|  |
| --- |
| **极**  **客**  **原**  **型**  **Java**  **前**  **端**  **技**  **术**  **集** |
|  |
| Create by Yangcl  2014年4月25日 星期五 |

**极**

**客**

**原**

**型**

**Java**

**前**

**端**

**技**

**术**

**集**

---- Create by Yangcl

2014年4月25日 星期五

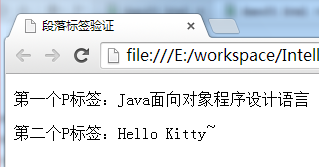
1. Html 基础
   1. Html基本标签
      1. 文本排版

* 段落标签<p>

主要用来放置一小段文字描述

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**段落标签验证**</title>**
5. **</head>**
6. **<body>**
7. **<p>**第一个P标签：Java面向对象程序设计语言**</p>**
9. **<p>**第二个P标签：Hello Kitty~ **</p>**
10. **</body>**
11. **</html>**

页面效果截图如下：

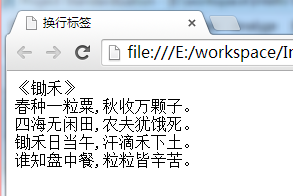


* 换行标签<br>

主要对一段文字进行换行操作

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**换行标签**</title>**
5. **</head>**
6. **<body>**
7. 《锄禾》**<br/>**
8. 春种一粒粟,秋收万颗子。**<br/>**
9. 四海无闲田,农夫犹饿死。**<br/>**
10. 锄禾日当午,汗滴禾下土。 **<br/>**
11. 谁知盘中餐,粒粒皆辛苦。**<br/>**
12. **</body>**
13. **</html>**

页面效果截图如下：

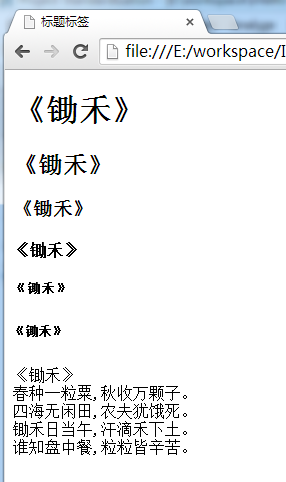


* 标题标签<h1>~<h6>

用来设定一段文字的大小，给出了6种不同的字体尺寸。默认还会帮助此段字体自动换行。

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**标题标签**</title>**
5. **</head>**
6. **<body>**
7. **<h1>**《锄禾》**</h1>**
8. **<h2>**《锄禾》**</h2>**
9. **<h3>**《锄禾》**</h3>**
10. **<h4>**《锄禾》**</h4>**
11. **<h5>**《锄禾》**</h5>**
12. **<h6>**《锄禾》**</h6>**
13. 《锄禾》**<br/>**
14. 春种一粒粟,秋收万颗子。**<br/>**
15. 四海无闲田,农夫犹饿死。**<br/>**
16. 锄禾日当午,汗滴禾下土。 **<br/>**
17. 谁知盘中餐,粒粒皆辛苦。**<br/>**
18. **</body>**
19. **</html>**

页面效果截图如下：

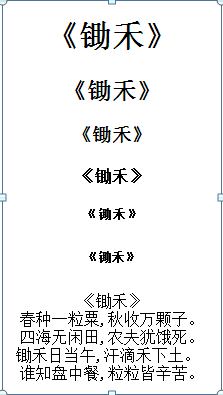


* 文字居中标签<center>

将一段文字在页面或一个区域进行居中，在Html5规范中已经废弃

1. **<head>**
2. **<meta** charset="UTF-8"**>**
3. **<title>**文字居中标签**</title>**
4. **</head>**
5. **<body>**
6. **<center>**   // 文字居中标签
7. **<h1>**《锄禾》**</h1>**
8. **<h2>**《锄禾》**</h2>**
9. **<h3>**《锄禾》**</h3>**
10. **<h4>**《锄禾》**</h4>**
11. **<h5>**《锄禾》**</h5>**
12. **<h6>**《锄禾》**</h6>**
13. 《锄禾》**<br/>**
14. 春种一粒粟,秋收万颗子。**<br/>**
15. 四海无闲田,农夫犹饿死。**<br/>**
16. 锄禾日当午,汗滴禾下土。 **<br/>**
17. 谁知盘中餐,粒粒皆辛苦。**<br/>**
18. **</center>**
19. **</body>**
20. **</html>**

页面效果截图如下：

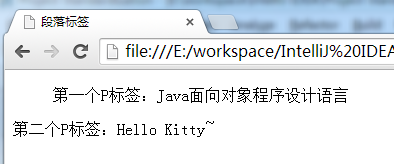


* 文字段落缩进标签<blockquote>

对一段文字进行缩进。

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**段落标签**</title>**
5. **</head>**
6. **<body>**
7. **<p>**
8. **<blockquote>**第一个P标签：Java面向对象程序设计语言**</blockquote>**
9. **</p>**
10. **<p>**第二个P标签：Hello Kitty~**</p>**
11. **</body>**
12. **</html>**

页面效果截图如下：



* + 1. 文字列表

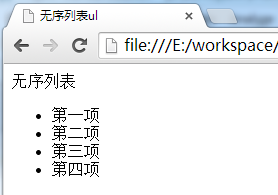
文字列表的主要作用是有序地编排一些信息资源，使其结构化和条理化，并以列表的样式显示出来。更多内容参见2.8节 列表高级操作。

* 无序列表 ul

无序列表，标签为 ul。

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**无序列表 ul **</title>**
5. **</head>**
6. **<body>**
7. 无序列表
8. **<ul>**
9. **<li>**第一项**</li>**
10. **<li>**第二项**</li>**
11. **<li>**第三项**</li>**
12. **<li>**第四项**</li>**
13. **</ul>**
14. **</body>**
15. **</html>**

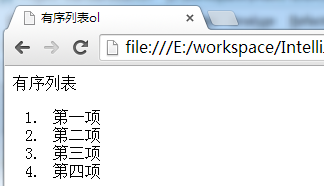
页面效果截图如下：



* 有序列表 ol

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**有序列表 ol **</title>**
5. **</head>**
6. **<body>**
7. 有序列表
8. **<ol>**
9. **<li>**第一项**</li>**
10. **<li>**第二项**</li>**
11. **<li>**第三项**</li>**
12. **<li>**第四项**</li>**
13. **</ol>**
14. **</body>**
15. **</html>**

页面效果截图如下：



* 1. HTML标记 与 属性

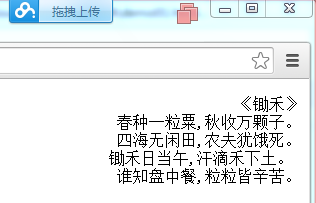
在大多数HTML标记中都可以对属性控制，属性的作用是帮助HTML标记进一步控制HTML文件的内容，比如内容的对齐方式（如本例），文字的大小、字体、颜色，网页的背景样式，图片的插入，等等。其基本语法为： <标记名称 属性名1= “属性值1” 属性名2= “属性值2” ......>

* + 1. align属性 控制段落的水平位置

align属性可以控制段落水平的位置，默认情况是左对齐，其可选值为：left/center/right。

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**align属性**</title>**
5. **</head>**
6. **<body>**
7. **<p** align="right"**>**
8. 《锄禾》**<br/>**
9. 春种一粒粟,秋收万颗子。**<br/>**
10. 四海无闲田,农夫犹饿死。**<br/>**
11. 锄禾日当午,汗滴禾下土。 **<br/>**
12. 谁知盘中餐,粒粒皆辛苦。**<br/>**
13. **</p>**
14. **</body>**
15. **</html>**

页面效果截图如下：



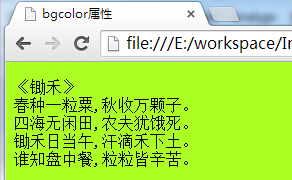
如上图，右对齐。

* + 1. bgcolo属性 设置背景颜色

背景色设置。body元素有的属性。

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**bgcolo属性**</title>**
5. **</head>**
6. **<body** bgcolor="#ABFF23"**>**
7. **<p** align="left"**>**
8. 《锄禾》**<br/>**
9. 春种一粒粟,秋收万颗子。**<br/>**
10. 四海无闲田,农夫犹饿死。**<br/>**
11. 锄禾日当午,汗滴禾下土。 **<br/>**
12. 谁知盘中餐,粒粒皆辛苦。**<br/>**
13. **</p>**
14. **</body>**
15. **</html>**

页面效果截图如下：



* 直接颜色表示 如red、blue

内容略。

* 利用R/G/B表示 16进制 #FFFF00

内容略。

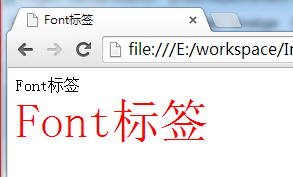
* + 1. 文字特殊样式

|  |  |
| --- | --- |
| **标记名称** | **显示效果** |
| <i></i> | 文字以斜体方式显示 |
| <u></u> | 文字以加下划线方式显示 |
| <s></s> | 文字以加下删除方式显示 |
| <b></b> | 文字以粗体方式显示 |
| <small></small> | 文字以缩小方式显示 |
| <strong></strong> | 文字以加强强调方式显示 |
| <em></em> | 文字以强调方式显示 |
| <address></address> | 用来显示电子邮件地址或网址 |
| <code></code> | 用来说明代码与指令 |
| <big></big> | 文字以放大方式显示 |

* + 1. 设置文字的大小和颜色 <font> 标签

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**Font标签**</title>**
5. **</head>**
6. **<body>**
7. Font标签**<br/>**
8. **<font** color="red" size="10" face="宋体"**>**Font标签**</font>**
9. **</body>**
10. **</html>**

页面效果截图如下：

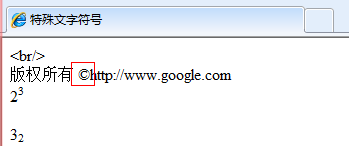


* 1. 特殊文字符号

|  |  |
| --- | --- |
| **符号** | **HTML代码** |
| < | &lt; |
| > | &gt; |
| © | &copy; |
| 上标 | <sup></sup> |
| 下标 | <sub></sub> |

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**特殊文字符号**</title>**
5. **</head>**
6. **<body>**
7. **<br/><br/>**
8. 版权所有 &copy;http://www.google.com**<br/>**
9. 2**<sup>**3**</sup>** **<br/>**
10. 3**<sub>**2**</sub>**
11. **</body>**
12. **</html>**

页面效果截图如下：



* 1. 网页图像img

示例代码如下：

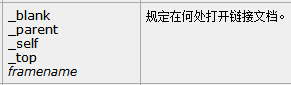
1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**用title属性为图像设置标题**</title>**
5. **</head>**
6. **<body>**
7. <!--相对路径 取得父级图片资源-->
8. **<img** width="150" height="200"src="../yan.jpg"**>**
10. <!--相对路径 取得本级图片资源 为图片添加标题-->
11. **<img** width="150" height="200" title="美女刘亦菲" src="liu.jpg"**/><br/>**
13. <!--使用绝对路径引用图片-->
14. **<img** src="http://www.baidu.com/img/bdlogo.gif"**/>**
16. <!--如果未找到liu2.jpg, alt属性会显示-->
17. **<img** width="150" height="200" alt="这是刘亦菲" src="liu2.jpg"**/>**
18. **</body>**
19. **</html>**
    * 1. 图片路径与尺寸
20. <!--相对路径 取得父级图片资源-->
21. **<img** **width="150" height="200"** src="../yan.jpg"**>**
23. <!--相对路径 取得本级图片资源 为图片添加标题-->
24. **<img** width="150" height="200" title="美女刘亦菲" src="liu.jpg"**/><br/**
    * 1. alt属性与图像替换文本
25. <!--如果未找到liu2.jpg, alt属性会显示-->
26. **<img** width="150" height="200" **alt="这是刘亦菲"** src="liu2.jpg"**/>** 
    * 1. t**itle属性与图像标题**
27. <!--相对路径 取得本级图片资源 为图片添加标题-->
28. **<img** width="150" height="200"**title="美女刘亦菲**" src="liu.jpg"**/><br/>**

页面效果截图如下：



* 1. HTML超链接

Html超链接主要使用 <a> 标签；在<a> 标签中有两个属性非常常用：href 和target。Href的内容是一个URL地址；target有以下几种可选状态：



经过测试，\_blank是在新的空白页打开链接；而framename则是在一个指定的区域打开，这个区域是你定的，后面会有介绍例子，其他的三个属性相互区别貌似不大。

* 基本文字超链接

1. **<a** href="http://http://www.csdn.net/"  target="\_blank"**>**
2. CSDN首页
3. **</a>**

* 图片超链接

1. **<a** href="https://www.google.com.hk" target="\_blank"**>**
2. **<img** src="google.jpg"**/>**
3. **</a>**

* 新窗口显示链接页面

进行如下设置即可：target="\_blank"。将会在一个新的空白页打开这个链接。

* 电子邮件链接

mailto是一个关键字，在<a>标签中键入即可。

**<a** href="mailto:794867067@qq.com"**>**联系我哦！**</a>**

* 1. **框架之间链接**

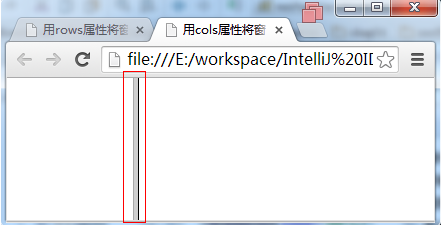
框架是一种常用的网页布局工具。它的作用是把浏览器的显示空间分割为几个部分，每个部分都可以独立显示不同的网页。框架链接常用的两个标签是<frameset>和<frame>。利用这两个标签可以将一个页面分区域，但在新的Html5规范中，他们被废弃了。

不能与 <frameset></frameset> 标签一起使用 <body></body> 标签。不过，如果您需要为不支持框架的浏览器添加一个 <noframes> 标签，请务必将此标签放置在 <body></body> 标签中。

* cols属性将窗口分为 左右 两部分

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**用cols属性将窗口分为左右两部分**</title>**
5. **</head>**
6. **<frameset** cols="30%,\*"**>**
7. **<frame>**
8. **<frame>**
9. **</frameset>**
10. **</html>**

页面效果截图如下：

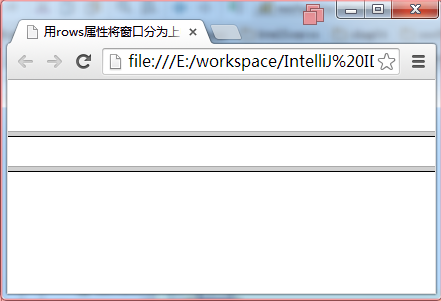


从图中红方框圈出来的地方就是页面被分割的样子

* 用rows属性将窗口分为上中下三部分

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**用rows属性将窗口分为上中下三部分**</title>**
5. **</head>**
6. **<frameset** rows="30%,10%,\*"**>**
7. **<frame>**
8. **<frame>**
9. **<frame>**
10. **</frameset>**
11. **</html>**

页面效果截图如下：

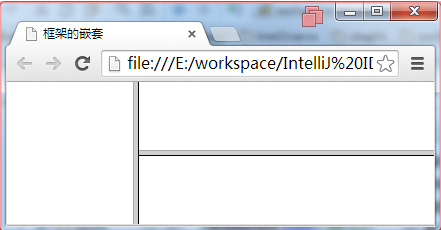


* 框架的嵌套

对页面分块

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**框架的嵌套**</title>**
5. **</head>**
6. **<frameset** cols="30%,\*"**>**  //第一步：页面先分左右
7. **<frame>**  // 左边包含的页面
8. **<frameset** rows="50%,\*"**>**  // 右边包含页面第二次分割
9. **<frame>**  // 上部分
10. **<frame>**  // 下部分
11. **</frameset>**
12. **</frameset>**
13. **</html>**

页面效果截图如下：

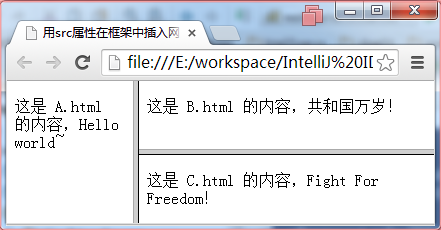


* <frame>标签src属性在框架中插入网页

根据上图中分的三个区域，对应放入3个简单的网页内容，代码如下：

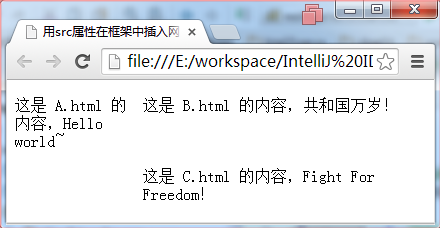
1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>** A.html **</title>**
5. **</head>**
6. **<body>**
7. **<p>**这是 A.html 的内容，Hello world~ **</p>**
8. **</body>**
9. **</html>**
11. **<html>**
12. **<head>**
13. **<meta** charset="UTF-8"**>**
14. **<title>** B.html **</title>**
15. **</head>**
16. **<body>**
17. **<p>**这是 B.html 的内容，共和国万岁！ **</p>**
18. **</body>**
19. **</html>**
21. **<html>**
22. **<head>**
23. **<meta** charset="UTF-8"**>**
24. **<title>** C.html **</title>**
25. **</head>**
26. **<body>**
27. **<p>**这是 C.html 的内容，Fight For Freedom！ **</p>**
28. **</body>**
29. **</html>**
31. **<html>**
32. **<head>**
33. **<meta** charset="UTF-8"**>**
34. **<title>**用src属性在框架中插入网页**</title>**
35. **</head>**
36. **<frameset** cols="30%,\*"**>**
37. **// 在frame 标签中，利用src属性可以引入页面**
38. **<frame**  src="a.html"**>**
40. **<frameset** rows="50%,\*"**>**
41. **<frame**  src="b.html"**>**
42. **<frame**  src="c.html"**>**
43. **</frameset>**
45. **</frameset>**
46. **</html>**

页面效果截图如下：



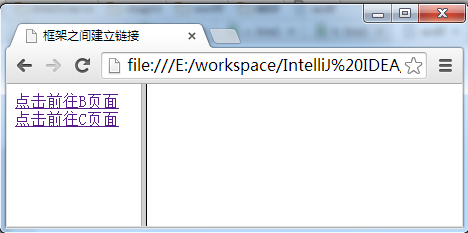
上图是有边框的页面，通过设置frameborder="0"，可以去除边框。

<frameset cols="30%,\*" frameborder="0"> 页面效果如下：



* 框架之间建立超链接

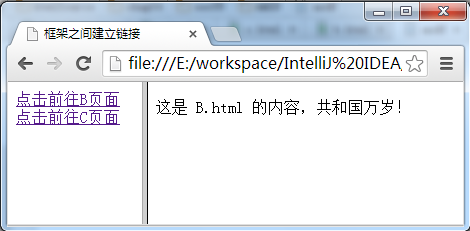
利用这个功能可以创建一个导航框架。

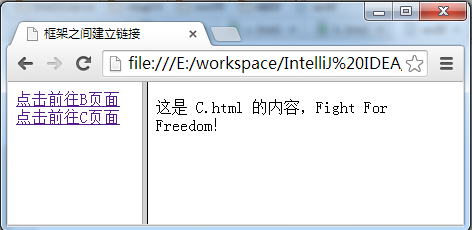


代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**B.html**</title>**
5. **</head>**
6. **<body>**
7. **<p>**这是 B.html 的内容，共和国万岁！ **</p>**
8. **</body>**
9. **</html>**
11. **<html>**
12. **<head>**
13. **<meta** charset="UTF-8"**>**
14. **<title>**C.html**</title>**
15. **</head>**
16. **<body>**
17. **<p>**这是 C.html 的内容，Fight For Freedom！ **</p>**
18. **</body>**
19. **</html>**
21. **<html>**
22. **<head>**
23. **<meta** charset="UTF-8"**>**
24. **<title>**A.html**</title>**
25. **</head>**
26. **<body>**
27. **<a** href="b.html" **target="myShow">**点击前往B页面**</a>**
28. **<br/>**
29. **<a** href="c.html" **target="myShow">**点击前往C页面**</a>**
30. **</body>**
31. **</html>**
33. **<html>**
34. **<head>**
35. **<meta** charset="UTF-8"**>**
36. **<title>**框架之间建立链接**</title>**
37. **</head>**
38. **<frameset** cols="30%,\*"**>**
39. **<frame** src="a.html"**>**
40. **<frame** name="myShow" **>**
41. **</frameset>**
42. **</html>**

页面效果截图如下：





主页面的右边框架给name属性命名，告诉链接跳往的地方那个框架的名字叫myShow

1. **<frameset** cols="30%,\*"**>**
2. **<frame** src="a.html"**>**
3. **<frame** name="myShow" **>**
4. **</frameset>**

<a>标签的target属性指向名字为myShow的框架

1. **<a** href="b.html" target="myShow"**>**点击前往B页面**</a>**
2. **<br/>**
3. **<a** href="c.html" target="myShow"**>**点击前往C页面**</a>**
   1. 嵌入式框架iframe
4. **<html>**
5. **<head>**
6. **<meta** charset="UTF-8"**>**
7. **<title>**嵌入式框架 iframe**</title>**
8. **</head>**
9. **<body>**
10. 页面嵌入CSDN网页**<br/>**
11. **<iframe** width="900" height="600" src="http://www.csdn.net"**></iframe>**
12. **</body>**
13. **</html>**

页面效果截图如下：



* + 1. TODO : frame与iframe的区别
  1. 表格基础

<table />标签定义 HTML 表格。简单的 HTML 表格由 table 元素以及一个或多个 tr、th 或 td 元素组成。tr 元素定义表格行，th 元素定义表头，td 元素定义表格单元。

|  |  |  |
| --- | --- | --- |
| **属性** | **值** | **描述** |
| [align](http://www.w3school.com.cn/tags/att_table_align.asp) | * Left * Center * right | 规定表格相对周围元素的对齐方式 |
| [bgcolor](http://www.w3school.com.cn/tags/att_table_bgcolor.asp) | * rgb(x,x,x) * #xxxxxx * colorname | 规定表格的背景颜色 |
| [border](http://www.w3school.com.cn/tags/att_table_border.asp) | pixels | 规定表格边框的宽度。 |
| [cellpadding](http://www.w3school.com.cn/tags/att_table_cellpadding.asp) | * *pixels%* | 规定单元边沿与其内容之间的空白。 |
| [cellspacing](http://www.w3school.com.cn/tags/att_table_cellspacing.asp) | *Pixels*  *%* | 规定单元格之间的空白。 |
| [frame](http://www.w3school.com.cn/tags/att_table_frame.asp) | * void * above * below * hsides * lhs * rhs * vsides * box * border | 规定外侧边框的哪个部分是可见的。 |
| [rules](http://www.w3school.com.cn/tags/att_table_rules.asp) | * none * groups * rows * cols * all | 规定内侧边框的哪个部分是可见的 |
| [summary](http://www.w3school.com.cn/tags/att_table_summary.asp) | *text* | 规定表格的摘要。 |
| [width](http://www.w3school.com.cn/tags/att_table_width.asp) | * *%* * *pixels* | 规定表格的宽度。 |

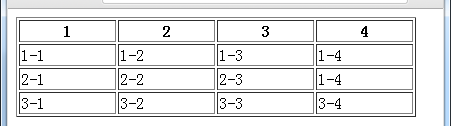
**<table />标签重要作用之一是设置页面内容的对其。**

* 表格基础属性

示例代码如下：

1. **<title>**表格的基本结构**</title>**
2. **</head>**
3. **<body>**
4. **<table** border="1" width ='400'**>**
5. **<tr>**
6. **<th>**1**</th>**
7. **<th>**2**</th>**
8. **<th>**3**</th>**
9. **<th>**4**</th>**
10. **</tr>**
11. **<tr>**
12. **<td>**1-1**</td>**
13. **<td>**1-2**</td>**
14. **<td>**1-3**</td>**
15. **<td>**1-4**</td>**
16. **</tr>**
17. **<tr>**
18. **<td>**2-1**</td>**
19. **<td>**2-2**</td>**
20. **<td>**2-3**</td>**
21. **<td>**1-4**</td>**
22. **</tr>**
23. **<tr>**
24. **<td>**3-1**</td>**
25. **<td>**3-2**</td>**
26. **<td>**3-3**</td>**
27. **<td>**3-4**</td>**
28. **</tr>**
29. **</table>**
30. **</body>**

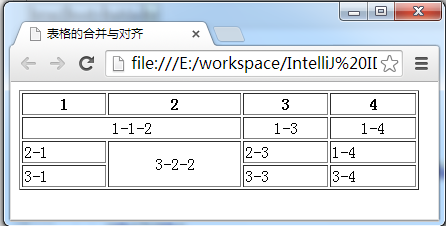
页面效果截图如下：



* 表格的合并与对其

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**表格的合并与对齐**</title>**
5. **</head>**
6. **<body>**
7. **<table** border="1" width="400"**>**
8. **<tr>**   // 第一行
9. **<th>**1**</th>**
10. **<th>**2**</th>**
11. **<th>**3**</th>**
12. **<th>**4**</th>**
13. **</tr>**
14. **<tr** align="center"**>**   // 对第二行进行整行全部居中
15. **<td** colspan="2"**>**1-1-2**</td>**  // 合并第二行的1列、2列
16. **<td>**1-3**</td>**
17. **<td>**1-4**</td>**
18. **</tr>**
19. **<tr>**
20. **<td>**2-1**</td>**
21. // 合并第二列的3行与4行
22. **<td** rowspan="2" align="center"**>**3-2-2**</td>**
23. **<td>**2-3**</td>**
24. **<td>**1-4**</td>**
25. **</tr>**
26. **<tr>**
27. **<td>**3-1**</td>**
28. **<!--<td>3-2</td>-->** 受行合并影响，这里删除
29. **<td>**3-3**</td>**
30. **<td>**3-4**</td>**
31. **</tr>**
32. **</table>**
33. **</body>**
34. **</html>**

在列合并中(colspan)同一行下的td标签减少；在行合并中(rowspan)下一行的td标签对应减少。页面效果截图如下：

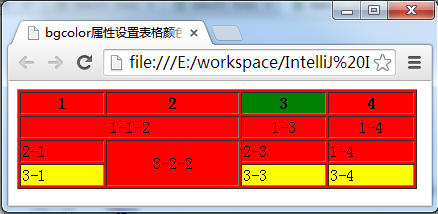


* 表格颜色设定

对<tr>标签设定整行的颜色都会变，对<td>标签进行颜色设定，只有那个单元格的颜色会变，不会影响到整个列的颜色改变。

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**用bgcolor属性设置表格背景色和边框颜色**</title>**
5. **</head>**
6. **<body>**
7. **<table** border="1" width="400" bgcolor="red"**>**
8. **<tr>**
9. **<th>**1**</th>**
10. **<th>**2**</th>**
11. **<th**  bgcolor="green"**>**3**</th>**
12. **<th>**4**</th>**
13. **</tr>**
14. **<tr** align="center"**>**
15. **<td** colspan="2"**>**1-1-2**</td>**
16. **<td>**1-3**</td>**
17. **<td>**1-4**</td>**
18. **</tr>**
19. **<tr>**
20. **<td>**2-1**</td>**
21. **<td** rowspan="2" align="center"**>**3-2-2**</td>**
22. **<td>**2-3**</td>**
23. **<td>**1-4**</td>**
24. **</tr>**
25. **<tr** bgcolor="yellow"**>**
26. **<td>**3-1**</td>**
27. **<td>**3-3**</td>**
28. **<td>**3-4**</td>**
29. **</tr>**
30. **</table>**
31. **</body>**
32. **</html>**

页面效果截图如下：

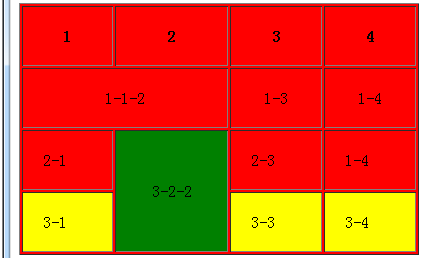


* cellpadding属性

Cellpadding用于控制表格内文字与表格内边距的值。

**<table** border="1" width="400" bgcolor="red" cellpadding="20px;" **>**

页面效果截图如下：

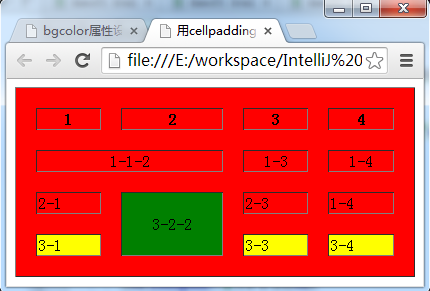


* cellspacing属性

cellspacing属性用于控制表格边线的宽度。

**<table** border="1" width="400" bgcolor="red" cellspacing="20px;"**>**

页面效果截图如下：



* 1. HTML表单
     1. Form表单

<form> 标签用于为用户输入创建 HTML 表单。表单能够包含 input 元素，比如文本字段、复选框、单选框、提交按钮等等。表单还可以包含 menus、textarea、fieldset、legend 和 label 元素。

<form />元素是块级元素，其前后会产生折行；Html的其他表单都依赖于Form表单，脱离了<form />元素的而存在的情况不多见。<form />元素表单最大的作用是：用于向服务器传输数据。

