图像（把图片放置到画布上）

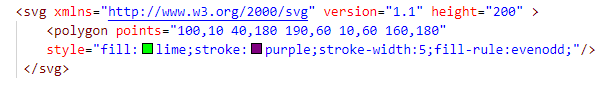


### 内联svg

* SVG 指**可伸缩矢量图形** (Scalable Vector Graphics)，即放大缩小尺寸不会改变其质量
* SVG 用于定义用于网络的基于矢量的图形

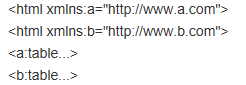


* SVG 使用 XML 格式定义图形
* SVG 是万维网联盟的标准



注1：xml namespace的缩写，也就是XML命名空间，可以在文档中定义一个或多个可供选择的命名空间。该属性可以放置在文档内任何元素的开始标签中。该属性的值类似于URL，它定义了一个命名空间，浏览器会将此命名空间用于该属性所在元素内的所有内容。

a.dtd中和b.dtd中都包含了table，但是如果一个xhtml文件同时包含了这两个dtd文件，而且使用了标志table，就会出现标志冲突问题



注2：fill 设置填充颜色；stroke 设置描边颜色；stroke-width 设置线条宽度；fill-rule 设置填充规则(nonzero(默认) | evenodd | inherit)

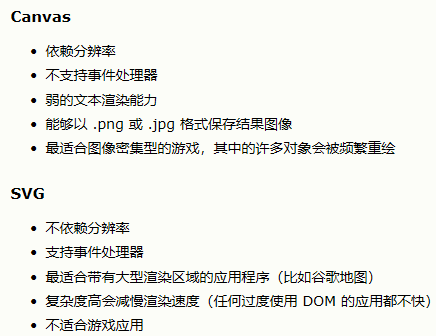
### SVG和canvas的异同

SVG：

1. SVG 是一种使用 XML 描述 2D 图形的语言。
2. SVG 基于 XML，这意味着 SVG DOM 中的每个元素都是可用的，可以为某个元素附加 JavaScript 事件处理器。
3. 在 SVG 中，每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化，那么浏览器能够自动重现图形。

canvas：

1. canvas通过js绘制2D图形
2. 是逐像素渲染
3. 一旦图形被绘制完成，就不会受到浏览器关注。位置发生变化，整个场景也需要重新绘制，包括任何或已被覆盖的对象



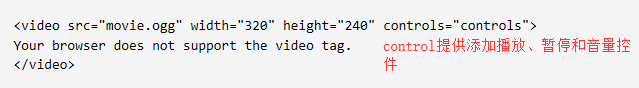
### 视频 video

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **格式** | **IE** | **Firefox** | **Opera** | **Chrome** | **Safari** |
| Ogg | No | 3.5+ | 10.5+ | 5.0+ | No |
| MPEG 4 | 9.0+ | No | No | 5.0+ | 3.0+ |
| WebM | No | 4.0+ | 10.6+ | 6.0+ | No |

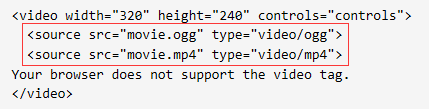
**Ogg** = 带有 Theora 视频编码和 Vorbis 音频编码的 Ogg 文件

**MPEG4** = 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



video元素允许多个**source元素** ，该元素可以链接不同的视频文件，浏览器使用第一个可识别的文件进行播放：



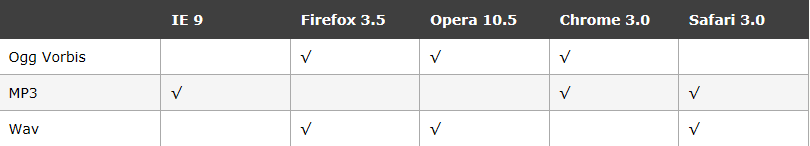
注：其他属性：1）autoplay和preload 前一个出现会在视频就绪马上播放，后一个会进行加载，并预备播放；2）loop属性：若有该属性，视频结束完之后会自动重复播放。

通过DOM修改视频大小

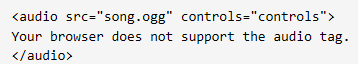
document.getElementById("rideo").width = "460"

### 音频 audio

支持格式：



使用，同视频：



同样有source元素。source 元素可以链接不同的音频文件。浏览器将使用第一个可识别的格式；标签属性同video.

### 拖放 drag & drop

拖放即抓取对象以后拖到另一个位置。HTML5中任何元素都能拖放。

**Internet Explorer 9、**Firefox、Opera 12、Chrome 以及 Safari 5 支持拖放。

<img id="drag1" src="img\_logo.gif" **draggable**="true" **ondragstart**="drag(event)" width="336" height="69">

1. 设置为可拖放<img draggable=”true”>
2. 拖放内容 ondragstart & setData()

ondragstart：规定拖动什么数据

function drag(ev) {

ev.dataTransfer.setData("text", ev.target.id);}

其中dataTransfer.setData() 方法设置被拖放的数据类型是text;值是可拖动元素的id("drag")

<div id="div1" **ondrop**="drop(event)" **ondragover**="allowDrop(event)"></div>

设置拖动位置：

1. 拖到何处：ondragover

function allowDrop(ev) {

ev.preventDefault();}

规定被拖动的数据能够放置到何处；因为默认是无法被放置到其他元素；这里通过preventDefault()方法阻止这种默认处理方式。

1. 进行放置：ondrop 放开鼠标发生的事件

function drop(ev) {

ev.preventDefault();

var data = ev.dataTransfer.getData("text");

ev.target.appendChild(document.getElementById(data));

}

调用 preventDefault() 来阻止数据的浏览器默认处理方式（drop 事件的默认行为是以链接形式打开）

通过 dataTransfer.getData() 方法获得被拖的数据。该方法将返回在 setData() 方法中设置为相同类型的任何数据

被拖数据是被拖元素的 id ("drag1")

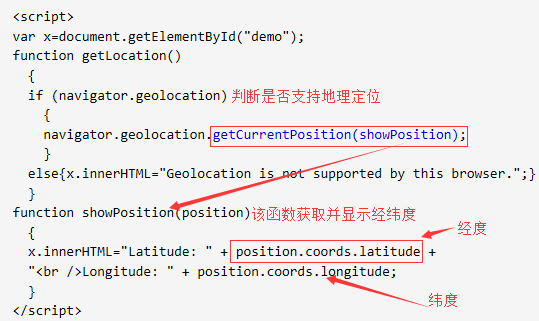
把被拖元素追加到放置元素中

### 地理定位

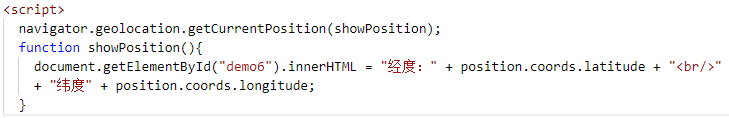
HTML5 Geolocation API 用于获得用户的地理位置

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

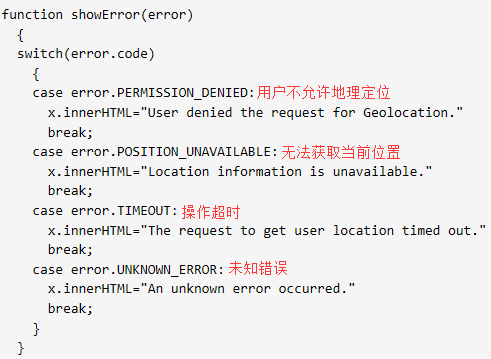
通过**getCurrentPosition()** 方法来获得用户的位置



简化：



错误处理：



### web存储数据

HTML5提供了两种在客户端存储数据的方法：

1. localStorage 没有时间限制的数据存储；优于cookie(会话存储)
2. sessionStorage 针对session的数据存储; 关闭浏览器标签页时丢失。

这两种方法可以在不影响网站性能的情况下存储大量数据；

对于不同的网站，数据会存储在不同的区域，并且每个网站只能返回该网站自身的数据；

HTML5中可以用js来存储和访问数据

与cookie的差别： 该存储内容不会被服务端读取；所以请求体中不会被携带；localStorage中数据永久保存；cookie内存4k; localStorage内存5M

使用：1. 缓存对象以提升用户下次打开应用时的加载速度、保存正在撰写的文件草稿，甚至还可以把它当成虚拟的内存容器来使用；2. 假如存储的数据量超过了5MB，那浏览器就会抛出QUOTA\_EXCEENED\_ERR的异常；所以使用时一定要加try…catch…进行异常捕捉。

localStorage对象设置：

if (typeof(Storage) !== "undefined") {

**localStorage.setItem("lastname", "Gates");** // 进行键值对设置

} else {

document.getElementById("result").innerHTML = "抱歉！您的浏览器不支持 Web Storage ...";

}

x = document.getElementById("result")

取值：x.innerHTML = localStorage.lastname;

删除：localStorage.removeItem("lastname");

清空：localStorage.clear()

sessionStorage对象存储方式和localStorage相同。

具体应用：

// 监听textarea变化

$("textarea").change(function(){

    var wikiContent = {'content' : $('#content').val()};

    // 存储值：将对象转换为Json字符串。如果有多条数据的话，key 最好做一下区分

    localStorage.setItem("wikiContent", JSON.stringify(wikiContent));

});

var wikiJsonStr = localStorage.getItem("wikiContent");

// 取值时：把获取到的Json字符串转换回对象

wikiLocalContent = JSON.parse(wikiJsonStr);

// 获取对应的值

console.log(wikiLocalContent.content);

API:

setItem (key, value) —— 保存数据，以键值对的方式储存信息。

getItem (key) —— 获取数据，将键值传入，即可获取到对应的value值。

removeItem (key) —— 删除单个数据，根据键值移除对应的信息。

clear () —— 删除所有的数据

key (index) —— 获取某个索引的key

### 应用缓存

应用优势：

1. 离线预览：用户可在离线时访问应用；
2. 速度：已缓存的资源加载速度快；
3. 减少服务器负载：浏览器只需要从服务器下载更新过的资源

注：IE不支持

使用：<html **manifest="demo.appcache"**>

通过manifest属性指定；

**manifest 文件:**

它的建议的文件扩展名是：".appcache"。告知浏览器被缓存的内容。

可分为三部分：

1. **CACHE MANIFEST**（cache[[kæʃ]] manifest）此标题下列出的文件将在首次下载后进行缓存：

CACHE MANIFEST

/theme.css

/logo.gif

/main.js

1. **NETWORK** 标题下列出的文件需要与服务器的连接，且不会被缓存

NETWORK:

login.asp // \* 表示所有资源不被缓存

1. **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）

FALLBACK:

/html5/ /404.html

注：第一个是实际加载资源，第二个是备用页面，即当/html5/无法通过因特网建立连接，就展示备用页面。

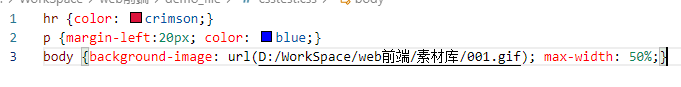
1. 更新缓存：1. 用户清空浏览器缓存；2. manifest文件被修改；3. 由程序来更新应用缓存。
2. 浏览器对缓存数据的容量限制可能不太一样（某些浏览器设置的限制是每个站点 5MB

# 二. css基础

## 1. 创建和基础语法

### 外部样式表、内部样式表、内联样式

1. 外部样式表



新建文件填写内容，用.css后缀表示，在html的<head>中用<link>标签链接样式表

例：<link rel="stylesheet" href="D:\WorkSpace\web前端\demo\_file\csstest.css" type="text/css">

1. 内部样式表



1. 内联样式：在标签中设置属性style的元素值

**注：**可以设置多重，优先级为

**内联样式）Inline style > （内部样式）Internal style sheet >（外部样式）External style sheet > 浏览器默认样式**

### 基础语法





## 背景(background-)与透明度

background-(color、image、repeat、attachment、position)

1. color

1) 十六进制 - 如："#ff0000" 2) RGB - 如："rgb(255,0,0)" 3) 颜色名称 - 如："red"

1. image

body {background-image: url(D:/WorkSpace/web前端/素材库/001.gif);}

注1图片链接可以是本地也可以是网络链接，本地链接尽量用相对路径，且路径分割符为“/”

注2：图片默认水平和垂直平铺(满页面都是)

1. repeat(解决平铺问题) 英文意思：重复
2. 水平平铺：background-repeat:repeat-x;
3. 不平铺并设置定位：background-repeat:no-repeat;

background-position:right top;

注：**vw(视窗宽度百分比)**、**vh(高度百分比)**、**vmin(当前vw与vh中较小者)**、**vmax**

1. attachment(背景固定)

background-attachment:{fixed、scroll、local、initial、inherit};

**scroll(**默认)：背景图片随着页面滚动而滚动(像是背景和前面内容粘在一起了)

**fixed**：不随页面页面滚动

clocal：随着元素内容滚动而滚动 initial：设置该属性默认值

**透明度：**

1. css3的[opacity:x](http://www.php100.com/manual/css3_0/opacity.shtml) [oʊˈpæsəti]，x 的取值从 0 到 1，如opacity: 0.8 IE9+
2. css3的rgba(red, green, blue, alpha)，alpha的取值从 0 到 1，如rgba(255,255,255,0.8)

同只兼容IE9+；让IE8支持的的方法：

background:rgba(0,0,0,0.5);

filter: progid:DXImageTransform.Microsoft.gradient(startcolorstr=#7F000000,endcolorstr=#7F000000);

1. IE专属滤镜 filter:Alpha(opacity=x)，x 的取值从 0 到 100，如filter:Alpha(opacity=80)

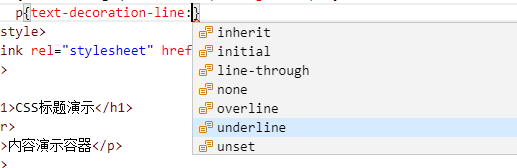
filter:alpha(opacity:50); opacity:0.5; -moz-opacity:0.5;-khtml-opacity: 0.5;

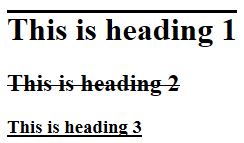
## 文本和字体 text-align text-decoration text-transform text-shadow

文本颜色用color=””,三种表示方式

**文本对齐方式：**text-align 可选：center、right、**justify**

**文本修饰：**text-decoration 属性用来设置或删除文本的装饰。





**文本转换**：text-transform：可用于所有字句变成大写uppercase或小写字母lowercase，或每个单词的首字母大写capitalize

**文本缩进**：text-indent:50px;

其他：[letter-spacing](https://www.runoob.com/cssref/pr-text-letter-spacing.html)：设置字符间距；[line-height](https://www.runoob.com/cssref/pr-dim-line-height.html)：设置行高；

text-shadow:2px 2px darkgreen;**设置文本阴影**(用在标题很有立体效果，IE9及以前不支持)；

white-space: nowrap;去除文本的换行符，变成一行(像csdn里写代码的部分)

text-overflow: ellipsis; 裁剪超出部分，并用…代替

两个配合使用

**字体设置**：p{**font-family**:"Times New Roman", Times, serif;}

字体样式：font-style:normal; italic(斜体);

字体大小：font-size:30px; 30em; 百分比

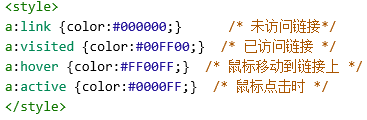
字体粗细：[font-weight](https://www.runoob.com/cssref/pr-font-weight.html)

**注：**在所有浏览器的解决方案中，设置 <body>元素的默认字体大小的是百分比

## 链接和伪类 lvha

伪类不需要标签里面定义，可直接使用，用：链接

anchor伪类



注：必须按以上顺序排列

:visited要写在:hover和:avtive之前，否则会覆盖:hover和:avtive

## 伪类和伪元素

伪类的语法：selector:pseudo-class {property:value;}

css中伪类的语法：selector.class:pseudo-class {property:value;}

1. 伪类和css类配合

a.red:visited {color:#FF0000;} /\*red类的链接访问后显示红色\*/

1. first-child伪类

当第一个是<p>元素时，第一个<p>或者第一个<p>元素中的<i>元素字体改变

p:first-child { color:blue; } /\*第一个<p>的内容字体为蓝色\*/

p>i:first-child { color:blue; }/\*第一个<p>中的<i>元素内容字体为蓝色\*/

匹配第一个是<p>元素时，里面所有<i>元素字体改变：

pi:first-child i { color:blue; }

1. :lang 伪类使你有能力为不同的语言定义特殊的规则

:lang 类为属性值为 no 的q元素定义引号的类型(不知道咋用，先记着)

q:lang(no) {quotes: "~" "~";}

1. 附件：[所有伪类伪元素表](#_所有伪类/元素)
2. 伪元素 伪元素一般采用双冒号的写法

例1： :first-line p:first-line { color:#ff0000; font-variant:small-caps; }

对P元素中第一行内容进行格式化

**注：**"first-line" 伪元素只能用于**块级元素**。

例2： :first-letter p::first-letter { color:#ff0000; font-size:xx-large; }

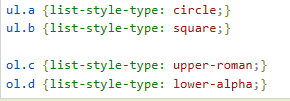
例3：p::before { content:"台词："; background-color:yellow; color:red; }

在p标签内容前加上 “台词：”

**注：伪类**选择元素基于的是当前元素处于的状态，或者说元素当前所具有的特性，而不是元素的id、class、属性等静态的标志。由于状态是动态变化的，所以一个元素达到一个特定状态时，它可能得到一个伪类的样式；当状态改变时，它又会失去这个样式。由此可以看出，它的功能和class有些类似，但它是**基于文档之外的抽象**，所以叫伪类。

与伪类针对特殊状态的元素不同的是，**伪元素**是对元素中的特定内容进行操作，它所操作的层次比伪类更深了一层，也因此它的动态性比伪类要低得多。实际上，设计伪元素的目的就是去选取诸如元素内容第一个字（母）、第一行，选取某些内容前面或后面这种普通的选择器无法完成的工作。它控制的内容实际上和元素是相同的，但是它本身只是基于元素的抽象，并不存在于文档中，所以叫伪元素

## 列表 list-style



list-style-type: upper-roman;

list-style-image: url('sqpurple.gif');

list-style-position: inside; outside inherit

**注1**：outside默认值。保持标记位于文本的左侧。列表项目标记放置在文本以外，且环绕文本不根据标记对齐。inside：列表项目标记放置在文本以内，且环绕文本根据标记对齐

注2：inherit：规定应该从父元素继承 list-style-position 属性的值。

## 表格

table {border-collapse: collapse;} [kəˈlæps]

双边框折叠边框变成一个cellspacing=”0”

