

HTML

1. `<html>`与`</html>`之间的文本描述网页
2. `<body>`与`</body>`之间的文本是可见的页面内容
3. HTML 标题是通过`<h1>`到`<h6>`等标签进行定义的。浏览器会自动地在标题前后添加空行。

```
<h1>This is a heading</h1>
```

```
<h2>This is a heading</h2>
```

```
<h3>This is a heading</h3>
```

h1 字体最大，h6 最小，依此类推。

4. html 的全局属性可以用于所有 HTML 元素，如`<div>`、`<a>`等。常见的全局属性如下：

(1) id: 设置元素的唯一 id。

(2) class: 设置元素的一个或多个 class 名。

(3) style: 设置元素的行内 CSS 样式。

(4) title: 设置元素的提示文本。

(5) hidden: 隐藏元素。

(6) draggable: 设置元素是否可拖动。

(7) dropzone: 设置在拖动被拖动数据时是否进行复制、移动或链接。

(8) tabindex: 设置元素的 tab 键次序。

5. HTML 段落是通过`<p>`标签进行定义的。浏览器会自动地在段落前后添加空行。

```
<p>This is a paragraph.</p>
```

```
<p>This is another paragraph.</p>
```

6. HTML 链接是通过`<a>`标签进行定义的。

```
<a href="http://www.w3school.com.cn">This is a link</a>
```

在 href 属性中指定链接的地址。

7. `<a>`标签除了全局属性外，还有以下常用属性：

- download: 设置被下载的超链接目标。
- href: 设置链接指向的页面的 URL。
- target: 设置在何处打开链接文档。值可以为`_blank`、`_parent`、`_self`、`_top`和`framename`。

- type: 设置被链接文档的 MIME 类型。

8. HTML 图像是通过标签进行定义的。

```

```

(1) 必选属性有两个:

- alt: 设置图像的替代文本, 必选。
- src: 设置显示图像的 URL, 必选。

(2) 以下是可选属性

- width: 设置图像的宽度。
- height: 定义图像的高度。
- ismap: 设置了这个属性之后, 当点击一个服务器端图像映射时, 点击坐标会以 URL 查询字符串的形式发送到服务器。服务器端可以根据这些坐标来做出响应。这种点击图片后的动作称为服务器端图像映射。
- usemap: 将图像定义为客户器端图像映射。和服务器端图像映射不同的是, 客户端图像映射通常将图像分成几个部分, 服务器可以根据用户点击图像的不同位置做出不同的响应。而服务器端图像映射只能做出一种响应。

9. <map>标签和<area>标签一起使用, 实现客户端图像映射功能, 用于将图像划分成几个部分, 每个部分作出不同的响应。

```


<map name="planetmap" id="planetmap">
  <area shape="circle" coords="180,139,14" href="venus.html" alt="Venus" />
  <area shape="circle" coords="129,161,10" href="mercur.html"
alt="Mercury" />
  <area shape="rect" coords="0,0,110,260" href="sun.html" alt="Sun" />
</map>
```

10.
标签用于插入一个换行符。

11. 属性值要使用引号。双引号是最常用的, 不过使用单引号也没有问题。在某些个别的情况下, 比如属性值本身就含有双引号, 那么您必须使用单引号。

12. <hr /> 标签在 HTML 页面中创建水平线。

```
<p>This is a paragraph</p>
<hr />
<p>This is a paragraph</p>
<hr />
<p>This is a paragraph</p>
```

效果如下：

hr 标签定义水平线：

这是段落。

这是段落。

这是段落。

13.html 使用"<!--"和"-->" 来定义注释：

```
<!-- This is a comment -->
```

14.
用于在不产生一个新段落的情况下进行换行：

```
<p>This is<br />a para<br />graph with line breaks</p>
```

执行结果如下：

To break
lines
in a
paragraph,
use the br tag.

15. 需要注意的是 HTML 代码中的所有连续的空行也被显示为一个空格。

```
<html>

<body>

<h1>春晓</h1>

<p>
  春眠不觉晓，
  处处闻啼鸟。
  夜来风雨声，
  花落知多少。
</p>
```

```
<p>注意，浏览器忽略了源代码中的排版。</p>
```

```
</body>
```

```
</html>
```

执行效果如下：

春晓

春眠不觉晓，处处闻啼鸟。夜来风雨声，花落知多少。

注意，浏览器忽略了源代码中的排版（省略了多余的空格和换行）。

16.style 属性用于改变 HTML 元素的样式。

```
<html>
```

```
<body style="background-color:PowderBlue;">
```

```
<h1>Look! Styles and colors</h1>
```

```
<p style="font-family:verdana;color:red">
```

```
This text is in Verdana and red</p>
```

```
<p style="font-family:times;color:green">
```

```
This text is in Times and green</p>
```

```
<p style="font-size:30px">This text is 30 pixels high</p>
```

```
</body>
```

```
</html>
```

执行效果如下：

Look! Styles and colors

This text is in Verdana and red

This text is in Times and green

This text is 30 pixels high

17. <blockquote>元素用于长的引用：

```
<p>以下内容引用自 WWF 的网站：</p>
<blockquote cite="http://www.worldwildlife.org/who/index.html">
五十年来，WWF 一直致力于保护自然界的未来。
世界领先的环保组织，WWF 工作于 100 个国家，
并得到美国一百二十万会员及全球近五百万会员的支持。
</blockquote>
```

执行效果如下：

浏览器通常会对 blockquote 元素进行缩进处理。

五十年来，WWF 一直致力于保护自然界的未来。WWF 工作于 100 个国家，并得到美国一百二十万会员及全球近五百万会员的支持。

18. 注释可以跨越多行：

```
<!-- 此刻不显示图片：

-->
```

19. <script> 标签用于定义客户端脚本，比如 JavaScript。

- src：规定外部脚本文件的 URL。
- async：设置异步执行脚本，该属性只有 async 一个值。
- defer：设置是否对脚本执行进行延迟，直到页面加载为止，该属性只有 defer 一个值。

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

20. <link> 标签用于链接到外部样式文件，它有以下常用属性：

- rel：必备属性，设置当前文档与被链接文档之间的关系，stylesheet表示要导入的样式表的 URL。
- href：设置被链接文档的位置。
- type：设置被链接文档/资源的 MIME 类型。

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

21. <style> 标签定义 HTML 文档的样式信息。它有以下常用属性：

- type: 设置样式表的 MIME 类型, 如果 css 文件就是 text/css。

```
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<style type="text/css">
  h1 {color:red;}
  p {color:blue;}
</style>
</head>
```

22. 样式表有三类:

- (1) 外部样式表: 适用于样式需要被应用到很多页面的时候。

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

- (2) 内部样式表: 适用于单个文件重复出现的样式。

```
<head>
  <style type="text/css">
    body {background-color: red}
    p {margin-left: 20px}
  </style>
</head>
```

使用 <style> 标签在文档头部定义内部样式表。

- (3) 内联样式: 适用于某个标签特殊的不重复的样式。

```
<p style="color: red; margin-left: 20px">
This is a paragraph
</p>
```

使用 Style 属性定义内联样式。

23. 使用 Target 属性, 你可以定义被链接的文档在何处显示。

```
<a href="http://www.w3school.com.cn/" target="_blank">Visit W3School!</a>
  在新窗口打开文档。
```

24. html 使用 <a> 标签的 name 属性来创建锚文本, 然后使用 href 来跳转到锚文本:

```
<a name="tips">基本的注意事项 – 有用的提示</a>
.....
<a href="#tips">有用的提示</a>
```

也可以在其他页面中创建指向该锚的链接：

```
<a href="http://www.w3school.com.cn/html/html_links.asp#tips">有用的提示</a>
```

25. 表格由 `<table>` 标签来定义，`<tr>` 定义表格行，`<td>` 定义表格列。`<table>` 标签常用属性如下：

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

- `border`：规定表格边框的宽度。
- `width`：规定表格的宽度。

26. 表格的表头使用 `<th>` 标签进行定义。常用属性有：

- `align`：规定单元格内容的水平对齐方式。
- `valign`：规定单元格内容的垂直排列方式。

```
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

27. 如果某个单元格是空的，一些浏览器可能无法显示出这个单元格的边框。为了避免这种情况，在空单元格中添加一个空格占位符，就可以将边框显示出来。

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>&nbsp;</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

28. 无序列表始于标签。每个列表项始于。

```
<ul>
<li>Coffee</li>
<li>Milk</li>
</ul>
```

29. 有序列表始于标签。每个列表项始于标签。

```
<ol>
<li>Coffee</li>
<li>Milk</li>
</ol>
```

30. 自定义列表以<dl>标签开始。每个自定义列表项以<dt>开始。每个自定义列表项的定义以<dd>开始。

```
<html>

<body>

<h2>一个定义列表: </h2>

<dl>
  <dt>计算机</dt>
  <dd>用来计算的仪器 ... ..</dd>
  <dt>显示器</dt>
  <dd>以视觉方式显示信息的装置 ... ..</dd>
</dl>
```



```
</body>
</html>
```

浏览器显示如下：

一个定义列表：

计算机
用来计算的仪器
显示器
以视觉方式显示信息的装置

31. <div>元素是块级元素，它是可用于组合其他 HTML 元素的容器。常见的用途是文档布局。它取代了使用表格定义布局的老式方法。使用<table>元素进行文档布局不是表格的正确用法。<table>元素的作用是显示表格化的数据。设置<div>元素的类，使我们能够为相同的 <div> 元素设置相同的类：

```
<body>

<div class="cities">
<h2>London</h2>
<p>London is the capital city of England.
It is the most populous city in the United Kingdom,
with a metropolitan area of over 13 million inhabitants.</p>
</div>

<div class="cities">
<h2>Paris</h2>
<p>Paris is the capital and most populous city of France.</p>
</div>

<div class="cities">
<h2>Tokyo</h2>
<p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area,
and the most populous metropolitan area in the world.</p>
</div>

</body>
```

- 32.元素是行内元素，能够用作文本的容器。设置元素的类，能够为相同的 元素设置相同的样式。

```
<!DOCTYPE html>
<html>
<head>
<style>
  span.red {color:red;}
</style>
</head>
<body>

<h1>My <span class="red">Important</span> Heading</h1>

</body>
</html>
```

执行效果如下：

我的重要的标题

33. <div>元素常用作布局工具，因为能够轻松地通过 CSS 对其进行定位。

34. frameset 和 frame 标签用于在一个 html 文件使用多个 HTML 文档。

```
<html>

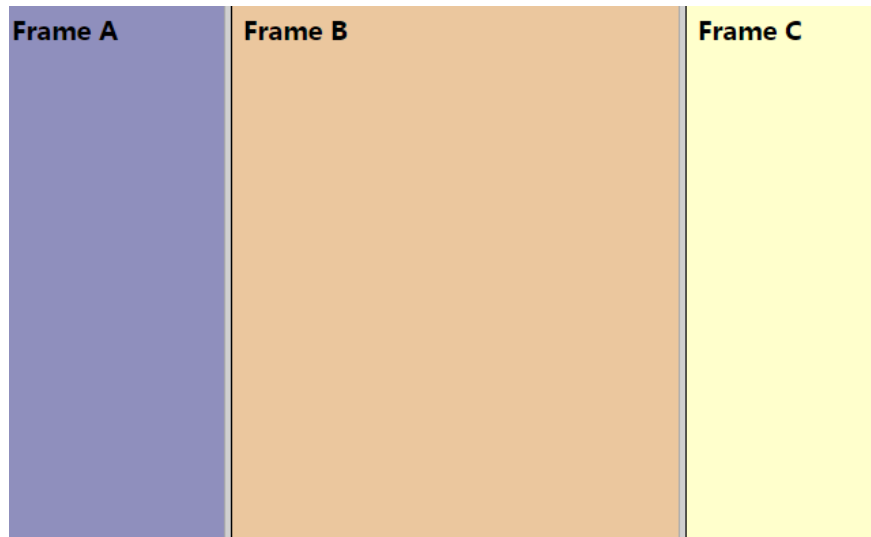
<frameset cols="25%,50%,25%">

  <frame src="/example/html/frame_a.html">
  <frame src="/example/html/frame_b.html">
  <frame src="/example/html/frame_c.html">

</frameset>

</html>
```

col 表示该框架为垂直框架，后面的数据表示框架面积占比。row 表示水平框架，用法和 col 一样。执行结果如下：



35.iframe 用于在网页内显示网页。

```
<iframe src="demo_iframe.htm" width="200" height="200"
frameborder="0"></iframe>
```

frameborder 属性规定是否显示 iframe 周围的边框。

36.iframe 可用作链接的目标。url 的 target 属性必须引用 iframe 的 name 属性：

```
<!DOCTYPE html>
<html>
<body>

<iframe src="/example/html/demo_iframe.html" name="iframe_a"></iframe>

<p><a href="http://www.w3school.com.cn" target="iframe_a">W3School.com.cn</a></p>

<p><b>注释：</b>由于链接的目标匹配 iframe 的名称，所以链接会在 iframe 中打开。</p>

</body>
</html>
```

执行结果如下：

点击链接后，会在框架中显示链接的网址：



W3School.com.cn

注释：由于链接的目标匹配 iframe 的名称，所以链接会在 iframe 中打开。

本页显示在内联框架中。

W3School.com.cn

注释：由于链接的目标匹配 iframe 的名称，所以链接会在 iframe 中打开。

37.<body>标签除全局属性外，其余的可选属性都不推荐使用，例如 background、bgcolor 等都不推荐使用，推荐使用 CSS 样式表。

38.<script>标签用于定义客户端脚本，script 元素既可包含脚本语句，也可通过 src 属性指向外部脚本文件。必需的 type 属性规定脚本的 MIME 类型。

```
<script type="text/javascript">
document.write("Hello World!")
</script>
```

39.<noscript>标签提供无法使用脚本时的替代内容，比方在浏览器禁用脚本时，或浏览器不支持客户端脚本时。

```
<script type="text/javascript">
document.write("Hello World!")
</script>
<noscript>Your browser does not support JavaScript!</noscript>
```

40. 以下标签都可以添加到 head 部分: <title>、<base>、<link>、<meta>、<script> 以及 <style>。

(1) <base> 标签为页面上的所有链接规定默认地址或默认目标:

```
<head>
<base href="http://www.w3school.com.cn/images/" />
<base target="_blank" />
</head>
```

(2) <link> 标签定义文档与外部资源之间的关系, 最常用于连接样式表。

(3) <style> 标签用于为 HTML 文档定义样式信息。您可以在 style 元素内规定 HTML 元素在浏览器中呈现的样式:

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

(4) meta 元素被用于规定页面的描述、关键词、文档的作者、最后修改时间以及其他元数据。

```
<meta name="description" content="Free Web tutorials on HTML, CSS, XML" />
```

41. <!DOCTYPE> 不是 HTML 标签。它为浏览器提供一项声明, 即 HTML 是用哪个版本编写的。

(1) HTML5

```
<!DOCTYPE html>
```

(2) HTML 4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

(3) XHTML 1.0

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

42.<form>元素用于定义 HTML 表单，常见元素如下：

- action：设置当提交表单时发送表单数据到哪个 URL。
- enctype：如何对表单数据进行编码。例如，multipart/form-data 表示不编码，通常用于上传文件。
- method：设置用于发送表单数据的 HTTP 方法。
- name：设置表单的名称。
- target：规定在何处打开 action URL。

```
<form action="action_page.php" method="GET">
.
form elements
.
</form>
```

43.<input>元素是最重要的表单元素。根据不同的 type 属性，<input>元素有很多形态。

(1) 文本 Text：<input type="text"> 定义用于文本输入的单行输入字段。

```
<form>
First name:<br>
<input type="text" name="firstname">
<br>
Last name:<br>
<input type="text" name="lastname">
</form>
```

(2) 单选 radio：<input type="radio"> 定义单选按钮。

```
<form>
<input type="radio" name="sex" value="male" checked>Male
<br>
<input type="radio" name="sex" value="female">Female
</form>
```

(3) 提交按钮 submit：定义用于向表单处理程序提交表单的按钮。

```
<form action="action_page.php">
First name:<br>
```

```
<input type="text" name="firstname" value="Mickey">
<br>
Last name:<br>
<input type="text" name="lastname" value="Mouse">
<br><br>
<input type="submit" value="Submit">
</form>
```

44.input 有以下常用属性:

- name: 设置 <input> 元素的名称。
- value: 设置输入字段的初始值。
- type : 设置要显示的 <input> 元素的类型。
- readonly: 设置输入字段为只读。
- disabled: 设置输入字段是禁用的。被禁用的元素是不可用和不可点击的。被禁用的元素不会被提交。
- size: 设置输入字段的尺寸。
- maxlength: 设置输入字段允许的最大长度。

```
<!DOCTYPE html>
<html>
<body>

<form action="">
First name:<br>
<input type="text" name="firstname" value="John" readonly>
<br>
Last name:<br>
<input type="text" name="lastname">
</form>

</body>
</html>
```

输入框为 John, 无法更改。

45.<fieldset>元素组合表单中的相关数据。<legend>元素为<fieldset>元素定义标题。

```
<!DOCTYPE html>
<html>
<body>
```

```
<form action="/demo/demo_form.asp">
<fieldset>
<legend>Personal information:</legend>
First name:<br>
<input type="text" name="firstname" value="Mickey">
<br>
Last name:<br>
<input type="text" name="lastname" value="Mouse">
<br><br>
<input type="submit" value="Submit">
</fieldset>
</form>

</body>
</html>
```

执行效果如下：



46.<select> 元素定义下拉列表：

```
<select name="cars">
<option value="volvo">Volvo</option>
<option value="saab">Saab</option>
<option value="fiat">Fiat</option>
<option value="audi" selected>Audi</option>
</select>
```

<option> 元素定义待选择的选项。列表通常会把首个选项显示为被选选项。您能够通过添加 selected 属性来定义预定义选项。<select>有以下常用属性：

- name：设置下拉列表的名称。
- multiple：当该属性为 true 时，可选择多个选项。
- size：设置下拉列表中可见选项的数目。

47.<textarea>元素定义多行输入字段:

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

它有以下常用属性:

- name: 设置文本区域的名称。
- rows: 设置文本区域内可见的行数。
- cols: 设置文本区域内可见的宽度。
- readonly: 设置文本区域为只读。
- autofocusNew: 设置当页面加载时, 文本区域自动获得焦点。

48.<button> 元素定义可点击的按钮:

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

它有以下常用属性:

- name: 设置按钮的名称。
- type: 设置按钮的类型, 值可以为 button、reset 和 submit。
- value: 设置按钮的初始值。

49.HTML5 的<datalist>元素为 <input> 元素规定预定义选项列表。用户会在他们输入数据时看到预定义选项的下拉列表。<input> 元素的 list 属性必须引用 <datalist> 元素的 id 属性。

```
<form action="action_page.php">
<input list="browsers">
<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
</form>
```

50.部分<input> 元素的输入类型:

- (1) 文本 text
- (2) 密码 password
- (3) 提交按钮 submit

- (4) 单选 radio
 - (5) 复选 checkbox
 - (6) 按钮 button
 - (7) 数字 number
 - (8) 日期 date: 一般用于选择生日。
 - (9) 范围 range: 包含一定范围内的值的输入字段。
- 51.

CSS

1. jQuery 的 `$()` 括号里面的内容其实就是 CSS 选择器, 所以 CSS 选择器分为基本选择器、层次选择器、过滤选择器和表单选择器四种, 具体见 jQuery 的内容。
2. 如果值为若干单词, 则要给值加引号:

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

3. 可以对选择器进行分组, 同一个组的选择器使用相同的 CSS 样式。

```
h1,h2,h3,h4,h5,h6 {  
  color: green;  
}
```

h1、h2 等划分为一组, 都使用相同的 CSS 样式。

4. id 选择器以 `"#"` 来定义。

```
#red {color:red;}  
#green {color:green;}
```

5. 在现代布局中, id 选择器常常用于建立派生选择器。

```
#sidebar p {  
  font-style: italic;  
  text-align: right;  
  margin-top: 0.5em;  
}
```

6. 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。

7. 元素也可以基于它们的类而被选择：

```
td.fancy {  
    color: #f60;  
    background: #666;  
}
```

在上面的例子中，类名为 fancy 的表格单元将是带有灰色背景的橙色。

```
<td class="fancy">
```

8. 可以为拥有指定属性的 HTML 元素设置样式，而不仅限于 class 和 id 属性。

(1) 下面的例子为带有 title 属性的所有元素设置样式：

```
[title]  
{  
color:red;  
}
```

(2) 下面的例子为 title="W3School" 的所有元素设置样式：

```
[title=W3School]  
{  
border:5px solid blue;  
}
```

(3) 设置表单的样式：

```
input[type="text"]  
{  
    width:150px;  
    display:block;  
    margin-bottom:10px;  
    background-color:yellow;  
    font-family: Verdana, Arial;  
}  
  
input[type="button"]  
{  
    width:120px;  
    margin-left:35px;  
    display:block;
```

```
font-family: Verdana, Arial;
}
```

9. CSS 的 text-align 属性用于设置文本的对齐方式:

```
h1 {text-align:center}
h2 {text-align:left}
h3 {text-align:right}
```

10. 设置字体的属性有以下几个:

(1) font 是一个简写属性, 在一个声明中设置所有字体属性。

```
p.ex2
{
    font:italic bold 12px/30px Georgia, serif;
}
```

(2) font-family 属性用于设置一个元素的字体。

```
p{
    font-family:"Times New Roman",Georgia,Serif;
}
```

将字体设置为 Times New Roman。

(3) font-size 属性设置字体大小。

```
h1 {font-size:250%}
h2 {font-size:200%}
p {font-size:100%}
```

(4) font-style 属性指定文本的字体样式。

```
p.normal {font-style:normal}
p.italic {font-style:italic}
p.oblique {font-style:oblique}
```

- normal 是 font-style 的默认值, 表示显示一个标准的字体样式。
- italic: 显示一个斜体的字体样式。
- oblique: 显示一个倾斜的字体样式。
- inherit: 从父元素继承字体样式。

(5) Color 属性指定文本的颜色。

```
p
{
  color:rgb(0,0,255);
}
```

11. 可以使用 background-color 属性为元素设置背景色。这个属性接受任何合法的颜色值。

```
p
{
  background-color:rgb(255,0,255);
}
```

12. 要把图像放入背景，需要使用 background-image 属性。

```
body {background-image: url(/i/eg_bg_04.gif);}
```

13. 如果需要在页面上对背景图像进行平铺，可以使用 background-repeat 属性。

```
body
{
  background-image: url(/i/eg_bg_03.gif);
  background-repeat: repeat-y;
}
```

- 属性值 repeat 导致图像在水平垂直方向上都平铺，就像以往背景图像的通常做法一样。
- repeat-x 和 repeat-y 分别导致图像只在水平或垂直方向上重复。
- no-repeat 则不允许图像在任何方向上平铺。

14. 可以利用 background-position 属性改变图像在背景中的位置。下面的例子在 body 元素中将一个背景图像居中放置：

```
body
{
  background-image:url('/i/eg_bg_03.gif');
  background-repeat:no-repeat;
  background-position:center;
}
```

background-position 属性的值有很多方法。

- 首先，可以使用一些关键字：top、bottom、left、right 和 center。
- 还可以使用长度值，如 100px 或 5cm，最后也可以使用百分数值。

- 还可以使用百分比方式。如果你想把一个图像放在水平方向 2/3、垂直方向 1/3 处, 可以这样声明:

```
body
{
  background-image:url('/i/eg_bg_03.gif');
  background-repeat:no-repeat;
  background-position:66% 33%;
}
```

15.如果文档比较长,那么当文档向下滚动时,背景图像也会随之滚动。当文档滚动到超过图像的位置时,图像就会消失。您可以通过 background-attachment 属性防止这种滚动。

```
body
{
  background-image:url('/i/eg_bg_02.gif');
  background-repeat:no-repeat;
  background-attachment:fixed
}
```

16.CSS 提供了 text-indent 属性,该属性可以方便地实现文本缩进。

```
p {text-indent: 5em;}
```

- text-indent 还可以设置为负值。
- text-indent 可以使用所有长度单位,包括百分比值。

17.word-spacing 属性可以改变字之间的标准间隔。其默认值 normal 设置值为 0。word-spacing 属性接受一个正长度值或负长度值。如果提供一个正长度值,那么字之间的间隔就会增加。

18.letter-spacing 属性与 word-spacing 的区别在于,字母间隔修改的是字符或字母之间的间隔。与 word-spacing 属性一样,letter-spacing 属性的可取值包括所有长度。默认关键字是 normal。

19.text-transform 属性控制文本的大小写。这个属性有 4 个值:

- none: 默认值,带有小写字母和大写字母标准的文本。
- uppercase: 定义仅有大写字母。
- lowercase: 定义仅有小写字母。
- capitalize: 文本中的每个单词以大写字母开头。

20.text-decoration 用于文本的修饰符,它有 5 个值:

- none: 默认, 标准的文本。
- underline: 添加下划线。
- overline: 添加上划线。
- line-through: 添加贯穿线。
- blink: 让文本闪烁。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<style>
h1 {text-decoration:overline;}
h2 {text-decoration:line-through;}
h3 {text-decoration:underline;}
</style>
</head>

<body>
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
</body>

</html>
```

显示效果如下:

This is heading 1

~~This is heading 2~~

This is heading 3

21.white-space 属性用于处理源文档中的空格、换行和 tab 字符, 默认值为 normal, 空白会被浏览器忽略:

```
{
white-space: nowrap
}
```

遇到换行符不换行，在同一行显示，直到遇到
 标签为止。

22.使用 font-family 属性定义文本的字体：

```
h1 {font-family: Georgia;}
```

当字体名中有空格时，要添加单引号。

23.font-weight 属性设置文本的粗细。

24.font-size 属性设置文本的大小。

25.链接

(1) 链接的四种状态：

- a:link - 普通的、未被访问的链接
- a:visited - 用户已访问的链接
- a:hover - 鼠标指针位于链接的上方
- a:active - 链接被点击的时刻

```
<!DOCTYPE html>
<html>
<head>
<style>
a:link {color:#FF0000;} /* 未被访问的链接 */
a:visited {color:#00FF00;} /* 已被访问的链接 */
a:hover {color:#FF00FF;} /* 鼠标指针移动到链接上 */
a:active {color:#0000FF;} /* 正在被点击的链接 */
</style>
</head>
```

(2) background-color 属性规定链接的背景色。

26.列表

(1) 要修改用于列表项的标志类型，可以使用属性 list-style-type：

```
ul {list-style-type : square}
```

上面的声明把无序列表中的列表项标志设置为方块。

(2) 有时，常规的标志是不够的。你可能想对各标志使用一个图像，这可以利用 list-style-image 属性做到：

```
ul li {list-style-image : url(xxx.gif)}
```


只需要简单地使用一个 url() 值，就可以使用图像作为标志。

(3) 利用 list-style-position 可以确定标志出现在列表项内容之外还是内容内部。

(4) 为简单起见，可以将以上 3 个列表样式属性合并为一个方便的属性：list-style，就像这样：

```
li {list-style : url(example.gif) square inside}
```

27. 表格

(1) 如需在 CSS 中设置表格边框，请使用 border 属性。

```
table, th, td
{
border: 1px solid blue;
}
```

上例中的表格具有双线条边框。这是由于 table、th 以及 td 元素都有独立的边框。

如果需要把表格显示为单线条边框，请使用 border-collapse 属性。

(2) border-collapse 属性设置是否将表格边框折叠为单一边框：

```
table
{
border-collapse:collapse;
}

table,th, td
{
border: 1px solid black;
}
```

(3) 通过 width 和 height 属性定义表格的宽度和高度。

(4) text-align 和 vertical-align 属性设置表格中文本的水平和垂直对齐方式。

```
td
{
text-align:right;
}
```

(5) 如需控制表格中内容与边框的距离，请为 td 和 th 元素设置 padding 属性：

```
td
{
```

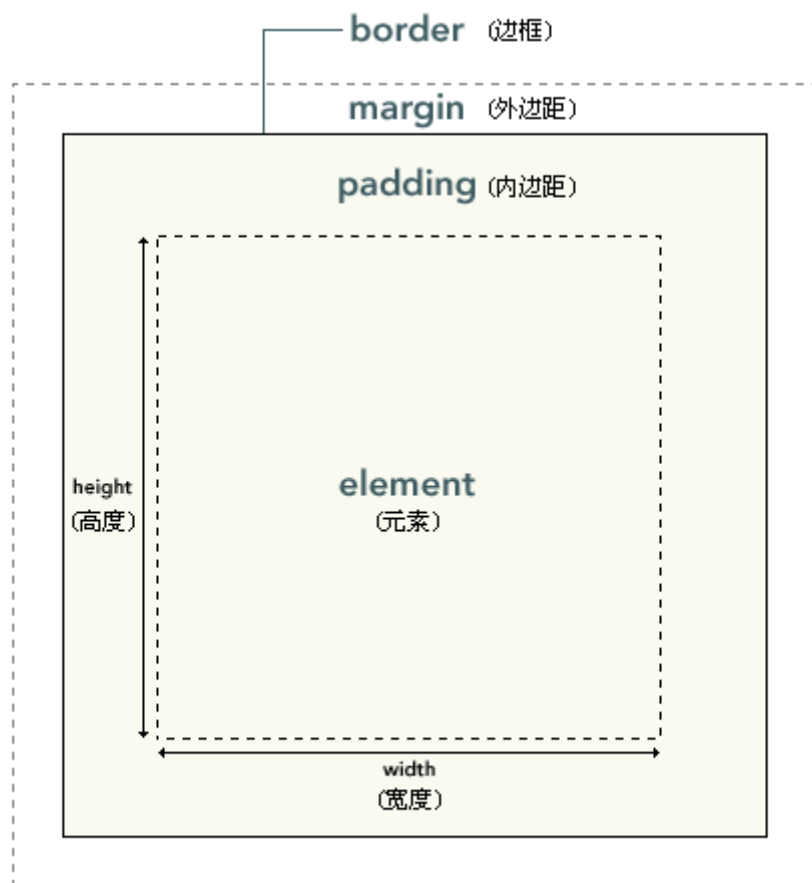
```
padding:15px;  
}
```

(6) CSS 边框属性:

- outline: 在一个声明中设置所有的轮廓属性。
- outline-color: 设置轮廓的颜色。
- outline-style: 设置轮廓的样式。
- outline-width: 设置轮廓的宽度。

注释: 只有在规定了 !DOCTYPE 时, Internet Explorer 8 (以及更高版本) 才支持 outline 属性。

28.CSS 框模型规定了元素框处理元素内容、内边距、边框和外边距的方式。



W3School.com.cn

29.内边距、边框和外边距都是可选的,默认值是零。但是,许多元素将由用户代理样式表设置外边距和内边距。可以通过将元素的 margin 和 padding 设置为零来覆盖这些浏览器样式。这可以分别进行,也可以使用通用选择器对所有元素进行设置:

```
* {  
  margin: 0;  
  padding: 0;  
}
```

30. 术语翻译

- element : 元素。
- padding : 内边距，也有资料将其翻译为填充。
- border : 边框。
- margin : 外边距，也有资料将其翻译为空白或空白边。

31. 内边距

- (1) 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。

```
h1 {padding: 10px;}
```