<input /> 标签用于搜集用户信息；<form />和<input />这两个标签组成了Html表单的基本结构。<input />标签根据不同的 type 属性值，输入字段拥有很多种形式。输入字段可以是文本字段、复选框、掩码后的文本控件、单选按钮、按钮等等。

所以，Html表单的重点有两个 <form />标签和<input />标签。

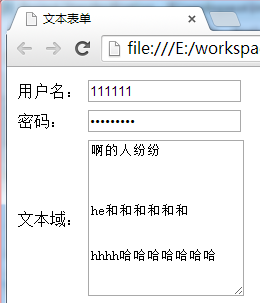
* + 1. 文本表单

文本表单包括：文本框(<input type="text" />)、密码框(<input type="password" />)和文本域(<**textarea** id="desc" /> 注意：文本域有自己独立的标签，不依赖<input />标签)。他们的区别是：<input />标签根据不同的 type属性值来确定到底是哪一个。

代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**文本表单**</title>**
5. **</head>**
6. **<body>**
7. **<form** action="testaaa.jsp" method="post"**>**
8. **<table>**
9. **<tr>**
10. **<td>**
11. 用户名：
12. **</td>**
13. **<td>**
14. **<input** type="text" id="userName" name="userName"**/>**
15. **</td>**
16. **</tr>**
18. **<tr>**
19. **<td>**
20. 密码：
21. **</td>**
22. **<td>**
23. **<input** type="password" id="pwd" name="pwd"**/>**
24. **</td>**
25. **</tr>**
27. **<tr>**
28. **<td>**
29. 文本域：
30. **</td>**
31. **<td** colspan="1"**>**
32. **<textarea** id="desc"  rows="10" cols="19"**></textarea>**
33. **</td>**
34. **</tr>**
35. **</table>**
37. **</form>**
38. **</body>**
39. **</html>**

页面效果截图如下：

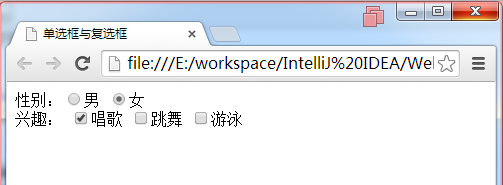


* + 1. 单选框与复选框

不多说，看代码。

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**单选框与复选框**</title>**
5. **</head>**
6. **<body>**
7. 性别：**<input** type="radio" name="sex" **/>**男
8. **<input** type="radio" name="sex" checked="checked"**/>**女
9. **<br/>**
10. 兴趣：
11. **<input** type="checkbox" name="hobby" checked="checked"**/>**唱歌
12. **<input** type="checkbox" name="hobby" **/>**跳舞
13. **<input** type="checkbox" name="hobby" **/>**游泳
14. **</body>**
15. **</html>**

页面效果截图如下：

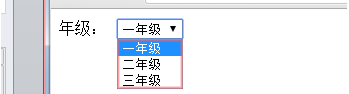


* + 1. 下拉框

不多说，看代码：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**下拉框**</title>**
5. **</head>**
6. **<body>**
7. 年级：
8. **<select** id="grade"  name="grade"**>**
9. **<option** value="1"**>**一年级**</option>**
10. **<option** value="2"**>**二年级**</option>**
11. **<option** value="3"**>**三年级**</option>**
12. **</select>**
13. **</body>**
14. **</html>**

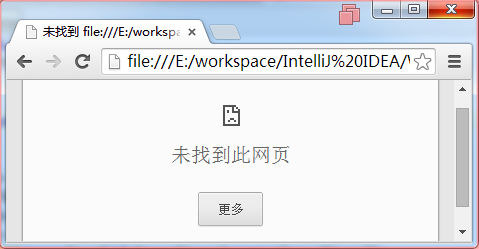
页面效果截图如下：



* + 1. 按钮 思密达

在Html中按钮分为两种：普通按钮(button) 和 提交按钮(submit)。值得指出的是提交按钮，这种按钮一般配合<form />标签，点击他会默认向服务器发送数据。加入服务器端没有定义一个servlet或Action的话，会报错；但普通按钮(button)类型不会报错。

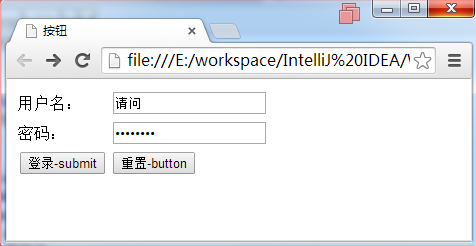
提交按钮(submit)方式报错截图如下：



示例代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**按钮**</title>**
5. **</head>**
6. **<body>**
7. **<form** action="a.jsp" method="get"**>**
8. **<table>**
9. **<tr>**
10. **<td>**
11. 用户名：
12. **</td>**
13. **<td>**
14. **<input** type="text" id="weq" name="userName"**/>**
15. **</td>**
16. **</tr>**
17. **<tr>**
18. **<td>**
19. 密码：
20. **</td>**
21. **<td>**
22. **<input** type="password" id="pwd" name="pwd"**/>**
23. **</td>**
24. **</tr>**
25. **<tr>**
26. **<td>**
27. **<input** type="submit" value="登录-submit"**/>**
28. **</td>**
29. **<td>**
30. **<input** type="button" value="重置-button"**/>**
31. **</td>**
32. **</tr>**
33. **</table>**
34. **</form>**
35. **</body>**
36. **</html>**

页面效果截图如下：

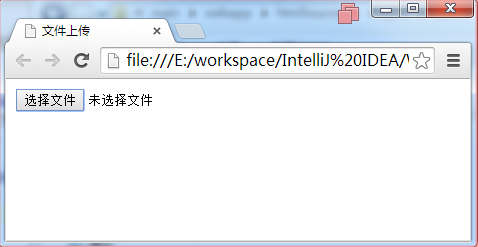


* + 1. 文件上传

需要指出的是：不同浏览器对文件上传的解释行为不同，样式也就不一样。IE的比较好看。

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**文件上传**</title>**
5. **</head>**
6. **<body>**
7. **<input** type="file" id="f" name="f"**/>**
8. **</body>**
9. **</html>**

页面效果截图如下：



* + 1. <fieldset /> 带标题的框

fieldset 元素可将表单内的相关元素分组。<fieldset> 标签将表单内容的一部分打包，生成一组相关表单的字段。当一组表单元素放到 <fieldset> 标签内时，浏览器会以特殊方式来显示它们，它们可能有特殊的边界、3D 效果，或者甚至可创建一个子表单来处理这些元素。

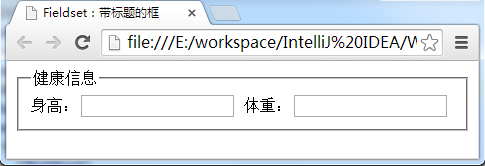
<fieldset> 标签没有必需的或唯一的属性。

<legend> 标签为 fieldset 元素定义标题。

代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**Fieldset：带标题的框**</title>**
5. **</head>**
6. **<body>**
7. **<form>**
8. **<fieldset>**
9. **<legend>**健康信息**</legend>**
10. 身高：**<input** type="text" **/>**
11. 体重：**<input** type="text" **/>**
12. **</fieldset>**
13. **</form>**
14. **</body>**
15. **</html>**

页面效果截图如下：

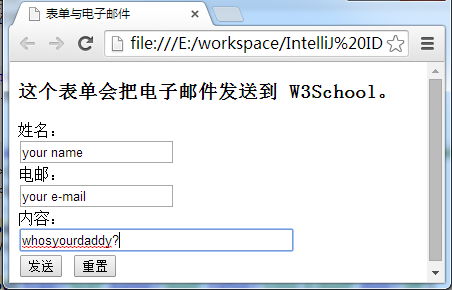


* + 1. Form表单与电子邮件

不多说，看代码

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**表单与电子邮件**</title>**
5. **</head>**
6. **<body>**
7. **<form** action="MAILTO:someone@w3school.com.cn" method="post" enctype="text/plain"**>**
8. **<h3>**这个表单会把电子邮件发送到 W3School。**</h3>**
9. **<p></p>**
10. 姓名：**<br** **/>**
11. **<input** type="text" name="name" value="your name" size="20"**>**
12. **<br** **/>**
13. 电邮：**<br** **/>**
14. **<input** type="text" name="mail" value="your e-mail" size="20"**>**
15. **<br** **/>**
16. 内容：**<br** **/>**
17. **<input** type="text" name="comment"  size="40"**>**
18. **<br** **/>**
19. **<input** type="submit" value="发送"**>**
20. **<input** type="reset" value="重置"**>**
22. **</form>**
23. **</body>**
24. **</html>**

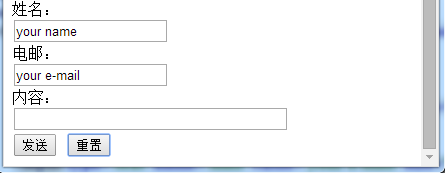
页面效果截图如下：



form标签的 enctype属性 规定在发送表单数据之前如何对其进行编码，enctype属性的可选值如下，更多参考查阅MIME的类型。

* text/html是html格式的正文
* text/plain是无格式正文
* text/xml忽略xml头所指定编码格式而默认采用us-ascii编码
* application/xml会根据xml头指定的编码格式来编码：

这里值得指出的另一点是：重置按钮。**<input** type="reset" value="重置"**>**  重置执行后只仅仅清空了【内容】下的文字，而【姓名】和【电邮】内的默认值没有清空。



* 1. HTML块：<div>与<span>
     1. <div />块元素
* 定义和用法

<div> 可定义文档中的分区或节（division/section）。

<div> 标签可以把文档分割为独立的、不同的部分。它可以用作严格的组织工具，并且不使用任何格式与其关联。

如果用 id 或 class 来标记 <div>，那么该标签的作用会变得更加有效。

* 用法

<div> 是一个块级元素。这意味着它的内容自动地开始一个新行。实际上，换行是 <div> 固有的唯一格式表现。可以通过 <div> 的 class 或 id 应用额外的样式。

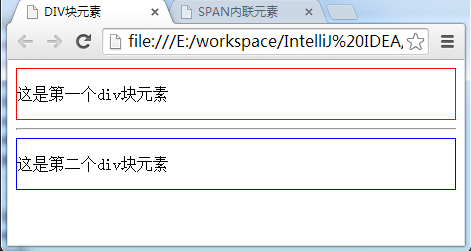
不必为每一个 <div> 都加上类或 id，虽然这样做也有一定的好处。

可以对同一个 <div> 元素应用 class 或 id 属性，但是更常见的情况是只应用其中一种。这两者的主要差异是，class 用于元素组（类似的元素，或者可以理解为某一类元素），而 id 用于标识单独的唯一的元素。

示例代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**DIV块元素**</title>**
5. **</head>**
6. **<body>**
7. **<div** id="d1" style="border: 1px red solid;"**>**  // **solid是边框属性**
8. **<p>**这是第一个div块元素**</p>**
9. **</div>**
10. **<hr/>**
11. **<div** id="d2" style="border: 1px blue solid;"**>**
12. **<p>**这是第二个div块元素**</p>**
13. **</div>**
14. **</body>**
15. **</html>**

页面效果截图如下：



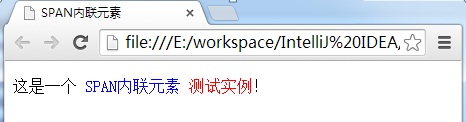
* + 1. <span />组合文档行内元素

<span />元素可以对行内文档内容进行小调整，类似一种小的块级元素。

示例代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**SPAN内联元素**</title>**
5. **</head>**
6. **<body>**
7. **<p>**这是一个
8. **<span** style="color: blue"**>**SPAN内联元素**</span>**
9. **<span** style="color: red"**>**测试实例**</span>**！
10. **</p>**
11. **</body>**
12. **</html>**

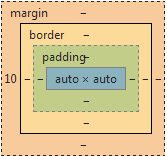
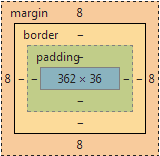
页面效果截图如下：



* + 1. 块级元素&行内元素

块级元素：以一个块的形式展现，跟同级的兄弟块依次竖直排列，左右撑满，比如<div>；

行内元素：以普通的一个DOM 节点展现，跟同级的兄弟元素横向排列，排满后，自动换行，比如<span>。<span>的盒模型content显示为auto x auto。

span盒模型 div盒模型

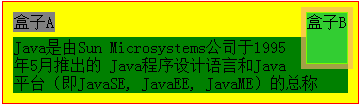
块级元素的盒模型的content则是一个具体的宽x 高的数值。这个数值是的宽是依据你浏览器窗口的大小来定的，比如362这个数字就是他在浏览器中的实际宽度。

1. CSS In Action

* **框**

在CSS中，一切都是：“框”。 这是CSS的基本思想，所看到浏览器的内容都是所有html元素“层次叠加”后展现的效果。

如下图所示：



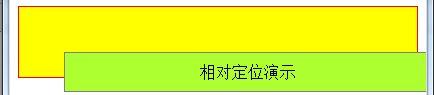
CSS盒模型的框体

在标准文档流中，不加入CSS控制，html元素分为“块级元素”和“行内元素”两种类型。块级元素在标准流中独占一行，撑满浏览器；典型代表是div、h1 或 p 元素，这些元素显示为一块内容，即“块框”。浏览器对它们的解释是一行一行向下堆积，组成完整的页面。

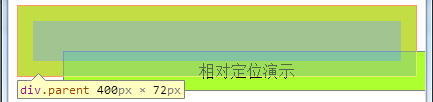
span 和 strong 等元素称为“行内元素”，它们的内容显示在行中，即“行内框”。行内元素则是从左至右，一个一个加入；就像文字的书写顺序。

* **层叠样式的真实模样**

CSS：层叠样式单。对Html元素的层叠展示，利用上下左右和前后全方位展示一个Html元素是对“层叠”最好的解释。如下面两幅图：



浏览器展示



盒模型真实解析

在浏览器对盒模型真实解析中可以清晰的看到，浅绿色的区域部分覆盖、叠加在了黄色的区域上。这两幅图是一个CSS position属性对层叠的效果展示。

* 1. Css核心基础

在CSS的定义中，class属性不是唯一的，但id属性是具有唯一性的。所以类选择器的灵活性是最高的。派生选择器、属性选择器和ID选择器的功能在类选择器中都可以控制。

**层叠次序**

当同一个HTML元素被不止一个样式所定义时，会使用哪个样式呢？一般而言，所有的样式会根据下面的规则层叠于一个新的虚拟样式表中，其中数字 4 拥有最高的优先权。

* 浏览器缺省设置
* 外部样式表
* 内部样式表（位于 <head> 标签内部）
* 内联样式（在 HTML 元素内部）

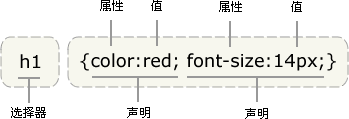
因此，内联样式（在 HTML 元素内部）拥有最高的优先权，这意味着它将优先于以下的样式声明：<head> 标签中的样式声明，外部样式表中的样式声明，或者浏览器中的样式声明（缺省值）。

* + 1. Css基本语法

CSS 规则由两个主要的部分构成：选择器，以及一条或多条声明。选择器通常是您需要改变样式的 HTML 元素。每条声明由一个属性和一个值组成。属性（property）是您希望设置的样式属性（style attribute）。每个属性有一个值。属性和值被冒号分开。

下面这行代码的作用是将 h1 元素内的文字颜色定义为红色，同时将字体大小设置为 14 像素。在这个例子中，h1 是选择器，color 和 font-size 是属性，red 和 14px 是值。

h1 {color:red; font-size:14px;}



**提示：**请使用花括号来包围声明。

CSS结构描述类似一个Key-Value的Map键值对结构。

**提示：**如果值为若干单词，则要给值加引号：

p {font-family: "sans serif";}

**多重声明：**

如果要定义不止一个声明，则需要用分号将每个声明分开。应该在每行只描述一个属性，这样可以增强样式定义的可读性，就像这样：

p {

text-align: center;

color: black;

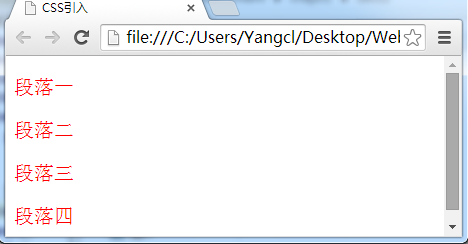
font-family: arial;

}

简单示例代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**CSS引入**</title>**
5. **<style** type="text/css"**>**
6. p{
7. **font-size**: 20px;
8. **color**: red;
9. }
10. **</style>**
11. **</head>**
12. **<body>**
13. **<p>**段落一**</p>**
14. **<p>**段落二**</p>**
15. **<p>**段落三**</p>**
16. **<p>**段落四**</p>**
17. **</body>**
18. **</html>**

页面效果截图如下：



* + 1. Css基本选择器

CSS基本选择器分为四类。其中派生选择器又分为5种使用情况，但在实际开发中【类选择器】是使用的最普遍的。

* 1 属性选择器-标记选择器

属性选择器也有人叫“标记选择器”，但在W3School中称作属性选择器，这里依从W3School的命名方法。属性选择器可以为拥有指定Html属性的 HTML 元素设置样式，而不仅限于 class 和 id 属性。

语法格式：

Html元素[该元素内联属性]{

Key: value;

Key: value;

……

Key: value;

}

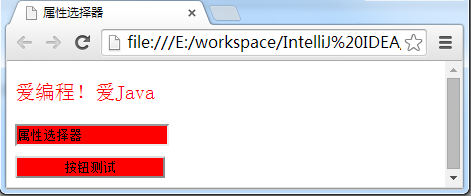
##### 大范围属性选择

以html元素作为属性的选择者，并不精确到该元素的内联属性。产生的结果就是在整个页面中只要有该html标记的元素都会被重新赋予新定义的样式。示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>属性选择器</title>
5. <style type="text/css"  >
6. /\*代表所有<p>标签全部更改样式\*/
7. p{  **// Html的属性名称作为CSS属性选择器的选择者 ！！**
8. **color**: red;
9. **font-size**: 20px;
10. }
12. /\*代表所有<input>标签全部更改样式\*/
13. input  **// Html的属性名称作为CSS属性选择器的选择者 ！！**
14. {
15. **width**:150px;
16. **display**: block;
17. **margin-bottom**:10px;
18. **background-color**: red;
19. **font-family**: Verdana, Arial;
20. }
21. </head>
22. <body>
23. <p>爱编程！爱Java</p>
24. <input type="text" name="hehe" value="属性选择器" size="20" />
25. <input type="button" name="mybtn" value="按钮测试"/>
26. </body>
27. </html>

页面中的<input />标签全部变成了直观的红色，包括文本框和按钮。

页面效果截图如下：



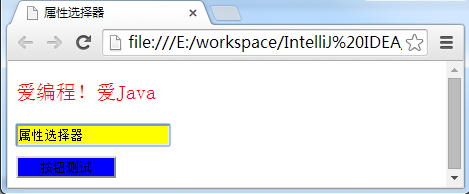
* 小范围属性选择

以html元素作为属性的选择者，同时精确到该元素的内联属性。这种方式更像是“手术刀似的操作”。产生的结果就是在整个页面中：有该html标记，同时还要含有其指定的内联属性的html标记会被重新赋予新定义的样式。示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>属性选择器</title>
5. <style type="text/css"  >
6. /\*代表所有<p>标签全部更改样式\*/
7. p{
8. **color**: red;
9. **font-size**: 20px;
10. }
11. input[type="text"]/\*不要有空格\*/
12. {
13. **width**:150px;
14. **display** : block;
15. **margin-bottom**:10px;
16. **background-color** : yellow;
17. **font-family**: Verdana, Arial;
18. }
19. input[type="button"]/\*不要有空格\*/
20. {
21. **width**:100px;
22. /\*margin-left:35px;\*/
23. **display**: block;
24. **background-color**: blue;
25. **font-family**: Verdana, Arial;
26. }
27. </style>
28. </head>
29. <body>
30. <p>爱编程！爱Java</p>
31. <input type="text" name="hehe" value="属性选择器" size="20" />
32. <input type="button" name="mybtn" value="按钮测试"/>
33. </body>
34. </html>

页面中的<input />标签通过小范围的设定，分别有了各自不同的样式：文本框有了自己的黄色背景，button按钮有了自己的蓝色背景。这里使用的是<input />标签的type属性，根据type属性的不同来进行小范围的更改。

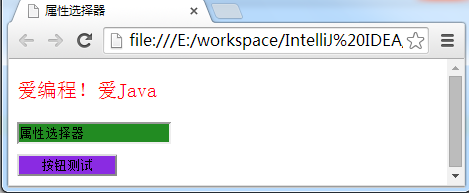
页面效果截图如下：



下面继续使用是<input />标签来展示，这次选择它的内联属性：name。源代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>属性选择器</title>
5. <style type="text/css"  >
6. /\*代表所有<p>标签全部更改样式\*/
7. p{
8. **color**: red;
9. **font-size**: 20px;
10. }
11. input**[name="hehe"]**/\*不要有空格\*/
12. {
13. **width**:150px;
14. **display**: block;
15. **margin-bottom**:10px;
16. **background-color**: forestgreen;
17. **font-family**: Verdana, Arial;
18. }
19. input**[name="mybtn"]**/\*不要有空格\*/
20. {
21. **width**:100px;
22. /\*margin-left:35px;\*/
23. **display**: block;
24. **background-color**: blueviolet;
25. **font-family**: Verdana, Arial;
26. }
27. </style>
28. </head>
29. <body>
30. <p>爱编程！爱Java</p>
31. <input type="text" name="hehe" value="属性选择器" size="20" />
32. <input type="button" name="mybtn" value="按钮测试"/>
33. </body>
34. </html>

页面效果截图如下：



* 2 类选择器-使用广泛

类选择器主要依赖html元素中的class属性。它以“.class值”的形式作为选择者主体。在CSS 中，类选择器以一个点号显示

.center {text-align: center}

在上面的例子中，所有拥有 center 类的 HTML 元素均为居中。

在下面的 HTML 代码中，h1 和 p 元素都有 center 类。这意味着两者都将遵守 ".center" 选择器中的规则。

<h1 class="center">

This heading will be center-aligned

</h1>

<p class="center">

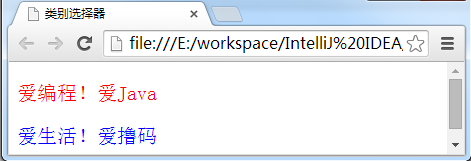
This paragraph will also be center-aligned.

</p>

示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>类别选择器</title>
5. <style type="text/css">
6. .r{
7. **color**: red;
8. **font-size**: 20px;
9. }
10. .b{
11. **color**: blue;
12. **font-size**: 20px;
13. }
14. </style>
15. </head>
16. <body>
17. <p  class="r">爱编程！爱Java</p>
18. <p  class="b">爱生活！爱撸码</p>
19. </body>
20. </html>

页面效果截图如下：



**据说类选择器使用的是最广泛的**。

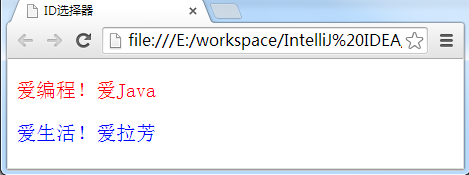
* 3 ID选择器

id 选择器可以为标有特定 id 的 HTML 元素指定特定的样式。id 选择器以 "#" 来定义。由于id这个属性在html中具有唯一性，也就是说ID选择器的可复用性很差。

示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title> ID选择器 </title>
5. <style type="text/css">
6. #p1{
7. **color**: red;
8. **font-size**: 20px;
9. }
10. #p2{
11. **color**: blue;
12. **font-size**: 20px;
13. }
14. </style>
15. </head>
16. <body>
17. <p id="p1">爱编程！爱Java</p>
18. <p id="p2">爱生活！爱拉芳</p>
19. </body>
20. </html>

页面效果截图如下：



* 4 派生选择器

通过依据元素在其位置的上下文关系来定义样式，可以使标记更加简洁。在 CSS1 中，通过这种方式来应用规则的选择器被称为上下文选择器 (contextual selectors)，这是由于它们依赖于上下文关系来应用或者避免某项规则。在 CSS2 中，它们称为派生选择器，但是无论你如何称呼它们，它们的作用都是相同的。

派生选择器允许你根据文档的上下文关系来确定某个标签的样式。通过合理地使用派生选择器，我们可以使 HTML 代码变得更加整洁。比方说，你希望列表中的 strong 元素变为斜体字，而不是通常的粗体字，可以这样定义一个派生选择器：

li strong {

font-style: italic;

font-weight: normal;

}

请注意标记为 <strong> 的蓝色代码的上下文关系

<p>

<strong>

我是粗体字，不是斜体字，因为我不在列表当中，所以这个规则对我不起作用

</strong>

</p>

<ol>

<li>

<strong>

我是斜体字。这是因为 strong 元素位于 li 元素内。

</strong>

</li>

<li>我是正常的字体。</li>

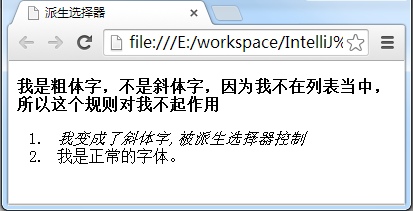
</ol>

在上面的例子中，只有 li 元素中的 strong 元素的样式为斜体字，无需为 strong 元素定义特别的 class 或 id，代码更加简洁。

示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title> 派生选择器 </title>
5. <style type="text/css">
6. **li strong**{
7. **font-style**: italic;
8. **font-weight**: normal;
9. }
10. </style>
11. </head>
12. <body>
13. <p>
14. <strong>我是粗体字，不是斜体字，因为我不在列表当中，所以这个规则对我不起作用</strong>
15. </p>
17. <ol>
18. <li>
19. <strong>我是斜体字,被派生选择器控制</strong>
20. </li>
21. <li>我是正常的字体。</li>
22. </ol>
23. </body>
24. </html>

页面效果截图如下：

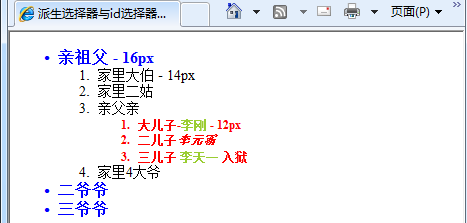


* A 派生选择器与 id选择器结合

在现代布局中，id 选择器常常用于建立派生选择器。代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>派生选择器与id选择器结合</title>
5. <style type="text/css">
6. /\*\*/
7. #root li{
8. **color**: blue;
9. **font-size**: 16px;
10. **font-weight**:bold;
11. }
13. #father li{
14. **color**: black;
15. **font-size**: 14px;
16. **font-weight**:normal;
17. }
19. #sons li {
20. **color**: red;
21. **font-size**: 12px;
22. **font-weight**:bold;
23. }
24. /\*块包含的html元素均可取\*/
25. #sons span{
26. **color**: #9acd32;
27. }
28. </style>
29. </head>
30. <body>
31. <ul id="root">
32. <li>亲祖父 - 16px
33. <ol id="father">
34. <li>家里大伯 - 14px</li>
35. <li>家里二姑</li>
36. <li>亲父亲
37. <ol id="sons">
38. <li>大儿子-<span>李刚</span> - 12px</li>
39. <li>二儿子<em>李元霸</em></li>
40. <li>三儿子 <span> 李天一</span> 入狱</li>
41. </ol>
42. </li>
43. <li>家里4大爷</li>
44. </ol>
45. </li>
46. <li>二爷爷</li>
47. <li>三爷爷</li>
48. </ul>
49. </body>
50. </html>

页面效果截图如下：



**注意：覆盖效应**。id选择器结合派生选择器使用有覆盖效应。即：当我设置【亲祖父】这一级的样式的时候，#root li{……}后面所有含有li元素和被li元素包裹的html元素都会继承他的特征。也就是说上图中的2级和3级样式都需要重新设定。

* B 派生选择器与 类选择器

派生选择器与类选择器的关系 和 派生选择器与id选择器的关系式一样的，只不过#号变成了.号，这里可以通过上例可以举一反三不再熬述。

语法格式如下：

.classname html元素{

Key :value:

Key :value:

Key :value:

}

注意：这种选择器依从**覆盖效应。**

* C 后代选择器

CSS派生选择器的一个变种，语法格式类似，但效率更加高效。又称为包含选择器。

后代选择器的功能极其强大。有了它，可以使 HTML 中不可能实现的任务成为可能。

假设有一个文档，其中有一个边栏，还有一个主区。边栏的背景为蓝色，主区的背景为白色，这两个区都包含链接列表。不能把所有链接都设置为蓝色，因为这样一来边栏中的蓝色链接都无法看到。

