HTML/CSS/JavsScript

极夜酱

目录

Part I

HTML

Chapter 1 HTML 简介

1.1 前后端

1.1.1 前端工程师（Front-End Engineer）

前端工程师互联网时代软件产品研发中不可缺少的一种专业研发角色。前端是一个相对比较新的行业，大约从 2005 年开始，正式的前端工程师角色被行业所认可。到了 2010 年，互联网开始全面进入移动时代，前端工程师的地位也越来越重要。

现在一些后端开发工作也可以由前端工程师来完成。最初所有的开发工作都是由后端工程师完成的，但是随着业务越来越繁杂、工作量过大，后端工程师们不堪重负，于是将可视化和部分交互功能的开发剥离出来，形成了前端开发。

前端工程师主要负责的工作就是使用 HTML、CSS、JavaScript 等专业技能和工具将产品 UI 设计稿实现成网站产品。涵盖用户 PC 端和移动端网页，处理视觉和交互问题。

广义上讲，所有用户终端产品，只要是与视觉和交互有关的部分都是前端工程师的专业领域。产品从前期开发到后期的维护、更新、升级都需要前端工程师来完成。

1.1.2 后端工程师（Back-End Engineer）

后端工程师主要负责服务器的数据逻辑和业务逻辑等，主要研究怎么把数据更好地传输给前端工程师。

如果一个人除了完成前端开发和后端开发工作以外，从产品设计到项目开发，再到后期运维都是同一个人，甚至可能还要负责 UI、配动画、写文档等，那么就被称为全栈工程师（Full Stack Engineer）。

1.1.3 前端应用领域

前端技术可以被应用在一系列领域中：

- 网页网站
- APP、微信小程序
- 移动端小游戏
- 炫酷的特效
- 大数据可视化
- VR 虚拟现实

前端工程师需要具备大量必要的技能，其中前端三大基础语言为 HTML、CSS、JavaScript，除此之外还需要学习 jQuery、网络、es6、webpack4.0、小程序、VUE、React、Node.js、Mongo DB 数据库等内容。

1.2 结构层/表现层/行为层

1.2.1 结构层/表现层/行为层

- 结构层 HTML（HyperText Markup Language）：一个超文本标记语言，负责描绘出网页内容的架构。
- 表示层 CSS（Cascading Style Sheets）：层叠样式表，负责如何显示结构层的有关内容。
- 行为层 JavaScript：目前在 Web 上使用的最主要的客户端脚本语言，是 Web 脚本语言的一个标准，可以对结构层和表现层的内容随意进行更改。

1.2.2 代码注释

在 HTML、CSS、JavaScript 中代码的注释是不一样的。

HTML 注释

```
1 <!-- 注释内容 -->
```

CSS 注释

```
1 /* 注释内容 */
```

JavaScript 注释

```
1 // 注释内容
2 /* 注释内容 */
```

1.3 Hello World!

1.3.1 Hello World!

问候一下世界，制作人生中的第一个 HTML 网页吧。

Hello World!

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World!</title>
</head>
<body>
    Hello World!
</body>
</html>
```

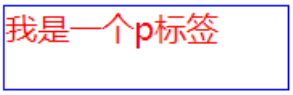
1.3.2 HTML 和 CSS 的关系

先来看一下单纯的 HTML 标签长什么样：

我是一个p标签

再来看一下经过 CSS 修饰过后的 HTML 标签：

```
p {
    color: red;
    border: 1px solid red;
    width: 140px;
    height: 40px;
}
```



CSS 是用来修饰 HTML 样式的。虽然 HTML 本身是有一些默认样式，但如果想改变 HTML 标签的样式，就需要借助 CSS。

HTML + CSS 构成了网页的基本页面结构和样式。

添加样式

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>添加样式</title>
    <style type="text/css">
        p {
            font-size: 12px;
            color: #930;
            text-align: center;
        }
    </style>
</head>
<body>
    <p>Hello World!</p>
</body>
</html>
```

1.4 标签

1.4.1 标签

HTML 中标签的语法有以下几点：

1. 标签（tag）是由英文尖括号【<】和【>】括起来的，如 <html> 就是一个标签。

2. HTML 中的标签一般都是成对出现的，分开始标签和结束标签。结束标签比开始标签多了一个【/】。

```
<p></p>
<div></div>
<span></span>
```

3. 标签与标签之间是可以嵌套的，但先后顺序必须保持一致。如， 里嵌套 ，那么 必须放在 的前面。

```
<div><p>Hello World!</p></div>
```

4. HTML 标签不区分大小写，如 <h1> 和 <H1> 是一样的，但建议小写，因为大部分程序员都已小写为准。

1.5 HTML 文档结构

1.5.1 HTML 文档结构

HTML 文档结构

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>HTML文档结构</title>
</head>
<body>

</body>
</html>
```

- <!DOCTYPE html>：文档类型声明，表示该文件为 HTML 文件。声明必须是 HTML 文档的第一行，位于 <html> 之前。
- <html></html>：<html> 用来标识 HTML 文档的开始，</html> 位于 HTML 文档的最后面，用来标识 HTML 文档的结束。这两个标签对成对存在，中间的部分是文档的头部和主题。
- <head></head>：标签包含有关 HTML 文档的信息，可以包含一些辅助性标签。如 <title></title>、<link />、<meta />、<style></style>、<script></script> 等，浏览器除了会在标题栏显示 <title> 元素的内容外，不会向用户显示 <head> 元素内的其他任何内容。
- <body></body>：HTML 文档的主体部分，在此标签中可以包含 <p>、<h1>、
 等众多标签。<body> 出现在 </head> 之后，且必须在闭标签 </html> 之前闭合。

1.6 小哥，做头吗？——head 标签

1.6.1 head 标签

文档的头部描述了文档的各种属性和信息，包括文档的标题等，绝大多数文档头部包含的数据都不会真正作为内容显示给读者。

<head></head> 为双标签，表示头部标签，通常用来嵌套 meta、title、style 等标签。

- <title>：在 <title> 和 </title> 之间的文字内容是网页的标题信息，它会出现在浏览器的标题栏中。网页的 <title> 用于告诉用户和搜索引擎这个网页的主要内容是什么，搜索引擎可以通过网页标题，迅速的判断出网页的主题。每个网页的内容都是不同的，每个网页都应该有一个独一无二的 title。
- <meta charset="UTF-8">：设置当前文件字符编码。
- <style>：设置当前文件样式。

1.7 你就是馋人家的身子！——body 标签

1.7.1 body 标签

在网页上要展示出来的页面内容一定要放在 <body> 中。

body 标签

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>body标签</title>
</head>
<body>
    <!-- 标题标签 -->
    <h1>HTML简介</h1>
    <!-- 段落标签 -->
    <p>HTML的全称为超文本标记语言，是一种标记语言。</p>
    <!-- 段落标签 -->
    <p>它包括一系列标签．可以将网络上的文档格式统一。</p>
</body>
</html>
```

Chapter 2 语义化标签

2.1 开始我们的第一段对话吧

2.1.1 语义化（Semantic）

学习 HTML 标签需要注意标签的用途和标签在浏览器中的默认样式。

语义化，通俗的讲就是明白每个标签的用途，即在什么情况下使用此标签合理。比如，网页上文章的标题可以用标题标签、各个栏目的名称也可以使用标题标签、文章内容的段落就得放在段落标签中。

语义化可以带来一些好处：

- 更容易被搜索引擎收录
- 更容易让屏幕阅读器读出网页内容

2.1.2 p 标签

如果想在网页上显示文章，就需要使用 <p> 了，把文章的段落放到 <p> 中。

```
1 <p>段落文本</p>
```

注意一段文字一个 <p>，如在一篇文章中有三段文字，就要分别放到三个 <p> 中。

p 标签

```
1 <!DOCTYPE HTML>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
```

```
    <title>p标签</title>
</head>
<body>
    <p>所有主流浏览器都支持p标签。</p>
    <p>p标签定义段落。</p>
    <p>p元素会自动在其前后创建一些空白。</p>
</body>
</html>
```

<p> 的默认样式，在段前段后都会有空白，如果不喜欢这个空白，可以用 CSS 样式来删除或改变它。

2.1.3 span 标签

 是没有语义的，它的作用就是为了设置单独的样式用的。如果现在想把一段中某些字设置成蓝色，这种情况下就可以用到 了。

span 标签

```
<!DOCTYPE HTML>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>span标签</title>
    <style type="text/css">
        span {
            color: blue;
        }
    </style>
</head>
<body>
    <p>
        <span>莫里亚蒂</span>有份包裹指明要交给<span>夏洛克</span>
    </p>
</body>
</html>
```

2.2 做个标题党——hx 标签

2.2.1 hx 标签

文章的段落用 <p>，那么文章的标题可以使用标题标签。标题标签一共有 6 个，<h1>、<h2>、<h3>、<h4>、<h5>、<h6> 分别为一级标题、二级标题、三级标题、四级标题、五级标题、六级标题，并且依据重要性递减，<h1> 是最高的等级。

```
<h1>标题文本</h1>
<h2>标题文本</h2>
<h3>标题文本</h3>
<h4>标题文本</h4>
<h5>标题文本</h5>
<h6>标题文本</h6>
```

网页上的各个栏目标题也可使用标题标签。因为 <h1> 在网页中比较重要，一般 <h1> 被用在网站名称上。

标题标签的样式都会加粗，<h1> 字号最大，<h2> 字号相对 <h1> 要小，以此类推 <h6> 的字号最小。

2.3 div 标签

2.3.1 div 标签

网页制作过程过中，可以把一些独立的逻辑部分划分出来，放在一个 中， 的作用就相当于一个容器。

逻辑部分是页面上相互关联的一组元素，如网页中的独立的栏目版块，就是一个典型的逻辑部分。如下图所示，图中用红色边框标出的部分就是一个逻辑部分，就可以使用 作为容器。

图 2.1: div 容器

div 标签

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>div标签</title>
</head>
<body>
    <div>
    <h2>热门课程排行榜</h2>
        <ol>
```

```
            <li>前端开发面试心法</li>
            <li>零基础学习HTML</li>
            <li>JavaScript全攻略</li>
        </ol>
    </div>

    <div>
        <h2>最新课程排行</h2>
        <ol>
            <li>版本管理工具介绍—Git篇</li>
            <li>Canvas绘图详解</li>
            <li>QQ5.0侧滑菜单</li>
        </ol>
    </div>
</body>
</html>
```

2.4 写代码都不换行吗？——br 标签

2.4.1 br 标签

在 HTML 代码中输入回车、空格都是没有作用的，在 HTML 中是忽略回车和空格的，输入再多的回车和空格也是现实不出来的。如果需要在 HTML 文本中输入回车换行，就必须使用
。在需要加回车换行的地方加入
，
 的作用相当于 Word 文档中的回车。

 是一个单标签，没有 HTML 内容的标签就是单标签。单标签只需要写一个开始标签，这样的标签有
、<hr/> 和 <img/>。

br 标签

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>br标签</title>
</head>
<body>
    <h2>《望庐山瀑布》</h2>
    <p>唐·李白</p>
    <p>
        日照香炉生紫烟，<br/>
        遥看瀑布挂前川。<br/>
        飞流直下三千尺，<br/>
        疑是银河落九天。
    </p>
</body>
</html>
```

2.5 再加点空格呢？——特殊字符

2.5.1 特殊字符

在 HTML 代码中输入空格、回车都是没有作用的，输出多个空格只会显示 1 个空格。如果需要输入空格，必须使用特殊字符 。

特殊字符

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>空格</title>
</head>
<body>
    <h2>新闻纵览</h2>
    来源：XXX     作者：XXX
</body>
</html>
```

2.6 再来个水平分割线——hr 标签

2.6.1 hr 标签

在信息展示时，有时会需要加一些用于分隔的横线，这样会使文章看起来整齐些。

<hr/> 和
 一样也是一个单标签，所以只有一个开始标签，没有结束标签。

<hr/> 在浏览器中的默认样式线条比较粗，颜色为灰色。有些人觉得这种样式不美观，这些在样式可以通过 CSS 进行修改。

hr 标签

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>hr标签</title>
</head>
<body>
    <p>深夜，一道黑影潜入贝克街221B。</p>
    <hr/>
    <p>坐在暗处的夏洛克已等候多时，正等着贝尔纳多自投罗网。</p>
</body>
</html>
```

Chapter 3 列表、图片、链接

3.1 有序列表

3.1.1 有序列表（Ordered List）

如果想在网页中展示有前后顺序的信息列表，如热度排行榜等，这类信息展示可以使用 - 来实现有序列表。

```
<ol>
    <li>信息</li>
    <li>信息</li>
</ol>
```

