HTML

教程：菜鸟学习网

HTML： **H**yper**T**ext **M**arkup **L**anguage，超文本**标记语言**，用于编写网页。

HTML 元素以**开始标签** `<label>` 起，素以**结束标签** `</label>` 终止。

元素的内容介于开始标签与结束标签之间 `<label>content</label>`，**元素属性**在开始标签以键值对的形式设定，属性值以（双）引号包裹 `<label attr='val'>content</label>`

文档结构

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>菜鸟教程(runoob.com)</title>
        <style type='text/css'></style>
        <script></script>
    </head>

    <body>
        hello world
    </body>
</html>
```

文档声明+根+头+体

- 文档声明：`!DOCTYPE html`：声明本文档为Html4文档，不区分大小写
- 根标签：`<html>`：网页的根标签
- 头：`<head>`，定义html的元数据部分，在其中可以插入脚本（scripts）、样式文件（CSS）等。
  - title：网页在工具栏、收藏夹显示的标题，**是必须具备的标签**
  - meta：声明字符编码
- 体：`<body>`，**定义可见的页面内容**

注释

```
<!--这是一个注释-->
```

标题 段落

- 文本处理：关于其他文本格式信息可以参考本链接
- 标题级别：h1-6，浏览器会自动地在标题的前后添加空行。

  ```
  <h1>这是一个标题。</h1>
  ```

- 段落：p，浏览器会自动地在段落的前后添加空行。

  ```
  <p>这是一个段落 </p>
  ```

- 文字格式：b，加粗；i，斜体；
- 换行：`<br/>` 。

  ```
  <b>加粗文本</b><br>
  <i>斜体文本</i><br>
  ```

- 空格：**代码文本中的多个空格实际渲染时会被html渲染为一个空格**，使用空格字符实体 ` `（notbreaing space）解决。

```
<p>abc  def</p>
<p>abc  </pp>
```

- 大于：`>`，小于：`<` 直接使用>、<会被误认为尖括号标签。

```
3 &lt; 5
5 &gt; 3
```

表格

表格：`<table>` 标签。使用 `<tr>` 绘制表格行，`<td>` 绘制单元格，使用 `<th>` 定义表头。单元格的内容可以包含文本、图片、列表、段落、表单、水平线、表格等等。（点击表格查看源码）

name	age
hollis	24

table标签的常用属性：

- border：**定义表格的边框**，设置值是数值。border=0时就可以隐藏表格边框
- cellpadding： 定义单元格内容与边框的距离
- cellspacing： 定义单元格与单元格之间的距离，设置值是数值
- align：定义表格整体相对于浏览器窗口的**水平**对齐方式，设置值有：left| center|right
- width：表格的宽度，可以是数值（px）、百分比。表格宽度一般选择具体的像素，这样子就不会受到屏幕大小的影响。
- height：高度。

td标签的常用属性：

- align：**单元格内容**相对单元格的水平对齐方式，值为left、right、center
- valign：单元格内容相对单元格的水平对齐方式，值为top、middle、bottom
- colspan：水平方向合并单元格，设置值是数值
- rowspan：垂直方向合并单元格，设置值是数值
- width：单元格宽度，可以是数字、百分比。**此时的百分比是相对于父元素的占比。**当元素内容超过单元格约束大小时，会被挤压变形，若是图片，就会自动撑大单元格。多行多列的单元格长宽取决于同一列中最宽的那个单元格长度。（vh单位）

多个单元格可以设置 `rowspan`、`colspan` 来设置表格跨行或者跨列。

如要绘制空行，使用指定height属性的tr标签，`<tr height=""></tr>`。

以下代码展示表格的基本结构和单元格跨列的使用。

```
<table width="500px" border="1px">

<thead>
    <th>t1</th>
    <th>t2</th>
</thead>

<tbody align="center">
```

```
    <tr>
        <td>d1</td>
        <td>d2</td>
    </tr>
    <tr>
        <td colspan="2">一行占据两个单元格</td>
    </tr>
    <tr>
        <td rowspan="2">第一个单元格设置rowspan是没有用的</td>
    </tr>
</tbody>


</table>
```

表格也可以用于设计简单的布局。

列表

列表：分为无序（unorderList）列表`<ul>`和有序（orderList）列表`<ol>`，列表项使用`<li>`标签。

```
<!无序列表>
<ul>
    <li>Coffee</li>
    <li>Milk</li>
</ul>

<!有序列表>
<ol type="a">
    <li>Coffee</li>
    <li>Milk</li>
</ol>
```

超链接

超链接：格式为`<a href="your link" target="xxx">content</a>`。href若指向外部链接，需要在前面加上`https://`，否则默认为相对路径跳转。内容部分也可以是图片。

使用了超链接的文字下面会有一道下划线，可以设置样式取消。

```
a{
    text-decoration:none;
}
```

a标签也可以设置锚点，跳转到页面具体的某一个地方。

如下代码，**点击a标签将会跳转到指定div。**

```
<a href="#second">跳转第2章</a>
<div id="first" style="height: 500px">
    第一章内容
</div>
<div id="second" style="height: 500px">
    第二章内容
</div>
```

图片

图片：`<img src="src" alt="img_title"width="w" height="h">`，空标签，无闭合标签。

背景图：可以设置重复与否、宽和高大小

```
div
{
    background:url(img_flwr.gif);
    background-size:80px 60px;
    background-repeat:repeat;
}
```

表单

表单：使用标签`<form>`来设置，是一个**包含若干表单元素的区域**，`<form>`标签本身不会被显示，**但表单元素会被显示。**提交时会将标签内的所有元素与键值对形式提交，默认是GET。

表单元素均是`input`标签，设置`type`类型可分为文本域(text)、单选框(radio)、复选框(checkbox)、提交按钮（submit）。

文本域：text；passwowd。用户名、密码输入。

元素的name在提交表单时会以字典的形式，`{name:value}`。这就是**设置name的作用。**

```
<form>
<!type确定了输入类型；password不会对用户密码进行明文显示;提交按钮>
username: <input type="text" name="firstname"><br>
Password: <input type="password" name="pwd">
<input type="submit" value="Submit">
</form>
```

单选框：radio

```
<form>
<!单选框必须保证name一致，否则会失去单选的效果>
<input type="radio" name="sex" value="male">Male<br>
<input type="radio" name="sex" value="female" checked='checked'>Female
</form>
```

复选框：checkbox

```
<form>
<!复选框示例>
<input type="checkbox" name="vehicle" value="Bike">I have a bike<br>
<input type="checkbox" name="vehicle" value="Car">I have a car
</form>
```

文件上传： file

```
<form>选择文件<input type="file"></form>
```

下拉框选择：select。 `<option>` 为实际可选项

```
<select>
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="mercedes">Mercedes</option>
    <option value="audi">Audi</option>
</select>
```

提交：submit。当用户单击确认按钮时，表单元素的内容会被传送到另一个文件。表单的 `action` 属性定义了负责处理表单内容的地址， `method` 表示表单的提交方式。

```
<form name="input" action="html_form_action.php" method="get">
Username: <input type="text" name="user">
<input type="submit" value="Submit">
</form>
```

重置：reset。重置reset所属form表单的所有元素内容。value属性为表单元素显示文字。

```
<form><input type="reset" value="重置"></form>
```

块元素 行元素

标准文档流的块元素（block）包括：h、div、p、li，其余均为行内元素（inline）。

块元素会占据一行，无视内容实际长度。行内元素则会依据元素实际宽度显示，在一行之内从左到右排列，直至不能容纳，然后换行。

行内元素无法设置高度、宽度，但是块元素可以。

行内元素、块元素也可以通过样式属性重新设置， `style:"display:inline|block"` 。

清楚浏览器默认样式

```
a,
abbr,
acronym,
address,
applet,
article,
```

```
aside,
audio,
b,
big,
blockquote,
body,
canvas,
caption,
center,
cite,
code,
dd,
del,
details,
dfn,
div,
dl,
dt,
em,
embed,
fieldset,
figcaption,
figure,
footer,
form,
h1,
h2,
h3,
h4,
h5,
h6,
header,
hgroup,
html,
i,
iframe,
img,
ins,
kbd,
label,
legend,
li,
mark,
menu,
nav,
object,
ol,
output,
p,
pre,
q,
ruby,
s,
samp,
section,
small,
span,
strike,
```

```
strong,
sub,
summary,
sup,
table,
tbody,
td,
tfoot,
th,
thead,
time,
tr,
tt,
u,
ul,
var,
video {
  margin: 0;
  padding: 0;
  border: 0;
  font-family: "PingFangSC-Regular", Hiragino Sans GB, Arial, Helvetica,
"\5B8B\4F53", sans-serif;
}
```

CSS

CSS样式：层叠样式表(Cascading Style Sheets)，渲染html标签。

css引入

引入方式

- 内联样式：直接在标签的style属性上赋值