还可以按照上、右、下、左的顺序分别设置各边的内边距，各边均可以使用不同的单位或百分比值：

```
h1 {padding: 10px 0.25em 2ex 20%;}
```

- (2) 也通过使用下面四个单独的属性，分别设置上、右、下、左内边距：

- padding-top
- padding-right
- padding-bottom
- padding-left

- (3) 内边距的百分比数值。下面这条规则把段落的内边距设置为父元素 width 的 10%：

```
p {padding: 10%;}
```

32. 边框

- (1) 每个边框有 3 个方面：宽度、样式，以及颜色。边框绘制在“元素的背景之上”。

- (2) 边框的样式使用 border-style 属性定义，它定义了 10 个不同的非 inherit 样式，包括 none。

```
a:link img {border-style: outset;}
```

- (3) 可以为一个边框定义多个样式，例如：

```
p.aside {border-style: solid dotted dashed double;}
```

上面这条规则为类名为 aside 的段落定义了四种边框样式：实线上边框、点线右边框、虚线下边框和一个双线左边框。

(4) 如果希望为元素框的某一个边设置边框样式，而不是设置所有 4 个边的边框样式，可以使用下面的单边边框样式属性：

- border-top-style
- border-right-style
- border-bottom-style
- border-left-style

(5) 通过 border-width 属性为边框指定宽度。也可以通过下列属性分别设置边框各边的宽度：

- border-top-width
- border-right-width
- border-bottom-width
- border-left-width

(6) 使用 border-color 属性设置边框颜色。

33. 通过值复制功能，下面的语句可以改写：

```
p {border-style: solid; border-width: 15px 5px 15px 5px;}
```

改写为：

```
p {border-style: solid; border-width: 15px 5px;}
```

34. margin 属性是以下四个属性的简写：

- margin-top
- margin-right
- margin-bottom
- margin-left

35. 外边距合并指的是，当两个垂直外边距相遇时，它们将形成一个外边距。例如毗邻的两个兄弟元素之间的外边距会合并。

36. CSS 定位的基本思想很简单，它允许你定义元素框相对于其正常位置应该出现的位置，或者相对于父元素、另一个元素甚至浏览器窗口本身的位置。

37. div、h1 或 p 元素常常被称为块级元素。这意味着这些元素显示为一块内容，即“块框”。与之相反，span 和 strong 等元素称为“行内元素”，这是因为它们的内容

显示在行中，即“行内框”。

38. `display` 属性规定元素应该生成的框的类型。

```
<html>
<head>
<style type="text/css">
p {display: inline}
div {display: none}
</style>
</head>

<body>
<p>本例中的样式表把段落元素设置为内联元素。</p>

<p>而 div 元素不会显示出来! </p>

<div>div 元素的内容不会显示出来! </div>
</body>
</html>
```

`display: none` 表示 `div` 的元素不会显示出来。

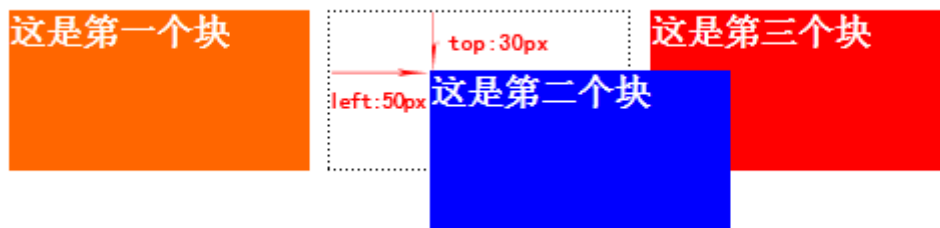
39. CSS 有三种基本的定位机制：普通流、浮动和绝对定位。默认定位机制是普通流定位，也就是说，块级框从上到下一个接一个地排列，框之间的垂直距离是由框的垂直外边距计算出来。

40. `position` 属性指定一个元素的定位方法的类型，默认为 `static`。

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

设置为绝对定位。

41. `left`、`right`、`bottom` 和 `top` 这 4 个属性用于设置块元素的偏移值，通常如果 `position` 值为 `static` 时，这 4 属性是没有任何效果的。



42.z-index 属性用于设置两个元素出现重叠时到底显示哪一个，position 为 static 时该属性无效。z-index 值大的元素会覆盖在 z-index 值小的元素上面，默认值为 0。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<style>
img
{
    position:absolute;
    left:0px;
    top:0px;
    z-index:1;
}
</style>
</head>

<body>
<h1>This is a heading</h1>

<p>因为图像元素设置了 z-index 属性值为 -1, 所以它会显示在文字之后。</p>
</body>
</html>
```

效果如下：



a heading

设置了 z-index 属性值为 -1, 所以它会显示在文字之后。

43.CSS 相对定位：以元素在文档中的初始位置为标准进行浮动。

```
#box_relative {  
  position: relative;  
  left: 30px;  
  top: 20px;  
}
```

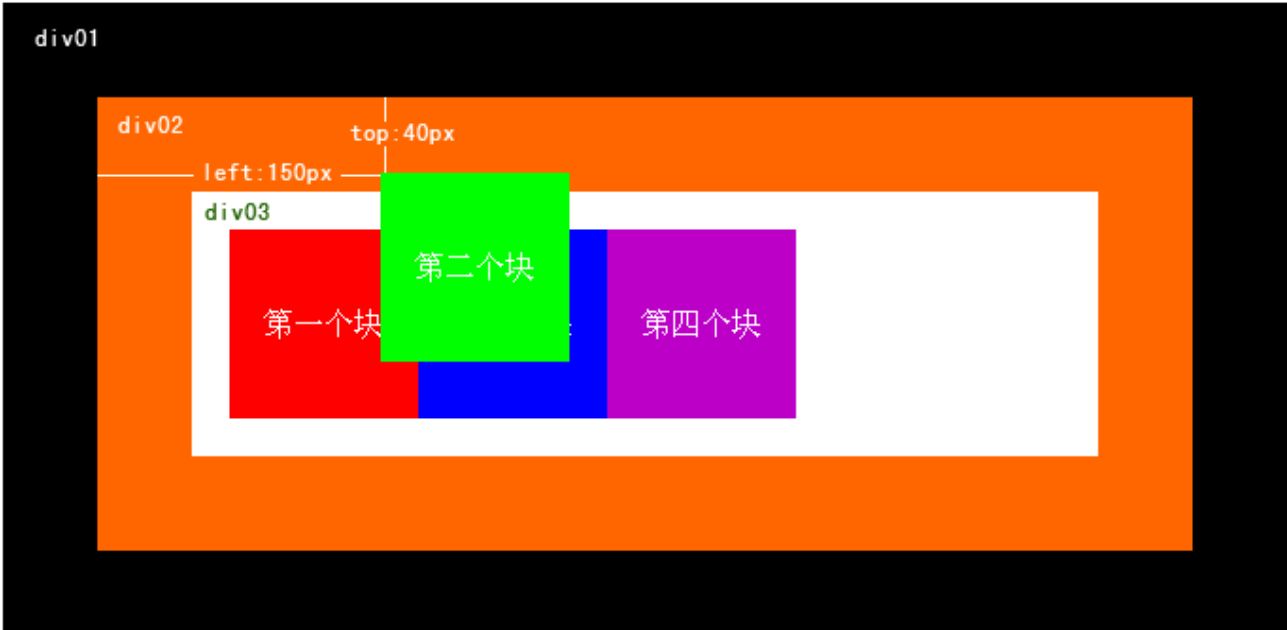
图片的左边缘在文档初始位置的 30px 处，上边缘在初始位置的 20px 处。

44.CSS 绝对定位：相对于最近的已定位祖先元素，如果不存在已定位的祖先元素，那么相对于最初的包含块。

```
<div class="div01"> —— 设置了定位属性: position:relative 但不是最近的定位祖先元素, 不是参照物  
  
<div class="div02"> —— 设置了定位属性:position:relative , 也是最近的定位祖先元素, 是参照物  
  
  <div class="div03"> —— 没有设置定位属性, 虽然是最近的祖先元素, 但不是最近的定位祖先元素, 不是参照物  
  
    <div class="box01">第一个块</div>  
    <div class="box02">第二个块</div> —— 这个块在CSS中设置了绝对定位position:absolute  
    <div class="box03">第三个块</div>  
    <div class="box04">第四个块</div>  
    <div style="clear:both"></div>  
  
  </div>  
  
</div>  
  
</div>
```

```
#box_relative {  
  position: absolute;  
  left: 30px;  
  top: 20px;  
}
```

45.绝对定位在文档流中不占空间，一旦块被设置为绝对定位，该块就会被从文档流中删除。从效果上看，绝对定位的块可能会覆盖某些元素，例如：



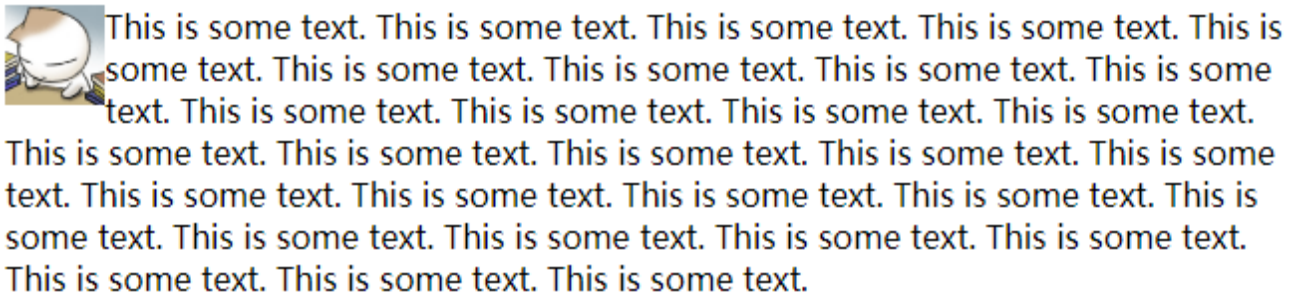
第二小块就是绝对定位的块，覆盖了第三小块和第一小块部分内容。绝对定位的应用就是某些网站的购物车提示：



46.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。



通俗地说，左浮动就是类似下面的效果，图片在左边：



右浮动就是类似下面的效果，图片在右边：

47.CSS clear 属用于指定方向的浮动清除浮动

```
<html>

<head>

<style type="text/css">

img
{
float:left;
clear:both;

}

</style>
</head>

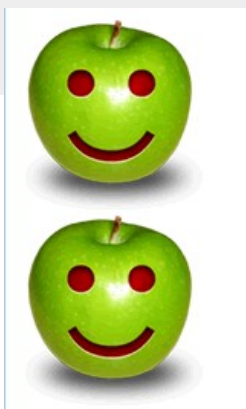

<body>




</body>

</html>
```

也就是说，图像的左侧和右侧均不允许出现浮动元素：效果如下：



48.通配符选择器，显示为一个星号（*）。该选择器可以与任何元素匹配，就像是一个通配符。

```
* {color:red;}
```

49.类选择器允许以一种独立于文档元素的方式来指定样式。该选择器可以单独使用，也可以与其他元素结合使用。假设有下面代码：

```
<h1 class="important">
This heading is very important.
</h1>

<p class="important">
This paragraph is very important.
</p>
```

使用下面选择器来选择所有类名相同的元素：

```
*.important {color:red;}
```

50.类选择器可以结合元素选择器来使用。例如，您可能希望只有段落显示为红色文本：

```
p.important {color:red;}
```

51.CSS 多类选择器：在 HTML 中，一个 class 值中可能包含一个词列表，各个词之间用空格分隔。

```
<html>
<head>
<style type="text/css">
.important {font-weight:bold;}
.warning {font-style:italic;}
.important.warning {background:silver;}
</style>
</head>
```

```
<body>
<p class="important">This paragraph is very important.</p>

<p class="warning">This is a warning.</p>

<p class="important warning">This paragraph is a very important warning.</p>

<p>This is a paragraph.</p>

<p>...</p>
</body>
</html>
```

执行效果如下：

This paragraph is very important.

This is a warning.

This paragraph is a very important warning.

This is a paragraph.

...

也就是说，class="important warning"一行，既应用了important的样式，也应用了warning的样式，还应用了important warning的样式。

52. 在一个HTML文档中，ID选择器会使用一次，而且仅一次。

53. 不同于类选择器，ID选择器不能结合使用，因为ID属性不允许有以空格分隔的词列表。

54. 简单属性选择：

(1) 把包含title的所有元素变为红色，可以写作：

```
*[title] {color:red;}
```

(2) 可以只对有href属性的a元素应用样式：

```
a[href] {color:red;}
```

(3) 还可以根据多个属性进行选择，只需将属性选择器链接在一起即可。例如，为了将同

时有 href 和 title 属性的 HTML 超链接的文本设置为红色，可以这样写：

```
a[href][title] {color:red;}
```

55.除了选择拥有某些属性的元素，还可以进一步缩小选择范围，只选择有特定属性值的元素。

```
a[href="http://www.w3school.com.cn/about_us.asp"] {color: red;}
```

56.相邻兄弟选择器使用了加号（+）表示，该 css 样式只应用在第二个元素上，也就是加号后面的元素，而不会应用到加号前面的样式：

```
h1 + p {margin-top:50px;}
```

选择紧接在 h1 元素后出现的段落，h1 和 p 元素拥有共同的父元素。例如：

```
<!DOCTYPE HTML>
<html>
<head>
<style type="text/css">
h1 + p {margin-top:50px;}
</style>
</head>

<body>
<h1>This is a heading.</h1>
<p>This is paragraph.</p>
<p>This is paragraph.</p>
<p>This is paragraph.</p>
<p>This is paragraph.</p>
<p>This is paragraph.</p>
</body>
</html>
```

执行结果为：

This is a heading.

This is paragraph.

This is paragraph.

This is paragraph.

This is paragraph.

This is paragraph.

57.CSS 伪类用于向某些选择器添加特殊的效果。伪类的语法：

```
p:first-child {font-weight: bold;}  
li:first-child {text-transform:uppercase;}
```

(1) 可以使用 :first-child 伪类来选择元素的第一个子元素

```
p:first-child {font-weight: bold;}  
li:first-child {text-transform:uppercase;}
```

第一个规则将作为某元素第一个子元素的所有 p 元素设置为粗体。第二个规则将作为某个元素第一个子元素的所有 li 元素变成大写。

58.CSS 伪元素用于向某些选择器设置特殊效果。伪元素的语法：

```
selector:pseudo-element {property:value;}
```

(1) :first-line 伪元素

用于向文本的首行设置特殊样式。

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

(2) :first-letter 伪元素

"first-letter" 伪元素用于向文本的首字母设置特殊样式：

```
p:first-letter  
{  
color:#ff0000;
```

```
font-size:xx-large;
}
```

(3) 多重伪元素

可以结合多个伪元素来使用。

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

p:first-line
{
  color:#0000ff;
  font-variant:small-caps;
}
```

(4) :before 伪元素

":before" 伪元素可以在元素的内容前面插入新内容。

(5) :after 伪元素

":after" 伪元素可以在元素的内容之后插入新内容。下面的例子在每个 <h1> 元素后面插入一幅图片：

```
h1:after
{
  content:url(logo.gif);
}
```

59. 设置字体大小时可以使用 em 单位代替像素，1em 和当前字体大小相等。在浏览器中默认的文字大小是 16px。

```
h1 {font-size:2.5em;}
h2 {font-size:1.875em;}
p {font-size:0.875em;}
```

60. css 中也可以定义下拉菜单。

- 在父元素中使用 “position: relative;”。

- 在子元素中使用 “position: absolute;” 。

```
<style>
.dropdown {
  position: relative;
  display: inline-block;
}

.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  padding: 12px 16px;
  z-index: 1;
}

.dropdown:hover .dropdown-content {
  display: block;
}
</style>

<div class="dropdown">
  <span>鼠标移过来</span>
  <div class="dropdown-content">
    <p>Hello World!</p>
  </div>
</div>
```

鼠标移动到文本“鼠标移过来”就会出现下拉菜单。

- 下拉菜单也可以使用 “a:hover” 等来改变样式。

61.visibility 属性指定一个元素是否是可见的。

```
h2
{
visibility:hidden;
}
```

隐藏 h2。

62.CSS 可以设置提示文本，当鼠标移动到指定位置时，弹出一段文本。

- 在父元素中使用 “position: relative;”
- 在子元素中使用 “position: absolute;”。
- 设置 hover 选择器，hover 选择器用于在鼠标移动到指定元素 <div> 时显示的提示。

```
<style>
/* Tooltip 容器 */
.tooltip {
  position: relative;
  display: inline-block;
  border-bottom: 1px dotted black; /* 悬停元素上显示点线 */
}

/* Tooltip 文本 */
.tooltip .tooltiptext {
  visibility: hidden;
  width: 120px;
  background-color: black;
  color: #fff;
  text-align: center;
  padding: 5px 0;
  border-radius: 6px;

  /* 定位 */
  position: absolute;
  z-index: 1;
}

/* 鼠标移动上去后显示提示框 */
.tooltip:hover .tooltiptext {
  visibility: visible;
}
</style>

<div class="tooltip">鼠标移动到这
  <span class="tooltiptext">提示文本</span>
</div>
```

63. 提示文本可以使用 left 或 right 来设置显示在左边还是右边。

```
<!DOCTYPE html>
```



```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<style>
.tooltip {
  position: relative;
  display: inline-block;
  border-bottom: 1px dotted black;
}

.tooltip .tooltiptext {
  visibility: hidden;
  width: 120px;
  background-color: black;
  color: #fff;
  text-align: center;
  border-radius: 6px;
  padding: 5px 0;

  /* 定位 */
  position: absolute;
  z-index: 1;
  top: -5px;
  left: 105%;
}

.tooltip:hover .tooltiptext {
  visibility: visible;
}
</style>
<body style="text-align:center;">

<div class="tooltip">鼠标移动到我这
  <span class="tooltiptext">提示文本</span>
</div>

</body>
```

</html>

显示效果如下:

鼠标移动到我这

提示文本

- 使用其中的“top: -5px;”是因为提示文本的顶部和底部的内边距是 5px。

64. 如果提示文本要显示在上面, 可以使用 bottom 和 margin-left 属性:

```
.tooltip .tooltiptext {  
  width: 120px;  
  bottom: 100%;  
  left: 50%;  
  margin-left: -60px; /* 使用一半宽度 (120/2 = 60) 来居中提示工具 */  
}
```

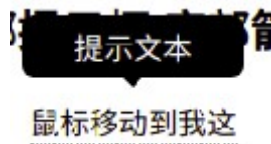
相反, 如果提示文本要显示在下面, 可以使用 top 和 margin-left 属性:

```
.tooltip .tooltiptext {  
  width: 120px;  
  top: 100%;  
  left: 50%;  
  margin-left: -60px; /* 使用一半宽度 (120/2 = 60) 来居中提示工具 */  
}
```

65. 我们可以用 CSS 伪元素 ::after 及 content 属性为提示文本创建一个小箭头标志, 箭头是由边框组成的。

```
.tooltip .tooltiptext::after {  
  content: "";  
  position: absolute;  
  top: 100%; /* 提示工具底部 */  
  left: 50%;  
  margin-left: -5px;  
  border-width: 5px;  
  border-style: solid;  
  border-color: black transparent transparent transparent;  
}
```

border-width 属性指定了箭头的大小。如果修改了它, 也要修改 margin-left 值。这样箭头在能居中显示。border-color 用于将内容转换为箭头。设置顶部边框为黑色, 其他是透明的。显示效果如下:



66. @media 规则允许在相同样式表中为不同设备设置不同的样式，调用形式如下：

```
@media screen
{
  p.test {font-family:verdana,sans-serif;font-size:14px;}
}
@media print
{
  p.test {font-family:times,serif;font-size:10px;}
}
@media screen,print
{
  p.test {font-weight:bold;}
```

其中的 screen 是设备种类，它有以下值：

- all：用于所有的媒体设备。
- aural：用于语音和音频合成器。
- braille：用于盲人用点字法触觉回馈设备。
- embossed：用于分页的盲人用点字法打印机。
- handheld：用于小的手持的设备。
- print：用于打印机。
- projection：用于方案展示，比如幻灯片。
- screen：用于电脑显示器。
- tty：用于使用固定密度字母栅格的媒体，比如电传打字机和终端。
- tv：用于电视机类型的设备。

67. viewport 是用户网页的可视区域。

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

- width=device-width：表示宽度是设备屏幕的宽度。
- initial-scale=1：表示初始的缩放比例。
- shrink-to-fit=no：自动适应手机屏幕的宽度。

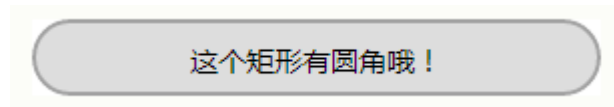
68.在前端设计的时候，往往需要先将文件分为几个大的<div>块，设计好总体布局，然后逐一在这些<div>块中添加内容。

CSS3

1. 在 CSS3 中，border-radius 属性用于创建圆角：

```
div
{
border:2px solid;
border-radius:25px;
-moz-border-radius:25px; /* Old Firefox */
}
```

效果类似这样：



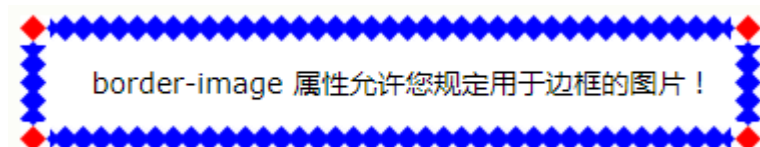
2. 在 CSS3 中，box-shadow 用于向方框添加阴影：

```
div
{
box-shadow: 10px 10px 5px #888888;
}
```

效果类似这样：



3. 通过 CSS3 的 border-image 属性，您可以使用图片来创建边框。

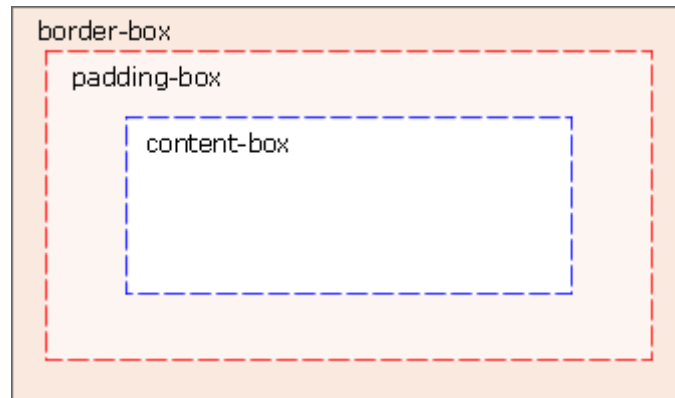


4. background-size 属性规定背景图片的尺寸。

```
div
{
background:url(bg_flower.gif);
}
```

```
background-size:63px 100px;
background-repeat:no-repeat;
}
```

5. background-origin 属性规定背景图片的定位区域。背景图片可以放置于 content-box、padding-box 或 border-box 区域。



例如：

```
div
{
background:url(bg_flower.gif);
background-repeat:no-repeat;
background-size:100% 100%;
-webkit-background-origin:content-box; /* Safari */
background-origin:content-box;
}
```

6. CSS3 允许您为元素使用多个背景图像。

```
body
{
background-image:url(bg_flower.gif),url(bg_flower_2.gif);
}
```

7. 在 CSS3 中，text-shadow 可向文本应用阴影。

文本阴影效果！

```
h1
{
```

```
text-shadow: 5px 5px 5px #FF0000;
}
```

8. word-wrap 属性允许您允许文本强制文本进行换行：

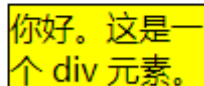
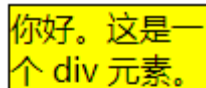
```
p {word-wrap:break-word;}
```

9. 2D 转换

(1) 通过 translate() 方法，元素从其当前位置移动，根据给定的 left（x 坐标）和 top（y 坐标）位置参数：

```
div
{
transform: translate(50px,100px);
-ms-transform: translate(50px,100px);          /* IE 9 */
-webkit-transform: translate(50px,100px); /* Safari and Chrome */
-o-transform: translate(50px,100px);          /* Opera */
-moz-transform: translate(50px,100px);        /* Firefox */
}
```

效果类似：



(2) 通过 rotate() 方法，元素顺时针旋转给定的角度。允许负值，元素将逆时针旋转。

```
div
{
transform: rotate(30deg);
-ms-transform: rotate(30deg);          /* IE 9 */
-webkit-transform: rotate(30deg);      /* Safari and Chrome */
-o-transform: rotate(30deg);          /* Opera */
-moz-transform: rotate(30deg);        /* Firefox */
}
```

```
}
```

效果类似：



(3) 通过 `scale()` 方法，元素的尺寸会增加或减少，根据给定的宽度和高度参数：

```
div
{
transform: scale(2,4);
-ms-transform: scale(2,4);    /* IE 9 */
-webkit-transform: scale(2,4); /* Safari 和 Chrome */
-o-transform: scale(2,4); /* Opera */
-moz-transform: scale(2,4);    /* Firefox */
}
```

效果类似：



(4) 通过 skew() 方法, 元素翻转给定的角度, 根据给定的水平线和垂直线参数:

```
div
{
transform: skew(30deg,20deg);
-ms-transform: skew(30deg,20deg); /* IE 9 */
-webkit-transform: skew(30deg,20deg); /* Safari and Chrome */
-o-transform: skew(30deg,20deg); /* Opera */
-moz-transform: skew(30deg,20deg); /* Firefox */
}
```

效果类似:



(5) matrix() 方法把所有 2D 转换方法组合在一起。

10.3D 转换:

(1) 通过 rotateX() 方法, 元素围绕其 X 轴以给定的度数进行旋转。

11.transition 属性用于实现过渡, 也就是类似动画的效果:

```
<!DOCTYPE html>
<html>
<head>
<style>
div
{
width:100px;
height:100px;
background:yellow;
transition:width 2s;
-moz-transition:width 2s; /* Firefox 4 */
-webkit-transition:width 2s; /* Safari and Chrome */
-o-transition:width 2s; /* Opera */
}
```



```

}

div:hover
{
width:300px;
}
</style>
</head>
<body>

<div></div>

```

当鼠标点到 div 后，图像宽度变为 300px，鼠标离开又恢复原样。

12. 多列

(1) column-count 属性规定元素应该被分隔的列数：

```

div
{
-moz-column-count:3; /* Firefox */
-webkit-column-count:3; /* Safari 和 Chrome */
column-count:3;
}

```

多列的效果类似：

注释：Internet Explorer 不支持 column-count 属性。

人民网北京2月24日电 (记者刘阳)国家发展改革委近日发出通知，决定自2月25日零时起将汽、柴油价格每吨分别提高300元和290元，折算到90号汽油和0号柴油（全国平均）每升零售价格分别提高0.22元和0.25元。此次国内成品油价格调整幅度，是按照现行国内形成机制，根据油价变化情况确定。2月16日国内成品油价格调整以来，受市场预期欧美经济复苏前景向好以及中东局势持续动荡等因素影响，国际

月上旬WTI和布伦特原油期货价格再次回升至每桶95美元和115美元以上。虽然近两日价格有所回落，但国内油价挂钩的国际市场三种原油连续22个工作日移动平均价格上涨幅度已超过4%，达到国内成品油价格调整的边界条件。通知指出，这次成

这个 div 元素可由用户调整尺寸（在 Firefox 4+、Chrome 以及 Safari 中）。

农村道路客运（含岛际和农村水路客运）等给予补贴。同时，为保证市场物价基本稳

民生活密切相关的铁路客运、城市公交、农村道路客运（含岛际和农村水路客运）价格不作调整。通知要求，中石油、中石化、中海油三大公司要组织好成品油生产和调运，保持合理库存，加强综合协调和应急调度，保障成品油供应。各级门要加大市场监管，依法查处不执行政策，以及囤积居奇、合谋涨价、搭车涨价等违法行为，维护正常市场秩序。

HTML 和 CSS 总结

1. 使用 “margin: 0 auto” 或 “margin: auto” 实现块元素水平居中，两者的实现效果是一样的。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>块元素水平居中</title>
    <style>
      #demo{
        width: 500px;
        margin: auto; /* 或者 margin: 0 auto; */
      }
    </style>
  </head>
  <body>

    <div id="demo">
      <p>恩，我就是那个需要水平居中的家伙。</p>
    </div>

  </body>
</html>
```

2. 要让图片居中对齐，可以使用 “margin: auto;”，并将它放到块元素中：

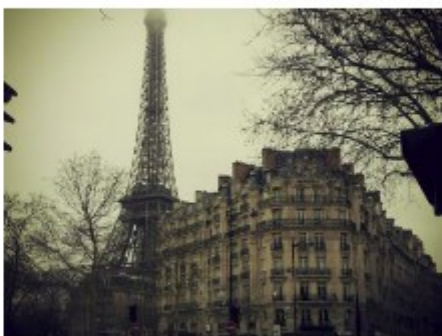
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<style>
img{
  display: block;
  margin: 0 auto;
}
</style>
</head>
```

```
<body>
<h2>图片居中对齐</h2>

</body>
</html>
```

显示效果如下：

图片居中对齐



3. 用 auto、100% 等非具体数字值修饰某个块元素时，相关的块的 width 或 height 通常需要是一个固定值，否则会出错。
- 100% 修饰块元素的 width 或 height：该块元素父元素以及之前的兄弟元素的 width 或 height 必须是一个固定的。

```
<html style="height=100%">
<body style="height=100%">
  <div style="height=100%">
    ...
  </div>
  <div class="footer" style="margin-top:-60px">
    ...
  </div>
</body>
</html>
```

注意，上面的例子中，<html>和<body>元素的 height 都设置为 100%，实际上这两个元素就是当前屏幕的高度，所以是个固定的，因此第一个 div 的 height 也是个固定值。

```
html, body {
  height: 100%;
}
```

- 将 margin 设置为 auto：该块元素 width 值必须是一个固定的。在上面的例子中：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>块元素水平居中</title>
    <style>
      #demo{
        width: 500px;
        margin: auto; /* 或者 margin: 0 auto; */
      }
    </style>
  </head>
  <body>
    <div id="demo">
      <p>恩，我就是那个需要水平居中的家伙。</p>
    </div>

  </body>
</html>
```

如果删掉“width: 500px;”，则居中就无法实现了，因为 width 不是固定值。

4. 有 2 种方法可以让两个 div 显示在同一行：

- 使用 display:

```
<div>
  <div style="display:inline;"></div>
  <div style="display:inline;"></div>
  <div style="display:inline;"></div>
</div>
```

需要一个父 div 和几个子 div、子 div 需要使用“display:inline;”来令其在同一行显示。

- 使用 float:

```
<div>
  <div style="float:left;"></div>
  <div style="float:left;"></div>
```