图 3.1: 有序列表

有序列表

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>有序列表</title>
</head>
<body>
    <h3>2020年7月编程语言排行榜</h3>
    <ol>
        <li>C</li>
        <li>Java</li>
        <li>Python</li>
        <li>C++</li>
        <li>C#</li>
    </ol>
</body>
</html>
```

3.1.2 序号类型

- 在网页中显示的默认样式一般为每项 <li> 前都自带一个序号，序号默认从 1 开始。通过修改 <ol> 的 type 属性，也能对序号进行修改。

 的 type 属性有 5 个值：

1. 默认为数字序号：type="1"
2. 小写字母序号：type="a"
3. 大写字母序号：type="A"
4. 小写罗马数字序号：type="i"
5. 大写罗马数字序号：type="I"

 中设置 start 属性，可以指定开始序号。

 中设置reversed="reversed"属性，可以将列表序号降序排序。

3.2 无序列表

3.2.1 无序列表（Unordered List）

网页上也有很多信息是无需按照先后次序排列的，如新闻列表、图片列表等，这类信息展示可以使用 - 来实现无序列表。

```
<ul>
    <li>信息</li>
    <li>信息</li>
</ul>
```

图 3.2: 无序列表

- 在网页中显示的默认样式一般为每项 <li> 前都自带一个圆点。通过修改 <ul> 的 type 属性，也能对前面的符号进行修改。

 的 type 属性有 2 个值：

1. 默认为实心小圆点：type="disc"
2. 实心正方形：type="square"

无序列表

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

```
<head>
    <meta charset="UTF-8">
    <title>无序列表</title>
</head>
<body>
    <h3>前端三剑客</h3>
    <ul>
        <li>HTML</li>
        <li>CSS</li>
        <li>JS</li>
    </ol>
</body>
</html>
```

淘宝网的导航栏就是利用 - 的父子结构进行实现的。

图 3.3: 淘宝导航栏

淘宝导航栏

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>淘宝</title>
    <style type="text/css">
        ul {
            list-style: none;    /* 无序列表前面不带点 */
```

```
        }

        li {
            float: left;
            margin: 20px;
            padding: 5px;
            color: #000;
        }

        li:hover {
            background-color: #f40;      /* 淘宝红 */
            color: #fff;
            border-radius: 15px;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <ul>
        <li>天猫</li>
        <li>聚划算</li>
        <li>天猫超市</li>
    </ul>
</body>
</html>
```

3.3 不发一张自拍吗？——img 标签

3.3.1 img 标签

在网页的制作中为使网页炫丽美观，肯定是缺少不了图片，<img> 用于插入图片。

```
<img src="图片地址" alt="下载失败时的替换文本" title="提示文本">
```

- src 属性用于标识图像的资源地址，资源地址可以来源于网上的 URL，也可以来源于本地。
- alt 属性指定图像的描述性文本，当图像下载不成功时，可看到该属性指定的文本。
- title 属性提供当鼠标停留在图片时显示的文本。

img 标签

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>img标签</title>
</head>
<body>
    <img src="https://ss0.bdstatic.com/70cFuHSh_Q1YnxGkpoWK1HF6hhy
        /it/u=3014023147,616635741&fm=26&gp=0.jpg" alt="皮卡丘.jpg
        " title="皮卡丘">
</body>
</html>
```

3.4 百度一下，你就知道——a 标签

3.4.1 a 标签

<a> 可实现超链接，它在网页制作中可以说是无处不在，只要有链接的地方，就会有这个标签。

```
<a href="目标网址" title="鼠标停留显示的文本">链接显示的文本</a>
```

<a> 中 title 属性提供的功能是当鼠标停留在链接文字时显示这个属性的文本内容，这个属性在实际网页开发中作用很大，主要方便搜索引擎了解链接地址的内容（语义化更友好）。

a 标签

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>a标签</title>
</head>
<body>
    <a href="http://www.baidu.com" title="点击跳转百度">
        百度一下，你就知道
    </a>
</body>
</html>
```

只要为文本加入 <a> 后，文字的颜色就会自动变为蓝色，被点击过的文本颜色会变为紫色，通过 CSS 样式可以对文字的颜色进行修改。

<a> 中还有一个 target 属性，默认值为 _self，表示在同页面中打开被链接的文档。如果需要在新窗口打开被链接的文档，需要将 target 属性的值设置为 _blank。

3.4.2 锚点（Anchor）

<a> 最初的功能是记录锚点（记录位置），通过 <a> 回到那个位置去。

锚点

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>锚点</title>
</head>
<body>
    <div id="div1" style="width: 100px; height: 100px;
        background-color: red;"></div>
    <div id="div2" style="width: 100px; height: 100px;
        background-color: blue;"></div>
    <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
        <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
        <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
        <br><br><br><br><br><br><br>
    <a href="#div1">跳转到div1</a>
    <a href="#div2">跳转到div2</a>
</body>
</html>
```

<a> 作为锚点的应用场景也很多，比如目录跳转。

图 3.4: 目录跳转

3.4.3 协议限定符

<a> 还能作为协议限定符，在点击链接时会强制执行 JavaScript 代码。

协议限定符

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>访问限定符</title>
</head>
<body>
    <a href="JavaScript: while(1) { alert('嘿嘿'); }">点我</a>
</body>
</html>
```

Chapter 4 表格、表单

4.1 table 标签

4.1.1 table 标签

有时候需要在网页中添加一些表格数据，如企业员工通讯录等。

图 4.1: 企业员工通讯录

创建表格主要包含 5 个元素：

1. <caption>：定义表格的标题。
2. <table>：整个表格以 <table> 开始、</table> 结束，<table> 在没有添加 border 属性之前，在浏览器中显示是没有表格线的。
3. <tr>：表格的一行，有几对 <tr> 表格就有几行，<tr> 里只能放 <th> 或者 <td>。
4. <th>：表格头部的一个单元格，会加粗居中显示。
5. <td>：表格的一个单元格，有几对 <td> 一行中就有几列。

企业员工通讯录

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>企业员工通讯录</title>
</head>
<body>
    <h3>企业员工通讯录</h3>
    <hr/>

    <!-- 表格标签 border属性代表给表格加上边框 -->
    <table border="1">
        <!-- 表格标题 -->
        <caption>企业员工通讯录</caption>
        <!-- tr代表一行 -->
        <tr>
            <!-- th代表表格头部的一个单元格 -->
            <th>姓名</th>
            <th>电话</th>
            <th>电子邮件</th>
            <th>职务</th>
        </tr>
        <tr>
            <!-- td代表一个单元格 -->
            <td>张三</td>
            <td>18278900988</td>
            <td>zhangsan@163.com</td>
            <td>研发工程师</td>
        </tr>
        <tr>
            <td>王二</td>
            <td>16589012689</td>
            <td>wanger@163.com</td>
            <td>研发经理</td>
        </tr>
        <tr>
```

```
37              <td>李四</td>
38              <td>17230019065</td>
39              <td>lisi@163.com</td>
40              <td>研发工程师</td>
41          </tr>
42      </table>
43  </body>
44  </html>
```

4.2 与用户交互——form 标签

4.2.1 form 标签

使用 HTML 表单（form）可以实现网站与用户的交互，表单可以把浏览者输入的数据传送到服务器端，这样服务器端程序就可以处理表单传过来的数据。

```
1 <form method="传送方式" action="服务器文件">表单内容</form>
```

<form> 是成对出现的，以 <form> 开始、</form> 结束。method 属性表示数据传送的方式，包括 get 和 post 两种方式，get 和 post 的区别属于后端程序员需要考虑的问题，完全取决于后端人员要求什么方式进行传输。action 属性表示浏览者输入的数据被传送到的地方，比如一个 PHP 页面。

所有的表单控件（文本框、文本域、按钮、单选框、复选框等）都必须放在 <form> 之间，否则用户输入的信息无法提交到服务器上。

4.3 先来填用户名和密码——文本、密码输入框

4.3.1 文本、密码输入框

当用户需要在表单中输入字母、数字等内容时，就需要使用到文本输入框，文本框也可转化为密码输入框。

```
<form>
    <input type="text/password" name="名称" value="文本" />
</form>
<form>
    <input type="text/password" name="名称" value="文本" />
</form>
```

- type="text"：输入框为文本输入框
- type="password"：输入框为密码输入框
- name：为文本框命名，以备后台程序使用
- value：为文本输入框设置默认值，一般起提示作用

4.3.2 给点提示呗——placeholder 属性

有时候需要提示用户输入框需要输入的内容，这时候就需要使用 <input> 中的占位符 placeholder 属性。

图 4.2: placeholder 提示文本

placeholder 属性的值可以根据情况合理填写，当输入框输入内容时，占位符内容消失，当输入框无内容时，占位符内容显示。注意，占位符内容不是输入框真正的内容。

4.3.3 重置按钮

当用户需要重置表单信息到初始状态时，比如输入“账号”或“密码”有误，就可以使用重置按钮使输入框恢复到初始状态。通过设置 <input> 中 type 属性为 reset 即可实现重置按钮。

4.3.4 提交按钮

当用户需要提交表单信息到服务器时，需要用到提交按钮。通过设置 <input> 中 type 属性为 submit 即可实现提交按钮。表单信息会被发送给后端，在数据库对比账户和密码信息。

登录功能

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录</title>
</head>
<body>
    <form action="get/post">
        账号: <input type="text" placeholder="请输入账号"
            name="username">
        <br/>
        密码: <input type="password" placeholder="请输入密码"
            name="password">
        <br/>
        <input type="submit">
        <input type="reset">
    </form>
</body>
</html>
```

4.4 数字、网址、邮箱输入框

4.4.1 数字输入框

将 <input> 的 type 属性设置为 number，则表示该输入框的类型为数字。数字框只能输入数字，输入其它字符无效。数字框最右侧会有一个加减符号，可以调整输入数字的大小，不同浏览器表现不一致。

4.4.2 网址输入框

将 <input> 的 type 属性设置为 url，表示输入类型为网址。网址输入框必须以 http://或 https://开头，且后面必须有内容，否则提交时会报错误提示。

4.4.3 邮箱输入框

将 <input> 的 type 属性设置为 email，则表示该输入框的类型为邮箱。邮箱输入框的值必须包含 '@'，并且之后必须有内容，否则会报错误提示。

个人信息

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>个人信息</title>
</head>
<body>
    <form action="get/post">
        姓名: <input type="text" name="name"><br/>
        年龄: <input type="number" name="age"><br/>
        主页: <input type="url" name="webpage"><br/>
        邮箱: <input type="email" name="email"><br/>
        <input type="submit"><input type="reset">
    </form>
</body>
</html>
```

4.5 文本域

4.5.1 文本域

当用户需要在表单中输入大段文字时，就需要用到文本输入域。<textarea> 中也可以设置 placeholder 属性来显示提示信息。

```
<textarea rows="行数" cols="列数">文本</textarea>
```

- rows：多行输入域的行数，可以用 CSS 样式的 height 代替
- cols：多行输入域的列数，可以用 CSS 样式的 width 代替

文本域

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>文本域</title>
</head>
<body>
    <form action="get/post">
        <textarea cols="30" rows="10"
            placeholder="请输入..."></textarea>
    </form>
</body>
</html>
```

4.6 label 标签

4.6.1 label 标签

<label> 不会向用户呈现任何特殊特效，它的作用是为鼠标用户改进了可用性。如果在 <label> 内点击文本，就会触发此控件。也就是说，当用户单击选中该 <label> 时，浏览器就会自动将焦点转到和标签相关的表单控件上。

```
<label for="控件id名称">
```

注意，<label> 的 for 属性的值必须要与相关空间的 id 属性值相同。

label 标签

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>label标签</title>
</head>
<body>
    <form action="get/post">
        <label for="username">用户名：</label>
        <input type="text" id="username">
    </form>
</body>
</html>
```

4.7 单选框、复选框

4.7.1 单选框、复选框

在使用表单设计调查表时，为了减少用户的操作，使用选择框是一个好主意。HTML 中有两种选择框，即单选框和复选框，两者的区别是单选框中的选项用户只能选择一项，而复选框中用户可以任意选择多项，甚至全选。