```
<p style="font-family:arial;color:red;font-size:20px;">更改断过的字体，颜色，字体大</p>
<div style="text-align:center;">居中对齐的标题</div>
```

- 嵌入样式：在html的头部 `<head>` 声明 `<style>` 标签，该html文档内都可以使用该样式。

```
<head>
    <style type="text/css">
        body {background-color:yellow;}
        p {color:blue;}
    </style>
</head>
```

- 外部样式：通过link标签连接到外部样式文件，使用时与嵌入样式无异。 rel属性表示引入的是一个样式表，必需属性。

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

常见文本设置

- 文字颜色：`color:red`
- 文字大小：`font-size:"12px"`
- 文字字体：`font-family:"微软雅黑"`
  - 是否倾斜：`font-style:"normal|italic"`
  - 是否加粗：`font-weight:"normal|bold"`
- 行高：`line-weight:"12px"`。行高的效果相当于在文字上下同时加间隙，**当行高设置的与盒子等高时，就会使文字上下居中，这是一个技巧**
- 文字首行缩进：`text-indent:"12px"`

css选择器

css选择器：寻找指定的HTML元素的方法，然后为其应用样式。

编辑样式时，以**键值对**的形式定义某个属性，**分号分隔多个属性。**注释时使用`/*comment*/`。

标签选择器

标签选择器： 以`lable_name{key:value;...}`的格式定义1个标签的样式。作用域为全文档，**创建该标签即生效。**可以自定义标签，也可以内置标签重名。

```
<style>
    div{
        color:red;
        font-size:12px}
</style>

<!--创建标签的时候就被调用-->
<div>hello world</div>
```

id选择

id选择器：以`#`开头定义，格式为`#id{key:value;}`，标签使用`id=`属性调用。id可以重用，但不推荐。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        #greet{
```

```
            color: red;
            font-size: 12px;
        }
    </style>
</head>
<body>

<!--调用 使用id属性-->
<div id="greet">hello world</div>
</body>
</html>
```

类选择器

类选择器：以`.`开头定义，格式为`.clsName{key:value;}`，标签使用`class=`属性调用。最常使用的选择器。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        .mycls{
            color: red;
            font-size: 12px;
        }
    </style>
</head>
<body>


<!--调用 使用class属性-->
<div class="mycls">hello world</div>
</body>
</html>
```

层级选择器

层级选择器： 父子样式，以`.parentCls .childCls{key:value}`的形式（空格分隔父子）定义，深度可以大于2，可以配合标签选择器。**子样式只有在父样式被调用的情况下才可见。**

使用层级选择器可以避免命名冲突，因为相当于增加了类名前缀。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        .mycls {
            color: red;
            font-size: 12px;
        }
        .mycls p{
            color: deepskyblue;
```

```
            font-size: 24px;
        }
    </style>
</head>
<body>


<!--层级选择-->
<div class="mycls">
    <span>hello world</span>
    <p>这个p在mycls之下，会调用样式</p>
</div>
<p>这个p不会生效</p>
</body>
</html>
```

组选择器

组选择器：`.cls1,.cls2 [,..]{key:value;}`的形式（逗号分隔多个选择器）**一次性设置（创建）多个选择器的共有样式**，允许各个选择器在之后**补充定义**（继承共有属性的前提下）、**覆盖定义**（属性名与共有属性重名）。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        .mycls1,.mycls2 {
            color: red;
            font-size: 12px;
        }
        .mycls2{
            font-style: italic;
            color: deepskyblue;
        }
    </style>
</head>
<body>


<!--组选择 ，cls2重写了字体颜色，补充了字体斜体-->
<p class="mycls1">共有属性</p>
<p class="mycls2">cls2的补充定义：字体斜体 颜色深蓝</p>
</body>
</html>
```

属性选择器

属性选择器：以`clsName[attr:value]{key:value}`的形式定义，**先找到该类，再看是否应用了指定了属性，若是，就应用样式。**

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
    input[type="text"]{ /*input标签使用时触发*/
        font-size: 12px;
        color: red;
    }
    </style>
</head>
<body>
<!--属性选择器，type=text触发样式应用-->
username<input type="text">
</body>
</html>
```

伪类选择器

伪类选择器：以`selector:pseudo {key:value;}`的形式定义，**冒号前后不能有空格。**可以理解伪类为一种装饰效果（类python的装饰器）。常见的伪类选择器有hover（定义鼠标悬停时的动作）、first-child 等。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
    p:hover{
        color: red;
    }
    p i:first-child{
        color: deepskyblue; /*p标签中出现的第一个斜体会变色*/
    }
    </style>
</head>
<body>
<!--伪类选择器-->
<p>鼠标停留在此  变为红色</p>
<p><i>该斜体为天蓝色</i><br>
<i>该斜体为黑色</i></p>
</body>
</html>
```

优先级

当一个元素应用了多个CSS样式，应用的顺序为“就近原则”。**优先使用标签属性`style=""`，然后应用CSS的定义顺序（从上到下，同属性覆盖）。**

```
<head>
    <style>
        .cs1{}
        .cs2{}
    </style>
</head>
<body class="cs2 cs1">
    cs2的同名样式会覆盖使用cs1
    应用多个样式时使用空白分隔
</body>
```

盒子模型

将html元素视作盒子，便于设计布局。

相关名词：

- margin：外边距，盒子与盒子间的距离，透明
- padding：盒子内容与边框的距离，透明
- border：盒子边框，可以设置，不透明
- content：盒子元素，可以是文本、图片等 。width、height等样式属性针对的是content。

margin和padding设置语法类似

```
margin:0px;/*上下左右间隔均为0*/
margin:0px 10px;/*上下间隔0，左右间隔10*/
margin:0px 10px 0px 10px;/*顺时针方向，上右下左间隔*/

/*通过left-right-top-bottom设置某一边的属性 */
margin-left:0px;
```

margin设置的值可以是数值(px)、百分比或者auto（要求div具备width或height属性），其中margin可以为负数，两个元素重叠。

浮动

元素默认情况下，按照html的书写顺序，依次嵌入**文档流。**

通过设置`style="float:direction"`来使元素浮动，浮动方向可以是left、right。

```
<div style="float:left">
    该块状区域就会浮动
</div>
```

当元素样式设置为浮动float后，**元素就会脱离文档流，好像漂浮在文档的上方，位置也会重新发生变化。**这就是浮动。打个比方，就好比默认的文档流是陆地，而浮动之后是空中，元素的位置不再拘泥于html的书写顺序。

同时，浮动元素也有可能会遮罩文档元素，对于文字，元素会自动重新布局（这就形成了"文字绕图"效果）。

浮动元素的位置会漂浮起来，**直至遇到1个元素的边界（margin）为止。**假设有多个元素设置了往左浮动，则第1个浮动元素就会浮起，通常情况下，会碰到父元素的边界，然后停下，接着，第2个元素浮起，碰到第1个元素停下……如果，**父元素的宽度不能再容纳下一个浮动元素，则该浮动元素就会换行。**

一旦元素设置为浮动后，块状属性就会转为行内块元素。 块元素占一行，行内元素宽度就是实际内容宽度，在一行之内从左到右排列。

边框崩塌

当父元素没有设置高度，而其子元素全部设置成浮动元素后，父元素就不会在页面上显示（高度为0），这就是边框崩塌。

尝试如下代码，观看崩塌效果。代码中，父元素高度为0，为了明显，背景色设置为了金色，子元素背景色为粉色。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>浮动测试</title>
    <style>
        .mycss{
            float: left;
            background-color: pink;
            margin:0px 20px;
        }
    </style>
</head>
<body>

<div style="background-color: gold">
   <div class="mycss">
       hello world
   </div>

    <div class="mycss">
```

```
        hello world
    </div>

</div>

</body>
</html>
```

解决崩塌的办法：

- 为父元素设置高度`height:100px`
- 父元素设置overflow属性，`<div style="overflow:hidden"></div>`。hidden用来清除浮动（仅在边框塌陷时）。
- 为父元素添加伪类`:after`，after的作用就是自动已有内容之后增加1个元素。

```
.clear:after{

    content: '';          /*在clear类后面添加内容为空*/

    display: block;       /*把添加的内容转化为块元素*/

    clear: both;          /*清除这个元素两边的浮动*/

}
```