```
<div style="float:right;"></div>

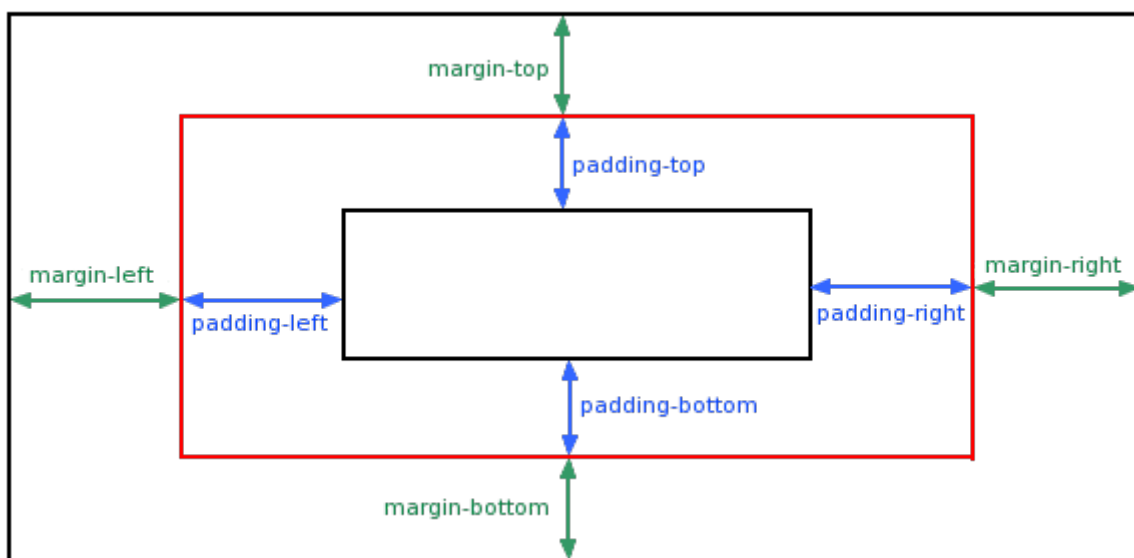
<!--必须清除浮动,才能换行-->
<div style="clear:both;"></div>

</div>
```

一个父 div，每个子 div 需要使用 float 来进行浮动。

➤ 两种方式都需要父 div 来包含子 div。

4. margin 用于定义元素的外边距，而 padding 用于定义元素的内边距。



5. 通过 padding 元素可以实现垂直居中，原理是将上下内边距设置为相等值：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<style>
.center {
padding: 70px 0;
border: 3px solid green;
}
</style>
</head>
<body>

<h2>垂直居中</h2>
```

<p>以上实例，我们使用 padding 属性实现元素的垂直居中：</p>

```
<div class="center">
  <p>我是垂直居中的。</p>
</div>

</body>
</html>
```

6. padding、margin 这类属性是一个简写属性，定义元素边框与元素内容之间的空间，即上下左右的内边距和外边距。

- padding:25px 50px 75px;
上填充为 25px，左右填充为 50px，下填充为 75px。
- padding:25px 50px;
上下填充为 25px，左右填充为 50px。
- padding:25px;
上下左右的填充都是 25px。

7. 对于块级元素来说，宽度设置为 auto，则会尽可能的宽。详细来说，元素宽度等于父级块宽度减外边距和内边距。

高度设置为 auto，则会尽可能的窄。详细来说，元素高度恰好足以包含其内联内容的高度。

8. 块级元素的 width 和 height 默认为 auto，所以块级元素 width 默认等于其父元素的 width，而 height 默认只够刚好包含其内容。

9. vertical-align 属性设置一个元素的垂直对齐方式，该属性只对行内元素有效，对块元素无效。

```
img
{
  vertical-align:text-top;
}
```

text-top 表示把元素的顶端与父元素字体的顶端对齐。

➤ 注意，居中的值是 middle，不是 center，也就是“vertical-align:middle”。

3. overflow 属性规定当内容溢出文本框时是否隐藏文本：

```
div
{
  width:150px;
```

```
height:150px;
overflow:scroll;
}
```

以滚动条的形式显示文本。

4. 文本框

```
<input style=" height:33px;" type="text" name="username" placeholder="请输入用户名" />
```

5. 等非块级元素要使用 style 时，不能直接在里面使用，而应该在外面加一层块级元素，因为很多属性只对块级元素起作用：

```
<p style="text-align:center;"></p>
```

6. 对于 class 和 id，推荐的做法是 class 用于定义样式，而 id 不添加样式，用于 js 操作 dom。

7. “Margin:0;padding:0” 用于消除不同的浏览器的默认值，进行浏览器兼容：

```
ul{
  margin:0;
  padding:0
}
```

8. css 代码优化的 12 个技巧：

- 避免过度约束：

```
// 糟糕
ul#someid {..}
.menu#otherid{..}

// 好的
#someid {..}
#otherid {..}
```

- 避免使用后代选择符。
- 避免链式（交集）选择符

```
// 糟糕
.menu.left.icon {..}
```

```
// 好的
.menu-left-icon {..}
```

- 使用复合（紧凑）语法

```
// 糟糕
.someclass {
padding-top: 20px;
padding-bottom: 20px;
padding-left: 10px;
padding-right: 10px;
background: #000;
background-image: url(../imgs/carrot.png);
background-position: bottom;
background-repeat: repeat-x;
}

// 好的
.someclass {
padding: 20px 10px 20px 10px;
background: #000 url(../imgs/carrot.png) repeat-x bottom;
}
```

9. CSS 样式表继承指的是，特定的 CSS 属性向下传递到子孙元素。

10. 大型网站往往需要一个比较好的组织结构，比如这种

base.css+common.css+page.css 组织方式：

- base 提供最底层的、功能和粒度最小的通用类样式，可以被所有页面引用，力求精简和通用；
- common 则是楼上提到的模块化组件，即将功能相对独立的划分为模块，重复次数多的模块划分为组件，提取到 common 层，这一层需要尽可能实现封装，对可能经常变化的部分提供灵活的接口；
- page 是页面级别的样式表，提供各页面独有的样式，可以再根据需求细分如 page-index.css, page-connect.css。

11. CSS 水平居中：

(1) 行内或类行内元素（比如文本和链接）：在块级父容器中让行内元素居中，只需使用 text-align: center;

(2) 块级元素：让块级元素居中的方法就是设置 margin-left 和 margin-right 为 auto（前提是已经为元素设置了适当的 width 宽度，否则块级元素的宽度会被拉伸为父级容器的宽度）。常见用法如下所示：

```
.center-me
{
  margin: 0 auto;
}
```

无论父级容器和块级元素的宽度如何变化，都不会影响块级元素的居中效果。请注意，float 属性是不能实现元素居中的。

12. HTML 块级元素和内联元素的区别：

- 块元素总是在新行上开始；内联元素和其他元素在一行；
- 块元素能容纳其他块元素或者内联元素；内联元素只能容纳文本或其他内联元素；
- 块元素中高度、行高以及顶和底边距都可以控制；内联元素中高、行高及顶和底边距不可改变。

13. css 样式 display:inline 的作用是设置对象做为行内元素显示，让块级元素转换为行内元素并显示。

14. 行内元素无法使用 margin 和 padding。

15. 在设置网页页脚时，通用的做法有几种：

(1) 借助 margin-top：

- ①. 将 html 标签和 body 的标签设置的 height 设置为 100%。
- ②. 将 body 的第一个 div 元素的 height 设置为 100%。
- ③. 新建一个该 div 元素的兄弟元素，通过设置 margin-top 为负数来完成页脚设置。

大概代码如下：

```
<html style=" height=100%" >
<body style=" height=100%" >
  <div style=" height=100%" >
    ...
  </div>
  <div class=" footer" style=" margin-top:-60px" >
    ...
  </div>
</body>
</html>
```

(2) 借助绝对定位和 “bottom:0;” :

①. 使用绝对定位。

②. html 和 html 子元素高为 100%。

③. 使用 bottom:0

```
<html>
<head>
<style>
html {
  height:100%;
}
.container {
  min-height:100%;
  position:relative;
}
.main {
  padding-bottom:100px; /* Height of the footer */
}
.footer {
  position:absolute;
  text-align:center;
  bottom:0;
  width:100%;
  height:30px;
}
</style>
</head>
<body>
  <div class="container">
    <div class="header"> 头部内容 </div>
    <div class="main"> 主框架 </div>
    <div class="footer"> 页脚 </div>
  </div>
</body>
</html>
```

注意：将 container 的 “min-height:100%” 修改为 “height:100%” 时，当页面出现滚动条时，页脚不会随着滚动条变化。只有 “min-height:100%” 时页脚才会随着滚动条变化。（min-height:100%、随滚动条而变化）

18. display 属性可以改变元素的显示方式，例如，将块元素改变为以行内元素显

示，将行内元素改变成以表格方式显示等。通过将该元素和 vertical-align 组合可以简单地实现将 li 元素居中显示：

```
.menu_head_ul li a{
  font-size:15px;
  color:#ffffff;
  vertical-align: middle;
  display: table-cell;
}
```

这个元素会作为一个表格单元格显示，类似 <td> 和 <th>。

19. 当容器的内容中有浮动的元素时，在这种情况下，容器的高度不能自动伸长以适应内容的高度，使得内容溢出到容器外面而影响（甚至破坏）布局的现象。这个现象叫浮动溢出，为了防止这个现象的出现而进行的 CSS 处理，就叫 CSS 清除浮动。使用” clear: both;” 清除浮动：

```
<div>
    <div style="float:left;"></div>
    <div style="float:left;"></div>
    <div style="float:right;"></div>

    <!--必须清除浮动,才能换行-->
    <div style="clear:both;"></div>
</div>
```

20. Html 的 label 标签和 span 标签差不多，不同之处在于，label 标签可以通过 for 属性和文本框、单选框、复选框等表单元素配对。

label 标签的 for 属性用于和表单元素关联，for 属性和表单元素的 id 属性相同，则代表两个标签是关联标签。

```
<html>
<body>

<p>请点击文本标记之一，就可以触发相关控件：</p>

<form>
<label for="male">Male</label>
<input type="radio" name="sex" id="male" />
<br />
<label for="female">Female</label>
<input type="radio" name="sex" id="female" />
```

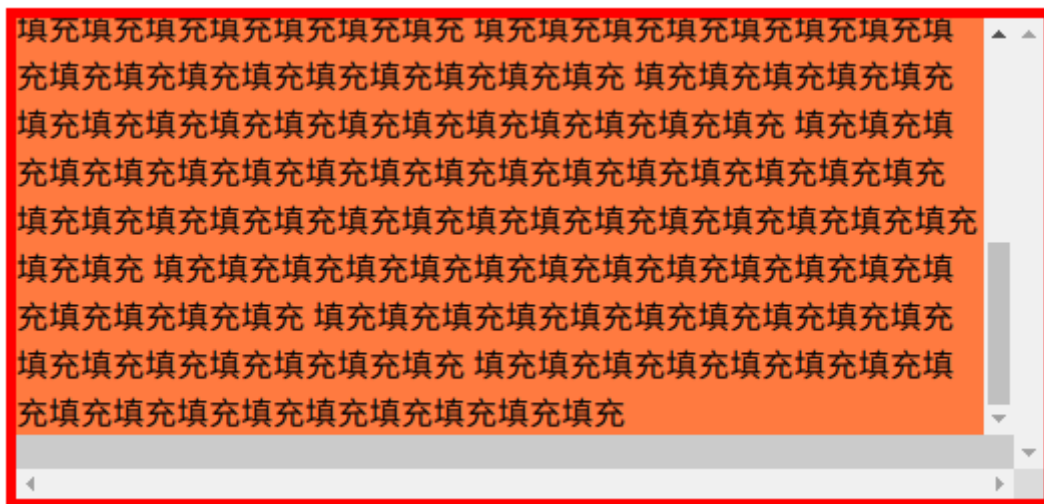
</html>

21. 要使用滚动条，需要两个条件：

- 填充填充填充填充填充填充填充填充填充填充填充填充填充填充填充
填充填充

```
.....  
    <span>填充填充填充填充填充填充填充填充填充填充填充填充填充填充填充  
填充填充</span>  
    </div>  
  
</div>  
  
</body>  
  
</html>
```

效果如下：



22. 实现 div 高度填满父容器剩余空间的原理其实就是除最后一个块之外的所有块的高度固定，则最后一个块可以实现填满父容器剩余空间：

(1) 最通俗的方式:

- ①. 父元素和第一个子元素的高度必须是固定。
- ②. 第一个子元素设置左浮动“float: left;”。
- ③. 第二个子元素的高度设置 100%。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>高度充满父容器</title>
  </head>
  <style>
    .parent {
      height: 500px;
      width: 300px;
```

```

border: 1px solid red;/***/
padding: 2px 2px;/***/
}

.nav {
height: 100px;
width: 100%;/*必须沾满宽度防止浮动*/
float: left;/*必须*/
background-color: red;
}

.content {
height:100%;/*必须*/
background-color: green;
}
</style>

<body>

<div class="parent">
  <div class="nav">
    固定高度
  </div>
  <div class="content">
    自适应父容器, 充满剩余的空间
  </div>
</div>

</body>

</html>

```

- .nav 固定高度 ,必须设置左浮动 float:left .且其宽度为 100%充满父容器宽度
- .content 设置 height:100% 就自适应充满剩余空间

(2) 绝对定位方式:

- ①. 跟使用绝对定位定义页脚的方式差不多。
- ②. 目录元素的 top 标签必须是之前已占用的 div 元素的高度, 防止覆盖了之前的元素。

```

.parent {
position: relative;

```

```

    height: 500px;
    width: 300px;
    border: 1px solid red;/***/
    padding: 2px 2px;/***/
}

.nav {
    height: 100px;
    width: 100%;
    background-color: red;
}

.content {
    position: absolute;
    top: 100px;
    bottom: 0px;

    background-color: green;
    width: 100%;

```

- 父容器设置 position: relative ,
- 需要自适应的容器设置 position: absolute 、 top: 100px (固定高度容器 .nav 的高度) , bottom: 0px (与父容器底部)。

23. 如果一个 div 元素需要自适应，填充父元素剩余的所有空间，则在其之前的所有元素大小必须是固定的。

24. 通过 和 来设置导航条：

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<style>
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;

```

```

}

li {
  float: left;
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

li a:hover {
  background-color: #111;
}
</style>
</head>
<body>

<ul>
  <li><a class="active" href="#home">主页</a></li>
  <li><a href="#news">新闻</a></li>
  <li><a href="#contact">联系</a></li>
  <li><a href="#about">关于</a></li>
</ul>

</body>
</html>

```

效果如下：

主页 新闻 联系 关于

25. Flex 布局又名弹性布局，用来为盒状模型提供最大的灵活性。盒状模型难以完成一些特殊的布局，例如垂直居中，Flex 布局就可以很简单地实现这种特殊的布局。

26. Flex 布局的属性分为容器和子项目两种，有些属性是设置在容器中的，有些属性是

设置在子项目中的：

```
<div class="box">
  <div class="item"></div>
  <div class="item"></div>
</div>
```

其中类名为 box 的 div 块就是容器，而类名为 item 的 div 就是子项目。

27. Flex 布局是按轴进行，分为主轴和交叉轴两种，默认的主轴为水平轴，交叉轴为垂直轴。

28. Flex 主轴和交叉轴的起点称为 start，而终点称为 end，默认是最左边和最上边。

29. Flex 布局设置在容器上的属性：

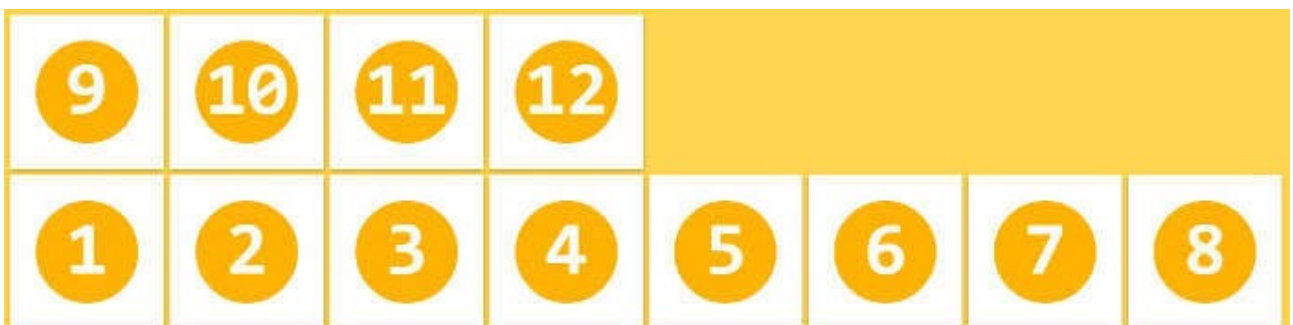
(1) flex-direction：设置主轴的方向，有 4 个值：

- row：默认值，主轴为水平方向，起点在左端。
- row-reverse：主轴为水平方向，起点在右端。
- column：主轴为垂直方向，起点在上沿。
- column-reverse：主轴为垂直方向，起点在下沿。



(2) flex-wrap：如果一条轴线排不下，如何换行。

- nowrap：默认值，表示不换行。
- wrap：换行，第一行在上方。
- wrap-reverse：换行，第一行在下方。



图中即为 wrap-reverse 的布局。

(3) flex-flow: flex-direction 属性和 flex-wrap 属性的简写形式，用法类似于 margin。

```
.parent {  
width: 200px;  
height: 150px;  
background-color: black;  
display: flex;  
flex-flow: row wrap;  
align-content: flex-start;  
}
```

(4) justify-content: 设置项目在主轴上的对齐方式，它可能取 5 个值。

- flex-start: 默认值，左对齐。
- flex-end: 右对齐。
- center: 居中。
- space-between: 两端对齐，项目之间的间隔都相等。
- 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。

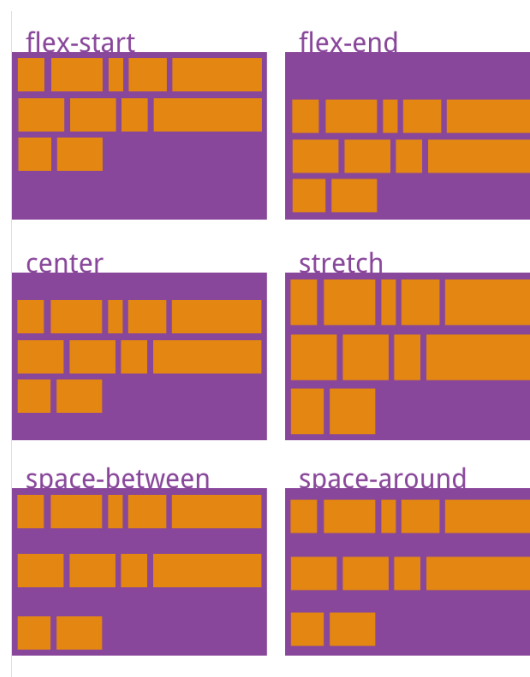
(5) align-items: 设置项目在交叉轴上对齐方式。

- stretch: 如果项目未设置高度或设为 auto，将占满整个容器的高度。
- flex-start: 交叉轴的起点对齐。
- flex-end: 交叉轴的终点对齐。
- center: 交叉轴的中点对齐。
- baseline: 项目的第一行文字的基线对齐。

(6) align-content: 设置多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

- stretch: 默认值，轴线占满整个交叉轴。
- flex-start: 与交叉轴的起点对齐。
- flex-end: 与交叉轴的终点对齐。
- center: 与交叉轴的中点对齐。
- space-between: 与交叉轴两端对齐，轴线之间的间隔平均分布。

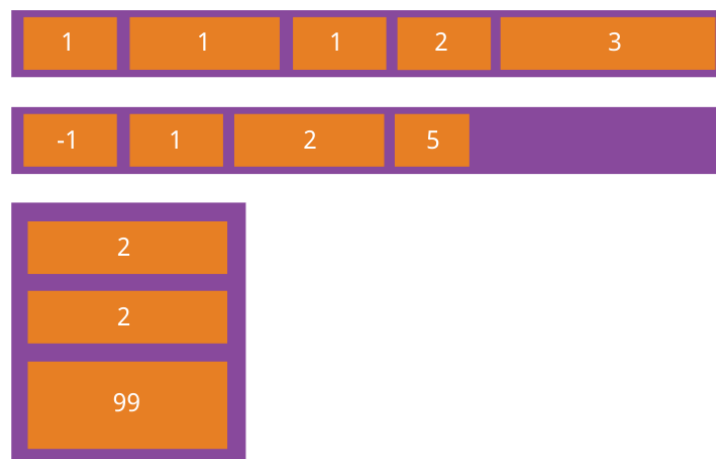
- `space-around`: 每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。



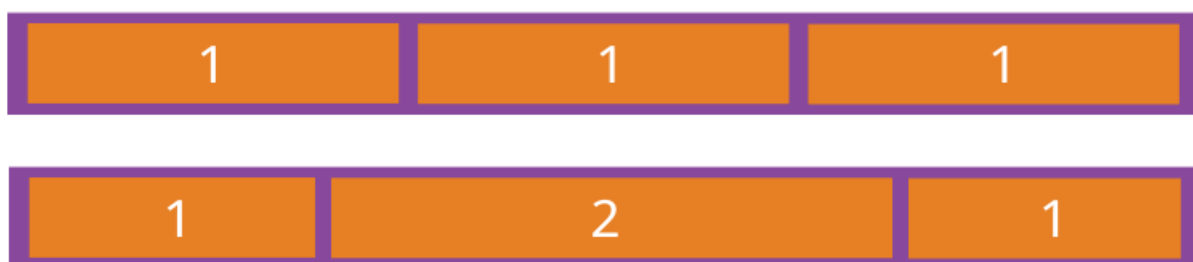
`align-content` 图示。

30. Flex 布局设置在子项目上的属性：

(1) `order`: 设置项目的排列顺序。数值越小，排列越靠前，默认为 0。图示如下



(2) `flex-grow`: 设置项目的放大比例，默认为 0，即如果存在剩余空间，也不放大。图示如下：



如果所有项目的 flex-grow 属性都为 1，则它们将等分剩余空间。如果一个项目的 flex-grow 属性为 2，其他项目都为 1，则前者占据的剩余空间将比其他项多一倍。

(3) flex-shrink: 设置项目的缩小比例，默认为 1，即如果空间不足，该项目将缩小。



如果所有项目的 flex-shrink 属性都为 1，当空间不足时，都将等比例缩小。如果一个项目的 flex-shrink 属性为 0，其他项目都为 1，则空间不足时，前者不缩小。

(4) flex-basis: 设置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，也可以设置为一个固定的数值，例如设置为和宽或高一样的数值，这样项目将占据固定空间。

```
.row{
  flex-basis: 100%;
  display:flex;
}
```

(5) flex: 是 flex-grow、flex-shrink 和 flex-basis 的简写，默认值为 0 1 auto。

(6) align-self: 允许单个项目有与其他项目不一样的对齐方式，可覆盖 align-items 属性。除了默认值 auto 之外，其取值和 align-items 属性完全一致。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<style>
.box{
  width:500px;
```

```
    height:100px;
    display: flex;
    margin:auto;
    align-items: center;
    justify-content: center;
    background-color:#005f87;
}

.item{
    width:30px;
    height:30px;
    background-color:#000000;
}
</style>
</head>
<body>
<div class="box">
  <div class="item"></div>
</div>
</body>
</html>
```

效果如下:



31.

JavaScript 入门集合

第 1 章 快速入门

1. `<script>`元素用于向 html 页面插入 JavaScript 代码。

```
<html>
```

```
<head>
  <script>
    alert('Hello, world');
  </script>
</head>
<body>
  ...
</body>
</html>
```

➤ `<script>` 标签有一个 `type` 属性，默认值为 `text/javascript`，所以使用时可以省略。

2. 通过 `<script>` 标签的 `src` 属性指定外部 JavaScript 脚本：

```
<script type="text/javascript" defer="defer" src="example2.js"></script>
```

3. `<script>` 标签如果定义了 `src` 属性，则不应该在其 `<script>` 和 `</script>` 标签之间再包含额外的 JavaScript 代码，否则只会下载并执行脚本文件里的代码，嵌入的代码会被忽略。

4. `<script>` 标签的 `defer` 属性用于表明脚本在整个页面加载完成之后才执行。

```
<!DOCTYPE html>
<html>
<head>
<title>Example HTML Page</title>
<script type="text/javascript" defer="defer" src="example1.js"></script>
<script type="text/javascript" defer="defer" src="example2.js"></script>
</head>
<body>
<!-- 这里放内容 -->
</body>
</html>
```

5. 按照惯例，所有 `<script>` 元素都应该放在页面的 `<head>` 元素中，例如：

```
<!DOCTYPE html>
<html>
<head>
<title>Example HTML Page</title>
<script type="text/javascript" src="example1.js"></script>
<script type="text/javascript" src="example2.js"></script>
</head>
```

```
<body>
<!-- 这里放内容 -->
</body>
</html>
```

在<head>元素中的所有的 JavaScript 文件，必须等到全部 JavaScript 代码都被下载、解析和执行完成以后，才能开始显示 html 页面。如果在<head>元素中的 JavaScript 代码出现 getElementById 之类的选择器的内容会出错，因为此时的 DOM 树还没构建。

为了避免这个问题，Web 应用程序一般都把全部 JavaScript 引用放在<body>元素中，放在页面的内容后面，如下例所示：

```
<!DOCTYPE html>
<html>
<head>
<title>Example HTML Page</title>
</head>
<body>
<!-- 这里放内容 -->
<script type="text/javascript" src="example1.js"></script>
<script type="text/javascript" src="example2.js"></script>
</body>
</html>
```

6. 如果浏览器不支持或禁用了 JavaScript，使用<noscript>标签在浏览器中显示替代的内容。

```
<html>
<head>
<title>Example HTML Page</title>
<script type="text/javascript" defer="defer" src="example1.js"></script>
<script type="text/javascript" defer="defer" src="example2.js"></script>
</head>
<body>
<noscript>
<p>本页面需要浏览器支持（启用）JavaScript。
</noscript>
</body>
</html>
```

7. JavaScript 的语法和 Java 语言类似，每个语句以分号结束，语句块用“{...}”。但是，JavaScript 并不强制要求在每个语句的结尾加分号，浏览器中负责执行

JavaScript 代码的引擎会自动在每个语句的结尾补上分号。

- 让 JavaScript 引擎自动加分号在某些情况下会改变程序的语义，导致运行结果与期望不一致。
- 8. 浏览器不会检查 JavaScript 文件的扩展名，所以 JS 文件的可以为任意扩展名，习惯上定义为 .js 扩展名。
- 9. JS 注释有两种：
 - 单行注释//。
 - 多行注释/*...*/

```
/* 从这里开始是块注释  
仍然是注释  
仍然是注释  
注释结束 */
```

10. JS 的数据类型

- Boolean：布尔值只有 true、false 两种值。
- Number：JavaScript 不区分整数和浮点数，统一用 Number 表示。
- String：字符串是以单引号'或双引号"括起来的任意文本，比如'abc'，"xyz"等等。
- Null：只有一个值 null。null 值表示一个空对象指针，所以使用 typeof 操作符检测 null 值时会返回"object"

```
var car = null;  
alert(typeof car);  
    输出 object。
```

- Undefined 类型只有一个值，即 undefined。未初始化的变量默认值就是 undefined，例如：

```
var message;  
alert(message == undefined); //true
```

- JavaScript 的设计者希望用 null 表示一个空的值，而 undefined 表示值未定义。事实证明，这并没有什么卵用，区分两者的意义不大。大多数情况下，我们都应该用 null。undefined 仅仅在判断函数参数是否传递的情况下有用。

11. 有两种比较运算符：

- 相等运算符“==”：它会自动转换数据类型再比较，很多时候，会得到非常诡异的结果；
- 全等运算符“===”：它不会自动转换数据类型，如果数据类型不一致，返回 false，如果一致，再比较。

12. Number 类型有个特殊的 NaN 与所有其他值都不相等，包括它自己：

```
NaN === NaN;           // false
```

通过 isNaN() 函数来判断其值是否为 NaN：

```
isNaN(NaN);           // true
```

13. 浮点数在运算过程中会产生误差，所以带运算的浮点数在比较时需要注意：

```
1 / 3 === (1 - 2 / 3); // false
```

要比较两个浮点数是否相等，只能计算它们之差的绝对值，看是否小于某个阈值：

```
Math.abs(1 / 3 - (1 - 2 / 3)) < 0.0000001; // true
```

14. 有 3 个函数可以把非数值转换为数值：

- Number()：把对象的值转换为数字。
- parseInt()：把字符串转换为整形。
- parseFloat()：把字符串转换为浮点型。

```
var num1 = Number("Hello world!"); //NaN
var num2 = Number(""); //0
var num3 = Number("000011"); //11
var num4 = Number(true); //1
```

15. 创建数组的方式有两种。

- 使用 Array 类：

```
var colors = new Array();
```

在使用 Array 构造函数时也可以省略 new 操作符。

- 数组字面量表示法：

```
var colors = ["red", "blue", "green"];
var names = [];
```

16. JavaScript 的对象是一组由键值对组成的无序集合，键都是字符串类型，值可以是任意数据类型。

```
var person = {
  name: 'Bob',
  age: 20,
  tags: ['js', 'web', 'mobile'],
```

```
city: 'Beijing',  
hasCar: true,  
zipcode: null  
};
```

17.JavaScript 可以使用点表示法和方括号表示法访问对象属性:

```
alert(person["name"]);  
alert(person.name);
```

18.JS 的变量规则是由字母、数字、下划线和美元符组成,且不能用数字开头。

19.按照惯例,ECMAScript 标识符采用驼峰大小写格式,也就是第一个单词小写,剩下的每个有意义的单词的首字母大写,例如:

```
firstSecond  
myCar  
doSomethingImportant
```

20.ECMAScript 使用 C 风格的注释,包括单行注释和块级注释。单行注释以两个斜杠开头,块级注释以一个斜杠和一个星号 (/*) 开头,以一个星号和一个斜杠 (*/) 结尾。

21.JS 使用 var 来定义变量:

```
var a;  
var $b = 1;  
var s_007 = '007';  
var Answer = true;  
var t = null;
```

➤ JavaScript 是弱类型语言,不区别整形浮点型之类的。

22.如果字符串本身包含引号,则用另一种引号将字符串括起来:

```
"I'm OK"
```

字符串本身包含单引号,所以用双引号括起来。

23.如果字符串内部既包含'又包含"怎么办? 可以用转义字符\来标识,比如:

```
I\'m \"OK\"!;
```

表示 I'm "OK"!

24.使用反引号来定义多行字符串。

```
`这是一个`
```

多行
字符串`;

例如:

```
console.log(`多行  
字符串  
测试`);
```

25. 要把多个字符串连接起来, 可以用+号连接:

```
var name = '小明';  
var age = 20;  
var message = '你好, ' + name + ', 你今年' + age + '岁了!';  
alert(message);
```

输出“你好, 小明, 你今年 20 岁了!”。

26. 如果有很多变量需要连接, 用+号就比较麻烦。ES6 新增了一种模板字符串, 可以使用\${}来引用变量:

```
var name = '小明';  
var age = 20;  
var message = `你好, ${name}, 你今年${age}岁了!`;   
alert(message);
```

27. 字符串可以通过索引来访问某个字符:

```
var s = 'Hello, world!';  
s[0]; // 'H'
```

28. 字符串的 length 属性保存字符串的长度:

```
var s = 'Hello, world!';  
s.length;
```

输出为 13。

29. 字符串是不可变的, 如果对字符串的某个索引赋值, 不会有任何错误, 但是, 也没有任何效果:

```
var s = 'Test';  
s[0] = 'X';  
alert(s);
```

输出仍然是 Test。

30. 字符串函数：

- toUpperCase()：把一个字符串全部变为大写。
- toLowerCase()：把一个字符串全部变为小写。
- indexOf()：会搜索指定字符串出现的位置。
- substring()：返回指定索引区间的子串。

31. 要取得 Array 的长度，直接访问 length 属性：

```
var arr = [1, 2, 3.14, 'Hello', null, true];  
arr.length;
```

输出 6。

➤ 直接给 Array 的 length 属性赋值会导致 Array 大小的变化：

```
var arr = [1, 2, 3];  
arr.length;           // 输出 3  
arr.length = 6;  
arr;                   // arr 变为[1, 2, 3, undefined, undefined, undefined]  
arr.length = 2;  
arr;                   // arr 变为[1, 2]
```

32. Array 可以通过索引把对应的元素修改为新的值：

```
var arr = ['A', 'B', 'C'];  
arr[1] = 99;  
arr;                   // arr 现在变为['A', 99, 'C']
```

➤ 请注意，如果通过索引赋值时，索引超过了范围，也会改变 Array 的 length 属性的大小。

33. Array 也可以通过 indexOf() 来搜索一个指定的元素的位置。

34. Array 的 slice() 就是对应 String 的 substring() 版本，它截取 Array 的部分元素，然后返回一个新的 Array

35. Array 的 push() 向 Array 的末尾添加若干元素，pop() 则把 Array 的最后一个元素删除掉。

36. 如果要往 Array 的头部添加若干元素，使用 unshift() 方法，shift() 方法则把 Array 的第一个元素删掉。

37. sort() 可以对当前 Array 进行排序，它会直接修改当前 Array 的元素位置，直接调用时，按照默认升序排序。

38. reverse() 把整个 Array 元素的顺序进行反转。

```
var arr = ['one', 'two', 'three'];
arr.reverse();
arr;
```

arr 数组变为['three', 'two', 'one']。

39.concat()方法把当前的 Array 和另一个 Array 连接起来，并返回一个新的 Array。

```
var arr = ['A', 'B', 'C'];
var added = arr.concat([1, 2, 3]);
added;    // ['A', 'B', 'C', 1, 2, 3]
arr;      // ['A', 'B', 'C']
```

40.join()方法是一个非常实用的方法，它把当前 Array 的每个元素都用指定的字符串连接起来，然后返回连接后的字符串。

```
var arr = ['A', 'B', 'C', 1, 2, 3];
arr.join('-');    // 'A-B-C-1-2-3'
```

41.JS 支持多维数组：

```
var arr = [[1, 2, 3], [400, 500, 600], '-'];
```

42.JavaScript 用一个{...}表示一个对象，键值对以“xxx: xxx”形式申明，用逗号隔开。

```
var xiaoming = {
  name: '小明',
  birth: 1990,
  school: 'No.1 Middle School',
  height: 1.70,
  weight: 65,
  score: null
};
```

注意，最后一个键值对不需要在末尾加逗号，如果加了，有的浏览器（如低版本的 IE）将报错。

43.通过.操作符访问属性，如果属性名包含特殊字符，就必须用引号括起来。

```
var xiaohong = {
  name: '小红',
  'middle-school': 'No.1 Middle School'
};
```

xiaohong 的属性名 middle-school 不是一个有效的变量，就需要用"括起来。访问

这个属性也无法使用.操作符，必须用['xxx']来访问：

```
xiaohong['middle-school'];    // 'No.1 Middle School'
xiaohong['name'];              // '小红'
xiaohong.name;                // '小红'
```

44. JavaScript 对象的所有属性都是字符串，不过属性对应的值可以是任意数据类型。

45. 为了消除 Javascript 语法的一些不合理之处，ECMAScript 5 引入了严格模式的概念。启用 strict 模式的方法是在 JavaScript 代码的第一行写上：

```
'use strict';
```

例如如果不启用严格模式，定义变量时省略 var 默认是全局变量，如果忘记写 var，可能会造成难以理解的错误。

46. 所有的 JavaScript 代码都应该使用 strict 模式。

47. JS 访问不存在的属性不报错，而是返回 undefined。

```
'use strict';

var xiaoming = {
  name: '小明'
};

console.log(xiaoming.name);
console.log(xiaoming.age); // undefined
```

输出：

```
小明
undefined
```

48. JavaScript 的对象是动态类型，所以可以自由地给一个对象添加或删除属性：

```
var xiaoming = {
  name: '小明'
};
xiaoming.age;      // undefined
xiaoming.age = 18;  // 新增一个 age 属性
xiaoming.age;      // 18
delete xiaoming.age; // 删除 age 属性
xiaoming.age;      // undefined
```

49.要检测对象是否拥有某一属性，可以用 in 操作符：

```
var xiaoming = {
  name: '小明',
  birth: 1990,
  school: 'No.1 Middle School',
  height: 1.70,
  weight: 65,
  score: null
};
'name' in xiaoming;      // true
'grade' in xiaoming;     // false
```

➤ 不过要注意，即使是继承得到的属性，in 返回真。

50.hasOwnProperty()方法用于判断对象自身是否拥有某个属性，继承链上得到的不算在内。

51.JavaScript 的循环控制语句和 java 的循环控制语句基本相同，例如使用 if 语句来进行条件判断。

```
if (age >= 6) {
  console.log('teenager');
} else if (age >= 18) {
  console.log('adult');
} else {
  console.log('kid');
}
```

52.JavaScript 的循环有两种，

- for 循环：

```
var x = 0;
var i;
for (i=1; i<=10000; i++) {
  x = x + i;
}
```

- for ... in 循环：它可以把一个对象的所有属性依次循环出来。

```
var o = {
  name: 'Jack',
  age: 20,
  city: 'Beijing'
};
```

```
for (var key in o) {  
  console.log(key);  
}
```

输出:

```
name  
age  
city
```

➤ 注意: for ... in 和 for ...of 中的 in 和 of 都是在括号里面的。

53.Array 也是对象, 而它的每个元素的索引被视为对象的属性, 因此, for ... in 循环可以直接循环出 Array。

```
var a = ['A', 'B', 'C'];  
for (var i in a) {  
  console.log(i);  
  console.log(a[i]);  
}
```

输出:

```
0  
A  
1  
B  
2  
C
```

54.while 循环:

```
var x = 0;  
var n = 99;  
while (n > 0) {  
  x = x + n;  
  n = n - 2;  
}
```

55.do ... while 种循环:

```
var n = 0;  
do {  
  n = n + 1;  
} while (n < 100);
```


56.label 语句用于在代码中添加标签，以便将来使用。以下是 label 语句的语法：

```
label: statement
```

示例：

```
start: for (var i=0; i < count; i++) {  
  alert(i);  
}
```

57.break 和 continue 语句用于在循环中精确地控制代码的执行。其中，break 语句会立即退出循环，强制继续执行循环后面的语句。而 continue 语句虽然也是立即退出循环，但退出循环后会从循环的顶部继续执行。

58.break 和 continue 语句都可以与 label 语句结合使用，从而返回代码中特定的位置。这种情况多发生在循环嵌套的情况下，如下面的例子所示：

```
var num = 0;  
  
outermost:  
for (var i=0; i < 10; i++) {  
  for (var j=0; j < 10; j++) {  
    if (i == 5 && j == 5) {  
      break outermost;  
    }  
    num++;  
  }  
}  
  
alert(num); //55
```

59.switch 语句的语法与 C 语言非常接近，如下所示：

```
switch (expression) {  
  case value: statement  
    break;  
  case value: statement  
    break;  
  case value: statement  
    break;  
  case value: statement  
    break;  
  default: statement
```

```
}
```

60.

第2章 函数

1. JS 使用 function 来定义函数，定义方式有两种：

- 普通函数：

```
function abs(x) {  
  if (x >= 0) {  
    return x;  
  } else {  
    return -x;  
  }  
}
```

- 匿名函数：

```
var abs = function (x) {  
  if (x >= 0) {  
    return x;  
  } else {  
    return -x;  
  }  
};
```

将匿名函数赋值给一个变量，然后通过变量来调用。

- 匿名函数也称函数表达式。

2. 函数不支持重载。

3. 声明提升是 JS 的一个重要特征，有了声明提升，可以让函数或变量实现“先使用，再定义”的效果。JS 在执行代码之前会先读取所有的函数或变量声明。变量或函数的声明会被提升到作用域的最前面：

```
sayHi();  
function sayHi(){  
  alert("Hi!");  
}
```

这个例子不会抛出错误，因为在代码执行之前会先读取函数声明。

4. 函数声明使用前不需要先赋值，但函数表达式在使用前必须先赋值。以下代码会导致错误。

```
sayHi(); //错误：函数还不存在
var sayHi = function(){
    alert("Hi!");
};
```

5. 通过在匿名函数后面加个括号传递参表示创建立即执行的匿名函数：

```
console.log((function (x) { return x * x }) (3));
```

如果单独定义的匿名函数单独一行，需要加括号：

```
(function (x) { return x * x }) (3);
```

6. 通过函数名或函数变量名来调用函数：

```
abs(10); // 返回 10
abs(-9); // 返回 9
```

7. JavaScript 允许传入任意数量的参数而不影响调用，因此传入的参数比定义的参数多也没有问题，虽然函数内部并不需要这些参数。

```
abs(10, 'blablabla'); // 返回 10
abs(-9, 'haha', 'hehe', null); // 返回 9
```

当然，传入的参数也可以比定义的少：

```
abs(); // 返回 NaN
```

8. JS 有一个关键字 `arguments`，它只在函数内部起作用，并且永远指向当前函数的调用者传入的所有参数。`arguments` 类似 `Array`，但不是一个 `Array`：

```
function foo(x) {
    console.log('x = ' + x); // 10
    for (var i=0; i<arguments.length; i++) {
        console.log('arg ' + i + ' = ' + arguments[i]); // 10, 20, 30
    }
}
foo(10, 20, 30);
```

输出：

```
x = 10
arg 0 = 10
arg 1 = 20
arg 2 = 30
```

9. 利用 `arguments` 可以获得调用者传入的所有参数。也就是说，即使函数不定义任何参数，还是可以拿到参数的值：

```
function abs() {  
  if (arguments.length === 0) {  
    return 0;  
  }  
  var x = arguments[0];  
  return x >= 0 ? x : -x;  
}
```

```
abs();           // 0  
abs(10);         // 10  
abs(-9);         // 9
```

10. ES6 标准引入了 `rest` 参数，它是个数组，用来获取传参时比定义时多出来的参数。`rest` 参数只能写在最后，前面用 `...` 标识：

```
function foo(a, b, ...rest) {  
  console.log('a = ' + a);  
  console.log('b = ' + b);  
  console.log(rest);  
}
```

```
foo(1, 2, 3, 4, 5);  
foo(1);
```

输出：

```
a = 1  
b = 2  
3,4,5  
a = 1  
b = undefined
```

11. 如果传入的参数连正常定义参数都没填满，`rest` 参数会接收一个空数组。
12. 如果 `return` 后面为空，JS 默认会加分号，导致出现意想不到的结果。

```
function foo() {  
  return
```

```
{ name: 'foo' };  
}
```

```
foo();
```

此时输出为 `undfine`，因为 JS 自动在 `return` 后面添加分号，返回值为空。可以像下面这样修改：

```
function foo() {  
  return {  
    name: 'foo'  
  };  
}
```

在 `return` 后面加个大括号表示语句还没结束。

13. 如果一个变量在函数体内部申明，则该变量是局部变量，作用域为整个函数体，在函数体外不可引用该变量：

```
'use strict';
```

```
function foo() {  
  var x = 1;  
  x = x + 1;  
}
```

14. JavaScript 的函数可以嵌套，此时，内部函数可以访问外部函数定义的变量，反过来则不行。

15. JavaScript 函数的变量提升指的是先扫描整个函数体的语句，把所有声明、定义语句“提升”到函数顶部：

```
'use strict';
```

```
function foo() {  
  var x = 'Hello, ' + y;  
  console.log(x);  
  var y = 'Bob';  
}
```

```
foo();
```

`y` 变量在函数最后定义，“`var x = 'Hello, ' + y`”也没有报错，因为“`var y = 'Bob'`”被提升到了函数前面。

16. JavaScript 默认有一个全局对象 window，全局作用域的变量实际上被绑定到 window 的一个属性。

```
'use strict';

var course = 'Learn JavaScript';
alert(course);           // 'Learn JavaScript'
alert(window.course);    // 'Learn JavaScript'
```

17. 匿名函数中的变量函数实际上也是一个全局变量，所以可以通过 window 来调用，但普通函数不行：

```
'use strict';

foo=function () {
    alert('foo');
}

window.foo();
```

18. JS 的命名空间可以减少命名冲突，命名空间实际上就是一个全局变量：

```
var MYAPP = {};

MYAPP.name = 'myapp';
MYAPP.version = 1.0;

MYAPP.foo = function () {
    return 'foo';
};
```

定义命名空间 MYAPP。

19. 关键字 let 用于定义作用域为语句块的变量：

```
'use strict';

function foo() {
    for (let i=0; i<10; i++) {

    }
    console.log(i);
```

```
}  
foo();
```

输出为 0，如果将 let 修改为 var，则输出为 10。

20. 从 ES6 开始，JavaScript 引入了解构赋值，可以同时为一组变量进行赋值，就是把一个数组的元素分别赋值给几个变量：

```
var [x, y, z] = ['hello', 'JavaScript', 'ES6'];
```

21. 如果数组本身还有嵌套，也可以通过下面的形式进行解构赋值，注意嵌套层次和位置要保持一致：

```
let [x, [y, z]] = ['hello', ['JavaScript', 'ES6']];
```

22. 解构赋值还可以忽略某些元素：

```
let [, , z] = ['hello', 'JavaScript', 'ES6'];
```

忽略前两个元素，只对 z 赋值第三个元素。

23. 解构赋值可以用于从一个对象中取出若干属性：

```
'use strict';  
  
var person = {  
  name: '小明',  
  age: 20,  
  gender: 'male',  
  passport: 'G-12345678',  
  school: 'No.4 middle school'  
};  
var {name, age, passport} = person;
```

➤ 解构赋值在对象中要用大括号。

24. 对一个对象进行解构赋值时，同样可以直接对嵌套的对象属性进行赋值，只要保证对应的层次是一致的：

```
var person = {  
  name: '小明',  
  age: 20,  
  gender: 'male',  
  passport: 'G-12345678',  
  school: 'No.4 middle school',  
  address: {
```

```
    city: 'Beijing',
    street: 'No.1 Road',
    zipcode: '100001'
  }
};
var {name, address: {city, zip}} = person;
```

25. 解构赋值时使用等于 (=) 定义默认值，这样就避免了不存在的属性返回 undefined 的问题：

```
var person = {
  name: '小明',
  age: 20,
  gender: 'male',
  passport: 'G-12345678'
};

var {name, single=true} = person;
```

single 的默认值为 true。

26. 以大括号开头的语句会被 JS 解析为语句块，语句块单独成行会报错，在解构赋值是加上圆括号解决这个问题：

```
var x, y;
({x, y} = { name: '小明', x: 100, y: 200});
```

省略圆括号会报错。

27. 解构赋值在很多时候可以大大简化代码。例如，交换两个变量 x 和 y 的值，可以这么写，不再需要临时变量：

```
var x=1, y=2;
[x, y] = [y, x]
```

28. 使用下列方式在对象中定义方法：

```
var xiaoming = {
  name: '小明',
  birth: 1990,
  age: function () {
    var y = new Date().getFullYear();
    return y - this.birth;
  }
}
```



```
};
```

29.调用对象方法时如果不加括号，则输出方法的定义，加括号才输出方法结果：

```
var xiaoming = {  
  name: '小明',  
  birth: 1990,  
  age: function () {  
    var y = new Date().getFullYear();  
    return y - this.birth;  
  }  
};  
  
console.log(xiaoming.age);  
console.log("-----");  
console.log(xiaoming.age());
```

输出：

```
function () {  
  var y = new Date().getFullYear();  
  return y - this.birth;  
}  
-----  
29
```

30.对象方法有 this 指针。

31.只有通过 obj.xxx()调用的函数，this 才能指向 obj 对象：

```
function getAge() {  
  var y = new Date().getFullYear();  
  return y - this.birth;  
}  
  
var xiaoming = {  
  name: '小明',  
  birth: 1990,  
  age: getAge  
};  
  
console.log(getAge());
```

这个程序输出 NaN，因为 this 指向错误。

32. 在对象的嵌套方法里，内层函数的 this 指针为 undefined。

```
'use strict';

var xiaoming = {
  name: '小明',
  birth: 1990,
  age: function () {
    function getAgeFromBirth() {
      var y = new Date().getFullYear();
      return y - this.birth;
    }
    return getAgeFromBirth();
  }
};
xiaoming.age();
```

输出 NaN，因为内层函数 this 指针为空。

33. 在对象的外层函数中使用一个变量捕获 this，可以传入内层函数中。

```
'use strict';

var xiaoming = {
  name: '小明',
  birth: 1990,
  age: function () {
    var that = this; // 在方法内部一开始就捕获 this
    function getAgeFromBirth() {
      var y = new Date().getFullYear();
      return y - that.birth; // 用 that 而不是 this
    }
    return getAgeFromBirth();
  }
};

console.log(xiaoming.age());
```

输出 29。

34. 用函数本身的 apply 方法可以指定函数的 this 指向哪个对象，它接收两个参数，第一个参数就是需要绑定的 this 变量，第二个参数是 Array，表示函数本身的参数：

```
function getAge() {
  var y = new Date().getFullYear();
  return y - this.birth;
}

var xiaoming = {
  name: '小明',
  birth: 1990,
  age: getAge
};

console.log(xiaoming.age());
console.log(getAge.apply(xiaoming, []));
```

getAge 函数内部的 this 指向 xiaoming，参数为空。也就是说，只要是在 getAge 函数内部出现 this，这个 this 就是指 xiaoming，而不是 getAge() 本身，this.name 就是指小明。

➤ apply() 和 call() 方法的作用是调用一个方法，并修改该方法的 this 指向。

35. 另一个与 apply() 类似的方法是 call()，唯一区别是：

- apply()：将参数打包成数组，再传入。
- call()：单独按顺序传入参数。

```
Math.max.apply(null, [3, 5, 4]);
Math.max.call(null, 3, 5, 4);
```

对普通函数调用，通常把 this 绑定为 null。

36. JavaScript 的所有对象都是动态的，即使内置的函数，我们也可以重新指向新的函数。

```
'use strict';

var count = 0;
var oldParseInt = parseInt; // 保存原函数

window.parseInt = function () {
  count += 1;
  return oldParseInt.apply(null, arguments); // 调用原函数
};

parseInt('10');
parseInt('20');
```

```
parseInt('30');  
console.log('count = ' + count); // 3
```

通过修改 this 指针来统计调用了多少次 parseInt()。

37.高阶函数其实就是接收回调函数为参数的函数：

```
function add(x, y, f) {  
  return f(x) + f(y);  
}
```

38.Array 的 map() 方法接收一个函数为参数，该数组的每个值都调用该函数，将结果组成一个数组，然后返回：

```
'use strict';  
  
function pow(x) {  
  return x * x;  
}  
  
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];  
var results = arr.map(pow);  
console.log(results);
```

将 arr 数组每个元素都调用 pow()，将结果组成一个数组然后返回。输出：

```
1,4,9,16,25,36,49,64,81
```

39.map() 不会对空数组进行检测，也不会改变原始数组。

40.Array 的 reduce() 方法接收一个回调函数，使用 Array 中的每个值都作为参数传入该回调函数进行计算，最后的返回值为单个值。

```
const array1 = [1, 2, 3, 4];  
const reducer = (accumulator, currentValue) => accumulator - currentValue;  
  
console.log(array1.reduce(reducer));
```

输出-8。

➤ reduce() 对于空数组是不会执行回调函数的。

41.reduce 的回调函数在两个必选参数：

- accumulator：累计器，不需要赋值，默认值为 0，计算之后的累计值，它是上一次调用回调时返回的累积值。

- currentValue: 数组中正在处理的元素。

```
const array1 = [1, 2, 3, 4];  
const reducer = (accumulator, currentValue) => accumulator - currentValue;  
  
console.log(array1.reduce(reducer));
```

accumulator 就是累计器，保存相减之后的累计值。currentValue 是当前数组。代码输出-8。

42.Array 的 filter() 方法用于过滤数组，使用符合条件的所有元素创建一个新的数组。

```
var ages = [32, 33, 16, 40];  
  
function checkAdult(age) {  
  return age >= 18;  
}  
  
console.log(ages.filter(checkAdult));
```

输出：

```
32,33,40
```

43.filter() 不会对空数组进行检测，也不会改变原始数组。

44.filter()的回调函数只有一个必选参数，那就是数组的当前元素：

```
var ages = [32, 33, 16, 40];  
  
function checkAdult(age) {  
  return age >= 18;  
}  
  
console.log(ages.filter(checkAdult));
```

45.sort() 方法用于对数组的元素进行排序。sort()方法默认把所有元素先转换为 String 再排序，所以使用默认的 sort()对数字进行排序时可能会有意想不到的结果。

```
console.log([10, 20, 1, 2].sort());
```

输出：

```
1,10,2,20
```

46.如果要按数值进行排序，可以使用自定义比较器：

```
'use strict';
var arr = [23, 9, 4, 78, 3];
var compare = function (x, y) {
  if (x < y) {
    return -1;
  } else if (x > y) {
    return 1;
  } else {
    return 0;
  }
}
console.log(arr.sort(compare));
```

47.闭包就是返回函数的函数。返回的函数不要引用任何循环变量，或者后续会发生变化的变量。

```
function count() {
  var arr = [];
  for (var i=1; i<=3; i++) {
    arr.push(function () {
      return i * i;
    });
  }
  return arr;
}
```

```
var results = count();
var f1 = results[0];
var f2 = results[1];
var f3 = results[2];
console.log(f1());
console.log(f2());
console.log(f3());
```

返回一个保存三个函数的数组，然后通过数组来访问三个函数，结果由于返回的函数中有可变的变量*i*，三个函数的输出都是16。

```
console.log(f1());
console.log(f2());
console.log(f3());
```

48. 如果要在闭包返回的函数引用循环变量，可以再创建一个函数，用该函数的参数绑定循环变量当前的值，无论该循环变量后续如何更改，已绑定到函数参数的值不变：

```
function count() {  
  var arr = [];  
  for (var i=1; i<=3; i++) {  
    arr.push((function (n) {  
      return function () {  
        return n * n;  
      }  
    })(i));  
  }  
  return arr;  
}
```

```
var results = count();  
var f1 = results[0];  
var f2 = results[1];  
var f3 = results[2];
```

```
console.log(f1());  
console.log(f2());  
console.log(f3());
```

输出：

```
1  
4  
9
```

49. 词法作用域：在函数定义的时候就决定了。

动态作用域：函数的作用域是在函数调用的时候才决定的。它并不关心函数和作用域是如何声明以及在何处声明的，只关心它们从何处调用。换句话说，作用域链是基于调用栈的，而不是代码中的作用域嵌套。

```
var a = 2;  
  
function foo() {  
  console.log(a); // 会输出 2 还是 3?  
}
```

```
function bar() {  
  var a = 3;  
  foo();  
}  
  
bar();
```

这段代码对于词法作用域，输出为 2，动态作用域，输出为 3。

50. ES6 标准新增了一种新的函数称为箭头函数，箭头函数相当于匿名函数，并且简化了函数定义。箭头函数有两种格式：

- 单行语句：可以省略{ ... }和 return。

```
x => x * x;
```

等号前面的 x 是参数。通过下面这种方式使用：

```
var fn = x => x * x;  
console.log(fn(4));
```

输出 16。

- 多行语句：不能省略{ ... }和 return。

```
x => {  
  if (x > 0) {  
    return x * x;  
  }  
  else {  
    return -x * x;  
  }  
}
```

如果有多个参数，需要用括号()括起来。

51. 箭头函数和匿名函数有个明显的区别：箭头函数内部的 this 是词法作用域，由上下文确定。

```
var obj = {  
  birth: 1990,  
  getAge: function () {  
    var b = this.birth; // 1990  
    var fn = () => new Date().getFullYear() - this.birth; // this 指向 obj 对象  
    return fn();  
  }  
};
```



```
obj.getAge();
```

内部的 this 始终指向外层的 obj。

52. 可以为引用类型添加属性和方法，也可以改变和删除其属性和方法：

```
var person = new Object();  
person.name = "Nicholas";  
alert(person.name); // "Nicholas"
```

创建了一个对象并将其保存到变量 person 中。然后，为该对象添加了一个名为 name 的属性。

53. ECMAScript 中所有函数的参数都是按值传递的。

54. instanceof 操作符用于检测某个值是什么类型的对象。

```
result = variable instanceof constructor
```

示例如下：

```
alert(person instanceof Object); // 变量 person 是 Object 吗?  
alert(colors instanceof Array); // 变量 colors 是 Array 吗?  
alert(pattern instanceof RegExp); // 变量 pattern 是 RegExp 吗?
```

所有引用类型的值都是 Object 的实例。

55. JavaScript 具有自动垃圾收集机制。

56. 而函数表达式可以定义在块级作用域中，而函数声明无法定义在块级作用域里：

```
// 不要这样做!  
if(condition){  
    function sayHi(){  
        alert("Hi!");  
    }  
} else {  
    function sayHi(){  
        alert("Yo!");  
    }  
}
```

这在 JavaScript 中是无效语法。而在函数表达式中却可以使用：

```
// 可以这样做  
var sayHi;  
  
if(condition){  
    sayHi = function(){  
        alert("Hi!");  
    };  
};
```

```
} else {  
    sayHi = function(){  
        alert("Yo!");  
    };  
}
```

57. 匿名函数可以括号来实现块级作用域：

```
(function(){  
    //这里是块级作用域  
})();
```

第3章 标准对象

1. 创建 Object 对象的方式有两种。

- 使用 new 操作符：

```
var person = new Object();  
person.name = "Nicholas";  
person.age = 29;
```

- 对象字面量表示法：使用冒号来定义。

```
var xiaoming = {  
    name: '小明',  
    birth: 1990,  
    age: function () {  
        var y = new Date().getFullYear();  
        return y - this.birth;  
    }  
};
```

2. typeof 操作符获取对象的类型，它返回一个字符串。

```
typeof 123;
```

返回 number。

3. JavaScript 为 number、boolean 和 string 提供了包装对象，类似于 Java 中的包装类。

```
var n = new Number(123);
var b = new Boolean(true);
var s = new String('str');
```

4. 如果我们在使用 Number、Boolean 和 String 时，Number()、Boolean 和 String() 被当做普通函数，结果就是把数据转换为 number、boolean 和 string 类型。

```
var n = Number('123');
```

结果将字符串转换为数字型 123。

5. 包装对象看上去和原来的值一模一样，显示出来也是一模一样，但他们的类型已经变为 object 了！所以，包装对象和原始值用 === 比较会返回 false：

```
typeof new Number(123);      // 'object'
new Number(123) === 123;      // false

typeof new Boolean(true);     // 'object'
new Boolean(true) === true;    // false

typeof new String('str');     // 'object'
new String('str') === 'str';   // false
```

所以不要使用包装对象。

6. 除了 null 和 undefined，任何对象都有 toString() 方法。
7. 使用 number 对象调用 toString() 时，需要使用两个点号或用括号括起来：

```
123..toString();
(123).toString();
```

8. 在 JavaScript 中，Date 对象用来表示日期和时间。
- 注意，当前时间是浏览器从本机操作系统获取的时间。
9. 如果要创建一个指定日期和时间的 Date 对象，可以用：

```
var d = new Date(2015, 5, 19, 20, 15, 30, 123);
```

分别为 2015 年 6 月 19 日 20 点 15 分 30 秒 123 毫秒。

- 特别注意 JavaScript 的月份是从 0 开始的，0 表示 1 月。
10. JavaScript 有两种方式创建一个正则表达式：
- 通过 /正则表达式/：

```
var re1 = /ABC\-001/;
var re2 = new RegExp('ABC\\-001');
```

- 通过 new RegExp('正则表达式') 创建一个 RegExp 对象：

```
var re2 = new RegExp('ABC\\-001');
```

11. 在 JSON 中，一共只有 6 种数据类型：

- boolean：就是 JavaScript 的 true 或 false。
- number：和 JavaScript 的 number 完全一致。
- string：就是 JavaScript 的 string。
- null：就是 JavaScript 的 null。
- array：就是 JavaScript 的 Array 表示方式——[]。
- object：就是 JavaScript 的 { ... } 表示方式。

12. JSON 定死了字符集必须是 UTF-8，字符串必须用双引号，Object 的键也必须用双引号。

```
{"name":"小明","age":14,"gender":true,"height":1.65,"grade":null,"middle-school":"W3C Middle School","skills":["JavaScript","Java","Python","Lisp"]}
```

13. 最简单的 JSON 数据形式就是简单值。

```
"Hello world!"
```

JavaScript 字符串与 JSON 字符串的最大区别在于，JSON 字符串必须使用双引号，单引号会导致语法错误。

14. JSON 中的对象与 JavaScript 字面量稍微有一些不同。下面是一个 JavaScript 中的对象字面量：

```
var person = {
  name: "Nicholas",
  age: 29
};
```

JSON 中的对象要求给属性加引号：

```
var object = {
  "name": "Nicholas",
  "age": 29
};
```

JSON 表示上述对象的方式如下：

```
{
```

```
"name": "Nicholas",  
"age": 29  
}
```

15. JSON 中的第二种复杂数据类型是数组。JSON 数组采用的就是 JavaScript 中的数组字面量形式。例如，下面是 JavaScript 中的数组字面量：

```
var values = [25, "hi", true];
```

在 JSON 中，可以采用同样的语法表示同一个数组：

```
[25, "hi", true]
```

16. 把数组和对象结合起来，可以构成更复杂的数据集合，例如：

```
[{  
  "title": "Professional JavaScript",  
  "authors": ["Nicholas C. Zakas"],  
  edition: 3,  
  year: 2011  
},  
{  
  "title": "Professional JavaScript",  
  "authors": ["Nicholas C. Zakas"],  
  edition: 2,  
  year: 2009  
}]
```

17. JSON 的 `stringify()` 函数用于将 JavaScript 数据序列化为 JSON 格式的字符串：

```
'use strict';  
  
var xiaoming = {  
  name: '小明',  
  age: 14,  
  gender: true,  
  height: 1.65,  
  grade: null,  
  'middle-school': "\"W3C\" Middle School",  
  skills: ['JavaScript', 'Java', 'Python', 'Lisp']  
};
```

```
var s = JSON.stringify(xiaoming);  
console.log(s);
```

18. JSON 的 `parse()` 函数用于将 JSON 格式的字符串反序列化为 JS 数据。

```
JSON.parse('[1,2,3,true]');
```

第 4 章面向对象编程

19. JavaScript 面向对象编程和 Java 等面向对象编程语言不一样，JS 没有类的概念，所以不区分类和对象，只有构造函数以及对象之分。对象都是使用构造函数创建出来的。

20. 构造函数跟普通函数没有区别，当使用 `new` 来创建对象时，它就是构造函数，否则就是普通函数。

21. 每个构造函数都有一个 `prototype` 属性，这个属性是一个指针，指向原型。