```
<input type="radio/checkbox" name="名称" value="值"
    checked="checked">
```

- type="radio"：单选框
- type="checkbox"：复选框
- name：为控件命名，具有相同名称的选择框属于同一组
- checked：设置默认选中的值

在网页开发中，往往需要优化用户体验。一个好的产品有 3 个特点：解决刚需的问题、用户体验、用户黏性（产品定位）。用户体验就是养成用户的懒习惯，减少用户操作，能不让用户操作就不让用户操作，节省用户时间，用户不动手就会觉得方便。例如一个选择性别的单选框，概率上看，假设有 100 个人，50 男 50 女，使用默认选项，可以节省一半人的操作。

单选框/复选框

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>单选框/复选框</title>
</head>
<body>
    <form action="get/post">
        1. 您是否喜欢运动?
```

```
10          <input type="radio" name="sport" checked="checked">是
11          <input type="radio" name="sport">否
12          2. 您喜欢的运动是?
13          <input type="checkbox" name="sport_item">跑步
14          <input type="checkbox" name="sport_item">篮球
15          <input type="checkbox" name="sport_item">羽毛球
16      </form>
17  </body>
18  </html>
```

4.8 下拉列表

4.8.1 下拉列表

下拉列表在网页中也常会用到，它可以有效的节省网页空间。下拉列表既可以单选也可以多选。

```
<select>
    <option value="值">选项</option>
    <option value="值">选项</option>
</select>
```

- value：向服务器提交的值，而双标签中间的内容才是显示的值
- selected：设置selected="selected"表示该选项被默认选中

下拉列表

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>下拉列表</title>
</head>
<body>
    <form action="get/post">
        <select>
            <option value="male" selected="selected">男</option>
            <option value="female">女</option>
        </select>
    </form>
</body>
</html>
```

Part II

CSS

Chapter 5 CSS 简介

5.1 给 HTML 打扮——CSS 样式

5.1.1 层叠样式表（CSS, Cascading Style Sheets）

CSS 用于定义 HTML 内容在浏览器内的显示样式，如文字大小、颜色、字体加粗等。使用 CSS 的好处是通过定义某个样式，可以让网页不同位置的文本有统一的字体、字号或颜色等。CSS 由选择器和声明组成，而声明又由属性和值组成。

选择器用于指明网页中要应用样式规则的元素。声明的内容写在大括号内，属性和值之前用【:】分隔。当有多条声明时，中间可以用【;】分隔。为了使样式更加容易阅读，可以将每条声明单独成行。

修改字体大小和颜色

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>修改字体大小和颜色</title>
    <style type="text/css">
        p {
            font-size: 20px;
            color: red;
        }
    </style>
</head>
<body>
    <p>修改字体大小和颜色</p>
</body>
</html>
```

5.2 既然那么好，那就引入 CSS 吧——内联式 CSS

5.2.1 内联式 CSS

内联式 CSS 样式，也称行间样式，就是把 CSS 代码直接写在现有的 HTML 标签中。注意 CSS 样式必须写在元素的开始标签里，不能在结束标签里。

```
<开始标签 style="属性: 值;">文本</结束标签>
```

CSS 样式必须写在 style 属性的双引号中，如果有多条 CSS 样式代码可以设置可以写在一起，中间用【;】隔开。

内联式 CSS

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>内联式CSS</title>
</head>
<body>
    <p style="color: red; font-size: 20px;">内联式CSS</p>
</body>
</html>
```

5.3 换个地方吧，行内太挤了——嵌入式 CSS

5.3.1 嵌入式 CSS

嵌入式 CSS 样式，也称页面级 CSS 样式，就是把 CSS 样式代码写在 <style> 之间，嵌入式 CSS 样式一般放在 <head> 内。

嵌入式 CSS

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>嵌入式CSS</title>
    <style type="text/css">
        span {
            color: red;
        }
    </style>
</head>
<body>
    <p><span>乔恩</span>找到<span>艾洛莉</span></p>
</body>
</html>
```

5.4 还是把 HTML 和 CSS 分开吧——外部式 CSS

5.4.1 外部式 CSS

外部式 CSS 样式，也称外联式 CSS 样式，就是把 CSS 样式代码写一个单独的外部文件中，这个 CSS 样式文件以.css 为扩展名。在 <head> 内使用 <link> 将 CSS 外部样式文件链接到 HTML 文件内。

```
<link href="CSS样式文件名" rel="stylesheet" type="text/css" />
```

CSS 样式文件名以有意义的英文命名，<link> 中 rel="stylesheet" type="text/css"是固定写法，不需要修改。

外部式 CSS

external_css.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>外部式CSS</title>
    <link type="text/css" rel="stylesheet"
        href="external_css.css">
</head>
<body>
    <div></div>
</body>
</html>
```

external_css.css

```
div {
    width: 100px;
    height: 100px;
    background-color: blue;
}
```

5.5 总有个先来后到吧——三种链接方式的优先级

5.5.1 CSS 引入方式优先级

如果有一种情况：对于同一个元素同时使用了三种方法设置 CSS 样式，那么哪种方式真正有效呢?

三种 CSS 引入方式是有优先级的：内联式 > 嵌入式 > 外部式。但是嵌入式 > 外部式有一个前提，那就是嵌入式 CSS 样式的位置一定在外部式的后面。在实际开发中也会将 <link> 写在 <style> 的前面。

总的来说，优先级遵循“就近原则”，离被设置元素越近优先级别越高。

但是以上总结的优先级有一个前提，那就是内联式、嵌入式、外部式样式表中 CSS 样式是在相同权值的情况下。那权值是什么呢?

Chapter 6 选择器

6.1 选一个标签——标签选择器

6.1.1 标签选择器

标签选择器其实就是 HTML 代码中的标签。

```
tag_selector {
    attribute: value;
}
```

标签选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>标签选择器</title>
    <style type="text/css">
        h2 {
            color: green;
            font-size: 30px;
        }
    </style>
</head>
<body>
    <h2>CSS特点</h2>
    <p>CSS为HTML标记语言提供了一种样式描述。</p>
</body>
</html>
```

6.2 再选一个类——类选择器

6.2.1 类选择器

类选择器在 CSS 样式中是最常用的。类选择器使用【.】开头，后加类选择器的名称，CSS 样式代码会被作用到属于该类的 HTML 标签中。在标签中使用 class 属性为标签设置一个类。

类选择器与标签是多对多的关系，即类选择器名称可以多个标签共用，一个元素可以用多个 class，一个 class 值可以对应多个元素。多个 class 值之间使用空格分隔。

类选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>类选择器</title>
    <style type="text/css">
        .title {
            color: green;
        }
    </style>
</head>
<body>
    <h3 class="title">丰富的样式定义</h3>
    <p>CSS提供了丰富的文档样式外观。</p>
    <h3 class="title">易于使用和修改</h3>
    <p>CSS样式表可以将所有的样式声明统一存放，进行统一管理。</p>
</body>
</html>
```

6.3 取个唯一表示——ID 选择器

6.3.1 ID 选择器

使用 ID 选择器，必须给标签添加上 id 属性，即为标签设置 id 属性。ID 选择器名称的前面使用【#】。ID 选择器与标签是一对一的关系，即一个元素只能有一个 id 值，一个 id 值只能对应一个元素。id 是全局唯一的，就像身份证号码一样。

ID 选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ID选择器</title>
    <style type="text/css">
        div {
            width: 100px;
            height: 100px;
        }

        #square1 {
            background-color: red;
        }
        #square2 {
            background-color: blue;
        }
    </style>
</head>
<body>
    <div id="square1"></div>
    <div id="square2"></div>
</body>
</html>
```

6.4 捡了个儿子——子选择器

6.4.1 子选择器

子选择器【>】，用于选择指定标签元素的第一代子元素。

子选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>子选择器</title>
    <style type="text/css">
        .food > li {
            border: 1px solid red;
        }
    </style>
</head>
<body>
    <h2>食物</h2>
    <ul class="food">
        <li>水果
            <ul>
                <li>苹果</li>
            </ul>
        </li>
        <li>蔬菜
            <ul>
                <li>白菜</li>
            </ul>
        </li>
    </ul>
</body>
</html>
```

6.5 这么快就当爷爷了——后代选择器

6.5.1 后代选择器

后代选择器，也称包含选择器，用于选择指定标签元素的后辈元素。

```
ancestor_selector descendant_selector {
    attribute: value;
}
```

后代选择器与子选择器的区别在于，子选择器仅是指它的直接后代，而后代选择器是作用于所有子后代元素。

后代选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>后代选择器</title>
    <style type="text/css">
        .food li {
            border: 1px solid red;
        }
    </style>
</head>
<body>
    <h2>食物</h2>
    <ul class="food">
        <li>
            水果
            <ul>
                <li>苹果</li>
                <li>香蕉</li>
                <li>橘子</li>
            </ul>
```

```
        </li>
        <li>
            蔬菜
            <ul>
                <li>白菜</li>
                <li>油菜</li>
                <li>卷心菜</li>
            </ul>
        </li>
    </ul>
</body>
</html>
```

6.6 我全都要——通配符选择器

6.6.1 通配符选择器

通配符选择器【*】，也称通用选择器，是功能最强大的选择器，用于匹配 HTML 中所有的标签元素，包括 <html>、<body> 等。

通配符选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>通配符选择器</title>
    <style type="text/css">
        * {
            background-color: yellow;
        }
    </style>
</head>
<body>

</body>
</html>
```

6.7 给选择器分个组——分组选择器

6.7.1 分组选择器

分组选择器【,】，用于为 HTML 中多个标签元素设置同一个样式。

分组选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>分组选择器</title>
    <style type="text/css">
        h1, h2, h3 {
            color: red;
        }
    </style>
</head>
<body>
    <h1>HTML</h1>
    <h2>CSS</h2>
    <h3>JavaScript</h3>
</body>
</html>
```

6.8 伪装者——伪类选择器

6.8.1 伪类选择器

伪类选择器【:】允许给 HTML 标签的某种状态设置样式，例如给一个标签元素的鼠标滑过的状态设置字体颜色。

伪类选择器	功能
:link	未访问
:visited	已访问
:hover	鼠标悬停
:active	鼠标按下
:enabled	可用的时候触发
:disabled	不可用的时候触发

表 6.1: 常用伪类选择器

到目前为止，可以兼容所有浏览器的伪类选择器就是在 <a> 上使用:hover。

伪类选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>伪类选择器</title>
    <style type="text/css">
        a:hover {
            color: orange;
        }
    </style>
</head>
<body>
    <a href="http://www.baidu.com">百度一下，你就知道</a>
</body>
</html>
```

6.9 为所欲为——选择器最高层级!important

6.9.1 选择器优先级

每个选择器都是有优先级的，如果一个元素使用了多个选择器，则会按照选择器的优先级来给定样式。

选择器的优先级依次是：内联样式 > ID 选择器 > 类选择器 > 标签选择器 > 通配符选择器。

6.9.2 选择器权重

浏览器是根据权值来判断使用哪种 CSS 样式的，权值高的优先级更高。

选择器	权值
!important	∞
行间样式	1000
ID	100
class	10
标签	1
通配符	0

表 6.2: 选择器权重

6.9.3 !important

有些特殊情况需要为某些样式设置具有最高权值，这时候可以使用!important，注意!important 要写在分号的前面。

!important

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
```

```
5       <title>!important</title>
6       <style type="text/css">
7           h1 {
8               color: red !important;
9           }
10
11          h1 {
12              color: blue;
13          }
14      </style>
15  </head>
16  <body>
17      <h1>为所欲为</h1>
18  </body>
19  </html>
```

Chapter 7 字体、文本样式

7.1 字体样式

7.1.1 字体样式

使用 CSS 样式可以为网页中的文字设置字体。注意不要设置不常用的字体，因为如果用户本地电脑上没有安装该字体，就会显示浏览器默认的字体。

浏览器默认的字号为 16px，使用 font-size 可以修改字号大小。

为文字设置粗体是有单独的 CSS 样式来实现的，再也不用为了实现粗体样式而使用 <h1>-<h6> 或 了。

font-weight 的默认值为 normal，通过设置属性值为 lighter、bold、bolder 或 100-900 之间的整百数值改变文字的粗细。注意，字体能否被 bolder 或 lighter 更改取决于字体包是否存在该样式。

font-style 可以设置字体样式，并且有 3 种设置方式：

1. 正常字体为 normal，也是 font-style 的默认值
2. italic 为字体设置为斜体，用于字体本身就有倾斜的样式
3. oblique 强制将字体倾斜

字体样式

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>字体样式</title>
```

```
6       <style type="text/css">
7           p {
8               font-family: "arial";
9               font-size: 20px;
10              font-weight: bold;
11              font-style: italic;
12          }
13      </style>
14  </head>
15  <body>
16      <p>Cascading Style Sheets (CSS) is a style sheet language
            used for describing the presentation of a document
            written in a markup language such as HTML.</p>
17  </body>
18  </html>
```

7.2 上个色——color

7.2.1 color

color 属性可以设置字体颜色。color 的值有 3 种设置方式：

1. 英文命令颜色。