解决方法是使用后代选择器。在这种情况下，可以为包含边栏的 div 指定值为 sidebar 的 class 属性，并把主区的 class 属性值设置为 maincontent。然后编写以下样式：

div.sidebar {background: blue;}

div.maincontent {background: white;}

div.sidebar a:link {color:white;}

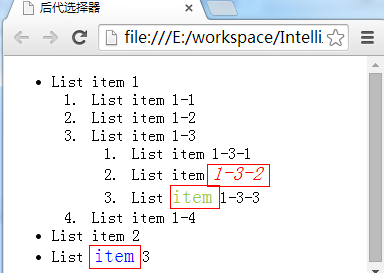
div.maincontent a:link {color:blue;}

有关后代选择器有一个易被忽视的方面，即两个元素之间的层次间隔可以是无限的。例如，如果写作 ul em，这个语法就会选择从 ul 元素继承的所有 em 元素，而不论 em 的嵌套层次多深。因此，ul em 将会选择以下标记中的所有 em 元素：多说无益，看代码：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title> 后代选择器 </title>
5. <style type="text/css" >
6. ul em {
7. **color**: red;
8. **font-size**: 20px;
9. }
10. ul ol span{
11. **color**: yellowgreen;
12. **font-size**: 20px;
13. }
14. ul span{
15. **color**: blue;
16. **font-size**: 20px;
18. }
19. </style>
20. </head>
21. <body>
22. <ul>
23. <li>List item 1
24. <ol>
25. <li>List item 1-1</li>
26. <li>List item 1-2</li>
27. <li>List item 1-3
28. <ol>
29. <li>List item 1-3-1</li>
30. <li>List item **<em>1-3-2</em>**</li>
31. <li>List **<span> item</span>** 1-3-3</li>
32. </ol>
33. </li>
34. <li>List item 1-4</li>
35. </ol>
36. </li>
37. <li>List item 2</li>
38. <li>List**<span> item</span>** 3</li>
39. </ul>
40. </body>
41. </html>

CSS的后代选择器相比较CSS的【属性选择器-小范围选择】而言是一种小范围的广谱选择。

页面效果截图如下：

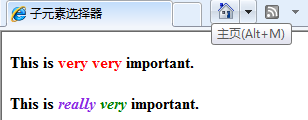


* D 子元素选择器

CSS派生选择器的一个变种，语法格式类似。与后代选择器相比，子元素选择器（Child selectors）只能选择作为某元素子元素的元素。如果你不希望选择任意的后代元素，而是希望缩小范围，只选择某个元素的子元素，请使用子元素选择器（Child selector）。

1. <html>
2. <head>
3. <meta charset="UTF-8">
5. <title> 子元素选择器 </title>
7. <style type="text/css">
8. h4 > strong {**color**:red;}
10. h4 > em {**color**: #8a2be2}
12. h4 > em >strong{**color**: green}
13. </style>
14. </head>
16. <body>
17. <h4>This is <strong>very</strong> <strong>very</strong> important.</h4>
18. <h4>This is <em>really <strong>very</strong></em> important.</h4>
19. </body>
21. </html>

页面效果截图如下：

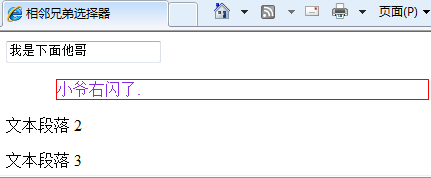


* E 相邻选择器

相邻选择器，也叫做“相邻兄弟选择器”。使用了加号（+），即相邻兄弟结合符（Adjacent sibling combinator）。注释：与子结合符一样，相邻兄弟结合符旁边可以有空白符。

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title> 相邻兄弟选择器 </title>
5. <style type="text/css">
6. input + p{
7. **border**: red 1px solid;
8. **margin-left**: 50px;
9. **color**: #8a2be2;
10. }
11. </style>
12. </head>
13. <body>
14. <input type="text" name="hehe"  value="我是下面他哥"/>
15. <p>小爷右闪了.</p>
16. <p>文本段落 2</p>
17. <p>文本段落 3</p>
18. </body>
19. </html>

页面效果截图如下：



相邻选择器在对表格的高级操作中非常实用，尤其是操作表格的不同列。更加深入的介绍请参照2.7.5<td />列样式控制。

* F 交集选择器 TODO

E 并集选择器 TODO

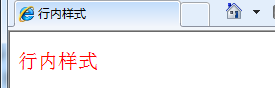
很常用的选择器，在对表格的高级操作中非常实用。更加深入的介绍请参考2.7.2 表格边框控制。

* + 1. HTML 中引入CSS的方法
* 行内样式

行内样式，即：在html元素内定义CSS样式，这种方式不推荐使用。

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>行内样式</title>
5. </head>
6. <body>
7. <p style="color: red;font-size: 20px;">行内样式</p>
8. </body>
9. </html>

页面效果截图如下：

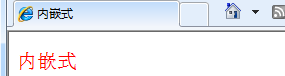


* 内嵌式

内嵌式CSS层叠样式单是比较常用的方法，但最好将CSS样式和Html文本进行分离使用CSS文件的方式来管理样式。

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>内嵌式</title>
5. <style type="text/css">
6. p{
7. **color**: red;
8. **font-size**: 20px;
9. }
10. </style>
11. </head>
12. <body>
13. <p>内嵌式</p>
14. </body>
15. </html>

页面效果截图如下：



* 链接式

页面CSS修饰最正规的方式。通过\*.css文件的方式统一管理CSS样式，在Html文档中分离出CSS，易于维护。如果一个页面中需要大量的CSS修饰通常采用这种方式。同理，在Html文档中也会出现大量的JavaScript脚本，他也可以通过js脚本的方式和Html文档分离。这种方式也可以用来对Html文档进行分块修饰，以此来提高可维护性。

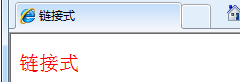
1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>链接式</title>
5. <link href="red.css" type="text/css" rel="stylesheet">
6. </head>
7. <body>
8. <p>链接式</p>
9. </body>
10. </html>

red.css的内容如下

1. p{
2. **color**: red;
3. **font-size**: 20px;
4. }

<link href="red.css" type="text/css" rel="stylesheet"> 这是一种固定的写法。

页面效果截图如下：

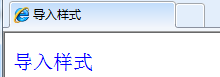


* 导入样式

导入CSS的一种方式，不常用。@import url(blue.css)

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>导入样式</title>
5. <style type="text/css">
6. @import url(blue.css);
7. </style>
8. </head>
9. <body>
10. <p>导入样式</p>
11. </body>
12. </html>
14. blue.css样式如下：
15. p{
16. **color**: blue;
17. **font-size**: 20px;
18. }

页面效果截图如下：



* 链接样式与导入样式的区别

链接的方式导入CSS文件，只有当解析到那个Html元素的时候，才会被浏览器所加载；而导入式的方法来引入CSS样式则是一次性全部将CSS定义代码全部引入到Html文档中，如果导入的CSS文件包含上千行定义，这对浏览器解析在性能上会造成很大影响，所以并不提倡使用导入的方式来引入CSS样式，因为他可能影响浏览器性能。

* 优先级比较

行内样式的优先级最高；链接式、内嵌式和导入式优先级向平，但后面的样式会覆盖前面的样式。

* 1. CSS盒模型

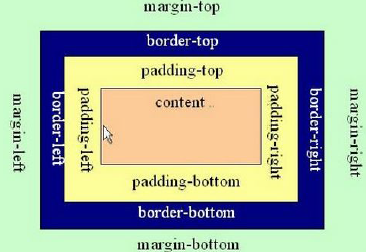


CSS对Html文档中的每一个元素的控制都是采用盒子模型的方式来完成的。在html文档中的每一个html元素(包括这个html元素所包含的内容)都被CSS视为一个盒子模型，利用这个盒子去控制每一个Html元素在浏览器中的状态。

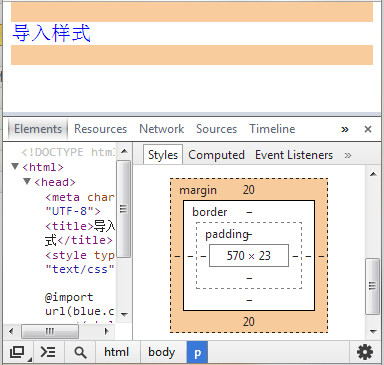
对CSS的盒模型我们可以把它理解成“墙面、画框和画”的关系。假设上图是四幅名画，他们都被放在了4个不同的相框内。

* 1. 每一幅画可以理解成content，即：一个html元素以及这个html元素所包含的内容；
  2. 每个画框中紫色的部分可以理解成为padding，即：画距离画框边距的距离；
  3. 每个黑色的长方形视作border，即：画框的属性(如颜色、宽带等)；
  4. 两个画框间的间距就是margin，即：两幅作品间的距离。

在W3Cschool中定义的盒子模型如下所示：

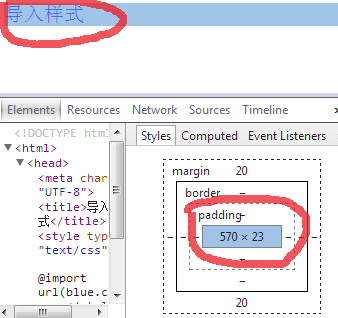


在谷歌内核浏览器中，审查文档元素时每一个html元素的盒模型体现如下图所示：



内容(content),填充(padding),边框(border),边界(margin)

当鼠标移动到盒模型的margin区域时，页面中【导入样式】这段文字的上下也会出现对应的颜色变化。移动到content区域时，变化如下：



* + 1. CSS边框 border属性

CSS边框有三个最近基本的属性，分别是：宽度(border-width)、风格(border-style)和边框的颜色(border-color)。他们有各自不同的参考值来展现html元素在浏览器中的样式外观，这三个值的描述如下：

* border-width属性参考值与描述

|  |  |
| --- | --- |
| **属性名称** | **描述** |
| medium | 默认宽度 |
| thin | 定义细的边框。 |
| thick | 定义粗的边框。 |
| length | px像素值。允许您自定义边框的宽度。 |
| inherit | 规定应该从父元素继承边框宽度。 |

* border-style 属性参考值与描述

|  |  |
| --- | --- |
| **属性参考值** | **描述** |
| none | 定义无边框。 |
| hidden | 与 "none" 相同。不过应用于表时除外，对于表，hidden 用于解决边框冲突。 |
| dotted | 定义点状边框。 在大多数浏览器中呈现为实线。 |
| dashed | 定义虚线。 在大多数浏览器中呈现为实线。 |
| solid | 定义实线。 |
| double | 定义双线。 双线的宽度等于 border-width 的值。 |
| groove | 定义3D凹槽边框。 其效果取决于 border-color 的值。 |
| ridge | 定义3D垄状边框。 其效果取决于 border-color 的值。 |
| inset | 定义3D inset边框。 其效果取决于 border-color 的值。 |
| outset | 定义3D outset边框。其效果取决于 border-color 的值。 |
| inherit | 规定应该从父元素继承边框样式。不常用。 |

* border-color属性参考值与描述

|  |  |
| --- | --- |
| **属性参考值** | **描述** |
|  |  |
| 颜色值 | 没什么可说的 |
|  |  |
|  |  |

当然在这三个核心基本值之外还有一些边框的细分属性。比如，CSS允许你单独设置整个边框的上部宽度、风格和颜色。他们分别是border-top、boreder-left、border-right和

border-bottom以及他们的衍生属性：\*-color、\*-width和\*-style(这里的\*号指代border-top等内容)。CSS 全部边框属性罗列如下：

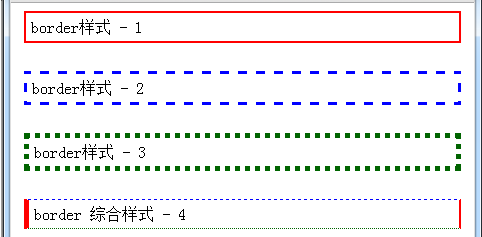
* CSS 全部边框属性

|  |  |
| --- | --- |
| **属性名称** | **描述** |
| border | 简写属性，用于把针对四个边的属性设置在一个声明。 |
| border-style | 用于设置元素所有边框的样式，或者单独地为各边设置边框样式。 |
| border-width | 简写属性，用于为元素的所有边框设置宽度，或者单独地为各边边框设置宽度。 |
| border-color | 简写属性，设置元素的所有边框中可见部分的颜色，或为4 个边分别设置颜色。 |
|  | |
| border-bottom | 简写属性，用于把下边框的所有属性设置到一个声明中。 |
| border-bottom-color | 设置元素的下边框的颜色。 |
| border-bottom-style | 设置元素的下边框的样式。 |
| border-bottom-width | 设置元素的下边框的宽度。 |
|  | |
| border-left | 简写属性，用于把左边框的所有属性设置到一个声明中。 |
| border-left-color | 设置元素的左边框的颜色。 |
| border-left-style | 设置元素的左边框的样式。 |
| border-left-width | 设置元素的左边框的宽度。 |
|  | |
| border-right | 简写属性，用于把右边框的所有属性设置到一个声明中。 |
| border-right-color | 设置元素的右边框的颜色。 |
| border-right-style | 设置元素的右边框的样式。 |
| border-right-width | 设置元素的右边框的宽度。 |
|  | |
| border-top | 简写属性，用于把上边框的所有属性设置到一个声明中。 |
| border-top-color | 设置元素的上边框的颜色。 |
| border-top-style | 设置元素的上边框的样式。 |
| border-top-width | 设置元素的上边框的宽度。 |

示例源代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>CSS边框属性：border</title>
5. <style type="text/css">
6. #borderStyle1{
7. **border-width**: 2px;
8. **border-color**: red;
9. **border-style**:solid;
10. **margin**: 5px;
11. **padding**: 5px;
12. }
14. #borderStyle2{
15. **border**: medium blue dashed;
16. **margin**: 5px;
17. **padding**: 5px;
18. }
19. #borderStyle3{
20. **border**: thick darkgreen dotted;
21. **margin**: 5px;
22. **padding**: 5px;
23. }
24. #borderStyle4{
25. **border-top**: thin blue dashed;
26. **border-right**: 2px red solid;
27. **border-bottom**: thin darkgreen dotted;
28. **border-left**: 5px red solid;
29. **margin**: 5px;
30. **padding**: 5px;
31. }
32. </style>
33. </head>
34. <body>
35. <div id="borderStyle1">border样式 - 1</div>
36. <br/>
37. <div id="borderStyle2">border样式 - 2</div>
38. <br/>
39. <div id="borderStyle3">border样式 - 3</div>
40. <br/>
41. <div id="borderStyle4">border 综合样式 - 4</div>
42. </body>
43. </html>

页面效果截图如下：



* + 1. CSS内边距padding属性

CSS padding 属性定义元素的内边距。padding 属性接受长度值或百分比值，但不允许使用负值。按照上、右、下、左的顺序分别设置各边的内边距。

padding属性可接受的参考值如下：

h1 {

padding-top: 10px;

padding-right: 0.25em;

padding-bottom: 2ex;

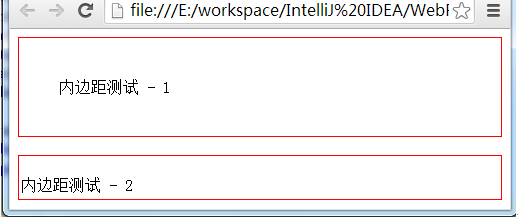
padding-left: 20%;

}

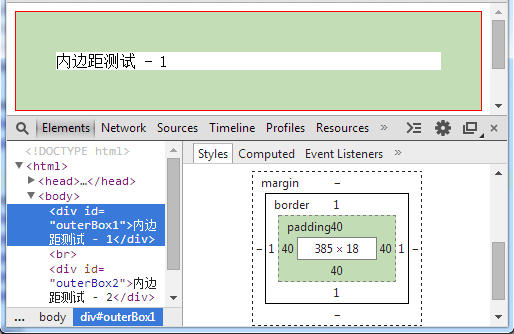
示例源代码如下：

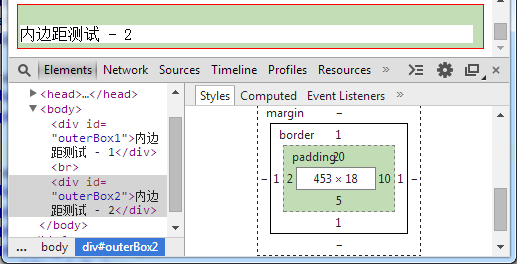
1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title> 内边距 </title>
5. <style type="text/css">
6. #outerBox1{
7. **border**: 1px red solid;
8. **padding**: 40px;
9. }
10. #outerBox2{
11. **border**: 1px red solid;
12. **padding-top**: 20px;
13. **padding-right**: 10px;
14. **padding-bottom**: 5px;
15. **padding-left**: 2px;
16. }
17. </style>
18. </head>
19. <body>
20. <div id="outerBox1">内边距测试 - 1</div>
21. <br/>
22. <div id="outerBox2">内边距测试 - 2</div>
23. </body>
24. </html>

页面效果截图如下：



这两个内边距测试在盒模型中的表现形式如下两幅图所示：





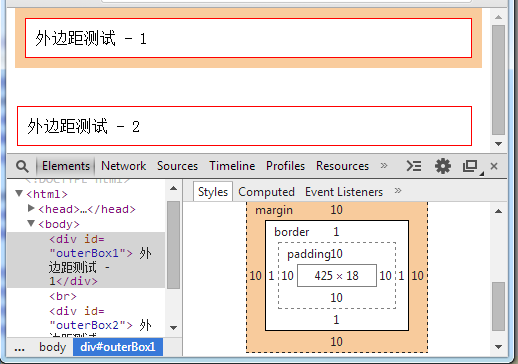
* + 1. CSS外边距margin属性

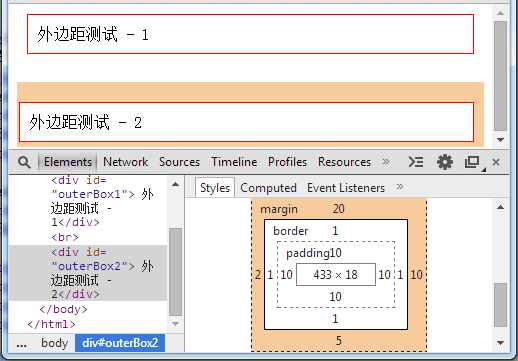
围绕在元素边框的空白区域是外边距。设置外边距会在元素外创建额外的“空白”。

设置外边距的最简单的方法就是使用 margin 属性，这个属性接受任何长度单位、百分数值甚至负值。margin的负值设置会出现叠加效果，应用比较广泛，比如相册。外边距的可接受值与padding属性相似，不在介绍。示例源代码如下：

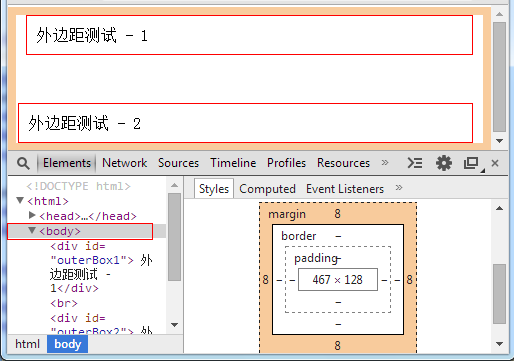
1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title> 外边距测试 </title>
5. <style type="text/css">
6. #outerBox1{
7. **border**: 1px red solid;
8. **padding**: 10px;
9. **margin**: 10px;
10. }
11. #outerBox2{
12. **border**: 1px red solid;
13. **padding**: 10px;
15. **margin-top**: 20px;
16. **margin-right**: 10px;
17. **margin-bottom**: 5px;
18. **margin-left**: 2px;
19. }
20. </style>
21. </head>
22. <body>
23. <div id="outerBox1"> 外边距测试 - 1</div>
24. <br/>
25. <div id="outerBox2"> 外边距测试 - 2</div>
26. </body>
27. </html>

这两个外边距测试在盒模型中的表现形式如下两幅图所示：





这个示例在<body />体中的体现如下：



* + 1. 盒模型在标准流中的定位

所谓html标准流：指在不使用其他与排列和定位相关的特殊CSS 规则时，各种元素的排列规则；在浏览器中从上到下一层一层的渲染，类似水流的走向。

* 行内元素与水平margin

marin-right + margin-left，非叠加。参考1.10.3。

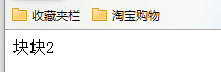
* 块级元素与竖直margin

margin-bottom margin-top 以大的为标准。大的会吞并小的margin，或者说覆盖小的。

* 盒子嵌套与margin

子块的margin 将以父块的内容(content)为参考。

* margin 属性负值效果

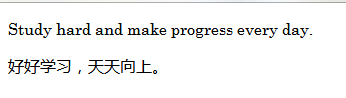


* 1. CSS文本样式控制
     1. 文字字体设置

font-family，设置字体类型名称或者针对某元素的类属族名名称的优先列表。

1. <head>
2. <meta charset="UTF-8">
3. <title> 设置文字的字体 </title>
4. <style type="text/css">
5. .p1{
6. **font-family**:  Century Schoolbook,Times New Roman
7. }
8. .p2{
9. **font-family**: "微软雅黑" , "黑体" , "宋体"
10. }
11. </style>
12. </head>
13. <body>
14. <p class="p1">Study hard and make progress every day.</p>
15. <p class="p2">好好学习，天天向上。</p>
16. </body>

页面效果截图如下：



* + 1. 文字倾斜

font-style: italic; 没什么可说的。

页面效果截图如下：

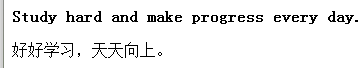


* + 1. 文字的加粗

font-weight;设置加粗效果

1. <style type="text/css">
2. p{
3. **font-weight**: bold;
4. }
5. </style>
6. </head>
7. <body>
8. <p>Study hard and make progress every day.</p>
9. <p style="font-weight: normal;">好好学习，天天向上。</p>
10. </body>

页面效果截图如下：



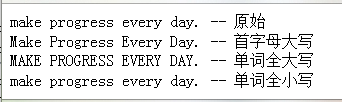
* + 1. 英文字母大小写转换

|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
| text-transform | capitalize | 首字母大写 |
| uppercase | 单词全大写 |
| lowercase | 单词全小写 |

示例源代码如下：

1. <head>
2. <meta charset="UTF-8">
3. <title>设置英文字母大小写转换</title>
4. <style type="text/css">
5. p{
6. **margin-bottom**: 1px;
7. **margin-top**: 1.5px;
8. }
9. .p1{
10. **text-transform**: capitalize;
11. }
12. .p2{
13. **text-transform**: uppercase;
14. }
15. .p3{
16. **text-transform**: lowercase;
17. }
18. </style>
19. </head>
20. <body>
21. <p class="p0">make progress every day. -- 原始</p>
22. <p class="p1">make progress every day. -- 首字母大写</p>
23. <p class="p2">make progress every day. -- 单词全大写</p>
24. <p class="p3">MAKE PROGRESS EVERY DAY. -- 单词全小写</p>
25. </body>

页面效果截图如下：



* + 1. 文字大小

font-size: 36px; px代表像素。

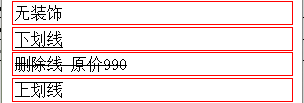
* + 1. 文字装饰效果

|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
| text-decoration | none | 无装饰 |
| underline | 下划线 |
| line-through | 删除线 |
| overline | 上划线 |

示例代码如下：

1. <head>
2. <meta charset="UTF-8">
3. <title>设置文字的装饰效果</title>
4. <style type="text/css">
5. p{
6. **margin**: 1.5px;
7. **border**: 1px red solid;
8. **padding**: 2px;
9. }
10. .p1{
11. **text-decoration**: none;
12. }
13. .p2{
14. **text-decoration**: underline;
15. }
16. .p3{
17. **text-decoration**: line-through;
18. }
19. .p4{
20. **text-decoration**: overline;
21. }
22. </style>
23. </head>
24. <body>
25. <p class="p1">无装饰</p>
26. <p class="p2">下划线</p>
27. <p class="p3">删除线 原价990</p>
28. <p class="p4">上划线</p>
29. </body>

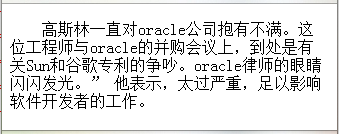
页面效果截图如下：



* + 1. 段落首行缩进

text-indent: 2em;

页面效果截图如下：



* + 1. 字词间距

word-spacing，针对单词和汉字调节间距；letter-spacing，针对字母调节间距，对汉字不起作用。示例代码如下：

1. <head>
2. <meta charset="UTF-8">
3. <title>设置字词间距</title>
4. <style type="text/css">
5. p{
6. **margin**: 4px;
7. **font-family**:  Century Schoolbook;
8. }
9. .p2{
10. **word-spacing**: 20px;
11. **letter-spacing**: 5px;
12. }
13. </style>
14. </head>
15. <body>
16. <p class="p1">make progress every day.</p>
17. <p class="p2">make progress every day.</p>
18. <p class="p2">好好学习，天天向上。</p>
19. </body>

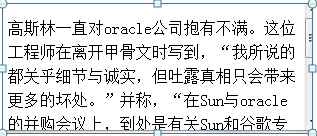
页面效果截图如下：



* + 1. 文字行高

line-height: 1.5; 不建议使用px像素表示。行高设置为1.3~1.5最为美观。

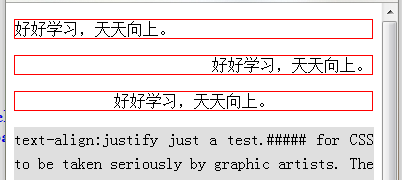
页面效果截图如下：



* + 1. 文本水平位置

|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
| text-align | left | 靠左，默认选项。 |
| right | 靠右。 |
| center | 居中。 |
| justify | 左右对齐到边，很特殊。 |

页面效果截图如下：



* + 1. 文字色和背景色

color: blue; background-color: red;

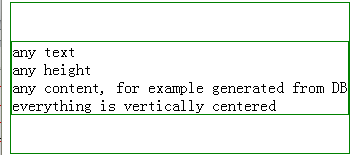
* + 1. 段落的垂直对齐 – 综合

一段文本在一个段落内居中对齐，这是一个小的综合应用。

示例源代码如下：

1. <html>
2. <head>
3. <title>Universal vertical center with CSS</title>
4. <style>
5. .greenBorder {**border**: 1px solid green;}
6. </style>
7. </head>
9. <body>
10. <div class="greenBorder" style="**display**: table;
11. **height**: 150px; #position: relative; **overflow**: hidden;">
12. <div  style=" #position: absolute; #top: 50%;
13. **display**: table-cell; **vertical-align**: middle;">
14. <div class="greenBorder" style=" #position: relative; #top: -50%">
15. any text<br>
16. any height<br>
17. any content, for example generated from DB<br>
18. everything is vertically centered
19. </div>
20. </div>
21. </div>
22. </body>
23. </html>

页面效果截图如下：



* 1. CSS图像控制
     1. 图片大小及缩放

对于一张固定尺寸的图片来讲，要控制他的大小或者说是缩放等问题，CSS有两种控制方式，第一种：使用像素控制图片；第二种：使用宽度百分比控制图片。

* 像素控制

width: 160px; height: 180px; 利用像素来控制图片的缩放，需要width和height两个属性。当浏览器大小在屏幕中进行调整的时候，使用像素控制的图片大小不会改变，依然是固定的。**缺点**是：需要控制新像素与原始像素的比例，否则图片会走形。

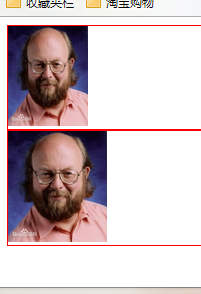
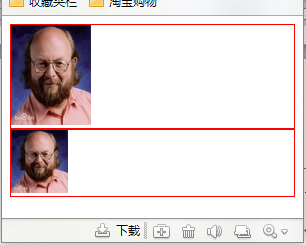
* 百分比控制

width: 50%; 使用百分比控制图片只需要一个width属性。百分比控制图片的特点是：当浏览器再屏幕中进行调整，图片大小会随着承载他的容器的宽度变化而变化。**注意**：这里的width:50%; 指的是承载他的容器、或者html块元素宽度的50%。极少数情况会遇到height来调整的情况。**优点**是：图片不用调整，不会走形。

示例源代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>设置图片大小及缩放</title>
5. <style type="text/css" >
6. .imgpx{
7. **width**: 80px;
8. **height**: 100px;
9. }
10. .img1{
11. **width**: 20%;
12. **height**: 20%;
13. }
14. </style>
15. </head>
16. <body>
17. <div style="border: 1px red solid;">
18. <img class="imgpx" src="../java.jpg" style=""/>
19. </div>
20. <div style="border: 1px red solid; ">
21. <img class="img1" src="../java.jpg" />
22. </div>
23. </body>
24. </html>

对比当浏览器调整，两幅图片的变化



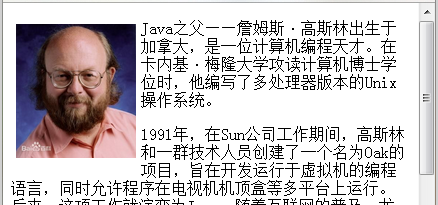
两幅图片对比显示：当浏览器大小调整时，利用百分比来控制的图片大小会出现明显变化，但主动保存了像素比例；而是用像素控制的图片，明显感觉詹姆斯.]\高斯林瘦了。

* + 1. 图文混排 – float

这只是一个简单小应用，结合了CSS提供的悬浮-float。后面会有非常详细的关于悬浮的介绍和示例。源代码如下：

1. <style type="text/css">
2. img{
3. **width**:30%;
4. **float**: left;  /\*靠左悬浮\*/
5. **margin**: 5px; /\*间隔\*/
6. }
7. </style>
8. <body>
9. <img src="../java.jpg">
10. <p>Java之父——詹姆斯•高斯林出生于加拿大，是一位计算机编程天才。 </p>
11. <p>1991年 ，Java成为全球 va之父。</p>
12. <p>高斯林一直对orac 严重，足以影响软件开发者的工作。</p>
13. </body>