定位

自定义元素的位置，通过设置元素的样式属性`position:`来使元素定位生效。在设置定位后，紧跟其后，就要通过top、left等属性设置元素的具体位置，这些位置是当前元素相对于参照物的位置。例如`left:50px;top:100px`就表示当前元素距离参照物的顶部100px，左边缘距离参照物左边缘50px，两个方向结合在一起，当前元素的（左上角）顶点自然确定。

```
position:fixed;/*fixed absolute relative*/
top:10px;   /*位置*/
/*
left:
right:
bottom:*/
```

position的设置有3类：

1. fixed，**位置相对于浏览器窗口**，冻结元素。top等相关属性均表示距离窗口位置。例如，一个网页导航栏的冻结就使用了fixed。
2. relative（相对定位）、absolute（绝对定位）单独使用没有意义，两者搭配使用。当元素使用`position:absolute`时，此时的top等距离就是**相对最近的一个**设置了`relative`定位的元素。例如，消息来临时，app的右上角的小红点就可以使用相对定位。

值	描述
absolute	生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。 元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。
fixed	生成绝对定位的元素，相对于浏览器窗口进行定位。 元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。
relative	生成相对定位的元素，相对于其正常位置进行定位。 因此，"left:20" 会向元素的 LEFT 位置添加 20 像素。
static	默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。
inherit	规定应该从父元素继承 position 属性的值。

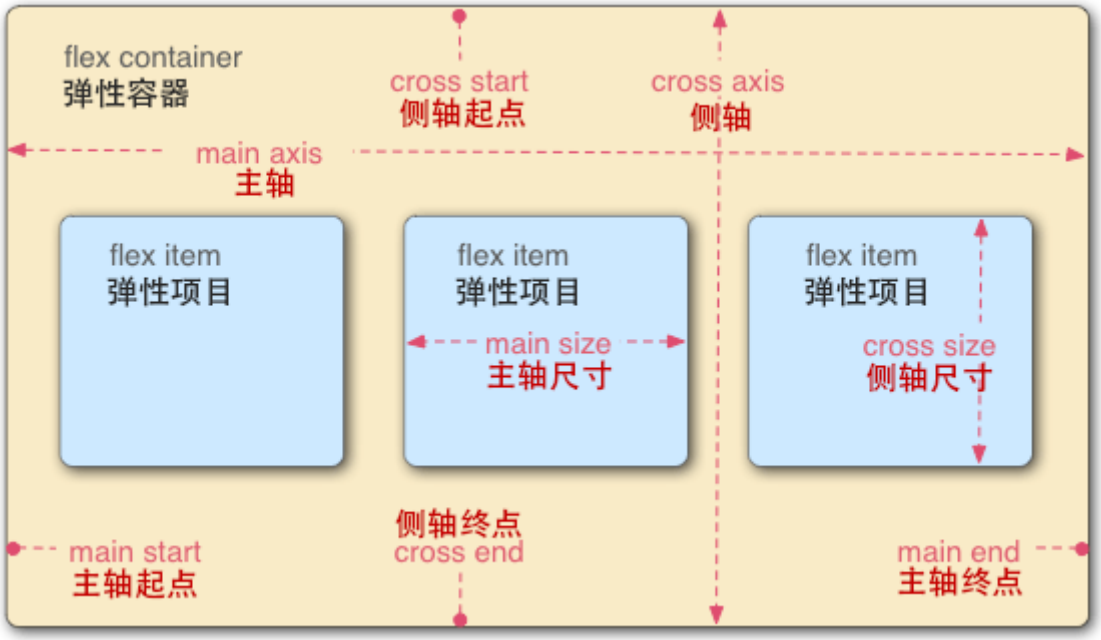
元素一旦定位，就会和浮动一样，脱离标准文档流。

当有多个元素都使用了定位以后，可能会出现彼此覆盖的问题。此时，就要设置样式的`style="z-index:layout_num"`属性。从字面意思上看，就是设置元素的Z属性，类比PS的图层顺序。**数字大的在顶层，同一层的元素按照书写顺序从上到下堆砌**

flex布局

参考：http://www.ruanyifeng.com/blog/2015/07/flex-grammar.html

flex弹性布局，比float、display等传统方式更灵活。



1. 弹性容器：**包含弹性项目的父元素容器。**通过给这个元素设置`display`属性的值为`flex`或者`inline-flex`来定义弹性容器。

```
// 开启flex布局
display:flex;
```