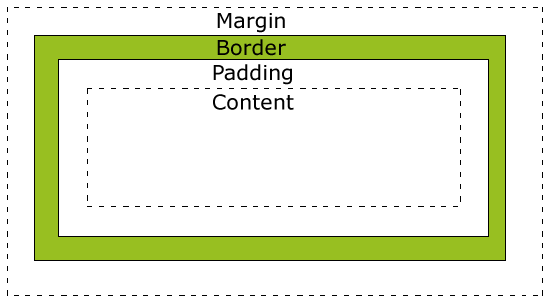
table, th, td { border: 1px solid black; } 指定边框的大小颜色

table {width:100%;} 常用指定：表宽百分比

th{height:50px;} 标题框高度占50像素

th{text-align:right} 表格中文本对齐

## 重要：盒子模型div

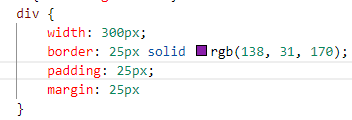


**Content(内容)** - 盒子的内容，显示文本和图像。

**Padding(内边距)** - 清除内容周围的区域，内边距透明的。

**Border(边框)** - 围绕在内边距和内容外的边框。

**Margin(外边距)** - 清除边框外的区域，外边距是透明的



### border边框、轮廓、外边距和填充

#### 边框 border：



常用于<p>、<table>

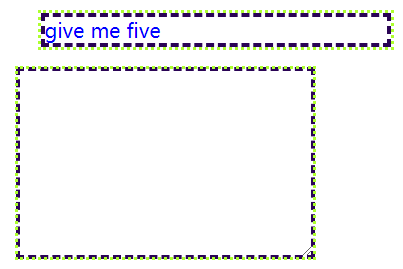
groove ridge inset outset

每边单独设边框：

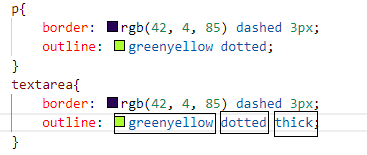
**border-style:dotted solid double dashed; 对应上 右 底 左边框**

注：定义边线的顺序是从”上边框”开始顺时针旋转，若只定义了三条，则未定义那条与对边格式相同；若定义了两条，则对边相同。

#### 轮廓 outline



在框架的边缘附近，特征字同边框



**注：**1.outline是**不占空间**的，既不会增加额外的width或者height（这样不会导致浏览器渲染时出现reflow或是repaint）

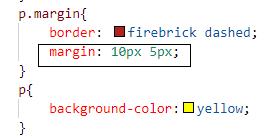
1. outline有可能是**非矩形**的（火狐浏览器下）
2. 一般在格式清零的时候，把outline: none；因为在input中当发生focus事件时，会默认出现蓝色轮廓。

#### margin 外边距 [ˈmɑːrdʒɪn]

margin:100px 50px; 该方法为简写，定义上右，则底同上，左同右

**margin:25px 50px 75px 100px;** [margin-top](https://www.runoob.com/cssref/pr-margin-top.html) [margin-right](https://www.runoob.com/cssref/pr-margin-right.html) [margin-bottom](https://www.runoob.com/cssref/pr-margin-bottom.html) [margin-left](https://www.runoob.com/cssref/pr-margin-left.html)

相关规则同**边框**





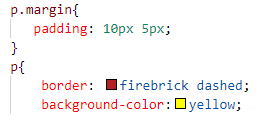
margin值为负的时候：

left为负：向左移，right为负：向左切（切小）

top为负：向上移，bottom为负，横向上切。

#### padding 填充

padding: 30px 10px; 相关定义同边框





## 选择器

### id选择器、class选择器、标签选择器、组合选择器、属性选择器

1. id选择器

例：#para1 /\*在id的前面加上#,定位id位置\*/

{ text-align:center; color:red; }

<h1 id=”para1”>hello world</p>

注1：CSS 中 id 选择器以 "#" 来定义

注2：id属性不已数字开头，否则在Mozilla/Firefox浏览器中不起作用

注3：该例可以指定特定h1标题进行修改，而不会影响其他h1标题

1. class选择器

例1：.student

{ text-align:center; color:red; }

<h1 class=”student”>2018级</h1>

<p class=”student”>新年快乐</p>

注：该情况是选择所有student类的标签

例2：h1.student

{ text-align:center; color:red; }

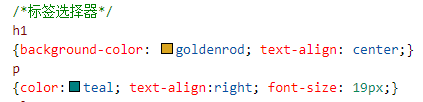
<h1 class=”student”>2018级</h1>

<p class=”student”>新年快乐</p>

注：该例则是只针对有student类的h1标签

1. 标签选择器

例3：标签选择器



**总结**：1.<h1 style="color:red;">菜鸟教程</h3> 在**标签内部**定义style的优先级 > **id选择器** > **class选择器** > **标签选择器**

若一个属性同时在多个选择器中定义，以优先级高的为有效输出

1. 属性选择器

[title=runoob] {border:5px solid green;}

.class[style="color:red;"]

### css的分组选择器、嵌套选择器和交叉选择器

**分组选择器:**

1. 并集选择器：

h1,h2,p { color:green; }

1. 后代选择器 空格相隔

div p { background-color:yellow; } /\*影响div块里的所有p标签内容\*/

1. 子元素选择器 > 分割

div>p { background-color:yellow; }/\*只影响div块子元素p标签内容\*/

<div><span><p>hello world</p></span></div> 将不受影响

1. 相邻兄弟选择器 + 分割，但是是选择的第二个

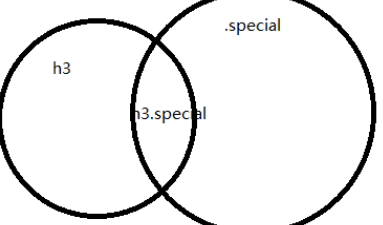
div+p { background-color:yellow; }/\*div块同级且后面一个为p标签时生效\*/

1. 后续兄弟选择器 ~ 分割

div~p { background-color:yellow; } /\*影响div块后面同级的所有p标签的内容\*/

**交集选择器：**

h3.className 有空格就变成后台选择器了



## 显示(display)和可见性(visibility)

### 隐藏元素

可见

例：h1{display:none;} h1{visibility:hidden;} visibility:visible

注：display隐藏后不会占用内存空间，不影响布局。visibility则相反

display隐藏可能会导致位置的变动，想要隐藏而后显示，可用visibility

### 块与内联元素及相互转化 display: inline; display: block;

**块元素**占用全部宽度，前后都有换行符，例：div、p、h1、li等

**内联元素**只需要必要宽度，默认不换行：例：span、a等

**块转内联：**例：li {display: inline;} 可将列表转换成不换行

**内联转块：**例：span { display: block} 估计用于很多个span换行

**inline-block 转换成内联元素，里面的内容还是块元素**

**注1：块元素主要有：**

address , blockquote , center , dir , div , fieldset , form , h1 , h2 , h3 , h4 , h5 , h6 , hr , isindex , menu , noframes , noscript , li , ul , ol ,dl , p , pre , table

**注2：内联元素主要有：**

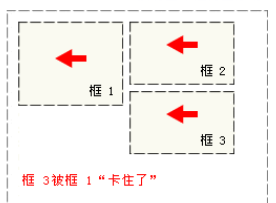
**a** , **abbr** , acronym , **b** , bdo , big , br , cite , code , dfn , em , font , i , **img** , **input** , kbd , label , q , s , samp , **select** , small , **span** , strike , strong , sub , sup ,textarea , tt , u , var

**注3：** 块元素宽度(width)、高度(height)、内边距(padding)和外边距(margin)都可控制;内联元素则不能改变

## 布局：

### 静态布局：（float、position、display）

浮动特点：



position：通过各种定位

静态布局适用于pc端，但也不能完全使用静态布局

优点：兼容性强，简单；缺点：对移动端不友好，需要新布局一套

pc端可以对<html>设置最小宽度min-width超出后出现滚轮，也可以使用overflow-y: auto

移动端：设置最大宽度max-width;margin: 0 auto；超出后居中显示

### \*响应式布局\*：flexbox&bootstrap

响应式布局就是实现不同屏幕分辨率的终端上浏览网页的不同展示方式。通过响应式设计能使网站在手机和平板电脑上有更好的浏览阅读体验。换句话说就是一个网站能够兼容多个终端，而不是为了每一个终端做一个特定的版本。

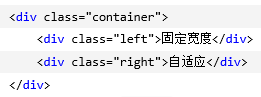
### 自适应布局

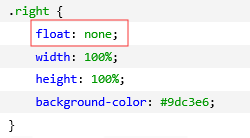
自适应布局就是指能忘了使网页自适应的显示在不同大小终端设备上的新网页设计方式及技术，它需**要开发多套界面来适应不同的终端**

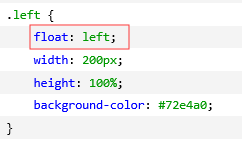
使用flex布局  
左侧浮动或者绝对定位，然后右侧margin撑开  
使用div包含，然后靠负margin形成bfc布局

两种形式：1. 左列固定右列自适应 2. 左右两列固定，中间自适应

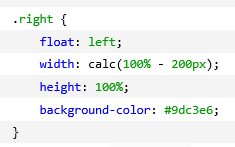
**第一种形式：**



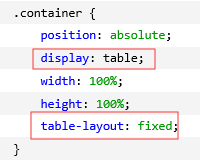


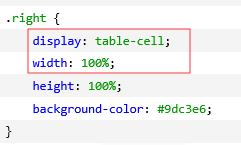


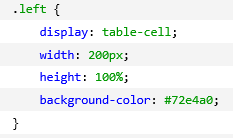
1. 自适应列的width根据calc()自动计算，如：父容器width - 固定列width。



1. **父元素采用display: table时 兼容ie 8+**

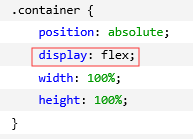


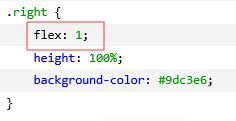


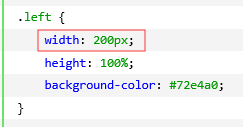


当父元素时table时，里面的子元素会平分父元素空间；当左边元素定宽之后，右边的子元素平分父元素的宽度。

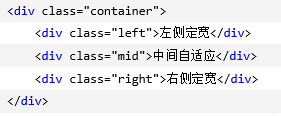
1. **父元素采用 display: flex 兼容ie10+**





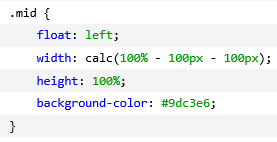


**第二种形式：**



1. **子元素width: calc()**

**同第一种形式的该方式类似，都需要左浮动，只是中间子元素：**



**左右模块width: 100px; 左浮动**

1. **父元素display: flex**

**同第一种形式类似，左右元素左浮动且定宽，中间元素flex: 1**



1. **两侧绝对定位，中间模块不给宽度**

### 弹性布局(flex)

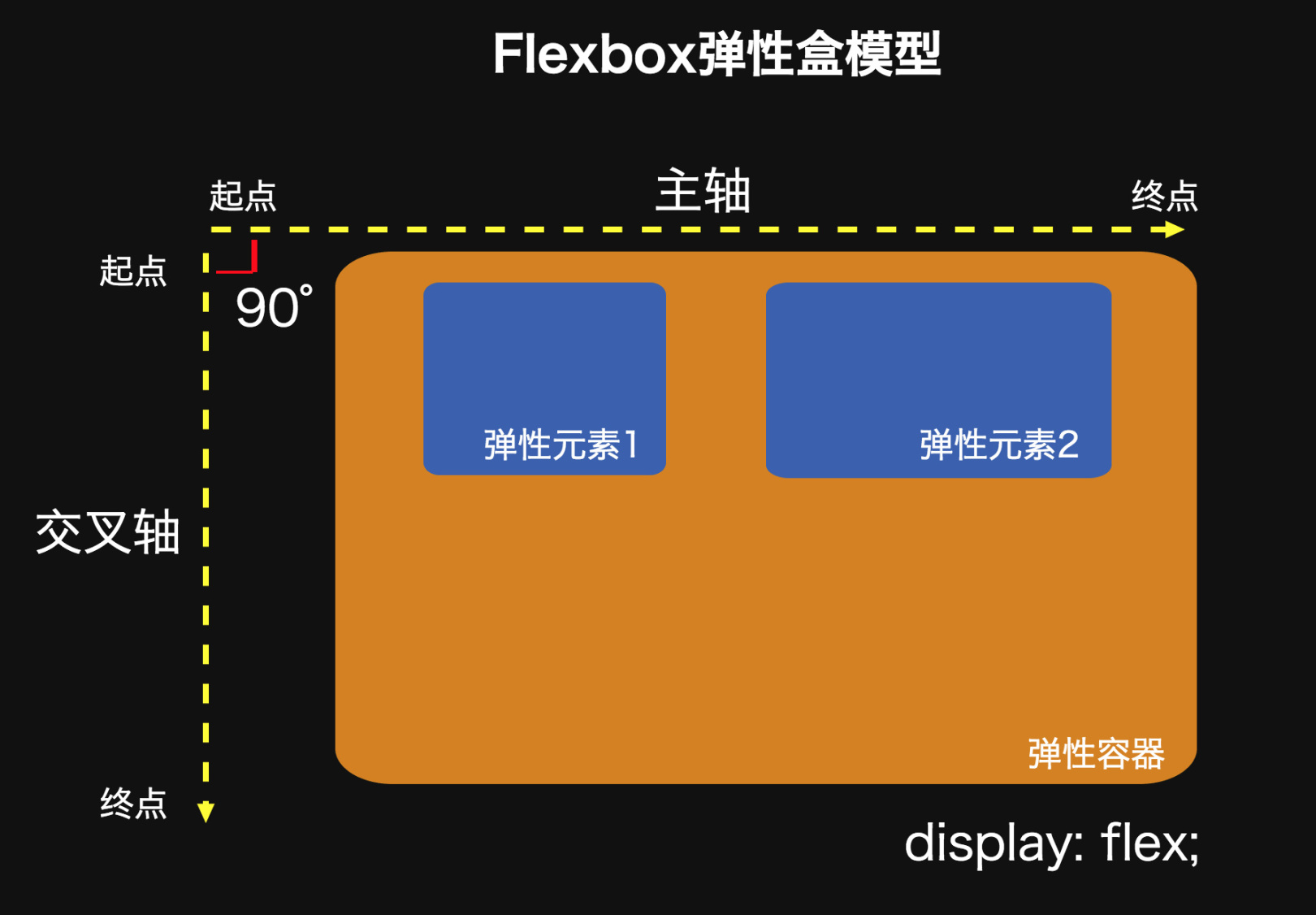
弹性容器属性：

**flex-flow = (flex-direction + flex-wrap)** **justify-content**  **align-item** salign-content

弹性元素属性：

**flex** = (flex-grow + flex-shrink + flex-basis) order align-self

1. 声明display: flex 就看可以成为弹性容器



弹性容器中的所有子元素称为<弹性元素>，**弹性元素永远沿主轴排列**。

弹性元素也可以通过display:flex设置为另一个弹性容器，形成嵌套关系。因此**一个元素既可以是弹性容器也可以是弹性元素**。

1. 选择主轴方向

主轴方向默认从左到右；可以使用**flex-direction**改变主轴方向：

flex-direction: row(默认) flex-direction: column (上—>下)

flex-direction: row-reverse(右-->左) flex-direction: column-reverse(下-->上)

1. 弹性元素如何沿主轴排**flex-wrap**

**flex-wrap: nowrap | wrap | wrap-reverse**

可使得主轴上的元素**不折行(默认)**、**折行**、**反向折行**。

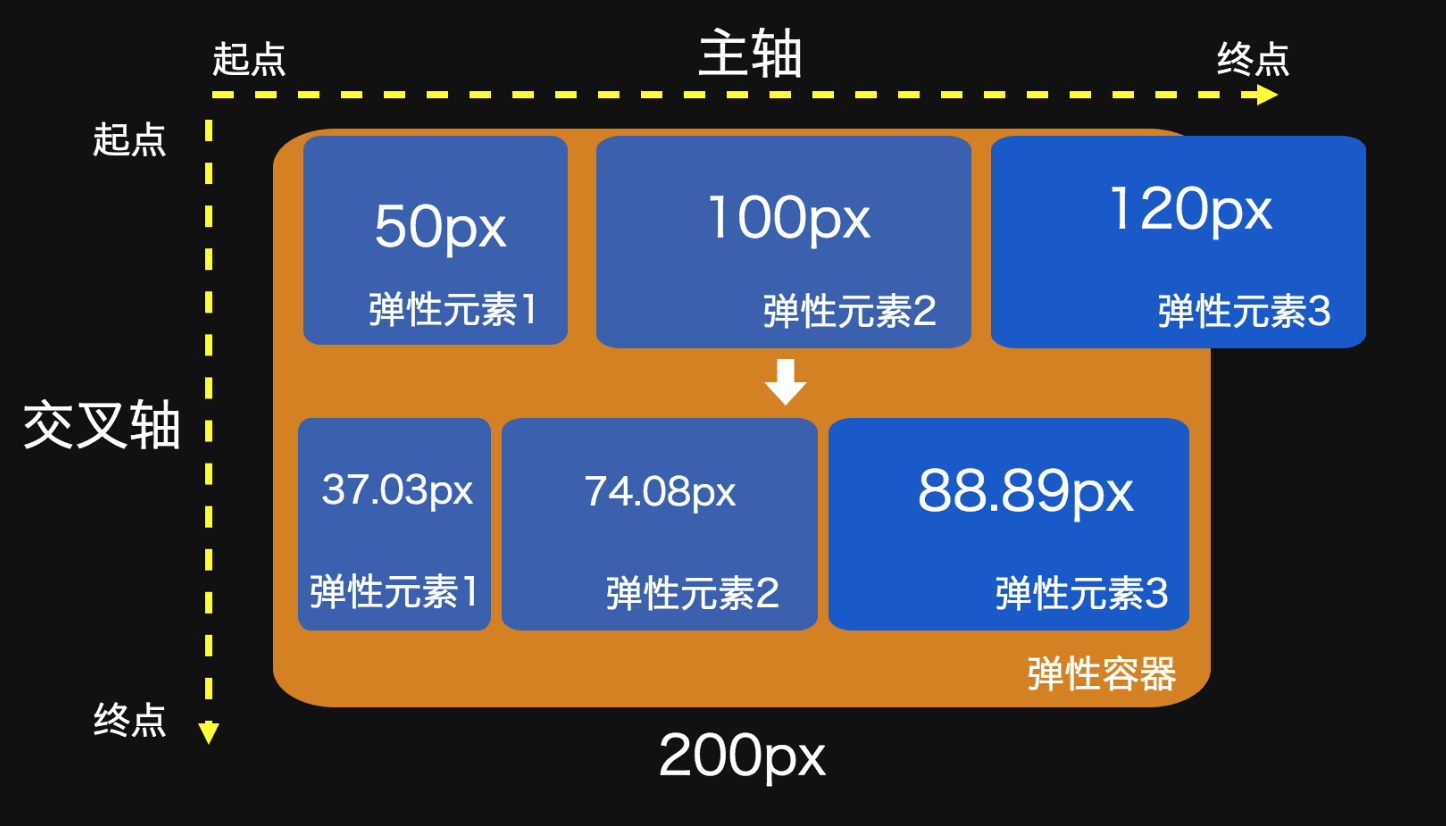
1. **flex-flow = flex-direction + flex-wrap**