页面效果如下：

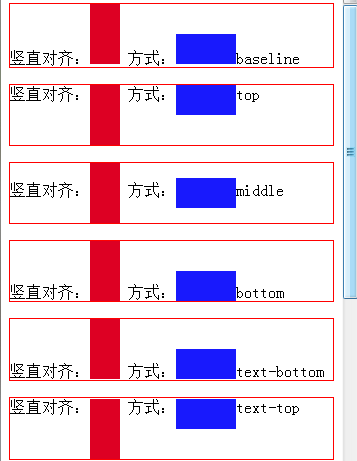


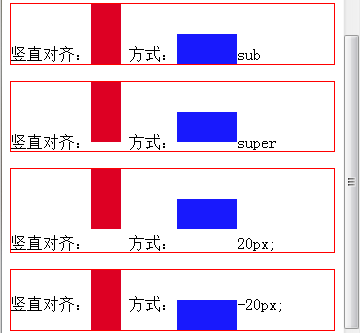
* + 1. 图片与文字的对齐方式

vertical-align 属性设置元素的垂直对齐方式。该属性定义行内元素的基线相对于该元素所在行的基线的垂直对齐。允许指定负长度值和百分比值。这会使元素降低而不是升高。在表单元格中，这个属性会设置单元格框中的单元格内容的对齐方式。

vertical-align所有取值及效果如下表：

|  |  |
| --- | --- |
| **值** | **描述** |
| baseline | 元素放置在父元素的基线上。**默认**。 |
| sub | 垂直对齐文本的下标。 |
| super | 垂直对齐文本的上标。 |
| top | 把元素的顶端与行中最高元素的顶端对齐。 |
| text-top | 把元素的顶端与父元素字体的顶端对齐。 |
| middle | 把此元素放置在父元素的中部。 |
| bottom | 把元素的顶端与行中最低的元素的顶端对齐。 |
| text-bottom | 把元素的底端与父元素字体的底端对齐。 |
| % | 使用 "line-height" 属性的百分比值来排列此元素。允许使用负值。 |
| length |  |





看效果，注意他们细小的区分。

* 1. CSS背景样式控制
     1. CSS 背景属性汇总

|  |  |  |
| --- | --- | --- |
| **属性** | **描述** | **CSS** |
| background | 简写属性，作用是将背景属性设置在一个声明中。 | 1 |
| background-color | 规定要使用的背景颜色。 | 1 |
| background-image | 规定要使用的背景图像。 | 1 |
| background-position | 规定背景图像的位置 | 1 |
| background-repeat | 规定如何重复背景图像。 | 1 |
| background-attachment | 背景图像是否固定或者随着页面的其余部分滚动。 | 1 |
| *background-size* | 规定背景图片的尺寸。 | 3 |
| *background-origin* | 规定背景图片的定位区域。 | 3 |
| *background-clip* | 规定背景的绘制区域。 | 3 |

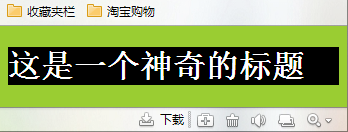
* + 1. background-color

|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
| background-color | color\_name | 规定颜色值为颜色名称的背景颜色（比如 red）。 |
| hex\_number | 规定颜色值为十六进制值的背景颜色（比如 #ff0000）。 |
| rgb\_number | 规定颜色值为 rgb 代码的背景颜色（比如 rgb(255,0,0)）。 |
| transparent | 默认。背景颜色为透明。 |

示例源代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>设置背景颜色</title>
5. <style type="text/css" >
6. h1{
7. **background-color**: black;
8. }
9. body{
10. **color**: white;
11. **background-color**: #9acd32;
12. }
13. </style>
14. </head>
15. <body >
16. <h1 >这是一个神奇的标题</h1>
17. </body>
18. </html>

页面效果如下：



* + 1. background-image

**定义和用法**

background-image 属性为元素设置背景图像。元素的背景占据了元素的全部尺寸，包括内边距和边框，但不包括外边距。默认地，背景图像位于元素的左上角，并在水平和垂直方向上重复。

提示：请设置一种可用的背景颜色，这样的话，假如背景图像不可用，页面也可获得良好的视觉效果。

**详细说明**

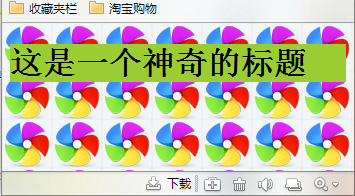
background-image 属性会在元素的背景中设置一个图像。根据 background-repeat 属性的值，图像可以无限平铺、沿着某个轴（x 轴或 y 轴）平铺，或者根本不平铺。初始背景图像（原图像）根据 background-position 属性的值放置。

|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
| background-image | url('URL') | 指向图像的路径。 |
| none | 默认值。不显示背景图像。 |

下面将以这个图片为背景做一个示例，源代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>设置背景图像</title>
5. <style type="text/css" >
6. h1{
7. **background-color**: #9acd32;
8. }
9. body{
10. **background-image**: url(../fengche.png)
11. }
12. </style>
13. </head>
14. <body>
15. <h1>这是一个神奇的标题</h1>
16. </body>
17. </html>

页面效果如下：



* + 1. background-repeat

**定义和用法**

background-repeat 属性设置是否及如何重复背景图像。默认地，背景图像在水平和垂直方向上重复。

**详细说明**

background-repeat 属性定义了图像的平铺模式。从原图像开始重复，原图像由 background-image 定义，并根据 background-position 的值放置。

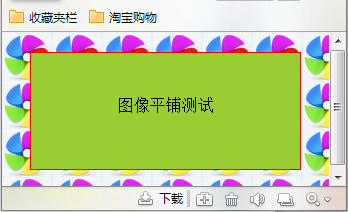
|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
| background-repeat | repeat | 默认。背景图像将在垂直方向和水平方向重复。 |
| repeat-x | 背景图像将在水平方向重复。 |
| repeat-y | 背景图像将在垂直方向重复。 |
| no-repeat | 背景图像将仅显示一次。 |

下面将以这个图片为例，结合2.5.3的内容作一个示例，源代码如下：

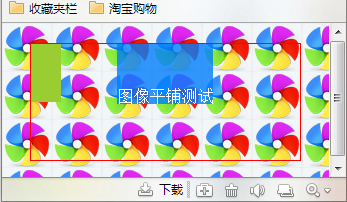
1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>设置背景图像平铺</title>
5. <style type="text/css" >
6. body{
7. **line-height**:6.5;
8. **text-align**:center;
9. **background-image**: url(../fengche.png)
10. }
11. div{
12. **margin**: 20px;
13. **height**: 116px;
14. **border**: 1px red solid;
15. **background-image**: url(../green.png);
16. **background-repeat**: repeat;
17. /\*background-repeat: no-repeat;\*/
18. /\*background-repeat: repeat-x;\*/
19. /\*background-repeat: repeat-y;\*/
20. }
21. </style>
22. </head>
23. <body >
24. <div>图像平铺测试</div>
25. </body>
26. </html>

针对这4种不同的取值，页面效果展现分别如下：

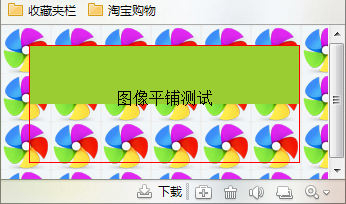
1. background-repeat: repeat;



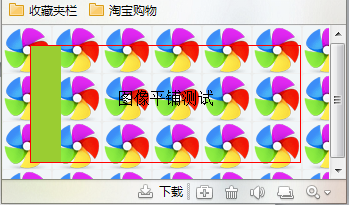
1. background-repeat: no-repeat;



1. background-repeat: repeat-x;



1. background-repeat: repeat-y;



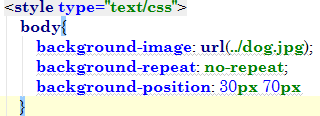
* + 1. background-position

background-position 属性设置背景图像的起始位置。这个属性设置背景原图像（由 background-image 定义）的位置，背景图像如果要重复，将从这一点开始。

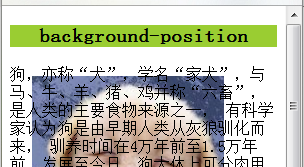
**提示**：您需要把 background-attachment 属性设置为 "fixed"，才能保证该属性在 Firefox 和 Opera 中正常工作。

|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
| background-position | top left  top center  top right  center left  center center  center right  bottom left  bottom center  bottom right | 如果您仅规定了一个关键词，那么第二个值将是"center"。  默认值：0% 0%。 |
| x% y% | 第一个值是水平位置，第二个值是垂直位置。  左上角是 0% 0%。右下角是 100% 100%。  如果您仅规定了一个值，另一个值将是 50%。 |
| xpos ypos | 第一个值是水平位置，第二个值是垂直位置。  左上角是 0 0。单位是像素 (0px 0px) 或任何其他的 CSS 单位。  如果您仅规定了一个值，另一个值将是50%。  您可以混合使用 % 和 position 值。 |

属性比较简单，省略冗余代码，精简如下：



页面效果如下：

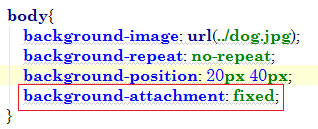


* + 1. background-attachment

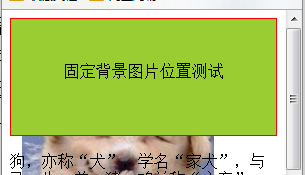
设置背景图像是否固定或者随着页面的其余部分滚动。

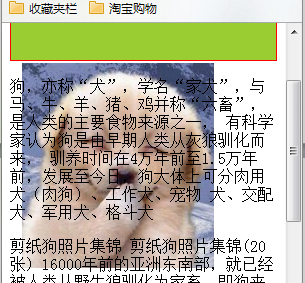
|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
| background-attachment | scroll | 默认值。背景图像会随着页面其余部分的滚动而移动。 |
|  | |
| fixed | 当页面的其余部分滚动时，背景图像不会移动。 |

属性比较简单，省略冗余代码，精简如下：



页面悬停效果对比展示如下：





* + 1. CSS3.0新增背景属性

此处暂略

|  |  |  |
| --- | --- | --- |
| **属性名称** | **值** | **效果** |
|  |  |  |
|  |  |
|  |  |
|  |  |

* 1. 定位与浮动

CSS 为定位和浮动提供了一些属性，利用这些属性，可以建立列式布局，将布局的一部分与另一部分重叠

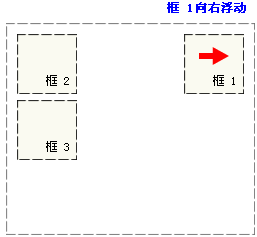
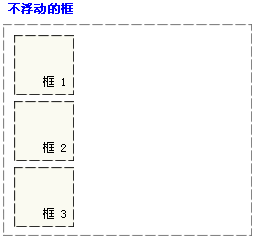
**CSS 定位机制**

CSS 有三种基本的定位机制：普通流、浮动和绝对定位。除非专门指定，否则所有框都在普通流中定位。也就是说，普通流中的元素的位置由元素在 (X)HTML 中的位置决定。块级框从上到下一个接一个地排列，框之间的垂直距离是由框的垂直外边距计算出来。行内框在一行中水平布置。可以使用水平内边距、边框和外边距调整它们的间距。但是，垂直内边距、边框和外边距不影响行内框的高度。由一行形成的水平框称为行框（Line Box），行框的高度总是足以容纳它包含的所有行内框。不过，设置行高可以增加这个框的高度。

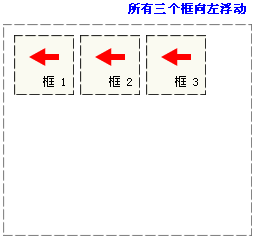
* + 1. 浮动与Html盒模型

浮动的框可以向左或向右移动，直到它的外边缘碰到包含框或另一个浮动框的边框为止。

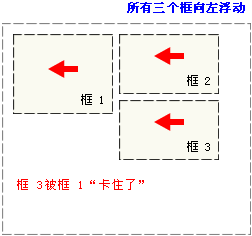
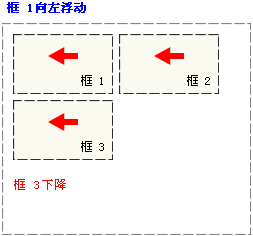
请看下图，当把框 1 向右浮动时，它脱离文档流并且向右移动，直到它的右边缘碰到包含框的右边缘：



再请看下图，当框 1 向左浮动时，它脱离文档流并且向左移动，直到它的左边缘碰到包含框的左边缘。因为它不再处于文档流中，所以它不占据空间，实际上覆盖住了框 2，使框 2 从视图中消失。如果把所有三个框都向左移动，那么框 1 向左浮动直到碰到包含框，另外两个框向左浮动直到碰到前一个浮动框。



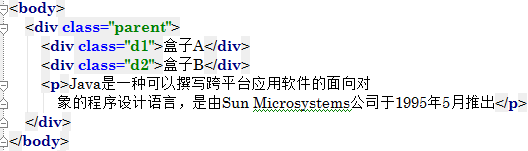
如下图所示，如果包含框太窄，无法容纳水平排列的三个浮动元素，那么其它浮动块向下移动，直到有足够的空间。如果浮动元素的高度不同，那么当它们向下移动时可能被其它浮动元素“卡住”。



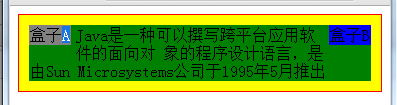
CSS 中，我们通过 float 属性实现元素的浮动。

**float 属性定义元素在哪个方向浮动**。定义了这个Html元素是在他这一行的左边浮动还是向右边浮动；与此同时，下一个html元素会自动与这个浮动的元素共同组成一个块级框。依此类推，如果有3个、4个……n个，如果他们前n-1个元素都有浮动属性，那么这n个元素会组成一个块级框。

举一个简单的例子：

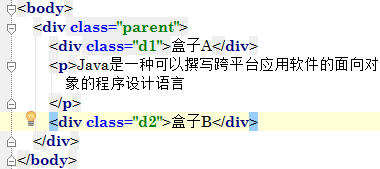


盒子A，float:left; 盒子B，float:right;



以往这个属性总应用于图像，使文本围绕在图像周围，不过在 CSS 中，任何元素都可以浮动。**浮动元素会生成一个块级框**，而不论它本身是何种元素。如果浮动非替换元素，则要指定一个明确的宽度；否则，它们会尽可能地窄。

浮动元素会生成一个块级框，那么当两个浮动的块级框相遇会出现下面的现象。举例如下：



如上代码所示，当盒子A与P元素产生浮动，盒子B没有浮动属性时，如右图所示他们三个在parent盒模型中和谐相处；但当盒子B设置浮动属性时(不论left 还是right)，盒子B被parent盒模型派出了，因为盒子B此时显示为灰色。

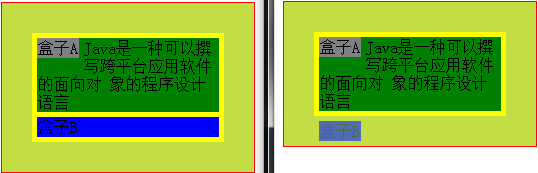


图2.6.1 - 1

示例源代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>盒子浮动float  </title>
5. <style type="text/css">
6. .parent{
7. **padding**:30px;
8. **background-color**: yellow;
9. **border**: 1px solid red;
10. }
11. .d1{
12. **float**:left;
13. **margin**:5px;
14. **background-color**: gray;
15. **border**: 1px solid gray;
16. }
17. .d2{
18. **float**:; right
19. **margin**:5px;
20. **background-color**: blue;
21. **border**: 1px solid blue;
22. }
23. p{
24. **float**:none;
25. **margin**:5px;
26. **background-color**: green;
27. **border**: 1px solid green;
28. }
29. </style>
30. </head>
31. <body>
32. <div class="parent">
33. <div class="d1">盒子A</div>
34. <p>Java是一种可以撰写跨平台应用软件的面向对
35. </p>
36. <div class="d2">盒子B</div>
37. </div>
38. </body>
39. </html>

依据图2.6.1 – 1，及上面的代码，下面进行更深入的发掘。上面的代码正常应该显示如下样式：(这里适当调节了浏览器窗口大小)



图2.6.1 – 2 – A



图2.6.1 – 2 – B

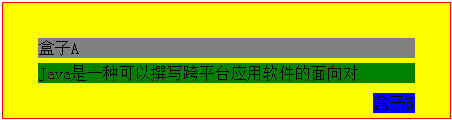


图2.6.1 – 2 – C

图2.6.1 – 2 – B设置了<p>标签的左悬浮，图2.6.1 – 2 – C取消了float属性。B图和C图的“盒子A”float属性为none，让他接受标准文档流的控制。

现在对<p>设置左悬浮float:left; 让这三个html元素同时出现悬浮，如下图的样式：



图2.6.1 - 3

经过截图测量，显示出上图2.6.1 - 3黄色区域的高度正好是60px。也就是说：在一个html文档流中的某一个块级元素内，只要有一个html元素没有利用float属性来脱离html标准文档流对他的控制，那么他的块级元素盒模型基本属性都不会丢失。

**结论：float生效的元素会脱离标准文档流控制**。

浮动元素会生成一个块级框，从而脱离标准流对他的控制，但前提是这个浮动元素的前面和后面没有标准文档流元素了。从图2.6.1 – 3可以看出三个元素都已经脱离了标准文档流的控制。只剩下class属性为parent的外围div框体，他的默认padding是30px，黄色区域的高度正好是60px；从这里也侧面反映出脱离控制。

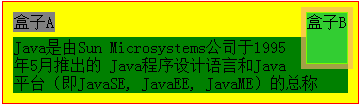
还应**注意**，有些html元素默认有margin值，比如：<p>，margin默认为上下16。在利用浏览器盒子模型调试的时候要注意。

* + 1. 浮动清除

**clear 属性定义了元素的哪边上不允许出现浮动元素**。在 CSS1 和 CSS2 中，这是通过自动为清除元素（即设置了 clear 属性的元素）增加上外边距实现的。在 CSS2.1 中，会在元素上外边距之上增加清除空间，而外边距本身并不改变。不论哪一种改变，最终结果都一样，如果声明为左边或右边清除，会使元素的上外边框边界刚好在该边上浮动元素的下外边距边界之下。

|  |  |
| --- | --- |
| **值** | **描述** |
| left | 在左侧不允许浮动元素。 |
| right | 在右侧不允许浮动元素。 |
| both | 在左右两侧均不允许浮动元素。 |
|  | |
| **JavaScript 语法** | object.style.clear="left" |

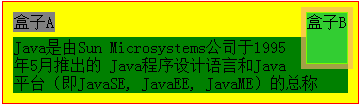
针对浮动属性，在使用的时候重点注意的依然是html元素的margin属性对层叠显示效果的影响。下面举一个例子说明，为了凸显clear:left的特征，代码中特意将盒子B的高度增加，并且图中给出了盒模型下显示出盒子B的效果及产生的影响。



源代码如下所示：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>使用clear清除浮动的影响 left</title>
5. <style type="text/css">
6. .parent{
7. **padding**:5px;
8. **background-color**: yellow;
9. **border**: 1px solid red;
10. }
11. .d1{
12. **float**:left;
13. **margin**:5px;
14. **background-color**: gray;
15. **border**: 1px solid gray;
16. }
18. .d2{
19. **float:right;**
20. **margin:5px;**
21. **height:50px;**   **// 定义出div的高度，凸显盒子B**
22. **background-color**: limegreen;
23. **border**: 1px solid lawngreen;
24. }
25. p{
26. **clear**:left;   **// 让P的内容左侧不允许有悬浮**
27. **margin**:5px;
28. **background-color**: green;
29. **border**: 1px solid green;
30. }
31. </style>
32. </head>
33. <body>
34. <div class="parent">
35. <div class="d1">盒子A</div>
36. <div class="d2">盒子B</div>
37. <p>Java是由Sun Microsystems公司于1995年5月推出的
38. Java程序设计语言和Java平台（即JavaSE, JavaEE, JavaME）的总称</p>
39. </div>
40. </body>
41. </html>
    * 1. 定位(position)与Html盒模型

一切都是“框”。



**定位的基本思想**：“框”相对于标准流中的标准位置，进行的上下移动和前后展现(z-index空间位置)。

**定位的基本依据**：父元素、另一个元素 和 浏览器窗口本身。

* CSS 相对定位 - relative

相对于原本的标准位置偏移指定的距离。标准位置：<body />本身或者包含该元素的父元素，如<div />、<table />；下面代码用的<div />。

设置为相对定位的元素框会偏移某个距离。元素仍然保持其未定位前的形状，它原本所占的空间仍保留。通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。

如果将 top 设置为 20px，那么框将在原位置顶部下面 20 像素的地方。如果 left 设置为 30 像素，那么会在元素左边创建 30 像素的空间，也就是将元素向右移动。

#box\_relative {

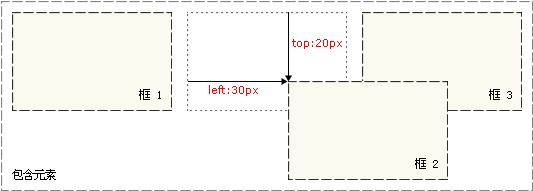
position: relative;

left: 30px;

top: 20px;

}

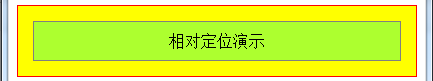
如下图所示：(注意，框2部分覆盖了框3。)



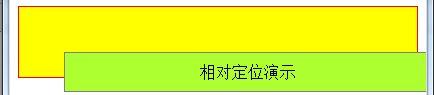
示例代码2.6.3 - 1如下：

1. <html>
2. <head>
3. <title>盒子定位position relative</title>
4. <style type="text/css">
5. .parent{
6. **padding**:15px;
7. **background-color**: yellow;
8. **border**: 1px solid red;
9. }
11. .son{
12. **position:relative;**
13. **left:30px;**
14. **top:30px;**
15. **padding**:10px;
16. **background-color**: greenyellow;
17. **border**: 1px solid gray;
18. **text-align**: center;
19. }
20. </style>
21. </head>
22. <body>
23. <div class="parent">
24. <div class="son">相对定位演示</div>
25. </div>
26. </body>
27. </html>

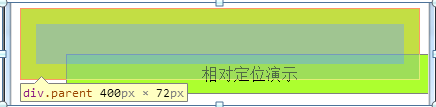
如上代码，如果没有position属性的影响，页面正常显示如下：



加入position属性后，页面显示如下：



在CSS盒模型中的真实显示：



从CSS盒模型中可以看出在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。相对定位实际上被看作普通流定位模型的一部分，因为元素的位置相对于它在普通流中的位置。

* CSS 绝对定位 - absolute

**矢量控制属性**：与position属性搭配使用的属性，包括left / top / right / bottom。

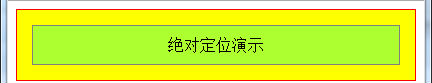
示例代码2.6.3 – 2

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title> 绝对定位 </title>
5. <style type="text/css">
6. .parent{
7. /\* position:relative;  \*/
8. **padding**:15px;
9. **background-color**: yellow;
10. **border**: 1px solid red;
11. }
12. .son{
13. **position:absolute;**
14. **left:0px;**
15. **top:0px;**
16. **padding**:10px;
17. **background-color**: greenyellow;
18. **border**: 1px solid gray;
19. **text-align**: center;
20. }
21. </style>
22. </head>
23. <body>
24. <div class="parent">
25. <div class="son">绝对定位演示</div>
26. </div>
27. </body>

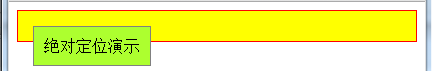
* 特性一 脱离文档流

设置为绝对定位的元素框会从文档流中完全删除，元素原先在正常文档流中所占的空间会关闭，就好像该元素原来不存在一样。元素定位后生成一个块级框，而不论原来它在正常流中生成何种类型的框。

以示 示例代码2.6.3 – 2 为例，注意标记红色的内容。没有position属性的影响，页面正常显示如下：



当son的position设置为absolute，但**不添加**矢量控制属性时(即：去掉代码中的left和top)，页面显示如下：



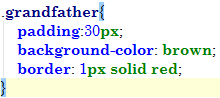
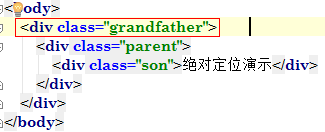
可以看出son脱离了parent对他的束缚，就是说绿色区域脱离了html标准文档流的控制，元素原先在正常文档流中所占的空间已经关闭，定位后生成一个块级框。

同时注意到：在不添加矢量控制属性时，绿色区域的**默认定位是存在的**。

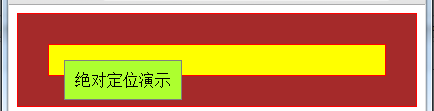
至此，展示了“父包含框”无position属性定义、子框为绝对定位但不包含矢量控制的情况和这种情况的页面效果。

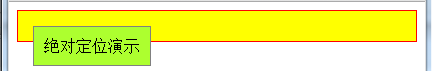
* 特性二 无矢量控制的绝对定位

在示例代码2.6.3 – 2中，为class=parent的div添加一个外框，叫grandfather同时为他加入背景和边框，代码其他内容不变。代码更改如下两幅图所示：



此时刷新页面并对比特性一中的图，显示如下：



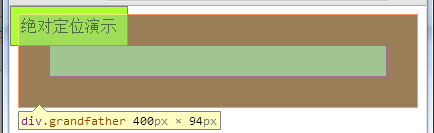


在示例代码2.6.3 – 2中padding设置为15px，此时页面中绿色区域的左上角正好默认定位在距离黄色区域上面15px、左边15px的位置。也就是说，如果不添加矢量控制属性，那么 **绝对定位** 默认会在他“父级包含框”盒模型content的最左边。

* 特性三 定位起点依据

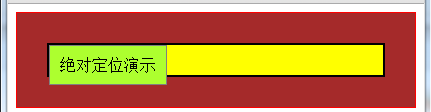
在 特性二 代码的基础上，下面添加矢量控制属性left:0px; top:0px；(**注意**：此时parent的position属性依然是被代码注释的无效状态，他不起作用)页面显示结果 和 浏览器盒模型调试时页面的显示结果如下2幅图：





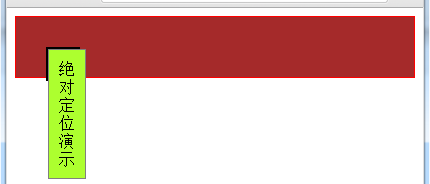
可见son的祖先“包含块”都没有position属性生效时，Html文档的初始包含块<body />默认会作为son定位的起始依据。

当对代码中 .parent的 /\*position:relative;\*/ 解开注释后(为了方便观察，parent的border改成了 2px solid #000000;)，页面效果显示如下：



**可见结论如下**(前提：position属性有4个矢量控制属性中的至少一个配合使用)：

当一个Html元素使用绝对定位的时候，它定位的依据是：离它最近的那个含有position属性的包含框(如div等等)，这个Html元素将使用这个包含框作为定位依据，而这个包含框的取值可以是：relative / absolute / fixed。relative显示效果如上图，absolute / fixed的显示效果如下图：(注意这两幅图的黑色边框。)



**提示：**因为绝对定位的框与文档流无关，所以它们可以覆盖页面上的其它元素。可以通过设置 z-index 属性来控制这些框的堆放次序。

* CSS 固定定位 - fixed

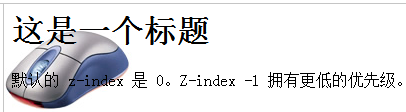
固定定位与绝对定位的特性一、特性二是一样的，不同点在于特性三，他们**定位起点**有明显的区别，在固定定位中不论他的“祖先包含框”有没有position声明，固定定位的起点都是Html文档的初始包含块<body />。



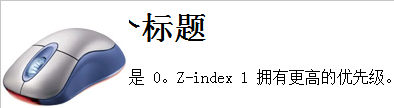
* + 1. z-index空间位置

z-index 属性设置元素的堆叠顺序。拥有更高堆叠顺序的元素总是会处于堆叠顺序较低的元素的前面。元素可拥有负的 z-index 属性值。Z-index 仅能在定位元素上奏效（例如 position:absolute;）！

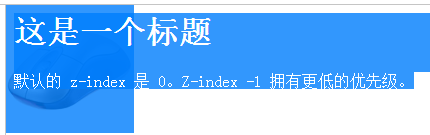
该属性设置一个定位元素沿 z 轴的位置，z 轴定义为垂直延伸到显示区的轴。如果为正数，则离用户更近，为负数则表示离用户更远。



图片z-index = -1



文字z-index = -1



元素堆叠效果

* + 1. 行内元素 与 块级元素转换

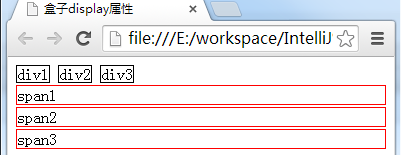
行内元素 与 块级元素之间的相互转换可以通过display属性来完成，

|  |  |  |
| --- | --- | --- |
| **属性名称** | **取值** | **效果** |
| display | inline | 块级元素变为行内元素。 |
| block | 行内元素变为块级元素。 |