```
function SuperType(){  
    this.property = true;  
}  
SuperType.prototype.getSuperValue = function(){  
};  
  
function SubType(){  
    this.subproperty = false;  
}  
  
SubType.prototype = new SuperType();  
SubType.prototype.getSubValue = function (){  
    return this.subproperty;  
};  
  
var instance = new SubType();  
alert(instance.getSuperValue());  
    return this.property;
```

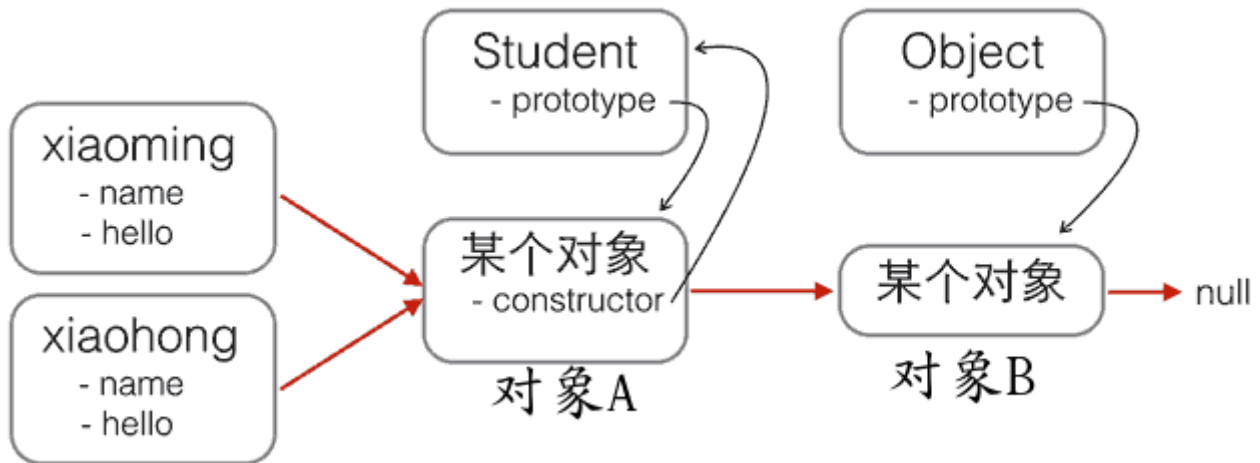
➤ 注意：对象是没有原型这个属性的。

22. 原型链上有 `prototype` 和 `constructor` 两个指针：

- `prototype`：构造函数的内部指针，指向原型。

- constructor: 原型对象的内部指针，指向构造函数本身。

举例来说，从 Student 构造函数 new 出 xiaoming 和 xiaohong 两个对象，两个对象的原型链如下：



红色线条代表原型链，xiaoming 和 xiaohong 的原型是对象 A，对象 A 的原型是对象 B，对象 B 的原型是 null。

xiaoming 和 xiaohong 的构造函数是 Student。

假设要访问 xiaoming 的某个方法，原型链的访问过程是先在 xiaoming 对象中查找是否有该方法，没有则向上查找它的原型，一直向上查找，一直找到最上面的对象，没找到则返回 undefined。

- 对象 A 中的 constructor 指向的并不是构造它自身的构造函数，而是指向把它当作原型、原型指针指向它的构造函数。

23. 为了区分普通函数和构造函数，按照约定，构造函数首字母应当大写，而普通函数首字母应当小写。

24. 通过两种方式来确定原型和实例之间的关系。

- instanceof:

```

alert(instance instanceof Object); //true
alert(instance instanceof SuperType); //true
alert(instance instanceof SubType); //true
  
```

- isPrototypeOf()方法:

```

alert(Object.prototype.isPrototypeOf(instance)); //true
alert(SuperType.prototype.isPrototypeOf(instance)); //true
alert(SubType.prototype.isPrototypeOf(instance)); //true
  
```

25. 给原型添加方法的代码一定要放在替换原型的语句之后。

```

function SuperType(){
  this.property = true;
}

SuperType.prototype.getSuperValue = function(){
  return this.property;
};

function SubType(){
  this.subproperty = false;
}

//继承了 SuperType
SubType.prototype = new SuperType();

//添加新方法
SubType.prototype.getSubValue = function (){
  return this.subproperty;
};

//重写超类型中的方法
SubType.prototype.getSuperValue = function (){
  return false;
};

var instance = new SubType();
alert(instance.getSuperValue()); //false

```

26. 通过原型链实现继承时，不能使用对象字面量创建原型方法。因为这样做就会重写原型链。

27. 原型链的问题是在通过原型来实现继承时，子类的原型实际上会变成父类原型的实例，所以当父类有属性时，该属性会变成所有子类共享。下列代码可以用来说明这个问题。

```

function SuperType(){
  this.colors = ["red", "blue", "green"];
}

function SubType(){
}

```



```
//继承了 SuperType
SubType.prototype = new SuperType();

var instance1 = new SubType();
instance1.colors.push("black");
alert(instance1.colors); //"red,blue,green,black"

var instance2 = new SubType();
alert(instance2.colors); //"red,blue,green,black"
```

当 SubType 通过原型链继承了 SuperType 之后，SubType.prototype 就变成了 SuperType 的一个实例，因此它也拥有了一个它自己的 colors 属性。结果是 SubType 的所有实例都会共享这一个 colors 属性。

28. 借用构造函数的基本思想相当简单，即在子类型构造函数的内部调用超类型构造函数。

通过使用 apply() 和 call() 方法也可以在新创建的对象上执行构造函数，如下所示：

```
function SuperType(){
    this.colors = ["red", "blue", "green"];
}

function SubType(){
    //继承了 SuperType
    SuperType.call(this);
}

var instance1 = new SubType();
instance1.colors.push("black");
alert(instance1.colors); //"red,blue,green,black"

var instance2 = new SubType();
alert(instance2.colors); //"red,blue,green"
```

这样，SubType 的每个实例就都会具有自己的 colors 属性的副本了。

- “SuperType.call(this)” 通过将 this 指针修改为 SubType，再调用父函数来实现每个实例都有一个 colors 属性的副本。

29. 相对于原型链而言，借用构造函数有一个很大的优势，即可以在子类型构造函数中向超类型构造函数传递参数。看下面这个例子。

```
function SuperType(name){
    this.name = name;
}
```

```
function SubType(){
  //继承了 SuperType, 同时还传递了参数
  SuperType.call(this, "Nicholas");

  //实例属性
  this.age = 29;
}

var instance = new SubType();
alert(instance.name); //"Nicholas";
alert(instance.age); //29
```

30. 组合继承, 使用原型链实现对原型属性和方法的继承, 而通过借用构造函数来实现对实例属性的继承。这样, 既通过在原型上定义方法实现了函数复用, 又保证每个实例都有它自己的属性。

```
<script>
function Parent(age){
  this.name = ['mike','jack','smith'];
  this.age = age;
}
Parent.prototype.run = function () {
  return this.name + ' are both' + this.age;
};
function Child(age){
  Parent.call(this,age);
}
Child.prototype = new Parent();
var test = new Child(21);          //写 new Parent(21)也行
alert(test.run());
</script>
```

31. Object 本身是一个函数, 可以当作工具方法使用, 将任意值转为对象。这个方法常用于保证某个值一定是对象。

```
var obj = Object();
// 等同于
var obj = Object(undefined);
var obj = Object(null);
```

```
obj instanceof Object // true
```

32. 寄生式继承：也就是一个对象的定义和赋值都在一个函数里实现。以下代码示范了寄生式继承模式。

```
function createAnother(original){
    var clone = object(original); //通过调用函数创建一个新对象
    clone.sayHi = function(){ //以某种方式来增强这个对象
        alert("hi");
    };
    return clone; //返回这个对象
}
```

createAnother()函数接收了一个参数，也就是将要作为新对象基础的对象。然后，把这个对象（original）传递给 object() 函数，将返回的结果赋值给 clone。再为 clone 对象添加一个新方法 sayHi()，最后返回 clone 对象。可以像下面这样来使用 createAnother() 函数：

```
var person = {
    name: "Nicholas",
    friends: ["Shelby", "Court", "Van"]
};

var anotherPerson = createAnother(person);
anotherPerson.sayHi(); // "hi"
```

33. 组合继承是 JavaScript 最常用的继承模式；不过，它也有自己的不足。组合继承最大的问题就是无论什么情况下，都会调用两次超类型构造函数：一次是在创建子类型原型的时候，另一次是在子类型构造函数内部。

34. 所谓寄生组合式继承，即通过借用构造函数来继承属性，通过原型链的混成形式来继承方法。寄生组合式继承的基本模式如下所示。

```
function inheritPrototype(subType, superType){
    var prototype = object(superType.prototype); //创建超类型原型的一个副本
    prototype.constructor = subType; //为创建的副本添加 constructor 属性
    subType.prototype = prototype; //将新创建的对象赋值给子类型的原型
}
```

这个函数接收两个参数：子类型构造函数和超类型构造函数。

这样，我们就可以用调用 inheritPrototype() 函数的语句，去替换前面例子中为子类型原型赋值的语句了，例如：

```
function SuperType(name){
```

```

    this.name = name;
    this.colors = ["red", "blue", "green"];
}

SuperType.prototype.sayName = function(){
    alert(this.name);
};

function SubType(name, age){
    SuperType.call(this, name);
    this.age = age;
}

inheritPrototype(SubType, SuperType);

SubType.prototype.sayAge = function(){
    alert(this.age);
};

```

这里要注意，调用 `inheritPrototype` 前，要先定义子类。

第 5 章 浏览器

1. `window` 对象既充当全局作用域，也表示浏览器窗口。
2. `window` 对象的 `innerWidth` 和 `innerHeight` 属性可以获取浏览器窗口的内部宽度和高度。内部宽高是指除去菜单栏、工具栏、边框等占位元素后，用于显示网页的净宽高。

```

console.log('window inner size: ' + window.innerWidth + ' x ' +
window.innerHeight);

```

3. `window` 对象的 `outerWidth` 和 `outerHeight` 属性可以获取浏览器窗口的整个宽高。
4. `navigator` 对象表示浏览器的信息，最常用的属性包括：
 - `navigator.appName`：浏览器名称；
 - `navigator.appVersion`：浏览器版本；
 - `navigator.language`：浏览器设置的语言；
 - `navigator.platform`：操作系统类型；

- navigator.userAgent: 浏览器设定的 User-Agent 字符串。

```
console.log('appName = ' + navigator.appName);
console.log('appVersion = ' + navigator.appVersion);
console.log('language = ' + navigator.language);
console.log('platform = ' + navigator.platform);
console.log('userAgent = ' + navigator.userAgent);
```

- navigator 的信息可以很容易地被用户修改, 所以 JavaScript 读取的值不一定是正确的。很多人为了针对不同浏览器编写不同的代码, 喜欢用 if 判断浏览器版本:

```
var width;
if (getIEVersion(navigator.userAgent) < 9) {
    width = document.body.clientWidth;
} else {
    width = window.innerWidth;
}
```

但这样既可能判断不准确, 也很难维护代码。正确的方法是充分利用 JavaScript 对不存在属性返回 undefined 的特性, 直接用短路运算符 || 计算:

```
var width = window.innerWidth || document.body.clientWidth;
```

5. screen 对象表示屏幕的信息, 常用的属性有:

- screen.width: 屏幕宽度, 以像素为单位;
- screen.height: 屏幕高度, 以像素为单位;
- screen.colorDepth: 返回颜色位数, 如 8、16、24。

```
console.log('Screen size = ' + screen.width + ' x ' + screen.height);
```

6. location 对象表示当前页面的 URL 信息。

7. document 对象表示当前页面。由于 HTML 页面在浏览器中以 DOM 形式表示为树形结构, document 对象就是整个 DOM 树的根节点。

```
document.title = '努力学习JavaScript!';
```

- 注意, 这个 DOM 树是指单个页面的 DOM 树。

8. 用 document 对象提供的 getElementById() 和 getElementsByTagName() 可以按 ID 获得一个 DOM 节点和按 Tag 名称获得一组 DOM 节点:

```
var menu = document.getElementById('drink-menu');
var drinks = document.getElementsByTagName('dt');
var i, s, menu, drinks;

menu = document.getElementById('drink-menu');
```

```

menu.tagName; // 'DL'

drinks = document.getElementsByTagName('dt');
s = '提供的饮料有: ';
for (i=0; i<drinks.length; i++) {
    s = s + drinks[i].innerHTML + ' ';
}
console.log(s);

```

9. document 对象还有一个 cookie 属性，可以获取当前页面的 Cookie。
10. history 对象保存了浏览器的历史记录，JavaScript 可以调用 history 对象的 back()或 forward ()，相当于用户点击了浏览器的“后退”或“前进”按钮。
➤ 这个对象造成的用户体验不好，任何情况，都不应该使用 history 这个对象了。
11. document 对象的 getElementById() 方法可返回对拥有指定 ID 的第一个对象的引用。
12. document 对象的 getElementsByTagName() 方法可返回带有指定标签名的对象的集合。
13. getElementsByClassName() 方法返回文档中所有指定类名的元素集合。
14. 要删除一个节点，首先要获得该节点本身以及它的父节点，然后，调用父节点的 removeChild 把自己删掉：

```

var self = document.getElementById('to-be-removed');
var parent = self.parentElement;
var removed = parent.removeChild(self);
removed === self; // true

```

15. 用 JavaScript 操作表单和操作 DOM 是类似的，因为表单本身也是 DOM 树。
16. 如果我们获得了一个<input>节点的引用，就可以直接调用 value 获得对应的用户输入值：

```

// <input type="text" id="email">
var input = document.getElementById('email');
input.value;

```

17. 对于单选框和复选框，value 属性返回的永远是 HTML 预设的值，要获取用户是否勾上的选项，应该用 checked 判断：

```

// <label><input type="radio" name="weekday" id="monday" value="1">
Monday</label>
// <label><input type="radio" name="weekday" id="tuesday" value="2">

```

```
Tuesday</label>
var mon = document.getElementById('monday');
var tue = document.getElementById('tuesday');
mon.value; // '1'
tue.value; // '2'
mon.checked; // true 或者 false
tue.checked; // true 或者 false
```

18. 设置值和获取值类似，对于 text、password、hidden 以及 select，直接设置 value 就可以：

```
// <input type="text" id="email">
var input = document.getElementById('email');
input.value = 'test@example.com';
```

19. JavaScript 可以以两种方式来处理表单的提交。

- 通过<form>元素的 submit()方法提交一个表单：

```
<!-- HTML -->
<form id="test-form">
  <input type="text" name="test">
  <button type="button" onclick="doSubmitForm()">Submit</button>
</form>

<script>
function doSubmitForm() {
  var form = document.getElementById('test-form');
  form.submit();
}
</script>
```

响应一个<button>的 click 事件，在 JavaScript 代码中提交表单。这种方式的缺点是扰乱了浏览器对 form 的正常提交。浏览器默认点击<button type="submit">时提交表单，或者用户在最后一个输入框按回车键。

- 响应<form>本身的 onsubmit 事件，在提交 form 时作修改：

```
<form id="test-form" onsubmit="return checkForm()">
  <input type="text" name="test">
  <button type="submit">Submit</button>
</form>

<script>
function checkForm() {
  var form = document.getElementById('test-form');
  // 可以在此修改 form 的 input...
```

```
// 继续下一步:
return true;
}
</script>
```

注意要 return true 来告诉浏览器继续提交, 如果 return false, 浏览器将不会继续提交 form, 这种情况通常对应用户输入有误, 提示用户错误信息后终止提交 form。

20. 通常, 上传的文件都由后台服务器处理, JavaScript 可以在提交表单时对文件扩展名做检查, 以便防止用户上传无效格式的文件:

```
var f = document.getElementById('test-file-upload');
var filename = f.value;
if (!filename || !(filename.endsWith('.jpg') || filename.endsWith('.png') ||
filename.endsWith('.gif'))) {
    alert('Can only upload image file. ');
    return false;
}
```

21. 在 JavaScript 中, 浏览器的 JavaScript 执行引擎在执行 JavaScript 代码时, 总是以单线程模式执行, 也就是说, 任何时候, JavaScript 代码都不可能同时有多于 1 个线程在执行。
22. 在 JavaScript 中, 执行多任务实际上都是异步调用。
23. 在现代浏览器上写 AJAX 主要依靠 XMLHttpRequest 对象。
24. 如果页面中包含框架, 则每个框架都拥有自己的 window 对象, 并且保存在 frames 集合中。
25. 在 frames 集合中, 可以通过数值索引来访问 window 对象, 索引值从 0 开始, 从左至右, 从上到下; 也可以通过框架名称来访问相应的 window 对象。每个 window 对象都有一个 name 属性, 其中包含框架的名称。下面是一个包含框架的页面:

```
<html>
<head>
<title>Frameset Example</title>
</head>
<frameset rows="160,*">
<frame src="frame.htm" name="topFrame">
<frameset cols="50%,50%">
<frame src="anotherframe.htm" name="leftFrame">
<frame src="yetanotherframe.htm" name="rightFrame">
</frameset>
</frameset>
</html>
```


26.top 对象始终指向最高（最外）层的框架，也就是浏览器窗口。下图是访问框架的方式：

<pre>window.frames[0] window.frames["topFrame"] top.frames[0] top.frames["topFrame"] frames[0] frames["topFrame"]</pre>	
<pre>window.frames[1] window.frames["leftFrame"] top.frames[1] top.frames["leftFrame"] frames[1] frames["leftFrame"]</pre>	<pre>window.frames[2] window.frames["rightFrame"] top.frames[2] top.frames["rightFrame"] frames[2] frames["rightFrame"]</pre>

27.parent 对象始终指向当前框架的直接上层框架。某些情况下，parent 有可能等于 top；如果没有框架，parent 一定等于 top，此时它们都等于 window。

28.除非最高层窗口是通过 window.open()打开的，否则其 window 对象的 name 属性不会包含任何值。

29.事件处理程序的名字以"on"开头，因此 click 事件的事件处理程序就是 onclick，load 事件的事件处理程序就是 onload。

```
<input type="button" value="Click Me" onclick="alert('Clicked')" />
```

➤ 当一个资源及其依赖资源已完成加载时，将触发 load 事件。

30.事件处理程序中能使用 event 对象：

```
<!-- 输出 "click" -->
<input type="button" value="Click Me" onclick="alert(event.type)">
```

31. 事件处理程序中的 this 值等于事件的目标元素：

```
<!-- 输出 "Click Me" -->
<input type="button" value="Click Me" onclick="alert(this.value)">
```

这个 this 值也就是 input 元素。

32. 每个元素都有自己的事件处理程序属性，这些属性通常全部小写，例如 onclick。将这种属性的值设置为一个函数，就可以指定事件处理程序，如下所示：

```
var btn = document.getElementById("myBtn");
btn.onclick = function(){
    alert("Clicked");
};
```

33. JavaScript 是单线程的，所以浏览器事件等需要异步执行，异步执行通过回调函数来实现。但回调函数不利于代码复用，使用 Promise 对象可以解决这个问题。

34. Promise 有各种开源实现，在 ES6 中被统一规范，由浏览器直接支持。

35. Promise 对象：先统一执行 AJAX 逻辑，不关心如何处理结果，然后，根据结果是成功还是失败，在将来的某个时候调用 success 函数或 fail 函数。这种对象称为 Promise 对象。

```
var p1 = new Promise(test);
var p2 = p1.then(function (result) {
    console.log('成功: ' + result);
});
var p3 = p2.catch(function (reason) {
    console.log('失败: ' + reason);
});
```

变量 p1 是一个 Promise 对象，它负责执行 test 函数。由于 test 函数在内部是异步执行的，当 test 函数执行成功时，执行 p2；当 test 函数执行失败时，执行 p3。Promise 对象可以串联起来，所以上述代码可以简化为：

```
new Promise(test).then(function (result) {
    console.log('成功: ' + result);
}).catch(function (reason) {
    console.log('失败: ' + reason);
});
```

36. Promise 还可以做更多的事情，比如，有若干个异步任务，需要先做任务 1，如果成功后再做任务 2，任何任务失败则不再继续并执行错误处理函数。要串行执行这样的异步任务，不用 Promise 需要写一层一层的嵌套代码。有了 Promise，我们只需要

简单地写：

```
job1.then(job2).then(job3).catch(handleError);
```

其中，job1、job2 和 job3 都是 Promise 对象。

37.

第 6 章 错误调试

1. JS 引入了 try-catch 语句，基本的语法如下所示：

```
try {  
    // 可能会导致错误的代码  
} catch(error) {  
    // 在错误发生时怎么处理  
}
```

2. finally 子句：

```
function testFinally() {  
    try {  
        return 2;  
    } catch(error) {  
        return 1;  
    } finally {  
        return 0;  
    }  
}
```

3. throw 操作符用于随时抛出自定义错误。抛出错误时，必须要给 throw 操作符指定一个值，这个值是什么类型，没有要求。

```
throw 12345;  
throw "Hello world!";  
throw true;  
throw { name: "JavaScript"};  
throw new Error("Something bad happened.");
```

上面代码都是有效的。

4. JS 定义了下列 7 种错误类型：

- Error：基类型，其他错误类型都继承自该类型。
- EvalError：EvalError 类型的错误会在使用 eval() 函数而发生异常时被抛出。

- **RangeError**: **RangeError** 类型的错误会在数值超出相应范围时触发。
- **ReferenceError**: 在找不到对象的情况下，会触发 **ReferenceError**。
- **SyntaxError**: 把语法错误的 JavaScript 字符串传入 **eval()** 函数时，就会导致此类错误。
- **TypeError**: 在变量中保存着意外的类型时，或者在访问不存在的方法时，都会导致这种错误。
- **URIError**: **URI** 格式不正确时，就会导致 **URIError** 错误。

5.

第 7 章 集合

1. JavaScript 的集合有 **Array**、**set** 和 **map** 三个类。
2. JavaScript 中 **Array**、**set**、**map** 和字符串都是可迭代对象。
3. **Map** 是一组键值对的结构，具有极快的查找速度。

```
var m = new Map([['Michael', 95], ['Bob', 75], ['Tracy', 85]]);  
m.get('Michael');
```

4. **map** 函数:

- **has()**: 是否包含键。
- **set()**: 设置指定 **key** 对应的值。
- **get()**: 获取指定 **key** 对应的值。
- **keys()**: 返回一个新的 **Iterator** 对象，它按插入顺序包含了 **Map** 对象中每个元素的键。
- **values()**: 返回一个新的 **Iterator** 对象，它按插入顺序包含了 **Map** 对象中每个元素的值。
- **delete()**: 如果 **Map** 对象中存在该元素，则移除它并返回 **true**；否则如果该元素不存在则返回 **false**。
- **entries()**: 返回一个新的 **Iterator** 对象，按插入顺序包含了 **Map** 对象中每个元素的 **[key, value]** 数组。

```
var m = new Map();  
m.set('Adam', 67);  
m.set('Bob', 59);  
m.has('Adam');
```

```
m.get('Adam');  
m.delete('Adam');  
m.get('Adam');
```

5. map 的一个 key 只能对应一个 value，所以，如果多次对一个 key 赋值，后面的值会把前面的值覆盖：

```
var m = new Map();  
m.set('Adam', 67);  
m.set('Adam', 88);  
m.get('Adam');
```

6. Set 是一组值的集合，Set 中的值不能重复。

```
var s = new Set([1, 2, 3, 3, '3']);
```

- has(value)：返回一个布尔值，表示集合中是否存在该值。
- keys()：返回一个新的迭代器对象，该对象包含 Set 对象中的按插入顺序排列的所有元素的值。
- values()：返回一个新的迭代器对象，该对象包含 Set 对象中的按插入顺序排列的所有元素的值。该方法是 keys()方法相同。
- add()：在 Set 对象尾部添加一个元素。返回该 Set 对象。。
- delete()：删除 Set 中与指定值相等的元素。
- entries()：返回一个新的迭代器对象，该对象包含 Set 对象中的按插入顺序排列的所有元素的值的[value, value]数组。

```
let mySet = new Set();  
  
mySet.add(1);  
  
mySet.add("some text");  
var o = {a: 1, b: 2};  
mySet.add(o);  
  
mySet.add({a: 1, b: 2});  
  
mySet.has(1);
```

7. 遍历 Array 可以采用下标循环，遍历 Map 和 Set 就无法使用下标。为了统一集合类

型，ES6 标准引入了新的 iterable 类型，Array、Map 和 Set 都属于 iterable 类型。

➤ iterable 其实就是可迭代对象。

8. 具有 iterable 类型的集合可以通过新的 for ...of 循环来遍历。

```
'use strict';
var a = [1, 2, 3];
for (var x of a) {
}
console.log(x);
```

9. for ... in 和 for ... of 的区别：

(1) for ... in 循环遍历的实际上是 key，而不是 value。一个 Array 数组实际上也是一个对象，它的每个元素的索引被视为 key。

```
var a = ['A', 'B', 'C'];
a.name = 'Hello';
for (var x in a) {
  console.log(x);
}
```

输出：

```
0
1
2
name
```

(2) for ... of 修复了这个问题：

```
var a = ['A', 'B', 'C'];
a.name = 'Hello';
for (var x of a) {
  console.log(x);
}
```

输出：

```
A
B
C
```

10. iterable 内置的 forEach 方法，它接收一个函数，每次迭代就自动回调该函数。

```
a.forEach(function (element, index, array) {  
  console.log(element + ', index = ' + index);  
});
```

输出:

```
A, index = 0  
B, index = 1  
C, index = 2
```

- element: 指向当前元素的值。
- index: 指向当前索引。
- array: 指向 Array 对象本身。

11.Set 与 Array 类似, 但 Set 没有索引, 因此回调函数的前两个参数都是元素本身:

```
var s = new Set(['A', 'B', 'C']);  
s.forEach(function (element, sameElement, set) {  
  console.log(element);  
});
```

12.Map 的回调函数参数依次为 value、key 和 map 本身:

```
var m = new Map([[1, 'x'], [2, 'y'], [3, 'z']]);  
m.forEach(function (value, key, map) {  
  console.log(value);  
});
```

13.如果对某些参数不感兴趣, 由于 JavaScript 的函数调用不要求参数必须一致, 因此可以忽略它们。例如, 只需要获得 Array 的 element:

```
var a = ['A', 'B', 'C'];  
a.forEach(function (element) {  
  console.log(element);  
});
```

14.

Javascript 总结

1. 使用一个括号来定义并立即执行函数：

```
var uniqueInteger=(function(){ //定义函数并立即执行
    var counter=0; //函数的私有状态
    return function(){return counter++;};
})();
```

2. 如果一个函数使用 new 来调用，则该函数是一个构造函数，返回一个对象，否则，该函数为普通函数。
3. 普通函数内不建议使用 this 关键字，此时 this 指向的对象为 window。
4. Javascript 和 python 一样，支持为类动态地添加类属性：

```
function Complex(real, imaginary) {
    if (isNaN(real) || isNaN(imaginary)) // 确保两个实参都是数字
        throw new TypeError(); // 如果不都是数字则抛出错误
    this.r = real; // 复数的实部
    this.i = imaginary; // 虚部
}

// 为 Complex 类动态添加类属性
Complex.ZERO = new Complex(0,0);
Complex.ONE = new Complex(1,0);
Complex.I = new Complex(0,1);
```

5. 使用 this 可以遍历一个类的属性：

```
var generic = {
    // 该函数返回一个字符串，这个字符串包含构造函数的名字（如果构造函数包含名字）
    // 以及所有非继承来的、非函数属性的名字和值
    toString: function() {
        var s = '[';
        // 如果这个对象包含构造函数，且构造函数包含名字
        if (this.constructor && this.constructor.name)
            s += this.constructor.name + ": ";

        // 枚举所有的非继承的、非函数属性
        var n = 0;
        for(var name in this) {
```



```

    if (!this.hasOwnProperty(name)) continue; // 跳过继承来的属性
    var value = this[name];
    if (typeof value === "function") continue; // 跳过方法
    if (n++) s += ", ";
    s += name + '=' + value;
  }
  return s + '];
},

```

6. 使用对象字面量重写原型时，使用类似 `constructor: xxxx` 的方式来重新指定 `constructor`：

```

function Range(from, to) {
  this.from = function() { return from; };
  this.to = function() { return to; };
}

Range.prototype = {
  constructor: Range,           // 重新设置构造函数
  includes: function(x) { return this.from() <= x && x <= this.to(); },
  foreach: function(f) {
    for(var x=Math.ceil(this.from()), max=this.to(); x <= max; x++)
      f(x);
  },
  toString: function() { return "(" + this.from() + "..." + this.to() + ")"; }
};

```

7. javascript 使用 `arguments` 来接收参数。`arguments` 并非是数组对象，但可以像数组一样使用：

<code>arguments[0]</code>	表示接收的第一个参数
<code>arguments[1]</code>	表示接收的第二个参数

8. 虽然 `arguments` 对象并不是真正意义上的 Javascript 数组，但是我们可以使用数组的 `slice` 方法将其转换成数组，类似下面的代码：

```

var args = Array.prototype.slice.call(arguments);

```

9. javascript 函数不介意传递进来多少个参数，也不在乎传进来的参数是什么数据类型，甚至可以不传参数。当实参比形参少时，多出来的形参会被设置为 undefined。

10. 在实际生产中，创建数组的方式基本有两种：

- 使用 Array 构造函数：

```
var arr = new Array();
```

- 使用 [] 创建数组：

```
var arr1 = ['a','b'];
```

这种定义允许有可选的结尾的逗号，故[,]只有两个元素而非三个。

11. 数组元素的添加和删除。

- 添加数组元素最简单的方法是直接使用 []：

```
A=[]           //开始是一个空数组
a[0]="zero";    //然后向其中添加元素
a[1]="one";
```

- 也可以使用 push() 方法在数组末尾增加一个或多个元素：

```
A=[];           //开始是一个空数组
a.push("zero")   //在末尾添加一个元素。a=["zero"]
a.push("one","two") //再添加两个元素。a=["zero","one","two"]
```

- 使用 delete 运算符来删除数组元素：

```
a=[1,2,3];
delete a[1];
```

12. 创建对象的方式有几种：

- new 操作符：

```
var Base = function () {}
var o1 = new Base();
```

使用 new 创建的对象拥有继承链。

- Object.create()：

```
var Base = function () {}
```

```
var o2 = Object.create(Base);
```

使用 `Object.create` 创建的对象没有继承链，所以这个对象会丢失父类的所有方法和属性。

- 对象字面量：

```
var person = {  
  name: "Nicholas",  
  age: 29,  
  job: "Software Engineer",  
  
  sayName: function(){  
    alert(this.name);  
  }  
};
```

13. Array 类的方法：

(1) `from()`：从一个类似数组或可迭代对象中创建一个新的数组实例。