**flex-flow: column wrap;**

1. 弹性元素的弹性伸缩应对\*\*\*\*\*

**flex-wrap: nowrap;** 即默认设置时：当弹性容器中的弹性元素总宽度大于容器宽度后：

* 1. flex-shrink: 缩小比例；默认为1，容器中的元素将等比例缩小。



* 1. flex-group: 等比放大；默认为0，即若容器宽度有剩余，不放大

若要对剩余部分进行填充需要对弹性元素设置比重；

例：弹性容器width: 800px 三个弹性元素width: 200px剩余200px

给A元素添加flex-group: 1 C元素添加flex-group: 4

最后：A宽240px B宽200px C宽360px

其实就是常用的flex: 1 flex: 4

1. 固定元素尺寸(**主轴方向**)：**flex-basis (优先级最高) > width/height**

例**：** .text1{width: 0}  .text2{flex-basis: 0;}

当元素中有内容时，text1将不会出现，text2将会根据内容自适应展开

若同时设置flex-basis和宽时，以flex-basis为准。

注：上面的情况为默认主轴方向为左到右，flex-basis设置的是宽；

当flex-direction: column时，flex-basis设置的便是高

1. **flex = flex-grow + flex-shrink + flex-basis**

flex: 1 = flex: 1 1 0%

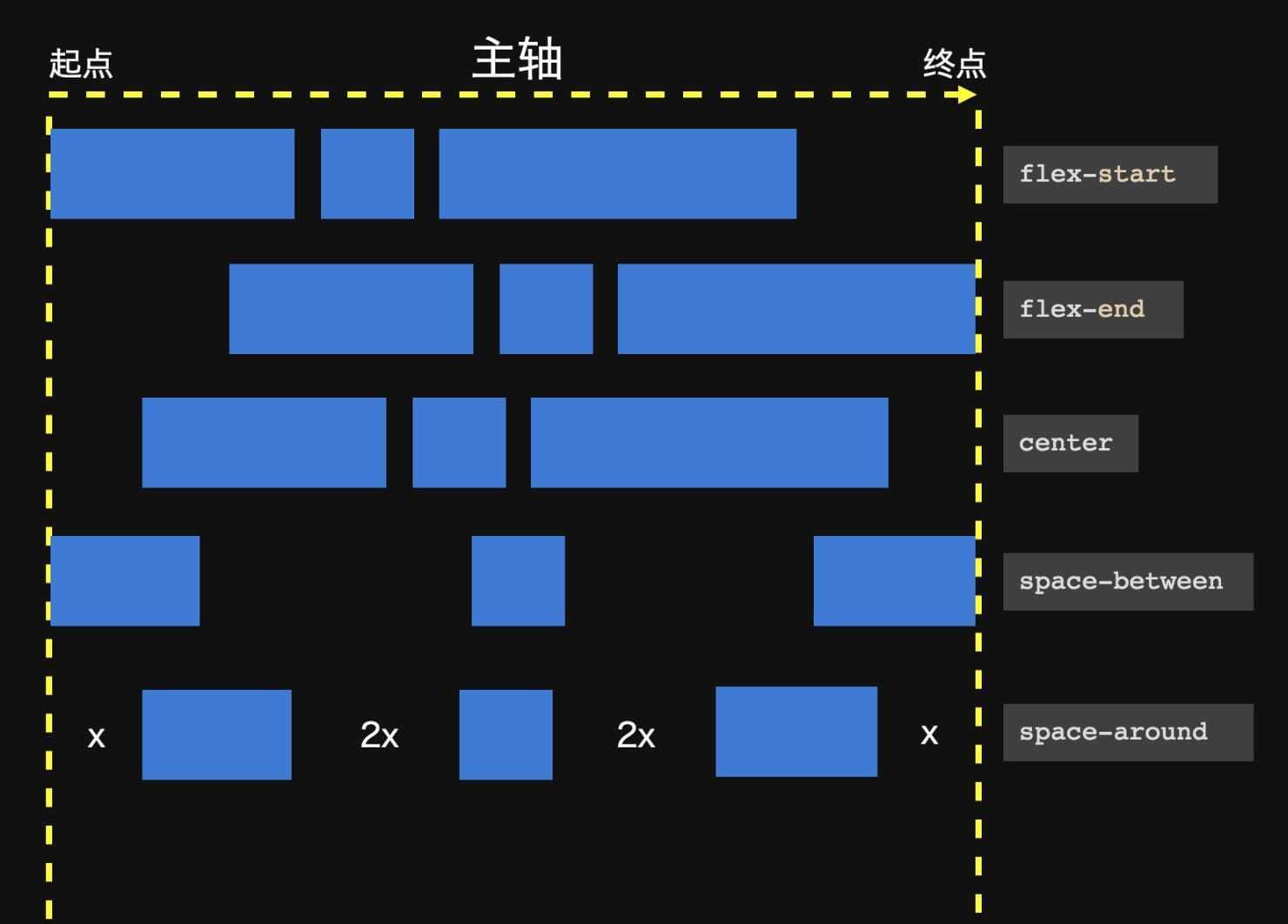
flex: 2 = flex: 2 1 0%

flex: auto = flex: 1 1 auto;

flex: none = flex: 0 0 auto; // 常用于固定尺寸 不伸缩

flex:1和flex: auto区别在于flex-basis的初始尺寸时，前者不考虑flex-grow和flex-shrink后者考虑

1. 弹性元素的对齐
2. 主轴上 **justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly**



space-enenly: 容器中主轴方向上的单行元素间隔相同

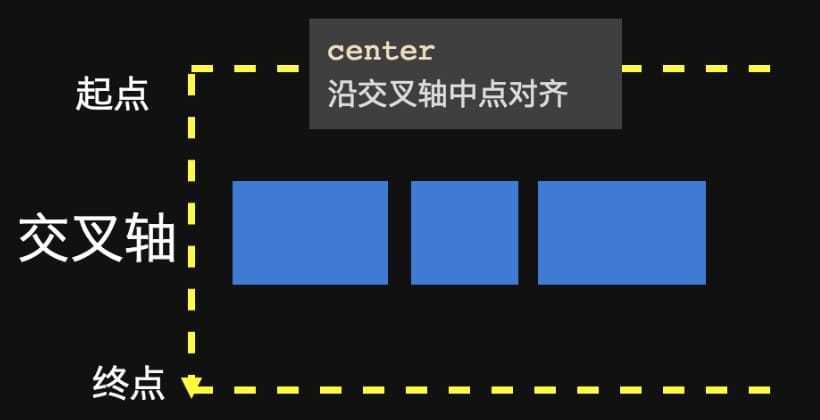
1. 交叉轴上的单行对齐**align-items**

默认值为：stretch: 当元素未设置具体尺寸时，元素高度在交叉轴方向将容器撑满

设置：align-items: flex-start 将会沿交叉轴起点对齐

flex-end则沿交叉轴终点对齐

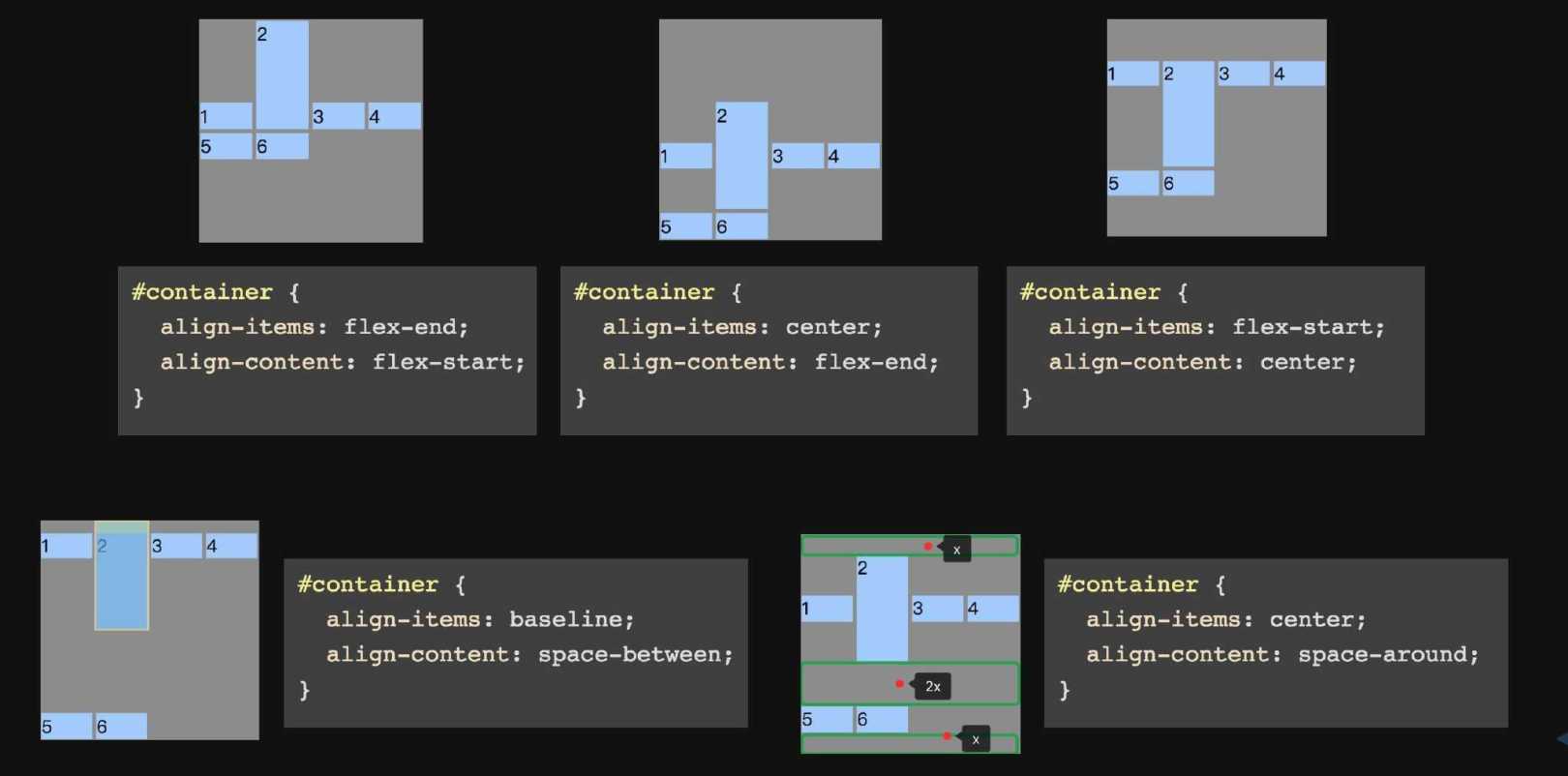
flex-center则于容器中间位置对齐



1. 交叉轴上的多行对齐 **align-content**

**align-content: stretch | flex-start | flex-end | center | space-between | space-around**

只有当出现多行时触发



1. order 调整元素顺序



#container > div:first-child { order: 2; } #container > div:nth-child(2) { order: 4; } #container > div:nth-child(3) { order: 1; } #container > div:nth-child(4) { order: 3; }

### 流式布局(百分比布局)

采用百分比为单位

引入自适应布局，常用设计

左侧固定+右侧自适应  
左右固定宽度+中间自适应

优：元素的宽高用百分比做单位，元素宽高按屏幕分辨率调整，布局不发生变化

缺：屏幕尺度跨度过大的情况下(内容撑爆或太空)，页面不能正常显示

使用：pc端一般不用，移动端结合rem使用

rem是相对根元素的字体大小的单位；em是相对父元素，嵌套太麻烦

**浏览器默认字体高16px或14px**

目前，IE9+，Firefox、Chrome、Safari、Opera 的主流版本都支持了rem。

兼容：就算对不支持的浏览器，应对方法也很简单，就是多写一个绝对单位的声明。这些浏览器会忽略用rem设定的字体大小。

使用：首先需要在<body>或者<html>的css中声明：font-size: 62.5%

16px = 100% 1px = 6.25% 10px = 62.5%

当根元素设置62.5%之后，10px = 1rem

这样设置的好处是主需要修改跟节点的font-size属性，整个页面的字体就能发生改变；这样就可以很好的做各种屏幕大小的适配了

@media only screen and (min-width: 320px){

html{ font-size: 62.5% !important; }

}

@media only screen and (min-width: 640px){

html{ font-size: 125% !important; }

}

@media only screen and (min-width: 750px){

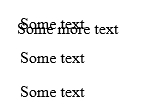
html{ font-size: 150% !important; }

}

……

### 定位： static fixed relative absolute sticky z-index

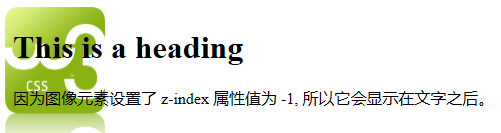
1. static(默认值，无定位) **静态定位的元素不会受到 top, bottom, left, right影响**。
2. fixed：页面滚动，元素在窗口的位置不变。p{position:fixed; top:30px; right:5px; } Fixed定位使元素的位置与文档流无关，因此不占据空间。可与其他元素重合(就你不动，当然会挡住其他内容)



1. relative：相对正常位置进行偏移：h2 { position:relative; left:20px; }
2. absolute：绝对定位，有最近父元素，按父元素偏移，若无则现对于html

h2 { position:absolute; left:100px; top:150px; }

1. stick：粘性定位：滚动窗口，在窗口超过粘性定位的元素时，该元素将会黏在窗口的固定位置。且与fixed和relation之间，一般设置在top: 0,在最上方粘住
2. **z-index：可以让某元素覆盖在某元素之上或之下**



例：img

{

position:absolute;

left:0px;

top:0px;

z-index: -1;

} /\* -1表示图片作为背景在下，1表示图片在上\*/

|  |  |  |
| --- | --- | --- |
| **属性** | **说明** | **CSS** |
| [bottom](https://www.runoob.com/cssref/pr-pos-bottom.html) | 定义了定位元素下外边距边界与其包含块下边界之间的偏移。  Auto *length %* inherit | 2 |
| [clip](https://www.runoob.com/cssref/pr-pos-clip.html) | 剪辑一个绝对定位的元素  Shape auto inherit | 2 |
| [cursor](https://www.runoob.com/cssref/pr-class-cursor.html) | 显示光标移动到指定的类型  url auto crosshair default pointer move  e-resize ne-resize nw-resize n-resize se- sw- s- w-  text wait help | 2 |
| [left](https://www.runoob.com/cssref/pr-pos-left.html) | 定义了定位元素左外边距边界与其包含块左边界之间的偏移。  Auto length % inherit | 2 |
| [overflow](https://www.runoob.com/cssref/pr-pos-overflow.html) | 设置当元素的内容溢出其区域时发生的事情。  Auto hidden scroll visible inherit | 2 |
| [overflow-y](https://www.runoob.com/css/cssref/css3-pr-overflow-y.html) | 指定如何处理顶部/底部边缘的内容溢出元素的内容区域  Auto hidden scroll visible no-display no-content | 2 |
| [overflow-x](https://www.runoob.com/css/cssref/cssref/css3-pr-overflow-x.html) | 指定如何处理右边/左边边缘的内容溢出元素的内容区域  Auto hidden scroll bisible no-display no-content | 2 |
| [position](https://www.runoob.com/cssref/pr-class-position.html) | 指定元素的定位类型  Absolute fixed relative static inherit | 2 |
| [right](https://www.runoob.com/cssref/pr-pos-right.html) | 定义了定位元素右外边距边界与其包含块右边界之间的偏移。  auto length % inherit | 2 |
| [top](https://www.runoob.com/cssref/pr-pos-top.html) | 定义了一个定位元素的上外边距边界与其包含块上边界之间的偏移。  auto length % inherit | 2 |
| [z-index](https://www.runoob.com/cssref/pr-pos-z-index.html) | 设置元素的堆叠顺序  Number auto inherit |  |

### overflow：auto

overflow元素的元素值如下

1. visible：默认值。内容不会被修剪，会呈现在元素框之外。
2. hidden：内容会被修剪，并且其余内容是不可见的。
3. scroll：内容会被修剪，但是浏览器会显示滚动条以便查看其余的内容。
4. **auto：**如果内容被修剪，浏览器会显示滚动条以便查看其余的内容(相比scroll能只能)

注1：overflow用于**指定高度**的**块元素**中

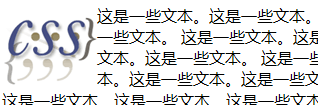
注2：mac系统中滚动条默认隐藏

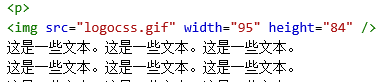
### 浮动 float

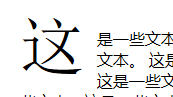
会使元素向左或向右移动，其周围的元素也会重新排列。常用于**图像**，但它在布局时一样非常有用。

例1：在段落中插入图片，要求图片显示在左上角做开头

img { float: left}







{ **float**:left; **width**:1.2em; **font-size**:400%;

**font-family**:algerian,courier;

**line-height**:80%; }

例2：在段落中插入多张图片，它们会顺序水平排开，且可以给img元素设置浮动和尺寸，让每张图都一样大小。

.thumbnail {float:left; width:**50px**; height:90px; margin:5px;}

<img class="thumbnail" src="/images/klematis\_small.jpg" width="**107**" height="90">

<img class="thumbnail" src="/images/klematis2\_small.jpg" width="**107**" height="80">



**清除 clear:** 元素浮动之后，周围的元素会重新排列，为了避免这种情况，使用 clear 属性。clear 属性指定元素两侧不能出现浮动元素。使用 clear 属性往文本中添加图片廊：

.text\_line{**clear:both**;margin-bottom:2px;}

<img class="thumbnail" src="/images/klematis\_small.jpg" width="107" height="90">

<img class="thumbnail" src="/images/klematis2\_small.jpg" width="107" height="80">

<h3 class=**"text\_line"**>第二行</h3>

<img class="thumbnail" src="/images/klematis3\_small.jpg" width="116" height="90">

<img class="thumbnail" src="/images/klematis4\_small.jpg" width="120" height="90">

**注：该方法可以让图片重新换行排列，用<br>不行**

### 居中、对齐

水平居中：

1. 文本水平居中对齐：text-align: center;
2. 定宽块级：元素(例div)、图片居中对齐：width小于100%，**margin: 0 auto**
3. 不定宽块级：text-align：center 或者给它的父元素添加text-align：center；常用于网页的分页导航，不能定宽来限制他的弹性

左右对齐用：**position: absolute 和 float: left text-align: right**

垂直居中：

1. 文本垂直居中：line-height
2. 块级不限高：给父元素添加display:table-cell; vertical-align:middle
3. 块级限高：计算子元素的margin-top或margin-bottom，计算方法为父(元素高度-子元素高度)/2

浮动元素水平垂直居中：定宽高-50px和-100px为宽度的一半

{ … left: 50%; top: 50%; margin: -50px 0 0 -100px; }

绝对定位的左右居中：{ … margin: 0 auto; left: 0; right: 0; }

display: flex;justify-content: center;align-items: center;

### 元素尺寸



### less 和scss（预处理器）

CSS预处理器为CSS增加一些编程的特性，**无需考虑浏览器的兼容性问题**，例如你可以在CSS中使用变量、简单的逻辑程序、函数等等在编程语言中的一些基本特性，可以让你的CSS更加简洁、适应性更强、可读性更佳，更易于代码的维护等诸多好处。