2. 弹性项目 item：容器的每个子元素

3. 轴：分为主轴main和纵轴cross。

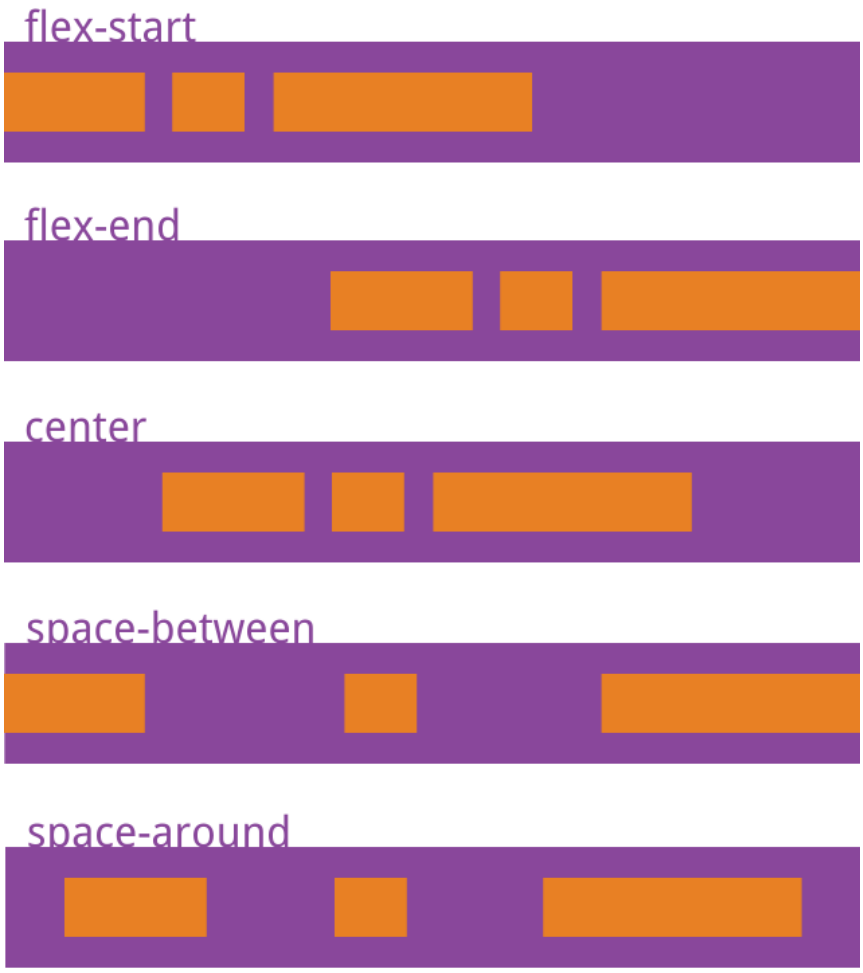
容器属性

`flex-direction`：确定主轴的方向。默认是横向的。

- row：横向布局（默认）
- row-reverse：横向反转布局。
- column：纵向布局。
- column-reverse：纵向反转布局。

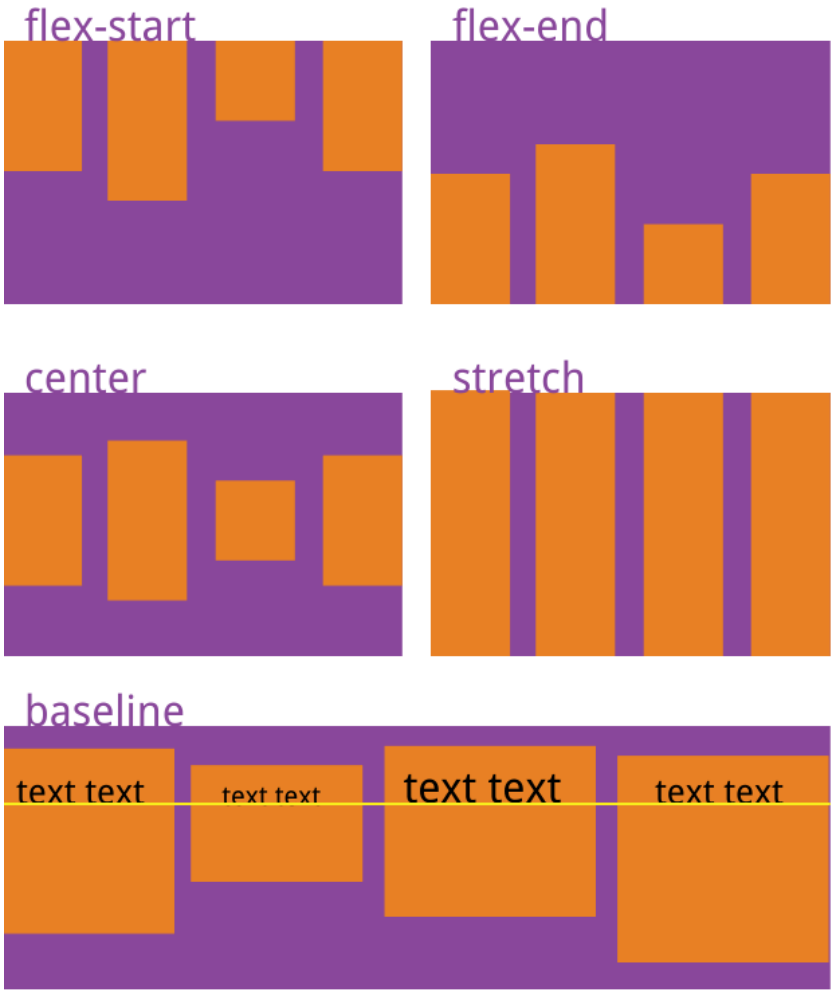
`justify-content`：**定义了item在主轴上的对齐方式。**

- flex-start：靠近主轴的起始点对齐（默认）。
- flex-end：靠近主轴的结束点对齐。
- center：主轴中心排列。
- space-between：**对齐主轴两端**，项目之间的间隔相等。
- space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。



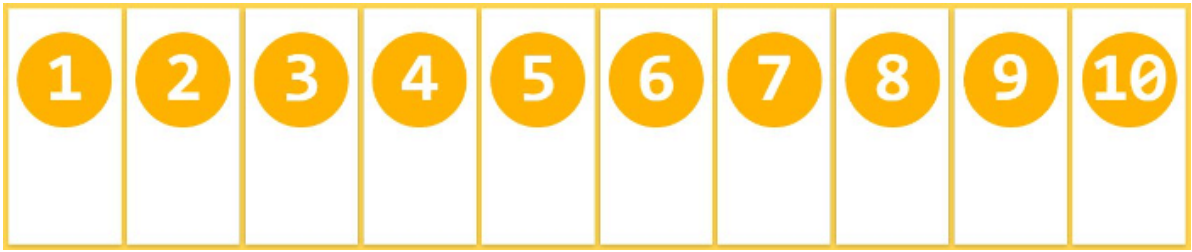
`align-items`：**确定项目在侧轴上如何对齐。**

- flex-start：纵轴的起始点对齐。
- flex-end：纵轴的结束点对齐。
- center：纵轴的中点对齐。
- stretch：默认值。如果没有设置高度。弹性元素被在侧轴方向被拉伸到与容器相同的高度或宽度。

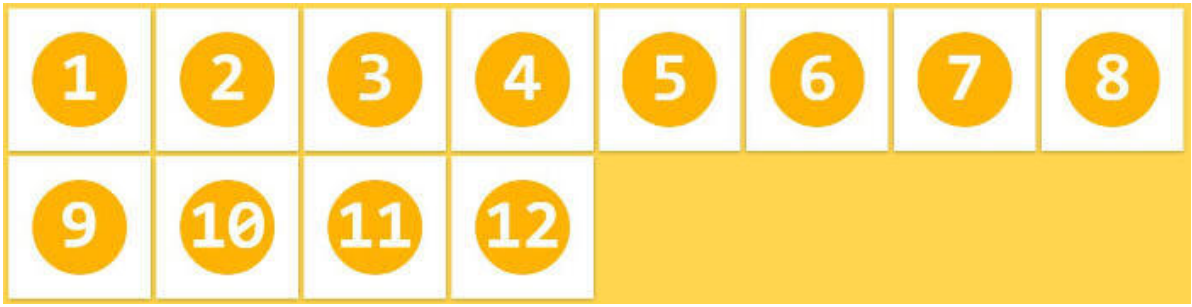


`flex-wrap`：指定子元素在一行排列不下的时候，该如何表现。

- nowrap（默认）：不换行，被挤到一行。



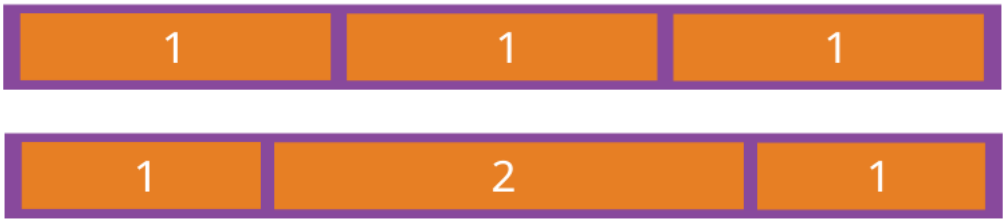
- wrap：被打断到多行中。



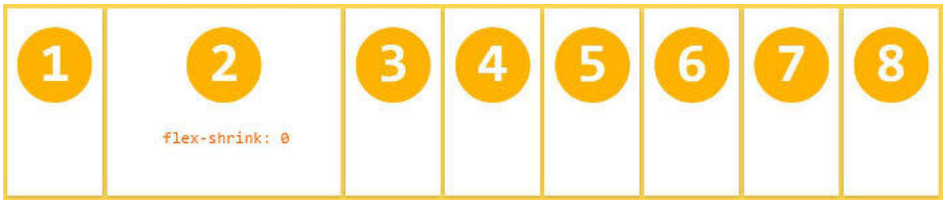
项目属性

项目属性：

`flex-grow`：定义项目的放大比例，默认为`0`，即**如果存在剩余空间，也不放大**



`flex-shrink`：属性定义了项目的缩小比例，默认为1，即**如果空间不足，该项目将缩小**



`flex-basis`：定义了在分配多余空间之前，项目占据的主轴空间。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为`auto`，即项目的本来大小

`flex`：`flex-grow`，`flex-shrink` 和 `flex-basis`的简写，默认值为`0 1 auto` 。该属性有两个快捷值：`auto`(`1 1 auto`)和 none(`0 0 auto`)。

```
flex:1; // 所有设置flex=1的item均分主轴空间 可以自由缩放
```

浏览器样式初始化

```
a,
abbr,
acronym,
address,
applet,
article,
aside,
audio,
b,
big,
blockquote,
body,
canvas,
caption,
center,
cite,
code,
dd,
del,
```

```
details,
dfn,
div,
dl,
dt,
em,
embed,
fieldset,
figcaption,
figure,
footer,
form,
h1,
h2,
h3,
h4,
h5,
h6,
header,
hgroup,
html,
i,
iframe,
img,
ins,
kbd,
label,
legend,
li,
mark,
menu,
nav,
object,
ol,
output,
p,
pre,
q,
ruby,
s,
samp,
section,
small,
span,
strike,
strong,
sub,
summary,
sup,
table,
tbody,
td,
tfoot,
th,
thead,
time,
tr,
tt,
```

```
u,
ul,
var,
video {
  margin: 0;
  padding: 0;
  border: 0;
  text-decoration: none;
  font-family: "Arial",
    "Hiragino Sans GB",
    \5fae\8f6f\96c5\9ed1,
    "Helvetica",
    "sans-serif";
}
```

JavaScript

- javascript教程
- script：包含脚本

JavaScript是一门**解释型脚本语言**，由浏览器解释执行，负责网页的行为。

嵌入方式

- 行间事件：`<input type="button" value="点我看看" onclick="alert('ok! ');">`
- script标签引入：

```
<script type="text/javascript">
alert('ok! ');
</script>

通过src属性外部引用
<script type="text/javascript" src="js/index.js"></script>
```

变量

JavaScript 是一种**弱类型语言**，变量类型由它的值来决定。定义变量需要用关键字`var`。

变量类型

1. number，数字类型
2. string，字符串类型
3. boolean，布尔类型
4. undefined ,当变量声明未初始化，它的值就是undefined
5. null ，表示空对象，类似于python中的None，一般用于释放某个变量的内存或即将为变量赋值。
6. object，复合类型。例如字典、Date、Array类型都是object。

```
var person = {firstName:"John", lastName:"Doe"};
//person['firstname'] 索引
```

7. undefined，变量定义但尚未赋值。

使用`typeof var`语句判断变量类型，返回类型的字符串形式

```
typeof "John"                // 返回 'string'
typeof 3.14                  // 返回 'number'
typeof NaN                   // 返回 'number'
typeof false                 // 返回 'boolean'
typeof [1,2,3,4]             // 返回 'object'
typeof {name:'John', age:34} // 返回 'object'
typeof null                  // 返回 'object'
typeof undefined             // 返回 'undefined'
```