```
color: red;
```

2. 十六进制颜色代码：使用 6 位十六进制数表示光学三原色“红绿蓝”。

R	H	B
00 - FF	00 - FF	00 - FF

表 7.1: 颜色代码

如果每两位十六进制数都相同，可简写。

颜色代码	颜色
#F00	红色
#0F0	绿色
#00F	蓝色
#000	黑色
#FFF	白色
#0FF	青色
#F40	淘宝红

表 7.2: 常见颜色代码

3. 颜色函数 rgb()：由光学三原色 RGB 的比例来配色。rgb() 函数中每一项的值可以是 0-255 之间的整数，也可以是 0%-100% 的百分数。

```
color: rgb(133, 45, 200);
color: rgb(20%, 33%, 25%);
```

7.3 文本样式

7.3.1 文本样式

使用 text-decoration 可以设置对文本的修饰，默认值为 none；属性值为 underline 为下划线，属性值为 overline 为上划线，属性值为 line-through 为穿过文本的线，一般用于商品折扣价。使用 line-height 可以设置段落中的行间距离（行高）。使用 text-align 可以为文本设置对齐方式，属性值包括 left、right 和 center。

文本样式

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>文本样式</title>
    <style type="text/css">
        h2, p {
            text-decoration: underline;
            line-height: 2em;   /* 两倍行间距 */
            text-align: center;
        }
    </style>
</head>
<body>
    <h2>《望庐山瀑布》</h2>
    <p>唐·李白</p>
    <p>
        日照香炉生紫烟，<br/>
        遥看瀑布挂前川。<br/>
        飞流直下三千尺，<br/>
        疑是银河落九天。
    </p>
</body>
</html>
```

Chapter 8 盒子模型与布局模型

8.1 元素分类

8.1.1 我要独占一行——块级元素

在 HTML 中 、<p>、<h1>、<form>、<ul>、 等都是块级元素。每个块级元素都从新的一行开始，并且其后的元素也另起一行。块级元素的高度、宽度、行高遗迹顶和底边距都可以设置，宽度在不设定的情况下，是它本身父容器的 100%（和父元素的宽度一致）。

块级元素

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>块级元素</title>
</head>
<body>
    <div>这是一个div标签</div>
    <p>这是一个p标签</p>
    <h1>这是一个h1标签</h1>
</body>
</html>
```

通过设置display：block可以将元素显示为块级元素。

8.1.2 我要和你站一起——内联元素

在 HTML 中，、<a>、<label>、、 等都是内联元素（行内元素）。内联元素和其它元素都在一行上，元素的高度、宽度及顶部和底部编剧不可设置，元素的宽度就是它包含的文字或图片的宽度，不可改变。

块级元素也可以设置display：inline将元素设置为内联元素。

内联元素与块级元素转换

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>内联元素与块级元素转换</title>
    <style type="text/css">
        a {
            display: block;
        }

        div {
            display: inline;
        }
    </style>
</head>
<body>
    <a>我要变成块级元素</a>
    <a>我也要变成块级元素</a>
    <div>我要变成内联元素</div>
    <div>我也要变成内联元素</div>
</body>
</html>
```

8.1.3 我还要占个大位置——内联块状元素

内联块状元素就是同时具备内联元素和块级元素的特点。内联块状元素的特点是和其它元素都在一行上，但元素的高度、宽度、行高以及顶和底边距都可以设置。

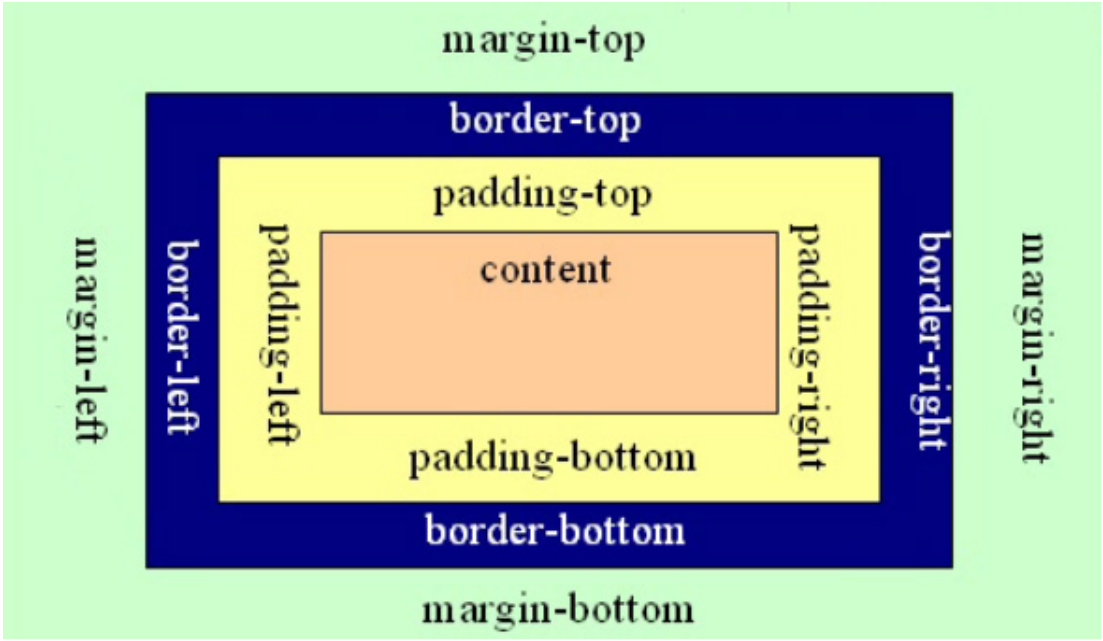
通过设置display：inline－block就可以将元素设置为内联块状元素。如 <img>、<input> 就是这种内联块状标签。

8.2 盒子模型

8.2.1 盒子模型

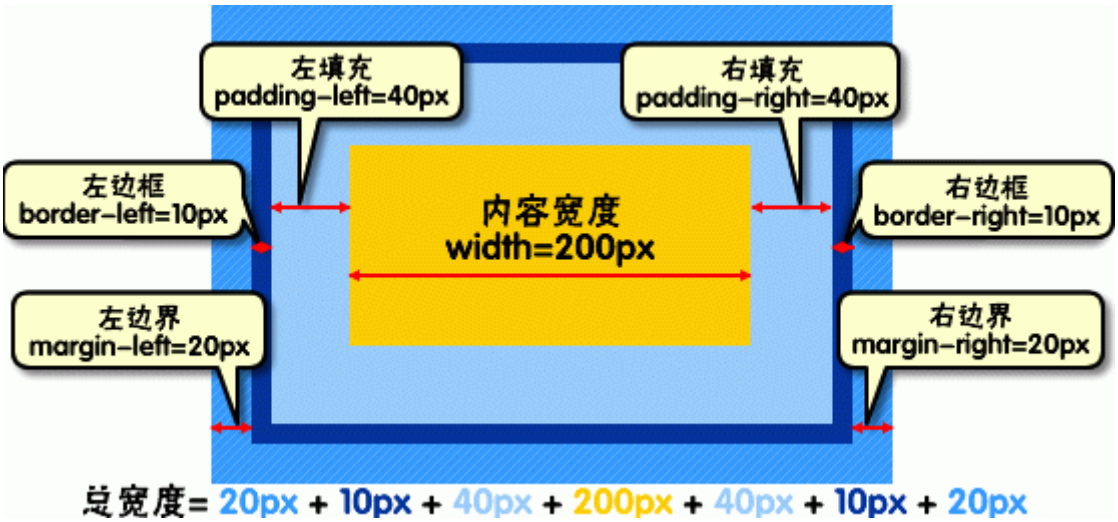
盒子模型包含 4 个部分：

1. 外边距（margin）
2. 外边框（border）
3. 内边距（padding）
4. 内容（content）



盒模型的宽度和高度和平常所说的物体的宽度和高度的理解是不一样的，CSS 内定义的宽和高，指的是盒模型中内容的宽和高。

元素实际的宽度（盒子的宽度）= 左外边距 + 左边框 + 左内边距 + 内容宽度 + 右内边距 + 右边框 + 右外边距



8.2.2 外边框（Border）

盒子模型的外边框可以设置粗细、样式和颜色：

1. border-width：设置边框的宽度，常用单位为像素（px）
2. border-style：边框样式包括实线 solid、虚线 dashed、点线 dotted
3. border-color：设置边框颜色

当三个属性都需要被设置时，也可以使用 border 属性简写。例如设置一个粗细为 1px 实心的黑色边框：

```
1 border: 1px solid black;
```

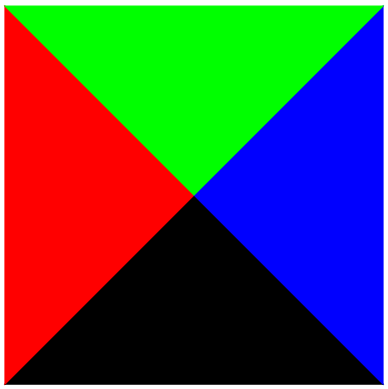
CSS 中也允许只为一个方向的边框设置属性，使用 border-top、border-bottom、border-left、border-right 可以单独设置上、下、左、右四条边框。

元素边框的圆角效果可以使用 border-radius 属性来设置。设置圆角的值的顺序为左上、右上、右下、左下。如果 border-radius 属性只给出一个值表示四个圆角都被设置成该值。如果只给出两个值表示左上角和右下角设置为第一个值，右上角和左下角设置为第二个值。当给一个正方形的圆角效果值设置为其宽度一半时，显示效果为圆形。

圆形

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>圆形</title>
    <style type="text/css">
        div {
            width: 100px;
            height: 100px;
            background-color: red;
            border-radius: 50%;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

通过单独设置每条边的属性，可以在正方形内展现不同的颜色。

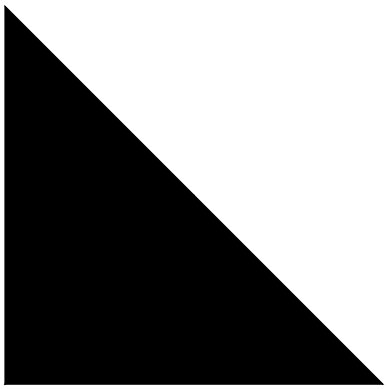


四色正方形

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>四色正方形</title>
```

```
    <style type="text/css">
        div {
            width: 0;
            height: 0;
            border: 100px solid black;
            border-left-color: red;
            border-top-color: green;
            border-right-color: blue;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

通过设置透明色（transparent），可以绘制出三角形。



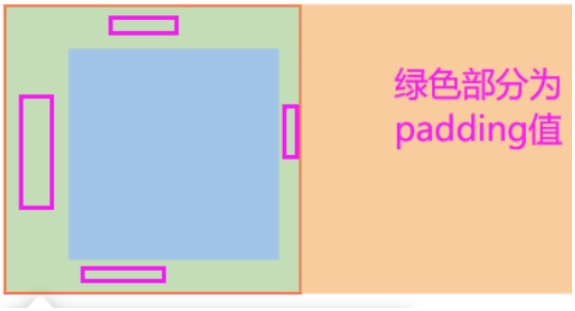
三角形

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>三角形</title>
    <style type="text/css">
        div {
            width: 0;
            height: 0;
```

```
            border: 100px solid black;
            border-left-color: black;
            border-top-color: transparent;
            border-right-color: transparent;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

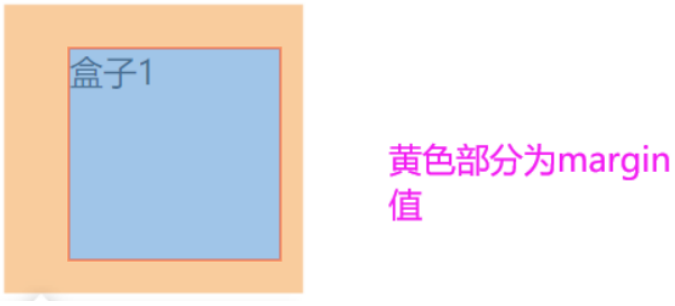
8.2.3 内边距（Padding）

元素内容与边框之间是可以设置距离的，称之为内边距或填充。内边距也可分为上、右、下、左（顺时针）。



8.2.4 外边距（Margin）

元素与其他元素之间的距离可以使用外边距来设置，外边距也可分为上、右、下、左。



margin 与 padding 的区别在于，padding 在外边框里，margin 在外边框外。

8.3 布局模型

8.3.1 布局模型

布局模型建立在盒子模型的基础上，在网页中，元素有 3 种布局模型：

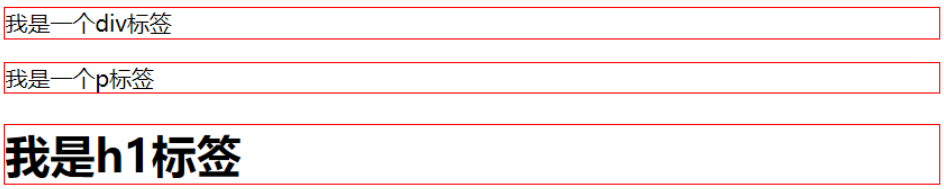
1. 流动模型（flow）
2. 浮动模型（float）
3. 层模型（layer）

