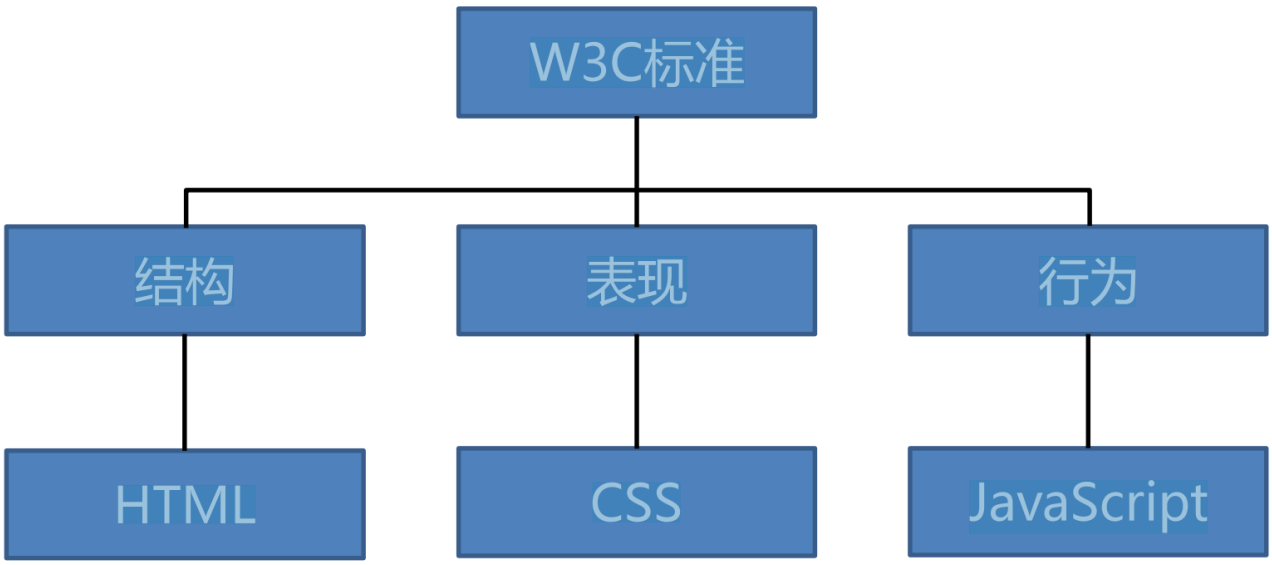
WEB前端

- **HTML**
- **CSS**
- **JavaScript**
- **jQuery（js框架）**
- **Bootstrap（CSS框架）**
- **AJAX**
- **JSON**

WEB前端三大核心技术

根据W3C标准，一个网页主要由三部分组成：结构、表现还有行为。



1. 结构
    1. HTML用于描述页面的结构（页面上有哪些东西）
    2. HTML(5):是一门标记型语言，主要由一些具备特殊含义的标签构成（建筑物结构）
2. 表现
    1. CSS用于控制页面中元素的样式（如何摆放页面上的东西）
    2. CSS(3):是一门描述性语言，主要一系列选择器(html元素)和属性构成(建筑的外部与内部装饰)
3. 行为
    1. JavaScript用于响应用户操作
    2. JavaScript:是一门脚本编程语言，包含类似java的语法(数据类型，数组，条件分支，循环，对象..)(对外运营)

Web开发工具

文本编辑器

- 记事本
- Notepad++/EditPlus

- SublimeText
- Atom
- VIM
- VS Code

集成开发环境（IDE）

- Dreamweaver
- Webstorm
- HBuilder

浏览器选择

- IE(IE9+):不推荐
- Edge
- Google Chrome(推荐)
- Firefox(推荐)
- Opera

HTML

什么是 HTML?

1. HTML 是用来描述网页的一种语言。
2. HTML 指的是超文本标记语言 (**H**yper **T**ext **M**arkup **L**anguage)
3. HTML 不是一种编程语言，而是一种*标记语言* (markup language)
4. 标记语言是一套*标记标签* (markup tag)
5. HTML 使用*标记标签*来描述网页

HTML版本变迁

- W3C(万维网联盟)-早期指定web标准的组织
    - World Wid Web Consortium
    - W3C专门为了定义网页相关的标准而成立。
    - W3C定义了网页中的HTML、CSS、DOM、HTTP、XML等标准。
- WHATWG(超文本应用技术工作组)-由各大浏览器生产商联合成立
    - 是一个以推动网络HTML 5 标准为目的而成立的组织。在2004年，由Opera、Mozilla基金会和苹果这些浏览器厂商组成。
- HTML1.0 ->HTML4.01(1999.12)->XHTML1.0->XHTML2.0
- HTML5(2008)-移动
    - HTML5中已经废弃了很多过时的标签，同时也新增部分新标签，并且将一些老标签的属性部分废弃
    - **HTML5推荐结构与样式分离**（至少避免在html标签中直接使用属性控制样式，推荐使用CSS）

HTML-HelloWorld

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>我的第一个html页面</title>
    </head>
    <body>
        <h1>你好，世界！！！ </h1>
    </body>
</html>
```

HTML 文档 = 网页

1. HTML 文档*描述网页*
2. HTML 文档包含 HTML 标签和纯文本
3. HTML 文档也被称为*网页*

HTML 标签

1. HTML 标记标签通常被称为 HTML 标签 (HTML tag)。
2. HTML 标签是由*尖括号*包围的关键词，比如 <html>
3. HTML 标签通常是*成对出现*的，比如 和
4. 标签对中的第一个标签是*开始标签*，第二个标签是*结束标签*
5. 开始和结束标签也被称为*开放标签*和*闭合标签*

属性（Attribute）

属性用于为标签设置不同的显示效果或者基本信息，属性只能在开始标签中编写，一个标签可能包含多个属性

```
<img src="pic/a.png" alt="用户头像">
```

HTML 常用标签

格式标签

标签	描述
<p>	标题
 	换行
<nobr>	默认不换号
<hr>	分割线
	无序列表
	有序列表
	定义列表的项
<dl>	定义定义列表
<dt>	定义定义列表中的项目
<dd>	定义定义列表中项目的描述

文本标签

标签	描述
<h1>~<h6>	标题
<b>	加粗
<i>	斜体字
<u>	不赞成使用。定义下划线文本。
<sub>、<sup>	下标、上标
<tt>	定义打字机文本。
<cite>	定义引用(citation)。书籍或杂志的标题。
	把文本定义为强调的内容。
	把文本定义为语气*更强*的强调的内容。
<font>	不赞成使用。定义文字的字体、尺寸和颜色。
<pre>	预处理标签
<blockquote>	定义文本缩进
	不赞成使用。定义居中文本。
<marquee>	跑马灯

> **颜色类型**
>
> RGB
>
> RGBA
>
> 16进制颜色值
>
> 英文单词

超链接

标签	描述
<a>	超链接
<iframe>	定义内联框架

> **绝对路径**
>
> > 从根目录开始计算的路径
>
> **相对路径**
>
> > 相对于当前路径计算的路径
>
> **<a>属性**
>
> - href:规定链接指向的页面的 URL。
> - target:规定在何处打开链接文档。
>
> | _blank | **在新窗口中打开被链接文档。** |
> | --- | --- |
> | _self | 默认。在相同的框架中打开被链接文档。 |
> | _parent | 在父框架集中打开被链接文档。 |
> | _top | 在整个窗口中打开被链接文档。 |
> | *iframename* | 在指定的框架中打开被链接文档。 |
>
> 例:
>
> ```
> <!--服务器地址-->
> <a href="http://www.softeem.com">软帝官网</a>
> <!--项目地址-->
> <a href="text.html">文本标签</a>
> <!--发送邮件-->
> <a href="mailto:service_softeem2018@softeem.com">联系我们</a>
> <!--JS代码-->
> <a href="javascript:alert('疼')">点击我</a>
> <a href="javascript:void(0)" onclick="alert(123)">失效的连接</a>
> <!--设置网页打开的位置-->
> <a href="01_helloworld.html" target="_self">第一个页面（在当前页面打开）</a>
> <a href="01_helloworld.html" target="_blank">第一个页面（在新标签页打开）</a>
> ```

```
<a href="http://www.baidu.com" target="content">百度</a>
<a href="http://www.sina.com" target="content">新浪</a>
<a href="http://www.qq.com" target="content">腾讯</a>
<!-- 内嵌框架(结构)) -->
<iframe src="02_html1.html" name="content" width="100%" height="400px">
</iframe>
<!--锚链接 :  略-->
```

多媒体标签

标签	描述
<img>	图片标签
<video>	视频标签
<audio>	音频标签
<canvas>	画布标签