```
console.log(Array.from('foo'));
```

输出：

```
f,o,o
```

(2) `isArray()`：判断参数是否是一个数组。

```
Array.isArray([1, 2, 3]);
```

(3) `of()`：创建一个具有可变数量参数的新数组实例，而不考虑参数的数量或类型。

- `Array.of()` 和 `Array` 构造函数之间的区别在于处理整数参数：`Array.of(7)` 创建一个具有单个元素 7 的数组，而 `Array(7)` 创建一个长度为 7 的空数组（注意：这是指一个有 7 个空位的数组，而不是由 7 个 `undefined` 组成的数组）。

(4) `concat()`：合并两个或多个数组。此方法不会更改现有数组，而是返回一个新数组。

```
var array1 = ['a', 'b', 'c'];  
var array2 = ['d', 'e', 'f'];  
  
console.log(array1.concat(array2));
```

输出：

```
a,b,c,d,e,f
```

(5) `copyWithin()`: 浅复制数组的一部分到同一数组中的另一个位置，并返回它，而不修改其大小。

```
var array1 = ['a', 'b', 'c', 'd', 'e'];  
  
console.log(array1.copyWithin(0, 3, 4));
```

输出:

```
d,b,c,d,e
```

(6) `entries()`: 返回一个新的 Array Iterator 对象，该对象包含数组中每个索引的键/值对。

```
var array1 = ['a', 'b', 'c'];  
var iterator1 = array1.entries();  
console.log(iterator1.next().value);
```

输出:

```
0,a
```

(7) `every()`: 测试数组的所有元素是否都通过了指定函数的测试。

```
function isBelowThreshold(currentValue) {  
  return currentValue < 40;  
}  
  
var array1 = [1, 30, 39, 29, 10, 13];  
  
console.log(array1.every(isBelowThreshold));
```

输出 true。

(8) `fill()`: 用一个固定值填充一个数组中从起始索引到终止索引内的全部元素。

```
var array1 = [1, 2, 3, 4];  
console.log(array1.fill(0, 2, 4));
```

将第 2 个到第 4 个值用 0 填充，第二个和第三个参数是可选值。输出:

```
1,2,0,0
```

(9) `filter()`: 过滤元素，用过滤后的值创建一个新数组并返回。

```
var words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
const result = words.filter(word => word.length > 6);
console.log(result);
```

输出：

```
exuberant,destruction,present
```

(10) find(): 查找满足测试条件的第一个值，参数是一个回调函数。

```
var array1 = [5, 12, 8, 130, 44];

var found = array1.find(function(element) {
  return element > 10;
});

console.log(found);
```

输出 12。

(11) findIndex(): 查找满足测试条件的第一个值的索引，参数是一个回调函数。

(12) flat(): 递归到指定深度将所有子数组连接，并返回一个新数组。

```
var arr1 = [1, 2, [3, 4]];
console.log(arr1.flat());
```

输出：

```
1,2,3,4
```

(13) forEach(): 对数组的每个元素执行一次提供的函数。

```
var array1 = ['a', 'b', 'c'];

array1.forEach(function(element) {
  console.log(element);
});
```

输出：

```
a
b
c
```

(14) includes(): 判断一个数组是否包含一个指定的值，根据情况，如果包含则返

回 true, 否则返回 false。

```
var array1 = [1, 2, 3];  
console.log(array1.includes(2));
```

输出 true。

- (15) indexOf(): 返回在数组中可以找到一个给定元素的第一个索引, 如果不存在, 则返回-1。

```
var beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];  
console.log(beasts.indexOf('bison'));
```

输出 1。

- (16) join(): 将一个数组的所有元素连接成一个字符串并返回这个字符串, 这里要注意分隔符也返回。如果数组只有一个值, 那么将返回该值, 但不返回分隔符。

```
var elements = ['Fire', 'Wind', 'Rain'];  
console.log(elements.join());
```

输出:

Fire,Wind,Rain

- (17) keys(): 获取由所有索引组成的一个数组。

```
var array1 = ['a', 'b', 'c'];  
var iterator = array1.keys();  
  
for (let key of iterator) {  
  console.log(key); // expected output: 0 1 2  
}
```

输出:

0
1
2

- (18) lastIndexOf(): 获取指定元素在数组中的最后一个的索引, 如果不存在则返回 -1。从数组的后面向前查找, 从 fromIndex 处开始。

```
var animals = ['Dodo', 'Tiger', 'Penguin', 'Dodo'];  
console.log(animals.lastIndexOf('Dodo'));
```

输出 3。

- (19) map(): 使用数组中的每个元素作为参数传入回调函数, 并将结果组合成一个

数组返回。

```
var array1 = [1, 4, 9, 16];  
const map1 = array1.map(x => x * 2);  
console.log(map1);
```

输出：

```
2,8,18,32
```

- (20) `pop()`: 从数组中删除最后一个元素，并返回该元素的值。此方法更改数组的 `length` 属性。
- (21) `push()`: 将一个或多个元素添加到数组的末尾，并返回该数组的新长度。
- (22) `reduce()`: 接收一个回调函数，使用 `Array` 中的每个值都作为参数传入该回调函数进行计算，最后的返回值为单个值。
- (23) `reduceRight()`: 和 `reduce()` 作用差不多，只是该函数按从右到左的顺序将数组值传入回调函数。
- (24) `reverse()`: 返回数组元素的顺序。
- (25) `shift()`: 从数组中删除第一个元素，并返回该元素的值。
- (26) `slice()`: 获取指定的索引区间的值，将这些值组合成一个新数组并返回。

```
var animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];  
console.log(animals.slice(2));  
console.log(animals.slice(2, 4));
```

开始区间默认是 0。输出：

```
camel,duck,elephant  
camel,duck
```

- (27) `some()`: 测试是否至少有一个元素通过由提供的函数实现的测试。

```
var array = [1, 2, 3, 4, 5];  
  
var even = function(element) {  
  return element % 2 === 0;  
};  
  
console.log(array.some(even));
```

输出 `true`。

- (28) `sort()`: 对数组的元素进行排序，并返回数组。

(29) splice():

(30) toString(): 将数组转化为字符串, 并返回。

```
var array1 = [1, 2, 'a', '1a'];  
console.log(array1.toString());
```

输出:

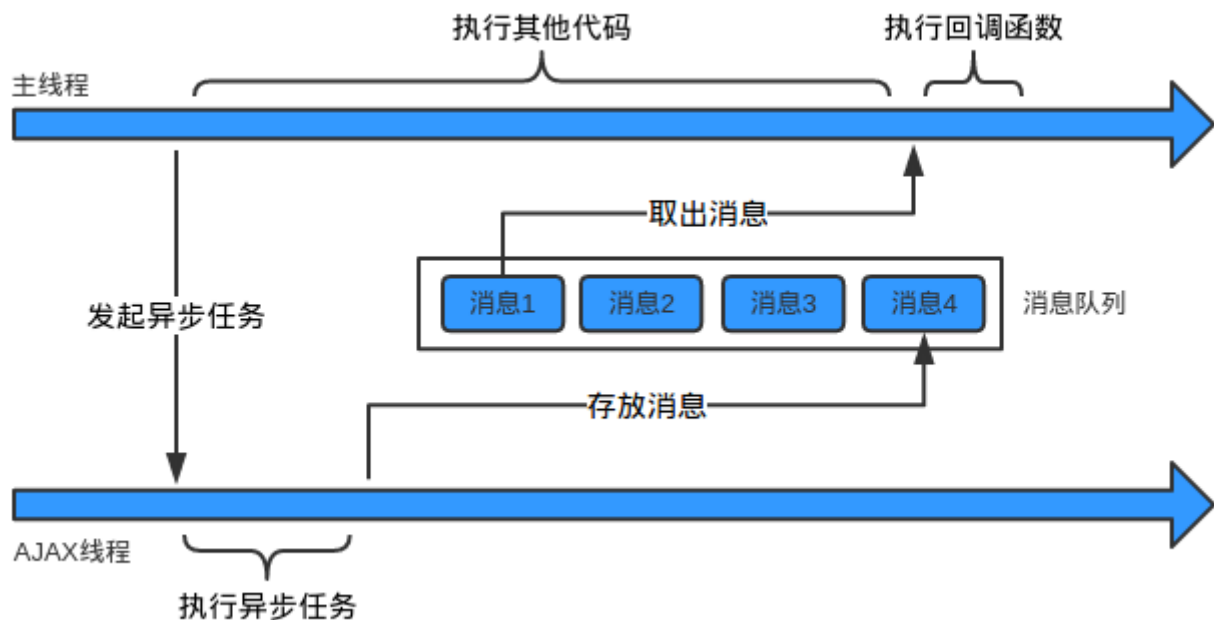
```
1,2,a,1a
```

(31) unshift(): 添加一个或多个元素到数组的开头, 并返回该数组的新长度。

```
var array1 = [1, 2, 3];  
console.log(array1.unshift(4, 5));  
console.log(array1);
```

14. JavaScript 是单线程的, 同一个时间只能做一件事。Javascript 任务分为同步任务和异步任务:

- (1) 同步任务是主线程上排队执行的任务, 只有前一个任务执行完毕, 才能执行后一个任务;
 - (2) 异步任务指的是, 不进入主线程、而进入"消息队列"的任务, 只有"消息队列"通知主线程, 某个异步任务可以执行了, 该任务才会进入主线程执行。消息队列中的每条消息实际上都对应着一个 DOM 事件。
- Javascript 等待所有主线程任务都执行完毕之后, 再执行异步任务, 以 Ajax 为例, 执行过程如图:



15.除了放置异步任务的事件, "消息队列"还可以放置定时事件。定时器功能主要由 `setTimeout()`和 `setInterval()`这两个函数来完成, 它们的内部运行机制完全一样, 区别在于前者指定的代码是一次性执行, 后者则为反复执行。

16.在 Javascript 中, 能够转换为 `false` 的表达式有:

- `null`;
- `NaN`;
- `0`;
- 空字符串 (`""`) ;
- `undefined`。

这个应用于 `if` 语句为:

```
if(!var)
{
    .....
}
```

如果 `var` 变量为空或者未定义, 则执行语句块里面的语句。

17.`window.location.search()`用于获取 URL 中`?`及`?`之后的内容, 而 `window.location.search.substring(1)`则是去除了`?`, 获取`?`之后的内容。

18.通过 `indexOf()`和 `substring()`可以方便地提取 “`a=xxx`” 这种形式的值:

```
var pos = pairs[i].indexOf('=');    // indexOf()方法返回指定的字符串值首次出现的位
```

置。

```
if (pos == -1) continue;           // 如果没有找到的话，就跳过
var name = pairs[i].substring(0,pos);    // 提取 name
var value = pairs[i].substring(pos+1);    // 提取 value
```

19.html 的 meta 标签共有两个属性，分别是 name 属性和 http-equiv 属性：

- name 属性主要用于描述网页，比如网页的关键词，叙述等。与之对应的属性值为 content，content 中的内容是对 name 填入类型的具体描述，便于搜索引擎抓取。格式：

```
<meta name="参数" content="具体的描述">。
```

例如：

```
<meta name="keywords" content="Lxxyx,博客，文科生，前端">
```

- http-equiv 顾名思义，相当于 http 协议中文件头的作用，它可以向浏览器传回一些有用的信息，以帮助正确和精确地显示网页内容，与之对应的属性值为 content，content 中的内容其实就是各个参数的变量值。格式：

```
<meta http-equiv="参数" content="具体的描述">
```

例如：

```
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>
```

34.当函数参数是一个函数时，传参时直接使用函数名：

```
function addPrivateProperty(o, name, predicate) {
  var value;
  o["get" + name] = function() { return value; };
  o["set" + name] = function(v) {
    if (predicate && !predicate(v))
      throw Error("set" + name + ": invalid value " + v);
    else
      value = v;
  };
}

var o = {}; // 空对象

addPrivateProperty(o, "Name", function(x) { return typeof x == "string"; });
```

35.立即执行函数有两种形式:

- 第一种形式:

```
var=(myfun()  
{})();
```

- 第二种形式:

```
var=(myfun()  
{})();
```

两种形式的区别在于括号的位置不一样。

36.window 对象指窗口对象, 而 document 对象是网页对象, 在没有 iframe 的情况下, 大致可以认为是等同的, 有 iframe 就不一样。

37.\$() 方法是在 DOM 中使用过于频繁的 document.getElementById() 方法的一个便利的简写, 这个方法返回参数传入的 id 的那个元素。例如, \$("id")即获取页面 id 为" id "的元素。\$()可以传入多个 id 作为参数:

```
function test2()  
{  
    var divs = $(' myDiv' , ' myOtherDiv' );  
    for(i=0; i<divs.length; i++)  
    {  
        alert(divs[i].innerHTML);  
    }  
}
```

38.Window 对象的 location 属性和 Document 对象的 location 属性是一样的, 都是引用 Location 对象, 它表示该窗口中当前显示的文档的 URL。

39.在 URL 后面跟一个 javascript:协议限定符, 是另一种嵌入 JavaScript 代码到客户端的方式。这种特殊的协议类型指定 URL 内容为任意字符串, 这个字符串是会被 JavaScript 解释器运行的 JavaScript 代码。

```
<a href="javascript:new Date().toLocaleTimeString();">  
What time is it?  
</a>
```

40.跨站脚本攻击, 简称为 XSS, 指的是恶意攻击者往 Web 页面里插入恶意 html 代码, 当用户浏览该页之时, 嵌入其中 Web 里面的 html 代码会被执行, 从而达到恶意用户的特殊目的。

- 41.通常，防止 XSS 攻击的方式是，在使用任何不可信的数据来动态的创建文档内容之前，从中移除 HTML 标签。
- 42.Window 对象的方法 open()返回代表新创建的窗口的 Window 对象。而且这个新窗口具有 opener 属性，该属性可以打开它的原始窗口。这样，两个窗口就可以相互引用，彼此都可以读取对方的属性或是调用对方的方法。
- 43.Javascript 各种语句的使用方式基本和 Java 相同。
- 44.点击提交按钮时，一般先触发 click 事件，然后再触发 submit 事件。
- 45.如果设置了 form 的 action，则提交按钮会向 action 设置的页面提交数据；如果没有设置 action，则提交会自动刷新当前页面。通过事件处理函数返回 false 可以阻止刷新页面：

```
<form action="submitpage.htm" onsubmit="return validate_form(this);"
method="post">
  Email: <input type="text" name="email" size="30">
  <input type="submit" value="Submit">
</form>
```

- 46.form 在处理点击事件处理函数时，一般先进行 Javascript 代码处理，函数返回时 submit 再提交数据。
- 47.当 submit 的 click 事件返回 false 时，不会执行 form 表单的 submit 事件，返回非 false 值或者没有 return 语句返回内容都会执行表单的 submit 事件。
- 48.document.getElementById 的返回值是一个 HTMLElement 对象的子类，具体而言，对应不同的 Html 元素的 id，其返回的 HTMLElement 子类也不同。例如，如果是 li 标签的 id，则返回一个 HTMLLIElement 对象。
- 49.使用 javascript 来实现滚动文字的原理是设置一个大的块元素，将该元素的可视区域设置小一点以令部分块元素部分内容不可见，这就能移动该块元素以滚动显示不同内容。
- 50.在动态网页中，常常需要在单击超链接时处理一些数据，而不是跳转一个网页。在这种情况下，通常有以下三种处理方式：
- 不设置<a>标签的 href 属性，只设置 onclick 属性。在这种处理方式下，通常超链接文本会和正文的文本以相同的形式出现，即不会有默认的下划线。当鼠标放在超链接上也不会显示小手的形状（除非为该超链接设置了 CSS）。因此，用户很难知道这是一个可以点击的超链接。
 - 将<a>标签的 href 属性值设置为"#", 并设置 onclick 属性。在这种处理方式下，浏览器会自动跳转到当前网页的顶部。
 - 将 href 属性值设置为一个 JavaScript 语句，那么浏览器就会执行该语句。因此，可

以直接将 JavaScript 语句写在<a>标签的 href 属性值中，让 href 属性代替 onclick 属性。

```
<a href="javascript:void(0)" onclick="subgo()">点我</a>
```

将 href 设置为 “javascript:void(0)” ，表示死链接，不执行任何动作，真正执行的是后面的 subgo()函数。

51.html 文件从上到下依次执行，所以<head>会比<body>先执行，只是<head>不在页面上显示。

```
<!DOCTYPE html>
<script>
    alert("顶部脚本");
</script>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        alert("头部脚本");
    </script>
</head>
<body>
    <script>
        alert("页面脚本");
    </script>
</body>
<script>
    alert("底部脚本");
</script>
</html>
```

52.当 html 流碰到 “<script src='xxx'>” 这样的外部 js 文件时，浏览器会在 html 流的当前位置阻塞，去执行外部的 js 文件，js 文件的所有未指定位置的输出都在当前的 html 流中输出。

html 文件

```
<html>
<head>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
</head>
```

```
<body>
<h1 id="test">开始进入 js 文件</h1>
<script type="text/javascript" src='log.js'></script>
<h1 id="test">已经从 js 文件中退出</h1>
</body>
</html>
```

log.js

```
var myH2=$('<h2>这是 js 文件里的输出</h2>');
$('body').append(myH2);
```

输出如下：

开始进入js文件

这是js文件里的输出

已经从js文件中退出

53.

锋利的 jQuery

第 1 章 jQuery 简介

1. 使用 jQuery 需要在页面的<head>引入jQuery 文件：

```
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
  ...
</head>
<body>
  ...
</body>
</html>
```

2. jQuery 函数分为全局函数和对象函数两种：

- 全局函数：就是 jQuery 对象的方法，实际上就是位于 jQuery 命名空间内部的函数。以 “\$.” 开头，例如 \$.get、\$.post 等。这些函数有一个共同特征，就是不直接操作 DOM 元素，而是操作 Javascript 的非元素对象，或者执行其他非对象的特定操作
 - 对象函数：就是对象方法，通过 jQuery 对象方法提供的，通常用于操作 DOM。
3. jQuery 对象就是通过 jQuery 包装 DOM 对象后产生的对象，如果一个对象是 jQuery 对象，那么就可以使用 jQuery 里的方法。
 4. jQuery 提供了两种方法将一个 jQuery 对象转换成 DOM 对象：
 - jQuery 对象是一个数组对象，可以通过中括号[]进行转换，其中 index 是索引。

```
var $cr=$("#cr");           // jQuery 对象
var cr=$cr[0];              // DOM 对象
```

- 在本书中所有以美元符\$开头的变量都是 jQuery 变量，这只是这本书的约束而已。
- 通过 get()方法来获取相应的 DOM 对象。

```
var $cr=$("#cr");           // jQuery 对象
var cr=$cr.get(0);          // DOM 对象
```

5. 对于一个 DOM 对象，只需要用\$()把 DOM 对象包装起来，就可以转换为 jQuery 对象。

```
var cr=document.getElementById("cr");
var $cr=$(cr);
```

6. 许多 JavaScript 库都使用 \$ 作为函数或变量名，在其他库和 jQuery 库都被加载完毕后，可以在任何时候调用 jQuery.noConflict()函数来释放对\$标识符的控制，这样其他脚本就可以使用\$了。

```
<script type="text/javascript" src="other_lib.js"></script>
<script type="text/javascript" src="jquery.js"></script>

<script type="text/javascript">
  $.noConflict();
  // 使用另一个库的 $ 的代码
</script>
```

向其他库移交\$的控制权。

- 这种释放控制权的方式只限于脚本中只有两种 JS 库的\$存在冲突，如果有三个或三个以上的\$冲突，这种方式无法完全解决冲突，因为释放控制权的方式只保证 jQuery 不抢\$的控制权。

7. noConflict() 返回对 jQuery 的引用，所以可以把它存入变量，方便使用。

```
var jq = $.noConflict();
jq(document).ready(function(){
  jq("button").click(function(){
    jq("p").text("jQuery 仍在运行! ");
  });
});
```

其实这个就相当于给 jQuery 变量定义别名。

8. 释放控制权之后，把 \$ 符号作为变量传递给一个方法，这样就可以在函数内使用 \$ 符号了，但在函数外，依旧必须使用 "jQuery"。

```
$.noConflict();
jQuery(document).ready(function($){
  $("button").click(function(){
    $("p").text("jQuery 仍在运行! ");
  });
});
```

第 2 章 jQuery 选择器

1. jQuery 选择器完全继承了 CSS 选择器的风格。利用 jQuery 选择器，可以非常快捷地找出特定的 DOM 元素，然后为它们添加相应的行为。
2. 用 jQuery 选择器获取网页中不存在的元素也不会报错，当要用 jQuery 检查某个元素在网页中是否存在时，应该根据获取到的元素的长度来判断：

```
if($("#tt").length>0){
  .....
}
```

或者转化为 DOM 对象来判断：

```
if($("#tt")[0]){
  .....
}
```

3. jQuery 选择器分为基本选择器、层次选择器、过滤选择器和表单选择器。
4. 基本选择器是 jQuery 中最常用的选择器，也是最简单的选择器，它通过元素 id、class 和标签名等来查找 DOM 元素。在网页中，每个 id 名称只能使用一次，class 允许重复使用。

表 2-2

基本选择器

选 择 器	描 述	返 回	示 例
#id	根据给定的 id 匹配一个元素	单个元素	\$("#test")选取 id 为 test 的元素
.class	根据给定的类名匹配元素	集合元素	\$(".test")选取所有 class 为 test 的元素
element	根据给定的元素名匹配元素	集合元素	\$(p)选取所有的<p>元素
*	匹配所有元素	集合元素	\$("*")选取所有的元素
selector1, selector2, ……, selectorN	将每一个选择器匹配到的元素合并后一起返回	集合元素	\$(div,span,p.myClass)选取所有<div>,和拥有 class 为 myClass 的<p>标签的一组元素

例如：

```
$( '*').css("background"," #bbffaa");
```

改变所有元素的背景色。

- 层次选择器通过 DOM 元素之间的层次关系来获取特定元素，例如后代元素、子元素、相邻元素和兄弟元素等。

表 2-4

层次选择器

选 择 器	描 述	返回	示 例
\$("ancestor descendant")	选取 ancestor 元素里的所有 descendant（后代）元素	集合元素	\$(div span)选取<div>里的所有的元素
\$("parent > child")	选取 parent 元素下的 child（子）元素，与\$("ancestor descendant")有区别，\$("ancestor descendant")选择的是后代元素	集合元素	\$(div > span)选取<div>元素下元素名是的子元素
\$(prev + next')	选取紧接在 prev 元素后的 next 元素	集合元素	\$(one + div)选取 class 为 one 的下一个<div>元素
\$(prev~siblings')	选取 prev 元素之后的所有 siblings 元素	集合元素	\$(#two~div)选取 id 为 two 的元素后面的所有<div>兄弟元素

例如：

```
$('body>div').css("background"," #bbffaa");
```

改变<body>内子<div>元素的背景色。

- 第三个选择器\$(prev+next')可以使用 next()来代替。
- 过滤选择器又分为基本过滤选择器、内容过滤选择器、可见性过滤选择器、属性过滤选择器、子元素过滤选择器和表单对象属性过滤选择器 6 类。
 - 过滤选择器与 CSS 中的伪类选择器语法相同，以一个冒号开头：

表 2-8 基本过滤选择器			
选 择 器	描 述	返 回	示 例
:first	选取第 1 个元素	单个元素	<code>\$("div:first")</code> 选取所有<div>元素中第 1 个<div>元素
:last	选取最后一个元素	单个元素	<code>\$("div:last")</code> 选取所有<div>元素中最后一个<div>元素
:not(selector)	去除所有与给定选择器匹配的元素	集合元素	<code>\$("input:not(.myClass)")</code> 选取 class 不是 myClass 的<input>元素
:even	选取索引是偶数的所有元素，索引从 0 开始	集合元素	<code>\$("input:even")</code> 选取索引是偶数的<input>元素
:odd	选取索引是奇数的所有元素，索引从 0 开始	集合元素	<code>\$("input:odd")</code> 选取索引是奇数的<input>元素
:eq(index)	选取索引等于 index 的元素（index 从 0 开始）	单个元素	<code>\$("input:eq(1)")</code> 选取索引等于 1 的<input>元素
:gt(index)	选取索引大于 index 的元素（index 从 0 开始）	集合元素	<code>\$("input:gt(1)")</code> 选取索引大于 1 的<input>元素（注：大于 1，而不包括 1）
:lt(index)	选取索引小于 index 的元素（index 从 0 开始）	集合元素	<code>\$("input:lt(1)")</code> 选取索引小于 1 的<input>元素（注：小于 1，而不包括 1）

续表			
选 择 器	描 述	返 回	示 例
:header	选取所有的标题元素，例如 h1, h2, h3 等等	集合元素	<code>\$(":header")</code> 选取网页中所有的<h1>，<h2>，<h3>……
:animated	选取当前正在执行动画的所有元素	集合元素	<code>\$("div:animated")</code> 选取正在执行动画的<div>元素

例如：

```
$("p :first")
```

选择第一个 <p> 元素。

8. 内容过滤选择器的过滤规则体现在它所包含的子元素或文本内容上。

表 2-10 内容过滤选择器			
选 择 器	描 述	返回	示 例
:contains(text)	选取含有文本内容为“text”的元素	集合元素	<code>\$("div:contains('我')")</code> 选取含有文本“我”的<div>元素
:empty	选取不包含子元素或者文本的空元素	集合元素	<code>\$("div:empty")</code> 选取不包含子元素（包括文本元素）的<div>空元素
:has(selector)	选取含有选择器所匹配的元素的元素	集合元素	<code>\$("div:has(p)")</code> 选取含有<p>元素的<div>元素
:parent	选取含有子元素或者文本的元素	集合元素	<code>\$("div:parent")</code> 选取拥有子元素（包括文本元素）的<div>元素

例如：

```
$('#div :contains(di)').css("background","#bbffaa");
```

改变含有文本“di”的<div>元素的背景色。

9. 可见性过滤选择器是根据元素的可见和不可见状态来选择相应的元素。

表 2-12 可见性过滤选择器

选 择 器	描 述	返回	示 例
:hidden	选取所有不可见的元素	集合元素	<code>\$(":hidden")</code> 选取所有不可见的元素。包括 <input type="hidden"/> , <code><div style="display:none;"></code> 和 <code><div style="visibility:hidden;"></code> 等元素。如果只想选取 <input>元素, 可以使用<code="">\$("input:hidden")</input>元素,>
:visible	选取所有可见的元素	集合元素	<code>\$("div:visible")</code> 选取所有可见的<div>元素

例如:

```
$('#div:visible').css("background","#ff6500");
```

改变所有可见的<div>元素的背景色。

10. 属性过滤选择器的过滤规则是通过元素的属性来获取相应的元素。

表 2-14 属性过滤选择器

选 择 器	描 述	返回	示 例
[attribute]	选取拥有此属性的元素	集合元素	<code>\$("div[id]")</code> 选取拥有属性 id 的元素
[attribute=value]	选取属性的值为 value 的元素	集合元素	<code>\$("div[title=test]")</code> 选取属性 title 为“test”的<div>元素
[attribute!=value]	选取属性的值不等于 value 的元素	集合元素	<code>\$("div[title!=test]")</code> 选取属性 title 不等于“test”的<div>元素 (注意: 没有属性 title 的<div>元素也会被选取)
[attribute^=value]	选取属性的值以 value 开始的元素	集合元素	<code>\$("div[title^=test]")</code> 选取属性 title 以“test”开始的<div>元素
[attribute\$=value]	选取属性的值以 value 结束的元素	集合元素	<code>\$("div[title\$=test]")</code> 选取属性 title 以“test”结束的<div>元素
[attribute*=value]	选取属性的值含有 value 的元素	集合元素	<code>\$("div[title*=test]")</code> 选取属性 title 含有“test”的<div>元素
[selector1][selector2] [selectorN]	用属性选择器合并成一个复合属性选择器, 满足多个条件。每选择一次, 缩小一次范围	集合元素	<code>\$("div[id][title\$=test]")</code> 选取拥有属性 id, 并且属性 title 以“test”结束的<div>元素

例如:

```
$('#div[title]').css("background","#ff6500");
```

改变含有属性 title 的<div>元素的背景色。

11. 子选择器过滤规则:

表 2-16 子元素过滤选择器			
选 择 器	描 述	返回	示 例
:nth-child (index/even/ odd/equation)	选取每个父元素下的第 index 个子元素或者奇偶元素.(index 从 1 算起)	集合元素	:eq(index)只匹配一个元素,而:nth-child 将为每一个父元素匹配子元素,并 且 :nth-child(index) 的 index 是 从 1 开 始的,而:eq(index) 是从 0 算起的
:first-child	选取每个父元素的第 1 个子元素	集合元素	:first 只返回单个元素,而:first-child 选 择符将为每个父元素匹配第 1 个子元 素。 例如\$("ul li:first-child"); 选取每个 中第 1 个元素
:last-child	选取每个父元素的最后一个子 元素	集合元素	同样, :last 只返回单个元素, 而:last-child 选择符将为每个父元素匹 配最后一个子元素。 例如\$("ul li:first-child"); 选择每个 中最后一个元素
:only-child	如果某个元素是它父元素中惟 一的子元素,那么将会被匹配。 如果父元素中含有其他元素,则 不会被匹配	集合元素	\$("ul li:only-child") 在中选取是惟 一子元素的元素

例如:

```
$('#div.one :nth-child(2)').css("background","#ff6500");
```

改变每个 class 为 one 的<div>父元素下的第 2 个子元素的背景色。

12. :nth-child()是很常用的子元素过滤器,详细功能如下:

- :nth-child(event): 选取每个元素下的索引值是偶数的子元素。
- :nth-child(odd): 选取每个元素下的索引值是奇数的子元素。
- :nth-child(2): 选取每个元素下的索引值等于 2 的子元素。
- :nth-child(3n): 选取每个元素下的索引值是 3 的倍数的子元素。

13. 表单对象属性过滤选择器用于对表单元素进行过滤,例如下拉框、多选框等:

表 2-18 表单对象属性过滤选择器			
选 择 器	描 述	返回	示 例
:enabled	选取所有可用元素	集合元素	\$("#form1 :enabled"); 选取 id 为 “form1” 的表单内的所有可用元素
:disabled	选取所有不可用元素	集合元素	\$("#form2 :disabled")选取 id 为 “form2” 的表单内的所有不可用元素
:checked	选取所有被选中的元素 (单选 框, 复选框)	集合元素	\$(“input:checked”); 选取所有被选中 的<input>元素
:selected	选取所有被选中的选项元素 (下拉列表)	集合元素	\$(“select :selected”); 选取所有被选中 的选项元素

14. 表单选择器用于操作表单：

表 2-20 表单对象属性过滤示例			
选 择 器	描 述	返回	示 例
:input	选取所有的<input>、<textarea>、<select> 和<button>元素	集合元素	<code>\$(":input")</code> 选取所有<input>、<textarea>、<select>和<button>元素
:text	选取所有的单行文本框	集合元素	<code>\$(":text")</code> 选取所有的单行文本框
:password	选取所有的密码框	集合元素	<code>\$(":password")</code> 选取所有的密码框
:radio	选取所有的单选框	集合元素	<code>\$(":radio")</code> 选取所有的单选框
:checkbox	选取所有的多选框	集合元素	<code>\$(":checkbox")</code> 选取所有的复选框
:submit	选取所有的提交按钮	集合元素	<code>\$(":submit")</code> 选取所有的提交按钮
:image	选取所有的图像按钮	集合元素	<code>\$(":image")</code> 选取所有的图像按钮
:reset	选取所有的重置按钮	集合元素	<code>\$(":reset")</code> 选取所有的重置按钮
:button	选取所有的按钮	集合元素	<code>\$(":button")</code> 选取所有的按钮
:file	选取所有的上传域	集合元素	<code>\$(":file")</code> 选取所有的上传域
:hidden	选取所有不可见元素	集合元素	<code>\$(":hidden")</code> 选取所有不可见元素（已经在不可见性过滤选择器中讲解过）

例如：