示例源代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>盒子display属性</title>
5. <style type="text/css">
6. div{
7. **border**: 1px solid #000000;
8. }
9. span{
10. **margin-top**: 2px;
11. **border**: 1px solid red;
12. }
13. </style>
14. </head>
15. <body>
16. <div  style="**display: inline;**">div1</div>
17. <div  style="**display: inline;**">div2</div>
18. <div  style="**display: inline;**">div3</div>
20. <span style="**display: block;**">span1</span>
21. <span style="**display: block;**">span2</span>
22. <span style="**display: block;**">span3</span>
23. </body>
24. </html>

页面效果显示如下：



* + 1. clip 属性剪裁绝对定位元素

"clip" 属性允许您规定一个元素的可见尺寸，这样此元素就会被修剪并显示为这个形状。这个属性用于定义一个剪裁矩形。对于一个绝对定义元素，在这个矩形内的内容才可见。出了这个剪裁区域的内容会根据 overflow 的值来处理。剪裁区域可能比元素的内容区大，也可能比内容区小。

JavaScript 语法： object.style.clip="rect(0px,50px,50px,0px)"

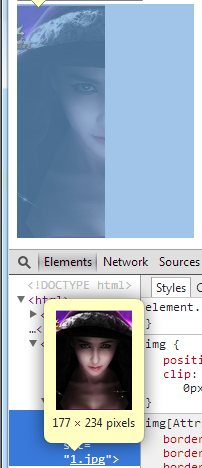
示例源代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <style type="text/css">
5. img
6. {
7. **position**:absolute;
8. **clip**:rect(0px 88px 234px 0px)
9. }
10. </style>
11. </head>
12. <body>
13. <p>clip 属性剪切了一幅图像：</p>
14. <p>
15. <img border="0" src="1.jpg" >
16. </p>
17. </body>
18. </html>

页面效果截图如下：



在盒模型中展示效果如下：



使用clip属性时，首先这个元素被进行了绝对定位；其次clip:rect(上 右 下 左)。

* 1. 表格高级操作

在html 基础中谈及了谈及了<table />这个非常常用的html标签的基本结构，这一部分将针对表格的高级特性进行详细分析。

首先列举在实际开发中，“表格”涉及到的所有Html相关标签与CSS修饰属性。这次不仅简单的介绍一个<table />标签，而是<table />辐射出的所有内容。

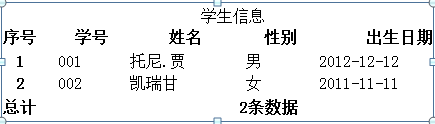
* + 1. <table />辐射

在Html页面中，表格所涉及到的所有Html元素和CSS控制属性全部罗列如下。

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **作用** | **特性** | **所属** |
| <table /> | 声明表格 |  | html |
| <tr /> | 定义一行 |  | html |
| <td /> | 定义一列 |  | html |
| <th /> | 表头单元格 | 覆盖tr | html |
| <caption /> | 标题 |  | html |
| <thead /> | 表格表头 |  | html |
| <tbody /> | 表格主体 |  | html |
| <tfoot /> | 表格页脚 |  | html |
|  | | | |
| border-spacing | 相邻单元格边框之间的距离 | 受border-collapse影响 | CSS |
| border-collapse | 是否合并表格边框 | 忽略border-spacing属性 | CSS |
| width | 规定table宽度 |  | CSS |
| table-layout | 表格布局 | auto/fixed | CSS |
| 相邻选择器 | 控制列CSS效果 | th+td+td等 | CSS |
| hover | 行变色效果 | 鼠标悬停处，行换色 | CSS |
| colspan | td/th所占列数 |  | CSS |

simple.html的示例源代码用于展示一个最基础的html元素表格，这段代码不包含任何CSS控制内容。2.7.2至2.7.6将全部在simple.html的基础上进行更改。

simple.html页面效果如下：

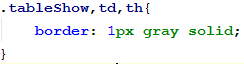


simple.html源代码如下所示：

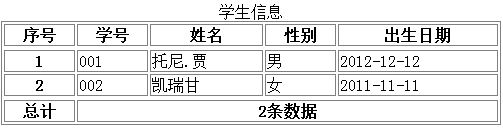
1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**html表格标签**</title>**
5. **</head>**
6. **<body>**
7. **<table** class="tableShow" style="width: 500px;"**>**
8. **<caption>**学生信息**</caption>**
9. **<thead>**
10. **<tr>**
11. **<th>**序号**</th>**
12. **<th>**学号**</th>**
13. **<th>**姓名**</th>**
14. **<th>**性别**</th>**
15. **<th>**出生日期**</th>**
16. **</tr>**
17. **</thead>**
18. **<tbody>**
19. **<tr>**
20. **<th>**1**</th>**
21. **<td>**001**</td>**
22. **<td>**托尼.贾**</td>**
23. **<td>**男**</td>**
24. **<td>**2012-12-12**</td>**
25. **</tr>**
26. **<tr>**
27. **<th>**2**</th>**
28. **<td>**002**</td>**
29. **<td>**凯瑞甘**</td>**
30. **<td>**女**</td>**
31. **<td>**2011-11-11**</td>**
32. **</tr>**
33. **</tbody>**
34. **<tfoot>**
35. **<tr>**
36. **<th>**总计**</th>**
37. **<th** colspan="4"**>**2条数据**</th>**
38. **</tr>**
39. **</tfoot>**
40. **</table>**
41. **</body>**
42. **</html>**
    * 1. 表格边框控制

**第一步**：为simple.html中的基础表格添加出边框，这里使用**并集选择器**。

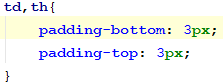
加入脚本如下：



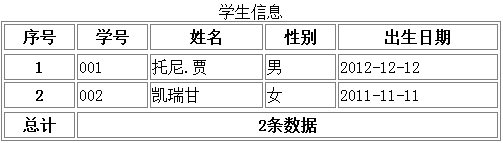
页面效果显示如下：



**第二步**：为td、th填充内边距。加入脚本如下：



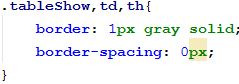
页面效果显示如下：



第三步：border-spacing属性与border-collapse属性。

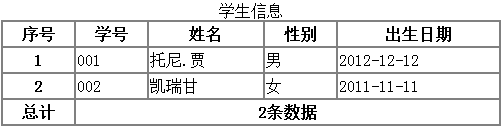
border-spacing 属性设置相邻单元格的边框间的距离（仅用于“边框分离”模式）。该属性指定分隔边框模型中单元格边界之间的距离。其取值可以是px、em等，但不允许有负值出现。

如下代码所示，将会合并相邻的两个单元格，边界会变成2px。



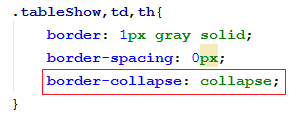
对于表格，在实际开发中大量使用了并集选择器。

页面效果显示如下：



border-spacing属性会受到border-collapse属性的影响，如果border-collapse的取值为：collapse(合并)。

border-collapse属性设置表格的边框是否被合并为一个单一的边框。默认值是separate(分离)，代表不。如果border-collapse的取值为：collapse则会让border-spacing属性失效。示例代码如下：



页面效果显示如下：

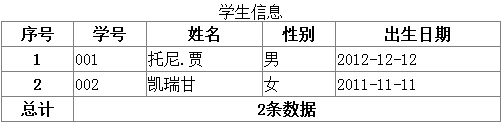


图2.7.2 - 1

* + 1. 表格布局与宽度

控制表格的布局使用table-layout属性，这个属性的默认值是auto即自适应行列宽度，如果不显示的声明table-layout的值为fixed，那么浏览器会默认执行auto。table-layout属性的默认执行效果如图2.7.2 – 1所示。现在在2.7.2基础上加入代码如下：

设置为fixed布局对比图2.7.2-1如下：

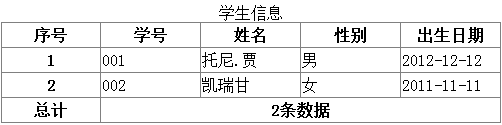
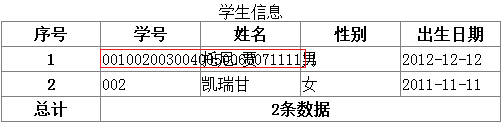


图2.7.3 - 1

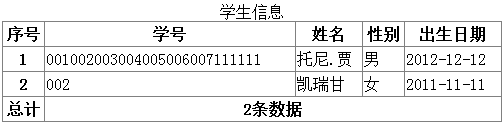
fixed布局对比auto，fixed让列变成等宽的。

在simple.html源代码中设置了table的宽度是固定的：width:500px。通过CSS的width属性可以控制表格的固定宽度。

如果表格被设置为固定宽度后，再对table-layout属性设置为fixed，那么需要注意的地方是“文字内容超过列宽的情况”，截图示例如下：



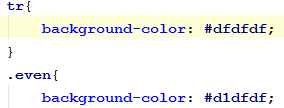
但是在这种情况下，如果将table-layout属性设置为auto，则会自动撑满这一列，如下：



**特别注意**：如果要使用表格的table-layout属性，那么必须指定width属性，否则table-layout属性的fixed是不起作用的。也就是说要先指定表格的固定宽度，如width:500px，然后table-layout:fixed;这句话才会起作用。

* + 1. 表格斑马线（隔行换色）

在2.7.3代码基础上，添加2行数据。将序号为偶数的行设置为：<tr class="even">。然后添加CSS代码控制如下：



首先为所有行赋背景值，然后在为偶数行单独赋值；页面效果显示如下：

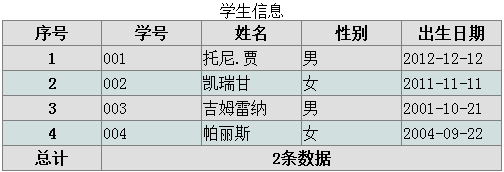
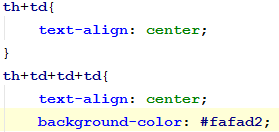


图2.7.4 - 1

* + 1. <td />列样式控制

控制表格的列样式最常用的方法是：相邻选择器。例如，在图2.7.4 – 1的基础上我想要对学号列和性别列进行设置，比如剧中、背景色等等，那么相邻选择器是最方便的。

添加代码如下：



页面效果如下：

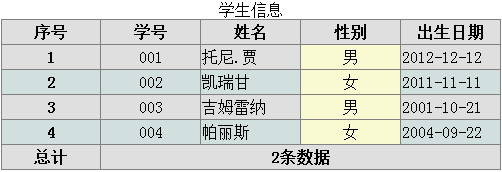


图2.7.5 – 1

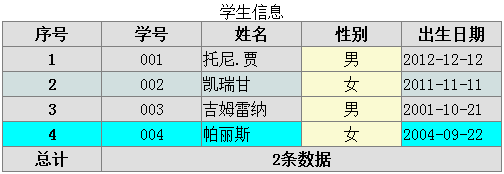
* + 1. 行变色

在图2.7.5 – 1的基础上，有时候需要实现这样一个效果：鼠标停在表格上的某一行，然后这行需要变一个颜色来凸显这条数据。这种效果需要用到CSS的:HOVER，这是一个CSS伪类，后面会专门介绍。参见2.9节CSS伪类。

添加代码如下：



指定tbody下的所有行，当鼠标悬浮在<tr />元素上方时改变整行颜色。



* 1. 列表高级操作

CSS 列表属性允许你放置、改变列表项标志，或者将图像作为列表项标志。列表分为有序列表 (<ol>) 和无序列表 (<ul>)，但他们都依从CSS列表属性的控制，CSS的列表控制属性及其相对应的功能描述如下：

|  |  |
| --- | --- |
| **属性** | **描述** |
| list-style | 简写属性。用于把所有用于列表的属性设置于一个声明中。 |
| list-style-image | 将图象设置为列表项标志。 |
| list-style-position | 设置列表中列表项标志的位置。 |
| list-style-type | 设置列表项标志的类型。 |

* + 1. 默认列表样式

有序列表 (<ol>) 和无序列表 (<ul>)的默认样式源代码如下，后面小节会对这段源代码添加CSS修饰。示例源代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**默认列表符号**</title>**
5. **<style** type="text/css"**>**
6. div{
7. position: absolute;
8. left: 140px;
9. top:0px;
10. width: 200px;
11. }
12. **</style>**
13. **</head>**
14. **<body>**
15. **<p>**无序列表**</p>**
16. **<ul>**
17. **<li>**主页**</li>**
18. **<li>**我的博客**</li>**
19. **<li** class="photo"**>**我的相册**</li>**
20. **<li>**留言**</li>**
21. **<li>**关于我**</li>**
22. **</ul>**
23. **<div>**
24. **<p>**有序列表**</p>**
25. **<ol** **>**
26. **<li>**主页**</li>**
27. **<li** **>**我的博客**</li>**
28. **<li** class="photo"**>**我的相册**</li>**
29. **<li>**留言**</li>**
30. **<li>**关于我**</li>**
31. **</ol>**
32. **</div>**
33. **</body>**
34. **</html>**

有序列表 (<ol>) 和无序列表 (<ul>)的默认样式截图如下：

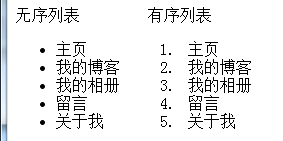


图2.8 – 1

* + 1. 列表项标志

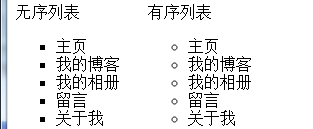
如图2.8 – 1所示，有序列表 (<ol>)的列表项标志默认样式为一个实心的黑点，无序列表 (<ul>)的列表项标志默认样式为具体的阿拉伯数字。在实际开发中列表项的样式是可以自定义的，有很多中选择，其中包括可以使用自定义的图片，如2.8.3节介绍。如果不采用图片，而是采用CSS自己提供的一些样式，可以使用**list-style-type**属性来定义。

**list-style-type**属性包含了很多可取值的内容，具体请查阅文档。常见的有：circle(空心圆)、square(实心方块)、upper-alpha(大写英文字母T)和lower-alpha(小写英文字母)等等。如下代码列举了“实心方块”和“空心圆”的情况。





页面效果如下：



* + 1. 列表图片项标志

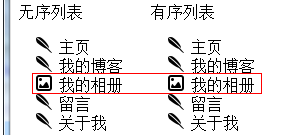
很多时候为了特殊的美观效果，并不使用自带的列表项标志，而是用自定义的内容来美化网页。这时候就可以使用**list-style-image**属性来引入自定义的图片，但这里对图片有要求，一般都是16px \* 16px的尺寸。

添加如下CSS修饰：





页面效果如下：

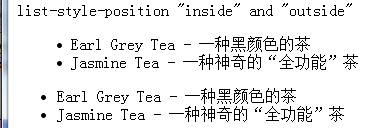


* + 1. list-style-position

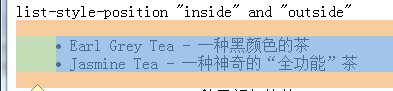
**list-style-position**属性是指将“列表项标志”放在内容区内还是内容区外。示例代码如下：

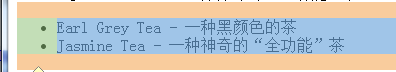
1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <style type="text/css">
5. ul.inside
6. {
7. **list-style-position**: inside
8. }
9. ul.outside
10. {
11. **list-style-position**: outside
12. }
13. </style>
14. </head>
15. <body>
16. <p>list-style-position "inside"  and  "outside"</p>
17. <ul class="inside">
18. <li>Earl Grey Tea - 一种黑颜色的茶</li>
19. <li>Jasmine Tea - 一种神奇的“全功能”茶</li>
20. </ul>
21. <ul class="outside">
22. <li>Earl Grey Tea - 一种黑颜色的茶</li>
23. <li>Jasmine Tea - 一种神奇的“全功能”茶</li>
24. </ul>
25. </body>
26. </html>

页面效果显示如下：



下面对比他们在CSS盒模型中的展现：





从CSS盒模型的对比展现中可以看出“列表项标志”在内容区内和内容区外的不同。

* + 1. 列表与导航菜单

列表的重要应用之一是创建导航菜单，导航菜单有横着的有竖着的，创建一个简单的导航菜单，不加入任何修饰，其样式如下：



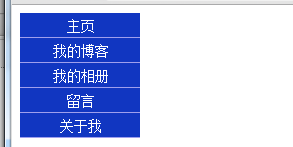
示例代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**创建简单导航菜单**</title>**
5. **</head>**
6. **<body>**
7. **<div** id="navigation"**>**
8. **<ul>**
9. **<li><a** href="#"**>**主页**</a></li>**
10. **<li><a** href="#"**>**我的博客**</a></li>**
11. **<li><a** href="#"**>**我的相册**</a></li>**
12. **<li><a** href="#"**>**留言**</a></li>**
13. **<li><a** href="#"**>**关于我**</a></li>**
14. **</ul>**
15. **</div>**
16. **</body>**
17. **</html>**

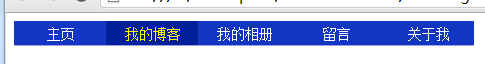
在次代码的基础上，现在用CSS进行进一步修饰，其脚本代码如下所示：

1. <style type="text/css">
2. #navigation {
3. /\* 删减  标题效果需注释此处\*/
4. **width**:120px;
6. **font-family**:Arial;
7. **font-size**:14px;
8. **text-align**:center;    /\* right改为center \*/
9. }
11. #navigation ul {
12. **list-style-type**:none;    /\* 不显示项目符号 \*/
13. **margin**:0px;
14. **padding**:0px;
15. }
16. #navigation li {
17. **border-bottom**:1px solid #9F9FED;    /\* 添加下划线 \*/
19. /\* 添加 标题效果需添加此处 \*/
20. /\*float: left;\*/
21. }
22. #navigation li a{
24. /\* 添加 标题效果需添加此处\*/
25. /\*width:120px;\*/
26. **display**:block;
27. **height**:1em;
28. **padding**:5px 5px 5px 0.5em;
29. **text-decoration**:none;
30. }
31. #navigation li a:link, #navigation li a:visited{
32. **background-color**:#1136c1;
33. **color**:#FFFFFF;
34. }
35. #navigation li a:hover{                 /\* 鼠标经过时 \*/
36. **background-color**:#002099;           /\* 改变背景色 \*/
37. **color**:#ffff00;                      /\* 改变文字颜色 \*/
38. }
39. </style>

添加完此CSS脚本后，页面呈现的是一个“功能菜单”，页面效果如下：



上图是一个功能菜单的雏形，在脚本中修改对应的第4、20和25行后，这个菜单就变成了一个“导航菜单”，页面效果如下：



* 1. CSS伪类

CSS 伪类用于向选择器添加特殊的效果。常见的伪类有三个，都是在CSS1.0中定义的。他们是：:hover / :visited / :link；:active也是CSS1.0中的伪类，但他的效果显示不明显，并不常用。

所有的伪类罗列如下：

|  |  |  |
| --- | --- | --- |
| **属性名称** | **行为描述** | **CSS版本** |
| :hover | 当鼠标悬浮在元素上方时，向元素添加样式。 | 1.0 |
| :link | 向未被访问的链接添加样式。 | 1.0 |
| :visited | 向已被访问的链接添加样式。 | 1.0 |
| :active | 向被激活的元素添加样式。 | 1.0 |
| :focus | 向拥有键盘输入焦点的元素添加样式。 | 2.0 |
| :first-child | 向元素的第一个子元素添加样式。 | 2.0 |
| :lang | 向带有指定 lang 属性的元素添加样式。 | 2.0 |

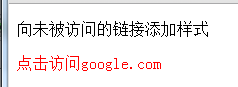
* + 1. :link

伪类：link，向未被访问的链接添加样式。

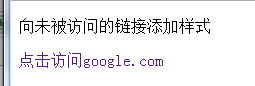
示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>：link 示例</title>
5. <style type="text/css">
6. a:LINK {
7. **color**: red;
8. **text-decoration**: none;
9. }
10. </style>
11. </head>
12. <body>
13. <p>向未被访问的链接添加样式</p>
14. <a href="http://www.google.com">点击访问google.com</a>
15. </body>
16. </html>

页面效果如下：



在CSS代码中，我们设定了当一个链接没有被访问时他的颜色是红色的。但是当访问了这个地址后，在回到这个页面的时候，颜色会默认变为蓝色，如下图：



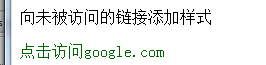
也就是说，当我们没有指定visited伪类的时候，浏览器默认提供：visited属性为蓝色。

* + 1. :visited

向已被访问的链接添加样式。在上面代码的基础上，添加代码如下：

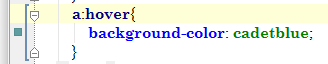
我们设定已经被访问的链接为深绿色。

页面效果如下：



* + 1. :hover

当鼠标悬浮在一个html元素上方时，为这个元素添加的展示效果。:hover伪类经典应用之一就是table表格行变色效果。在上面代码基础上添加如下代码：



页面效果展示如下：



当鼠标悬停在<a />标签上方的时候为他添加背景色。

* + 1. 按钮式超链接

:hover / :visited / :link是最常用的三个伪类，下面利用他们模仿一个按钮式的超链接。

没有鼠标悬停的页面效果如下：



鼠标悬停后的页面效果如下：



代码示例如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>创建 **按钮式** 超链接</title>
5. <style type="text/css">
6. body{
7. **margin**: 20px;
8. }
9. a:LINK, a:VISITED {
10. **color**:#A62020;
11. **padding**:4px 10px 4px 10px;
12. **background-color**:#DDD;
13. **text-decoration**: none;
14. /\*下面是左上和右下的边框模拟\*/
15. **border-top**: 1px solid #EEEEEE;
16. **border-left**: 1px solid #EEEEEE;
17. **border-bottom**: 1px solid #717171;
18. **border-right**: 1px solid #717171;
19. }
20. a:HOVER {
21. **color**: #821818;
22. **padding**: 5px 8px 3px 12px;
23. **background-color**: #CCC;
24. /\*左上颌右下边框模拟效果进行倒置，产生按下去的效果\*/
25. **border-top**: 1px solid #717171;
26. **border-left**: 1px solid #717171;
27. **border-bottom**: 1px solid #EEEEEE;
28. **border-right**: 1px solid #EEEEEE;
29. }
30. </style>
31. </head>
32. <body>
33. <a href="http://www.google.com">Java</a>
34. <a href="http://www.facebook.com">Php</a>
35. <a href="http://www.sina.com">.Net</a>
36. </body>
37. </html>
    1. CSS高级页面布局控制

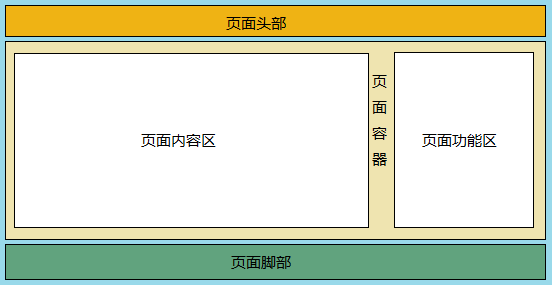


图2.10 – 1

图2.10 – 1是一个标准的网页页面布局模型。网页布局整体上分为三部分：head、bodyContainer和foot。其中bodyContainer作为一个容器来放置页面的内容部分和页面的功能区域部分。在实际开发中，CSS+DIV是页面布局的常用方法。

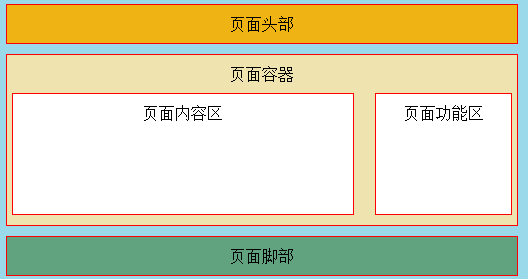
页面布局控制有两种常见的方法：Position绝对定位法和Float浮动法。其中Float浮动法常用且控制较为简洁。下面给出基本Html代码，后面会在此基础上进行CSS布局控制。代码如下：

1. **<html>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. **<title>**布局与定位**</title>**
5. **</head>**
6. **<body>**
7. **<div** id="head"**>**页面头部 **</div>**
9. **<div** id="container"**>** 页面容器
10. **<div** id="content"**>**页面内容区**</div>**
11. **<div** id="side"**>**页面功能区**</div>**
12. **</div>**
14. **<div** id="foot"**>**页面脚部**</div>**
15. **</body>**
16. **</html>**
    * 1. 绝对定位法

全部内容都在脚本中，脚本代码如下：

1. <style type="text/css">
3. body{
4. **background-color**: #99D9EA;
5. **text-align**: center;
6. }
7. /\*第一步：调整头、容器和脚的宽度\*/
8. #head,#bodyContainer,#foot{
9. **width**: 500px;
10. }
12. #head,#bodyContainer,#content,#side,#foot{
13. /\*第二步：让整个页面居中\*/
14. /\*这是居中的一种常用方法，左右自适应\*/
15. **margin**:10px auto 10px auto;
17. **padding**:10px 5px 10px 5px;
18. **border**: 1px solid red;
19. }
21. /\*第三步：为content和side设置定位依据\*/
22. /\*absolute定位依据：离它最近的那个含有position属性的包含框\*/
23. #bodyContainer{
24. /\*border:0px;\*/
25. **background-color**:#EFE4B0;
26. **position**: relative;
27. **height**: 150px;
28. }
30. #content{
31. **position**: absolute;
32. **width**: 330px;
33. **height**: 100px;
34. **background-color**:white;
35. }
37. #side{
38. **margin-left**: 363px;
39. **height**: 100px;
40. **background-color**:white;
41. }
42. #head{
43. **background-color**: #EFB314;
44. }
45. #foot{
46. **background-color**: #61A37E;
47. }
48. </style>

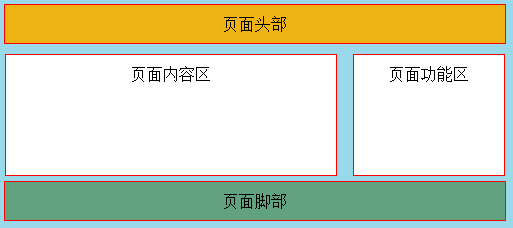
页面效果如下：



* + 1. 浮动定位法

利用浮动来设定页面布局，现在比较流行的一种方法，相比较绝对定位来设定页面布局，这种方法相对简单一些。

利用浮动来设定页面布局，页面效果显示如下：



代码相对与绝对定位布局控制只是做了一点小修改，如下：

1. <style type="text/css">
2. body{
3. **background-color**: #99D9EA;
4. **text-align**: center;
5. }
7. #head{
8. **background-color**: #EFB314;
9. }
11. /\*第一步：调整头、容器和脚的宽度\*/
12. #head,#bodyContainer,#foot{
13. **width**: 500px;
14. }
16. #head,#bodyContainer,#content,#side,#foot{
17. /\*第二步：让整个页面居中\*/
18. /\*这是居中的一种常用方法，左右自适应\*/
19. **margin**:5px auto 5px auto;
21. **padding**:10px 0px 10px 0px;
22. **border**: 1px solid red;
23. }
25. #bodyContainer{
26. **border**:0px;
27. **padding**: 0px;
28. **margin**:0px auto 0px auto;
29. }
31. #content{
32. **float**:left;
33. **width**: 330px;
34. **height**: 100px;
35. **background-color**:white;
36. }
38. #side{
39. **float**:right;
40. **width**: 150px;
41. **margin-left**: 10px;
42. **height**: 100px;
43. **background-color**:white;
44. }
45. #foot{
46. **clear**: both;
47. **background-color**: #61A37E;
48. }
49. </style>
    * 1. 变宽布局

在2.10.2代码的基础上，脚本中的width属性设置改为百分比，这样的话网页布局会随着浏览器的大小而产生变化，而不是固定不变的、浏览器有滚动条出现。但在实际开发中用的很少。这里不在过多介绍。

1. JavaScript 高级特性

一个完整的JavaScript实现由三个部分组成：核心(ECMAscript)、文档对象模型(Document Object Model)和浏览器对象模型(Browser Object Model)。JavaScript为浏览器而生。网页设计要素分为三个层次：内容、样式和行为。在w3c的标准要求中，内容用Html表现；样式用CSS规定；行为用脚本控制。

Html的基本作用就是为网页内容分块并作出必要的标记，如id，class属性。CSS则对这些小块内容进行颜色、字体、边框和位置上的控制，通俗点就是为每一个小个体画好妆并在浏览器中排好队。到现在为止，页面还是安静的，体现不出任何动作；如果想让页面有反应，去做一些事，这时就需要引入JavaScript脚本了。

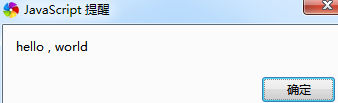
如果Html页面是一个“女人”，那么CSS就是“化妆品”，javascript则是女人美丽的笑容。

* 1. 返回值方法

直接看代码，代码如下：

1. <script type="text/javascript">
2. //定义参数
3. **function** returnFunc(param1,param2){
4. **return** param1+" , "+param2; //返回参数
5. }
6. //调用方法，参数赋值。
7. **var** result = returnFunc("hello","world");
8. alert(result);
9. </script>