**less的使用**：

<link rel="stylesheet/less" type="text/css" href="styles.less">

<script src="less.js" type="text/javascript"></script>

vue中：

**使用：**

**变量：** @bgColor: #f5f5f5; body { background: @bgColor;}

**嵌套：**

#id {aaa}

#id > .className {bbb}

#id > .className a {ccc}

#id > .className a:hover {ddd}

#id { aaa

> .className { bbb

a { ccc

&hover{ddd}

}

}

}

**Mixin:**

Mixin 的作用是把**重复的代码**放到一个类当中，每次只要引用类名，就可以引用到里面的代码了

#header {

.roundedCorners; // 默认5px

}

#footer {

.roundedCorners(10px);

} // 改成10px

.roundedCorners(@radius: 5px) {

-moz-border-radius: @radius;

-webkit-border-radius: @radius;

border-radius: @radius;

} // 定义一个类

**import:**

在开发阶段，我们可以将不同的样式放到多个文件中，最后通过@import 的方式合并。意思就是，当出现多个 less 文件时，怎么引用它们。

这个很好理解， css 文件可以有很多个，less文件也可以有很多个。

1. 定义一个被引用的less文件： \_xxx.less

.btn{ line-height: 100px; color: @btnColor; }

1. 引入：

.btn { line-height: 100px; color: red; }

body { width: 1024px; }

@btnColor: red;

@import url(“./xxx.less:”)

body{ width: 1024px; }

**内置函数: (lighten darken)**

background-color: lighten(#000, 10%); //变亮10%

background-color: darken(#000, 10%) // 变暗10%

**运算：**

@the-border: 1px; => body { border: @the-border \* 10 } // 10px

@base-color: #111; => body {border: @base-color @base-color + #003300;} // #114411

**SCSS:**

**变量： $**

**运算符**

**混合：**

**继承**

## CSS3

### 边框

1. **圆角边框 border-radius: 50% 背景变成圆形**



div{

border:2px solid; **border-radius:25px;**}

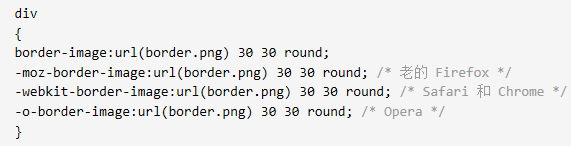
1. **边框阴影**



div{

box-shadow: 10px 10px 5px #888888;}

1. **边框图片**



### 背景

两种新方法：1. background-size 2. **background-origin(常用于背景位置居中)**

div

{

background:url(bg\_flower.gif);

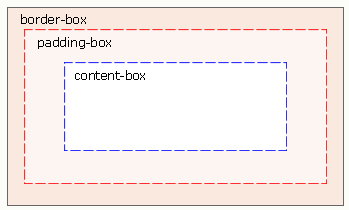
-moz-background-size:63px 100px; /\* 老版本的 Firefox \*/

**background-size:63px 100px;** 修改背景图片尺寸

background-repeat:no-repeat; 平铺设为否，只出现一个背景

}

注：设置40%百分比尺寸，可以自适应内容



div

{

background:url(bg\_flower.gif);

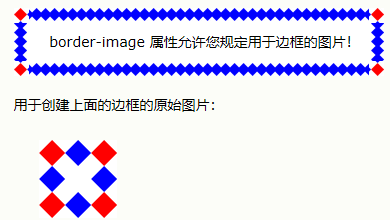
background-repeat:no-repeat;

background-size:100% 100%;

-webkit-background-origin:content-box; /\* Safari \*/

**background-origin:content-box;**

}



### 文本及字体

1. **文本阴影**

h1{ **text-shadow**: 5px 5px 5px #FF0000;}

1. **自动换行：**p {word-wrap:break-word;}

### 渐变 linear-gradient radial-gradient

渐变可以在两种或者多种颜色之间平稳的过渡；

相关属性：background-image:

相关语法：

background-image: **linear-gradient**(direction, color-stop1, color-stop2, ...);

线性渐变：由向上下左右对角方向(默认)

background-image: linear-gradient(#e66465, #9198e5);

background-image: linear-gradient(to right bottom, #e66465, #9198e5, blue);

background-image: linear-gradient(45deg, #e66465, #9198e5)；

径向渐变：由内到外

background-image: radial-gradient(circle, red 5%, yellow 15%, green 60%);

### 2D/3D转换（用的时候需要考虑浏览器兼容） transform

2D转换有：

translate(): transform: translate(50px,100px); 水平、垂直移动

rotate(): transform: rotate(30deg); 顺时针旋转30度

scale(): transform: scale(2,3)； 宽变成原来的2倍，高变成3倍 [skeɪl]

3D转换有：

retateX(): transform: rotateX(120deg); 绕着x旋转120度

retateY(): transform: rotateY(120deg);

### 过渡 (注意浏览器兼容问题) transition

-webkit-transition: width 2s, height 2s, transform 2s;

transition: width 2s, height 2s, transform 2s;

   div:hover{

    width: 2000px; height: 2000px;

       transform: rotate(360deg);

-webkit-transform: rotate(360deg);

### 动画 animation

从一种状态慢慢变成另一种状态

**定义动画**：

@keyframes myfirst{ /\*需要考虑兼容，再添加@-webkit- keyframes myfirst{ }\*/

0% {background:red; left:0px; top:0px;}

25% {background:yellow; left:200px; top:0px;}

50% {background:blue; left:200px; top:200px;}

75% {background:green; left:0px; top:200px;}

100% {background:red; left:0px; top:0px;}

}

**调用：animation: myfirst 5s;**

### 多列 column-{ count, gap,rule-style,rule-width,rule-color,width}

count(4): 分成四列；gap(40px): 列与列之间的空隙为40像素

rule: 1px solid blue; 设置列与列之间的分割线的粗细、类型、颜色

width 100px; 规定每列的宽度

# js基础

## 输出

使用 window.alert() 写入警告框 直接用alert()也行

使用 document.write() 写入 HTML 输出

<button onclick="document.write(5 + 6)">试一试</button>

**注意：**在 HTML 文档完全加载后使用 **document.write()** 将**删除所有已有的 HTML**,即点击按钮将删除原html输出只有11的页面

使用 innerHTML 写入 HTML 元素

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML = 5 + 6; // 输出11

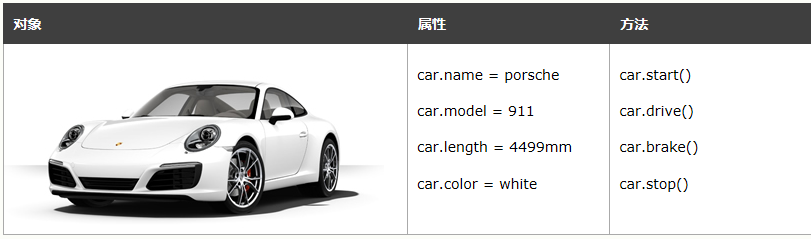
</script>

使用 console.log() 写入浏览器控制台

在浏览器中，可使用 console.log() 方法来显示数据。

通过 F12 来激活浏览器控制台，并在菜单中选择“控制台”。

## 重要：对象 & this & 继承



var person = {

firstName: "Bill",

lastName : "Gates",

id : 678,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

### this:

首先分析this所在的函数是当做哪个对象的方法调用的，则该对象就是this所引用的对象。 **this的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定this到底指向谁**，**实际上this的最终指向的是那个调用它的对象**

**例1**：

var obj = {}; obj.x = 100; obj.y = function(){ alert( this.x ); }; obj.y();

这里的this外的函数是作为obj对象的方法调用的，所以this是指向obj对象。

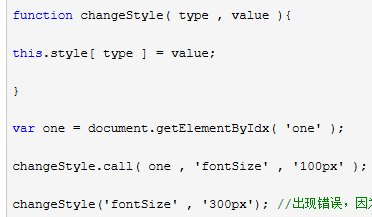
**例2**：在JavaScript的变量作用域里，全局变量都是window对象的属性。所以用var声明的函数里面的this指向window。

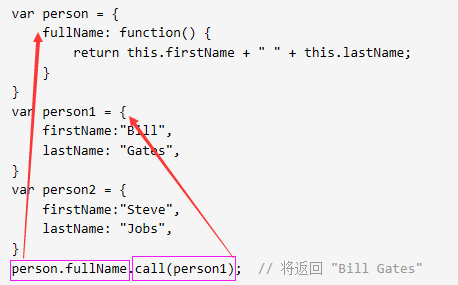
**call() & apply()**

**例3**：call()函数可以重新定义函数的执行环境，即可以改变this的指向

原本changeStyle()中的this是指向window的，window没有该属性，如果直接传参就会报错。

如果在传参的时候加上call(),call()函数第一个参数会指定将被谁调用，左图则是被id=one的模块调用，而该模块才有css属性

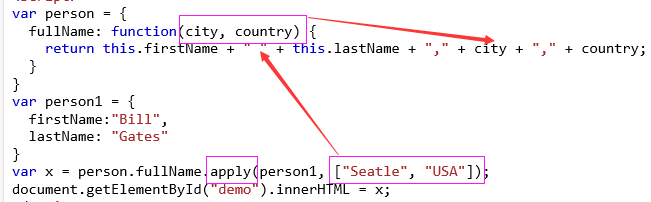




apply()作用和使用方法同call()，区别在于：call() 方法分别接受参数。

apply() 方法接受数组形式的参数。

接例3



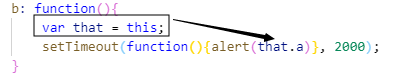
**补充**：

**情况一**：如果一个函数中有this，但是它**没有被上一级的对象所调用**，那么this指向的就是window。

例4：

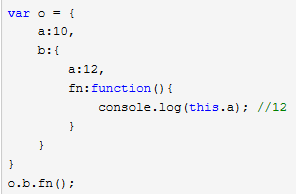


上面的this指向window，应修改为：或者改为**箭头函数**



情况二：如果一个函数中有this，有多个对象，这个函数有被上一级的对象所调用，那么this指向的就是上一级的对象。

例5



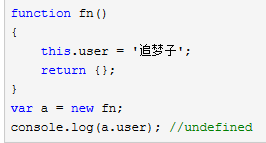
虽然是从o对象开始调用的，但是最后调用this的对象是b，所以this会指向b而不是o

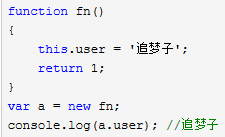
注：this永远指向的是**最后调用它的对象**。

若var j = o.b.fn; j(); this则会指向window，因为在赋值给j的时候没有调用，而j()调用的时候是window.j()，就不会指向对象b了。

当this遇到return时：

例6





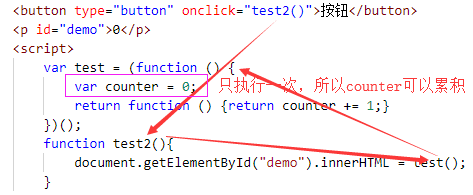
若return返回的是一个对象，则this就指向这个对象；若不是，则返回函数实例function(){}；若没有返回，new会自动创建一个空对象。

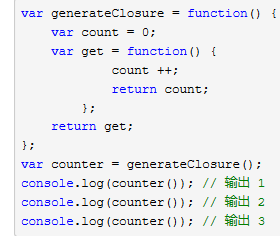
### 闭包 和 匿名函数

#### 闭包

全局变量能够通过闭包实现局部（私有）

例：计数器





**闭包的本质是函数内部创建一个函数**。

全局作用域的变量无法影响局部作用域的，局部作用域的变量可以应用全局的；所以test()函数中的函数在将counter初始化为0后，没有关于counter的调用，则test()函数的初始化并不影响里面函数counter的值，因此可以累加。

**用途：**一个是实现嵌套的回调函数实现异步编程，二是隐藏对象的细节实现模块儿化编程

**好处**

①保护**函数内的变量安全** ，实现封装，防止变量流入其他环境发生命名冲突

②在内存中维持一个变量，可以做**缓存**（但使用多了同时也是一项缺点，消耗内存）

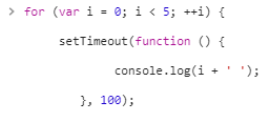
③匿名自执行函数可以减少内存消耗

**坏处**

①其中一点上面已经有体现了，就是被引用的私有变量不能被销毁，增大了内存消耗，造成内存泄漏，解决方法是可以在使用完变量后手动为它赋值为null；

②其次由于闭包涉及跨域访问，所以会导致性能损失，我们可以通过把跨作用域变量存储在局部变量中，然后直接访问局部变量，来减轻对执行速度的影响

常见问题：



会同时输出5个5；原因如下：js是单线程运行，在进行for循环的时候，setTimeout会被安排到任务列表等待执行，等set开始执行时，for循环已经结束，所以也就会同一时间输出。若是setInterval则会等for循环结束之后一直重复输出for循环结果。

#### 匿名函数

匿名函数常用场景：

1. document.querySelector().onclick = function(){ … }
2. setInterval(function(){ … }, 1000);
3. var myFunction = function(){ … }
4. 对象中的方法（vue中的方法）

**匿名函数自执行：**

1. (function(a){ … })(b);
2. (function(){ … } ());
3. ! (function(a){ … })(b);
4. var fun = function(a){ … }(b);
5. 匿名自执行函数最常见的作用是用于实现闭包的情况中。见闭包中的图一
6. 匿名自执行函数还可以用于在js中模拟创建**块级作用域**，即如果使用匿名自执行函数将某些代码包裹起来可以实现块级作用域的效果，减少全局变量的数量，在匿名自执行函数执行结束后变量就会被内存释放掉，从而也会节省了内存。

### 对象与继承

面向对象编程(oop)的三要素：封装、继承、多态

**封装性**：一种信息隐蔽技术，使用户只能看到对象的外特性，对象中的内特性对用户是隐蔽的。目的，把对象的设计者和对象的使用者分开，使用者不需知道细节，只需使用设计者提供的消息来访问该对象。

总结： 封装就是事物抽象为类，把对外接口暴露，将实现和内部数据隐藏。

**继承性**：复用原有功能并能修改和扩充

继承创建的新类称为“子类”或“派生类”

被继承的类称为“基类”、“父类”或“超类”。

**多态性**：对象根据所接收的消息而做出动作，同一消息为不同对象接受时可产生完全不同的行为。从系统层面讲，一个系统会有多个面向对象，而当不同的对象接收到同一消息后，所表现出的行为是各不相同的，所以面向对象具有多态性。

**组成：属性和方法 类：手机**

属性：描述一种状态 例：长方体、黑色、屏幕

方法：描述一种行为 例：手机的功能，能做什么

类相当于模具，根据传递的数据来创建对象，并且可以重复的创建

调用必须用new

一般名字首字母大写

类中的this指向new构造的对象

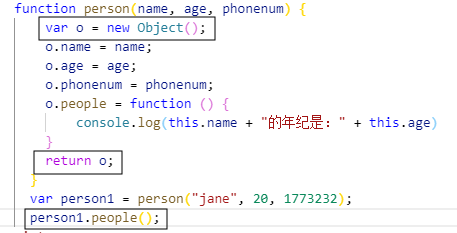
具备隐式返回，所有不用写return

父类子类，例:animal是父类，cat、dog是子类

适用场景：以处理集合为中心的函数式方式更适合处理与人有关的数据，而面向对象的方法最适合模拟人

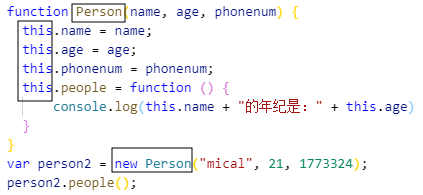
**创建对象：**

1. **工厂模式：**



可以解决创建多个相似对象的问题，但无法解决对象识别问题（不知道对象的类型）

1. **构造函数模式：**



注一：1. 函数名首字母大写，作为一个对象名；2. 没有工厂模式显式的创建对象；3. 没有return，会自动隐式创建；4. 直接将属性和方法赋给this对象； 5. 在创建新实例的时候必须用new，通过new创建实例，会将this指向给person2。

注二：当该对象被另一个对象的作用域中调用时。需要用call()把this指向obj

var obj = new Object();

Person.call(obj, “tang”, 23, 1776459);

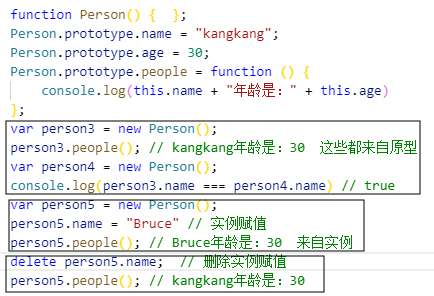
obj.people();

1. **原型模式**

上面的构造函数模式无法自定义方法。

每个函数创建后都有一个prototype(原型)属性，他是一个指向对象的指针，可直接指定具体信息到原型对象中。

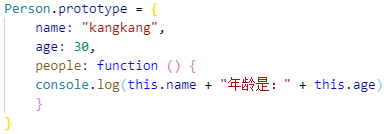
所有对象实例共享该对象所包含的属性和方法。



对象实例添加的属性，会覆盖对象中原有的同名属性，但是会保留原型属性；当实例属性删除后，原型属性再次出现。

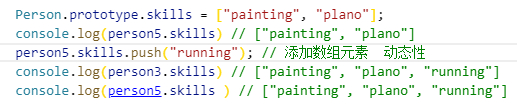
注1：判断一个属性是否在对象中console.log( “name” in persion3) // true

注2：原型语法简写：



注3：动态性，即当person3实例对象创建之后，再对原型对象进行补充：Person.prototype.phonenum = 1776543; 该实例也能访问到。

注4：缺陷；因为所有对象实例共享对象的属性和方法，所以，当属性包含有数组，一个对象实例改变数组元素的时候，其他实例对象访问到的数组也是改变之后的；只有数组会如此。



1. **构建函数和原型模式的组合 (广泛使用)**



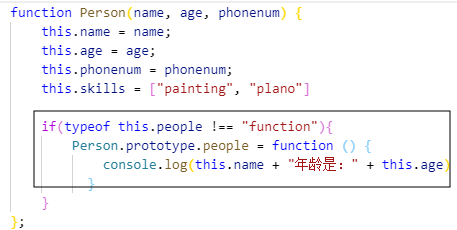
强行指向Person

用构造函数定义**实例属性**，用原型定义**方法**和**共享属性**

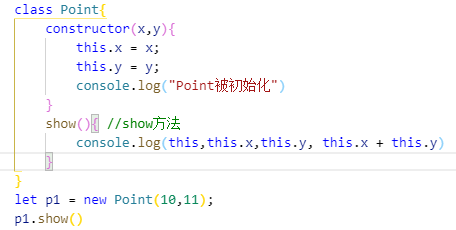
**注：简写的prototype，他的实例对象不再指向Person而是Object，可通过person6.constructor查看，而上面的constructor: Person 进行强制指向。**

1. **动态原型模式**

上诉方法是将构建函数模式和原型模式分开写，可读写不佳，可以把这两者封装到一个构建函数中：



1. **class ES6新提供**

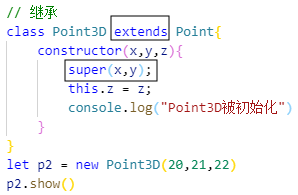


class创建对象的本质是函数构建，只是比较特别

一个类中只能有一个名为constructor()的构造器；定义方法为show(){}

类中没有私有属性；

继承可以extends关键字：调用父类有super方法



1. json方式的面向对象



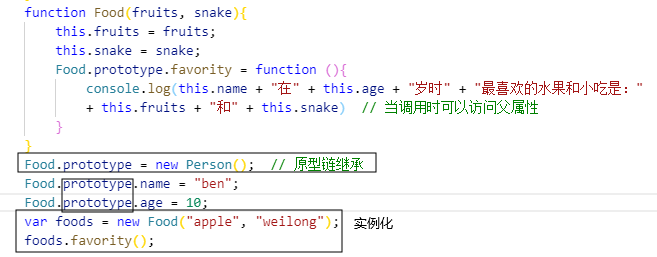
#### 继承

1. **原型链**

通常继承分为接口继承和实现继承；但ECMAScript只支持实现继承；原型链作为实现继承的主要方法。

基本思想：利用**原型**让一个引用类型继承另一个引用类型的属性和方法。

例：继承上面的动态原型模式



继承父类和修改父类属性都用prototype, 不用将不会被定义.