使用各种变量的构造函数可以进行不同类型间的转换。

```
"John".constructor                 // 返回函数 String()  { [native code] }
(3.14).constructor                 // 返回函数 Number()  { [native code] }
false.constructor                  // 返回函数 Boolean() { [native code] }
new Date().constructor             // 返回函数 Date()    { [native code] }s
```

js语句理应**以`;`结尾**，不过这不是必须的，解释器会自己尝试断句，但执行速度可比不加`;`的时候慢。使用`//`单行注释、`/*这是注释*/`来多行注释，和C语言一样。

数组

js中，数组允许有不同类型。

数组创建：

```
var aList = new Array(1,2,3);
var aList2 = [1,2,3]; //方法2
```

数组常用方法：js的数组一些方法即使是原地修改，也会返回一个值，这可能是自身的引用。这一点和python有所区别，python的原地修改只会返回none。

- 长度获取：`.length`
- 索引：`a[index]`
- 获取索引：`.indexOf(val)`
- 元素合并为字符串：`a.join('分隔符')`
- 增删：`.push(val)`、`.pop()`，在尾部增删；`.unshift(val)`、`.shift()`在头部增删。
- 拼接：**注意concat()方法生成了一个新的数组，并不改变原来的数组**

```
var arr = ['tom', 'jerry'];
var arr2 = [1, 2];

var newArr = arr.concat(arr2);
console.log(newArr);
// ["tom", "jerry", 1, 2]
```

- 反转：`.reverse()` 反转。原地修改。
- 删除/添加元素：`splice(index,howmany,item1,.....,itemX)`，找到原数组中index的位置，删除howmany个元素，然后插入新的元素，item1.....itemX 。原地修改。
- 切片：`slice(start,<end>)`，从start位置开始切片，默认直到最后。返回新的数组。
- 过滤：`filter(function(item){})`，匿名函数，参数item表示数组的每个元素。当匿名函数中返回真时，元素保留，否则元素被过滤。返回一个新的数组。

```
var a=[1,2,3,10];
var res=a.filter(item=>item>=3); // 只保留>=3的元素
console.log(res);
```

字符串

- +号运算：字符串拼接，当有number类型变量参与时，会将其转为string类型，如 `12+'13'='1213'`
- parseInt()：将数字字符串转化为整数
- parseFloat()：将数字字符串转化为小数
- 分割：`str.split('分隔符')`，返回数组
- 索引：`str[index]`，`str.charAt(index)`都会得到字符串中的该位置字符。当位置越界时，前者会返回undefined，后者会返回空字符
- 判断是否含子串：`.indexOf('substring')`
- 子串：`.substring(start,end)`，左闭右开
- 大小写：`toUpperCase()`、`toLowerCase()`
- 反转：没有直接的办法，但可以通过数组间接实现。`s.split('').reverse().join('')`

js中的字符串没有提供内置的格式化方法，但是可以自己编写。

```
String.prototype.format=function () {
    //字符串模板替换
    //'{0} {1}'.format(arg1,arg2,..)
    var args=arguments;
    return this.replace(/\{(\d)+\}/g,(d,i)=>String(args[i]));
};

'{0} world'.format('hello'); //hello world
```

元素获取

获取方法

- document.getElementById：获取页面上对应id的元素，返回一个html对象。**该id元素必须加载完后才可以获取到**，为了确保这个效果，做法有二：

  1. 将js脚本放在元素定义后

  ```
  <script type="text/javascript">
    var oDiv = document.getElementById('div1');
  </script>

  <div id="div1">这是一个div 元素</div>
  ```

  2. 将js脚本放在`window.onload=function()`里执行，该函数会在页面所有元素加载完后执行

  ```
  <script type="text/javascript">
  window.onload = function(){
      var oDiv = document.getElementById('div1');
  }
  </script>
  ```

- document.getElementsByTagName('tagName')：获取的是元素集合，但可以用下标的方式来操作。

  ```
  var aLi = document.getElementsByTagName('li');
  ```

属性读写

元素获取后，就可以对其属性读写。以`ele.attr`的形式获取属性，支持链式访问。注意一下几点：

- style属性里，有横杆的改成驼峰写法，如`style.font-size`改为`style.fontSize`
- class改为`className`

```
var oDiv = document.getElementById('div1');
var oid=oDiv.id; //读属性
oDiv.style.color = "red";  //写属性
oDiv.className='box2';  //更改class
```

当以变量存储属性名的时候，`.`操作符要改成`[]`，即`ele[attr_val]`

```
var sMystyle = 'fontSize'; //字符串
var sValue='30px';
// oDiv.style.sMystyle = sValue; sMystyle被认为是一个属性，不起作用
oDiv.style[sMystyle] = sValue; //以[]的形式
```

当然，高效的方法推荐使用JQuery的DOM

函数

函数的定义与执行

```
//定义函数
function fnMyalert(val1,val2='hello') //默认参数
{
    console.log(val1,val2);
    for(var i=0;i<arguments.length;i++){console.log(arguments[i]);} //收集参数
}

//执行
fnMyalert('world');
/*
world hello 第一行打印
world  第二行打印
*/
```

关于参数，js在实参个数不确定的情况下，可以使用每个func的内置变量`arguments`，它是一个类数组对象，收**集了全部实参。**

参数传递

js的参数传递与python的变量标签机制类似，都是隐式的按址传递，在作为实参传入函数时，内存会增加一个新的引用。

函数中，对于不可变类型，如整型、字符串，任何改变都将创建一个新的对象而不会影响原变量。对于可变类型，原地修改，函数外也会受到影响。

作用域

分为全局global局部local，**以函数体作为作用域，和python的L/G机制一样。**示例如下。

```
var a = 12; //定义在函数外，即是全局变量
//在for、while循环等定义的变量i也是全局变量
function myalert()
{
    //局部变量
    var b = 23;
    alert(a);
    alert(b);
}
```

当然，也可以实现像python这样的闭包enclosure。

```
function add() {
    var counter = 0;
    function plus() {counter += 1;} //依旧能够访问到counter属性
    plus();
    return counter;
}
```

使用`const`声明的变量是常量，在声明时就要初始化。原理是地址引用，**不允许修改顶层**，意味着当变量是数组、字典这种时，允许变动（类似python的元组）

使用 `var` 声明的变量不具备块级作用域的特性，它在 {} 外依然能被访问到。

使用`let` 声明的变量只在 let 命令所在的代码块 **{}** 内有效，在 **{}** 之外不能访问。

```
const PI = 3.141592653589793;
{
    let x = 2;
    var y=20;
}
// 这里不能使用 x 变量
// 这里可以使用 y 变量
```

变量与函数与解析

js代码和python不同，不会完全按照代码顺序由上至下解释，而是会进行一次**预编译**，包括两部分工作：**函数会提前进行编译，var变量会提前定义并声明为**`undefined` 。简单的概括，函数调用不受函数定义时位置的约束，变量在使用`var`定义前调用为`undefined`，在其后调用为相应的赋值。

```
fnAlert(); //函数会提前编译，弹出hello!
alert(iNum); // 弹出undefined
function fnAlert(){
    alert('hello!');
    return;
}
var iNum =123; //iNum这个变量会提前，此处相当于重新赋值
```

匿名函数

省略函数名的写法，不需要重复使用，常见于和元素事件的绑定。

```
oBtn.onclick = function(){
    var oDiv = document.getElementById('div1');
    oDiv.style.color = "red";
    oDiv.style.fontSize = "30px";
};
```

对于非常简单的逻辑，使用`=>`定义匿名函数，叫做胖箭头符号，类似python的`lambda`。

```
g=x=>x**2;
add=(x,y)=>(x+y);
g(3); //9
add(2,4) //6
```

异常

```
try {
    ...    //异常的抛出
    throw  '这是一个自定义错误';
} catch(err) { //err变量名任意
    console.log(err.message)    //异常的捕获与处理
} finally { //try部分无论正确与否，最后都要执行
    ...    //结束处理
}
```

this关键字

1. 在对象方法中， **this 指向调用它所在方法的对象。**

```
var person = {
  name: "hollis",
  age : "24",
  show : function() {
      //this 指向person ，函数的所属者
    return this.name + ":" + this.age;
  }
};
person.show();
```