8.3.2 流动模型

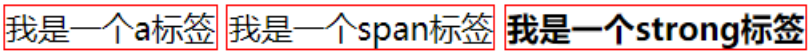
流动模型是默认的网页布局模式，也就是说网页在默认状态下的 HTML 网页元素都是根据流动模型来分布网页内容的。

流动布局模型有 2 个典型的特征：

1. 块级元素都会在所处的包含元素内自上而下按顺序垂直延伸分布，因为在默认状态下，块级元素的宽度都为 100%，即都会以行的形式占据位置。



2. 内联元素都会在所处的包含元素内从左到右水平分布显示。



8.3.3 浮动模型

块级元素这么霸道都是独占一行，如果想让两个块级元素并排显示，可以设置元素浮动。任何元素在默认情况下是不能浮动的，但可以设置 float 属性定义为浮动，如 、<p>、<table>、<img> 等元素都可以被定义为浮动。

浮动模型

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>浮动模型</title>
    <style type="text/css">
        div {
            width: 100px;
            height: 100px;
            border: 1px solid red;
        }

        #div1 {
            float: left;
        }

        #div2 {
            float: right;
        }
    </style>
</head>
<body>
    <div id="div1"></div>
    <div id="div2"></div>
</body>
</html>
```

8.4 定位

8.4.1 层模型

层布局模型就像是图像软件 PhotoShop 中非常流行的图层编辑功能一样，每个图形能够精确定位操作（positioning）。由于有些元素是定点展示的，定位技术可以让元素在特定的位置出现。

层模型有 3 种形式：

1. 绝对定位（absolute）
2. 相对定位（relative）
3. 固定定位（fixed）

8.4.2 万事无绝对——绝对定位

如果想为元素设置层模型中的绝对定位，需要设置position：absolute，这条语句的作用是将元素从文档流中拖出来，然后使用 left、right、top、bottom 属性相对于其最接近的一个具有定位属性的父包含块进行绝对定位。如果不存在这样的包含块，则相对于 body 元素，即相对于浏览器窗口。

当一个元素成为绝对定位元素，它就脱离了原来的层面，原来的位置会空出，下面的元素都会上来。

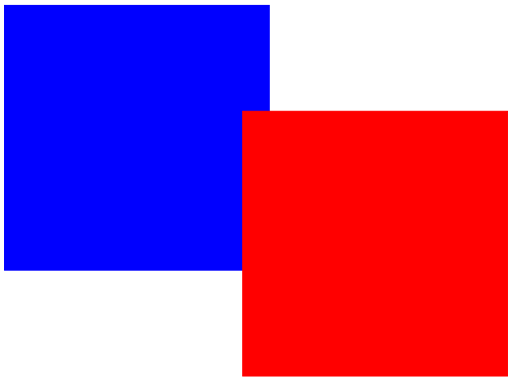
绝对定位

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>绝对定位</title>
    <style type="text/css">
        div {
            width: 100px;
```

```
            height: 100px;
        }

        #div1 {
            background-color: red;
            position: absolute;
            top: 50px;      /* 距离窗口上边50px */
            left: 100px;    /* 距离窗口左边100px */
        }

        #div2 {
            background-color: blue;
        }
    </style>
</head>
<body>
    <div id="div1"></div>
    <div id="div2"></div>
</body>
</html>
```



8.4.3 相对于自己的位置——相对定位

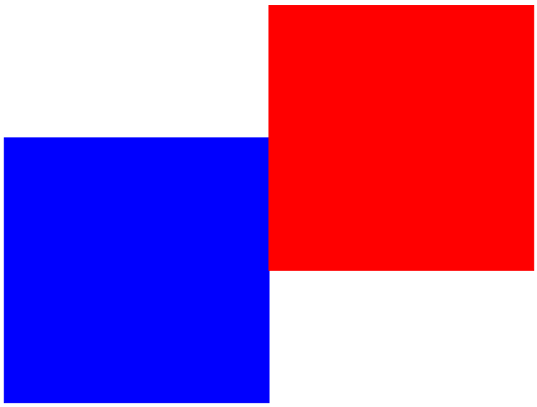
如果想为元素设置层模型中的相对定位，需要设置position：relative，它通过left、right、top、bottom 属性确定元素在正常文档流中的偏移位置。相对定位完成的过程是首先按 static (float) 方式生成一个元素，然后相对于以前的位置移动，并且偏移前的位置保留不动，因此下面的元素无法上来。

相对定位

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>相对定位</title>
    <style type="text/css">
        div {
            width: 100px;
            height: 100px;
        }

        #div1 {
            background-color: red;
            position: relative;
            top: 50px;
            left: 100px;
        }

        #div2 {
            background-color: blue;
        }
    </style>
</head>
<body>
    <div id="div1"></div>
    <div id="div2"></div>
</body>
</html>
```



使用position：absolute可以实现元素相对于浏览器（body）设置定位，那可不可以相对于其它元素进行定位呢？当然可以，但是必须要满足以下的 3 点条件：

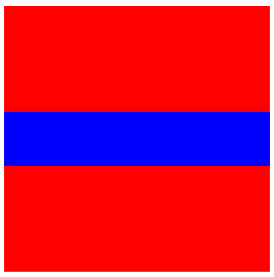
1. 参照定位的元素必须是相对定位元素的前辈元素。
2. 参照定位的元素必须加入position： relative 。
3. 定位元素加入 ‘position: absolute‘后便可以使用 top、bottom、left、right 来进行偏移了。

相对父元素定位

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>相对父元素定位</title>
    <style type="text/css">
        #div1 {
            width: 100px;
            height: 100px;
            background-color: red;
            position: relative;
        }

        #div2 {
            width: 100px;
            height: 20px;
            background-color: blue;
```

```
            position: absolute;
            top: 40px;
        }
    </style>
</head>
<body>
    <div id="div1">
        <div id="div2"></div>
    </div>
</body>
</html>
```



8.4.4 我就在那不动了——固定定位

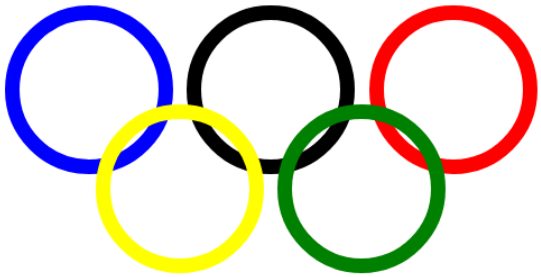
通过设置position：fixed让一个元素固定在一个位置，它的相对移动的坐标是视图（屏幕内的网页窗口）本身。由于视图本身是固定的，它不会随着浏览器窗口的滚动条滚动而变化，除非在屏幕中移动浏览器窗口的屏幕位置，或改变浏览器窗口的显示大小。因此固定定位的元素会始终位于浏览器窗口内视图的某个位置，不会受文档流影响。

固定定位

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>固定定位</title>
    <style type="text/css">
        img {
```

```
            position: fixed;
            top: 15%;
            left: 20%;
        }
    </style>
</head>
<body>
    <img src="https://gimg2.baidu.com/image_search/src=http%3A%2F
        %2Fc-ssl.duitang.com%2Fuploads%2Fitem%2F201702%2F02%2
        F20170202151743_rvLWi.jpeg&refer=http%3A%2F%2
        Fc-ssl.duitang.com&app=2002&size=f9999,10000&q=a80&n=0&
        g=0n&fmt=jpeg?sec=1618588336&
        t=2612dc782456b9fdb8eeac0d0a215a85" alt="灰原哀.jpg"
        title="灰原哀">
    <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
        <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
        <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
        <br><br><br><br><br><br><br>
</body>
</html>
```

奥运五环



olympic.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>奥运五环</title>
    <link rel="stylesheet" type="text/css" href="olympic.css">
```

```
</head>
<body>
    <!-- 展示奥运五环的区域 -->
    <div class="stage">
        <div id="circle1"></div>
        <div id="circle2"></div>
        <div id="circle3"></div>
        <div id="circle4"></div>
        <div id="circle5"></div>
    </div>
</body>
</html>
```

olympic.css

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

    .stage {
        position: absolute;
        left: 50%;
        top: 50%;
        margin-left: -190px;
        margin-top: -93px;
        height: 185px;
        width: 380px;
    }

    #circle1,
    #circle2,
    #circle3,
    #circle4,
    #circle5 {
        position: absolute;
        width: 100px;
        height: 100px;
```

```
        border: 10px solid black;
        border-radius: 50%;
    }

    #circle1 {
        border-color: blue;
        left: 0;
        top: 0;
    }

    #circle2 {
        border-color: black;
        left: 130px;
        top: 0;
    }

    #circle3 {
        border-color: red;
        left: 260px;
        top: 0;
    }

    #circle4 {
        border-color: yellow;
        left: 65px;
        top: 70px;
    }

    #circle5 {
        border-color: green;
        left: 195px;
        top: 70px;
    }
```

Part III

JavaScript

Chapter 9 JavaScript 简介

9.1 编程简介

9.1.1 编程简介

程序（program）是为了让计算机执行某些操作或者解决问题而编写的一系列有序指令的集合。由于计算机只能够识别二进制数字 0 和 1，因此需要使用特殊的编程语言来描述如何解决问题过程和方法。

算法（algorithm）是可完成特定任务的一系列步骤，算法的计算过程定义明确，通过一些值作为输入并产生一些值作为输出。

流程图（flow chart）是算法的一种图形化表示方式，使用一组预定义的符号来说明如何执行特定任务。

- 圆角矩形：开始和结束
- 矩形：数据处理
- 平行四边形：输入/输出
- 菱形：分支判断条件
- 流程线：步骤

9.1.2 编程语言（Programming Language）

编程语言主要分为面向机器、面向过程和面向对象三类。JavaScript 是一种具有函数优先的轻量级，解释型或即时编译型的编程语言。虽然它是作为开发 Web 页面的脚本语言而出名，但是它也被用到了很多非浏览器环境中。JavaScript 基于原型编程、多范式的动态脚本语言，并且支持面向对象、命令式、声明式、函数式编程范式。

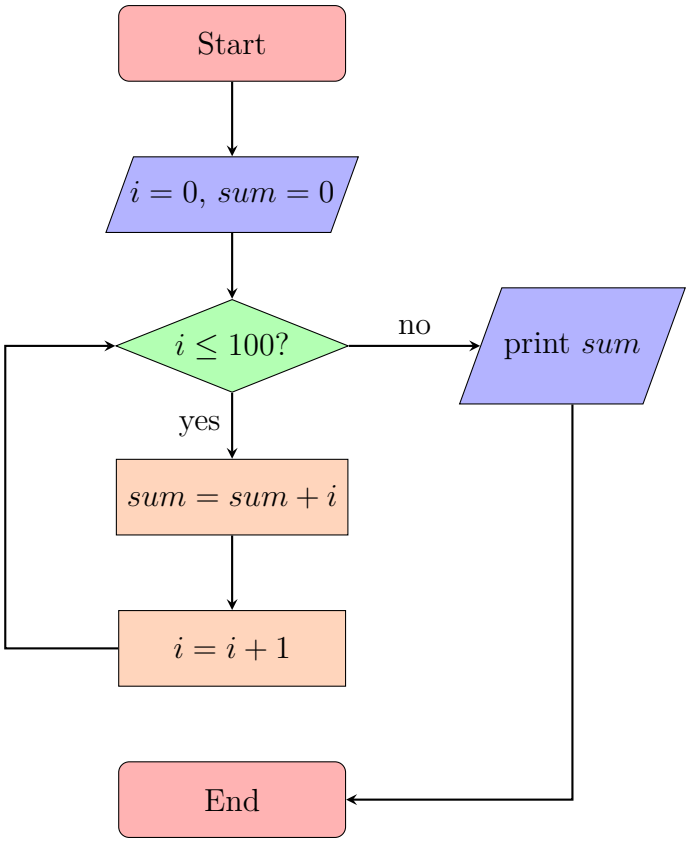


图 9.1: 计算 $\sum_{i=1}^{100} i$ 的流程图

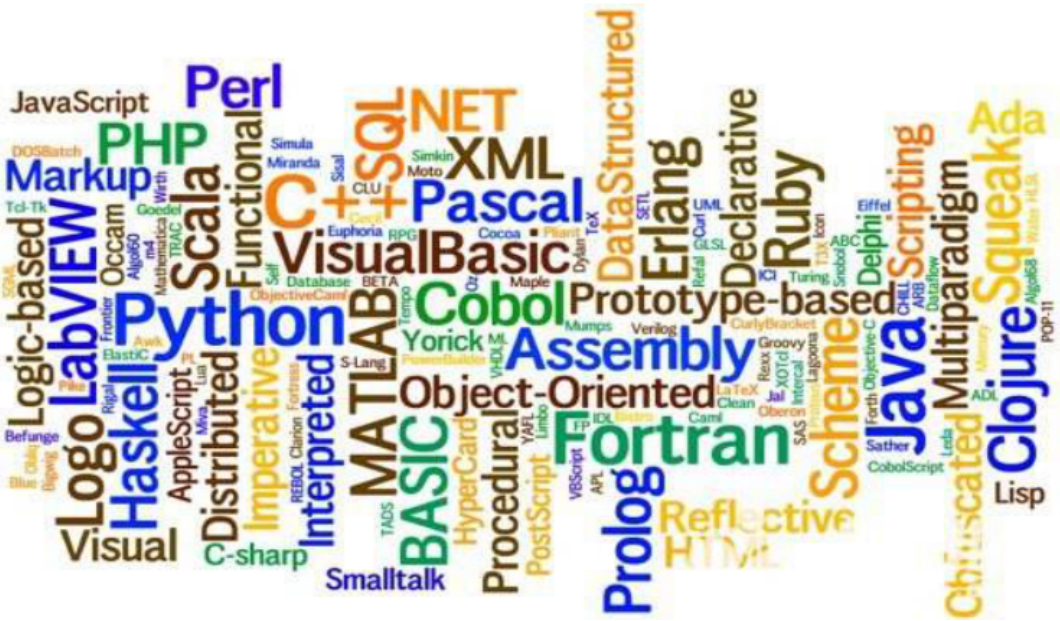


图 9.2: 常见编程语言

9.2 Hello World!

9.2.1 Hello World!

使用 <script> 在 HTML 网页中插入 JavaScript 代码。