缺点：还是原型共享问题，继承后定义或者改变属性，其他实例对象也会改变，所以实际很少单独使用。

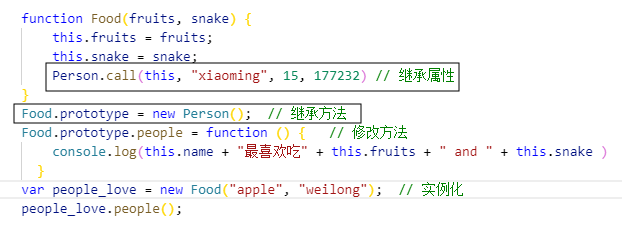
1. **借用构造函数**（经典继承）



传参：Person.call(this, “kangkang”, 10, 1773434)

缺点：方法在构造函数内部已经被定义，无法复用。

1. **组合继承**：继承 构建函数和原型模式的组合的父类 (最常用)



用原型链继承父类的方法实现共享；用call()继承父类的属性，变成实例私有

## 事件

HTML 事件是发生在 HTML 元素上的“事情”。当事件发生时，用户会希望做某件事。

例：点击按钮来获取时间

<p id="demo"></p>

<button type="button" onclick="document.getElementById('demo').innerHTML = Date()">Click Me!</button>

例：点击按钮，获取时间；时间显示在按钮框中，代替原字样

<button onclick="this.innerHTML=Date()">时间是？</button>

**常见HTML事件：**



## 方法

### 字符串与正则：

var str = “hello world a vo hello aa vo”

1. **.length**方法： var b = str.length
2. .indexof()方法：var b = str.index(“vo”) 获取字符串中首次出现vo的索引位置

var b = str.index(“vo”, 9) 从第9个字符开始

1. .lastIndexof()方法：var b = str.lastIndexof(“vo”) 最后一次出现o的索引位置

var b = str.lastIndexof(“vo”, 30) 从第30个字符开始往前匹配

1. **.search()**方法：var b = str.search(“vo”) 搜索特定字符串并返回索引位置

**注1：搜索不到字符串返回-1,**

**注2：search和indexof异同：**1. search() 方法无法设置第二个开始位置参数；2. indexOf() 方法无法设置更强大的搜索值（正则表达式）

1. **.slice()方法**：var b = str.slice(3, 5) 获取索引位置是3、4的两个字符 [ str )

var b = str.slice(-5, -3) ( str ]

var b = str.slice(-3) 取最后三个字符

var b = str.slice(3) 从第四个字符开始取

1. .substring类似于slice，但是不接受负索引
2. .substr()类似于slice, var b = str.substr(3, 4) 从第四个字符开始截取后4位
3. **.replace()** 替换首个匹配的字符

var b = str.replace(/vo/g,”vv”) g是替换所有匹配到的，i是变成大小写不敏感

1. **.concat()** 连接两个或者多个字符串
2. .trim() 删除字符串两端的空白符 str.trim() IE8以下不支持

转换大小写

1. **.toUpperCase() str.toLowerCase()** 把字符串转换为小写

提取字符串字符

1. .charAt(1) 返回第二个字符 str.charCodeAt(1) 返回第二个字符的Unicode编码

把字符串转换成数组

1. **.split()** var txt = str.split(“,”) 用，号分割字符，将分割的字符变成**数组**

反转一个字符串：a = str.split(“”).reverse().join(“”)

**正则：**

创建

第一种方式：var reg = /cnm/gi

第二种方式：var reg = new RegExp(“cnm”, “gi”)

修饰符：g i m

g表示全局匹配；不加则匹配到第一个之后停止；i表示不区分大小写；

m表示执行多行匹配。

量词：n\* n+ n？

\*表示匹配任意字符，+表示匹配至少一次n；？表示匹配n为0或者1次，优先0次；

[]、{}、 (|)：

[]: 表示一个范围，一个[]只表示一个字符，[a-z]，[abc], [^a-z]

n{2}, n{2,}, n{2,9}：匹配n两次，至少两次，两到九次

(red|blue|green): 匹配三个字符串中的其中一个

元字符： \w \W \d \D \s \S 大写相反

\w === [0-9a-zA-Z] \d === [0-9] \s : 查找空白字符

正则中常使用的字符串方法：str.search(/xxx/gi) 和 str.replace(/xxx/gi, yyy)

search() 方法匹配到对应的字符或字符串之后返回索引值.

var par = /(^\s+)|(\s+$)/g;

str.replace(par, ‘’)

**RegExp对象**：是一个预定义了属性和方法的正则表达式对象

test()方法：正则表达式方法，该方法用于检测一个字符串是否匹配某个模式，如果含有该字符串返回true。(判断字符串中有没有能匹配到的)

例：var reg = /oo/gi;

reg.test(“goole”) // true 或者 /oo/.test(”goole”)

exec()方法：用于检索字符串中正则表达式的匹配，返回数组，存放匹配到的值。

**var patt = new RegExp(“[a-z]9{2}\W$”, “gi”)**

// var patt = /fre{2,}/gi;

document.write(patt.exec("The best things in life are freE"));

**正则表达式中的RegExp.$1**

**RegExp是js的内置对象，**RegExp.$1是RegExp的一个属性：指正则匹配到的第一次子匹配(以括号为标识)

var reg= /^(\d{4})-(\d{1,2})-(\d{1,2})$/; //正则表达式 匹配出生日期(简单匹配)

reg.exec('1985-10-15');

s1=RegExp.$1;

s2=RegExp.$2;

s3=RegExp.$3;

document.getElementById('id1').innerHTML = s1 + ' ' + s2 + ' ' + s3

// 1985 10 15

### 数字方法

1. 数字转字符串 var nu = str.toString()
2. 字符串转数字var str = **str.toFixed(4**) 保留4位小数 适用于处理金钱
3. 把变量转换成数值
4. Number() 方法,参数转化，返回数字，比如日期转化Number(new Date("2019-04-15")); // 返回 1506729600000
5. **parseInt()** 返回整形 [pɑːrs]
6. parseFloat() 返回浮点数

注：无法转换的如有空格的字符，返回NaN

|  |  |
| --- | --- |
| **属性** | **描述** |
| MAX\_VALUE | 返回 JavaScript 中可能的最大数。 |
| MIN\_VALUE | 返回 JavaScript 中可能的最小数。 |
| NEGATIVE\_INFINITY | 表示负的无穷大（溢出返回）。 |
| NaN | 表示非数字值（"Not-a-Number"）。 |
| POSITIVE\_INFINITY | 表示无穷大（溢出返回）。 |

### 数组方法

**var fruits = ["Banana", "Orange", "Apple", "Mango"];**

**var points = [33, 23, 32, 66, 93, 4, 67]**

1. 数组转化成字符串 .toString()

document.getElementById("demo").innerHTML =

fruits.toString();

返回：Banana,Orange,Apple,Mango 以逗号隔开

1. 将所有数组元素转化为一个字符串 .join(“-”) 可设置分隔符
2. **删除最后一个数组元素：.pop()**

var fruits = ["Banana", "Orange", "Apple", "Mango"];

var a = fruits.pop(); a = Mango //也可以直接用fruits.pop()

1. **添加数组元素：.push()**

var a = fruits.push(“Kiwi”) //返回数组长度 a = 5

1. 位移方法删除：.shift() 数组元素向左移，即删除第一个数组元素

var a = fruits.shift(); a = Banana

1. **反向位移添加：.unshift()** 向右位移，即添加数组元素到第一位，后面位置顺移

var a = fruits.unshift(“kiwi”); a = 5 返回数组长度

1. 修改数组元素：fruits[0] = “apple”;
2. 简易添加新元素：fruits[fruits.length] = “kiwi” 效果同 .push()，但不会返回数组长度
3. **删除数组元素：delete.fruits[0]**

注：该方法删除会给数组留下未定义的空洞，可用pop()和shift()代替

1. **拼接数组**：可以批量替换、添加、删除数组元素 **.splice()**

fruits.splice(2, 0, “lemo”, “kiwi”) 从第二个元素后面插入两个元素

fruits.splice(2, 2, “lemo”, “kiwi”) 替换第二个元素后面的两个元素

fruits.splice(2,2) 删除第二个元素后面的两个元素

1. **合并（连接）数组**: .concat() 可连接多个数组

var tool = [“car”,”bike”]

tatal = fruits.concat(tool)

也可以添加元素，即数组和值合并

var total = fruits.concat(“car”,”bike”)

1. **数组切片**：.slice()

var newFruit = fruits.slice(1,3) 输出Orange,Apple

1. **数组排序**：.sort() && .reverse()
   1. fruits.sort() 对元素值进行排序
   2. fruits.reverse() 对元素值位置进行反转

注：用sort()对数字排序会有异常，需要添加为sort(function(a, b) {return a - b}),这样将会进行降序排序，return b – a 则是升序

随机排序：fruit.sort(function(a, b) { return 0.5 – Math.random()})

* 1. 输出数组最大、最小值：Math.max.apply(null, points)

document.getElementById("demo").innerHTML = Math.max.apply(null, points);

document.getElementById("demo").innerHTML = Math.min.apply(null, points);

1. 数组元素查询 array.indexOf() 字符串中有str.indexOf() 集合中有set.has()

若未查到返回-1，若查到返回索引值

1. **数组迭代**（遍历）：
2. .forEach()方法：会为每个数组元素调用一次函数

var arry = [1, 3, 4]

arry.forEach(function(value){ console.log(value \* 2) }) // 2; 6; 8

1. **.map()方法**（常用）：

map()对每个数组元素执行函数来创建**新数组**，不会对没有值的数组元素执行函数，不会更改原始数组。但还是一个浅拷贝

var test1 = [1, 3, 4, 6, 9]

var test2 = test1.map(funcTest)

function funcTest(value){ return value \* 2}

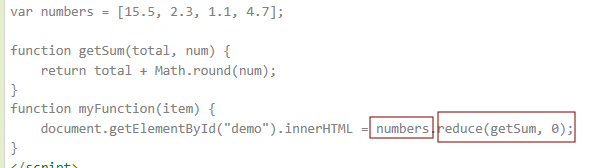
1. array.reduce() ：

接收一个函数作为**累加器**，数组中的每个值（从左到右）开始缩减，最终计算为一个值。空数组不会执行回调函数。

语法：

array.reduce(function(total, currentValue, currentIndex, arr), initialValue)

其中total和currentValue是必须的。



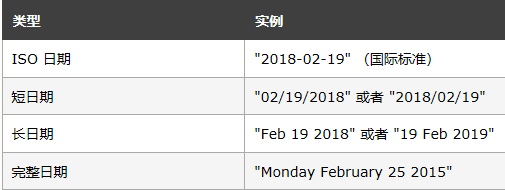
上例为四舍五入求和，其中initialValue是传递给函数的初始值

### 日期

var nowData = new Data() 用当前日期和时间创建新的日期对象：

var nowData = new Data(2019, 10, 21, 10, 49, 0) 最后一个是毫秒

日期输入格式：



日期获取方法：var d = new Date(); d.getDate(); or new Date().getDay()



日期设置方法：var d = new Date();

d.setFullYear(2020);



日期格式化：



### 数学方法

1. Math.PI; 输出圆周率
2. **Math.round(23.78);** 取整，四舍五入24
3. Math.pow(x, y); 返回x的y次幂
4. Math.sqrt(64); 返回平方根 8
5. Math.abs(-12); 返回绝对值12
6. **Math.floor(23.78)**; 取整，直接抹去小数位置 23
7. Math.sin(); 正弦值

document.getElementById("demo").innerHTML = Math.sin(Math.PI/2); 1

1. Math.max() & Math.min() 找出最大最小值

数组里面找要用Math.max.apply(null, arry)

1. **Math.random()** 返回0到1 的随机数

Math.floor(Math.random() \* 10); 返回0-9的随机数

Math.floor(Math.random() \* 11)；返回0-10的随机数

输出某个范围的随机数：[min,max)

function getRndInteger(min, max) {

return Math.floor(**Math.random() \* (max - min)** ) + min;

}

输出某个范围的随机整数:[min,max]

function getRndInteger(min, max) {

return Math.floor(**Math.random() \* (max - min) + 1** ) + min;

}

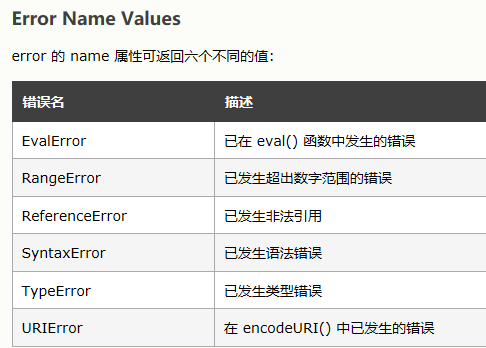
其他数学方法：[点击跳转](#_数学方法表)

## 异常处理 try..catch…finally



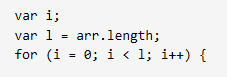
捕捉异常





## 性能

1. 减少循环中的活动





1. 减少DOM访问



若要访问多次，就把它做成本地变量保存

还需要尽量减少HTML DOM中的元素数量

1. 避免不必要的变量

可以直接输出的，就不要声明一个变量区保存

1. 延迟js加载

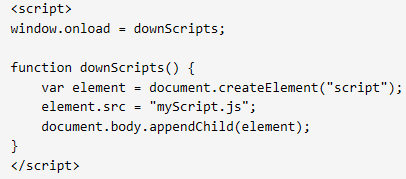
把脚本放在页面底部，使浏览器先加载页面

脚本在下载时，浏览器不会启动任何其他的下载。此外所有解析和渲染活动都可能会被阻塞。

HTTP 规范定义浏览器不应该并行下载超过两种要素。

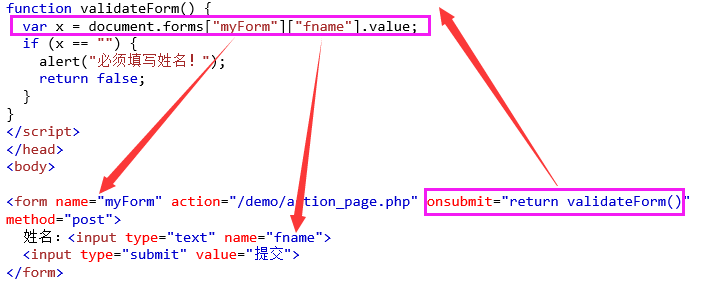
一个选项是在 script 标签中使用 defer="true"。defer 属性规定了脚本应该在页面完成解析后执行，但它只适用于外部脚本。

如果可能，可以在页面完成加载后，通过代码向页面添加脚本：

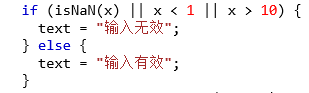


1. 不适用with

## 表单 控制输入内容



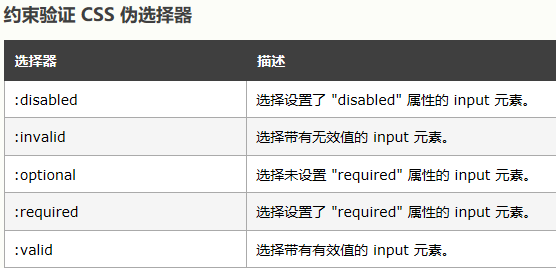
若是空表单提交则会弹窗提示

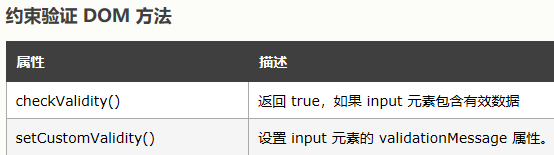


自动HTML表单验证：<input type="text" name="fname" **required**> IE9及以前不可用

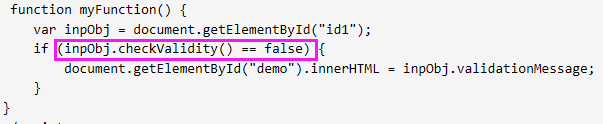


<input type="number" name="test" max="100" min="10" required>





<input id="id1" type="number" min="100" max="300" required>





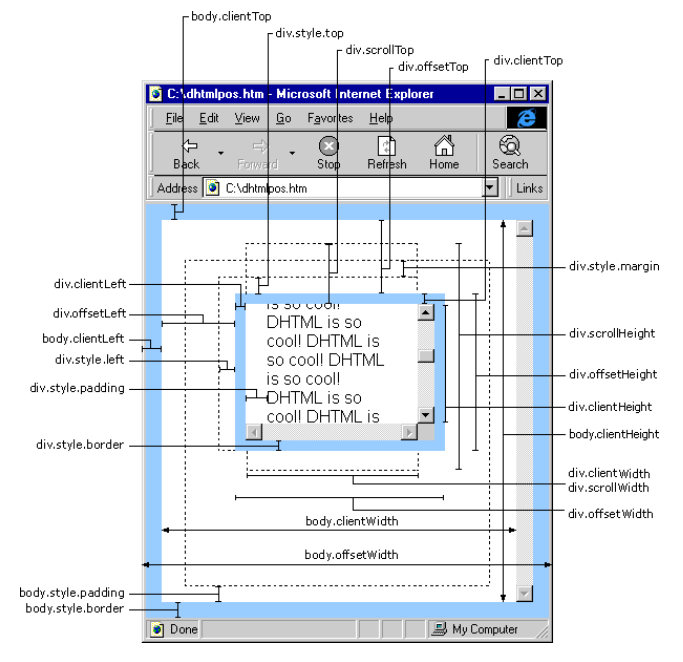


例：<input id="id1" type="number" max="100">



当输入元素大于HTML的max规定元素值，输出txt

## DOM（文档对象模型）



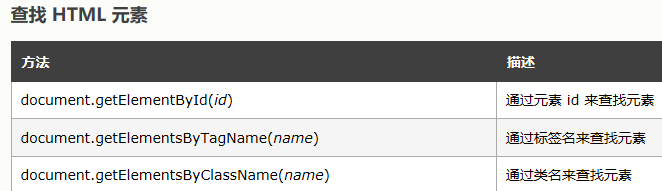
“W3C 文档对象模型（DOM）是中立于平台和语言的接口，它允许程序和脚本动态地访问、更新文档的内容、结构和样式。”