```
<img src="img/p5.jpg" alt="占位文本" width="270px" height="480px">

<audio src="mp3/带你去旅行.mp3" autoplay loop="-1" controls>
    您的浏览器太low了，不支持该标签
</audio>

<video src="mp4/demo.mp4" autoplay controls>
    您的浏览器太low了，不支持该标签
</video>

<canvas width="300px" height="300px">
    您的浏览器太low了，不支持该标签
</canvas>
```

图像地图

```
<!-- 图像地图：将一张图片划分为若干区域，每个区域可以链接到不同的目标 -->
<img src="img/mylinks.png" alt="链接" border="0" usemap="#mymap">
<map name="mymap">
    <area shape="rect" coords="0,0,148,50" href="http://www.baidu.com"
target="_blank">
    <area shape="rect" coords="148,0,290,50" href="http://www.sina.com"
target="_blank">
    <area shape="rect" coords="290,0,440,50" href="http://www.qq.com"
target="_blank">
</map>
```

表格标签

标签	描述
<table>	定义表格
<thead>	定义存放表头
<tbody>	定义存放表数据
<tr>	定义表格中的行
<th>	定义表格中的表头单元格。
<td>	定义表格中的单元

属性

属性	值	描述
border	pixels	规定表格边框的宽度。
cellpadding	pixels%	规定单元边沿与其内容之间的空白。
cellspacing	pixels%	规定单元格之间的空白。
width	%pixels	规定表格的宽度。
colspan	*number*	规定单元格可横跨的列数。
rowspan		规定单元格可横跨的行数。
align	left right center justify char	规定单元格内容的水平对齐方式。

表单标签

标签	描述
<form>	定义供用户输入的 HTML 表单
<input>	定义输入控件。
<textarea>	定义多行的文本输入控件
<select>	定义选择列表（下拉列表）
<datalist>	定义下拉列表（H5）

表单元素

1. action：一般用于填写表单数据提交的目标地址(服务端用于处理请求的资源路径)
2. method：设置表单提交数据的到后台的处理方式，常见:get,post,delete,put(restful)

3. enctype：表单数据提交的编码模式，默认值:application/x-www-form-urlencoded,但是在进行文件上传时，数据需要使用二进制流的方式提交，此时属性值应该设置为:multipart/form-data

例：

```
<select name="prov" id="">
    <option value="hubei">湖北省</option>
    <option value="hunan">湖南省</option>
</select>
<select name="city" id="">
    <option value="wuhan">武汉市</option>
    <option value="xiangyang">襄阳市</option>
    <option value="jingzhou" selected>荆州市</option>
</select>
<!-- 多选下拉列表 -->
<select name="power" id="" multiple>
    <option>文学历史</option>
    <option>计算机技术</option>
    <option>生物医学</option>
    <option>自然科学</option>
    <option>建筑工程</option>
</select>
<!-- 列表分组 -->
<select name="programing" id="">
    <optgroup label="Java">
        <option value="jdbc">JDBC</option>
        <option value="mybatis">Mybatis</option>
        <option value="springdata">SpringData</option>
        <option value="hibernate">Hibernate</option>
    </optgroup>
    <optgroup label="JavaScript">
        <option>Jquery</option>
        <option>Zepto</option>
        <option>Bootstrap</option>
        <option>VueJS</option>
        <option>AngluarJS</option>
    </optgroup>
</select>
```

分区标签

标签	描述
	定义文档中的节
	定义文档中的节

H5语义化标签

标签	描述
<article>	定义文章。
<aside>	定义页面内容以外的内容。
<details>	定义用户能够查看或隐藏的额外细节。
<figcaption>	定义 <figure> 元素的标题。
<figure>	规定自包含内容，比如图示、图表、照片、代码清单等。
<footer>	定义文档或节的页脚。
<header>	规定文档或节的页眉。
<main>	规定文档的主内容。
<mark>	定义重要的或强调的文本。
<nav>	定义导航链接。
<section>	定义文档中的节。
<summary>	定义 <details> 元素的可见标题。
<time>	定义日期/时间。

CSS

1. 层叠样式表 (Cascading Style Sheets)
2. css可以用来为网页创建样式表，通过样式表可以对网页进行装饰。
3. 所谓层叠，可以将整个网页想象成是一层一层的结构，层次高的将会覆盖层次低的。
4. 而css就可以分别为网页的各个层次设置样式。

基本语法

1. CSS的样式表由一个一个的样式构成，一个样式又由选择器和声明块构成。

语法：

```
//选择器 {样式名:样式值; 样式名:样式值 ; }
p {
    color:red;
    font-size:12px;
}
```

CSS引入的三种方式

行内样式

可以直接将样式写到标签内部的style属性中，这种样式不用填写选择器，直接编写声明即可

```
<p style="color: red;font-size: 30px"></p>
```

这种方式编写简单，定位准确。但是由于直接将css代码写到了html标签的内部，导致结构与表现耦合，同时导致样式不能够复用，所以这种方式我们不使用。

内部样式

可以直接将样式写到<style>标签中

```
<style>
p{color:red; font-size: 30px;}
</style>
```

1. 这样使css独立于html代码，而且可以同时为多个元素设置样式，这是我们使用的比较多的一种方式。
2. 但是这种方式，样式只能在一个页面中使用，不能在多个页面中重复使用

外部样式表

可以将所有的样式保存到一个外部的css文件中，然后通过<link>标签将样式表引入到文件中。

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

这种方式将样式表放入到了页面的外部，可以在多个页面中引入，同时浏览器加载文件时可以使用缓存，这是我们开发中使用的最多的方式

单位

- 尺寸:
  - 百分比(%)
  - in(英寸)，cm(厘米)，mm(毫米)
  - px(像素，显示屏上的一个点)
  - pt(磅,一般常见于字体单位1pt=1/72英寸)，pc(活点字1pc=12点)
  - em（常见于字体单位，16px=1em=12pt）
  - rem(常见与移动设备的尺寸设置)
- 颜色:
  - 颜色单词(red,blue,green,lightblue，orange...)
  - 16进制颜色编码(#F00,#0F0,#00F/#FF0000,#00FF00,#0000FF) #000/#FFF
  - rgb值(red,green,blue->rgb(0,0,0)~rgb(255,255,255))
  - rgba值(red,green,blue,alpha->rgba(0,0,0,0.0)~rgba(255,255,255,1.0)) alpha透明度

CSS选择器

选择器

1. 选择器（selector），会告诉浏览器：网页上的哪些元素需要设置什么样的样式。
2. 比如：p这个选择器就表示选择页面中的所有的p元素，在选择器之后所设置的样式会应用到所有的p元素上。

元素选择器

元素选择器（标签选择器），可以根据标签的名字来从页面中选取指定的元素。

```
标签名 {}
```

比如p则会选中页面中的所有p标签，h1会选中页面中的所有h1标签

ID选择器

ID选择器，可以根据元素的id属性值选取元素。

```
#id { }
```

比如#box会选中页面中id属性值为box的元素，和class属性不同，id属性是不能重复的。

类选择器·

类选择器，可以根据元素的class属性值选取元素。

```
.className { }
```

比如.hello会选中页面所有class属性为hello的元素。

复合选择器

复合选择器，可以同时使用多个选择器，这样可以选择同时满足多个选择器的元素。

```
选择器1选择器2{}
```

例如div.box1会选中页面中具有box1这个class的div元素。

选择器分组

群组选择器，可以同时使用多个选择器，多个选择器将被同时应用指定的样式。

```
选择器1,选择器2,选择器3 { }
```

比如p,.hello,#box会同时选中页面中p元素，class为hello的元素，id为box的元素。

通用选择器

通用选择器，可以同时选中页面中的所有元素。

```
* { }
```

标签之间的关系

- 祖先元素
    - 直接或间接包含后代元素的元素。

- 后代元素
  - 直接或间接被祖先元素包含的元素。
- 父元素
  - 直接包含子元素的元素。
- 子元素
  - 直接被父元素包含的元素。
- 兄弟元素
  - 拥有相同父元素的元素。

后代选择器

后代选择器可以根据标签的关系，为处在元素内部的后代元素设置样式。

```
祖先元素 后代元素 后代元素 { }
```

比如p strong 会选中页面中所有的p元素内的strong元素。

子选择器

子选择器可以根据标签的关系，为处在元素内部的子元素设置样式。