页面效果如下：



* 1. 字符串方法

实例化一个字符串：var str="实例化字符串abcdefg测试"; 后面依托他测试。

* + 1. 字符串length 属性

document.write(str+"<br/>");

document.write("字符串长度= "+str.length+" <br/>");

效果如下：



* + 1. indexOf 方法

使用indexOf() 来定位字符串中某一个指定的字符首次出现的位置。

var strIndexOf = str.indexOf("a", 0);

document.write("a首次出现在："+strIndexOf+"<br/>");



* + 1. replace()方法

使用replace() 方法在字符串中用某些字符替换另一些字符。

var strReplace = str.replace("abcde", "Yangcl-")

document.write(strReplace+"<br/>");

效果如下：



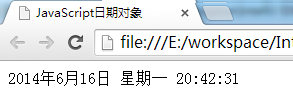
* 1. 日期方法

|  |  |
| --- | --- |
| **函数名** | **方法描述** |
| var date=new Date(); 初始化Date对象。 | |
| getFullYear() | 从 Date对象以四位数字返回年份。 |
| getMonth() | 从 Date 对象返回月份 (0~ 11)。 |
| getDate() | 从 Date 对象返回一个月中的某一天 (1~31)。 |
| getHours() | 返回 Date 对象的小时 (0~23)。 |
| getMinutes() | 返回 Date 对象的分钟 (0~59)。 |
| getSeconds() | 返回 Date 对象的秒数 (0~59)。 |
| getDay() | 从 Date 对象返回一周中的某一天 (0~6)。 |

以显示年月日星期和时分秒为例，示例源代码如下：

1. <script type="text/javascript">
2. **var** date=**new** Date();
3. **var** day=date.getDay();
4. **var** week;
5. **switch**(day)
6. {
7. **case** 0:week="星期日";**break**;
8. **case** 1:week="星期一";**break**;
9. **case** 2:week="星期二";**break**;
10. **case** 3:week="星期三";**break**;
11. **case** 4:week="星期四";**break**;
12. **case** 5:week="星期五";**break**;
13. **case** 6:week="星期六";**break**;
14. }
15. **var** today = date.getFullYear()+"年"+(date.getMonth()+1)
16. +"月"+date.getDate()+"日  ";
17. **var** time = date.getHours()+":"+date.getMinutes()+":"+date.getSeconds();
19. **var** theTime = today +" " + week + " " +time;
20. document.write(theTime+"<br/>");
21. </script>

页面效果显示如下：



* 1. JavaScript 数组
     1. JavaScript数组声明与遍历

JavaScript数组的声明有两种方式：第一种不指定数组的大小，第二种指定数组的大小。

示例源代码如下：

1. <script type="text/javascript">
2. **var** arr=**new** Array();
3. **var** arr2=**new** Array(3);
4. arr[0]="jack";
5. arr[1]="marry";
6. arr[2]="tom";
7. arr[3]="lucy";
9. arr2[0]=2;
10. arr2[1]=1;
11. arr2[2]=5;
13. **for**(**var** i=0;i<arr.length;i++)
14. {
15. document.write(arr[i]+"<br/>");
16. }
17. document.write("<hr/>");
18. **var** o;
19. **for**(o **in** arr2)
20. {
21. document.write("下标：" + o +" | 元素："+arr2[o]+"<br/>");
22. }
23. </script> "

数组的遍历也有2种方法，一种是传统的方法：

1. **for**(**var** i=0;i<arr.length;i++)
2. {
3. document.write(arr[i]+"<br/>");
4. }

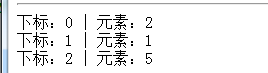
就是一个一个去遍历，结果如下图：



遍历数组的另一种方法是类似与Java的for-each，如下：

1. **var** o;
2. **for**(o **in** arr2)
3. {
4. document.write("下标：" + o +" | 元素："+arr2[o]+"<br/>");
5. }

这里声明的 var o; o 代表数组的下标。



值得了解的是，JavaScript对数组下标规定很宽泛，甚至允许使用非数字：

1. arr2[0]=2;
2. arr2[**"a"**]=1;
3. arr2[**"b"**]=5;

上面代码遍历后的结果如下：

但通常不建议这样做。

* + 1. 数组常用方法

在上面代码基础上添加如下4段测试代码。

* + - 1. 元素排序sort()方法

1. document.write("元素排序sort()："+arr.sort()+"<br/>");
2. document.write("元素排序sort()："+arr2.sort()+"<br/>");

结果如下：



* + - 1. 元素字符串组合join()方法

数组元素组合成字符串默认是使用逗号来隔离每个元素，然后组成一个字符串，你也可以自定义一个分隔符，比如“#”。

1. document.write("元素字符串组合join()："+arr.join()+"<br/>");
2. document.write("元素字符串组合join()："+arr2.join("#")+"<br/>");

结果如下：



* + - 1. 合并数组元素concat()方法

合并数组元素就是将一个数组中的元素放到另一个数组中。

1. document.write("合并数组元素concat()："+arr.concat(arr2)+"<br/>");
2. document.write("<hr/>");

结果如下：



* + - 1. 颠倒数组元素reverse()方法

将数组元素逆序排序，与sort()方法相反。

1. document.write("颠倒数组元素reverse()："+arr.reverse()+"<br/>");

结果如下：



* 1. JavaScript 全局函数

JavaScript的函数分为：全局函数 与 对象函数。全局函数不属于任何一个内置对象。

在W3School中：<http://www.w3school.com.cn/jsref/jsref_obj_global.asp>描述了所有全局函数的详细信息。JS包含以下7个常用的全局函数，用于一些常用的功能，如下表：

|  |  |
| --- | --- |
| **函数** | **功能描述** |
| eval() | 计算某个字符串，并执行其中的的 JavaScript 代码。 |
| escape() | 对字符串进行编码。 |
| isNan() | 检查某个值是否是数字。 |
| isFinite() | 检查某个值是否为有穷大的数。 |
| parseFloat() | 解析一个字符串并返回一个浮点数。 |
| parseInt() | 解析一个字符串，并返回一个整数 |
| unescape() | 对由 escape() 编码的字符串进行解码。 |

* + 1. eval()

该方法只接受原始字符串作为参数，如果 string 参数不是原始字符串，那么该方法将不作任何改变地返回。因此请不要为 eval() 函数传递 String 对象来作为参数。

1. <script type="text/javascript">
2. **var** str="1+2+3";
3. alert(eval(str));
4. </script>

效果如下：



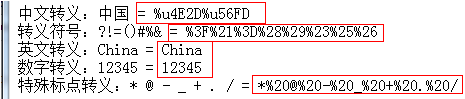
值得注意的是：如果**var** str="1+a+3";   是无法执行的。相比Java，这个函数是JavaScript最变态的。

* + 1. escape()

已编码的 string 的副本。其中某些字符被替换成了十六进制的转义序列。

该方法不会对 ASCII 字母和数字进行编码，也不会对下面这些 ASCII 标点符号进行编码： \* @ - \_ + . / 。其他所有的字符都会被转义序列替换。可以使用 unescape() 对 escape() 编码的字符串进行解码。

1. <script type="text/javascript">
2. document.write("中文转义：中国 = "+escape("中国") + "<br />")
3. document.write("转义符号：?!=()#%& = "+escape("?!=()#%&") + "<br />")
5. document.write("英文转义：China = "+escape("China") + "<br />")
6. document.write("数字转义：12345 = "+escape("12345") + "<br />")
7. document.write("特殊标点转义：\* @ - \_ + . / = "+
8. escape("\* @ - \_ + . /") + "<br />") ;
9. </script>



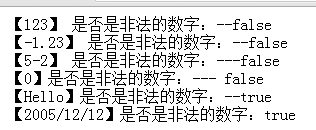
* + 1. isNan()

isNaN() 函数用于检查其参数是否是非数字值。isNaN() 函数可用于判断其参数是否是 NaN，该值表示一个非法的数字（比如被 0 除后得到的结果）。

isNaN() 函数通常用于检测 parseFloat() 和 parseInt() 的结果，以判断它们表示的是否是合法的数字。当然也可以用 isNaN() 函数来检测算数错误，比如用 0 作除数的情况。示例代码如下：

1. <script type="text/javascript">
2. document.write("【123】 是否是非法的数字：--" + isNaN(123) +  "<br />")
3. document.write("【-1.23】 是否是非法的数字：--" + isNaN(-1.23)+ "<br />")
4. document.write("【5-2】 是否是非法的数字：---" + isNaN(5-2) +  "<br />")
5. document.write("【0】是否是非法的数字：--- " + isNaN(0) +  "<br />")
6. document.write("【Hello】是否是非法的数字：--" + isNaN("Hello") + "<br />")
7. document.write("【2005/12/12】是否是非法的数字：" + isNaN("2005/12/12"))
8. </script>

结果如下：



* 1. JavaScript 对象函数

JavaScript对象函数有很多，主要分为Browser对象函数和DOM对象函数，这里以Window对象函数举例，Window对象函数属于Browser对象函数。

Alert()：弹出消息框

confirm()：确认对话框

prompt()：显示可提示用户输入的对话框

setTimeout()：执行一次

setinterval()：执行多次

open()：打开一个新的窗口，参数是一个URL地址。

下面以setTimeout()和setinterval()为例说明。

* + 1. setTimeout()

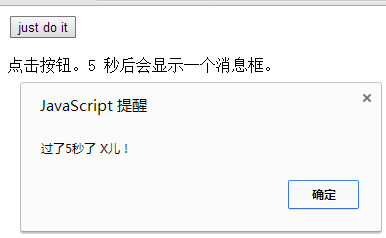
setTimeout() 方法用于在指定的毫秒数后调用函数或计算表达式。setTimeout() 只执行 code 一次。如果要多次调用，请使用 setInterval() 或者让 code 自身再次调用 setTimeout()。

* + - 1. 简单的计时

示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <script type="text/javascript">
5. **function** timedMsg()
6. {
7. **var** t=setTimeout("alert('过了5秒了 X儿！')",5000)
8. }
9. </script>
10. </head>
12. <body>
13. <form>
14. <input type="button" value="just do it" onClick = "timedMsg()">
15. </form>
16. <p>点击按钮。5 秒后会显示一个消息框。</p>
17. </body>
18. </html>

页面效果如下：

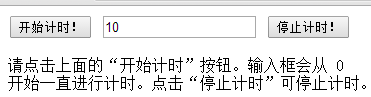


* + - 1. 无穷循环计时 - 带停止按钮

clearTimeout()方法可以取消由setTimeout()方法设置的timeout。示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <script type="text/javascript">
5. **var** c=0
6. **var** t
7. **function** timedCount()
8. {
9. document.getElementById('txt').value=c
10. c=c+1
11. t=setTimeout("timedCount()",1000)
12. }
14. **function** stopCount()
15. {
16. clearTimeout(t)
17. }
18. </script>
19. </head>
21. <body>
22. <form>
23. <input type="button" value="开始计时！" onClick="timedCount()">
24. <input type="text" id="txt">
25. <input type="button" value="停止计时！" onClick="stopCount()">
26. </form>
28. <p>
29. 请点击上面的“开始计时”按钮。输入框会从 0 <br/>
30. 开始一直进行计时。点击“停止计时”可停止计时。
31. </p>
32. </body>
33. </html>

页面显示如下：



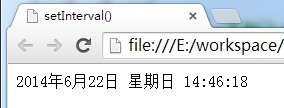
* + 1. setInterval()

setInterval() 方法可按照指定的周期（以毫秒计）来调用函数或计算表达式。

setInterval() 方法会不停地调用函数，直到 clearInterval() 被调用或窗口被关闭。由 setInterval() 返回的 ID 值可用作 clearInterval() 方法的参数。示例代码如下：

1. <html>
2. <head>
3. <meta charset="UTF-8">
4. <title>setInterval()</title>
5. <script type="text/javascript">
6. **function** dayTime(){
7. **var** date=**new** Date();
8. **var** day=date.getDay();
9. **var** week;
10. **switch**(day)
11. {
12. **case** 0:week="星期日";**break**;
13. **case** 1:week="星期一";**break**;
14. **case** 2:week="星期二";**break**;
15. **case** 3:week="星期三";**break**;
16. **case** 4:week="星期四";**break**;
17. **case** 5:week="星期五";**break**;
18. **case** 6:week="星期六";**break**;
19. }
20. **var** today = date.getFullYear()+"年"+(date.getMonth()+1)
21. +"月"+date.getDate()+"日  ";
22. **var** time = date.getHours()+":"+date.getMinutes()+":"+date.getSeconds();
24. **var** theTime = today +" " + week + " " +time;
25. document.getElementById("today").innerHTML=theTime;
26. }
28. window.setInterval("dayTime()", 1000);
29. </script>
30. </head>
31. <body>
32. <div id="today"></div>
33. </body>
34. </html>

显示结果如下：



* 1. Function 对象
     1. JavaScript方法定义

JavaScript中使用function关键字来定义一个方法。在JavaScript中方法也是一个对象：Object；所以说，方法也可以作为另一个方法的参数来使用。下面将定义一个简单的方法，并调用它。

**<script type="text/javascript">**

**//定义一个方法**

**function say(name , age)**

**{**

**alert(name+"今年"+age+"岁了");**

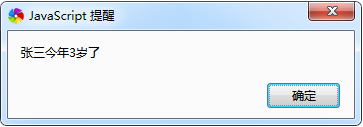
**}**

**//调用say方法**

**say("张三" , 3);**

**</script>**

页面效果截图如下：



下面改进代码，将方法作为另一个方法的参数来使用。

**<script type="text/javascript">**

**//定义一个方法**

**function say(name,age)**

**{**

**alert(name+"今年"+age+"岁了");**

**}**

**//say方法作为参数**

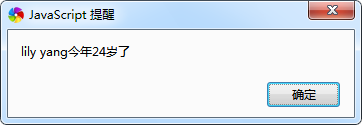
**var newSay = new Function("name" , "age" , "say(name , age)");**

**newSay("lily yang" , 24);**

**</script>**

注意：红色和蓝色标注的字体颜色需要相同，否则浏览器会报错。

页面截图效果如下：



* + 1. Function对象 的 属性与方法

function关键字用来定义一个方法，方法继承自Object，所以：方法也是对象。其中【方法对象】的属性length：他将返回一个方法参数的个数。

**<script type="text/javascript">**

**//定义一个方法**

**function say(name , age)**

**{**

**alert(name+"今年"+age+"岁了");**

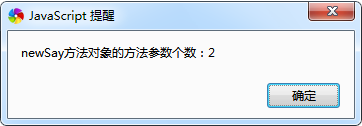
**}**

**var newSay = new Function("name","age","say(name , age)");**

**alert("newSay方法对象的方法参数个数：" + newSay.length)**

**</script>**

页面效果如下：



【方法对象】的方法，常用的有toString()和valueOf()。这两个方法的作用是一样的，他们都是将方法的源码找出来。

**<script type="text/javascript">**

**var sayFunc = new Function("name", "age", "alert(name+'今年'+age+'岁了')");**

**alert(sayFunc.toString());**

**alert(sayFunc.valueOf());**

**</script>**

页面显示如下：



aoonymous，返回一个匿名的方法。

* 1. JavaScript 闭包
     1. JavaScript变量作用域

JavaScript变量作用域指在<script type="text/javascript"> anything ……</script>中包含的全局变量和function方法体内包含的私有变量。其中全局变量是都可以访问的，而方法体内的私有变量其他方法是无法访问的。

针对私有变量，产生了 **闭包** 的概念：闭包就是能够读取其他函数内部变量的函数。由于在Javascript语言中，只有函数内部的子函数才能读取局部变量，因此可以把闭包简单理解成“定义在一个函数内部的函数”。所以，在本质上，闭包就是将函数内部和函数外部连接起来的一座桥梁。

下面给出全局变量的访问示例：

**<script type="text/javascript">**

**var a = 95568; // 定义一个全局变量**

**function funcGetA() {**

**alert("全局变量a = "+a);**

**}**

**funcGetA();**

**</script>**

页面显示效果如下：



事实上，全局变量不仅可以定义在JavaScript方法的外部，也可以定义在JavaScript方法体的内部，例如下面这段代码中，依然可以访问到方法内部的变量。

**<script type="text/javascript">**

**function funcGetA() {**

**a = 95568; //**在这里不使用“var”声明，虽然变量在方法内部，但他也是

**} //**一个全局变量。

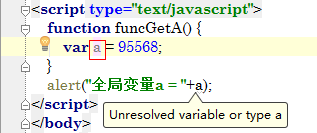
**alert("全局变量a = "+a);**

**</script>**

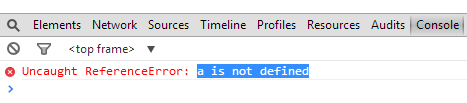
页面显示效果如下：



但是此时如果在变量a的前面加上“var”声明，那么a就变成了方法体内的局部变量，也就是方法的私有变量。



从代码提示中可以看到，无法解析变量a，方法体内的变量a是灰色的，代表没有任何方法、变量引用了他。当刷新浏览器后：



在控制台：a is not defined，a未定义。

* + 1. JavaScript闭包原理

OK，我们现在想要访问方法体内的变量a。这时，我们需要闭包来作为桥梁。

**<script type = "text/javascript" >**

**function func()**

**{**

**var a = 22; // 定义私有变量，属于func()本身外界无法访问。**

**function func\_2() // 相当于一个对外公开的接口，可以理解为getter()。**

**{**

**return ++a;**

**}**

**return func\_2; // 注意：这里返回的是一个方法对象**

**}**

**// 访问方式一**

**var result=func();**

**alert( "访问func的私有局部变量a，++a = " + result() );**

**// 访问方式二**

**alert( "访问func的私有局部变量a，++a = " + func()() );**

**</script>**

针对私有变量，产生了 **闭包** 的概念：闭包就是能够读取其他函数内部变量的函数。由于在Javascript语言中，只有函数内部的子函数才能读取局部变量，因此可以把闭包简单理解成“定义在一个函数内部的函数”。所以，在本质上，闭包就是将函数内部和函数外部连接起来的一座桥梁。

代码中的 func\_2()就是闭包；他访问了func()对象中的私有变量 **a**。注意代码中的红色字体，这两种访问方式中，第二种比较另类，但常常会有爱装逼的程序员会这么写。

JavaScript与回调函数

* 1. Html文档对象模型

文档对象模型是整个Web前端页面开发中最重要且核心的概念，相比较Jquery、ExtJS来讲他更加贴近浏览器的核心技术，而不是仅仅做做表面上肤浅的用户界面体验。对他熟练的应用往往能在开发中带来意想不到的效果。

DOM：Document Object Model，中文翻译：文档对象模型。DOM 定义了访问 HTML 和 XML 文档的标准，是W3C（万维网联盟）的定义的一套标准。

W3C DOM 标准被分为 3 个不同的部分：

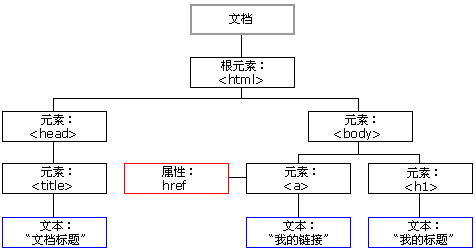
* 核心 DOM - 针对任何结构化文档的标准模型；
* XML DOM - 针对 XML 文档的标准模型；
* HTML DOM - 针对 HTML 文档的标准模型；

也可以这样理解DOM：HTML DOM 是关于如何获取、修改、添加或删除 HTML 元素的标准。

* + 1. HTML DOM 节点

在HTML DOM中，**所有事物都是节点**。DOM 是被视为节点树的 HTML。整个文档是一个文档节点，每个 HTML 元素是元素节点，HTML 元素内的文本是文本节点，每个 HTML 属性是属性节点。

HTML DOM 树：



节点树中的节点彼此拥有层级关系。父（parent）、子（child）和同胞（sibling）等术语用于描述这些关系。父节点拥有子节点。同级的子节点被称为同胞（兄弟或姐妹）。在节点树中，顶端节点被称为根（root），每个节点都有父节点、除了根（它没有父节点），一个节点可拥有任意数量的子，同胞是拥有相同父节点的节点。

* + 1. DOM的操作与控制

对DOM的操作与控制依托于DOM的方法和属性。

**方法**是我们可以在节点(HTML元素)上执行的动作；这些动作包括：获取、创建、删除、替换和添加等等。

**属性**是你能够获取或设置的值，比如节点的名称或内容。

常用的HTML DOM方法：

|  |  |
| --- | --- |
| **方法名称** | **动作** |
| getElementById(id) | 获取带有指定id的节点(元素) |
| appendChild(node) | 插入新的子节点(元素) |
| removeChild(node) | 删除子节点(元素) |

例如获取一个id为“intro”的节点：

var element=document.getElementById("intro");

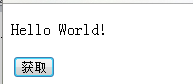
常用HTML DOM属性：

|  |  |
| --- | --- |
| **属性名称** | **动作** |
| innerHTML | 节点(元素)的文本值 |
| parentNode | 节点(元素)的父节点 |
| childNodes | 节点(元素)子节点 |
| attributes | 节点(元素)的属性节点 |

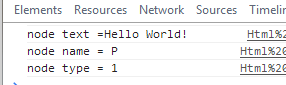
下面以getElementById()这个方法为例演示对属性的操作。示例代码如下：

1. <html>
2. <head>
3. <title>DOM的操作与控制</title>
4. <meta charset="UTF-8">
5. <script>
6. **function** getIWant(){
7. **var** nodetext=document.getElementById("intro").innerHTML;
8. console.log("node text =" + nodetext);
9. **var** nodename = document.getElementById("intro").nodeName;
10. console.log("node name = " + nodename);
11. **var** nodetype = document.getElementById("intro").nodeType;
12. console.log("node type = " + nodetype);
13. }
14. </script>
15. </head>
16. <body>
17. <p id="intro">Hello World!</p>
18. <input type="button" onclick="getIWant()" value="获取">
19. </body>
20. </html>

页面截图效果如下：



浏览器控制台内容输出如下：



方法和属性还有很多，具体参照API。建议都看看。

* + 1. 关于document

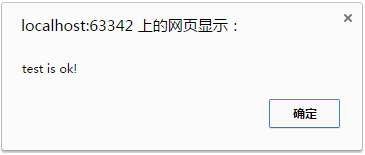
**注意！**只能通过节点的innerHTML属性来访问文本节点的值。

* 1. JavaScript if...else

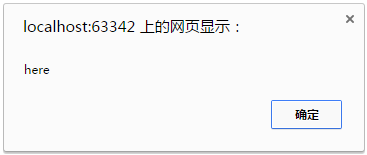
JavaScript 中 If...Else 中 if可以接受的条件相对较多，他不只接受true / false，还可以接受其他的参数作为判定依据。下面一一列举。

* + 1. 数字非0,非NaN (0 或NaN)

**var** i = 0 or NaN;  
**if**(i){  
 alert(**'here'**);  
}**else**{  
 alert(**'test is ok!'**);  
}

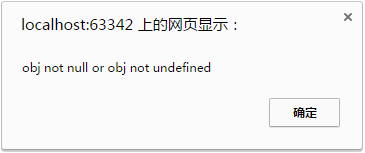


当**var** i = -1 or 1 or 其他非零数字时;

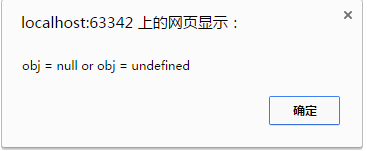


* + 1. 对象非null (null或undefined)

**var** obj = $(**'a'**);   
**if**(obj){  
 alert(**'obj not null or obj not undefined'**);  
}**else**{  
 alert(**'obj = null or obj = undefined'**);  
}

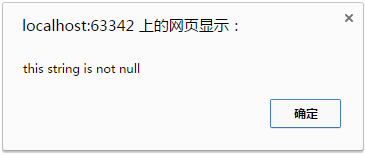


设置**var** obj = **null or undefined**;



* + 1. 字符串非空串 / 空串("")

**var** str = **'awe123'**;   
**if**(str){   
 alert(**'this string is not null'**);   
}**else**{   
 alert(**'string . length = 0'**);   
}



* + 1. 三元操作符

**if** (foo) bar(); **else** baz(); ==> foo?bar():baz();

1. Jsp

Java版的HTML。。。。。没得可说了

1. El表达式

EL又名表达式语言，Expression Language。从Jsp2.0开始引入，有Java Community Process的Jsp标准标记库专家组和JSP2.0专家组共同开发。表达式语言是一种类似于JavaScript的脚本类语言，主要用于在网页上显示动态内容(后台发过来的List、Object等)，目的是替换Jsp页面的Java脚本并完成页面java脚本的复杂功能，减少Jsp页面的维护代价。

* 1. EL语法

EL语法单一，且非常简单：

${EL表达式内容}。

所有的EL表达式都是以“${”未开始，以“}”为结束。例如：**${sessionScope.user.sex}**，意思是：从Session范围中取得用户性别。他等价于下面的JSP脚本：

**<%User user = (User)session.getAttribute(“user”);**

**String sex = user.getSex(); %>**

对比可以看出EL表达式比传统的JSP脚本更加简介。下面给出一个简单的例子：

**<%@ page contentType="text/html;charset=UTF-8" language="java" %>**

**<html>**

**<head>**

**<title>测试-1</title>**

**</head>**

**<body>**

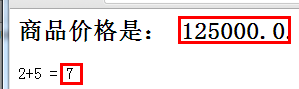
**<h2>商品价格是： ${12.5E4}.</h2>**

**2+5 = :${2 + 5}**

**</body>**

**<html>**

页面展示效果如下：



* 1. EL表达式的启用与关闭

在JSP页面中可以控制是否开启EL表达式的计算。

<%@ page isELIgnored="[ true|false] " %>。如果该属性设置为true则代表JSP容器(Tomcat)将忽略EL表达式的计算。

**<%@ page contentType="text/html;charset=UTF-8" language="java" %>**

**<%@ page isELIgnored="true" %>**

**<html>**

**<head>**

**<title>逻辑EL</title>**

**</head>**

**<body>**

**<H1>逻辑EL</H1>**

**<table border=2 style="width: 400px">**

**<tr>**

**<td>运算</td>**

**<td>EL表达式</td>**

**<td>结果</td>**

**</tr>**

**<tr>**

**<td>加法</td>**

**<td> ${'${'}1+1} </td>**

**<td>${1+1}</td>**

**</tr>**

**<tr>**

**<td>文字</td>**

**<td> ${'${'}Lucifur} </td>**

**<td>${"Lucifur"}</td>**

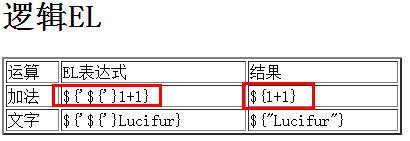
**</tr>**

**</table>**

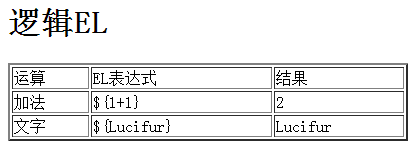
**</body>**

**</html>**

上面代码isELIgnored设置为true，代表忽略EL表达式计算。效果图如下：



在容器的运行时环境中，一般isELIgnored属性默认都为false，除非你显示的声明为true。上面代码将isELIgnored设置为false时页面显示如下：



在源代码标识为黄色的地方：${'${'}……}，注意如果在EL表达式中出现${'${'}，则EL表达式输出“${”并把“}”后面的表达式看作字符串输出。

* 1. EL+JSTL综合应用页面取值与传值

服务器端的页面跳转为EL表达式提供了值。直白的讲就是，JavaWeb的后台中提供

返回值，他返回的是一个页面。现在主流的Java后台框架有Struts2、SpringMVC等。

SpringMvc提供页面跳转的那个层叫做Controller层。即：\*\*\*\*\*Controller.java类中的一个方法。SpringMVC比Struts2先进的多，省去了Struts2中繁杂的页面跳转配置，直接就可以返回页面。

【代码5.3.1】如下：

@Controller

@RequestMapping("/account")

**public** **class** AccountController2 {

@Autowired

**private** ICinemaUserService cuService;

@RequestMapping(value="/logout")

**public** String logout(ModelMap model){

List<CinemaUser> userList = cuService.selectIdAndName();

model.put("userListTest", userList);

**return** "redirect:/login.jsp";

}

}

在上面的代码段中，logout.do返回的结果是login.jsp即退出的时候直接跳转到登录页。

同时！在login.jsp页面中还可以通过EL表达式拿到userList中的值。但EL表达式在这个实际项目中是如何取值的呢？请注意看：

${ userListTest}

Struts2提供页面返回的那个层是Action层。即：\*\*\*\*\*\*Action.java类中的一个方法，他的返回值是SUCCESS，然后再通过xml配置文件来实现页面跳转，这个框架比较繁琐，但貌似现在用的公司还是很多，这里就不再介绍了。