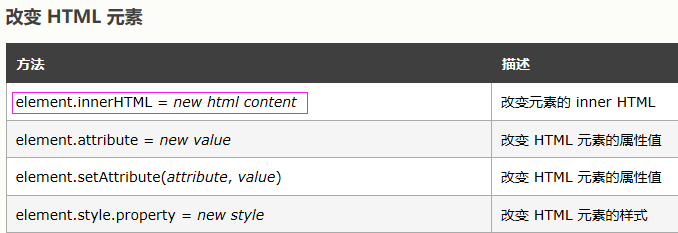
HTML DOM方法：能够（html元素上）执行的动作

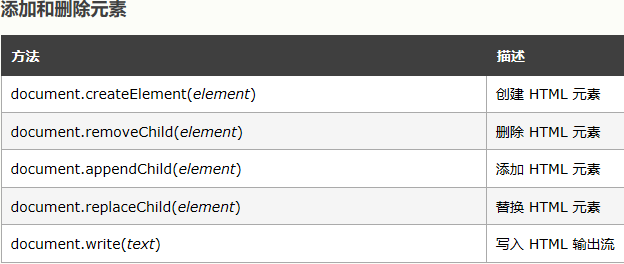
HTML DOM属性：能够设置、改变的HTML元素的值

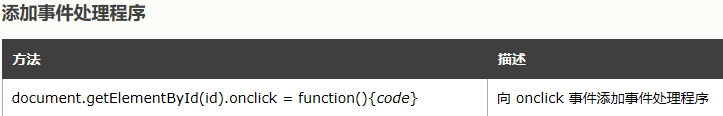
document.getElementById("demo").innerHTML = "Hello World!";

其中getElementById 是方法，innerHTML是属性









### 改变HTML 获取内容

1. 改变HTML输出流：

document.write() 可用于直接写入 HTML 输出流：document.write(Date()); 不能在文档加载后使用，会被覆盖

1. 改变HTML内容

document.getElementById(id).innerHTML = new text

1. 改变HTML属性值

document.getElementById("myImage").src = "landscape.jpg";

1. 获取元素中的内容

var a = document.getElementById("myImage").innerText/HTML

1. 获取用户输入的内容

var a = document.getElementById("myImage").value

### 改变css

1. 改变HTML样式：

document.getElementById("p2").**style**.color = "blue";

通过点击触发内容颜色变化：

<button type="button" onclick="document.getElementById('id1').style.color = 'red'">

1. 获取、改变属性

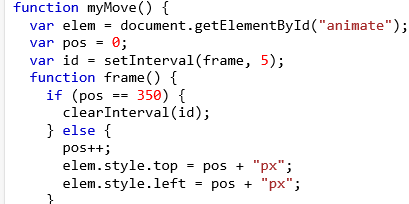
getAttribute setAttribute removeAttribute

var p = document.getElementsByTagName("p")[1].getAttribute("class")

document.setElementsByTagName("p")[1].getAttribute("class","xxx")

注： 必须要是标签内的属性，而不是css中的属性

### 动画：



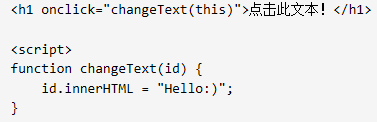
注1：setInterval()方法可用来指定周期(毫米),来调用函数或者表达式

如上为每个5毫秒执行一遍frame()函数，终止则是用clearInterval(id)

该动画便是每5毫秒，图案便会向右和向下移动一个像素

### 对事件作出反应：鼠标、键盘、html事件

1. 例1：当用户点击鼠标时 onclick属性是鼠标点击时生效

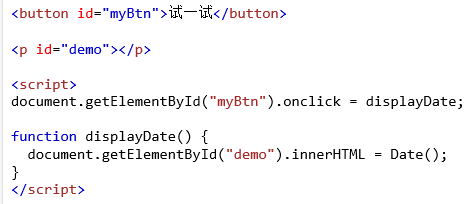


鼠标点击文本后，字样将变成Hello:)

1. 使用html dom分配事件

例2：先给button元素添加onclick属性，属性值为displayDate

效果为，生成按钮，点击按钮获得当前时间。



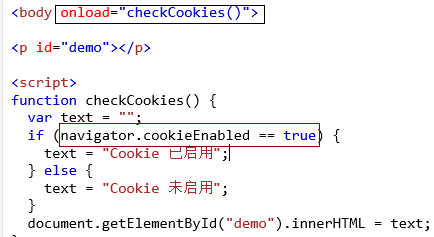
1. onload & onUnload

当户进入或者离开页面的时候，会触发onload和onunload

onload 事件可用于**检测访问者的浏览器类型和浏览器版本**，然后基于该信息加载网页的恰当版本。

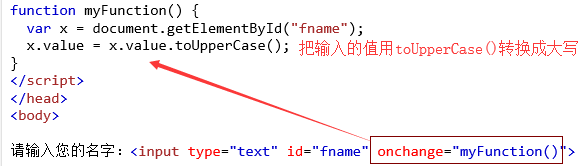
onload 和 onunload 事件可用于**处理 cookie**。

<body onload="checkCookies()">



1. onchange事件

onchange事件经常与输入字段验证结合使用

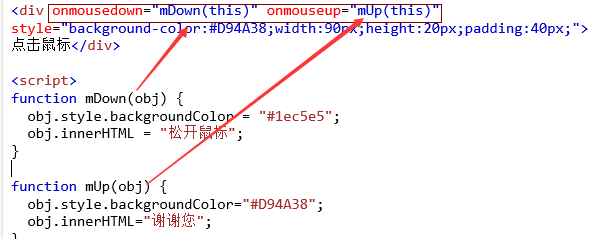


1. onmouseover & onmouseout事件

onmouseover 和 onmouseout 事件可用于当用户将**鼠标移至 HTML 元素上或移出时**触发某个函数：



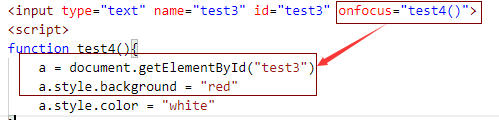
1. onmousedown & onmouseup



注1：将属性换成src = “xx.gif”，可以点击时改变图片

1. 关于<input>, <select>, 和<a>.中的onfocus元素

通常在事件获得焦点时发生，标准写法是：



### 事件监听 addEventListener() removeEventListener()

添加当用户点击按钮时触发的事件监听器：

document.getElementById("myBtn").addEventListener("click", function(){ alert("点击无效！"); });

注：可以同时添加多个监听器，会在你关掉一个后按顺序弹出

**事件冒泡**和**事件捕获**

假如 <div> 元素内有一个 <p>，然后用户点击了这个 <p> 元素

在**冒泡**中，**最内侧元素**的事件会首先被处理，然后是更外侧的：首先处理 <p> 元素的点击事件，然后是 <div> 元素的点击事件。

在**捕获**中，**最外侧元素**的事件会首先被处理，然后是更内侧的：首先处理 <div> 元素的点击事件，然后是 <p> 元素的点击事件。

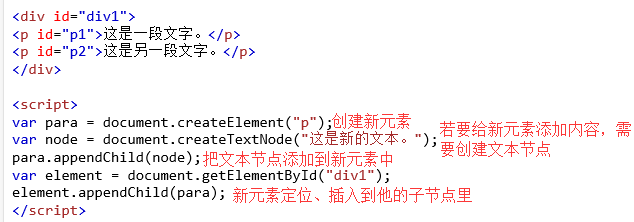
默认值是 false，将使用冒泡传播，如果该值设置为 true，则事件使用捕获传播



注：ie9及其以上

### 创建、删除、替换新元素（节点）

1. 创建新元素并添加 appendChild() insertBefore()



注：appendChild()方法只能添加到最后一个子元素

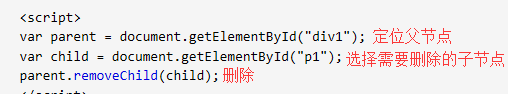
insertBefore()： 最后改成：通过id定位，可以添加到第一个子元素的前面。

var child = document.getElementById("p1");

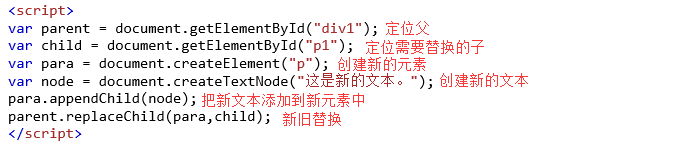
element.insertBefore(para,child);

不用插入到某个节点之后，因为可以找到那个节点下一个节点，插入下一个节点之前，如果后面没节点，则直接插入父节点的子节点，默认最后

1. 删除 removeChild()



1. 替换 replaceChild(new, old)



### 节点集合、节点列表

1. 节点集合：var x = document.**getElementsByTagName**("p");

x是所有p元素的集合 x[0].innerHTML x.length等获取集合内元素信息

例1：document.getElementById("demo").innerHTML = x[0].innerHTML

例2：for (var i = 0; i < x.length; i++) {

myCollection[i].style.backgroundColor = "red";

} 循环将所有的p元素内容的背景颜色改为红色

1. 节点列表：var x = document.**querySelectorAll**("p");

使用方法同节点集合

该方法可以使用选择器进行精准选取：

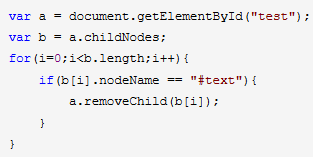
document. **querySelectorAll**("div p.className")

注：节点列表几乎等同于节点集合；节点列表不是数组，不能push(),pop(),join()

### 获取元素子节点

1. var child = document.getElementById("id1").childNodes;

上面的方法通过child[i]进行子元素遍历，根据dom标准，标签间的回车空格等特殊字符属于文本节点；所以如果子元素是有回车，则会占一个节点；所以我们需要过滤空白节点。**不推荐使用**



1. var child = document.getElementById("id1").children;

无上述问题，且可以通过child[i].style 操作css元素也可以通过setAttribute进行属性操作。推荐使用

1. var p\_list = document.querySelectorAll("#id1 p");

最推荐使用，和第二种相同效果，支持各种选择器，在批量获取节点元素时首先考虑。

## BOM(浏览器对象模型)

允许js和浏览器对话

### window对象：

代表浏览器窗口，全局变量是win对象的属性，函数是win对象的方法

DOM对象也是win对象的属性：

window.document.getElementById("header")；

等同于document.getElementById("header");

window.onload = function(){代码块}

文档加载完成后能立即触发，并且能够为该事件注册事件处理函数。

### 窗口尺寸

window.innerHeight - 浏览器窗口的内高度（以像素计）

window.innerWidth - 浏览器窗口的内宽度（以像素计）

window.open() - 打开新窗口

window.close() - 关闭当前窗口

window.moveTo() -移动当前窗口

window.resizeTo() -重新调整当前窗口

### window screen

属性有：可不带window

screen.width：返回以像素计的访问者屏幕宽度

document.getElementById("demo").innerHTML = "Screen Width: " + screen.width; Screen Width: 1600

screen.height

screen.availWidth

screen.availHeight

screen.colorDepth 色深 返回表示颜色的比特数

screen.pixelDepth 像素深度，等同于颜色深度

### window.location

location对象可用于获取当前页面的地址(url)，并把浏览器重定向到新页面

window.**location.href** 返回当前页面的 href (URL)

window.location.hostname 返回 web 主机的域名

window.location.pathname 返回当前页面的路径或文件名

window.location.protocol 返回使用的 web 协议（http: 或 https:）

window.**location.assign** 加载新文档

window.location.reload 重载该页面

window.location.replace 替换该url

注：可以不用写window



### window.history

对象包含浏览器历史记录

**history.back()** 等同于在浏览器点击回退按钮

**history.forward()** 等同于在浏览器点击前进按钮

history.go(n) n为0重载页面；n为正，往前进n个页面；n为负，后退n个页面



H5增加的新特性：

history.length

history.state

history.pushState(data, title, url)：改变网址(也有跨域限制)不刷新页面，vue中的vue-router和react-route-dom; 仅仅改变网址，不会跳转、获取新内容，本质上还是在原网页

history.replaceState(data, title, url)，同上只是替换url。不会在history留下记录

### window.navigator()

对象包含有关访问者的信息

navigator.appName 浏览器应用名称

返回"Netscape" 是 IE11、Chrome、Firefox 以及 Safari 的应用程序名称的统称

navigator.appCodeName 浏览器应用程序代码名称

返回"Mozilla" 是 Chrome、Firefox、IE、Safari 以及 Opera 的应用程序代码名称。

navigator.platform 返回浏览器平台(操作系统)

navigator.cookieEnabled 返回ture,则cookie已启用

navigator.appVersion； navigator.product； navigator.userAgent；navigator.language；navigator.onLine

navigator.userAgent: 其中有浏览器的名称

navigator.language | navigator. browserLanguage 获取当前浏览器系统语言

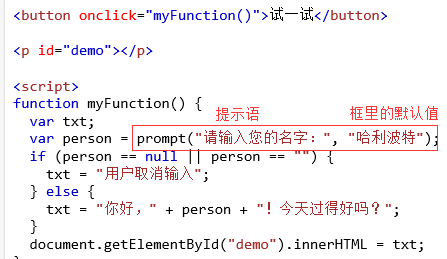
### 弹出框

告警框：window.alert() 可省略window alert(“hello /n world!”) 加/n会换行

确认框：window.confirm()

希望用户验证或接受某个东西，则通常使用“确认”框，可选确认或取消

提示框：window.prompt()



1. timing 事件，每个一段时间执行一次函数
   1. window.**setTimeout**(*function*, *milliseconds*);
   2. window.**setInterval**(*function*, *milliseconds*);

例：点击按钮3s后执行函数，用clearTimeout()停止运行

<button onclick="myVar = setTimeout(myFunction, 3000)">试一试</button>

<button onclick="clearTimeout(myVar)">停止执行</button>

计时器,定时器

## AJAX 和 跨域

AJAX是一种从网页访问web服务器的技术，它代表**异步**的js和XML

作用：1. 不刷新页面更新网页；2. 在页面加载后从服务器**请求数据**；3. 在加载页面后从服务器**接收数据**；4. 在后台向服务器**发送数据(注册)**

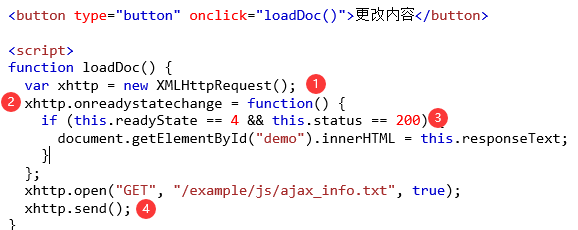
Ajax组合了：  
1. 浏览器内建的XMLHttpRequest对象（从web服务器请求数据）

2. js和HTML DOM显示数据

工作流程：网页发生一个事件(按钮点击，页面加载)-- >js创建XMLHttpRequest对象-->XMLHttpRequest对象向web服务器发送请求-->服务器处理该请求-->服务器将响应发送回页面-->js读取响应-->js执行正确的动作

所以Ajax的核心是XMLHttpRequest

创建XMLHttpRequest对象： *variable* = new XMLHttpRequest();



1. 创建XMLHttpRequest对象；
2. 监听请求**onreadystatechange**属性定义readystate变化时调用函数； 函数内判断readyState状态是否是4(请求完成，响应就绪)和返回请求的状态是否OK（200）,然后resposneText用字符串返回响应数据；

监听请求必须在发送请求前，因为必须保证监听先绑定后再发送信息并收到信息返回。

1. open()规定请求（请求类型GET，文件位置，异步），同步不推荐

send()将请求发送到服务器。**打开对象，发送请求**

**请求：GET POST**

**get的请求数据是放在url中，post请求是放在请求体中；这是本质的不同，其他不同都是从这种数据存放地方不同导致的。**

向服务器发送请求用open()和send()方法

GET比POST方法更快，但使用POST的场景：

1. 缓存文件不是选项(更新服务器上的文件和数据库)
2. 向服务器发送大量数据(post无大小限制)
3. 发送用户输入，更安全

GET请求：

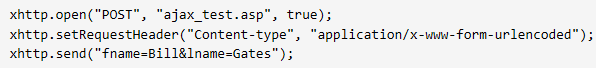


加上唯一的id不会获得一个缓存的结果



用GET方法发送信息

POST请求：



如需发送HTML表单这样的post数据，需要用setRequestheader()添加HTTP头部

send()方法中规定发送数据

XMLHttpRequest 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| **new XMLHttpRequest()** | 创建新的 XMLHttpRequest 对象 |
| abort() | 取消当前请求 |
| getAllResponseHeaders() | 返回头部信息 |
| getResponseHeader() | 返回特定的头部信息 |
| **open(*method*, *url*, *async*, *user*, *psw*)** | 规定请求   * method：请求类型 GET 或 POST * url：文件位置 * **async：true**（异步）或 false（同步） * user：可选的用户名称 * psw：可选的密码 |
| **send()** | 将请求发送到服务器，用于 GET 请求 |
| **send(*string*)** | 将请求发送到服务器，用于 POST 请求 |
| setRequestHeader() | 向要发送的报头添加标签/值对 |

XMLHttpRequest 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| **onreadystatechange** | 定义当 readyState 属性发生变化时被调用的函数 |
| **readyState** | 保存 XMLHttpRequest 的状态。   * 0：对象已创建，请求未初始化 * 1：服务器连接已建立，已经调用open * 2：请求已收到, 已发出ajax请求 * 3：正在处理请求，已返回部分数据 * **4：请求已完成且响应已就绪，数据已全部返回** |
| **responseText** | 以字符串返回响应数据 |
| responseXML | 以 XML 数据返回响应数据 |
| **status**  **200-300为正常** | 返回请求的状态号   * 200: "OK" * 403: "Forbidden" * 404: "Not Found"   如需完整列表请访问 https://www.runoob.com/http/http-status-codes.html |
| statusText | 返回状态文本（比如 "OK" 或 "Not Found"） |

**xml文件交互：**

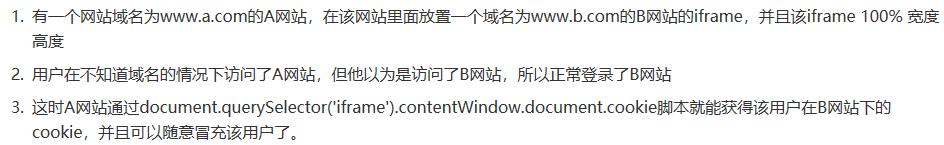
例：获取构建 HTML 表格，从 XML 文件提取节点（因素），最后使用由 XML 数据填充的 HTML 表格对元素“demo”进行更

### 跨域问题：

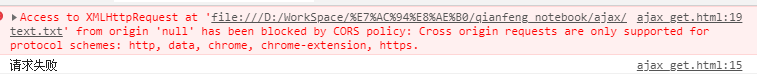
1. **域**：当前URL路径和要访问的目标服务器的地址是**同一个协议**+**域名**+**端口**

三者任意一种不满足都会产生跨域问题，违背同源策略。

同源原则：是为了保护网站的安全，防止用户信息泄露，防止身份伪造等(读取Cookie)