```
$("#form1 :input").length;
```

获取表单内元素的个数。

15. 选择器中含有“#”、“(”或“]”等特殊字符需要转义，转义使用的是双斜线\\，而不是单斜线：

```
$('#id\\#b');  
$('#id\\[1\\]');
```

匹配 id 值为 id#b 的元素以及 id 值为 id[1]的元素。

16. 在选择器中的空格是非常重要的，多一个空格或少一个空格的结果可能完全不同。

```
var $ta=$('.test :hidden');
```

这段代码冒号前面带空格，表示选取 class 为 test 的元素里面的隐藏元素。

```
var $tb=$('.test:hidden');
```

这段代码冒号没有空格，表示选取隐藏的 class 为 test 的元素，而不是 test 元素里面的隐藏的元素。

第 3 章 jQuery 中的 DOM 操作

1. 使用 jQuery 选择器来在文档树中查找节点。
2. 使用 `attr()` 函数来获取或设置标签的各种属性。

- attr()函数的参数可以是一个，也可以是两个。当 attr()参数只有一个时，参数是要查询的属性的属性名：

```
var $para=$("p");  
var txt=$para.attr("title");  
alert(txt);
```

获取<p>节点的 title 属性值并打印出来。

- 当 attr()有两个参数时，表示设置标签属性，这两个参数分别为属性名和属性对应的值。

```
$("p").attr("title","your title");
```

设置<p>标签的 title 属性。

3. removeAttr()用于为匹配的元素集合中的每个元素中移除一个属性。

```
div.removeAttr('name');
```

删除 name 属性。

4. \$()可以用于创建新元素，它会根据传入的 HTML 标记字符串，创建一个 DOM 对象，并将这个 DOM 对象包装成一个 jQuery 对象后返回。

```
var $li1=$("<li></li>");
```

创建一个节点。

- 这里的元素和标签都是一个意思，都是指<p>、<a>这类 html 标签。

5. 通过\$()和jQuery的 append()创建节点：使用\$()创建新元素，然后使用jQuery的 append()等函数将该元素插入文档中。

```
var $li1=$("<li title='文本内容'>文本内容</li>");  
$("ul").append($li1);
```

新建一个节点，并将其添加到 ul 中。

6. 除了接受字符串，append()还可以传入原始的 DOM 对象、jQuery 对象和函数对象：

```
// 创建 DOM 对象：  
var ps = document.createElement('li');  
ps.innerHTML = '<span>Pascal</span>';  
// 添加 DOM 对象：  
ul.append(ps);
```

```
// 添加 jQuery 对象：  
ul.append($('#scheme'));
```

```
// 添加函数对象：
```

```
ul.append(function (index, html) {  
    return '<li><span>Language - ' + index + '</span></li>';  
});
```

- 如果要添加的 DOM 节点已经存在于 HTML 文档中，它会首先从文档移除，然后再添加，也就是说，用 `append()`，相当移动一个 DOM 节点。

7. 插入节点的函数有以下几个：

- `append()`：向每个匹配的元素内部追加内容。
- `appendTo()`：将所有匹配的元素追加到指定的元素中。该函数和 `append()` 的区别在于，以 `$(A).append(B)` 为例，`append()` 是将 B 追加到 A，而 `appendTo()` 正好相反，将 A 追加到 B。
- `prepend()`：将节点插入匹配的元素前面。
- `prependTo()`：将节点插入匹配的元素前面。`prepend()` 和 `prependTo()` 的区别参考 `appendTo()` 和 `append()` 的区别。
- `after()`：在匹配的元素后面插入节点。
- `insertAfter()`：在匹配的元素后面插入节点。`after()` 和 `insertAfter()` 的区别参考 `appendTo()` 和 `append()` 的区别。
- `before()`：在匹配的元素前面插入节点。
- `insertBefore()`：在匹配的元素前面插入节点。`before()` 和 `insertBefore()` 的区别参考 `appendTo()` 和 `append()` 的区别。

8. jQuery 提供了两种删除节点的方法：

- `remove()`：`remove()` 的参数为空表示删除所有匹配的节点，不为空表示选择性地删除节点。

```
$("#ul li").remove("li[title!=abc]");
```

将 `` 元素中 `title` 不等于 `abc` 的 `` 删除。

- `empty()`：清空节点。

```
$("#ul li").empty();
```

清空 `` 节点。

9. `clone()` 函数用于复制节点。

```
$("#ul li").click(function(){  
    $(this).clone(true).appendTo("ul");  
})
```

复制当前单击的节点，并将其追加到 `` 元素中。

- `clone()` 的参数为 `true` 表示复制之后的新元素也可以被复制。

➤ 通过鼠标拖动将商品加入购物车的功能就是使用节点复制功能实现的。

10. 替换节点的函数有两个：

- `replaceWith()`：将所有匹配的元素替换成参数指定的 html 或 dom 元素。

```
$("#p").replaceWith("<strong>你最不喜欢的水果是？ </strong>");
```

将 p 元素文本替换成“你最不喜欢的水果是？”。

- `replaceAll()`：作用和 `replaceWith()` 相同，只是参数位置变了。

```
$("#<strong>你最不喜欢的水果是？ </strong>").replaceWith("p");
```

将 p 元素文本替换成“你最不喜欢的水果是？”。

11. 使用 `wrap()` 函数将某个节点用其他标记包裹起来，其实就是在外层插入一个标签：

```
$("#strong").wrap("<b></b>");
```

用 `` 标签将 `` 标签包裹起来，就是在 `` 标签外围插入一个 `` 标签。

12. `wrapAll()` 函数也用于将节点包裹起来，用法与 `wrap()` 函数相同，不同于 `wrap()` 函数的是，`wrapAll()` 函数将整个元素作为一个整体包裹，例如：

```
<b>
<strong>苹果</strong>
<strong>香蕉</strong>
</b>
```

而 `wrap()` 函数是将每个标签单独包裹：

```
<b><strong>苹果</strong></b>
<b><strong>香蕉</strong></b>
```

13. `wrapInner()` 函数用于将每个匹配的元素的内容用其他结构化的标记包裹起来。该函数包裹的是标签的文本，而不是这个标签本身。

```
$("#strong").wrapInner("<b></b>");
```

效果如下：

```
<strong><b>香蕉</b></strong>
```

14. 在 jQuery 中，用 `attr()` 方法来获取和设置元素属性，`removeAttr()` 方法删除元素属性。

```
$("#p").removeAttr("title");
```

删除标签的 `title` 属性。

15. `addClass()` 用于为标签添加 `class` 属性。


```
$("#p").addClass("another");
```

16.removeClass()方法用于删除标签的 class 属性:

```
$("#p").removeClass("another");
```

删除<p>标签中的 class 属性值 another。

17.toggleClass() 方法用于对被选元素的一个或多个类在设置或移除之间进行切换, 如果类名存在则删除, 如果类名不存在则添加。

```
$("#button").click(function(){  
    $("#p").toggleClass("main");  
});
```

如果<p>存在名为 main 的 class 属性值, 则删除 main; 如果不存在, 则添加。

18.hasClass()用于判断元素中是否含有某个 class 值, 如果有, 则返回 true, 否则返回 false。

```
$("#p").hasClass("another");
```

判断<p>标签是否有名为 another 的 class 值。

19.html()函数类似 JS 中的 innerHTML 属性, 可以用来读取或设置某个元素中的 HTML 内容。

```
var phtml=$("#p").html();
```

获取<p>标签的 HTML 代码。

20.text()类似于 JS 中的 innerText 属性, 可以用来读取或设置某个元素中的文本的内容。

```
var ptext=$("#p").text();
```

获取<p>标签的文本的内容。

21.val()可以用来设置和获取输入框等 html 标签的值。无论元素是文本框、下拉列表还是单选框, 它都可以返回元素的值。如果元素为多选, 则返回一个包含所有选择的值的数组。

22.val()也可以使下拉列表、多选框和单选框中相应的选项被选中。

23.children()方法用于获取匹配元素的子元素集合。注意是子元素, 而不是后代元素。

```
var $p=$("#p").children();
```

获取<p>元素的所有子元素。

24.next()方法用于获取匹配元素后面紧邻的同辈元素。

```
var $p=$("#p").next();
```

获取紧邻<p>元素的同辈元素。

- 注意，这里的同辈元素不光是相同的元素，例如<a>标签和<p>标签处于同一层次，则这两个标签就是同辈元素。

25.prev()用于获取匹配元素前面紧邻的同辈元素。

```
var $ul=$("ul").prev();
```

26.siblings()方法用于获取匹配元素前后所有的同辈元素。

```
var $p=$("p").siblings();
```

27.closest()方法用于获取最近的匹配元素。它首先检查当前元素是否匹配，如果匹配则直接返回，不匹配则向上查找父元素，逐级向上直到找到匹配选择器的元素。如果什么都没找到则返回一个空的jQuery对象。

28.CSS-DOM 技术用来读取和设置 style 对象的各种属性。

29.css() 方法用于设置或返回被选元素的一个或多个样式属性。一个参数时为获取属性值，两个参数时为设置。

```
$("p").css("background-color");
```

返回首个匹配元素的 background-color 值。

```
$("p").css("background-color","yellow");
```

所有匹配元素设置 background-color 值。

30.css()可以同时设置多个样式属性，使用大括号来实现：

```
$("p").css({"fontSize":"30px","backgroundColor":"#888888"});
```

同时设置字体大小和背景色。

- attr()和 css()一样，也可以同时设置多个属性值。

31.offset()方法用于获取元素在当前视窗的相对偏移，其中返回的对象包含两个属性，即 top 和 left，它只对可见元素有效。

```
var offset=$("p").offset();
```

```
var left=offset.left;
```

```
var top=offset.top;
```

获取<p>标签的偏移值。

32.position()方法用于获取元素相对于最近一个 position 样式属性设置为 relative 或者 absolute 的祖父节点的相对偏移，与 offset()一样，它返回的对象也包括 top 和 left 两个属性。

```
var position=$("#p").position();
var left=position .left;
var top=position .top;
```

33.scrollTop()方法用于获取元素的滚动条距顶端的距离，scrollLeft()方法用于获取元素的滚动条距左侧的距离。

```
var $p=$("#p");
var scrollTop=$p.scrollTop();
var scrollLeft=$p.scrollLeft();
```

为这两个方法指定一个参数，控制元素的滚动条滚动到指定位置。

34.find()函数用于查找指定元素的子元素。

```
<!-- HTML 结构 -->
<ul class="lang">
  <li class="js dy">JavaScript</li>
  <li class="dy">Python</li>
  <li id="swift">Swift</li>
  <li class="dy">Scheme</li>
  <li name="haskell">Haskell</li>
</ul>
```

find()使用:

```
var swf = ul.find('#swift');
```

获取 ul 标签子元素中 id 为 swift 的元素。

35.parent()函数用于查找指定元素的父元素:

```
var swf = $('#swift');
var parent = swf.parent();
var a = swf.parent('.red');
```

获取 id 为 swift 的元素的所有父元素以及类名为 red 的父元素。

36.使用 next()和 prev()在同一级元素中查找:

```
swift.next();
swift.next('[name=haskell]');
```

查找同级元素中 name 等于 haskell 的元素。

37.filter()方法可以过滤掉不符合选择器条件的节点:

```
var langs = $('ul.lang li');
var a = langs.filter('.dy');
```

过滤掉 class 等于 dy 的元素。

38.即使选择器没有返回任何 DOM 节点，调用 jQuery 对象的方法仍然不会报错：

```
$('#not-exist').text('Hello');
```

代码不报错，没有节点被设置为'Hello'。

39.利用 jQuery 对象的 height()和 weight()可以获取 DOM 的高宽等信息：

```
// 浏览器可视窗口大小:  
$(window).width(); // 800  
$(window).height(); // 600  
  
// HTML 文档大小:  
$(document).width(); // 800  
$(document).height(); // 3500  
  
// 某个 div 的大小:  
var div = $('#test-div');  
div.width(); // 600  
div.height(); // 300  
div.width(400); // 设置 CSS 属性 width: 400px, 是否生效要看 CSS 是否有效  
div.height('200px')
```

40.jQuery 对象统一提供 val()方法获取和设置对应的 value 属性：

```
var  
  input = $('#test-input'),  
input.val('abc@example.com');
```

设置文本框内容为 abc@example.com。

41.

第 4 章 jQuery 中的事件和动画

1. on 方法用来绑定一个事件，我们需要传入事件名称和对应的处理函数。

```
var a = $('#test-link');  
a.on('click', function () {  
  alert('Hello!');  
});
```

另一种更简化的写法是直接调用 click()方法：

```
a.click(function () {  
    alert('Hello!');  
});
```

两者完全等价，通常用后面的写法。

2. 在JS中，当一个资源及其依赖资源已完成加载时，将触发load事件，load事件的处理方法是window.onload方法。
3. jQuery使用\$(document).ready()方法来代替传统的window.onload方法。
4. \$(document).ready()方法和window.onload方法有相似的功能，但是在执行时机方面是有区别的。window.onload方法是在网页中所有的元素完全加载到浏览器后才执行，即JavaScript此时才可以访问网页中的任何元素。而通过jQuery中的\$(document).ready()方法注册的事件处理程序，在DOM完全就绪时就可以被调用。此时，网页的所有元素对jQuery而言都是可以访问的，但是，这并不意味着这些元素关联的文件都已经下载完毕。

举一个例子，有一个大型的图库网站，为网页中所有图片添加某些行为，例如单击图片后让它隐藏或显示。如果使用window.onload方法来处理，那么用户必须等到每一幅图片都加载完毕后，才可以进行操作。如果使用jQuery中的\$(document).ready()方法进行设置，只要DOM就绪就可以操作了，不需要等待所有图片下载完毕。很显然，把网页解析为DOM树的速度比把页面中的所有关联文件加载完毕的速度快很多。

5. window.onload无法同时指定多个事件处理函数，而\$(document).ready()方法可以。

```
window.onload=one;  
window.onload=two;
```

只能执行第二条。

6. \$(document).ready()方法可以简写，例如：

```
$(document).ready(function(){  
    alert(' i am ready');  
});
```

可以简写为：

```
$(function(){  
    alert("i am in");  
});
```

7. 使用bind()方法对匹配元素进行特定事件的绑定，它有三个参数，分别为事件类型、规定传递到函数的额外数据、以及事件处理函数，其中传递的额外数据是可选的：

```
$("button").bind("click",function(){
```

```
$("#p").slideToggle();
});
```

8. click、mouseover 和 mouseout 这类经常使用的事件，jQuery 提供了一套绑定事件处理程序的简写方法，类似于直接调用一个函数：

```
$(function(){
    $("#panel h5.head").mouseover(function(){
        $(this).next("div.content").show();
    });
});
```

设置 mouseover 事件处理函数。上面这段代码是下面这段代码的简写：

```
$(function(){
    $("#panel h5.head").bind("mouseover",function(){
        $(this).next("div.content").show();
    });
});
```

- 当鼠标指针位于元素上方时，会发生 mouseover 事件。当鼠标指针从元素上移开时，发生 mouseout 事件。
9. hover() 方法用于设置当鼠标指针悬停在被选元素上时要运行的两个函数，当光标移动到元素上时，会触发第 1 个函数；当光标移出这个元素时，会触发第 2 个函数。其中第 2 个函数是可选的。如果只有一个函数作为参数，则光标进入和移出都会触发该函数。

```
$("#p").hover(function(){
    $("#p").css("background-color","yellow");
},function(){
    $("#p").css("background-color","pink");
});
```

当鼠标指针悬停在上面时，改变 <p> 元素的背景颜色。

- 其实就是 mouseenter 和 mouseleave 的事件处理程序。

10. toggle()方法用于设置鼠标连续单击事件的事件处理程序。第 1 次单击元素，会触发指定的第 1 个函数；当再次单击同一元素时，会触发第 2 个函数，依此类推：

```
$("#p").toggle(
function(){
    $("body").css("background-color","green");},
function(){
    $("body").css("background-color","red");},
```

```
function(){  
    $("body").css("background-color","yellow");  
};
```

11. 事件冒泡指的是事件首先被当前元素的事件处理程序捕获并触发，然后逐级向上传递，父标签的事件处理程序也捕获事件并触发，依此类推。

12. 在事件处理函数中传入一个 event 参数，可以获取事件对象的一些属性：

```
$("#element").bind("click",function(event){  
    .....  
});
```

13. jQuery 提供了 event.stopPropagation()方法来停止事件冒泡。

```
$("#element").bind("click",function(event){  
    var txt=$('#msg').html()+"<p>内层 span 元素被单击</span>"  
    $('#msg').html(txt);  
    event.stopPropagation();  
});
```

停止事件冒泡。

14. jQuery 提供了 event.preventDefault()方法来阻止元素的默认行为。

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

15. event.type()方法可以获取到事件的类型。

```
$("#a").click(function(event){  
    alert(event.type);  
    return false;  
});
```

16. event.target()方法用于获取触发事件的元素。

17. event.pageX()方法和 event.pageY()方法用于获取到事件发生时光标相对页面的 x 坐标和 y 坐标。

18. event.which()方法用于获取哪个鼠标单击，左中右键分别会返回 1、2、3。如果是在键盘事件中，则返回键盘的按键。

19. unbind() 方法移除被选元素的事件处理程序。

```
$("#button").click(function(){
```

```
$("#p").unbind();
});
```

所有参数都是可选的。如果没有提供参数，则删除所有绑定的事件。如果提供参数，第 1 个参数是事件类型，第 2 个是要移除的函数。

20. `one()` 方法为被选元素附加一个或多个事件处理程序，并规定当事件发生时运行的函数。当使用 `one()` 方法时，每个元素的事件处理器函数只能运行一次，随后被删除。

```
$("#p").one("click",function(){
    $(this).animate({fontSize:"+=6px"});
});
```

21. `trigger()` 方法触发被选元素的指定事件类型。

```
$('#btn').trigger("click");
```

触发鼠标单击事件，也可以直接用简化写法 `click()`：

```
$('#btn').click();
```

22. 使用 `bind()` 方法和 `trigger()` 方法还可以设置自定义事件。

```
$('#btn').bind("myClick",function(){
    $('#test').append("<p>我的自定义事件</p>");
});
```

设置自定义事件处理程序，然后使用 `trigger()` 方法触发事件：

```
$('#btn').trigger("myClick");
```

23. `bind()` 方法可以通过指定多个事件类型来一次性绑定多个事件类型：

```
$(function(){
    $("div").bind("mouseover mouseout",function(){
        $(this).toggleClass("over");
    });
});
```

24. jQuery 可以为在事件类型后面添加命名空间来为事件定义命名空间：

```
$(function(){
    $('div').bind("click.plugin",function(){
        .....
    });
});
```


定义事件的命名空间 plugin，这样删除命名空间即可将命名空间内的所有事件全部删除：

```
$('#div').unbind(".plugin")
```

注意后面有点号。

25. show()和 hide()是jQuery 中最基本的动画方法，它通过不断地同时改变宽度和高度的方式来显示或隐藏元素。根据参数的不同，动画效果也不同：

- 不带任何参数：立即显示或隐藏，不会具有动画效果。
- slow、normal 和 fast：分别表示在 600 毫秒、400 毫秒或 200 毫秒内慢慢地显示或隐藏。

```
$("#element").show(slow);
```

- 指定一个数字：该数字的单位是毫秒，表示在指定毫秒内慢慢地显示或隐藏。

```
$("#element").show(1000);
```

26.toggle()方法则根据当前状态决定是 show()还是 hide()，如果当前是显示的，则调用 toggle()之后变为隐藏；如果当前是隐藏的，则调用之后变为显示。

```
$(document).ready(function(){  
    $(".btn1").click(function(){  
        $("p").toggle();  
    });  
});
```

当<p>为显示状态时，按一个按钮变为隐藏状态；当<p>为隐藏状态时，按一个按钮变为显示状态。

27. fadeIn()方法和 fadeOut()方法用于通过不断改变元素的不透明度的方式来显示或隐藏元素。

28. slideUp()方法和 slideDown()方法通过不断改变元素的高度的方式来显示或隐藏元素。

29.所有事件都会传入 Event 对象作为参数，可以从 Event 对象上获取到更多的信息：

```
$(function () {  
    $('#testMouseMoveDiv').mousemove(function (e) {  
        $('#testMouseMoveSpan').text('pageX = ' + e.pageX + ', pageY = ' + e.pageY);  
    });  
});
```

30.通过 off()函数来解除事件和事件处理程序之间的绑定：

```
function hello() {  
    alert('hello!');  
}  
  
a.click(hello); // 绑定事件  
  
// 10 秒钟后解除绑定:  
setTimeout(function () {  
    a.off('click', hello);  
}, 10000);
```

省略掉第二个参数表示解除所有 click 事件的处理函数。

```
off('click')
```

31. 在浏览器中，有些 JavaScript 代码只有在用户触发下才能执行，使用代码无法触发。例如，`window.open()` 函数：

```
$(function () {  
    window.open('/');  
});
```

无法弹出新窗口，将被浏览器屏蔽。

32. 在渐变式的显示或隐匿过程中，`show()` 和 `hide()` 是从左上角逐渐展开或收缩的，而 `slideUp()` 和 `slideDown()` 则是在垂直方向逐渐展开或收缩的。

```
var div = $('#test-slide');  
div.slideUp(3000);
```

在 3 秒钟内逐渐向上消失。

33. `slideUp()` 和 `slideDown()` 的作用和 `show()` 和 `hide()` 函数是相同的，只是展开或收缩的方向不同。

34. `slideToggle()` 的效果和 `Toggle()` 函数的作用是一样的，只是显示效果不同。

35. `fadeIn()` 和 `fadeOut()` 的动画效果是淡入淡出，也就是由深到浅地隐藏，或由浅到深地显示：

```
var div = $('#test-fade');  
div.fadeOut('slow');
```

36. `fadeToggle()` 则根据元素是否可见来决定下一步动作。如果当前可见，则调用之后变为不可见。

37. jQuery 的 `jqXHR` 对象类似一个 `Promise` 对象，我们可以用链式写法来处理各种回

调：

```
'use strict';

function ajaxLog(s) {
    var txt = $('#test-response-text');
    txt.val(txt.val() + '\n' + s);
}

$('#test-response-text').val("");
var jqxhr = $.ajax('/api/categories', {
    dataType: 'json'
}).done(function (data) {
    ajaxLog('成功, 收到的数据: ' + JSON.stringify(data));
}).fail(function (xhr, status) {
    ajaxLog('失败: ' + xhr.status + ', 原因: ' + status);
}).always(function () {
    ajaxLog('请求完成: 无论成功或失败都会调用');
});
```

38.

第 5 章 jQuery 对表单、表格的操作及更多应用

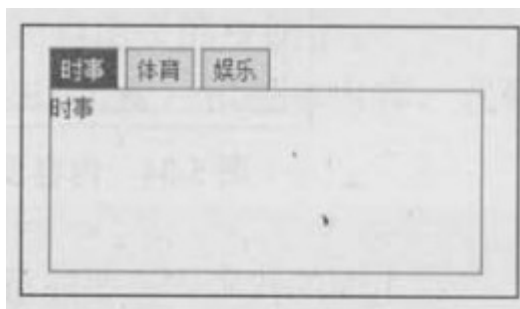
1. 表单分为 3 个组成部分：

- 表单标签：就是<input>这种 html 标签。
- 表单域：包含文本框、密码框等。
- 表单按钮。

2. focus() 方法用于指定元素获得光标焦点时的事件处理程序：

```
$("input").focus(function(){
    $("input").css("background-color","#FFFFCC");
});
```

3. 网页选项卡实际上就是使用高亮实现的，将当前的选项卡设定为不同于其它的选项卡：



4. 网页换肤的原理是通过调用不同的样式表文件来实现不同皮肤的切换，并且需要将换好的皮肤记入 cookie 中，这样用户下次访问时，就可以显示自定义的皮肤了。

第 6 章 jQuery 与 Ajax 的应用

1. Ajax 的核心是 XMLHttpRequest 对象，它是 Ajax 实现的关键，发送异步请求、接收响应和执行回调函数都是通过它来完成的。
2. load() 方法通过 AJAX 请求从服务器加载数据，并把返回的数据放置到指定的元素中。它有 3 个参数：
 - url: 将请求发送到哪个 URL。
 - data: 发送到服务器的数据，是一个字典类型的值。可选。
 - function(): 规定当请求完成时运行的回调函数。可选。

```
$("#button").click(function(){
    $("#div").load('demo.html');
});
```

当 demo.html 加载完成后，demo.html 将会被插入 div 标签中。

3. 可以在 load() 方法的 URL 参数中指定选择器来筛选出所需要的内容：

```
$("#button").click(function(){
    $("#div").load('demo.html .para');
});
```

加载 demo.html 页面中 class 为 para 的内容。注意 “html” 和 “.para” 之间有一个空格。

4. load() 方法的传递方式根据参数 data 来自动指定。如果没有 data 参数，则采用 get 方式传递，如果有 data 参数，则采用 post 方式传递。

```
$("#resText").load("test.php",{name:"rain",age:"22"},function(){
    .....
});
```

有 data 参数，所以采用 post 方式传递。

5. \$.get() 方法用于向服务器发送 get 请求，而 \$.post() 方法用于向服务器发送 post 请

求。这两个方法的作用和 load() 函数差不多，都是从服务器中下载数据到本地，不同的是，这两个方法是 jQuery 的全局函数，而 load() 是针对 jQuery 对象进行操作的，load() 的返回值会插入到指定的 DOM 树中。

6. \$.get() 方法的参数有四个：

- url: 将请求发送的哪个 URL。必需。
- data: 连同请求发送到服务器的数据。可选。
- success(response,status,xhr): 当请求成功时运行的函数。可选。
- dataType: 规定预计的服务器响应的数据类型。可选。jQuery 可以智能地判断数据类型。可能的类型有 "xml"、"html"、"text"、"script"、"json" 和 "jsonp"

```
$("#button").click(function(){
$.get("demo_ajax_load.txt", function(result){
    $("#div").html(result);
});
});
```

7. \$.getScript() 方法用于从服务器中加载 js 文件，可以实现按需要加载 js 文件，而不是一开始就将所有 js 文件加载下来。

```
$("#button").click(function(){
$.getScript("demo_ajax_script.js");
});
```

8. \$.getJSON() 方法用于从服务器中加载 JSON 文件。

```
$("#button").click(function(){
$.getJSON("demo_ajax_json.js",function(result){
    .....
});
});
});
```

9. \$.ajax() 方法是 Ajax 最底层的方法，\$.get() 方法和 \$.post() 方法等都是使用 \$.ajax() 方法来实现的。

10. serialize() 方法通过序列化表单值，创建 URL 编码文本字符串。该方法的返回值是字符串。

```
$("#button").click(function(){
$("#div").text($("#form").serialize());
});
```

```
});
```

11.serializeArray() 方法通过序列化表单值来创建对象数组。该方法的返回值是 JSON 格式的数据。

```
$("#button").click(function(){
    x=$("#form").serializeArray();
    $.each(x, function(i, field){
        $("#results").append(field.name + ":" + field.value + " ");
    });
});
```

12.param() 方法用于对数组或对象按照 key/value 进行序列化。

```
var params = { width:1900, height:1200 };
var str = jQuery.param(params);
$("#results").text(str);
```

结果为:

```
width=1680&height=1050
```

13.jQucry 提供了一些自定义全局函数，能够为各种与 Ajax 相关的事件注册回调函数。例如当 Ajax 请求开始时，会触发 ajaxStart() 方法的回调函数：为 Ajax 请求结束时，会触发 ajaxStop() 方法的回调函数。这些方法都是全局方法，因此无论创建它们的代码位于何处，只要有 Ajax 请求发生时，就会触发它们。

第 7 章 插件的使用和写法

1. Validation 是 jQuery 的一个表单验证插件，它内置了 e-mail、信用卡等验证规则。使用 Validation 需要在 <head> 标签中导入 Validation:

```
<script src="../js/jquery.validate.js" type="text/javascript"></script>
```

2. 通过将 <input> 等标签的 class 属性设置为指定的验证规则来进行表单验证:

```
<input id="password" name="password" type="password"
class="{required:true,minlength:5}" />
```

"{required:true,minlength:5}" 表示表单是必填表单，最小长度为 5。

3. 默认校验规则:

- (1)required:true: 必填字段。
- (2)remote:"check.php": 使用 ajax 方法调用 check.php 验证输入值。
- (3)email:true: 必须输入正确格式的电子邮件。
- (4)url:true: 必须输入正确格式的网址。
- (5)date:true: 必须输入正确格式的日期。
- (6)dateISO:true: 必须输入正确格式的日期(ISO), 例如: 2009-06-23, 1998/01/22 只验证格式, 不验证有效性。
- (7)number:true: 必须输入合法的数字。
- (8)digits:true: 必须输入整数。
- (9)creditcard: 必须输入合法的信用卡号。
- (10) equalTo:"#field": 输入值必须和#field 相同
- (11) accept: 输入拥有合法后缀名的字符串, 即上传文件的后缀。
- (12) maxlength:5: 输入长度最多是 5 的字符串, 汉字算一个字符。
- (13) minlength:10: 输入长度最小是 10 的字符串, 汉字算一个字符。
- (14) rangelength:[5,10]: 输入长度必须介于 5 和 10 之间的字符串, 汉字算一个字符。
- (15) range:[5,10]: 输入值必须介于 5 和 10 之间。
- (16) max:5: 输入值不能大于 5。
- (17) min:10: 输入值不能小于 10。

```
<input id="password" name="password" type="password"
class="{required:true,minlength:5}" />
```

4. Cookie 插件用于管理 Cookie。使用\$.cookie()方法写入或读取 cookie, 根据参数的数量来决定读取还是写入:

- 只有一个参数: 读取 cookie。

```
$.cookie('theCookie');
```

- 两个参数: 写入 cookie。

```
$.cookie('theCookie','cookieValue');
```

- 第 2 个参数为 null: 删除 cookie。

```
$.cookie('theCookie',null);
```

- 三个参数：写入 cookie，并设置 cookie 的相关属性。

```
$.cookie('the_cookie','the_value',  
{expires:7,path:'/',domain:'jquery.com',secure:true});
```

5.

Bootstrap

第 1 章 栅格系统

1. 前端设计分为响应式设计和自适应式设计两种；
 - 自适应式设计：针对不同的屏幕大小，在服务器端准备多套前端，然后服务器根据请求头中的浏览器信息来返回合适的前端文件。
 - 响应式设计：自始至终都只有一套前端，前端模板会根据屏幕宽度，自动调整网页内容大小。
2. 要使用 Bootstrap，必须导入以下的文件：
 - 在<head>标签中加载 Bootstrap 的 CSS：