通常来讲，EL表达式都是和JSTL标签来联合使用，比如上面的代码段【代码5.3.1】

<c:forEach items=*"*${userListTest}*"* var=*"*user*"*>

<option value=*"*${user.id }*"*

<c:if test="${user.id == 26}">selected="selected"</c:if> >

${user.name }

</option>

</c:forEach>

这是项目实际开发中常见的写法。

1. JSTL
   1. Jstl简介

JSTL 核心标签库标签共有13个，功能上分为4类：

1. 表达式控制标签：out、set、remove、catch

2. 流程控制标签：if、choose、when、otherwise

3. 循环标签：forEach、forTokens

4. Url操作标签：import、url、redirect

5. 格式化标签 fmt

使用标签时，根据标签类型的不同，要在jsp文件头加入以下代码：

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

前四种属于C标签，第五种属于格式化标签。C标签中比较常用的是流程控制标签和循环标签。格式化标签比较常用的是格式化后台传向页面的时间，其他的都比较少用，因为依托于JavaScript、JQuery等可以更轻松的替代JSTL的这些不常用标签，甚至依托于JQuery和他的衍生框架可以完全替换JSTL标签和EL表达式，只是看怎么用着方便会怎么用。通常采用JavaScript、JQuery、EL表达式和JSTL相结合的方式来进行页面开发，

总之，取他们各自最方便的地方使用

【代码6.1.1】如下：

<**c**:forEach items=*"*${userListTest}*"* var=*"*user*"*>

<option value=*"*${user.id }*"*

<**c:if** test="${user.id == 26}">selected="selected"</c:if> >

${user.name }

</option>

</**c**:forEach>

<**fmt**:formatDate value="${user.time}" pattern=*"*yyyy-MM-dd HH:mm:ss*"*/>

代码6.1.1给出了比较不错的例子，结合了EL表达式、JSTL。其他就没什么可说的了，很简单的一个技术点，但又很常用。 下面的两个链接比较不错，介绍的很好，可做参考，本文档不在熬述JSTL

C标签介绍

<http://www.cnblogs.com/lihuiyy/archive/2012/02/24/2366806.html>

fmt标签介绍

<http://www.cnblogs.com/lihuiyy/tag/JSTL/>

1. Ajax & Json

AJAX即“Asynchronous Javascript And XML”（异步JavaScript和XML），是指一种创建交互式网页应用的网页开发技术。通过在浏览器页面与服务器进行少量数据交换，AJAX 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。传统的网页（不使用 AJAX）如果需要更新内容，必须重载整个网页页面。

EL表达式的5.3节，就是传统的网页，他没有使用到Ajax，使用的是页面跳转。在实际开发中，**页面跳转是和Ajax联合使用**的，各取所长。他们各自的优势如下：

|  |  |
| --- | --- |
| 页面跳转 | 展示一个数据库表中的所有记录，利用页面跳转携带的所有后台数据，结合EL表达式、JSTL标签可以快速的构建出整个页面内容。如：EL表达式的5.3节。 |
| Ajax | 更新页面列表中的一条记录，ajax与后台进行交互，Java后台更新完数据库后向页面发送一个Json串、或0和1，JavaScript脚本收到值以后对指定的Html文档(html标签)进行添加、隐藏、删除、替换等操作。 |

Ajax不用服务器重新返回给浏览器整个页面，而是通过JavaScript来对页面的局部进行更新，将页面内容的处理和拼接交给用户浏览器去处理，节省服务器内存消耗，极大减轻服务器压力。

Asynchronous Javascript And XML，以前用的是XML文本格式，但随着Json的出现和普及，XML被替换。据说淘宝的页面生成都是通过服务器发来的Json串，然后配合前端工程师的JavaScript脚本进行解析，拼接成的整个页面。

Ajax完全可以替代页面跳转。

* 1. ajax原理XmlHttpRequest对象

XMLHttpRequest是ajax的核心机制，它是在IE5中首先引入的，是一种支持异步请求的技术。简单的说，也就是javascript可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

XMLHttpRequest对象属性：

|  |  |
| --- | --- |
| 属性名称 | 属性描述 |
| onreadyStateChange | 每次状态改变所触发事件的事件处理程序。 |
| responseText | 从服务器进程返回数据的字符串形式。 |
| responseXML | 从服务器进程返回的DOM兼容的文档数据对象。 |
| status | 从服务器返回的数字代码，如 404未找到|200已就绪 |
| statusText | 伴随状态码的字符串信息 |
| readyState | 对象状态值 |

重要的是实际应用，无需了解太深，没有意义。

* 1. ajax in jQuery

JQuery封装了完美的Ajax访问方法，没必要使用JavaScript原生的方法，JS原生方法复杂繁琐，还要考虑浏览器兼容性问题，不提倡。

封装好的代码如下：

7.2.1 浏览器端JS方法封装

/\*\*

\* @描述: 异步交互

\* @作者: Yangcl

\* @时间: 2014-11-26 : 16-13-25

\* @param url\_: var url\_ = "${basePath}user/r\_updateUser\_ajax.action";

\* @param data\_: {userId:userId, questionnaireType:0}

\* or $("#the-form-id").serializeArray()

\* @return msg\_

\*/

**function** createAjax(url\_ , data\_)

{

**var** msg\_ = **null**;

$.ajax({

dataType : "text",

type : "POST", // 一般不用GET

url : url\_,

data : data\_,

async : **false**,

success : **function**(msg) {

**if** (**null** != msg) {

msg\_ = msg;

}

},

error: **function**(msg) {

msg\_ = msg;

}

});

**return** msg\_;

}

7.2.2 服务器端Java代码封装

Struts2版本

**import** java.io.PrintWriter;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.struts2.ServletActionContext;

**import** com.opensymphony.xwork2.ActionContext;

/\*\*

\* **@描述**: 异步交互

\* **@作者**: Yangcl

\* **@时间**: 2014-11-26 : 16-13-25

\*/

**public** **class** AjaxUtil {

**public** **static** **void** ajaxOut(String msg) **throws** Exception {

// 验证模块名称

ActionContext context = ActionContext.*getContext*();

HttpServletResponse response =

(HttpServletResponse) context.

get(ServletActionContext.***HTTP\_RESPONSE***);

HttpServletRequest request =

ServletActionContext.*getRequest*();

// 设置接收信息的字符集

request.setCharacterEncoding("UTF-8");

// 设置输出信息的格式及字符集

response.setContentType("text/xml; charset=UTF-8");

response.setHeader("Cache-Control", "no-cache");

// 创建输出流对象

PrintWriter out = response.getWriter();

out.write(msg);

out.close();

}

}

SpringMvc版本-1

**import** java.io.PrintWriter;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

/\*\*

\* **@name** AjaxUtil

\* **@description** 异步交互

\* **@author** Yangcl

\* **@createTime** 2015-08-09 21:16

\*/

**public** **class** AjaxUtil

{

**public** **static** **void** ajaxOut(String msg ,

HttpServletRequest request ,

HttpServletResponse response) **throws** Exception

{

// 设置接收信息的字符集

request.setCharacterEncoding("UTF-8");

// 设置输出信息的格式及字符集

response.setContentType("text/xml; charset=UTF-8");

response.setHeader("Cache-Control", "no-cache");

// 创建输出流对象

PrintWriter out = response.getWriter();

out.write(msg);

out.close();

}

}

SpringMvc版本-2

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.ResponseBody;

**import** com.alibaba.fastjson.JSONObject;

@Controller

@RequestMapping(value = "category")

**public** **class** CategoryController {

@Autowired

**private** ICategoryService service;

// 方法内返回json

@RequestMapping(value="pageJson",

produces = { "application/json;charset=utf-8" })

@ResponseBody

**public** JSONObject findByPage()

{

JSONObject obj = **new** JSONObject();

obj.put("data", "the test data");

**return** obj;

}

// 方法外返回json

@RequestMapping(value = "update",

produces = { "application/json;charset=utf-8" })

@ResponseBody

**public** JSONObject update(){

**int** flag = service.update(**null**);

**if**(flag == 1){

**return** **this**.responeJsonStr("1");

}**else**{

**return** **this**.responeJsonStr("0");

}

}

// 1-成功|0-失败

**public** JSONObject responeJsonStr(String status) {

JSONObject re = **new** JSONObject();

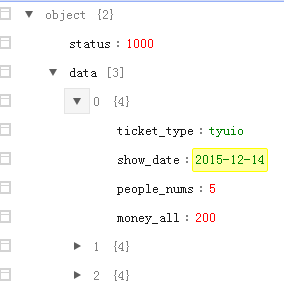
re.put("status", status);

**return** re;

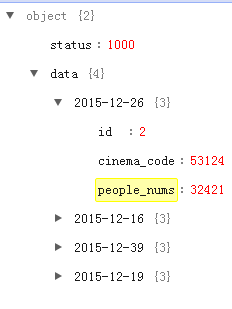
}

}

* 1. Json的结构原理



【图7.3.1】



【图7.3.1】

Json的结构分为2种，一种是以 { } 花括号分隔的，他想表现给阅读者的是对象，即：Object；另一种是以 [ ] 中括号分隔的，他想表现给阅读者的是数组，即Array。而Json结构的最小单位则是字段。对象中可以包含数组，数组中也可以包含对象。

通常JS解析Json数据有两种方法：JavaScript原生的eval()和JQuery的$.parseJSON()方法。容错率最高的是$.parseJSON()，推荐使用。

**function** stringToObject(json) {

**return** eval("(" + json + ")");

}

**var** obj = $.parseJSON('{"name": "字段"}');

代码段中的两个方法返回的都是JavaScript对象，当你打开游览器调试(F12开发人员选项)的时候，就能看到Json串被转化后的样子，和【图7.3.1】、【图7.3.2】结构几乎一模一样。当你在JS中写下这行代码的时候你就可以得到“字段”这个值：

**var** name\_ = obj.name; // 字段

**var** status\_ = obj.status; // 1000

然后配合for循环去操作Html的页面标签就可以方便的更新、删除、添加HTML元素，页面也就写活了。

大公司里常说的两个词：对接口和拆包，其实就是一种装逼的说法，说白了就是定义Json结构和拆解Json结构成形成对象。不知道的还以为多厉害呢。服务器A发出的Json串儿，用户浏览器可以接受，由JavaScript进行解析生成对象；但服务器B也可以接受，这就是俗称的调用接口(如安卓手机)。以Java类型的服务器为例，可以通过HttpClient调用服务器A的接口，如：http://www.json.com/10086.action 然后得到Json串，通过java来解析生成对象。Java解析Json串生成对象要比JavaScript复杂的多。如果接口提供者没有告诉你Json的生成规则，那么会非常考验解析者的Json功底。但高手一般都能猜对Json的Java反射结构。

【图7.3.1】的Java结构如下：

**import** java.util.Map;

**public** **class** RgcaModel {

**private** Integer status;

**private** Map<String,RgcaEntity> data;

**private** String errormessage;

// 省略get/set函数

}

**public** **class** RgcaEntity {

**private** Integer id;

**private** Integer cinema\_code;

**private** Integer people\_nums;

// 省略get/set函数

}

// json字符串反射生成对象

Gson gson = **new** Gson();

RgcaModel model =

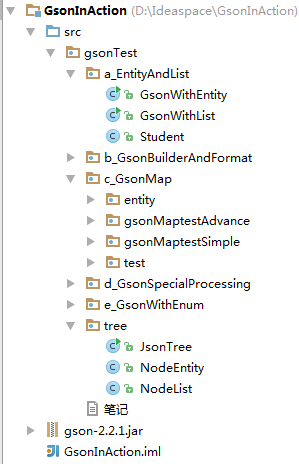
gson.fromJson(jsonStr,**new** TypeToken<RgcaModel>() {}.getType());

* 1. Json结构的Java序列化与反序列化

非常著名的两个开源项目：alibaba.fastjson 和 google.gson。这两个开源Jar包常常被用作项目中json字符串的序列化与反序列化，所谓序列化：Java对象(List|Map|实体类)生成Json字符串；反序列化：依据Json字符串，反射生成Java对象。

对于Json的生成、原理已经独立生成了一个IDEA项目，介绍的非常详细，永久链接如下：

链接：http://pan.baidu.com/s/1dDZ5EH7 密码：nard



没必要进行大篇幅介绍，直接看项目就可以了。

1. JQuery 基础
   1. 选择符
      1. 基本选择符

3种基本的选择符：标签名、ID和类。这些选择符可以单独使用，也可以与其他选择符组合使用。下表展示了这3种基本的选择符：

|  |  |  |  |
| --- | --- | --- | --- |
| **选择符** | **CSS** | **JQuery** | **说明** |
| 标签名 | P {} | $('p') | 取得文档中所有的段落<p> |
| ID | #some-id {} | $('#some-id') | 取得文档中ID为some-id的一个元素. |
| 类 | .some-class {} | $('.some-class') | 取得文档中类为some-class的所有元素 |

在将方法连缀到$()工厂函数后面时，包装在jQuery对象中的元素会被自动、隐式地循环遍历。换句话说，这样就避免了使用for循环之类的显式迭代 （这种迭代在DOM脚本编程中非常常见）。

* + 1. 子元素选择符：>

在基本选择符的基础上，JQuery提供了更多的“组合选择”方式，其中子元素选择符就是其中的一种。下面以列表Html元素的形式为例解释：

代码如下：

1. **<ul  id="selected-plays"**  class="clear-after"**>**
2. **<li>**喜剧
3. **<ul>**
4. **<li><a** href="/asyoulikeit/"**>**As You Like It**</a></li>**
5. **<li>**终成眷属 **</li>**
6. **<li>**仲夏夜之梦威廉·莎士比亚**</li>**
7. **<li>**Twelfth Night**</li>**
8. **</ul>**
9. **</li>**
10. **<li>**悲剧
11. **<ul>**
12. **<li><a** href="hamlet.pdf"**>**Hamlet**</a></li>**
13. **<li>**麦克白**</li>**
14. **<li>**罗密欧与朱丽叶**</li>**
15. **</ul>**
16. **</li>**
17. **<li>**历史
18. **<ul>**
19. **<li>**亨利四世 (**<a** href="mailto:henryiv@king.co.uk"**>**email**</a>**)
20. **<ul>**
21. **<li>**Part I**</li>**
22. **<li>**Part II**</li>**
23. **</ul>**
24. **</li>**
25. **<li><a** href="http://www.baidu.com"**>**Henry V**</a></li>**
26. **<li>**理查二世**</li>**
27. **</ul>**
28. **</li>**
29. **</ul>**

原始截图如下：



现在用子元素选择器实现这样一个效果：喜剧、悲剧和历史横过来，但他们下面的子元素依旧竖直排列。在代码中可以看到只有ul标签的 id="selected-plays"；其他标签都没有id标识和class标识。

首先，添加一个CSS类选择器：horizontal(水平的)，然后动态的将这个CSS类添加到顶级列表项喜剧、悲剧和历史中

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

**.horizontal {**

**float: left;**

**list-style: none;**

**margin: 10px;**

**}**

**</style>**

horizontal类会将元素浮动到它后面元素的左侧，如果这个元素是一个列表项，那么

会移除其项目符号，最后再为该元素的每一边各添加 10像素的外边距。

第二步：动态追加CSS类

**<script type="text/javascript">**

**$(document).ready(function()**

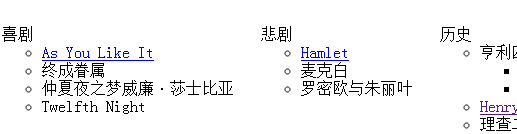
**{**

**$('#selected-plays > li').addClass('horizontal');**

**});**

**</script>**

页面显示结果如下：



【图-8.1.2】

* + 1. 否定式伪类选择符：:not()

**:not()** 是否定式伪类选择符。在上【图-8.1.2】的基础上，我们要为自列表添加一个灰色的背景。$('#selected-plays **li:not(.horizontal)**'). addClass('sub-level');

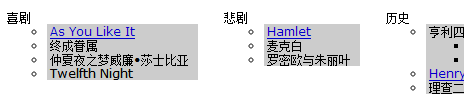
**目的**：取得符合条件的所有<li>标签；

**条件**：

1、ID为selected-plays的元素（#selected-plays）的后代元素。

2、class属性属性中没有horizontal类（ :not(.horizontal)）。

显示结果如下：



* 1. jQuery包裹被选Html元素 - wrapInner()

wrapInner() 方法使用指定的 HTML 内容或元素，来包裹每个被选元素中的所有内容 (inner HTML)。例如在每个 p 元素的内容上包围 b 元素：

$("p").wrapInner("<b></b>");

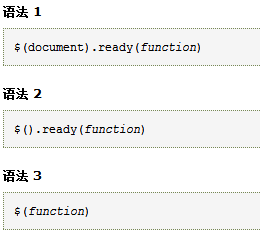
wrapInner()方法也有高级使用形式，如下：

$(selector).wrapInner(function()); function()必需，规定了返回包围元素的函数。

1. JQuery 高级特性
   1. JQuery事件

在JQuery中原生提供的所有事件函数都需要放在$(document).ready(function(){});方法体内，如果不在这个方法体中，这些事件函数是失效的。

$(document).ready(function(){})：当 DOM（文档对象模型） 已经加载，并且页面（包括图像）已经完全呈现时，会发生 ready 事件。由于该事件在文档就绪后发生，因此把所有其他的 jQuery 事件和函数置于该事件中是**正确**的做法。其三种语法如下：



|  |  |
| --- | --- |
| **参数** | **描述** |
| function | 必需。规定当文档加载后要运行的函数。 |

所有JQuery支持的事件如下：

http://www.w3school.com.cn/jquery/jquery\_ref\_events.asp

**提示和注释**

ready() 函数不应与 <body onload=""> 一起使用。

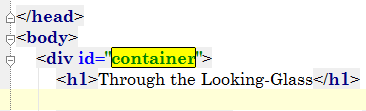
* 1. JQuery与CSS样式

JQuery通过 .addClass() 和 .removeClass()两个方法来针对Html元素来操作CSS样式。如：$('selecter').addClass('css-class-name');

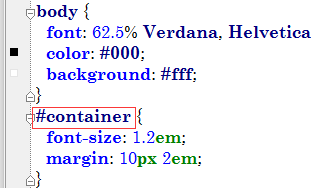
* + 1. CSS与Hml的结合方式

普通渲染

这是一种最常见的CSS渲染Html文档元素的方式。CSS通过被渲染Html元素的id、class和元素标签等方式来为一个Html标签添加CSS样式。如下图所示：



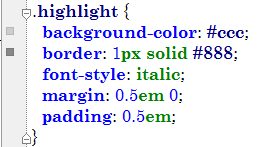
CSS渲染id =’container’的div标签：



这种方式当Html文档和CSS文件加载完毕后，页面就会按照CSS文件的预期定义显示出来。

* CSS类渲染

CSS类渲染是向指定的Html文档元素的class属性中添加一个\*\*\*.css文件中定义的Css类。\*\*\*.css必须已经在页面引入。首先在CSS文件中定义一个CSS类：



以英文的“.”开头，加一个样式类的名称，大括号中是样式的代码。

JQuery代码如下：

**$(document).ready(function()**

**{**

**$("#but\_1").click(function()**

**{**

**$('div.poem-stanza').addClass('highlight');**

**});**

**$("#but\_2").click(function()**

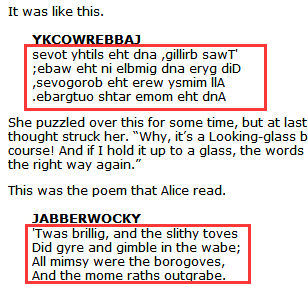
**{**

**$('div.poem-stanza').removeClass('highlight');**

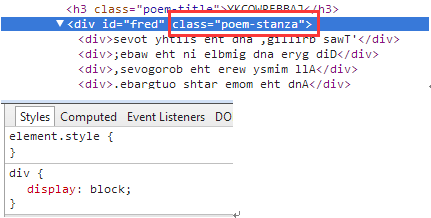
**});**

**});**

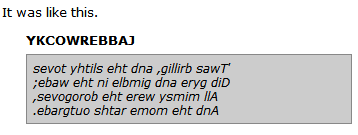
两个按钮分别对应添加和移除样式类。Html原始样式如下：

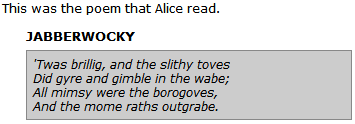
可见红框内无样式。

浏览器调试显示如下：此时没有显示追加的CSS定义的highlight类。

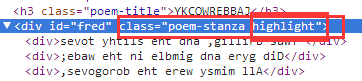


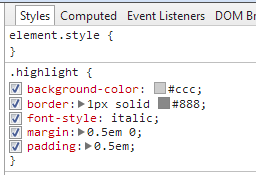
现在追加定义的highlight类：





可见样式已经被渲染在标签中。现在查看浏览器的Developer Tools中的Elements，截图如下：





* + 1. CSS类渲染实例

Index.html脚本如下：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Jquery in Action</title>

<link rel="stylesheet" href="01.css" type="text/css"/>

<script src="jquery.js"></script>

<script src="01.js"></script>

</head>

<body>

<div id="container">

<h1>Through the Looking-Glass</h1>

<div id="div\_author" class="author">by Lewis Carroll</div>

<div class="chapter" id="chapter-1">

<h2 class="chapter-title">1. Looking-Glass House</h2>

<p>There was a book lying near Alice on the table, and while she sat watching the White King</p>

<p>It was like this.</p>

<div class="poem">

<h3 class="poem-title">Archetype</h3>

<div id="fred" class="poem-stanza">

<div>Sevot yhtils eht dna ,gillirb sawT'</div>

<div>Ebaw eht ni elbmig dna eryg diD</div>

</div>

</div>

<p>She puzzled over this for some time, but at last a bright thought struck her.</p>

</div>

</div>

<input id="but\_1" type="button" value=" add-css" style="margin-right: 20px">

<input id="but\_2" type="button" value=" remove-css" style="margin-right: 20px">

<input id="but\_refresh\_1" type="button" value=" refresh" onclick="refreshPage()">

</body>

</html>

CSS文件脚本如下：

html, body {

margin: 0;

padding: 0;

}

body {

font: 62.5% Verdana, Helvetica, Arial, sans-serif;

color: #000;

background: #fff;

}

#container {

font-size: 1.2em;

margin: 10px 2em;

}

h1 {

font-size: 2.5em;

margin-bottom: 0;

}

h2 {

font-size: 1.3em;

margin-bottom: .5em;

}

h3 {

font-size: 1.1em;

margin-bottom: 0;

}

code {

font-size: 1.2em;

}

a {

color: #06581f;

}

.highlight {

background-color: #ccc;

border: 1px solid #888;

font-style: italic;

margin: 0.5em 0;

padding: 0.5em;

}

JavaScript脚本如下：

$(document).ready(function()

{

$("#but\_1").click(function()

{

$('div.poem-stanza').**addClass**('highlight');

});

$("#but\_2").click(function()

{

$('div.poem-stanza').**removeClass**('highlight');

});

});

function refreshPage()

{

window.location.reload();

}

1. JQuery 精深

这部分内容描述了JQuery在实际项目中的灵活应用。

* 1. 精确遍历结果集

jQuery提供过滤器，可以缩小结果集：

1. $('div').has('p');  //选择包含p元素的div元素
2. $('div').not('.myClass');  //选择class不等于myClass的div元素
3. $('div').filter('.myClass'); //选择class等于myClass的div元素
4. $('div').first();  //选择第1个div元素
5. $('div').eq(5);  //选择第6个div元素
   * 1. JQuery - has()

jQuery遍历的应用中has()方法用于检测B元素是否在A元素中。并返回所有包含B元素的A元素数组。测试代码如下：

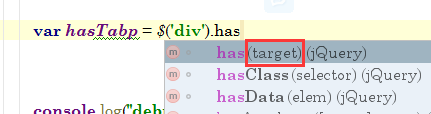
<**body**>   
 <**div id="id-1 " class="contain-span"**>  
 <**span**>this div contains tag span</**span**>  
 </**div**>  
 <**div class="contain-p"**>  
 <**p**>this div contains tag p 1</**p**>  
 </**div**>   
 <**div class="contain-p"**>  
 <**p**>this div contains tag p 2</**p**>  
 </**div**>  
</**body**>

【测试代码10.1.1】

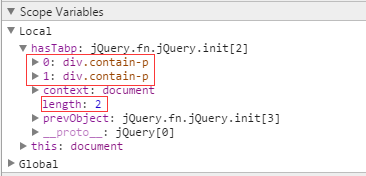
脚本代码如下：

**var** hasTabp = *$*(**'div'**).has(**'p'**);

在IDEA的代码提示中显示，has()方法的参数是一个html标签对象



浏览器控制台显示信息如下：



通常当我们选择到了这样一个结果集以后，会为这个结果集做一些其他的操作，比如为这个结果集添加一个额外的样式：

$(**'div'**).has(**'p'**).addClass(**"new-contain-p"**);

* + 1. JQuery – not()

从匹配元素集合中删除元素。该方法操作的是被选元素集合本身，而不涉及其父子节点。

语法一 .not(selector)

|  |  |
| --- | --- |
| **参数** | **描述** |
| selector | 字符串值，包含用于匹配元素的选择器表达式。 |

$('div').not('.contain-p');  //选择class不等于contain-p的div元素

语法二 .not(element)

|  |  |
| --- | --- |
| **参数** | **描述** |
| element | 一个或多个需要从匹配集中删除的 DOM 元素。 |

$('li').not(':even');  //选择偶数行的li元素，并返回该集合

语法三 .not(function(index))

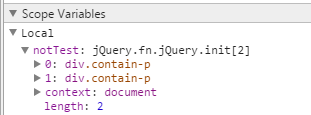
|  |  |
| --- | --- |
| **参数** | **描述** |
| function(index) | 字符串值，包含用于匹配元素的选择器表达式。 |

这里依从 测试代码10.1.1 ，针对语法三的示例代码如下：

**function** *getElement*()  
{  
 **var** theEle = **document**.getElementById(**"id-1"**);  
 **return** theEle;  
}

**var** notTest = *$*(**'div'**).not(*getElement*());

控制台跟踪显示结果如下：



语法三的做法并不常用。

* + 1. filter()过滤

filter() 方法将匹配元素集合缩减为匹配指定选择器的元素。

基本使用方法：

$('div').filter('.myClass'); //选择class等于myClass的div元素

filter方法的典型使用方法是选择单双行，包括ul标签下的li和table下的tr等等：

$(**'li'**).filter(**':even'**).css(**'background-color'**, **'red'**);

filter()与内置函数

使用该方法的第二个形式是，通过函数而不是选择器来筛选元素。对于每个元素，如果该函数返回 true，则元素会被包含在已筛选集合中；否则，会排除这个元素。

下面对这个内容进行系统分析，包括其每个参数及意义。首先html代码如下：

<**ul**>  
 <**li id="li-1" class="li-class-1"**> <**strong id="s-1"**>list</**strong**> item 1 </**li**>

<**li id="li-1-1" class="li-class-1"**> <**strong id="s-1-1"**>list</**strong**> item 1-1 </**li**>  
 <**li id="li-2"**>   
 <**strong id="s-2"**>list</**strong**> item   
 <**strong id="s-3"**>2</**strong**> two   
 <**span**>strong tags</**span**>   
 </**li**>  
 <**li id="li-3"**>list item 3</**li**>   
 <**li id="li-4"**>list item 4</**li**>   
 <**li id="li-5"**>list item 5</**li**>   
 <**li id="li-6"**>list item 6</**li**>   
</**ul**>

【测试代码10.1.3】

JavaScript测试脚本如下：

$(**'li'**).filter(  
 **function**(index , thisTag) {  
 **var** aa\_1 = $(**'strong'**); // 将取出一个全集 length=3  
 **var** aa\_2 = $(**'strong'**, **this**); // 将取出<li>下的子集length=1 or 2 or 3..  
 **var** bb = thisTag; // 当前这个<li>对象  
 **var** cc = index; // 此次遍历的索引，从0开始  
  
 // return将返回有2个<strong>的<li>对象  
 **return** $(**'strong'**, **this**).**length** == 2;  
 // 这里**this**代表当前正在遍历的<li>标签  
 }  
).css(**'background-color'**, **'red'**);

OK，开启万能的控制台，准备断点调试。



**var** aa\_1 = *$*(**'strong'**); // 将取出一个全集 length=3  
**var** aa\_2 = *$*(**'strong'**, **this**); // 将取出<li>下的子集length=1 or 2 or 3..