1. **安全角度**：浏览器会限制从脚本发起的跨域HTTP请求。比如遵守同源策略的“XMLHttpRequest”对象，简称**XHR**。这意味着使用XHR的Web应用程序只能从应用程序（如Ajax）当前的域请求HTTP资源，或者资源请求成功但**被浏览器拦截。**



https://img-blog.csdnimg.cn/20190829232428372.png

需要在服务器端给请求头添加“**Access-Control-Allow-Origin**”字段，来允许浏览器跨域请求，实现跨域资源共享。

总结：发生跨域安全问题条件：1.浏览器限制，后端没有限制，会正常响应；浏览器会对跨域请求做检查；2. **协议**+**域名**+**端口**三要素必须相同；3. 针对XHR（XMLHttpRequest）请求（ajax请求），如果不是，不会发生该问题。

1. **跨域资源共享：**跨域资源共享是一种机制，它使用**额外的 HTTP 头(Access-Control-Allow-Origin)**来告诉浏览器 ，让运行在一个 origin (域) 上的Web应用被准许跨域访问目标服务器上指定的资源。
2. **跨域问题解决方案：**
   1. **线上出现跨域问题：**
      1. 后端添加配置Allow-access-origin: ‘\*’ (开发实际中选用)
      2. jsonp只支持get请求，如果是post请求，依然会当成get请求，并把请求信息放在url中
   2. **本地出现(开发环境):**
      1. charles和fiddler工具进行请求转发
      2. **webpack的webpack-dev-server; 内置代理工具**
      3. vue.config.js

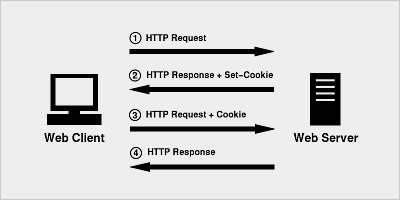
对于淘宝、百度、网易云音乐之类配置公共的api，会用jsonp

公司会用nginx反向代理

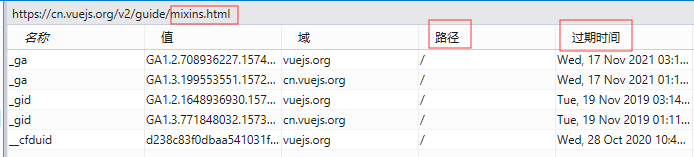
## cookie

HTTP协议本身是无状态的，所以无论请求多少次，都会成功；无状态的服务器无法判断用户。

Cookie实际上是一小段的**文本信息（key-value格式）**。客户端向服务器发起请求，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。(比如登录优酷之后，一定期限内，打开浏览器将会自动登录，不需要再输入密码)



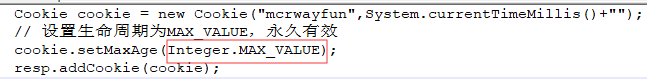
(2)的时候，服务器发送响应和set-cookie头部，客户端保存cookie，(3)当下次再请求时会将该cookie一并发送过去，此时服务器就知道是谁请求的了。



属性值：



**Expires**设置cookie有效期，Cookie中的maxAge用来表示该属性，单位为秒。Cookie中通过getMaxAge()和setMaxAge(int maxAge)来读写该属性。maxAge有3种值，分别为正数，负数和0。**为正时**，在规定秒数之后自动失效，在这其中无论是关闭浏览器或者电脑，cookie依然有效；**为负数时**，不会被持久化，即关闭浏览器窗口后失效；**为0时**，表示立即删除cookie。

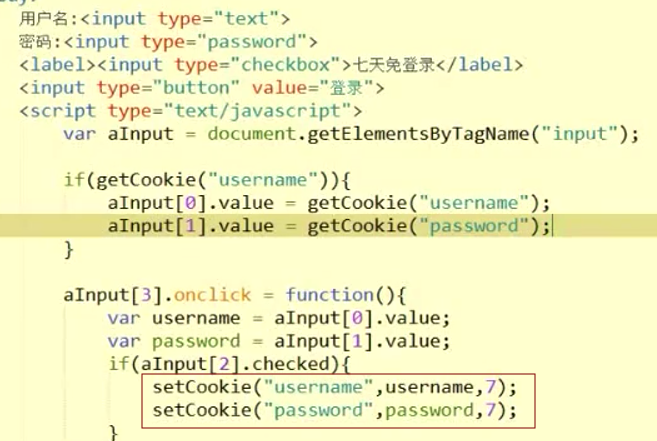


**修改cookie值**：通过同名覆盖的方法进行修改，详细说应该是name、path、domain相同后可以覆盖，如果删除，只需将过期时间进行修改。



**cookie无法跨域名**，隐私安全机制禁止网站非法获取其他网站的Cookie；即便一级域名下的两个域名a.text.com和b.text.com的cookie也都无法相互访问; 若要访问，domain需要修改为.text.com

cookie路径：path属性决定允许访问cookie的路径。一般设置为”/”，表示所有路径都可以访问。



设置7天免登录，即保存登录用户名密码7天后过期。

判断用户是否登录：

通过JS写cookies或者localStorage,**sessionStorage**，并且这个cookies也要保存一份到数据库里面（也可以后台数据库中添加标识并进行判断）。打开网页的时候，要读浏览器的cookie值，并和数据库的cookie进行比较判断，结果为true就是登陆了，否则没有登陆。此外，你还可以以此为基础，增加过期的时间。

## ES6

ES6介绍：

ECMAScript 6.0 是js的下一代标准；让js可以编写出复杂的大型应用程序

### let和const

let: 存在于块级作用域；不存在声明提升；不予许重复声明

const：常量声明，不能提前、不能重复声明，但是对于一些结构复杂比如：数组、对象时，虽然不能重新赋值，但是可以对里面的子元素进行增删改查。

### 函数：(1)箭头函数和this (2)函数默认参数

es5: var a = function(){return 1;}

es6: let a = () => 1;

console.log(a()); 都返回1

let b = (n) => (n-1) + (n-2)

多条语句：

let c = (n) => { let d = 1000;

return n \* d };

es6中this的指向问题；**this指向定义时所在的作用域，而不是执行时的作用域**

在函数中this指向window; 在对象中同样指向window，同es5不同

所以，箭头函数适用于各种回调函数，不适用于处理对象中含有this的方法，和需要动态this的情况，因为this指向window，所以this不会变化

可以指定函数默认参数：

var a = function(num = 200){}

若有传参使用所传的参数，若没有使用num = 200；

代替在函数中写入num = num || 200;

不定参数：function myFunction(…values){}

传入的参数都作为values数组的形式，通过for循环遍历

效果同arguments

### 解构赋值

数组的解构：

let [a, b, c] = [1, 2, 3];

可嵌套：let [a, [[b], c]] = [1, [[2], 3]]; 只要格式能对应上就行

可忽略、省略：let [a, , c] = [1, 2, 3]; let [a, …c] = [1, 2, 3];

c作为一个数组返回剩余参数

字符串解构：let [a, b, …c] = 'hello';

解构默认值设置：let [a, b=2, c=3] = ['x', 'y']; //x y 3

对象解构：同上，但是key值必须要对应且相等

1. **for of循环**

for(var i in obj)

for(var j of arry) 用于集合和数组，in不能遍历集合

### import、export导入导出 模块化开发

全部导入：import people from './example'

等价于：import {default as people} from './example'

特殊情况：整个模块当作**单一对象**进行导入

import **\* as example** from "./example.js"

调用： console.log(example.name, example.age)

部分导入：import {name, age} from './example'

默认导出：export default people 就用import people from … 导入

部分导出：export class People extend Component {};

注：1.当用export default people导出时，就用 import people 导入（不带大括号）

2.一个文件里，有且只能有一个export default。但可以有多个export。 3.当用export name 时，就用import { name }导入（记得带上大括号） 4.当一个文件里，既有一个export default people, 又有多个export name 或者 export age时，导入就用 import people, { name, age } 5.当一个文件里出现n多个 export 导出很多模块，导入时除了一个一个导入，也可以用import \* as example

1. **…展开运算符[… set(array)] 去重**

var new\_arr = [...new Set(arr)]

这里的set对象也是es6新特性

### class类的继承

见对象和继承篇的例子

class name {

constructor(x, y, z){

this.x = x …

}

myFunction(){…}

}

继承：extends super

class newClassName extends oldName {

constructor(x, y, z, a, b){

super(x, y, z);

this.a = a

}

}

### async、await

1. async 用于申明一个 function 是异步的，而 await 用于等待一个异步方法执行完成。
2. await必须在含有async的函数里面才有效

使用：**async** **function** **testAsync**() { **return** "hello async";}

console.log(testAsync()) // Promise{ 'hello async' }

返回的是一个promise对象，所以不能用await在外层进行取值；需要使用.then()：

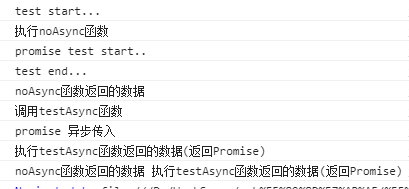
testAsync().then((res)=>{console.log(res)})可取值

到此其实和promise没什么区别，都是立即执行，在then中进行异步

await的使用：他需要在有async的函数中，它使用的函数需要返回promise对象，所以那个函数可以是有async或者返回数据被new Promise()包裹

1. async/await 是一种编写异步代码的新方法。之前异步代码的方案是回调和 promise。
2. async/await 是建立在 promise 的基础上。
3. async/await 像 promise 一样，也是非阻塞的。
4. async/await 让异步代码看起来、表现起来更像同步代码。这正是其威力所在。





注：在执行test()后，输出test start; 调用noAsync()函数，输出里面的console.log(),而await等到了return的值(可是promise也可是其他值)，让出线程，跳出整个test()函数进行等待；执行后面的同步事件，后面是创建了一个promise对象实例，但是构造函数是立即执行的，所以会输出里面的两个console.log()，而.then的部分也会跳出进行异步。

此时同步已全部执行完毕，开始执行最先跳出的异步，即noAsync()函数的返回值；然后是testAsync()函数，执行完console.log后返回的是promise对象，再次跳出进行异步等待，执行promise的.then的部分；此时整个流程再次执行完，之后再执行testAsync()的返回部分，最后异步同时输出两个函数的结果

### map对象和set对象

map和object的区别：map的键可以是任意值：字符串、对象、函数都行

map中的键值是有序的，且它的长度可以通过size属性获取

使用：

var myMap = new Map();

设置：myMap.set(“string”: “键可为字符串、对象、函数、nan”)

获取：myMap.get(“string”)

遍历：for (var [key, value] of myMap){}

myMap.forEach(function(value, key) { console.log(key + " = " + value); }, myMap)

转换：可以将**二维**数组转换成map: new Map(kvArray)

{"key1" => "value1", "key2" => "value2"}

也可通过Array.from(myMap)或者[…myMap]转换成二维数组

克隆：myMap1 = new Map(myMap)

合并：myMap2 = new Map([…myMap, …myMap1])

若有键重复，后面的值会覆盖前面的值

Set 对象存储的值总是**唯一**的，所以需要判断两个值是否恒等。有几个特殊值需要特殊对待：

1. +0 与 -0 在存储判断唯一性的时候是恒等的，所以不重复；
2. undefined 与 undefined 是恒等的，所以不重复；
3. NaN 与 NaN 是不恒等的，但是在 Set 中只能存一个，不重复

数组去重：var new\_arr = [...new Set(arr)]

var a = new Set([1, 2, 5]); var b = new Set([1, 3, 5])

并集：var mySet = new Set(…a, …b)

交集：var intersect = new Set([...a].filter(x => b.has(x)))

set.size 返回集合长度

set.delete(0) set.add(‘a’) set.has() 查询有没有元素true false

set.clear()清空

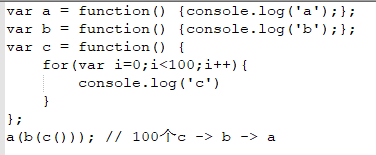
map同set

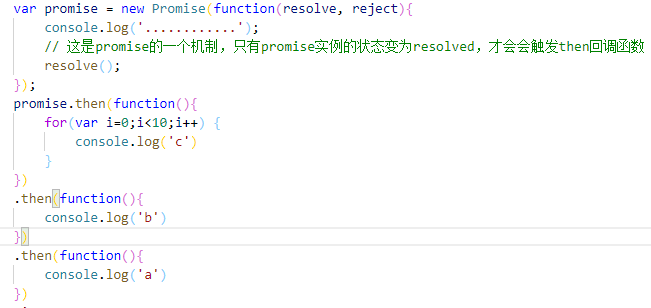
### promise（重要）

同步编程的方式来编写异步代码，保存线性的代码逻辑，极大的降低了代码耦合性而提高了程序的可扩展性。

异步编程的一种解决方案

没有promise实现异步

回调地狱



此时将你想要的执行的顺序放入对象的then中

从语法上说，**Promise 是一个对象**，用它**可以获取异步操作的消息**。

Promise的构造函数接收一个参数，该参数是一个函数，并且函数中传入两个参数：**resolve,reject**; 分别表示为异步操作执行**成功**后和**失败**后的回调函数。按标准将resolve将Promise的状态置为fullfiled, reject则置为rejected。

**了解pormise主要是了解同步和异步的关系**

console.log(‘a’)

let a = new Promise(function (res, rej){ res() console.log(‘b’) })

a.then(function () { console.log(‘c’) } )

console.log(‘d’)

最后的结果是: a b d c

注：promise的构造函数中还是同步的，只有then里面的是异步的

异步就是所有同步执行完之后才执行异步

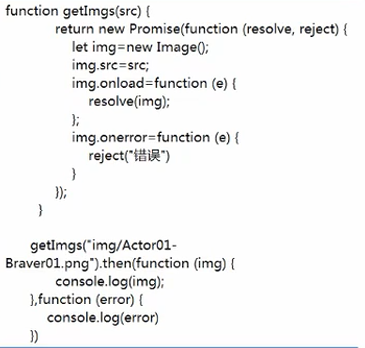
promise状态(三种)：未完成（unfulfilled）、完成（fulfilled）、失败（failed）。

而resolved代表fulfilled，使用rejected表示fail。

特性：

1. 状态只能从未完成到完成和未完成到失败，不可逆
2. promise的异步结果，只能在完成状态时才能返回，而且我们在开发中是根据结果来选择来选择状态的，然后根据状态来选择是否执行then()。
3. 实例化的Promise内部会立即执行，then方法中的异步回调函数会在脚本中**所有同步任务完成时才会执行**。因此，promise的异步回调结果**最后输出**

例:



getImgs().then(function (img){

console.log(img)

}).catch(function(error){

console.log(error)

}) // 两种写法结果一样，但多用该种。

img.src=src放在img.onload后面

该图为创建一个函数，传入src地址后，只有当img.onload，即浏览器图片缓存之后；如果存在这张图片则在console台上返回<img>；没有则会报错。

首先let img = new Image(); // 图片预加载

img.onload = function () {}

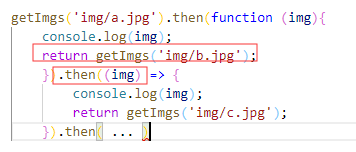
img.src = src; // 需要放在onload后面，不然ie会报错

其中第一步定义Image对象的src: img.src=”xxx.gif”; 这样做就相当于给浏览器缓存了一张图片（直接将第一步执行后执行第三步，通过dom添加child节点，就能看到该图片；总的说就是创建一个<img scr=’xxx’>）

在创建Image对象后进行图片加载，加载完之后异步调用resolve(), 并将img.res赋值，然后执行then里面调用成功后的函数打印图片。

下面调用时，then中两个函数，一个成功后执行，另一个失败后执行。

**链式异步加载：（可以完成多个图片的异步顺序加载）**



该方法解决了开发过程中通过**多层嵌套**回调函数实现异步；那样不美观可读性差；(函数a作为参数传入函数b，则a就是b的回调函数)

**多个图片异步并行加载：Promise.all()**



其中getImages()就是一个new Promise(…)

应用场景：一个页面上需要等两个或多个ajax的数据回来以后才正常显示，在此之前只显示loading图标。

注：需要特别注意的是，Promise.all获得的**成功结果的数组里面的数据顺序和Promise.all接收到的数组顺序是一致的**，这带来了一个绝大的好处：在前端开发请求数据的过程中，偶尔会遇到发送多个请求并根据请求顺序获取和使用数据的场景，使用Promise.all毫无疑问可以解决这个问题。

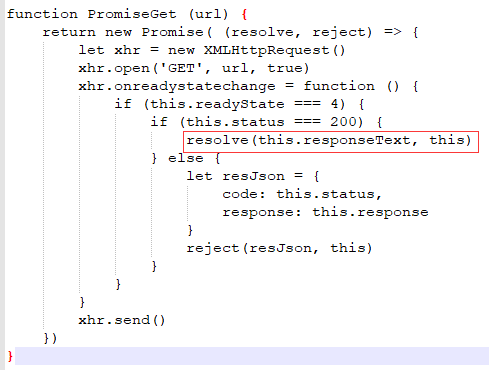
**Promise.race()**

let p1 = new Promise(…); let p2 = new Promise(…);

Promise.race([p1, p2]).then((result) => {console.log(result)})

哪个结果获取得快，返回哪个结果，不管成功失败。实际不常用。

**ajax和promise结合**



### Generator函数

一种异步解决方案：

组成：(1) function后面带有\* (2)函数内部有yield表达式



调用:

var func = func()

func.next() // one

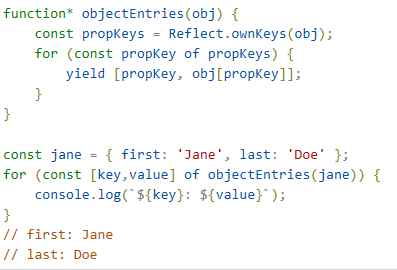
停止

func.next() // two

停止

。。。

使用场景：实现iterator，为不具备iterator接口的对象提供遍历方法

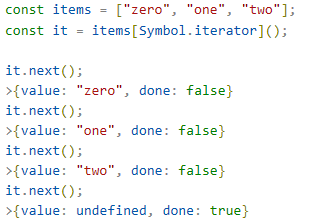


Reflect.ownKeys() 返回对象所有的属性，不管属性是否可枚举，包括 Symbol。

jane 原生是不具备 Iterator 接口无法通过 for... of遍历。这边用了 Generator 函数加上了 Iterator 接口，所以就可以遍历 jane 对象了。

### 迭代器Iterator

Symbol.iterator创建迭代器，初始指向数据结构起始位置，通过next()迭代指向下一个，返回一个对象{value: “值”, done: “false”}，done为true时遍历结束



可迭代对象：Array String Map Set Dom元素（正在进行中）

还可以使用for … of … 进行迭代

### symbol

新增数据类型，使其独一无二，用来定义对象的唯一属性名

let sy = Symbol("key1");

let obj = {}

obj[sy] = “hhhh” // {Symbol(key1): "kk"}

### proxy和 Reflect

是操作对象的api

Proxy 可以对目标对象的读取、函数调用等操作进行拦截，然后进行操作处理。它不直接操作对象，而是像代理模式，通过对象的代理对象进行操作，在进行这些操作时，可以添加一些需要的额外操作。