```
<script type="text/JavaScript">
    // JavaScript代码
</script>
```

type 属性表示在 <script> 之间的是文本类型，告诉了浏览器里面的文本属于 JavaScript 语言。

JS 作为一种脚本语言可以放在 HTML 页面中任何位置，但是一般放在网页的 <head> 或 <body> 部分。最常用的方式是在页面 <head> 部分放置 <script> 元素，浏览器解析 HTML 时是按先后顺序的，所以前面的 <script> 就先被执行。

9.2.2 引入外部 JS 文件

除了在 HTML 文件中使用 <script> 添加 JS 代码，还能把 HTML 和 JS 分开，单独创建一个 JS 文件，其文件后缀为 “.js”，然后直接将 JS 代码写在 JS 文件中。注意在 JS 文件中，不需要 <script>，直接编写 JS 代码即可。

JS 文件不能直接运行，需要嵌入到 HTML 文件中执行，因此需要在 HTML 中引入外部 JS 文件。

```
<script src="JS文件.js"></script>
```

Hello World!

hello_world.html

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

```
<head>
    <meta charset="UTF-8">
    <title>Hello World!</title>
    <script src="hello_world.js"></script>
</head>
<body>

</body>
</html>
```

hello_world.js

```
document.write("Hello World!");
```

9.3 语句与注释

9.3.1 语句（Statement）

JS 语句是发给浏览器的命令，这些命令的作用是告诉浏览器要做的事情。一行语句通常在结尾加上一个【;】表示语句的结束，虽然分号也可以不写，但是要养成编程的好习惯，记得在语句末尾加上分号。

9.3.2 注释（Comment）

在编程中加入注释可以增加程序的可读性和可维护性，编译器不会对注释的部分进行编译。

JS 中注释分为两类：

1. 单行注释：将一行内【//】之后的内容视为注释。
2. 多行注释：以【/*】开始，【*/】结束，中间的内容视为注释。

注释

```
1 document.write("单行注释");      // 我是单行注释
2 /*
3     我是
4     多行
5     注释
6 */
7 document.write("多行注释");
```

9.4 互动方法

9.4.1 document.write()

document.write() 可用于直接向 HTML 输出流写内容，简单地说就是直接在网页中输出内容。

```
1 document.write("互动方法：document.write()");
```

9.4.2 console.log()

console.log() 用于在控制台输出信息，该方法对于开发过程进行测试很有帮助。

```
1 console.log("互动方法：console.log()");
```

9.4.3 alert()

在访问网站的时候，有时会突然弹出一个小窗口，上面写着一段提示信息文字，如果不点击“确定”，就不能对网页做任何操作，这个小窗口就是使用 alert() 实现的。

```
1 alert("互动方法：alert()");
```

9.4.4 confirm()

confirm() 消息对话框通常用于允许用户做选择的动作，弹出对话框包含一个确定按钮和一个取消按钮。

confirm() 的返回值为 boolean 值，当用户点击“确定”按钮时返回 true，当用户点击“取消”按钮时返回 false。

```
1 confirm("互动方法：confirm()");
```

9.4.5 prompt()

prompt() 弹出消息对话框，通常用于询问一些需要与用户交互的信息，弹出的消息对话框包含一个确定按钮、取消按钮和一个文本输入框。

prompt() 接受两个参数，第一个参数为要显示在消息对话框中的文本，不可修改；第二个参数为文本框中的内容，可以修改。点击“确定”按钮，文本框中的内容将作为函数的返回值；点击“取消”按钮，将返回 null。

```
1 prompt("互动方法：prompt()", "文本内容");
```

9.5 变量

9.5.1 变量（Variable）

变量是计算机中一块特定的内存空间，由一个或多个连续的字节组成，不同数据存入具有不同内存地址的空间，相互独立，通过变量名可以简单快速地找到在内存中存储的数据。定义变量使用关键字 var。

```
1 var variable_name;
```

变量名需要符合以下的要求：

1. 由字母、数字和下划线组成，第一个字符必须为字母或下划线。
2. 不能包含除【_】以外的任何特殊字符，如【%】、【#】等。
3. 不可以使用保留字或关键字。
4. 准确、顾名思义，不要使用汉语拼音。

关键字是编程语言内置的一些名称，具有特殊的用处和意义。

break	else	new	var	case
finally	return	void	catch	for
switch	while	default	if	throw
do	instanceof	typeof	abstract	enum
short	boolean	export	interface	static
extends	long	super	char	final
synchronized	class	float	package	throws
goto	private	transient	debugger	implements
volatile	double	import	public	delete
in	try	int	byte	native
const	protected			

表 9.1: 关键字

9.5.2 初始化（Initialization）

变量可以在定义时初始化，也可以在定义后初始化。

在编程中，【=】不是数学中的“等于”符号，而是表示“赋值”，即将【=】右边的值赋给左边的变量。

```
1 var n = 10;
2 var wage = 8232.56;
```

JS 变量虽然也可以不声明直接使用，但是这样不规范。

9.6 数据类型

9.6.1 数据类型

JS 中变量主要有以下几种类型：

1. 数字 number：整型、浮点型
2. 字符串 string
3. 布尔 boolean
4. 空对象 null
5. 未定义 undefined
6. 复杂数据类型：数组 Array、对象 Object

使用 typeof() 可以检测一个变量的数据类型。

数据类型

```
1 var a = 12;
2 var b = 3.1415;
3 var c = "hello";
4 var d = false;
5
6 console.log(typeof(a))
7 console.log(typeof(b))
8 console.log(typeof(c))
9 console.log(typeof(d))
```

运行结果

```
number
number
string
boolean
```

9.6.2 类型转换

类型转换是把变量从一种类型转换为另一种数据类型。类型转换可以是隐式的，也可以是显式的，通过使用强制类型转换的方法来指定。

类型转换

```
1 console.log(Number("123"));
2 console.log(parseInt("456"));
3 console.log(parseFloat("78.9"));
4 console.log(String(123456789));
5 console.log(Boolean(0));
6 console.log(Boolean(1));
```

运行结果

```
123
456
78.9
123456789
false
true
```

9.7 算术运算符

9.7.1 四则运算

数学符号	JS 符号	含义
+	+	加法
−	-	减法
×	*	乘法
÷	/	除法
.....	%	取模

表 9.2: 四则运算

JS 中除法【/】的意义与数学中不同：

1. 当相除的两个运算数都为整型，则运算结果为两个数进行除法运算后的整数部分，例如 21 / 5 的结果为 4。

2. 如果两个运算数其中至少一个为浮点型，则运算结果为浮点型，如 21 / 5.0 的结果为 4.2。

取模（modulo）【%】表示求两个数相除之后的余数，如 22 % 3 的结果为 1；4 % 7 的结果为 4。

9.7.2 复合赋值运算符

运算符	描述
+=	a += b 等价于 a = a + b
-=	a -= b 等价于 a = a - b
*=	a *= b 等价于 a = a * b
/=	a /= b 等价于 a = a / b
%=	a %= b 等价于 a = a % b

表 9.3: 复合赋值运算符

9.8 表达式

9.8.1 字符串连接

【+】运算符不只代表加法，还可以用于连接两个字符串。

字符串连接

```
1 console.log("Hello" + "World");
```

运行结果

```
HelloWorld
```

9.8.2 转义字符

在一个字符串描述的过程中，有可能会有一些特殊字符的信息。

转义字符	描述
\\	表示一个反斜杠\
\'	表示一个单引号'
\"	表示一个双引号"
\n	换行
\t	横向制表符

表 9.4: 转义字符

转义字符

```
1 console.log("全球最大同性交友网站\n\'https://github.com\'");
```

运行结果

```
全球最大同性交友网站
'https://github.com'
```

9.8.3 表达式（Expression）

表达式与数学中的定义相似，表达式是值具有一定的值，用操作符把常数和变量连接起来的代数式。

使用 prompt() 可以获取用户输入的信息，返回值为字符串类型。

计算圆的面积

```
1 var PI = 3.14159;
2 var r = parseFloat(prompt("输入半径"));
3 var area = PI * r ** 2;
4 console.log("面积 = " + area);
```

运行结果

```
输入半径: 5
面积 = 78.53975
```

Math.floor() 的作用是向下取整，Math.ceil() 的作用是向上取整。

逆序三位数

```
1 var num = parseInt(prompt("输入一个正三位数"));
2 var a = Math.floor(num / 100);
3 var b = Math.floor(num / 10) % 10;
4 var c = num % 10;
5
6 num = c * 100 + b * 10 + a;
7 console.log("逆序: " + num);
```

运行结果

```
输入一个正三位数：520
逆序：25
```

Chapter 10 判断

10.1 逻辑运算符

10.1.1 关系运算符

数学符号	关系运算符
$<$	<
$>$	>
$\leq$	<=
$\geq$	>=
$\neq$	!=
$=$	==

表 10.1: 关系运算符

10.1.2 逻辑运算符

JS 中逻辑运算符有三种：

1. 逻辑与 && (logical AND)：当多个条件同时为真，结果为真。

条件 1	条件 2	条件 1 && 条件 2
T	T	T
T	F	F
F	T	F
F	F	F

表 10.2: 逻辑与

2. 逻辑或 || (logical OR)：多个条件有一个为真时，结果为真。

条件 1	条件 2	条件 1 || 条件 2
T	T	T
T	F	T
F	T	T
F	F	F

表 10.3: 逻辑或

3. 逻辑非!（logical NOT）：条件为真时，结果为假；条件为假时，结果为真。

条件	！条件
T	F
F	T

表 10.4: 逻辑非

10.2 if

10.2.1 if

当 if 语句的条件为真时，进入花括号执行内部的代码；若条件为假，则跳过花括号执行后面的代码。

if 语句主要有以下几种形式：

- 单分支

```
var age = 15;
if(age > 0 && age < 18) {
    console.log("未成年")
}
```

- 双分支

```
var age = 30;
if(age > 0 && age < 18) {
    console.log("未成年人")
} else {
    console.log("成年人");
}
```

- 多分支

```
var score = 76;

if(score >= 90 && score <= 100) {
    console.log("优秀");
} else if(score >= 60 && score < 90) {
    console.log("合格");
} else {
    console.log("不合格");
```