2. 单独使用 this，它指向全局(Global)对象。
3. 函数使用中，**this 指向函数的所属者。**
4. 严格模式(use strict)下函数是没有绑定到 this 上，这时候 this 是 undefined。
5. **在HTML 事件句柄中，this 指向了接收事件的 HTML 元素。**

```
<button onclick="cliclon(this){this.style.display='none'"}>点我后我就消失了
    行间事件绑定的时候，this要以参数的形式传给函数
</button>
```

```
<table border='1px'>
    <tr><td>1</td><td>1</td></tr>
    <tr><td>2</td><td>2</td></tr>
    <tr><td>3</td><td>3</td></tr>
</table>
<script type="text/javascript">
    var tr=document.getElementsByTagName('tr');
    console.log(tr);
    for(var i=0;i<tr.length;i++){
        tr[i].onmouseover=function(){
            this.style.color='red'; //这个this指的是每一个tr
        };
        tr[i].onmouseout=function(){
            this.style.color='';
        };
    }

</script>
```

6. 匿名函数 `()=>this`，可以访问到外面一层的this对象；而 `function(){this}` 只能是里面的this；

7. apply 和 call 允许切换函数执行的上下文环境（context），即 this 绑定的对象。

```
var person1 = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
var person2 = {
  firstName:"John",
  lastName: "Doe",
}
person1.fullName.call(person2);  // 返回 "John Doe"，也就是说this切换到了person2
```

void(0)

`void()` 关键字代表不返回任何值，但是括号内的表达式还是要运行。常见于定意死链（不进行任何跳转的a标签）。

```
// 阻止链接跳转，URL不会有任何变化
<a href="javascript:void(0)" rel="nofollow ugc">点击此处</a>
```

类

function.prototype

es5中创建对象的写法如下，**通过prototype属性可以向类添加方法和属性。**

```
//函数名可以看做类名，事实是一个构造器constructor
function Auth(name){ //同时担任init的作用，可以传参
    var self=this;
    var name=name; //函数变量
    self.name='jhk'; //对象属性，其余对象函数都能访问到

}

Auth.prototype.run=function(){
    var self=this;
    console.log(self.name); //将会得到'jhk'
}; //添加类方法
Auth.prototype.age=null; //添加类属性

function main(){
    var auth=new Auth('jhk'); //实例化，用new关键字
    auth.run();
}
```

构造器一旦定义完成，是不可以直接向其中添加属性方法的，例如`Auth.sex='male'`就是非法的。

js中，**所有对象都会从一个prototype对象中继承属性方法。**当访问一个对象的属性时，会遵循自底向上的搜索顺序（类似于python的继承搜索），直到到达继承链的末尾（object）或者找到属性。

class

在es6中，追加了`class`关键字，构造类更直观。本质是语法糖。

```
class Auth{
    // 构造函数
    constructor() {
        // 会在 new Auth() 的时候自动调用
        // 创建类属性
        this.user=null;
        this.token=null;

    }

    setToken(user,token){
        // 类方法 不需要functin关键字
    }

    // get 和 set 关键字会在读写属性时拦截
    // 相当于python的 @property
    // obj.isAuthed
    get isAuthed(){
        return (this.user&&this.token)
    }

}
```

单例设计

```
class Auth{
    static instance=null; // 是否已经存在实例
    constructor(){
    }

    static getInstance(){
        // 单例模式
        // static关键字
        if(!Auth.instance){
            Auth.instance=new Auth();
        }
        return Auth.instance;
    }
}

export Auth.getInstance() // 最后只会导出一份单例
```

条件语句

运算符：

- 算术：加+ 减- 乘* 除/ 求余%
- 赋值：=、+=（其余同理）
- 条件运算：==（判断值相等）、===（严格判断，值和类型相等）、>=、!=
- 逻辑运算：与&&、或||、非!

关于==和===：

- ==会在两个值类型相同时，进行===比较
- 当类型不同时，分为：
  - 当str类型和num类型比较时，数值相等即为真
  - 当null和undefined比较时，返回真

if-else if-else：

```
if(a>b){
    sTr = '大于';
}
else if(a==b){
    sTr = '等于';
}
else{
    sTr = '小于';
}
```

switch

```
switch (iNow){
    case 1:
        //...;
        break;
    case 2:
        //...;
        break;
    default:
        //...;
}
```

for循环，支持break、continue语句。

```
for(var i=0;i<len;i++)
{
    //......
    break;
    continue;
}

//当对象是object时 用in;
//当对象是array是 用of
for(var i in/of object) {}
```

while循环

```
var i=0;
while(i<8){
    i++;
}
```

export

es6新增的语法。

`export`**命令用于规定模块的对外接口。**

一个模块就是一个独立的文件。该文件内部的所有变量，外部无法获取。如果你希望外部能够读取模块内部的某个变量，就必须使用`export`关键字输出该变量。例如

```
// profile.js
// 可以向外部导出多个变量
export var firstName = 'Michael';
export var lastName = 'Jackson';
export var year = 1958;
```

与之等价的写法是在脚本末尾用`{}`包裹导出变量，这样会更清晰。

```
export {firstName, lastName, year};
```

除了变量，export还可以导出函数、对象等。**同时，它向外指定了变量名。**

导入使用import命令，`import`命令接受一对大括号`{}`，里面指定要从其他模块导入的变量名，名字需要和export导出时命名相同

```
import {lastName} from './profile.js';
```

如果想为输入的变量重新取一个名字，`import`命令要使用`as`关键字，将输入的变量重命名。

```
import { lastName as surname } from './profile.js';
```

可以看到，直接使用export的局限在于，用户必须知道导出模块的变量名叫做什么，但是这一点不应该让用户考虑。

简便起见，引入`export default`命令，为模块指定默认输出。 在import时，可以指定任意名字，也不再需要`{}`。

```
// customName 这个名字可以随便取
import customName from './export-default';
```

命名规范

参考：https://www.cnblogs.com/Hsong/p/9016950.html

常量：大写，下划线间隔

变量：**小驼峰**，名词开头

函数：小驼峰，动词开头

类：大驼峰

正则表达式

语法

语法：`/正则表达式主体/修饰符(可选)`，**没有加引号。头尾的斜杆`/表达式/`可以抑制js自身对`\`的转义，类似于python的`r'\w'`与`\\w`。**同时，`//`包裹的字符串不再加上引号（引号被认为一个正常的字符来处理）。

```
修饰符：i,无视大小写；g全局,不会遇到第一个匹配的就停下
var patt = /runoob/i
```

正则表达式常用于字符串匹配

```
var str = "Visit Runoob!";
var n = str.search(/runoob/i); //6 返回字符串位置 注意到正则是小写的r
n=str.replace(/runoob/i,'hollis!') //Visit hollis!!
```

RegExp对象

RegExp对象是一个**预定义了属性和方法的正则表达式对象**，类似python的`re.compile()`编译后的正则对象。但js的正则对象无需多此一举。

```
var patt = /e/;
patt.test("The best things in life are free!"); //直接调用了方法test
```

常用对象方法：

- test(str)：检测str是否包含正则对象

```
/*校验是否中文名称组成 */
function ischina(str) {
    var reg=/^[\u4E00-\u9FA5]{2,4}$/;   /*定义验证表达式*/
    return reg.test(str);     /*进行验证*/
}
```

- exec(str)：在str中匹配，返回完整形式以及分组的匹配结果，以数组形式包裹

```
/e/.exec("The best things in life are free!"); //E
/e(\b\w+\b)/.exec('The best things in life are free!') //['est','st']
```

JSON

JSON，英文全称 **J**ava**S**cript **O**bject **N**otation，是一种轻量级的数据交换格式。JSON仅仅是一个文本（字符串形式），可以被大部分语言支持，作为中介数据格式传递。

JSON以`{}`包裹键值对，也就是python的字典形式。

```
{"sites":[
    {"name":"Runoob", "url":"www.runoob.com"}
        ]
}
```

js中负责处理json的两个函数：

- JSON.parse(text)：将json字符串转为js对象
- JSON.stringify(val)：将js对象转为json字符串

对象副本：json使用的一个小技巧，来回倒腾一次，可以得到对象的副本

```
obj_copy=json.parse(json.stringofy(obj))
```

定时器

定时器常在js中用于制作动画、异步操作等。

定时器的类型及语法：

```
/*
setTimeout 只执行一次的定时器
clearTimeout 关闭只执行一次的定时器

setInterval 反复执行的定时器
clearInterval 关闭反复执行的定时器
*/

//函数名，时间（ms）
var time1 = setTimeout(myalert,2000); // 2s后执行1次 然后关闭
var time2 = setInterval(myalert,2000); //执行多次
/* 清理定时器，必须先提前将定时器保存为变量
```

```
clearTimeout(time1);
clearInterval(time2);
*/
//函数
function myalert(){
    alert('ok!');
}
```