Reflect 可以用于获取目标对象的行为，它与 Object 类似，但是更易读，为操作对象提供了一种更优雅的方式。它的方法与 Proxy 是对应的。

### 字符串

alert(`${a.name}的年龄是${a.age}`) 用反引号

string.includes(‘apple’): 返回布尔值，判断字符串中是否含apple字符串

string.startsWith(): 头部是否含有string

string.endsWith(): 尾部是否含有string

string.indexof()也可以，但是返回的是索引值

repeat():将字符串进行重复

string.repeat(‘apple’.repeat(4)) // 将apple重复4次，可为0，负，小数

字符串补全

console.log("h".padStart(5,"o")); //"ooooh"

console.log("h".padEnd(5,"o")); // "hoooo"

console.log("h".padStart(5)); // " h"

## JSON

JavaScript 对象表示法（JavaScript Object Notation）。

JSON 是存储和交换文本信息的语法。类似 XML。

JSON 比 XML 更小、更快，更易解析。

JSON 使用 JavaScript 语法来描述数据对象，但是 JSON 仍然独立于语言和平台。JSON 解析器和 JSON 库支持许多不同的编程语言

var JSONObject= {

"name":"Bill Gates",

"street":"Fifth Avenue New York 666",

"age":56,

"phone":"555 1234567"};

document.getElementById("jname").innerHTML=JSONObject.name

可以是数组类型：

var employees = [

{ "firstName":"Bill" , "lastName":"Gates" },

{ "firstName":"George" , "lastName":"Bush" },

{ "firstName":"Thomas" , "lastName": "Carter" }

];

document.getElementById("jname").innerHTML=employees[0].lastName;

访问或者修改employees[0].lastName="kangkang";

JSON 最常见的用法之一，是从 **web 服务器上读取 JSON 数据**（作为文件或作为 HttpRequest），将 **JSON 数据转换为 JavaScript 对象**，然后在网页中使用该数据。

严格来说从服务器过来的并不是json,而是json格式的字符串。  
所以要通过eval()转换成 js中的 json 对象 才能用的

总结：获取到的是json字符串，js不能直接用；需要用eval转换成json对象才能用。

json字符串：var txt = '{"employees":[' +

'{"firstName":"Bill","lastName":"Gates" },' +

'{"firstName":"George","lastName":"Bush" },' +

'{"firstName":"Thomas","lastName":"Carter" }]}';

eval转换：var obj = eval ("(" + txt + ")");

转换之后才是和上面的数组类型一样；调用：

document.getElementById("jname").innerHTML=obj.employees[1].firstName

### JSON.stringify() & JSON.parse()

常用于在localStorage和sessionStorage中

因为两者中存储的只能是字符串，而我们实际需要的是对象类型。



所以就需要这两个函数进行相互转换

### JSON.parse() 和 eval()的区别



虽然两者都能将字符串解析成一个JSON对象，但是图中只有JSON.parse只能转DEF，而eval都能转。

JSON.parse只能解析属性名是双引号包裹的字符串对象，并忽略换行和空行

JQuery的$.parseJSON都是直接使用JSON.parse()

eval函数可将一个JavaScript代码字符串求值成特定的对象，所以解析成JSON对象只不过是作用之一。

**为什么eval()解析JSON字符串要加上括号？**

原因是两点：

1. json对象是以”{}”的方式来开始以及结束的，在JS中，它会被当成一个语句块来处理。

2. 加上圆括号为了处理字符串为表达式，而不是语句（statement）来执行。

虽然eval()强大很多，但是并不建议使用，因为在解析G的时候eval("("+G+")");，会直接用url进行跳转，会引发数据安全性问题(不建议使用)

## 其他

### 深拷贝和浅拷贝

浅拷贝是只拷贝了一层，下一层作为第三方内存存储

1. 直接赋值：var a = 1; var b = a; 此时就是浅拷贝
2. 数组的concat、slice等

var a = [1,2,3,4]; var b = a.concat(); a[0] = 0;

虽然改变a[0]后打印a、b的结果不一致，但并不是深拷贝

当数组内部进行嵌套：var a = [1,2,[3,4],{name:'ccy'}];

此时改变a[2][1] = 5;a[3].name = ‘tgf’

1. b中的值相同都是改变之后的，因为他们指向的还是同一个内存地址
2. 对象同理

深拷贝是从新开辟一片内存来存这些值

1. 使用JSON内置对象进行深拷贝（支持IE8）

例：

var a = {age:1,name:'ccy',info:{address:'wuhan',interest:'playCards'}};

var b = **JSON.parse(JSON.stringify(a))**

注：只能对一般的对象进行深拷贝，函数、方法不可以

1. 如果只有一层，对象可以使用：

var obj2 = Object.assign({}, obj1)

1. Object.create()可以实现深拷贝
2. 使用jQuery的$.extend

var newObj = {}

$.extend(true, newObj, oldObj) // true是指deep是否深度合并

### 垃圾回收

# 基础知识点和技巧

## px、em、rem、vw、vh、vmin、vmax、%

rem相对于根元素的尺寸单位 用于ipnone和ipad分辨率差别大

em相对于父元素的

vw、vh为视口单位，根据浏览器窗口大小而相对设置，1vw即可视窗口的百分之一，和百分比的区别在于，百分比是相对于父元素而vw始终相对于可视窗口

缺点：兼容性要求较高

使用：响应式布局，能根据窗口大小自动调整字体大小

垂直居中：#box{ width: 50vw; height: 50vh; margin: 25vh auto }

原理：height(50vh) + margin-top + margin-bottom = 100vh

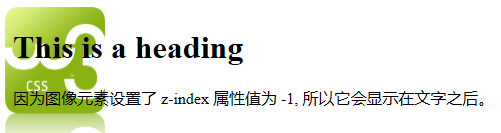
模仿bootstrap珊栏布局

float:left; width: 20vw; … 全都左浮动，且width加起来为100vw即可

**总结**：视口单位是个用来做响应式布局很不错的单位，当然如果要使用vw，vh，我的建议是rem结合vw来开发，因为视口单位有个缺点就是它没有最小或者最大限制，这就达不到我们都时候所希望的一个限制（比如窗口太小了，字都看不清）。所以我们可以在根元素上设置vw和vh，然后在根元素上限制最大最小值，然后配合body设置最大最小宽度。

## 需要某张图在某位置作为背景图 z-index

img



{

position:absolute;

left:0px;

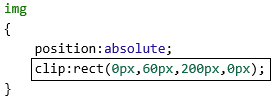
将图片绝对定位在左上角，然后指定z-index为

-1作为背景

top:0px;

z-index:-1;}

## 剪切元素(图片)外形



A B C D

<img src="w3css.gif" width="100" height="140" />

注：首先绝对定位，图片相对于HTML最上方；

A是从上方往下横切0个像素，保留下边的

B是从左往右竖切60个像素，保留切左边的

C是从上往下横切200个像素，保留上边

D是从左往右竖切0个像素，保留右边的

保留左上区用bc，左下区用ba，右上用dc, 右下用da

## 更改光标 <span style="cursor:auto">auto</span>

|  |  |
| --- | --- |
| cursor:auto text | 工 |
| cursor:crosshair | 十 |
| cursor:default | 默认箭头 |
| cursor:help | 箭头右下角有个问号 |
| cursor:move | 带箭头的十 |
| cursor:porter | 手指图标 |
| cursor:progress | 箭头上转圈 |
| cursor:wait | 圆圈一直转 |

## 更改链接的n 中样式

a:link {color:#ff0000;}

a:visited {color:#0000ff;}

a:hover {color:#ffcc00;font-size:150%;background:#66ff66;font-family:Georgia, serif;text-decoration:underline;}

a:action {color:#ffff00;}

使用伪类设置链接样式 ：

:link 鼠标未放置链接上时表现

:visited 鼠标点击后访问时

:hover 鼠标放在链接上时表现

:action 鼠标电视连接时

注：链接表现不只是颜色的变化，也可以设置字体样式大小、背景色、字体格式等

## 导航栏 导航栏=链接列表



**第一步：设置无序列表并格式化 第二步：格式化列表中的链接**

ul {

li a {

    display: inline-block;

    color: white;

    text-align: center;

    padding: 16px;

    text-decoration: none; 链接文本格式清除

}

    list-style-type: none;

    margin: 0;

    padding: 0;

    background: darkgrey;

    overflow: hidden;

    margin: 10px;

}

**第三部：设置水平还是垂直导航、设置鼠标滑过的效果、设置单个链接**

li{display:inline;float: left;}

a:hover{background-color: mediumblue;color: oldlace;}

a.index{background-color: green;color: honeydew;}



加边框：

li { text-align: center; border-bottom: 1px solid #555; }

li:last-child { border-bottom: none; }

全屏高度的固定的导航条：

ul {height: 100%;/\*全屏高或全屏宽\*/ position: fixed; overflow: auto; /\*导航内容多的话\*/}

# 附表

## 所有伪类/元素

|  |  |  |
| --- | --- | --- |
| **选择器** | **示例** | **示例说明** |
| [:checked](https://www.runoob.com/cssref/sel-checked.html) | input:checked | 选择所有选中的表单元素 |
| [:disabled](https://www.runoob.com/css/cssref/sel-disabled.html) | input:disabled | 选择所有禁用的表单元素 |
| [:empty](https://www.runoob.com/cssref/sel-empty.html) | p:empty | 选择所有没有子元素的p元素 |
| [:enabled](https://www.runoob.com/cssref/sel-enable.html) | input:enabled | 选择所有启用的表单元素 |
| [:first-of-type](https://www.runoob.com/cssref/sel-first-of-type.html) | p:first-of-type | 选择的每个 p 元素是其父元素的第一个 p 元素 |
| [:in-range](https://www.runoob.com/cssref/sel-in-range.html) | input:in-range | 选择元素指定范围内的值 |
| [:invalid](https://www.runoob.com/cssref/sel-invalid.html) | input:invalid | 选择所有无效的元素 |
| [**:last-child**](https://www.runoob.com/cssref/sel-last-child.html) | **p:last-child** | **选择所有p元素的最后一个子元素** |
| [:last-of-type](https://www.runoob.com/cssref/sel-last-of-type.html) | p:last-of-type | 选择每个p元素是其母元素的最后一个p元素 |
| [**:not(selector)**](https://www.runoob.com/cssref/sel-not.html) | **:not(p)** | **选择所有p以外的元素** |
| [**:nth-child(n)**](https://www.runoob.com/cssref/sel-nth-child.html) | **p:nth-child(2)** | **选择所有 p 元素的父元素的第二个子元素** |
| [**:nth-last-child(n)**](https://www.runoob.com/cssref/sel-nth-last-child.html) | **p:nth-last-child(2)** | **选择所有p元素倒数的第二个子元素** |
| [:nth-last-of-type(n)](https://www.runoob.com/cssref/sel-nth-last-of-type.html) | p:nth-last-of-type(2) | 选择所有p元素倒数的第二个为p的子元素 |
| [:nth-of-type(n)](https://www.runoob.com/cssref/sel-nth-of-type.html) | p:nth-of-type(2) | 选择所有p元素第二个为p的子元素 |
| [:only-of-type](https://www.runoob.com/cssref/sel-only-of-type.html) | p:only-of-type | 选择所有仅有一个子元素为p的元素 |
| [:only-child](https://www.runoob.com/cssref/sel-only-child.html) | p:only-child | 选择所有仅有一个子元素的p元素 |
| [:optional](https://www.runoob.com/cssref/sel-optional.html) | input:optional | 选择没有"required"的元素属性 |
| [:out-of-range](https://www.runoob.com/cssref/sel-out-of-range.html) | input:out-of-range | 选择指定范围以外的值的元素属性 |
| [:read-only](https://www.runoob.com/cssref/sel-read-only.html) | input:read-only | 选择只读属性的元素属性 |
| [:read-write](https://www.runoob.com/cssref/sel-read-write.html) | input:read-write | 选择没有只读属性的元素属性 |
| [:required](https://www.runoob.com/cssref/sel-required.html) | input:required | 选择有"required"属性指定的元素属性 |
| [:root](https://www.runoob.com/cssref/sel-root.html) | root | 选择文档的根元素 |
| [:target](https://www.runoob.com/cssref/sel-target.html) | #news:target | 选择当前活动#news元素(点击URL包含锚的名字) |
| [:valid](https://www.runoob.com/cssref/sel-valid.html) | input:valid | 选择所有有效值的属性 |
| [**:link**](https://www.runoob.com/cssref/sel-link.html) | **a:link** | **选择所有未访问链接** |
| [**:visited**](https://www.runoob.com/cssref/sel-visited.html) | **a:visited** | **选择所有访问过的链接** |
| [**:active**](https://www.runoob.com/cssref/sel-active.html) | **a:active** | **选择正在活动链接(点击时)** |
| [**:hover**](https://www.runoob.com/cssref/sel-hover.html) | **a:hover** | **把鼠标放在链接上的状态** |
| [:focus](https://www.runoob.com/cssref/sel-focus.html) | input:focus | 选择元素输入后具有焦点 |
| [:first-letter](https://www.runoob.com/cssref/sel-firstletter.html) | p:first-letter | 选择每个<p> 元素的第一个字母 |
| [:first-line](https://www.runoob.com/cssref/sel-firstline.html) | p:first-line | 选择每个<p> 元素的第一行 |
| [:first-child](https://www.runoob.com/cssref/sel-firstchild.html) | p:first-child | 选择器匹配属于任意元素的第一个子元素的 <p> 元素 |
| [:before](https://www.runoob.com/cssref/sel-before.html) | p:before | 在每个<p>元素之前插入内容 |
| [:after](https://www.runoob.com/cssref/sel-after.html) | p:after | 在每个<p>元素之后插入内容 |
| [:lang(*language*)](https://www.runoob.com/cssref/sel-lang.html) | p:lang(it) | 为<p>元素的lang属性选择一个开始值 |
| 选择器 | 示例 | 示例说明 |
| [:link](https://www.runoob.com/cssref/sel-link.html) | a:link | 选择所有未访问链接 |
| [:visited](https://www.runoob.com/cssref/sel-visited.html) | a:visited | 选择所有访问过的链接 |
| [:active](https://www.runoob.com/cssref/sel-active.html) | a:active | 选择正在活动链接 |
| [:hover](https://www.runoob.com/cssref/sel-hover.html) | a:hover | 把鼠标放在链接上的状态 |
| [:focus](https://www.runoob.com/cssref/sel-focus.html) | input:focus | 选择元素输入后具有焦点 |
| [:first-letter](https://www.runoob.com/cssref/sel-firstletter.html) | p:first-letter | 选择每个<p> 元素的第一个字母 |
| [:first-line](https://www.runoob.com/cssref/sel-firstline.html) | p:first-line | 选择每个<p> 元素的第一行 |
| [:first-child](https://www.runoob.com/cssref/sel-firstchild.html) | p:first-child | 选择器匹配属于任意元素的第一个子元素的 <p> 元素 |
| [:before](https://www.runoob.com/cssref/sel-before.html) | p:before | 在每个<p>元素之前插入内容 |
| [:after](https://www.runoob.com/cssref/sel-after.html) | p:after | 在每个<p>元素之后插入内容 |
| [:lang(language)](https://www.runoob.com/cssref/sel-lang.html) | p:lang(it) | 为<p>元素的lang属性选择一个开始值 |

## 颜色表

<https://www.runoob.com/cssref/css-colors.html>

## 数学方法表

**Math 对象属性**

|  |  |
| --- | --- |
| **属性** | **描述** |
| [E](https://www.w3school.com.cn/jsref/jsref_e.asp) | 返回算术常量 e，即自然对数的底数（约等于2.718）。 |
| [LN2](https://www.w3school.com.cn/jsref/jsref_ln2.asp) | 返回 2 的自然对数（约等于0.693）。 |
| [LN10](https://www.w3school.com.cn/jsref/jsref_ln10.asp) | 返回 10 的自然对数（约等于2.302）。 |
| [LOG2E](https://www.w3school.com.cn/jsref/jsref_log2e.asp) | 返回以 2 为底的 e 的对数（约等于 1.414）。 |
| [LOG10E](https://www.w3school.com.cn/jsref/jsref_log10e.asp) | 返回以 10 为底的 e 的对数（约等于0.434）。 |
| [PI](https://www.w3school.com.cn/jsref/jsref_pi.asp) | 返回圆周率（约等于3.14159）。 |
| [SQRT1\_2](https://www.w3school.com.cn/jsref/jsref_sqrt1_2.asp) | 返回返回 2 的平方根的倒数（约等于 0.707）。 |
| [SQRT2](https://www.w3school.com.cn/jsref/jsref_sqrt2.asp) | 返回 2 的平方根（约等于 1.414）。 |

**Math 对象方法**

|  |  |
| --- | --- |
| **方法** | **描述** |
| [abs(x)](https://www.w3school.com.cn/jsref/jsref_abs.asp) | 返回数的绝对值。 |
| [acos(x)](https://www.w3school.com.cn/jsref/jsref_acos.asp) | 返回数的反余弦值。 |
| [asin(x)](https://www.w3school.com.cn/jsref/jsref_asin.asp) | 返回数的反正弦值。 |
| [atan(x)](https://www.w3school.com.cn/jsref/jsref_atan.asp) | 以介于 -PI/2 与 PI/2 弧度之间的数值来返回 x 的反正切值。 |
| [atan2(y,x)](https://www.w3school.com.cn/jsref/jsref_atan2.asp) | 返回从 x 轴到点 (x,y) 的角度（介于 -PI/2 与 PI/2 弧度之间）。 |
| [ceil(x)](https://www.w3school.com.cn/jsref/jsref_ceil.asp) | 对数进行上舍入。 |
| [cos(x)](https://www.w3school.com.cn/jsref/jsref_cos.asp) | 返回数的余弦。 |
| [exp(x)](https://www.w3school.com.cn/jsref/jsref_exp.asp) | 返回 e 的指数。 |
| [floor(x)](https://www.w3school.com.cn/jsref/jsref_floor.asp) | 对数进行下舍入。 |
| [log(x)](https://www.w3school.com.cn/jsref/jsref_log.asp) | 返回数的自然对数（底为e）。 |
| [max(x,y)](https://www.w3school.com.cn/jsref/jsref_max.asp) | 返回 x 和 y 中的最高值。 |
| [min(x,y)](https://www.w3school.com.cn/jsref/jsref_min.asp) | 返回 x 和 y 中的最低值。 |
| [pow(x,y)](https://www.w3school.com.cn/jsref/jsref_pow.asp) | 返回 x 的 y 次幂。 |
| [random()](https://www.w3school.com.cn/jsref/jsref_random.asp) | 返回 0 ~ 1 之间的随机数。 |
| [round(x)](https://www.w3school.com.cn/jsref/jsref_round.asp) | 把数四舍五入为最接近的整数。 |
| [sin(x)](https://www.w3school.com.cn/jsref/jsref_sin.asp) | 返回数的正弦。 |
| [sqrt(x)](https://www.w3school.com.cn/jsref/jsref_sqrt.asp) | 返回数的平方根。 |
| [tan(x)](https://www.w3school.com.cn/jsref/jsref_tan.asp) | 返回角的正切。 |
| [toSource()](https://www.w3school.com.cn/jsref/jsref_tosource_math.asp) | 返回该对象的源代码。 |
| [valueOf()](https://www.w3school.com.cn/jsref/jsref_valueof_math.asp) | 返回 Math 对象的原始值。 |