在Scope中可以看到变量aa\_1取到了4个strong元素，他的数组长度是4. aa\_2取到了一个元素，他的数组长度是1。这里this变量提示此时所遍历的<li>是id为li-1的元素，在反过来看这个元素：

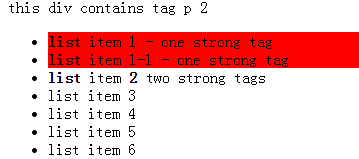
<**li id="li-1"**> <**strong id="s-1"**>list</**strong**> item 1 - one strong tag</**li**>

他的确值包含了一个<strong>元素。

其实**var** aa\_2 = *$*(**'strong'**, **this**);中的**this**指代的是此时正在遍历的<li>标签，不是什么全局变量之类的东西。

**function**(index , thisTag)会检查、过滤每一个<li>标签下满足条件的情况，并且返回给filter()，让filter()形成数组。

如果return $('strong', this).length == 1; 那么就会变成下面的样子：



这是一种高级应用。

* + 1. first()

first() 将匹配元素集合缩减为集合中的第一个元素。示例代码如下：

$('div').first(); //选择第1个div元素

**提示：**

$("p:first") 表示第一个P

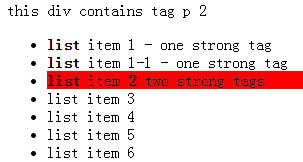
$("p").first() 将匹配元素集合缩减为集合中的第一个元素。

* + 1. eq(index)

eq()方法将匹配元素集合缩减为位于指定**索引**的新元素。如下：

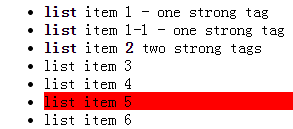
$('div').eq(5);  //选择第6个div元素

参照 测试代码10.1.3，$('li').eq(2).css('background-color', 'red'); 截图如下：



如果index提供负数，则指示从集合结尾开始的位置，而不是从开头开始。如：

$('li').eq(-2).css('background-color', 'red');



如果无法根据指定的 index 参数找到元素，则该方法构造带有空集的 jQuery 对象，length 属性为 0。

* + 1. jQuery 遍历 - each()

each() 方法规定 为每个匹配的元素执行用户指定运行的函数。返回 false 可用于及早停止循环。其方法格式如下：

$(selector).each(function(index , element))

其中function(index , element) 是必须的内容，他为每个匹配元素规定了运行的函数。其内置参数含义如下：

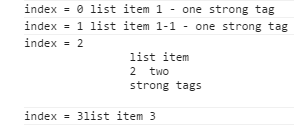
index：选择器的index 位置

element：当前的元素（也可使用 "this" 选择器）

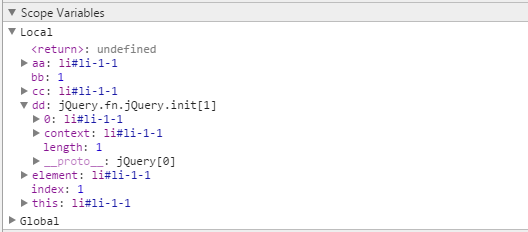
依托 测试代码10.1.3，完整的功能测试脚本如下：

**function** outputText(){   
 *$*(**"li"**).each(   
 **function**(index , element){   
 **var** aa = **this**;   
 **var** bb = index;   
 **var** cc = element;   
 **var** dd = $(**this**);   
  
 **console**.log(**"index = "** + index + $(**this**).text());   
  
 **if**(index == 3){   
 **return false**;   
 }   
 }   
 );   
}

在这段代码中，当遍历索引为3的<li>时候则跳出遍历，控制台将打印出0~3 一共4条记录。控制台截图如下：



当尝试控制台中跟踪的时候观察到this、index、element和$(this)的值的变化情况如下图所示，下图截取了Scope Variables：



aa = **this**; bb = index; cc = element; dd = $(**this**); 可以看到**this**和element指代相同内容。

* + 1. jQuery 遍历 - find()

获得当前匹配元素集合中每个元素的后代，由选择器进行筛选。

**语法**

.find(selector)

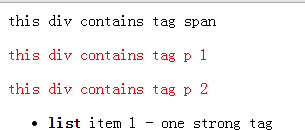
|  |  |
| --- | --- |
| **参数** | **描述** |
| selector | 字符串值，包含供匹配当前元素集合的选择器表达式。 |

**参数**

**举例**

$(**"div"**).find(**"span"**).css(**'color'**,**'red'**);   
$(**"body"**).find(**".contain-p"**).css(**'color'**,**'red'**);

效果如下：



* + 1. jQuery 遍历 - slice()

将匹配元素集合缩减为指定范围的子集。

.slice(selector,end)

|  |  |
| --- | --- |
| **参数** | **描述** |
| selector | 基于 0 的整数值，指示开始选取元素的位置。  如果是负数，则指示从集合末端开始的偏移量。 |
| end | 基于 0 的整数值，指示结束选取元素的位置。  如果是负数，则指示从集合末端开始的偏移量。  如果省略，则选取范围会在集合末端结束。 |

用法如下：

$(**'li'**).slice(2, 4).css(**'background-color'**, **'red'**);

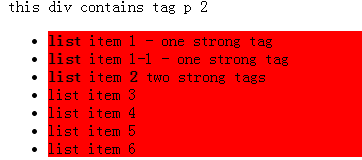
感觉这个方法比较鸡肋，目前所及用的很少。

* + 1. jQuery 遍历 - siblings()

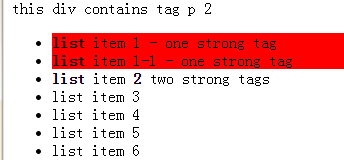
siblings() 获得匹配集合中每个元素的同胞节点，返回新的JQuery对象数组集合。该方法接受可选的选择器表达式，与我们向 $() 函数中传递的参数类型相同。如果应用这个选择器，则将通过检测元素是否匹配该选择器对元素进行筛选。

依托 测试代码10.1.3，示例使用对比如下：

$(**'li'**).siblings().css(**'background-color'**, **'red'**);



$(**'li'**).siblings(**'.li-class-1'**).css(**'background-color'**, **'red'**);



* + 1. jQuery add() 方法

add() 方法将元素添加到匹配元素的集合中。感觉w3c说的这句话比较扯淡，过于泛泛。依托 测试代码10.1.3，重新解释这个方法的意思：

首先我们想给<ul>标签内容添加背景色，则选中一个对象集合ul：$(**"ul"**) 。

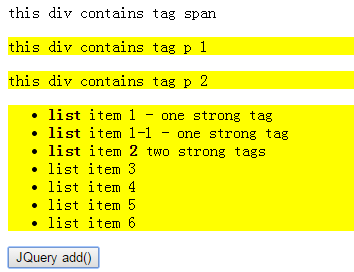
**var** aa = $(**"ul"**).css(**"background-color"**,**"yellow"**);

可是我们还想同时给所有的<p>标签也添加同样的背景色，则需要$(**"p"**) 。

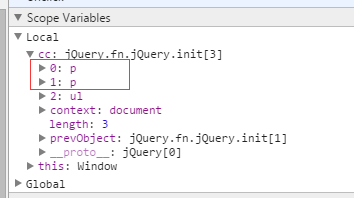
**var** bb = $(**"p"**).css(**"background-color"**,**"yellow"**);

如果使用add()方法那么可以这样做：

**var** cc = $(**"ul"**).add(**"p"**).css(**"background-color"**,**"yellow"**);



跟踪**var** cc = $(**"ul"**).add(**"p"**); 的断点截图如下：



add()：将新的元素 添加到已存在的JQuery对象中。

* + 1. prev() and prevAll()

prev() 获得匹配元素集合中每个元素紧邻的前一个同胞元素，通过选择器进行筛选是可选的。prevAll() 获得当前匹配元素集合中每个元素的前面的同胞元素，使用选择器进行筛选是可选的。没什么可废话的，创建一个新的页面，其脚本如下：

**CSS：**

<**style type="text/css"**>   
 .**prevss**{  
 **background**: **greenyellow**;  
 }   
</**style**>

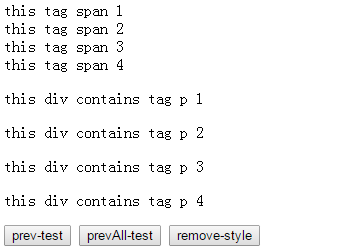
**Html：**

<**body**>  
 <**div class="contain-p"**>  
 <**span**>this tag span 1</**span**></**br**>  
 <**span**>this tag span 2</**span**></**br**>  
 <**span**>this tag span 3</**span**></**br**>  
 <**span**>this tag span 4</**span**></**br**>

<**p**>this div contains tag p 1</**p**>  
 <**p**>this div contains tag p 2</**p**>  
 <**p id="p-3"**>this div contains tag p 3</**p**>  
 <**p**>this div contains tag p 4</**p**>  
 </**div**>   
 <**button id="button-1"**> prev-test </**button**>  
 <**button id="button-2"**> prevAll-test </**button**>  
 <**button id="button-3"**> remove-style </**button**>  
</**body**>

【测试代码10.1.11-1】

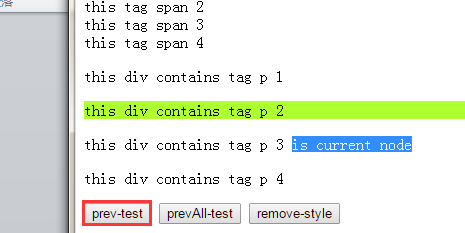
**默认页面截图：**

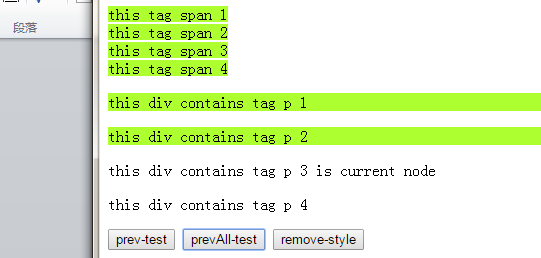


**Js脚本：**

$(**function**(){  
 $(**'#button-1'**).click(**function**(e){   
 $(**"#p-3"**).html(**'this div contains tag p 3'**);   
 $(**"#p-3"**).prev().addClass(**"prevss"**);   
 $(**"#p-3"**).html($(**"#p-3"**).html() + **' is current node'**);   
 });   
 $(**'#button-2'**).click(**function**(e){   
 $(**"#p-3"**).html(**'this div contains tag p 3'**);   
 $(**"#p-3"**).prevAll().addClass(**"prevss"**);   
 $(**"#p-3"**).html($(**"#p-3"**).html() + **' is current node'**);   
 });  
 $(**'#button-3'**).click(**function**(e){   
 $(**"p"**).removeClass(**"prevss"**);   
 $(**"span"**).removeClass(**"prevss"**);   
 $(**"#p-3"**).html(**'this div contains tag p 3'**);   
 });   
});

首先测试prev()方法，结果如下图所示：





然后测试prevAll()方法，测试结果如上图所示。很明显就这么点儿区别。

* 1. DOM树上的jquery对象移动

有时候，我们需要从结果集出发，移动到附近的相关元素。jQuery也提供了在DOM树上的移动方法。

1. $('div').next('p');  //选择div元素后面的第一个p元素
2. $('div').parent();  //选择div元素的父元素
3. $('div').closest('form');  //选择离div最近的那个form父元素
4. $('div').children();  //选择div的所有子元素
5. $('div').siblings();  //选择div的同级元素
   1. 链式操作-JQuery连缀

选中网页元素以后，就可以对它进行某种操作。jQuery允许将所有操作连接在一起，以链条的形式写出来，如：

$('div').find('h3').eq(2).html('Hello');

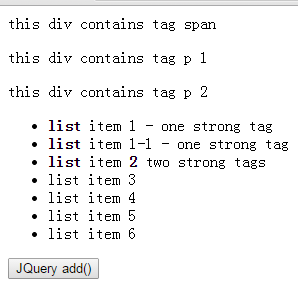
分解开来，就是下面这样：

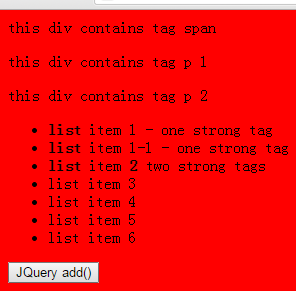
1. $('div') //找到div元素
2. .find('h3') //选择其中的h3元素
3. .eq(2) //选择第3个h3元素
4. .html('Hello');  //将它的内容改为Hello

这是jQuery最令人称道、最方便的特点。它的原理在于每一步的jQuery操作，返回的都是一个jQuery对象，所以不同操作可以连在一起。

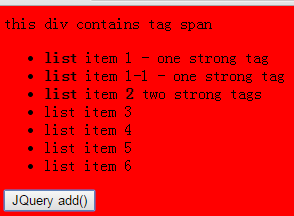
* 1. 高级事件操作
     1. 多个事件绑定同一个函数

$(**function**(){   
 $(**"button"**).bind({  
 click: **function**(){  
 $(**"p"**).**slideToggle**();  
 },  
 mouseover: **function**(){  
 $(**"body"**).css(**"background-color"**,**"red"**);  
 },  
 mouseout: **function**(){  
 $(**"body"**).css(**"background-color"**,**"#FFFFFF"**);  
 }  
 });   
});



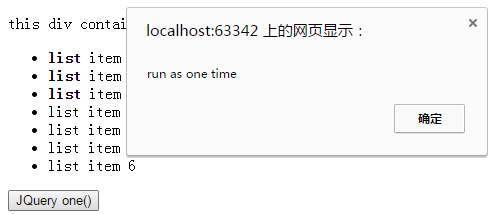


点击按钮后，隐藏所有的<p>标签内容



* + 1. 仅让事件运行一次

*$*(**function**(){  
 *$*(**"button"**).one(**"click"**, **function**(){  
 alert(**"run as one time"**); // 只运行一次，以后的点击不会运行  
 });  
});



再次点击按钮也不会弹出对话框。

* + 1. 解除事件绑定

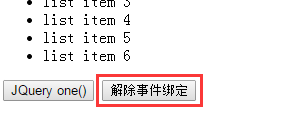
依据10.4.2中的js代码，下面在页面中添加一个按钮，通过这个按钮解除10.4.2中的单击事件绑定。

<**input type="button" value="解除事件绑定" onclick="***unbindClick*()**"**>

js脚本如下

**function** unbindClick()  
{  
 $(**'button'**).unbind(**'click'**);  
}

如下图：

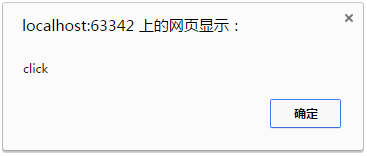


如过先点击右侧按钮，则左侧按钮的单击事件就会失效。

* 1. 事件处理深入：事件对象(event object)

所有的事件处理函数，都可以接受一个事件对象(event object)作为参数，比如下面例子中的e：

*$*(**function**(){  
 *$*(**"button"**).click(**function**(e){  
 **var** aa = e;  
 e.**altKey** = **true**;  
 alert(e.**type**);  
 });  
});



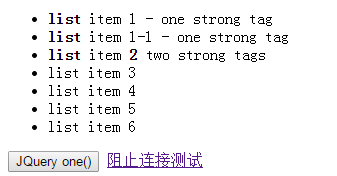
* + 1. 事件对象很有用的属性和方法：

|  |  |
| --- | --- |
| **属性 / 方法** | **方法描述** |
| event.pageX | 事件发生时，鼠标距离网页左上角的水平距离 |
| event.pageY | 事件发生时，鼠标距离网页右上角的垂直距离 |
| event.type | 事件的类型(比如click) |
| event.which | 鼠标单击事件中获取到鼠标的左、中、右键；在键盘事件中获取键盘的按钮；1 = 鼠标左键；2 = 鼠标中键；3 = 鼠标右键。 |
| event.data | 在事件对象上绑定数据，然后传入事件处理函数 |
| event.target | 事件针对的网页元素 |
| event.preventDefault() | 阻止事件的默认行为(比如点击链接，会自动打开新页面) |
| event.stopPropagation() | 停止事件向上层元素冒泡 |
| event.metaKey | 针对不同浏览器对键盘中的<ctrl>按键解释不同，jQuery也进行了封装，并规定event.metaKey()方法为键盘事件中获取<ctrl>按键。 |

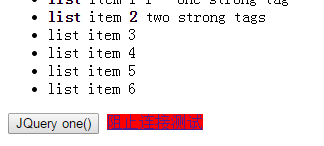
* + 1. 事件对象应用

在事件处理函数中，可以用this关键字，返回事件针对的DOM元素：

$(**function**(){  
 $(**'a'**).click(**function**(e){  
 **var** evilLink = **'http://www.baidu.com'**;  
 **if** ($(**this**).attr(**'href'**).match(evilLink)) // 如果确认为有害链接  
 {  
 e.preventDefault(); // 阻止打开  
 $(**this**).addClass(**'evil'**); // 加上表示有害的class  
 }  
 });  
});



点击【阻止连接测试】后，如下图：



* 1. 嵌套的过滤器

1. //允许你减少集合中的匹配元素的过滤器，
2. //只剩下那些与给定的选择器匹配的部分。在这种情况下，
3. //查询删除了任何没（:not）有（:has）
4. //包含class为“selected”（.selected）的子节点。
5. .filter(":not(:has(.selected))")
   1. 切换Html标签的类 - toggleClass()

该方法检查每个元素中指定的类。如果不存在则添加类，如果已设置则删除之。这就是所谓的切换效果。如果不使用toggleClass()方法，那么需要这样写：

$(**'li'**).hasClass(**'li-class-1'**) ? $(**'li'**).removeClass(**'li-class-1'**) : $(**'li'**).addClass(**'evil'**);

或者这么写：

**if**($(**'li'**).hasClass(**'li-class-1'**))  
{  
 $(**'li'**).removeClass(**'li-class-1'**);

$(**'li'**).addClass(**'evil'**);  
}**else**{  
 $(**'li'**).addClass(**'evil'**);  
}

如果使用toggleClass()方法，明显就简单多了：

$(**'button'**).click(**function**(e){  
 $(**'li'**).toggleClass(**'evil'**);  
});

**toggleClass()高级特性**

toggleClass()还支持更高级的特性，如下：

$(selector).toggleClass( function(index,class) , switch);

|  |  |
| --- | --- |
| **参数** | **参数描述** |
| function(index,class) | 必需。规定返回需要添加或删除的一个或多个类名的函数。  index - 可选。接受选择器的 index 位置。  class - 可选。接受选择器的当前的类。 |
| switch | 可选。布尔值。规定是否添加(true)或移除(false)类。 |

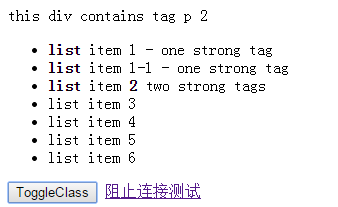
在CSS脚本中添加如下2个类：

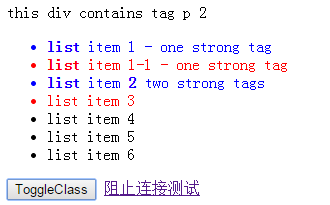
.**listitem\_1**, .**listitem\_3**{**color**:**red**;}  
.**listitem\_0**, .**listitem\_2**{**color**:**blue**;}

测试JS脚本如下：

$(**'button'**).click(**function**(e){  
 $(**'ul li'**).toggleClass(  
 **function**(){  
 **return 'listitem\_'** + $(**this**).index();  
 });  
});

运行截图如下：

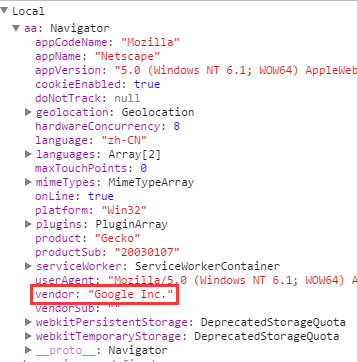




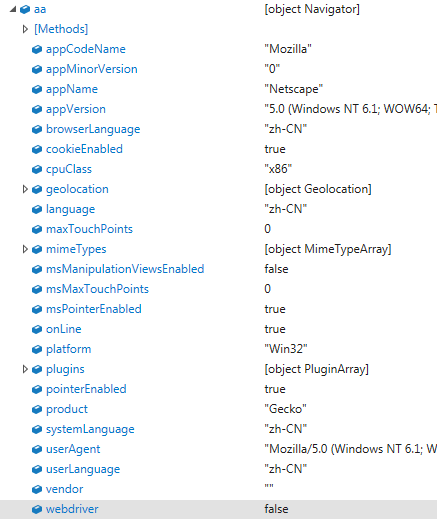
当再次点击按钮，颜色会消失。

* 1. 读取浏览器相关信息：navigator

所有谷歌浏览器信息如下：



所有IE浏览器信息如下，注意IE11的vendor属性是特么空的。



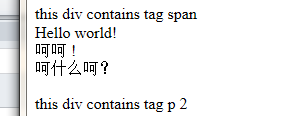
这部分内容其实是JavaScript中内容。

* 1. Html文档替换replaceWith()
     1. 基本操作

replaceWith() 用指定的 HTML内容或元素 替换 被选元素。

$(**'button'**).click(**function**(e){   
 $(**'div'**).eq(1).find(**'p'**).replaceWith(   
 **'<span> Hello world! </span><br/>'** + **'<span> 呵呵 !</span><br/>'** +**'<span> 呵什么呵？</span><br/>'** );   
});

截图如下：



相比较删除一个节点在去追加一个新的节点，这种方法显得更加简洁。

* + 1. 高级回调

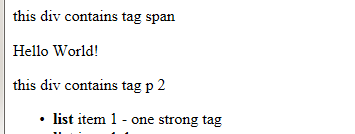
replaceWith()还有更高级的操作，他支持function回调，语法如下：

$(selector).replaceWith(function()); function() 返回待替换被选元素的新内容的函数。

具体应用示例如下：

$(**'button'**).click(**function**(e){  
 $(**'div'**).eq(1).find(**'p'**).replaceWith(   
 **function**(){   
 /\*\*   
 \* anything you wanna to do   
 \*/  
 **return "<p>Hello World!</p>"**;   
 }   
 );   
});

截图如下：



* 1. 从一个未排序的集合中找出某个元素的索引号

利用prevAll()方法。代码实现如下：

$(**function**(){   
 $(**"ul > li"**).click(**function** (){  
 **var** index = $(**this**).prevAll().**length**;  
 **console**.log(**"index = "** + index);  
 });   
});

prevAll() 获得当前匹配元素集合中每个元素的前面的同胞元素。他的详细说明在10.1.11。

* 1. 多属性过滤 – 复杂选择器 - input

在使用许多相类似的有着不同类型的input元素时，这种基于精确度的方法很有用。

**CSS：**

<**style type="text/css"**>  
 .**div-1**{  
 **margin-bottom**: 10**px**;  
  
 }   
 .**input-text**{  
 **margin-bottom**: 2**px**;  
 }   
</**style**>

**Html：**

<**body**>   
 <**div id="div-1" class="div-1"** >  
 <**input type="text" name="accept text" value="精确度" class="input-test input-text"** ></**br**>   
 <**input type="text" value="精确度" class="input-test input-text"**></**br**>  
 <**input type="text" value="预加载" class="input-test input-text"**></**br**>  
 <**input type="text" value="预加载" class="input-test input-text"**></**br**>  
 </**div**>   
 <**div id="div-2" class="div-1"** >  
 <**input type="button" value="button-1" class="input-test input-button"**>  
 <**input type="button" value="button-2" class="input-test input-button"**>  
 <**input type="button" name="accept" value="button-3" class="input-test input-button"**>  
 <**input type="button" value="button-4" class="input-test input-button"**>  
 </**div**>   
 <**div id="div-3" class="div-1"** >  
 <**input type="checkbox" value="" class="input-test input-checkbox"**>  
 <**input type="checkbox" name="accept" value="" class="input-test input-checkbox"**>  
 <**input type="checkbox" value="" class="input-test input-checkbox"**>  
 <**input type="checkbox" value="" class="input-test input-checkbox"**>  
 </**div**>   
 <**div id="div-4" class="div-1"** >  
 <**input type="radio" name="accept" value="" class="input-test input-radio"**>  
 <**input type="radio" value="" class="input-test input-radio"**>  
 <**input type="radio" value="" class="input-test input-radio"**>  
 <**input type="radio" value="" class="input-test input-radio"**>  
 </**div**>   
 <**div id="div-5" class="div-1"** >  
 <**input type="file" name="accept" value="" class="input-test input-file"**></**br**>  
 <**input type="file" value="" class="input-test input-file"**></**br**>  
 <**input type="file" value="" class="input-test input-file"**></**br**>  
 <**input type="file" value="" class="input-test input-file"**></**br**>  
 </**div**>   
 <**button id="button-1"**> 多属性过滤 </**button**>   
</**body**>

**JavaScript：**

$(**function**(){   
 $(**'#button-1'**).click(**function**(e){   
 // 属性等于过滤器|复合属性过滤器

// $("selector[selector1][selector2]...[selector[N]")  
 **var** aa = $(**".input-test[name=accept]"**);  
 **var** bb = $(**'input[type=text][value=预加载]'**);  
 **var** cc = $(**'.input-text[type="text"][value="精确度"]'**);  
  
 // 属性包含单词过滤器  
 **var** dd = $(**"input[name~='text']"**); // name="accept text"  
  
 // 属性不等于过滤器  
 **var** ee = $(**".input-text[name!='accept text']"**);   
 });   
});

上面的JavaScript脚本提供了使用范围最大的三种情况，语法结构照搬即可。

* 1. 图像预加载 / 延迟加载

老技术，简单看看实现方法及简单的说明：

**var** img = *$*(**'<img />'**).attr(**'src'**, **'imageurl.jpg'**);

jQuery创建了一个图片元素并设置了它的地址，如果将它放在document ready中处理时，当页面加载时就会告诉浏览器去加载指定的图片。当然大多数情况我们不会这么做，否则会出现页面一直在等待图片加载的现象。而当用户在刚打开某个图片专辑的时候，我们可以将要预加载的图片进行这样的处理，因为我们可以预见用户继续浏览下张图片的可能性很大。我们更可以根据图片的命名规则一次预加载特定几张图片，而不是一下子把所需图片都加载下来，网上看到这么段小代码：

$.preloadImages = **function**() {   
 **for** (**var** i = 0; i < arguments.**length**; i++) {   
 $(**"<img />"**).attr(**"src"**, arguments[i]);   
 }   
}   
$.preloadImages(**"image1.jpg"**,**"image2.jpg"**);

最后我们使用append或者appendTo方法将图片放置于我们指定的元素中即可：

$(**'#container'**).append(img);

这个东西在实际中还没用过，他适用于页面中包含大量图片的情况，如瀑布流这种布局。上面的内容博客地址如下：

http://www.jquery001.com/jquery-image-preload.html

* 1. jQuery扩展自定义方法
     1. jQuery.extend(object);

为扩展jQuery类本身，为类添加新的方法。可以理解为添加静态方法。

示例如下：

$.extend({  
 extendAdd:**function**(a , b){   
 **var** c = a + b;   
 alert(c);   
 }  
});

调用：

$.extendAdd(20,32);

* + 1. jQuery.fn.extend(object);

jQuery.fn.extend(object); 对jQuery.prototype进得扩展，就是为jQuery类添加“成员函数”。jQuery类的实例可以使用这个“成员函数”。比如我们要开发一个插件，做一个特殊的编辑框，当它被点击时，便alert 当前编辑框里的内容。

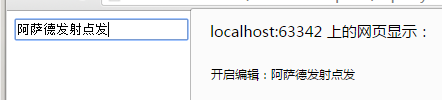
示例如下：

$.**fn**.extend({  
 alertEditClick : **function**(){  
 $(**this**).click(**function**(){  
 **if**($.**trim**($(**this**).val()).**length** > 0)  
 {  
 alert(**'开启编辑：'** + $(**this**).val());  
 }  
 });  
 }  
});

调用：

*$*(**function**(){  
 *$*(**"#input-test"**).alertEditClick();  
});

如图：



* 1. 禁用网页右键菜单 与 自定义页面右键菜单

$(**function**(){   
 $(**document**).bind(**'contextmenu'**,**function**(e){   
 **return false**;   
 });   
});

加入这段代码页面则无法开启鼠标右键。但是通过浏览器定义的组合快捷键还是能够使用右键中的内容，比如查看源代码和审查元素等等。

$(**document**).bind(**'contextmenu'**,**function**(e){}); 是为页面绑定右键行为，如果你的页面中有特殊的需求，需要添加右键操作那么应该扩展这段代码。

* 1. 定制的Jquery选择器

/\*\*  
 \* element- 一个DOM元素  
 \* index – 栈中的当前循环索引  
 \* meta – 有关选择器的元数据  
 \* stack – 要循环的所有元素的栈  
 \* 如果包含了当前元素就返回true  
 \* 如果不包含当前元素就返回false };  
 \*/  
$.**expr**[**':'**].mycustomselector = **function**(element, index, meta, stack){  
   
// 定制选择器的用法：  
$(**'.someClasses:test'**).doSomething();

* 1. 文本类型的input域中保留一个默认值

这段代码展示了在用户未输入值时，如何在文本类型的input域中保留一个默认值

$(**".swap"**).each(**function**(i){   
 **var** swap\_val38 = $(**this**).val();   
 $(**this**).focusin(**function**(){  
 **if** ($(**this**).val() == swap\_val38) {  
 $(**this**).val(**""**);  
 }  
 }).focusout(**function**(){  
 **if** ($.**trim**($(**this**).val()) == **""**) {  
 $(**this**).val(swap\_val38);  
 }  
 });  
});

暂时没用到，不明觉厉，先保存。

* 1. DDDDDDDDD
  2. DDDDDDDDD
  3. DDDDDDDDD
  4. DDDDDDDDD
  5. DDDDDDDDD

代码容器样式

代码容器样式 - 1

代码容器样式 - 1

代码容器样式 - 2

<script type="text/javascript">

//定义一个方法

function say(name,age)

{

alert(name+"今年"+age+"岁了");

}

//调用say方法

say("张三",3);

</script>

代码容器样式 - 3

<script type="text/javascript">

//定义一个方法

function say(name,age)

{

alert(name+"今年"+age+"岁了");

}

//调用say方法

say("张三",3);

</script>

代码容器样式 - 4

<script type="text/javascript">

//定义一个方法

function say(name,age)

{

alert(name+"今年"+age+"岁了");

}

//调用say方法

say("张三",3);

</script>

代码容器样式 - 5