```
祖先元素>后代元素>后代元素 { }
```

兄弟选择器

除了根据祖先父子关系，还可以根据兄弟关系查找元素。

```
查找后边一个兄弟元素
    兄弟元素 + 兄弟元素{}
查找后边所有的兄弟元素
    兄弟元素 ~ 兄弟元素{}
```

伪类和伪元素

有时候，你需要选择本身没有标签，但是仍然易于识别的网页部位，比如段落首行或鼠标滑过的连接。CSS为他们提供一些选择器：伪类和伪元素。

给链接定义样式

有四个伪类可以让你根据访问者与该链接的交互方式，将链接设置成4种不同的状态。

- 正常链接
  - a:link
- 访问过的链接
  - a:visited（只能定义字体颜色）
- 鼠标滑过的链接
  - a:hover

- 正在点击的链接
    - a:active

其他

- 获取焦点
    - :focus
- 指定元素前
    - :before
- 指定元素后
    - :after
- 选中的元素
    - ::select
- 首字母
    - :first-letter
- 首行
    - :first-line
- 选择第一个子标签
- :first-child
- 选择最后一个子标签
    - :last-child
- 选择指定位置的子
    - :nth-child
- 否定伪类
    - :not

属性选择器

属性选择器可以挑选带有特殊属性的标签。

[attribute]	**用于选取带有指定属性的元素。**
[*attribute*=*value*]	用于选取带有指定属性和值的元素。
[*attribute*~=*value*]	用于选取属性值中包含指定词汇的元素。
[*attribute*|=*value*]	用于选取带有以指定值开头的属性值的元素，该值必须是整个单词。（'-'分割）
[*attribute*^=*value*]	匹配属性值以指定值开头的每个元素。
[*attribute*$=*value*]	匹配属性值以指定值结尾的每个元素。
[attribute*=*value*]	匹配属性值中包含指定值的每个元素。

继承

1. 就像父亲的财产会遗传给儿子一样，在CSS中祖先元素的样式同样也会被子元素继承。
2. 继承是指应用在一个标签上的那些CSS样式会同时被应用到其内嵌标签上。
3. 比如为父元素设置了字体颜色，子元素也会应用上相同的颜色。
4. 当然并不是所有的样式都会被继承，这一点我们讲到具体样式时，再去讨论。

选择器的权重

在页面中使用CSS选择器选中元素时，经常都是一个元素同时被多个选择器选中。

```
body h1
h1
```

上边的两个选择器都会选择h1元素，如果两个选择器设置的样式不一致那还好不会产生冲突，但是如果两个选择器设置的是同一个样式，这样h1到底要应用那个样式呢？CSS中会默认使用权重较大的样式，权重又是如何计算的呢?

权重的计算

不同的选择器有不同的权重值：

- 内联样式：权重是 1000
- id选择器：权重是 100
- 类、属性、伪类选择器：权重是 10
- 元素选择器：权重是 1
- 通配符：权重是 0
- 计算权重需要将一个样式的全部选择器相加，比如上边的body h1的权重是2，h1的权重是1，所以第一个选择器设置的样式会优先显示。

文字样式

属性	取值	描述
font-size		规定文本的字体尺寸。
font-family		规定文本的字体系列
font-style		设置字体样式
font-weight		规定字体的粗细。
line-height		设置行高
text-decoration		规定添加到文本的装饰效果。
text-align		规定文本的水平对齐方式。
text-indent		规定文本块首行的缩进。
vertical-align		设置元素的垂直对齐方式。

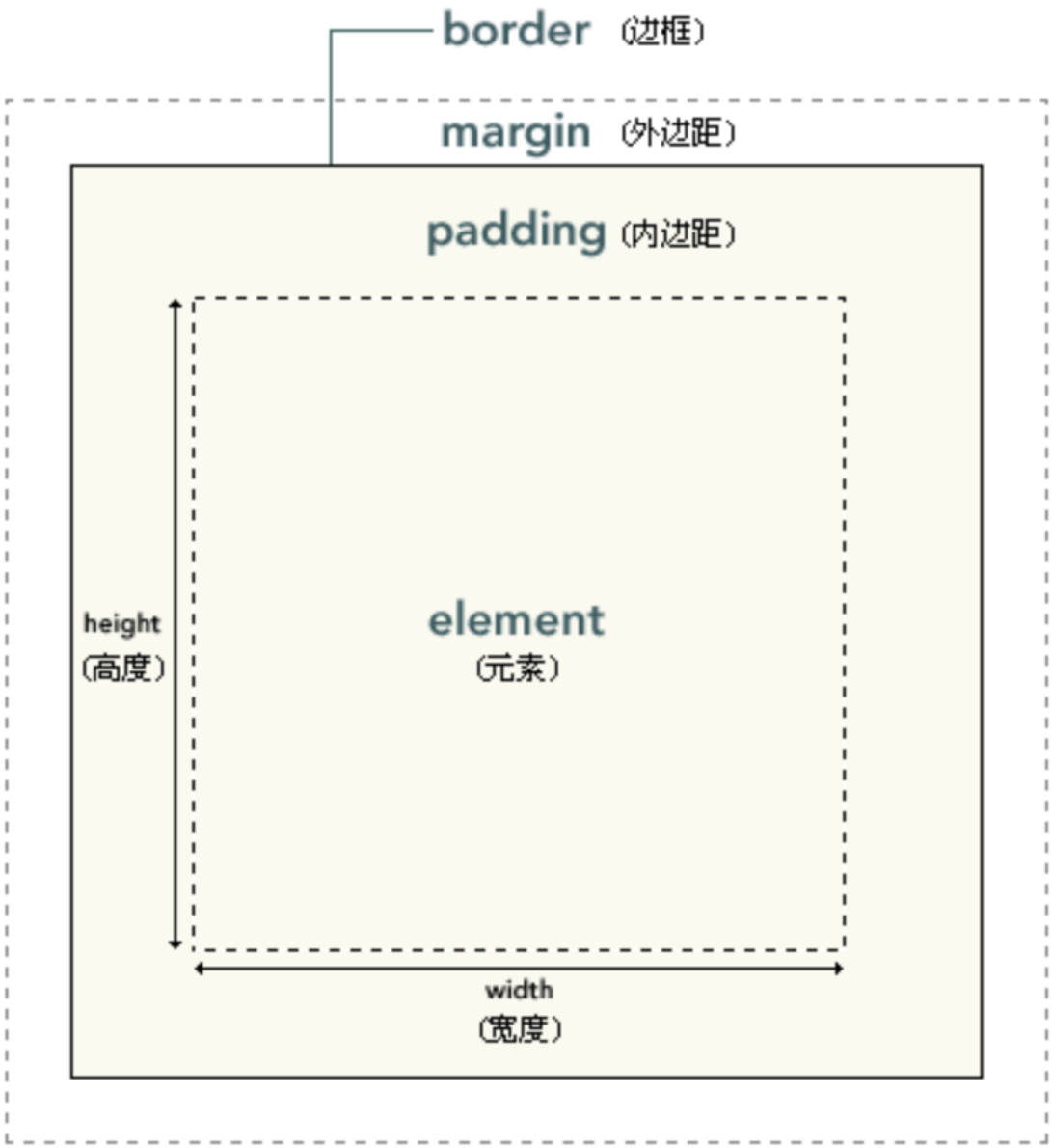
盒子模型

1. CSS处理网页时，它认为每个元素都包含在一个不可见的盒子里。

2. 为什么要想象成盒子呢？因为如果把所有的元素都想象成盒子，那么我们对网页的布局就相当于是摆放盒子。
3. 我们只需要将相应的盒子摆放到网页中相应的位置即可完成网页的布局。

一个盒子由几部分组成

- 内容区(content)
- 内边距(padding)
- 边框(border)
- 外边距(margin)



内容区

1. 内容区指的是盒子中放置内容的区域，也就是元素中的文本内容，子元素都是存在于内容区中的。
2. 如果没有为元素设置内边距和边框，则内容区大小默认和盒子大小是一致的。

3. 通过width和height两个属性可以设置内容区的大小。
4. width和height属性只适用于块元素。

内边距

1. 顾名思义，内边距指的就是元素内容区与边框以内的空间。
2. 默认情况下width和height不包含padding的大小。
3. 使用padding属性来设置元素的内边距。
4. 例如：
    1. padding:10px 20px 30px 40px
    2. 这样会设置元素的上、右、下、左四个方向的内边距

内边距

- padding:10px 20px 30px;
    - 分别指定上、左右、下四个方向的内边距
- padding:10px 20px;
    - 分别指定上下、左右四个方向的内边距
- padding:10px;
    - 同时指定上左右下四个方向的内边距
- 同时在css中还提供了padding-top、padding-right、padding-letf、padding-bottom分别用来指定四个方向的内边距

边框

- 可以在元素周围创建边框，边框是元素可见框的最外部。
- 可以使用border属性来设置盒子的边框：
    - border:1px red solid;
        - 上边的样式分别指定了边框的宽度、颜色和样式。
- 也可以使用border-top/left/right/bottom分别指定上右下左四个方向的边框。
- 和padding一样，默认width和height并包括边框的宽度
- 边框的样式
    - none（没有边框）
    - dotted（点线）
    - dashed（虚线）
    - solid（实线）
    - double（双线）
    - groove（槽线）
    - ridge（脊线）
    - inset（凹边）
    - outset（凸边）

外边距

- 外边距是元素边框与周围元素相距的空间。
- 使用margin属性可以设置外边距。

- 用法和padding类似，同样也提供了四个方向的margin-top/right/bottom/left。
- 当将左右外边距设置为auto时，浏览器会将左右外边距设置为相等，所以这行代码margin:0 auto可以使元素居中。

display

- 我们不能为行内元素设置width、height、margin-top和margin-bottom。
- 我们可以通过修改display来修改元素的性质。
- 可选值：
  - block：设置元素为块元素
  - inline：设置元素为行内元素
  - inline-block：设置元素为行内块元素
  - none：隐藏元素（元素将在页面中完全消失）

visibility

- visibility属性主要用于元素是否可见。
- 和display不同，使用visibility隐藏一个元素，隐藏后其在文档中所占的位置会依然保持，不会被其他元素覆盖。
- 可选值：
  - visible：可见的
  - hidden：隐藏的

overflow

- 当相关标签里面的内容超出了样式的宽度和高度是，就会发生一些奇怪的事情，浏览器会让内容溢出盒子。
- 可以通过overflow来控制内容溢出的情况。
- 可选值：
  - visible：默认值
  - scroll：添加滚动条
  - auto：根据需要添加滚动条
  - hidden：隐藏超出盒子的内容

文档流

- 文档流指的是文档中可现实的对象在排列时所占用的位置。
- 将窗体自上而下分成一行行，并在每行中按从左至右的顺序排放元素，即为文档流。
- 也就是说在文档流中元素默认会紧贴到上一个元素的右边，如果右边不足以放下元素，元素则会另起一行，在新的一行中继续从左至右摆放。
- 这样一来每一个块元素都会另起一行，那么我们如果想在文档流中进行布局就会变得比较麻烦

浮动

- 所谓浮动指的是使元素脱离原来的文本流，在父元素中浮动起来。
- 浮动使用float属性。
- 可选值：
  - none：不浮动

    - left：向左浮动
    - right：向右浮动
- 块级元素和行内元素都可以浮动，当一个行内元素浮动以后将会自动变为一个内联块级元素。
- 当一个块级元素浮动以后，宽度会默认被内容撑开，所以当漂浮一个块级元素时我们都会为其指定一个宽度。
- 当一个元素浮动以后，其下方的元素会上移。元素中的内容将会围绕在元素的周围。
- 浮动会使元素完全脱离文本流，也就是不再在文档中在占用位置。
- 元素设置浮动以后，会一直向上漂浮直到遇到父元素的边界或者其他浮动元素。
- 元素浮动以后即完全脱离文档流，这时不会再影响父元素的高度。也就是浮动元素不会撑开父元素。
- 浮动元素默认会变为块元素，即使设置display:inline以后其依然是个块元素。

清除浮动

- clear属性可以用于清除元素周围的浮动对元素的影响。
- 也就是元素不会因为上方出现了浮动元素而改变位置。
- 可选值：
    - left：忽略左侧浮动
    - right：忽略右侧浮动
    - both：忽略全部浮动
    - none：不忽略浮动，默认值

定位

- position属性可以控制Web浏览器如何以及在何处显示特定的元素。
- 可以使用position属性把一个元素放置到网页中的任何位置。
- 可选值：
    - static
    - relative
    - absolute
    - fixed

相对定位

- 每个元素在页面的文档流中都有一个自然位置。相对于这个位置对元素进行移动就称为相对定位。周围的元素完全不受此影响。
- 当将position属性设置为relative时，则开启了元素的相对定位。
- 当开启了相对定位以后，可以使用top、right、bottom、left四个属性对元素进行定位。
- 相对定位的特点
    - 如果不设置元素的偏移量，元素位置不会发生改变。
    - 相对定位不会使元素脱离文本流。元素在文本流中的位置不会改变。
    - 相对定位不会改变元素原来的特性。
    - 相对定位会使元素的层级提升，使元素可以覆盖文本流中的元素。

绝对定位

- 绝对定位指使元素相对于html元素或离他最近的祖先定位元素进行定位。
- 当将position属性设置为absolute时，则开启了元素的绝对定位。
- 当开启了绝对定位以后，可以使用top、right、bottom、left四个属性对元素进行定位
- 绝对定位的特点
  - 绝对定位会使元素完全脱离文本流。
  - 绝对定位的块元素的宽度会被其内容撑开。
  - 绝对定位会使行内元素变成块元素。
  - 一般使用绝对定位时会同时为其父元素指定一个相对定位，以确保元素可以相对于父元素进行定位。

固定定位

- 固定定位的元素会被锁定在屏幕的某个位置上，当访问者滚动网页时，固定元素会在屏幕上保持不动。
- 当将position属性设置为fixed时，则开启了元素的固定定位。
- 当开启了固定定位以后，可以使用top、right、bottom、left四个属性对元素进行定位。
- 固定定位的其他特性和绝对定位类似。

z-index

- 当元素开启定位以后就可以设置z-index这个属性。
- 这个属性可以提升定位元素所在的层级。
- z-index可以指定一个整数作为参数，值越大元素显示的优先级越高，也就是z-index值较大的元素会显示在网页的最上层。

背景样式

background-color

- background-color属性用来为元素设置背景颜色。
- 需要指定一个颜色值，当指定了一个颜色以后，整个元素的可见区域都会使用这个颜色作为背景色。
- 如果不设置背景颜色，元素默认背景颜色为透明，实际上会显示父元素的背景颜色。

background-image

- background-image可以为元素指定背景图片。
- 和background-color类似，这不过这里使用的是一个图片作为背景。
- 需要一个url地址作为参数，url地址需要指向一个外部图片的路径
- 例如：