事件

事件，包括浏览器的行为、用户的行为，如页面加载完毕，鼠标点击、滑动等。

给元素添加事件的方法：

1. 行间事件：`<element some-event='js 代码'>`

   ```
   <button onclick="getElementById('demo').innerHTML=Date()">现在的时间是?
   </button>
   ```

2. js代码：先获取元素，然后给属性赋值函数。如果获取到的元素有多个，例如通过`TagName`获取，则要以循环遍历的形式为每个元素单独绑定事件。

   ```
   var test = document.getElementById("test");
   test.onclick = function (){
       //do something
   };

   /*
   下面这段代码是同理的，调用对象的addEventListener("event_name", function)方法
   注意事件没有on前缀
   test.addEventListener("click", myFunction, true);
   test.removeEventListener("click", myFunction);
   */
   ```

常见HTML事件

事件	描述
onchange	HTML 元素改变
onclick	用户点击 HTML 元素
onmouseover	用户在一个HTML元素上移动鼠标
onmouseout	用户从一个HTML元素上移开鼠标
onkeydown	用户按下键盘按键
onload	浏览器已完成页面的加载

数据存储

COOKIE

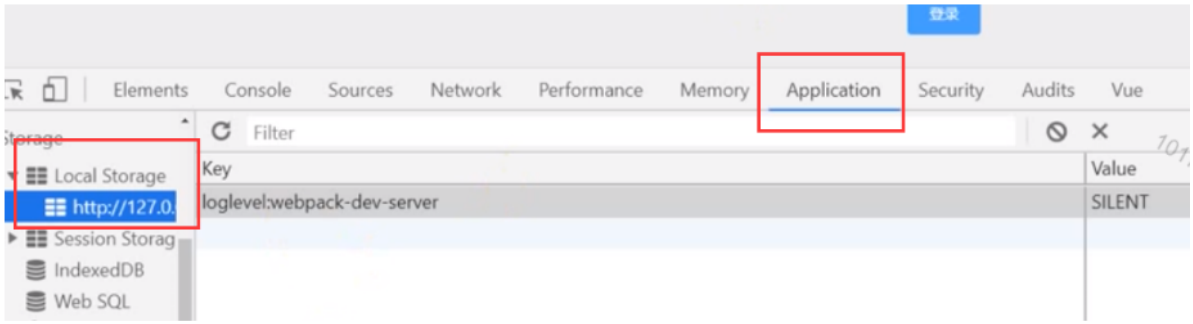
cookie属于document的属性，使用`document.cookie`得到cookie的字符串形式`username=John Doe;...`

```
//读取当前cookie
function getCookie(cname)
{
  var name = cname + "=";
  var ca = document.cookie.split(';'); //分割键值对
  for(var i=0; i<ca.length; i++)
  {
    var c = ca[i].trim(); //去除前后可能存在的空格
    //如果是我们想要的键
    if (c.indexOf(name)==0) return c.substring(name.length,c.length); //只返回val
部分
  }
  return "";
}
```

localstorage

lcoalstorage在高版本浏览器被支持，可以在浏览器本地存储键值对数据，比cookie拥有更大的存储空间。

```
// localstorage 只存储的数据字符串 如果是json 要转换
localStorage.setItem("lastname", "Smith"); //设置键值对
localStorage.getItem("lastname"); // 取出键
localStorage.setItem("lastname", "Smith");
localStorage.removeItem("lastname"); // 移除键
```



JQuery

jq是一个JavaScript的库（框架），可以高效地完成html元素选取、操作、事件绑定、动画制作等效果。

关于jq的引入，可以选择下载到本地然后引用，或者选择在线引用（CDN加速）。

选择器

格式：`$(selector).action()$`。

页面引入JQuery库后，会增加一个全局变量`jQuery`，简写为`$`，以`jQuery.action()`或`$.action()`的形式来调用库函数。

selector里为选择器语法，**可以同时选中多个html元素**，数组形式返回，每个元素为DOM对象。

action则是Jquery的对象方法，和js原生的DOM对象方法有所区别。当selector选中的是多个元素时，`.action()`表示批量对这些元素执行方法。如果要循环设置，可以调用`.each(function([index,]){})`。

```
$.each(allTpInfo, function (index, value) {//index当前索引
    if (value.username == username) {
       return false;// return false=break; true或者其他为continue
   }
 });
```

基本选择器：

```
$("#id")             //ID选择器
$("div")             //标签选择器
$(".classname")      //类选择器
$(".classname,.classname1,#id1")      //组合选择器
```

更多选择示例

语法	描述
$("*")	所有元素
$(this)	当前元素
$("p.intro")	所有class=intro的p元素
$('p,h1,h2')	相应的所有标签
$('div p'), 层级元素选择	所有div标签下的p元素
$('div>p'), > 层级元素选择	div标签下的儿子p元素
$("p:first"), : 过滤器选择	第一个（可以自定义）p元素
$("a[target='_blank']"), [] 属性选择	所有满足该属性的a标签

通常地，我们需要页面加载完DOM元素之后，再去执行我们定义的js代码，这称作“**文档就绪**”。在JQuery中，实现的方法有以下两种，两者等价：

```
$(document).ready(function(){

   // 开始写 jQuery 代码...

});

//简洁版
$(function(){
    // 开始写 jQuery 代码...
})
```

事件

绑定 解绑

Jquery中元素绑定事件的常用语法格式：`$(selector).action(function(){})`。

使用`.unbind('事件名'[,函数名])`来解绑事件。

常见事件：

鼠标事件	键盘事件	表单事件	文档/窗口事件
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload
hover			

例如：

```
 $("p").click(function(){
    // 动作触发后执行的代码!!
});
$("p").click(fn1); //多重绑定会叠加事件，而不是覆盖
$("p").unbind('click'); //解绑元素的所有click事件
```

使用委托`.delegate`可以**向未来可能被动态创建的元素添加事件**
`$(selector).delegate(childSelector,event,data,function)`。

```
//向div>button添加click事件
$("div").delegate("button","click",function(){
  $("p").slideToggle();
});
```

阻止默认事件

在一些情况下，需要阻止元素的默认事件，如使用ajax请求的submit表单按钮。语法：`event.preventDefault()`，event参数为事件函数的第一个实参

```
$("a").click(function(event){
    event.preventDefault();
});
```

DOM

获得内容

- text()：获取元素的文本内容
- html()：和text基本相同，但同时包含html标签
- val()：获取表单字段的值
- attr(属性名)，prop(属性名)：获取元素属性（对比JS属性读写）。attr返回字符串形式的结果，用于自定义属性。prop用于DOM原有属性，如id、class、checked等。
- `$('input[type="file"]')[0].FILES[0]`：专门用于表单文件上传，获取文件内容。第一个`[0]`得到DOM对象，FILES[0]是因为一个标签可能上传多个文件，例如`<input type="file" multiple="multiple">`。

设置内容

- text(文本)：设置元素文本内容
- html(文本)：在text的基础上，解析html标签
- val(文本)：设置表单值
- attr(属性名,值)，prop()：得到attr属性
    - removeAttr(属性名)：删除属性

```
$("#test1").text("Hello world!");
$("#test2").html("<b>Hello world!</b>"); //含有b标签
$("#test3").val("RUNOOB");
//设置单个属性
$("#runoob").attr("href","http://www.runoob.com/jquery");
//设置多个属性
$("#runoob").attr({
    "href" : "http://www.runoob.com/jquery",
    "title" : "jQuery 教程"
})
```

```
//反选按钮 示例
$('#exclude').click(function () {
    $('input[type="checkbox"]').each(function () {
        //checked属性取反
        $(this).prop('checked',!$(this).prop('checked'));
    });

});
```

添加元素

- append(文本/**JQ对象**)： 在被选元素的结尾插入内容
- prepend()： 在被选元素的开头插入内容
- after()： 在被选元素之后插入内容（元素内部）
- before()： 在被选元素之前插入内容

```
$("p").append("追加文本");  //插入普通文本
fucntion(){
    var txt2=$("<p></p>").text("文本。");  // 使用 jQuery 创建文本
    var txt3=document.createElement("p");  //js原生语句生成元素对象
    txt3.innerHTML="文本。";
    $("body").append(txt2,txt3);        // 批量追加新元素
}
//after、before同理
```

删除元素

- remove()： 删除被选元素（及其子元素）
- empty()： 删除被选元素的子元素

```
$("#div1").remove();
```

CSS设置

- addClass()： 向被选元素添加一个或多个类
- removeClass()： 从被选元素删除一个或多个类
- toggleClass()：开关class
- css(styleName,*val*)： 设置或返回样式属性

```
$("h1,h2,p").addClass("blue"); //批量选择元素，然后添加,blue是一个样式
$("body div:first").addClass("important blue"); //批量添加样式，空格分隔

//删除，与add同理
$("h1,h2,p").removeClass("blue");

//返回css属性
$("p").css("background-color");
$("p").css("background-color","yellow"); //设置css
$("p").css({"background-color":"yellow","font-size":"200%"}); //设置多个属性
```