```
9 }
```

10.2.2 嵌套结构

if 语句也可以嵌套使用：

```
1 if(条件1) {
2     if(条件2) {
3         // code
4     }
5 }
```

判断整数奇偶

```
1 var num = parseInt(prompt("输入一个正整数"));
2
3 if(num > 0) {
4     if(num % 2 == 0) {
5         console.log(num + "是偶数");
6     } else {
7         console.log(num + "是奇数");
8     }
9 }
```

运行结果

```
输入一个正整数：66
66是偶数
```

10.3 switch

10.3.1 switch

switch-case 结构可以对整数值的表达式进行判断。

```
switch(表达式) {
    case label:
        //code
        break;
    // ...
    default:
        //code
        break;
}
```

根据表达式的值，跳转到对应的 case 处进行执行。需要注意的是，当对应的 case 中的代码被执行完后，并不会跳出 switch，而是会继续执行后面的代码，所以需要使用 break 跳出 switch 结构。当所有 case 都不满足表达式的值时，会执行 default 语句中的代码，相当于 if-else 结构中的 else。

根据月份输出对应的英语简写

```
var month = parseInt(prompt("输入月份"));

switch(month) {
    case 1:
        console.log("Jan.");
        break;
    case 2:
        console.log("Feb.");
        break;
    case 3:
        console.log("Mar.");
        break;
    case 4:
```

```
        console.log("Apr.");
        break;
    case 5:
        console.log("May");
        break;
    case 6:
        console.log("Jun.");
        break;
    case 7:
        console.log("Jul.");
        break;
    case 8:
        console.log("Aug.");
        break;
    case 9:
        console.log("Sep.");
        break;
    case 10:
        console.log("Oct.");
        break;
    case 11:
        console.log("Nov.");
        break;
    case 12:
        console.log("Dec.");
        break;
    default:
        alert("输入有误");
        break;
}
```

运行结果

```
输入月份：5
May
```

Chapter 11 循环

11.1 自增/自减运算符

11.1.1 自增/自减运算符

单目运算符中自增 ++ 和自减--运算符可以将变量的值加 1 和减 1，但是 ++ 和--可以出现在变量之前或之后，即有四种情况：

1. 前缀自增
2. 前缀自减
3. 后缀自增
4. 后缀自减

表达式	含义
count++	执行完所在语句后自增
++count	执行所在语句前自增
count–	执行完所在语句后自减
–count	执行所在语句前自减

表 11.1: 自增/自减运算符

11.2 while

11.2.1 while

在 while 循环中，当条件满足时重复循环体内的语句。如果条件永远为真，循环会永无止境的进行下去（死循环），因此循环体内要有改变条件的机会。

控制循环次数的方法就是设置循环变量：初值、判断、更新。

while 循环的特点是先判断、再执行，所以循环体有可能会进入一次或多次，也有可能一次也不会进入。

```
while(条件) {
    // code
}
```

计算 5 个人的平均身高

```
var height;
var total = 0;
var average;
var i = 1;

while(i <= 5) {
    height = parseFloat(prompt("输入第" + i + "个人的身高"));
    total += height;
    i++;
}

average = total / 5;
console.log("平均身高：" + average);
```

运行结果

```
输入第1个人的身高：160.8
输入第2个人的身高：175.2
输入第3个人的身高：171.2
输入第4个人的身高：181.3
输入第5个人的身高：164
平均身高：170.5
```

11.3 do-while

11.3.1 do-while

do-while 循环在进入循环的时候不做检查，而是在执行完一轮循环体的代码之后，再来检查循环的条件是否满足，如果满足则继续下一轮循环，不满足则结束循环，即至少执行一次循环。

do-while 循环的主要特点是先执行、再判断。

```
do {
    // code
} while(条件);
```

计算整数位数

```
var num = 123;
var n = 0;

do {
    num = Math.floor(num / 10);
    n++;
} while(num != 0);

console.log("位数: " + n);
```

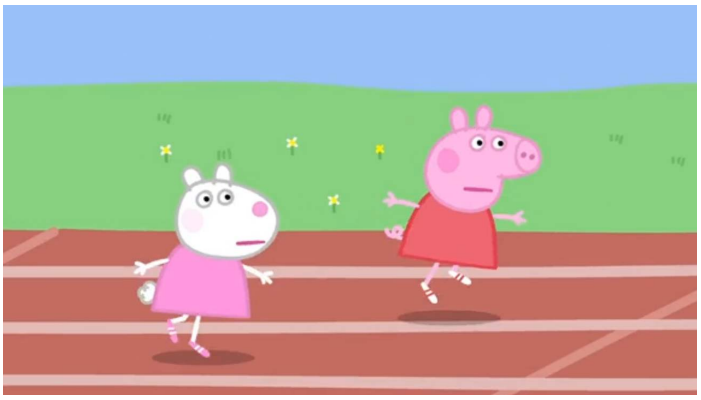
运行结果

```
输入整数: 123
位数: 3
```

11.3.2 while 与 do-while 区别

while 循环与 do-while 循环有以下区别:

1. 执行顺序不同。
2. 初始情况不满足循环条件时，while 循环一次都不会执行，do-while 循环不管任何情况都至少执行一次。
3. do-while 循环的 while 语句后有【;】。



猜数字

```
//产生1-100之间的随机数
var answer = Math.floor(Math.random() * 100) + 1;
var num = 0;
var cnt = 0;

do {
    num = parseInt(prompt("猜一个1-100之间的数字"));
    cnt++;
    if(num > answer) {
        alert("猜大了！ ");
    } else if(num < answer) {
        alert("猜小了！ ");
    }
} while(num != answer);

alert("猜对了！ 你一共用了" + cnt + "次猜对！ ");
```

运行结果

```
猜一个1-100之间的数字：50
猜大了！
猜一个1-100之间的数字：25
猜小了！
猜一个1-100之间的数字：37
猜小了！
猜一个1-100之间的数字：43
猜小了！
猜一个1-100之间的数字：46
猜小了！
猜一个1-100之间的数字：48
猜小了！
猜一个1-100之间的数字：49
猜对了！你一共用了7次猜对！
```

11.4 for

11.4.1 for

for 循环有三个表达式，中间用【;】分隔，【;】不可省略。

```
for(表达式1; 表达式2; 表达式3) {
    //code
}
```

- 表达式 1 通常是为循环变量赋初值，可省略。
- 表达式 2 是循环条件，判断是否继续执行循环，可省略。
- 表达式 3 为更新循环变量的值，可省略。

计算 1-100 的累加和

```
var sum = 0;
for(var i = 1; i <= 100; i++) {
    sum += i;
}
console.log("累加： " + sum);
```

运行结果

```
累加： 5050
```

计算 $1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}$

```
var n = 10;
var sum = 0;

for(var i = 1; i <= n; i++) {
    sum += 1 / i;
```

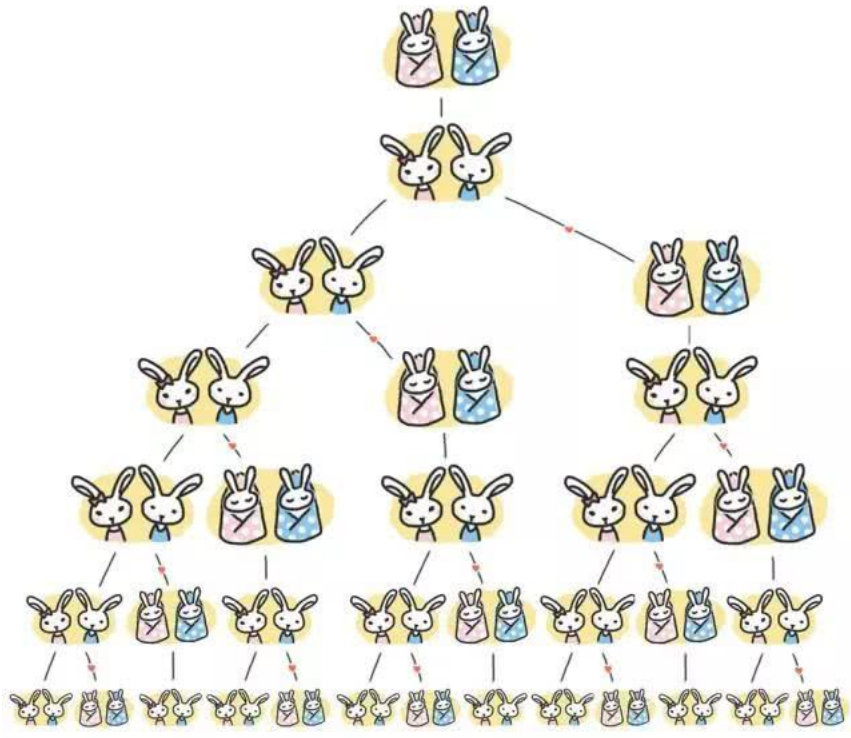
```
}

console.log(sum);
```

运行结果

```
2.928968
```

斐波那契数列



```
var n = 10;
var num1, num2, val;
var str = "";

if(n == 1) {
    console.log("1");
} else if(n == 2) {
    console.log("1, 1");
} else {
    num1 = 1;
```

```
    num2 = 1;
    str = "1, 1";
    for(var i = 3; i <= n; i++) {
        val = num1 + num2;
        str += ", " + val;
        num1 = num2;
        num2 = val;
    }
    console.log(str);
}
```

运行结果

```
1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

11.4.2 嵌套循环

循环也可以进行嵌套使用。

九九乘法表

1*1=1	1*2=2	1*3=3	1*4=4	1*5=5	1*6=6	1*7=7	1*8=8	1*9=9
2*1=2	2*2=4	2*3=6	2*4=8	2*5=10	2*6=12	2*7=14	2*8=16	2*9=18
3*1=3	3*2=6	3*3=9	3*4=12	3*5=15	3*6=18	3*7=21	3*8=24	3*9=27
4*1=4	4*2=8	4*3=12	4*4=16	4*5=20	4*6=24	4*7=28	4*8=32	4*9=36
5*1=5	5*2=10	5*3=15	5*4=20	5*5=25	5*6=30	5*7=35	5*8=40	5*9=45
6*1=6	6*2=12	6*3=18	6*4=24	6*5=30	6*6=36	6*7=42	6*8=48	6*9=54
7*1=7	7*2=14	7*3=21	7*4=28	7*5=35	7*6=42	7*7=49	7*8=56	7*9=63
8*1=8	8*2=16	8*3=24	8*4=32	8*5=40	8*6=48	8*7=56	8*8=64	8*9=72
9*1=9	9*2=18	9*3=27	9*4=36	9*5=45	9*6=54	9*7=63	9*8=72	9*9=81

表 11.2: 九九乘法表

```
var str = "";
```

```
2
3  for(var i = 1; i <= 9; i++) {
4      for(var j = 1; j <= 9; j++) {
5          str += i + "*" + j + "=" + i*j + "\t";
6      }
7      str += "\n";
8  }
9  console.log(str);
```

输出图案

```
*
**
***
****
*****
```

```
1  var str = "";
2
3  for(var i = 1; i <= 5 i++) {
4      for(var j = 1; j <= i; j++) {
5          str += "*";
6      }
7      str += "\n";
8  }
9  console.log(str);
```

11.5 break or continue?