```
background-image: url(1.jpg)
```

background-repeat

- background-repeat用于控制背景图片的重复方式。
- 如果只设置背景图片默认背景图片将会使用平铺的方式，可以通过该属性进行修改。
- 可选值：
  - repeat：默认值，图片左右上下平铺
  - no-repeat：只显示图片一次，不会平铺

- repeat-x：沿x轴水平平铺一张图片
- repeat-y：沿y轴水平平铺一张图片

background-position

- background-position用来精确控制背景图片在元素中的位置。
- 可以通过三种方式来确定图片在水平方向和垂直方向的起点。
  - 关键字：top right bottom left center
  - 百分比
  - 数值

background-attachment

- background-attachment用来设置背景图片是否随页面滚动。
- 可选值：
  - scroll：随页面滚动
  - fixed：不随页面滚动

background

- background是背景的简写属性，通过这个属性可以一次性设置多个样式，而且样式的顺序没有要求。
- 例如：
  - background: green url(1.jpg) no-repeat center center fixed;

CSS Sprite

- CSS Sprites是一种网页图片应用处理方式。
- 通过这种方式我们可以将网页中的零星图片集中放到一张大图上。
- 这样一来，一次请求便可以同时加载多张图片，大大提高了图片的加载效率。

JavaScript

JavaScript 是世界上最流行的编程语言之一。这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

JavaScript 是脚本语言

- JavaScript 是一种轻量级的编程语言，属于解释型语言。
  - JavaScript是一门解释型语言，所谓解释型值语言不需要被编译为机器码在执行，而是直接执行。
  - 由于少了编译这一步骤，所以解释型语言开发起来尤为轻松，但是解释型语言运行较慢也是它的劣势。
  - 不过解释型语言中使用了JIT技术，使得运行速度得以改善。
- JavaScript 是可插入 HTML 页面的编程代码。
- JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。
- JavaScript 很容易学习。
- 基于原型的面向对象

JS编写位置

- 我们目前学习的JS全都是客户端的JS，也就是说全都是需要在浏览器中运行的，所以我们我们的JS代码全都需要在网页中编写。
- 我们的JS代码需要编写到<script>标签中。
- 属性：
  - type：默认值text/javascript可以不写，不写也是这个值。
  - src：当需要引入一个外部的js文件时，使用该属性指向文件的地址。

HelloWorld

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>JS HelloWorld</title>
        <script type="text/javascript">
            alert("哈哈哈哈");
        </script>
    </head>
    <body>
    </body>
</html>
```