```
<link rel="stylesheet"  
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"  
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x  
9JvoRxT2MZw1T" crossorigin="anonymous">
```

- 在</body>的前面加载 Bootstrap 的 js 文件：

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-  
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"  
crossorigin="anonymous"></script>  
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"  
integrity="sha384-  
UO2eT0CpHqdSjQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W  
1" crossorigin="anonymous"></script>  
<script
```



```
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0x
IM+B07jRM" crossorigin="anonymous"></script>
```

例如：

```
<!doctype html>
<html lang="en">
  <head>
    .....
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x
9JvoRxT2MZw1T" crossorigin="anonymous">

  </head>
  <body>
    .....
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8a
btTE1Pi6jizo" crossorigin="anonymous"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-
UO2eT0CpHqdSjQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W
1" crossorigin="anonymous"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0x
IM+B07jRM" crossorigin="anonymous"></script>
  </body>
</html>
```

3. Bootstrap 要求使用 HTML5，所以需要添加 HTML5 doctype 声明。

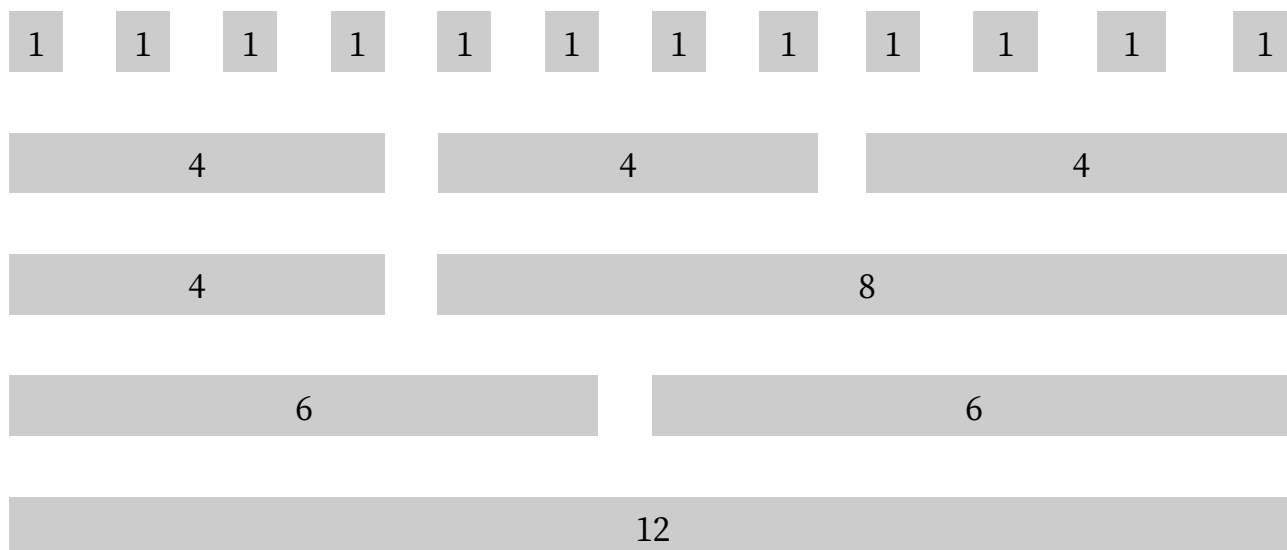
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
</html>
```

4. 在网页的 head 之中添加 viewport meta 标签，以便让 Bootstrap 开发的网站对移动设备友好，确保适当的绘制和触屏缩放。

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

- width=device-width: 表示宽度是设备屏幕的宽度。
- initial-scale=1: 表示初始的缩放比例。
- shrink-to-fit=no: 自动适应手机屏幕的宽度。

5. 网格系统: Bootstrap 提供了一套响应式、移动设备优先的流式网格系统，随着屏幕或视口 viewport 尺寸的增加，系统会自动分为最多 12 列。



6. Bootstrap 的容器类用于包裹网站的内容，Bootstrap 提供了两个容器类：

- .container : 用于固定宽度并支持响应式布局的容器。
 - .container-fluid: 用于 100% 宽度，占据全部视口 (viewport) 的容器。
- 事实上这两个的效果是差不多的，只不过 container 有 margin，而 container-fluid 则没有 margin，宽度始终是 100%。

```
<div class="container-fluid">  
<h1>我的第一个 Bootstrap 页面</h1>  
<p>使用了 .container-fluid, 100% 宽度，占据全部视口 (viewport) 的容器。</p>  
</div>
```

7. Bootstrap 4 网格系统预定义了 5 种类：

- .col-: 针对所有设备。

- `.col-sm-`: 平板, 屏幕宽度等于或大于 576px。
- `.col-md-`: 桌面显示器, 屏幕宽度等于或大于 768px。
- `.col-lg-`: 大桌面显示器, 屏幕宽度等于或大于 992px。
- `.col-xl-`: 超大桌面显示器, 屏幕宽度等于或大于 1200px。

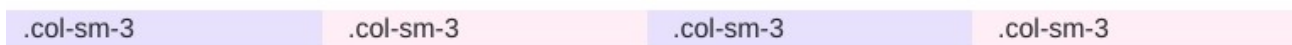
```
<body>
<div class="container-fluid">
  <h1>Hello World!</h1>
  <p>重置浏览器大小查效果。</p>
  <p>在移动设备上, 即屏幕宽度小于 576px 时, 四个列将会上下堆叠排版。</p>
  <div class="row">
    <div class="col-sm-3" style="background-color:lavender;">.col-sm-3</div>
    <div class="col-sm-3" style="background-color:lavenderblush;">.col-sm-3</div>
    <div class="col-sm-3" style="background-color:lavender;">.col-sm-3</div>
    <div class="col-sm-3" style="background-color:lavenderblush;">.col-sm-3</div>
  </div>
</div>
</body>
```

显示效果为:

Hello World!

重置浏览器大小查效果。

在移动设备上, 即屏幕宽度小于 576px 时, 四个列将会上下堆叠排版。



- 这些预定义类后接数字, 表示占据 12 列栅格中的多少列, 例如 `col-sm-6` 表示占据 12 列栅格中的 6 列, `col-sm-3` 表示占据栅格中的 3 列, 依此类推。如果占据的总列数不到 12 列, 则栅格最后几列会有空白:

Hello World!

创建三个相等宽度的列! 尝试在 `class="row"` 的 `div` 中添加新的 `class="col"` `div`, 会显示四个等宽的列。



8. Bootstrap4 网格系统规则:

(1) 使用 row 来创建水平的列组。

(2) 网格每一个 row 需要放在设置了 .container 或 .container-fluid 类的容器中, 这样就可以自动设置一些外边距与内边距。

(3) 内容需要放置在列中, 并且只有列可以是 row 的直接子节点。

```
<div class="row">
  <div class="col-sm-3" style="background-color:lavender;">.col-sm-3</div>
  <div class="col-sm-3" style="background-color:lavenderblush;">.col-sm-3</div>
  <div class="col-sm-3" style="background-color:lavender;">.col-sm-3</div>
  <div class="col-sm-3" style="background-color:lavenderblush;">.col-sm-3</div>
</div>
```

9. 下表列出了栅格系统在不同的屏幕上的类的对应关系:

	手机 <576px	平板 ≥576px	桌面显示器 ≥768px	大桌面显示器 ≥992px	超大桌面显示器 ≥1200px
.container 最大宽度	None (auto)	540px	720px	960px	1140px
类前缀	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
列数	12				

10. 如果屏幕像素小于当前类中使用的像素最小的类, 则所有的列都会垂直显示。例如:

```
<div class="container-fluid">
  <h1>Hello World!</h1>
  <div class="row">
    <div class="col-md-4 col-lg-4 col-sm-4" style="background-
color:lavender;">.col</div>
    <div class="col-md-4 col-lg-4 col-sm-4" style="background-color:orange;">.col</
div>
    <div class="col-md-4 col-lg-4 col-sm-4" style="background-
color:lavender;">.col</div>
  </div>
</div>
```

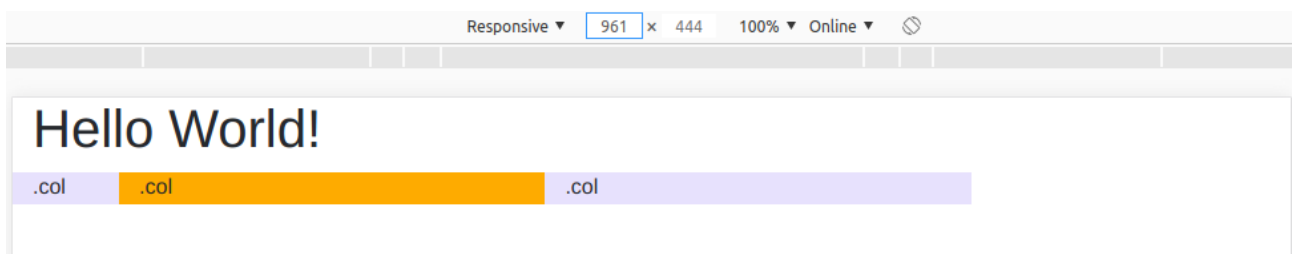
当前类中使用的像素最小的类为 col-sm, 表示平板, 最低像素 576, 所以当屏幕像素小于 576 时, 所有的列都会垂直显示:



11. 如果屏幕像素大于当前类中使用的像素最大的类，则所有的列都会按最大像素的类指定的规则显示。例如：

```
<div class="container-fluid">
  <h1>Hello World!</h1>
  <div class="row">
    <div class="col-sm-1" style="background-color:lavender;">.col</div>
    <div class="col-sm-4" style="background-color:orange;">.col</div>
    <div class="col-sm-4" style="background-color:lavender;">.col</div>
  </div>
</div>
```

显示效果为：



12. 列偏移用于将两个列隔开，如果不希望两个相邻的列紧挨在一起，就可以使用栅格系统的列偏移功能。使用 offset 来使用列偏移， offset-md-1 表示在桌面显示器中将按往右移动 1 格，例如：

```
<div class="container-fluid">
  <div class="row">
```

```
<div class="col-md-4 bg-success">.col-md-4</div>
<div class="col-md-4 offset-md-1 bg-warning">.col-md-4</div>
</div>
</div>
```

显示效果如下：

偏移列

.offset-md-4 是把.col-md-4 往右移了1列格。

.col-md-4

.col-md-4

第2章 文字排版

1. Bootstrap 4 默认设置如下：

- 默认的 font-size 为 16px, line-height 为 1.5。
- 默认的 font-family 为 "Helvetica Neue", Helvetica, Arial, sans-serif。

2. Bootstrap 还提供了四个 Display 类来控制标题的样式：display-1、display-2、display-3、display-4。

```
<div class="container">
  <h1>Display 标题</h1>
  <p>Display 标题可以输出更大更粗的字体样式。</p>
  <h1 class="display-1">Display 1</h1>
  <h1 class="display-2">Display 2</h1>
  <h1 class="display-3">Display 3</h1>
  <h1 class="display-4">Display 4</h1>
</div>
```

和<h1>一样，数字越小，字体越大。

3. <small> 元素用于创建字号更小的颜色更浅的文本：

```
<div class="container">
  <h1>h1 标题 <small>副标题</small></h1>
</div>
```

显示效果如下：

h1 标题 副标题

4. <mark> 用于为文字指定一定的内边距:

```
<div class="container">  
<p>使用 mark 元素来 <mark>高亮</mark> 文本。</p>  
</div>
```

5. text-left、text-center 和 text-right 分别表示文本左边对齐、中间对齐和右边对齐:

```
<div class="container">  
<h2>排版</h2>  
<p class="text-left">左对齐</p>  
<p class="text-right">右对齐</p>  
<p class="text-center">居中对齐文本</p>  
</div>
```

效果如下:

排版

左对齐

右对齐

居中对齐文本

第3章 颜色

1. Bootstrap 提供了一些有代表意义的文本颜色类:

- (1) text-muted: 柔和的文本。
- (2) text-primary: 重要的文本。
- (3) text-success: 执行成功的文本。
- (4) text-info: 代表一些提示信息的文本。
- (5) text-warning: 警告文本。
- (6) text-danger: 危险操作文本。
- (7) text-secondary: 副标题。
- (8) text-dark: 深灰色文字。

(9) text-light: 浅灰色文本。

(10) text-white: 白色文本。

```
<div class="container">
  <h2>代表指定意义的文本颜色</h2>
  <p class="text-muted">text-muted: 柔和的文本。</p>
  <p class="text-primary">text-primary: 重要的文本。</p>
  <p class="text-success">text-success: 执行成功的文本。</p>
  <p class="text-info">text-info: 代表一些提示信息的文本。</p>
  <p class="text-warning">text-warning: 警告文本。</p>
  <p class="text-danger">text-danger: 危险操作文本。</p>
  <p class="text-secondary">text-secondary: 副标题。</p>
  <p class="text-dark">text-dark: 深灰色文字。</p>
  <p class="text-light">text-light: 浅灰色文本（白色背景上看不清楚）。</p>
  <p class="text-white">text-white: 白色文本（白色背景上看不清楚）。</p>
</div>
```

执行效果如下：

代表指定意义的文本颜色

text-muted: 柔和的文本。

text-primary: 重要的文本。

text-success: 执行成功的文本。

text-info: 代表一些提示信息的文本。

text-warning: 警告文本。

text-danger: 危险操作文本。

text-secondary: 副标题。

text-dark: 深灰色文字。

2. Bootstrap 提供的背景色的类有：

(1) bg-primary

(2) bg-success

(3) bg-info

- (4) bg-warning
- (5) bg-danger
- (6) bg-secondary
- (7) bg-dark
- (8) bg-light

```
<div class="container">
  <h2>背景颜色</h2>
  <p class="bg-primary text-white">bg-primary: 重要的背景颜色。</p>
  <p class="bg-success text-white">bg-success: 执行成功背景颜色。</p>
  <p class="bg-info text-white">bg-info: 信息提示背景颜色。</p>
  <p class="bg-warning text-white">bg-warning: 警告背景颜色</p>
  <p class="bg-danger text-white">bg-danger: 危险背景颜色。</p>
  <p class="bg-secondary text-white">bg-secondary: 副标题背景颜色。</p>
  <p class="bg-dark text-white">bg-dark: 深灰背景颜色。</p>
  <p class="bg-light text-dark">bg-light 浅灰背景颜色。</p>
</div>
```

执行效果如下：

背景颜色



- 注意背景颜色不会设置文本的颜色，在一些实例中你需要与文本颜色类一起使用。
3. Bootstrap 通过 table 类来定义表格，通过在 table 类后指定一个类来使用不同风格的表格：

```
<table class="table table-striped">
```

```

<thead>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Email</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>john@example.com</td>
  </tr>
  <tr>
    <td>Mary</td>
    <td>Moe</td>
    <td>mary@example.com</td>
  </tr>
  <tr>
    <td>July</td>
    <td>Dooley</td>
    <td>july@example.com</td>
  </tr>
</tbody>
</table>

```

表格风格：

(1) table-striped: 条纹表格。

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

(2) table-bordered: 带边框表格。

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

(3) table-hover：具有鼠标悬停效果，鼠标悬停后变为灰色背景。

(4) table-dark：黑色背景表格。

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

4. rounded 类可以让图片显示圆角效果：

```

```

5. rounded-circle 类可以设置椭圆形图片：

```

```

效果如下：



6. `img-thumbnail` 类用于设置图片缩略图，缩略图图片上有边框：

```

```

7. `float-right` 类用于设置图片右对齐，`float-left` 类用于设置图片左对齐。

```
  

```

8. `alert` 类用于指定信息提示文本的 `css` 样式。

```
<div class="alert alert-success">  
  <strong>成功!</strong> 指定操作成功提示信息。  
</div>
```

其中 `alert-success` 是 `css` 样式类，`css` 样式类有 `alert-success`、`alert-info`、`alert-warning`、`alert-danger`、`alert-primary`、`alert-secondary`、`alert-light` 或 `alert-dark` 类几种：

alert-success 指定操作成功提示信息。

alert-info 请注意这个信息。

alert-warning 设置警告信息。

alert-danger 失败的操作

alert-primary 这是一个重要的操作信息。

alert-secondary 显示一些不重要的信息。

alert-dark 深灰色提示框。

alert-light 浅灰色提示框。

9. `alert-link` 类用于设置提示框的链接：

```
<div class="alert alert-success">  
  你应该认真阅读 <a href="#" class="alert-link">这条信息</a>。  
</div>
```

效果如下：

你应该认真阅读 这条信息。

10. 在提示框中的 `div` 中添加 `.alert-dismissible` 类，然后在关闭按钮的链接上添加 `class="close"` 和 `data-dismiss="alert"` 类来设置提示框的关闭操作。

```
<div class="alert alert-success alert-dismissible">  
  <button type="button" class="close" data-dismiss="alert">&times;</button>
```

```
<strong>成功!</strong> 指定操作成功提示信息。  
</div>
```

效果如下：

成功! 指定操作成功提示信息。



第4章 按钮

1. btn 类用于定义按钮，后接样式类可以指定按钮样式：

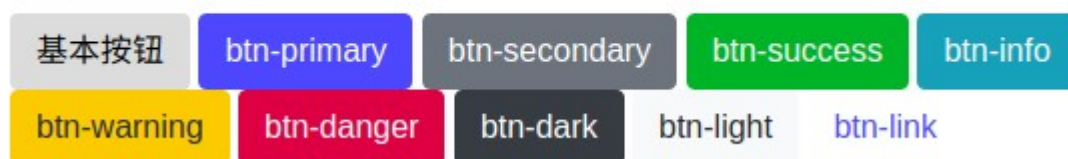
```
<button type="button" class="btn btn-primary">主要按钮</button>
```

显示效果为：

主要按钮

2. 不同样式的按钮主要区别在于背景色，对应关系如下：

按钮样式



3. 按钮类可用于 <a>、<button>或 <input> 元素上：

```
<a href="#" class="btn btn-info" role="button">链接按钮</a>  
<button type="button" class="btn btn-info">按钮</button>  
<input type="button" class="btn btn-info" value="输入框按钮">  
<input type="submit" class="btn btn-info" value="提交按钮">
```

显示效果如下；

链接按钮

按钮

输入框按钮

提交按钮

4. btn-primary 等是无边框按钮，相应的有边框按钮为 btn-outline-primary，多个

outline 关键词，其它的依此类推：

```
<button type="button" class="btn btn-outline-primary">主要按钮</button>
<button type="button" class="btn btn-outline-secondary">次要按钮</button>
<button type="button" class="btn btn-outline-success">成功</button>
<button type="button" class="btn btn-outline-info">信息</button>
<button type="button" class="btn btn-outline-warning">警告</button>
<button type="button" class="btn btn-outline-danger">危险</button>
<button type="button" class="btn btn-outline-dark">黑色</button>
<button type="button" class="btn btn-outline-light text-dark">浅色</button>
```

5. 按钮按大小可以分为大按钮、默认按钮和小按钮 3 种，大按钮和小按钮分别使用 btn-lg 和 btn-sm 来定义。

```
<button type="button" class="btn btn-primary btn-lg">大号按钮</button>
<button type="button" class="btn btn-primary">默认按钮</button>
<button type="button" class="btn btn-primary btn-sm">小号按钮</button>
```

效果如下：



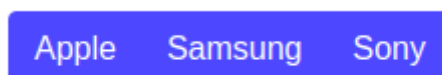
6. active 类可以用于激活按钮，disabled 可以用于禁用按钮。注意 <a> 元素不支持 disabled 属性，你可以通过添加 .disabled 类来禁止链接的点击。

```
<button type="button" class="btn btn-primary active">点击后的按钮</button>
<button type="button" class="btn btn-primary" disabled>禁止点击的按钮</button>
<a href="#" class="btn btn-primary disabled">禁止点击的链接</a>
```

7. 在 <div> 元素上添加 .btn-group 类来创建按钮组。

```
<div class="btn-group">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <button type="button" class="btn btn-primary">Sony</button>
</div>
```

效果如下：



8. btn-group-lg 类用于设置大按钮组, btn-group-sm 类用于设置小按钮组:

```
<div class="btn-group btn-group-lg">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <button type="button" class="btn btn-primary">Sony</button>
</div>
<h3>默认按钮:</h3>
<div class="btn-group">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <button type="button" class="btn btn-primary">Sony</button>
</div>
<h3>小按钮:</h3>
<div class="btn-group btn-group-sm">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <button type="button" class="btn btn-primary">Sony</button>
</div>
</div>
```

显示效果如下:

大按钮:

Apple Samsung Sony

默认按钮:

Apple Samsung Sony

小按钮:

Apple Samsung Sony

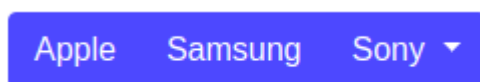
9. btn-group-vertical 类用于创建垂直的按钮组。

```
<div class="btn-group-vertical">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <button type="button" class="btn btn-primary">Sony</button>
</div>
```


10. 在使用 btn-group 修饰的<div>内部内嵌<div>元素，可以在按钮组内设置下拉菜单：

```
<div class="btn-group">
  <button type="button" class="btn btn-primary">Apple</button>
  <button type="button" class="btn btn-primary">Samsung</button>
  <div class="btn-group">
    <button type="button" class="btn btn-primary dropdown-toggle" data-
toggle="dropdown">
      Sony
    </button>
    <div class="dropdown-menu">
      <a class="dropdown-item" href="#">Tablet</a>
      <a class="dropdown-item" href="#">Smartphone</a>
    </div>
  </div>
</div>
```

显示效果如下：



11. Badges 主要用于突出显示新的或未读的消息。badge 类后接 badge-secondary 这种带有指定意义的颜色类添加到 即可：

```
<div class="container">
  <span class="badge badge-primary">badge-primary: 主要</span>
  <span class="badge badge-secondary">badge-secondary: 次要</span>
  <span class="badge badge-success">badge-success: 成功</span>
  <span class="badge badge-danger">badge-danger: 危险</span>
  <span class="badge badge-warning">badge-warning: 警告</span>
  <span class="badge badge-info">badge-info: 信息</span>
  <span class="badge badge-light">badge-light: 浅色</span>
  <span class="badge badge-dark">badge-dark: 深色</span>
</div>
```

效果如下：



12. badge-pill 类用于设置药丸形状徽章:

```
<div class="container">
  <h2>药丸形状徽章</h2>
  <span class="badge badge-pill badge-default">默认</span>
  <span class="badge badge-pill badge-primary">主要</span>
  <span class="badge badge-pill badge-success">成功</span>
  <span class="badge badge-pill badge-info">信息</span>
  <span class="badge badge-pill badge-warning">警告</span>
  <span class="badge badge-pill badge-danger">危险</span>
</div>
```

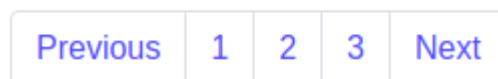
效果如下:



13. 在 元素上添加 pagination 类, 然后在 元素上添加 page-item 类可以实现分页:

```
<div class="container">
  <ul class="pagination">
    <li class="page-item"><a class="page-link" href="#">Previous</a></li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item"><a class="page-link" href="#">Next</a></li>
  </ul>
</div>
```

效果如下:



14. 当前页可以使用 active 类来高亮显示:

```

<ul class="pagination">
  <li class="page-item"><a class="page-link" href="#">Previous</a></li>
  <li class="page-item"><a class="page-link" href="#">1</a></li>
  <li class="page-item active"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item"><a class="page-link" href="#">Next</a></li>
</ul>

```

15.disabled 类用于禁用分页链接：

```

<ul class="pagination">
  <li class="page-item disabled"><a class="page-link" href="#">Previous</a></li>
  <li class="page-item"><a class="page-link" href="#">1</a></li>
  <li class="page-item"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item"><a class="page-link" href="#">Next</a></li>
</ul>

```

16.pagination-lg 类用于设置大字体的分页条目， pagination-sm 类用于设置小字体的分页条目：

```

<div class="container">
  <ul class="pagination pagination-lg">
    <li class="page-item"><a class="page-link" href="#">Previous</a></li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item"><a class="page-link" href="#">Next</a></li>
  </ul>

  <ul class="pagination pagination-sm">
    <li class="page-item"><a class="page-link" href="#">Previous</a></li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item"><a class="page-link" href="#">Next</a></li>
  </ul>
</div>

```

效果如下：



17.breadcrumb 和 breadcrumb-item 类用于设置面包屑导航：

```
<div class="container">
  <ul class="breadcrumb">
    <li class="breadcrumb-item"><a href="#">Photos</a></li>
    <li class="breadcrumb-item"><a href="#">Summer 2017</a></li>
    <li class="breadcrumb-item"><a href="#">Italy</a></li>
    <li class="breadcrumb-item active">Rome</li>
  </ul>
</div>
```

效果如下：

[Photos](#) / [Summer 2017](#) / [Italy](#) / Rome

第 5 章 导航

1. bootstrap 导航分为 navbar 和 navbar 两种，区别在于当屏幕小到一定程度时，navbar 会折叠起来。
2. nav 类、nav-item 类和 nav-link 类用于定义导航：

```
<ul class="nav">
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link disabled" href="#">Disabled</a>
  </li>
</ul>
```

```
</li>
</ul>
```

效果如下：

[Link](#) [Link](#) [Link](#) [Disabled](#)

3. `justify-content-center` 类用于设置导航居中显示，`justify-content-end` 类用于设置导航右对齐。

```
<!-- 导航居中 -->
<ul class="nav justify-content-center">

<!-- 导航右对齐 -->
<ul class="nav justify-content-end">
</div>
```

4. `flex-column` 类用于创建垂直导航：

```
<ul class="nav flex-column">
```

5. `nav-tabs` 类可以将导航转化为选项卡。然后对于选中的选项使用 `active` 类来标记。

```
<div class="container">
  <ul class="nav nav-tabs">
    <li class="nav-item">
      <a class="nav-link active" href="#">Active</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
      <a class="nav-link disabled" href="#">Disabled</a>
    </li>
  </ul>
</div>
```

效果如下：

[Active](#) [Link](#) [Link](#) [Disabled](#)

6. nav-pills 类可以将导航项设置成胶囊形状。

```
<div class="container">
  <ul class="nav nav-pills">
    <li class="nav-item">
      <a class="nav-link active" href="#">Active</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
      <a class="nav-link disabled" href="#">Disabled</a>
    </li>
  </ul>
</div>
```

效果如下：



7. nav-justified 类可以设置导航项齐行等宽显示。

```
<ul class="nav nav-justified">
  <li class="nav-item">
    <a class="nav-link active" href="#">Active</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link disabled" href="#">Disabled</a>
  </li>
</ul>
```

效果如下：



8. 如果要设置选项卡是动态可切换的，可以在每个链接上添加 data-toggle="tab" 属性。然后在每个选项对应的内容上添加 tab-pane 类。

```

<!-- Nav tabs -->
<ul class="nav nav-tabs">
  <li class="nav-item">
    <a class="nav-link active" data-toggle="tab" href="#home">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" data-toggle="tab" href="#menu1">Menu 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" data-toggle="tab" href="#menu2">Menu 2</a>
  </li>
</ul>

<!-- Tab panes -->
<div class="tab-content">
  <div class="tab-pane active container" id="home">...</div>
  <div class="tab-pane container" id="menu1">...</div>
  <div class="tab-pane container" id="menu2">...</div>
</div>

```

导航的 href 和 <div> 的 id 相关联。效果如下：



9. navbar 类来创建一个标准的导航栏，默认为垂直导航：

```

<!-- 垂直导航栏 -->
<nav class="navbar bg-light">

  <!-- Links -->
  <ul class="navbar-nav">
    <li class="nav-item">
      <a class="nav-link" href="#">Link 1</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link 2</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link 3</a>
    </li>
  </ul>

</nav>

```

10. 水平导航可以使用 navbar-expand-xl、navbar-expand-lg、navbar-expand-md 或 navbar-expand-sm 类定义，这四个类分别表示在不同当屏幕像素大于某个值时，导航水平排列：

```
<nav class="navbar navbar-expand-sm bg-light">

<!-- Links -->
<ul class="navbar-nav">
  <li class="nav-item">
    <a class="nav-link" href="#">Link 1</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link 2</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link 3</a>
  </li>
</ul>

</nav>
```

当像素大于等于 576px 时，导航水平排列。

11. 可以使用 bg-dark 等以 bg-开头的类来设置导航颜色：

```
<!-- 灰底黑字 -->
<nav class="navbar navbar-expand-sm bg-light navbar-light">
  <ul class="navbar-nav">
    <li class="nav-item active">
      <a class="nav-link" href="#">Active</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link</a>
    </li>
    <li class="nav-item">
      <a class="nav-link disabled" href="#">Disabled</a>
    </li>
  </ul>
</nav>

<!-- 黑底白字 -->
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">...</nav>
```



```
<!-- 蓝底白字 -->
```

```
<nav class="navbar navbar-expand-sm bg-primary navbar-dark">...</nav>
```

12. 如果导航列有图片，可以使用 `navbar-brand` 类来设置图片自适应导航栏。

```
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
  <a class="navbar-brand" href="#">
    
  </a>
  ...
</nav>
```

13. 要实现小屏幕上折叠导航栏，需要将一个 `<button>` 和 `<div>` 相关联：

```
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
  <!-- Brand -->
  <a class="navbar-brand" href="#">Navbar</a>

  <!-- Toggler/collapsible Button -->
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#collapsibleNavbar">
    <span class="navbar-toggler-icon"></span>
  </button>

  <!-- Navbar links -->
  <div class="collapse navbar-collapse" id="collapsibleNavbar">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
    </ul>
  </div>
</nav>
```

其中 `<button>` 的 `class`、`data-toggle` 和 `data-target` 必须要定义，而且值要等于上面这几个。`<div>` 的 `class` 也要加上 `collapse` 和 `navbar-collapse` 这两个类。

14. 导航栏的表单 `<form>` 元素使用 `class="form-inline"` 类来排版输入框与按钮：

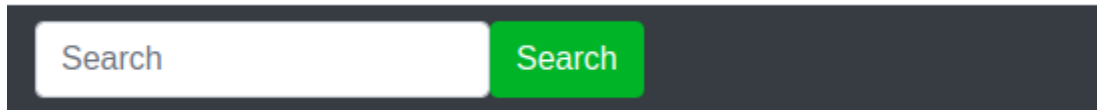
```
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
```

```

<form class="form-inline">
  <input class="form-control" type="text" placeholder="Search">
  <button class="btn btn-success" type="button">Search</button>
</form>
</nav>

```

效果如下：



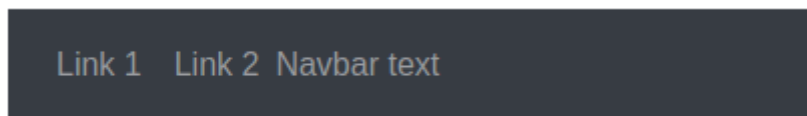
15. `navbar-text` 类用于设置导航栏上非链接文本，非链接文本可以保证水平对齐，颜色与内边距一样。

```

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
  <!-- Links -->
  <ul class="navbar-nav">
    <li class="nav-item">
      <a class="nav-link" href="#">Link 1</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Link 2</a>
    </li>
  </ul>
  <!-- Navbar text-->
  <span class="navbar-text">
    Navbar text
  </span>
</nav>

```

效果如下：



其中的 Navbar text 是没有链接的。

Vue 实战

第 1 章 初识 Vue.js

1.