DOM总结

相比于js原生语法的DOM，Jquery都是通过各种`.action()`来操作DOM。例如，通过`.css()`、`.attr()`、`.tetx()`来获取css、attr属性、文本内容。

而js原生语法，相同的一点就是先获取对象`document.getElementById()`，不同的是，通过`.attr`的形式来设置属性

遍历

JQuery的遍历，指的是在DOM树中，遍历寻找元素的方法。

祖先

- parent()：上一级父元素
- parents()：父元素，直至根节点`<html>`
- parentsUntil('界定标签')：返回介于起始标签、界定标签之间的父元素

```
$("span").parent();
$("span").parents("ul"); //支持过滤，向上遍历父元素，要求为ul
$("span").parentsUntil("div"); //返回介于 <span> 与 <div> 元素之间的所有祖先元素
```

后代

- children()：下一级子元素
- find()：所有符合条件的子元素

```
$("div").children();
$("div").children('p.test'); //class=test的p子元素

$("div").find("span");//div后的所有span元素
$("div").find("*");  //div后的所有元素
```

同胞

- siblings()：所有兄弟节点
- next()：下一个节点
- nextAll()：自己后面的所有兄弟
- nextUntil(*界定标签*)：在上一个all的基础上增加界定
- prev()：与next同理，方向相反；下同。
- prevAll()
- prevUntil()

过滤

- first()：返回被选元素的第一个
- last()：返回被选元素的最后一个
- eq(index)：得到指定索引的元素，从0开始
- filter('selector')：得到指定选择器的元素
- not()：filter的反选效果
- index()：得到本元素在同辈元素中的索引

```
$("p").not(".url"); //css!=url 的p
$("p").filter(".url"); //css=url 的p，等效于$('p:first')
$("p").eq(1); //得到第一个元素
$("div p").last(); //得到最后1个
```

动画

- 隐藏，显示：hide(*time,callback*)；show()
- 淡入，淡出：fadeIn();fadeOut();
- 线性动画：animate({css属性字典},time)。元素在初始状态要先设置一个css，animate的参数css是最终态的效果。注意，要进行位置动画时，元素要设置`position`属性，css字典的属性是驼峰命名(`fontSize`)。**可以链式调用**，以队列的形式存储。异步执行，不会阻塞当前程序。

- 停止动画：stop()

GET POST

Jquery使用AJAX来进行GET\POST

GET：`$.get(URL,callback)`

- URL，请求的URL
- callback，服务器响应成功（但响应不一定是200）后的回调函数。

```
// 回调函数形式
function(data,status){
/* data是服务器返回的数据
  status是服务器返回的状态，success,fail
*/
    if(status=='success'){
        //响应成功，如果自己定义了状态码，在这里判断
    }
    else{
        //响应失败
    }
}
```

POST：`$.post(URL,data,callback)`

- URL、callback的意义和GET相同
- data：post提交的数据，字典形式

```
// post 示例
$("button").click(function(){
    $.post(
    "/try/ajax/demo_test_post.php",
    {
        name:"菜鸟教程",
        url:"http://www.runoob.com"
    },
    function(data,status){
        alert("数据: \n" + data + "\n状态: " + status);
    });
});
```

ajax 方法

该方法自由度更高，通常用于其他方法不能完成的请求。

语法：`$.ajax({key:val})`

键值对参考下表

名称	值/描述
async	布尔值，表示请求是否异步处理。默认是 true。
beforeSend(*xhr*)	发送请求前运行的函数（例如携带cookie、设置请求头）
cache	布尔值，表示浏览器是否缓存被请求页面。默认是 true。
complete(*xhr*,*status*)	请求完成时运行的函数（在请求成功或失败之后均调用，即在 success 和 error 函数之后）。
contentType	发送数据到服务器时所使用的内容类型。默认是："application/x-www-form-urlencoded"。
context	为所有 AJAX 相关的回调函数规定 "this" 值。
data	规定要发送到服务器的数据。
dataFilter(*data*,*type*)	用于处理 XMLHttpRequest 原始响应数据的函数。
dataType	预期的服务器响应的数据类型。
error(*xhr*,*status*,*error*)	如果请求失败要运行的函数。
global	布尔值，规定是否为请求触发全局 AJAX 事件处理程序。默认是 true。
ifModified	布尔值，规定是否仅在最后一次请求以来响应发生改变时才请求成功。默认是 false。
jsonp	在一个 jsonp 中重写回调函数的字符串。
jsonpCallback	在一个 jsonp 中规定回调函数的名称。
password	规定在 HTTP 访问认证请求中使用的密码。
processData	布尔值，规定通过请求发送的数据是否转换为查询字符串。默认是 true。
scriptCharset	规定请求的字符集。
success(*result*,*status*,*xhr*)	当请求成功时运行的函数。
timeout	设置本地的请求超时时间（以毫秒计）。
traditional	布尔值，规定是否使用参数序列化的传统样式。
type	规定请求的类型（GET 或 POST）。
url	规定发送请求的 URL。默认是当前页面。
username	规定在 HTTP 访问认证请求中使用的用户名。
xhr	用于创建 XMLHttpRequest 对象的函数。

ajax请求示例

```
$.ajax({
            url: '/getUsers',
            type: 'get',
            dataType: 'json',
            data: {
                //'a': 1,
                //'b': 2,
            },
            success: function (response) {
                console.log(response);
            }
        })
```

AJAX

AJAX = Asynchronous JavaScript and XML，异步的 JavaScript 和 XML。可以在不重新刷新网页的情况下，与服务器交换数据，达到局部更新，因此，常用于动态更新网页。

XHR对象

XMLHttpRequest 是 AJAX 的基础。所有现代浏览器都内建XMLHttpRequest。

创建XHR对象：

```
variable xmlhttp=new XMLHttpRequest();
```

所有方法都建立在XHR对象的基础上

请求与响应

将请求发送到服务器上的方法：

方法	描述
open(*method*,*url*,*async*)	构造请求头，规定请求的类型、URL 以及是否异步处理请求。 • *method*：请求的类型；GET 或 POST • *url*：文件在服务器上的位置 • *async*：true（异步）或 false（同步）
send(*string*)	将请求发送到服务器。 • *string参数仅用于 POST 请求*

```
//GET示例
```

```
xmlhttp.open("GET","/try/ajax/demo_get2.php?fname=Henry&lname=Ford",true); //构建请求
xmlhttp.send(); //发送请求

//简单的POST示例
xmlhttp.open("POST","/try/ajax/demo_post.php",true);
xmlhttp.send();

/*
open只能构建method、url,
当需要像form表单这样形式提交数据时
就要通过setRequestHeader设置请求头
在send中设置POST的数据
*/
xmlhttp.open("POST","/try/ajax/demo_post2.php",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xmlhttp.send("fname=Henry&lname=Ford");
```

响应

属性	描述
responseText	获得字符串形式的响应数据。
responseXML	获得 XML 形式的响应数据（服务器返回XML数据）。

```
//更改元素的HTML文本
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

readyState

XHR请求自发出到响应，一共有以下若干状态，称作`readyState`。**当readyState状态改变时，就会自动触发`onreadystatechange()`函数。**

XHR属性	解释
onreadystatechange	每当 readyState 属性改变时，就会调用该函数。
readyState	存有 XMLHttpRequest 的状态。从 0 到 4 发生变化。0: 请求未初始化1: 服务器连接已建立2: 请求已接收3: 请求处理中4: 请求已完成，且响应已就绪
status	200: "OK" 404: 未找到页面

一般需要在请求成功且正常响应的情况下，来执行一些操作

```
xmlhttp.onreadystatechange=function()
{
    if (xmlhttp.readyState==4 && xmlhttp.status==200) //请求成功且响应200
    {
       //myfunction ，在这做一些操作，因为只有成功的时候才会走到这里
    }
}
```

示例:

```
// 自定义请求函数 传入请求url、请求成功时的回调函数
function xfzGet(url,myfunc)
{
    var xmlhttp=new XMLHttpRequest(); //创建XHR对象
    //当请求成功时执行
    xmlhttp.onreadystatechange=function()
    {
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {   //返回数据
            console.log(xmlhttp.responseText);
            myfunc(); //执行回调函数
        }
    }
    //构建请求并发送
    xmlhttp.open("GET",url,true);
    xmlhttp.send();
}

// 调用
xgzGet('/index/',function(){
    console.log('我就是AJAX请求成功时执行的回调函数！')
})
```