严格区分大小写

- JavaScript是严格区分大小写的，也就是abc和Abc会被解析器认为是两个不同的东西。
- 所以在编写上边的HelloWorld时，一定要注意区分大小写

注释

- 注释中的内容不会被解析器解析执行，但是会在源码中显示，我们一般会使用注释对程序中的内容进行解释。
- JS中的注释和Java的的一致，分为两种：
  - 单行注释：//注释内容
  - 多行注释：/*注释内容*/

变量

- 变量的声明：
- 使用var关键字声明一个变量。

数据类型

- 数据类型决定了一个数据的特征，比如：123和”123”，直观上看这两个数据都是123，但实际上前者是一个数字，而后者是一个字符串。
- 对于不同的数据类型我们在进行操作时会有很大的不同。
- JavaScript中一共有5种基本数据类型：

    - 字符串型（String）
    - 数值型（Number）
    - 布尔型（Boolean）
    - null型（Null）
    - undefined型（Undefined）
- 这5种之外的类型都称为Object，所以总的来看JavaScript中共有六种数据类型。

typeof运算符

- 使用typeof操作符可以用来检查一个变量的数据类型。
- 使用方式：typeof 数据，例如 typeof 123。
- 返回结果：
    - typeof 数值 number
    - typeof 字符串 string
    - typeof 布尔型 boolean
    - typeof undefined undefined
    - typeof null object

String

- String用于表示一个字符序列，即字符串。
- 字符串需要使用 '或" 括起来。
- 将其他数值转换为字符串有三种方式：toString()、String()、拼串。

Number

- Number 类型用来表示整数和浮点数，最常用的功能就是用来表示10进制的整数和浮点数。
- Number表示的数字大小是有限的，范围是：
    - ± 1.7976931348623157e+308
    - 如果超过了这个范围，则会返回± Infinity。
- NaN，即非数值（Not a Number）是一个特殊的数值，JS中当对数值进行计算时没有结果返回，则返回NaN。

数值的转换

- 有三个函数可以把非数值转换为数值：Number()、parseInt()和parseFloat()。
- Number()可以用来转换任意类型的数据，而后两者只能用于转换字符串。
- parseInt()只会将字符串转换为整数，而parseFloat()可以转换为浮点数

Boolean(布尔型)

- 布尔型也被称为逻辑值类型或者真假值类型。
- 布尔型只能够取真（true）和假（false）两种数值。除此以外，其他的值都不被支持。
- 其他的数据类型也可以通过Boolean()函数转换为布尔类型
- 转换规则：

-

数据类型	转换为true	转换为false
Boolean	true	false
String	任何非空字符串	“”（空字符串）
Number	任何非0数字	0和NaN
Object	任何对象	null
Undefined	n/a	undefined

Undefined

- Undefined 类型只有一个值，即特殊的 undefined 。
- 在使用 var 声明变量但未对其加以初始化时，这个变量的值就是 undefined。例如：
  - var message;
  - message 的值就是 undefined。
- 需要注意的是typeof对没有初始化和没有声明的变量都会返回undefined。
- undefined是由null派生出来的，所以使用 '==' 进行比较时会返回true

Null

- Null 类型是第二个只有一个值的数据类型，这个特殊的值是null 。
- 从语义上看null表示的是一个空的对象。所以使用typeof检查null会返回一个Object。
- undefined值实际上是由null值衍生出来的，所以如果比较undefined和null是否相等，会返回true；

运算符

算数运算符

- 算数运算符顾名思义就是进行算数操作的运算符。
- JS中为我们提供了多种算数运算符。
-

运算符	说明	运算符	说明
+	加法	++（前置）	自增
-	减法	++（后置）	自增
*	乘法	--（前置）	自减
/	除法	--（后置）	自减
%	取模	+	符号不变
		-	符号反转

逻辑运算符

- 一般情况下使用逻辑运算符会返回一个布尔值。
- 逻辑运算符主要有三个：非、与、或。
- 在进行逻辑操作时如果操作数不是布尔类型则会将其转换成Boolean类型进行运算，返回时返回其本身。
- 布尔类型在进行计算。
-

运算符	说明	短路规则
!	逻辑非（NOT）	无
&&	逻辑与（AND）	若左值为假，则不运算右值
||	逻辑或（OR）	若左值为真，则不运算右值

非

- 非运算符使用 ! 表示。
- 非运算符可以应用于任意值，无论值是什么类型，这个运算符都会返回一个布尔值。
- 非运算符会对原值取反，比如原值是true使用非运算符会返回false，原值为false使用非运算符会返回true。

与

- 与运算符使用 && 表示。
- 与运算符可以应用于任何数据类型，且不一定返回布尔值。
- 对于非布尔值运算，会先将非布尔值转换为布尔值。
- 对布尔值做运算时，如果两个值都为true则返回true，否则返回false。
- 非布尔值时：如果两个都为true，则返回第二个值，如果两个值中有false则返回靠前的false的值

或

- 或运算符使用 || 表示。
- 或运算符可以应用于任何数据类型，且不一定返回布尔值。
- 对于非布尔值运算，会先将非布尔值转换为布尔值。
- 对布尔值进行运算时，如果两个值都为false则返回false，否则返回true。
- 非布尔值时：如果两个都为false ，则返回第二个值，否则返回靠前true的值。

关系运算符

相等

- JS中使用==来判断两个值是否相等，如果相等则返回true。
- 使用!=来表示两个值是否不相等，如果不等则返回true。
- 注意：null和undefined使用==判断时是相等的。
- NaN和任何数比较相等都是false

表达式	值	表达式	值
null == undefined	true	true == 1	true
“NaN” == NaN	false	true == 2	false
5 == NaN	false	undefined == 0	false
NaN == NaN	false	null == 0	false
NaN != NaN	true	“5” == 5	true
false == 0	true		

全等

- 除了==以外，JS中还提供了===
- ===表示全等，他和==基本一致，不过==在判断两个值时会进行自动的类型转换，而===不会。
- 也就是说”55”==55会返回true，而”55”===55会返回false；
- 同样我们还有!==表示不全等，同样比较时不会自动转型。
- 也就是说”55”!=55会返回false，而”55”!==55会返回true；

语句

> **和java完全相同**

Object对象

- Object类型，我们也称为一个对象。是JavaScript中的引用数据类型。
- 它是一种复合值，它将很多属性值聚合到一起，可以通过名字访问这些值。
- 对象也可以看做是属性的无序集合，每个属性都是一个名/值对。
- 对象除了可以创建自有属性，还可以通过从一个名为原型的对象那里继承属性。
- 除了字符串、数字、true、false、null和undefined之外，JS中的值都是对象。

创建对象

方法一

```
var person = new Object();
person.name = "孙悟空";
person.age = 1
```

方法二

```
var person = {
    name:"孙悟空",
    age:18
};
```

对象属性的访问

- 访问属性的两种方式：
  - "." 访问
    - 对象.属性名
  - "[]" 访问
    - 对象['属性名']
- 检查类中是否含有某属性
  - ```
    console.log("name" in obj);
    ```
- 删除属性
  - delete 对象.属性名

函数

- 函数是由一连串的子程序（语句的集合）所组成的，可以被外部程序调用。向函数传递参数之后，函数可以返回一定的值。
- 通常情况下，JavaScript 代码是自上而下执行的，不过函数体内部的代码则不是这样。如果只是对函数进行了声明，其中的代码并不会执行。只有在调用函数时才会执行函数体内部的代码。
- 这里要注意的是JavaScript中的函数也是一个对象。

函数的声明

- 首先明确一点函数也是一个对象，所以函数也是在堆内存中保存的。
- 函数声明比较特殊，需要使用function关键字声明。
  - ```
    var sum = function(a,b){
        return a+b
    };
    ```
- 上边的例子就是创建了一个函数对象，并将函数对象赋值给了sum这个变量。其中()中的内容表示行函数时需要的参数，{}中的内容表示函数的
- 可以通过函数声明语句来定义一个函数。函数声明语句以关键字function 开始，其后跟有函数名、参数列表和函数体。其语法如下所示：
  - ```
    function sum(a,b){
        return a+b
    }
    ```
- 上边我们定义了一个函数名为sum，两个参数a和b。函数声明时设置的参数称为形参（形式参数），这个函数对两个参数做了加法运算并将结果返回。

函数的调用

- 调用函数时，传递给函数的参数称为实参（实际参数）。
- 如果想调用我们上边定义的sum函数，可以这样写：
  - ```
    var result = sum(123,456);
    ```

- 这样表示调用sum这个函数，并将123和456作为实参传递给函数，函数中会将两个参数求和并赋值给result。

this

- this 引用的是一个对象。对于最外层代码与函数内部的情况，其引用目标是不同的。
- 此外，即使在函数内部，根据函数调用方式的不同，引用对象也会有所不同。需要注意的是，this 引用会根据代码的上下文语境自动改变其引用对象。

构造函数

- 构造函数是用于生成对象的函数，像之前调用的Object()就是一个构造函数。
- 创建一个构造函数：
  - ```
    function MyClass(x,y) {
        this.x = x;
        this.y = y;
    }
    ```
- 调用构造函数：
  - 构造函数本身和普通的函数声明形式相同。
  - 构造函数通过 new 关键字来调用，new 关键字会新创建一个对象并返回。
  - 通过 new关键字调用的构造函数内的 this 引用引用了（被新生成的）对象

new关键字

- 使用new关键字执行一个构造函数时：
- 首先，会先创建一个空的对象。
- 然后，会执行相应的构造函数。构造函数中的this将会引用这个新对象。
- 最后，将对象作为执行结果返回。
  - 构造函数总是由new关键字调用。
  - 构造函数和普通函数的区别就在于调用方式的不同。
  - 任何函数都可以通过new来调用，所以函数都可以是构造函数。
  - 在开发中，通常会区分用于执行的函数和构造函数。
    - 构造函数的首字母要大写

原型

原型继承

- JS是一门面向对象的语言，而且它还是一个基于原型的面向对象的语言。
- 所谓的原型实际上指的是，在构造函数中存在着一个名为原型的(prototype)对象，这个对象中保存着一些属性，凡是通过该构造函数创建的对象都可以访问存在于原型中的属性。
- 最典型的原型中的属性就是toString()函数，实际上我们的对象中并没有定义这个函数，但是却可以调用，那是因为这个函数存在于Object对应的原型中。

设置原型
    ```

- 原型就是一个对象，和其他对象没有任何区别，可以通过构造函数来获取原型对象。
  - 构造函数. prototype
- 和其他对象一样我们可以添加修改删除原型中的属性，也可以修改原型对象的引用。
- 需要注意的是prototype属性只存在于函数对象中，其他对象是没有prototype属性的。
- 每一个对象都有原型，包括原型对象也有原型。特殊的是Object的原型对象没有原型。

获取原型对象的方法

- 除了可以通过构造函数获取原型对象以外，还可以通过具体的对象来获取原型对象。
- ```
  - – Object.getPrototypeOf(对象)
  - – 对象.__proto__
  - – 对象. constructor.prototype
  ```
- 需要注意的是，我们可以获取到Object的原型对象，也可以对它的属性进行操作，但是我们不能修改Object原型对象的引用。

原型链

- 基于我们上边所说的，每个对象都有原型对象，原型对象也有原型对象。
- 由此，我们的对象，和对象的原型，以及原型的原型，就构成了一个原型链。
- 比如这么一个对象：
  - var mc = new MyClass(123,456);
  - 这个对象本身，原型MyClass.proprototype原型对象的原型对象是Object，Object对象还有其原型。这组对象就构成了一个原型链。
  - 这个链的次序是：mc对象、mc对象原型、原型的原型（Object）、Object的原型
- 当从一个对象中获取属性时，会首先从当前对象中查找，如果没有则顺着向上查找原型对象，直到找到Object对象的原型位置，找到则返回，找不到则返回undefined。

instanceof

- 之前学习基本数据类型时我们学习了typeof用来检查一个变量的类型。
- 但是typeof对于对象来说却不是那么好用，因为任何对象使用typeof都会返回Object。而我们想要获取的是对象的具体类型。
- 这时就需要使用instanceof运算符了，它主要用来检查一个对象的具体类型。
- 语法：
  - var result = 变量 instanceof 类型

JavaScript常用类

Object

- 目前为止，我们看到的最多的类型就是Object，它也是我们在JS中使用的最多的对象。
- 虽然Object对象中并没有为我们提供太多的功能，但是我们会经常会用途来存储和传输数据。
- 创建Object对象有两种方式：
  ```

```
var obj = new Object();
var obj = {} //字面量的方式进行创建
```

- 上边的两种方式都可以返回一个Object对象。
- 但是第一种我们使用了一个new关键字和一个Object()函数。
- 这个函数就是专门用来创建一个Object对象并返回的，像这种函数我们称为构造函数。

Array

- Array用于表示一个有序的数组。
- JS的数组中可以保存任意类型的数据。
- 创建一个数组的方式有两种：
  - – 使用构造器：
  - ```
    var arr = new Array(数组的长度);
    var arr = new Array(123,'hello',true);
    ```
  - – 使用[]
  - ```
    var arr = [];
    var arr = [123,'hello',false];
    ```
- 读取数组中的值使用数组[索引]的方式，注意索引是从0开始的。

Date

- Date类型用来表示一个时间。
- Date采取的是时间戳的形式表示时间，所谓的时间戳指的是从1970年1月1日0时0秒0分开始经过的毫秒数来计算时间。
- 直接使用new Date()就可以创建一个Date对象。创造对象时不传参数默认创建当前时间。可以传递一个毫秒数用来创建具体的时间。
- 也可以传递一个日期的字符串，来创建一个时间。
  - 格式为：月份/日/年 时:分:秒
  - 例如：06/13/2004 12:12:12

Function

- Function类型代表一个函数，每一个函数都是一个Function类型的对象。而且都与其他引用类型一样具有属性和方法。
- 由于函数是对象，因此函数名实际上也是一个指向函数对象的指针，不会与某个函数绑定。
- 函数的声明有两种方式：
  - ```
    function sum(){}
    var sum = function(){};
    ```
- 由于存在函数声明提升的过程，第一种方式在函数声明之前就可以调用函数，而第二种不行
    ```

函数也可以作为参数

- 函数也是一个对象，所以函数和其他对象一样也可以作为一个参数传递给另外一个函数。
- 但是要注意的是使用函数作为参数时，变量后边千万不要加()，不加()表示将函数本身作为参数，加上以后表示将函数执行的结果作为参数

函数对象的方法

- 每个函数都有两个方法call()和apply()。
- call()和apply()都可以指定一个函数的运行环境对象，换句话说就是设置函数执行时的this值。
- 使用方式：
  - 函数对象.call(this对象,参数1,参数2,参数N)
  - 函数对象.apply(this对象,参数数组)

闭包

- 闭包是JS一个非常重要的特性，这意味着当前作用域总是能够访问外部作用域中的变量。因为函数是JS中唯一拥有自身作用域的结构，因此闭包的创建依赖于函数。
- 也可以将闭包的特征理解为，其相关的局部变量在函数调用结束之后将会继续存在

String

- String 类型是字符串的对象包装类型，可以像下面这样使用 String 构造函数来创建。
  - ```
    var str = new String("hello world");
    ```
- 可以使用length属性来获取字符串的长度

Math

- JS 还为保存数学公式和信息提供了一个公共位置，即 Math 对象。
- 与我们在 JavaScript 直接编写的计算功能相比， Math 对象提供的计算功能执行起来要快得多。 Math 对象中还提供了辅助完成这些计算的属性和方法。

Math方法

- 最大最小值
- Math.max()获取最大值
- Math.min()获取最小值
- • 舍入：
  - 向上舍 Math.ceil()
  - 向下舍 Math.floor()
  - 四舍五入 Math.round()
- • 随机数： Math.random()
  - 选取某个范围内的随机值：
    - 值 = Math.floor(Math.random() * 可能值的总数 + 第一个可能的值)

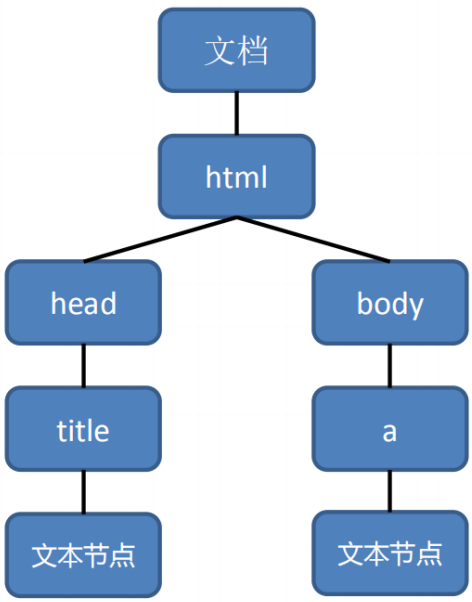
DOM

- DOM，全称Document Object Model文档对象模型。
    ```

- JS中通过DOM来对HTML文档进行操作。只要理解了DOM就可以随心所欲的操作WEB页面。
- 文档
  - 文档表示的就是整个的HTML网页文档
- 对象
  - 对象表示将网页中的每一个部分都转换为了一个对象。
- 模型
  - 使用模型来表示对象之间的关系，这样方便我们获取对
-

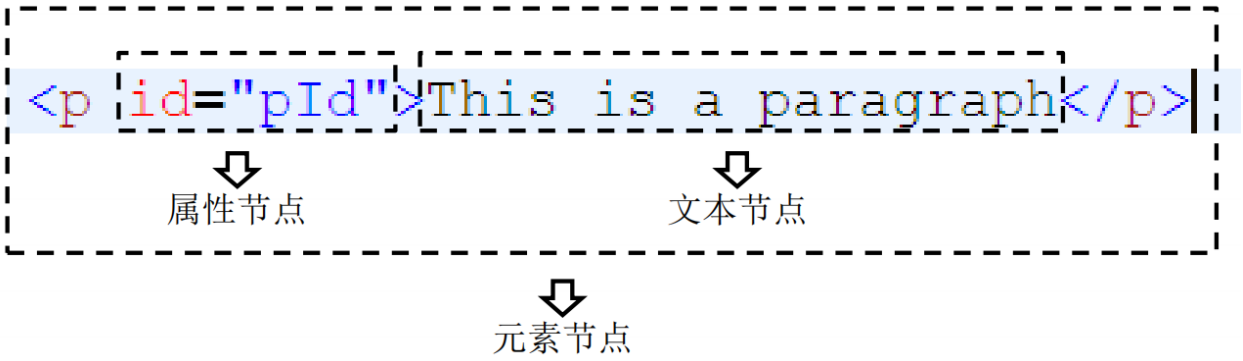
1.html

```
<html>
        <head>
                <title>网页的标题</title>
        </head>
        <body>
                <a href="1.html" >超连接</a>
        </body>
</html>
```



节点

- 节点Node，是构成我们网页的最基本的组成部分，网页中的每一个部分都可以称为是一个节点。
- 比如：html标签、属性、文本、注释、整个文档等都是一个节点。
- 虽然都是节点，但是实际上他们的具体类型是不同的。
- 比如：标签我们称为元素节点、属性称为属性节点、文本称为文本节点、文档称为文档节点。
- 节点的类型不同，属性和方法也都不尽相同。
- 节点：Node——构成HTML文档最基本的单元。
- 常用节点分为四类
  - 文档节点：整个HTML文档
  - 元素节点：HTML文档中的HTML标签
  - 属性节点：元素的属性
  - 文本节点：HTML标签中的文本内容
-



节点的属性

	nodeName	nodeType	nodeValue
文档节点	#document	9	null
元素节点	标签名	1	null
属性节点	属性名	2	属性值
文本节点	#text	3	★文本内容

文档节点（document）

- 文档节点document，代表的是整个HTML文档，网页中的所有节点都是它的子节点。
- document对象作为window对象的属性存在的，我们不用获取可以直接使用。
- 通过该对象我们可以在整个文档访问内查找节点对象，并可以通过该对象创建各种节点对象。

元素节点（Element）

- HTML中的各种标签都是元素节点，这也是我们最常用的一个节点。
- 浏览器会将页面中所有的标签都转换为一个元素节点，我们可以通过document的方法来获取元素节点。
- 比如：
  - document.getElementById()
  - 根据id属性值获取一个元素节点对象。

文本节点（Text）

- 文本节点表示的是HTML标签以外的文本内容，任意非HTML的文本都是文本节点。
- 它包括可以字面解释的纯文本内容。
- 文本节点一般是作为元素节点的子节点存在的。

- 获取文本节点时，一般先要获取元素节点。在通过元素节点获取文本节点。
- 例如：
  - 元素节点.firstChild;
  - 获取元素节点的第一个子节点，一般为文本节点

属性节点（Attribute）

- 属性节点表示的是标签中的一个一个的属性，这里要注意的是属性节点并非是元素节点的子节点，而是元素节点的一部分。
- 可以通过元素节点来获取指定的属性节点。
- 例如：
  - 元素节点.getAttributeNode("属性名");
- 注意：我们一般不使用属性节点。

事件

- 事件，就是文档或浏览器窗口中发生的一些特定的交互瞬间。
- JavaScript 与 HTML 之间的交互是通过事件实现的。
- 对于 Web 应用来说，有下面这些代表性的事件：点击某个元素、将鼠标移动至某个元素上方、按下键盘上某个键，等等。

获取元素节点

通过document对象调用

1. getElementById()– 通过id属性获取一个元素节点对象
2. getElementsByTagName()– 通过标签名获取一组元素节点对象
3. getElementsByName()– 通过name属性获取一组元素节点对象

获取元素节点的子节点

1. getElementsByTagName()– 方法，返回当前节点的指定标签名后代节点
2. childNodes– 属性，表示当前节点的所有子节点
3. children- 得到所有的元素子节点
4. firstChild– 属性，表示当前节点的第一个子节点
5. lastChild– 属性，表示当前节点的最后一个子节点

获取父节点和兄弟节点

1. parentNode– 属性，表示当前节点的父节点
2. previousSibling– 属性，表示当前节点的前一个兄弟节点
3. nextSibling– 属性，表示当前节点的后一个兄弟节点

元素节点的属性

获取，元素对象.属性名

```
element.value
element.id
element.className
```

设置，元素对象.属性名=新的值

```
element.value = “hello”
element.id = “id01”
element.className = “newClass”
```

其他属性

- nodeValue
  - 文本节点可以通过nodeValue属性获取和设置文本节点的内容
- innerHTML
  - 元素节点通过该属性获取和设置标签内部的html代码

使用CSS选择器进行查询

- querySelector()
- querySelectorAll()
- 这两个方法都是用document对象来调用，两个方法使用相同，都是传递一个选择器字符串作为参数，方法会自动根据选择器字符串去网页中查找元素。
- 不同的地方是querySelector()只会返回找到的第一个元素，而querySelectorAll()会返回所有符合条件的元素。

节点的修改

- 这里的修改我们主要指对元素节点的操作。
- 创建节点
  - document.createElement(标签名)
- 删除节点
  - 父节点.removeChild(子节点)
- 替换节点
  - 父节点.replaceChild(新节点 , |旧节点)
- 插入节点
  - 父节点.appendChild(子节点)
  - 父节点.insertBefore(新节点 , |旧节点)

事件

- 关于事件实际上我们已经初步接触过了，指的就是用户与浏览器交互的一瞬间。
- 我们通过为指定事件绑定回调函数的形式来处理事件，当指定事件触发以后我们的回调函数就会被调用，这样我们的页面就可以完成和用户的交互了。
- 这里我们还要更加深入的聊一聊事件的其他内容。

事件处理程序

- 我们可以通过两种方式为一个元素绑定事件处理程序：
  - 通过HTML元素指定事件属性来绑定
  - 通过DOM对象指定的属性来绑定
- 这两种方式都是我们日常用的比较多的，但是更推荐使用第二种方式。

- 还有一种方式比较特殊我们称为设置事件监听器。使用如下方式：
  - 元素对象.addEventListener()

通过HTML标签的属性设置

```
<button onclick="alert('hello');alert('world')">按钮</button>
```

- 这种方式当我们点击按钮以后，onclick属性中对应的JS代码将会执行，也就是点击按钮以后，页面中会弹出两个提示框。
- 这种方式我们直接将代码编写到了onclick属性中，可以编写多行js代码，当然也可以事先在外部定义好函数。
- 这种方式的优点在于，设定步骤非常简单，并且能够确保事件处理程序会在载入时被设定。
- 如果在函数的最后return false则会取消元素的默认行为。

通过DOM对象的属性绑定

但是其实上面的写法虽然简单，但却将JS和HTML的代码编写到了一起，并不推荐使用，我们更推荐如下的写法：

```
var btn = document.getElementById('btn');
btn.onclick = function(){alert("hello");
```

这种写法将HTML代码和JS写在不同的位置，维护起来更加容易。

设置事件监听器

- 前边两种方式都可以绑定事件处理程序，但是它们都有一个缺点就是都只能绑定一个程序，而不能为一个事件绑定多个程序。
- 这是我们就可以使用addEventListener()来处理，这个方法需要两个参数：一个是事件字符串，一个是响应函数。
- ```
  btn.addEventListener('click' , function(){alert("hello");}
  ```
- 但是要注意的是ie8以下的浏览器是不支持上边的方法的，需要使用attachEvent代替。
- 也可以使用removeEventListener()和detachEvent()移除事件

事件处理中的this

- 在事件处理程序内的 this 所引用的对象即是设定了该事件处理程序的元素。
- 也就是事件是给那个对象绑定的this就是哪个对象。

事件对象

- 在DOM对象上的某个事件被触发时，会产生一个事件对象Event，这个对象中包含着所有事件有关的信息。包括导致事件的元素、事件的类型以及其他与特定事件相关的信息。
- 例如，鼠标操作导致的事件对象中，会包含鼠标位置的信息，而键盘操作导致的事件对象中，会包含与按下的键有关的信息。所有浏览器都支持 event 对象，但支持方式不同。

事件对象

- DOM标准的浏览器会将一个event对象传入到事件的处理程序当中。无论事件处理程序是什么都会传入一个event对象。
- 可以通过这种方式获取：
  ```

- ```
  btn.onclick = function(event){
  alert(event.type);
  };
  ```
- Event对象包含与创建它的特定事件有关的属性和方法。触发的事件类型不一样，可用的属性和方法也不一样。
属性/方法	类型	读/写	说明
bubbles	Boolean	只读	事件是否冒泡
cancelable	Boolean	只读	是否可以取消事件的默认行为
currentTarget	Element	只读	当前正在处理的事件元素
defaultPrevented	Boolean	只读	是否调用了preventDefault()
detail	Number	只读	与事件相关的细节信息
eventPhase	Number	只读	阶段 1:捕获 2:目标 3:冒泡
preventDefault()	Function	只读	取消事件的默认行为
stopImmediatePropagation()	Function	只读	取消事件的进一步捕获或冒泡
stopPropagation()	Function	只读	取消事件的进一步捕获或冒泡
target	Element	只读	事件的目标
trusted	Boolean	只读	是否是浏览器内置事件
type	String	只读	被触发的事件的类型

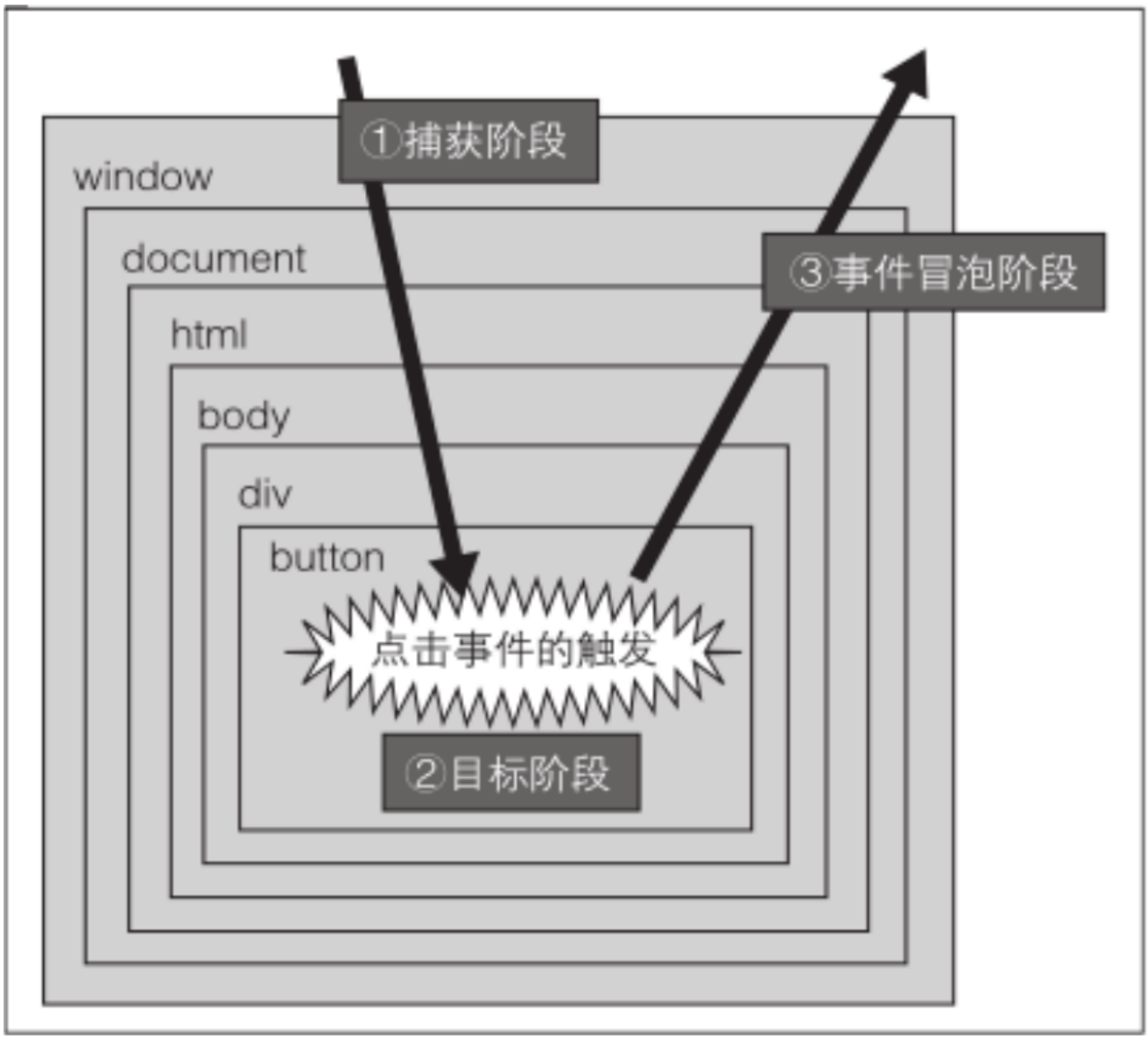
事件的触发

- 事件的发生主要是由用户操作引起的。
- 比如mousemove这个事件就是由于用户移动鼠标引起的，在鼠标指针移动的过程中该事件会持续发生。
- 当指定事件被触发时，浏览器就会调用对应的函数去响应事件，一般情况下事件没触发一次，函数就会执行一次。
- 因此设置鼠标移动的事件可能会影响到鼠标的移动速度。所以设置该类事件时一定要谨慎

事件的传播

- 在网页中标签与标签之间是有嵌套关系的，比如这样一个页面：
- ```
  <html>
      <body>
          <div id="foo">
              <button id="bar">sample</button>
          </div>
      </body>
  </htm
  ```
- 如果这时用户点击了sample按钮，则会以该按钮作为事件目标触发一次点击事件。
- 这时，事件的处理将会分为捕获阶段、目标阶段、事件冒泡这三个阶段

事件的传播流程



事件的传播

- 捕获阶段
  - 这一阶段会从window对象开始向下一直遍历到目标对象，如果发现有对象绑定了响应事件则做相应的处理。
- 目标阶段
  - 这一阶段已经遍历结束，则会执行目标对象上绑定的响应函数。
- 事件冒泡阶段
  - 这一阶段，事件的传播方式和捕获阶段正好相反，会从事件目标一直向上遍历，直至window对象结束，这时对象上绑定的响应函数也会执行。

取消事件传播

- 我们可以使用event对象的两个方法完成：
  - stopPropagation()
  - stopImmediatePropagation()
- 取消默认行为：
  - preventDefault()

BOM

- ECMAScript无疑是JavaScript的核心，但是要想在浏览器中使用JavaScript，那么BOM（浏览器对象模型）才是真正的核心。
- BOM 提供了很多对象，用于访问浏览器的功能，这些功能与任何网页内容无关。
- BOM将浏览器中的各个部分转换成了一个一个的对象，我们通过修改这些对象的属性，调用他们的方法，从而控制浏览器的各种行为。

window

- window对象是BOM的核心，它表示一个浏览器的实例。
- 在浏览器中我们可以通过window对象来访问操作浏览器，同时window也是作为全局对象存在的。
- 全局作用域：
    - window对象是浏览器中的全局对象，因此所有在全局作用域中声明的变量、对象、函数都会变成window对象的属性和方法。

窗口大小

浏览器中提供了四个属性用来确定窗口的大小：

网页窗口的大小 • innerWidth • innerHeight – 浏览器本身的尺寸 • outerWidth • outerHeight

系统对话框

- 浏览器通过 alert() 、 confirm() 和 prompt() 方法可以调用系统对话框向用户显示消息。
- 的外观由操作系统及（或）浏览器设置决定，而不是由 CSS 决定。
- 显示系统对话框时会导致程序终止，当关闭对话框程序会恢复执行。