11.5.1 循环控制

循环控制语句的作用是控制当前的循环结构是否继续向下执行，如果不进行控制，那么会根据既定的结构重复执行。如果有一些特殊的情况导致循环的执行中断，就称为循环的控制语句。循环控制语句的关键字有 break 和 continue。

break 的作用是跳出当前循环，执行当前循环之后的语句。break 只能跳出一层循环，如果是嵌套循环，那么需要按照嵌套的层次，逐步使用 break 来跳出。break 语句只能在循环体内和 switch 语句内使用。

continue 的作用是跳过本轮循环，开始下一轮循环的条件判断。continue 终止当前轮的循环过程，但它并不跳出循环。

break

```
var str = "";

for(var i = 1; i <= 10; i++) {
    if(i == 5) {
        break;
    }
    str += i + " ";
}

console.log(str);
```

运行结果

```
1 2 3 4
```

continue

```
var str = "";

for(var i = 1; i <= 10; i++) {
    if(i == 5) {
        continue;
    }
    str += i + " ";
}

console.log(str);
```

运行结果

```
1 2 3 4 6 7 8 9 10
```

Chapter 12 数组

12.1 一维数组

12.1.1 数组（Array）

一个变量只能存储一个内容，如果需要存储更多数据，就需要使用数组解决问题。一个数组变量可以存放多个数据，数组是一个值的集合，它们共享同一个名字，数组中的每个变量都能被其下标所访问。

a[0]	a[1]	a[2]	a[3]	a[4]

- 元素：数组中的每个变量
- 大小：数组的容量
- 下标 / 索引（index）：元素的位置，下标从 0 开始，必须为非负整数

使用数组之前首先要创建数组，数组的创建有 2 种方式：

1. 直接创建

```
1 var arr1 = [];          //创建一个空数组
2 var arr2 = [1, 2, 3];   //创建有内容的数组
```

2. 利用构造函数创建

```
1 var arr1 = new Array();                 //创建空数组
2 var arr2 = new Array(10);               //创建长度为10的数组
3 var arr3 = new Array(5, 4, 3, 2, 1);    //创建数组并初始化
```

虽然创建数组时指定了长度，但实际上数组都是变长的，也就是说即使指定了长度，仍然可以将元素存储在规定长度以外。

12.1.2 数组初始化

很多时候在使用数组之前需要将数组的内容全部清空，这可以利用循环来实现。

```
var arr = new Array(100);
for(var i = 0; i < arr.length; i++) {
    arr[i] = 0;
}
```

数组最大值和最小值

```
var num = [7, 6, 2, 9, 3, 1, 4, 0, 5, 8];
var max = num[0];
var min = num[0];

for(var i = 1; i < num.length; i++) {
    if(num[i] > max) {
        max = num[i];
    } else if(num[i] < min) {
        min = num[i];
    }
}

console.log("max = " + max);
console.log("min = " + min);
```

运行结果

```
max = 9
min = 0
```

12.2 二维数组

12.2.1 二维数组（2D Array）

二维数组包括行和列两个维度，可以看成是由多个一维数组组成。

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

直接创建二维数组

```
1 var arr = [[1, 2], [3, 4]];
2 console.log(arr);
```

运行结果

```
(2) [Array(2), Array(2)]
0: (2) [1, 2]
1: (2) [3, 4]
length: 2
```

构造函数创建二维数组

```
1 var arr = new Array(3);
2 for(var i = 0; i < arr.length; i++) {
3     arr[i] = new Array(4);
4 }
```

运行结果

```
(3) [Array(4), Array(4), Array(4)]
0: (4) [empty× 4]
1: (2) [empty× 4]
2: (4) [empty× 4]
length: 3
```

利用两层循环来初始化二维数组。

初始化二维数组

```
var arr = new Array(3);
for(var i = 0; i < arr.length; i++) {
    arr[i] = new Array(4);
    for(var j = 0; j < arr[i].length; j++) {
        arr[i][j] = 0;
    }
}
```

矩阵运算

矩阵的加法/减法是指两个矩阵把其相对应元素进行加减的运算。

矩阵加法：两个 $m \times n$ 矩阵 A 和 B 的和，标记为 $A+B$，结果为一个 $m \times n$ 的矩阵，其内的各元素为其相对应元素相加后的值。

矩阵减法：两个 $m \times n$ 矩阵 A 和 B 的差，标记为 $A-B$，结果为一个 $m \times n$ 的矩阵，其内的各元素为其相对应元素相减后的值。

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1-0 & 3-0 \\ 1-7 & 0-5 \\ 1-2 & 2-1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ -6 & -5 \\ -1 & 1 \end{bmatrix}$$

```
var A = [
    [1, 3],
    [1, 0],
    [1, 2]
];

var B = [
    [0, 0],
    [7, 5],
    [2, 1]
];

var C = new Array(3);
for (var i = 0; i < C.length; i++) {
    C[i] = new Array(2);
}

for(var i = 0; i < C.length; i++) {
    for (var j = 0; j < C[i].length; j++) {
        C[i][j] = A[i][j] + B[i][j];
    }
}
console.log("矩阵加法: " + C);

for(var i = 0; i < C.length; i++) {
    for(var j = 0; j < C[i].length; j++) {
```

```
27          C[i][j] = A[i][j] - B[i][j];
28      }
29  }
30  console.log("矩阵减法： " + C);
```

运行结果

```
矩阵加法： [ [ 1, 3 ], [ 8, 5 ], [ 3, 3 ] ]
矩阵减法： [ [ 1, 3 ], [ -6, -5 ], [ -1, 1 ] ]
```

12.3 数组操作

12.3.1 计算数组长度

引用数组的 length 属性获取数组长度，需要注意的是，JS 数组的 length 属性是可变的。

计算数组长度

```
1 var arr = [0, 1, 2, 3, 4];
2 console.log(arr.length);
3 arr.length = 10;
4 console.log(arr.length);
```

运行结果

```
5
10
```

12.3.2 增加元素

使用下一个未使用的索引，任何时刻可以不断向数组增加新元素。

增加元素

```
1 var arr = [0, 1, 2, 3, 4];
2 arr[5] = 5;
3 console.log(arr);
```

运行结果

```
[0, 1, 2, 3, 4, 5]
```

使用 unshift() 可以向数组第一个元素前面添加一个元素，返回值为数组长度。

unshift()

```
1 var arr = [0, 1, 2, 3, 4];
2 arr.unshift(5);
3 console.log(arr);
```

运行结果

```
[5, 0, 1, 2, 3, 4]
```

push() 可以向数组最后一个元素后面添加一个元素，返回值为数组长度。

push()

```
1 var arr = [0, 1, 2, 3, 4];
2 arr.push(5);
3 console.log(arr);
```

运行结果

```
[0, 1, 2, 3, 4, 5]
```

12.3.3 删除元素：

shift() 可以删除数组的第一个元素，返回值为被删除元素。

shift()

```
1 var arr = [0, 1, 2, 3, 4];
2 arr.shift();
3 console.log(arr);
```

运行结果

```
[1, 2, 3, 4]
```

pop() 可以删除数组的最后一个元素，返回值为被删除元素。

pop()

```
1 var arr = [0, 1, 2, 3, 4];
2 arr.pop();
3 console.log(arr);
```

运行结果

```
[0, 1, 2, 3]
```

12.3.4 合并数组

使用 concat() 可以合并两个或多个数组，该方法不会改变原有数组，而是返回一个新的合并完的数组。

concat()

```
1 var arr1 = [1, 2, 3, 4, 5];
2 var arr2 = [6, 7, 8, 9, 10];
3 console.log(arr1.concat(arr2));
4 console.log(arr1);
5 console.log(arr2);
```

运行结果

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5]
[6, 7, 8, 9, 10]
```

12.3.5 数组转字符串

使用 join(separator) 方法可以将数组转换为字符串，其中 ‘separator‘参数可选，用于指定要使用的分隔符，如果该参数省略，则使用逗号作为分隔符。

join()

```
1 var arr = [0, 1, 2, 3, 4];
2 console.log(arr.join());
```

运行结果

```
0,1,2,3,4
```

12.3.6 字符串转数组

6. 使用 split(separator, n) 可以将字符串转换为数组，其中参数 separator 必选，用于指定将字符串按某个字符切割成若干个子字符串，并以数组的形式返回。参数 n 可选，用于指定返回的数组的最大长度，如果设置了该参数，返回的子串数量不会多于 n；如果没有设置该参数，整个字符串都会被分隔。

split()

```
1 var str = "hello HTML hello CSS hello JavaScript";
2 var arr = str.split(' ');
3 console.log(arr);
```

运行结果

```
["hello", "HTML", "hello", "CSS", "hello", "JavaScript"]
```

12.3.7 翻转数组

使用 reverse() 可以颠倒数组中元素的顺序。

reverse()

```
1 var arr = [1, 2, 3, 4, 5];
2 console.log(arr.reverse());
```

运行结果

```
[5, 4, 3, 2, 1]
```

12.3.8 数组排序

使用 sort(sortfun) 可以将数组进行排序，其中参数 sortfun 可选，用于指定排序规则，而且必须是函数，该参数省略则按照字符编码顺序排序。

sort()

```
1 var arr = [98, 1, 21, 8, 12, 2, 10, 25];
2 arr.sort(function(a, b) {
3     return a > b ? 1 : -1;
4 });
5 console.log(arr);
```

运行结果

```
[1, 2, 8, 10, 12, 21, 25, 98]
```

12.3.9 数组切片

使用 slice(start, end) 可以返回数组中被选定的元素，不包含下标为 end 的元素。其中参数 start 必选，用于指定开始位置，如果是负数则从数组尾部开始算起。

参数 end 可选，用于指定结束位置，没有该参数省略，则切分到数组结束为止，如果是负数则从数组尾部开始算起。

slice()

```
1 var arr = [0, 1, 2, 3, 4, 5, 6];
2 console.log(arr.slice(2, 5));
```

运行结果

```
[2, 3, 4]
```

12.3.10 查找元素

使用 indexOf(item, start) 可以查找指定元素，如果查找成功则返回该元素的下标，如果查找失败者返回-1。其中参数 item 必选，用于指定需要查找的元素，参数 start 可选，用于指定在数组中开始检索的位置，如省略则从第一个元素开始检索。

indexOf()

```
1 var arr = [0, 1, 2, 3, 4, 5, 6];
2 console.log(arr.indexOf(4));
```

运行结果

```
4
```

Chapter 13 函数

13.1 函数

13.1.1 函数（Function）

函数执行一个特定的任务，JS 提供了大量内置函数，例如 alert() 用来显示警告对话框、parseInt() 用来将字符串转换为整型等。

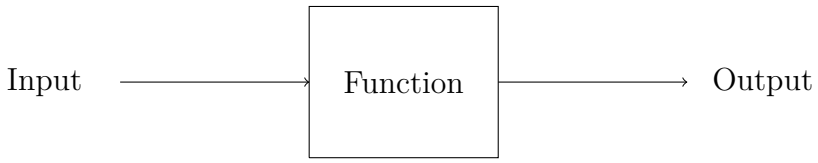


图 13.1: 函数

当调用函数时，程序控制权会转移给被调用的函数，当函数执行结束后，函数会把程序序控制权交还给其调用者。

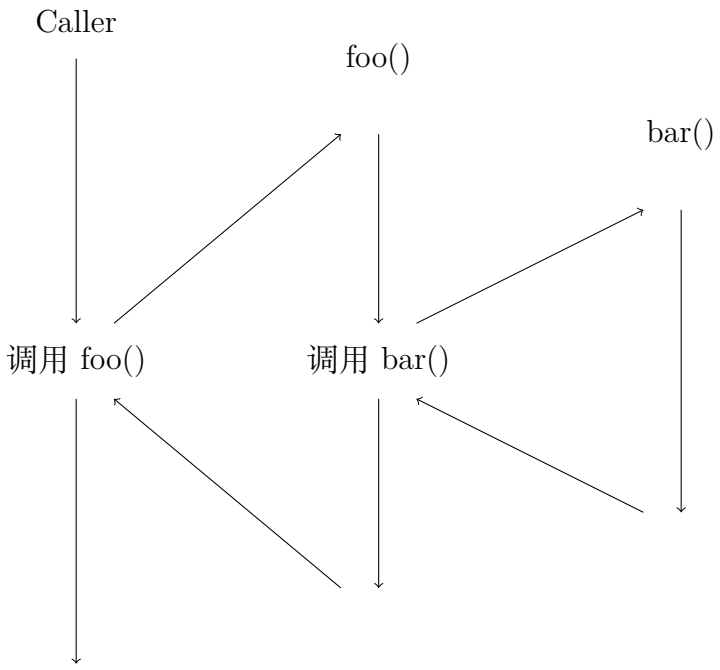


图 13.2: 函数调用

函数的定义需要使用关键字 function，函数的参数列表包括参数的类型、顺序、

数量等信息，参数列表可以为空。

```
function funcName(parameterList) {
    // code
}
```

13.1.2 函数设计方法

为什么不把所有的代码全部写在一起，还需要自定义函数呢?

使用函数有以下好处:

1. 避免代码复制，代码复制是程序质量不良的表现
2. 便于代码维护
3. 避免重复造轮子，提高开发效率

在设计函数的时候需要考虑以下的几点要素:

1. 确定函数的功能
2. 确定函数的参数
    - 是否需要参数
    - 参数个数
    - 参数类型
3. 确定函数的返回值
    - 是否需要返回值
    - 返回值类型

函数实现返回最大值

```
function max(num1, num2) {
    // if(num1 > num2) {
    //  return num1;
    // } else {
    //  return num2;
    // }

    return num1 > num2 ? num1 : num2;
}

console.log(max(4, 12));
console.log(max(54, 33));
console.log(max(0, -12));
console.log(max(-999, -774));
```

运行结果

```
12
54
0
-774
```

函数实现累加和

```
function sum(start, end) {
    var total = 0;
    for(var i = start; i <= end; i++) {
        total += i;
    }
    return total;
}

console.log("1-100的累加和 = " + sum(1, 100));
console.log("1024-2048的累加和 = " + sum(1024, 2048));
```

运行结果

```
1-100的累加和 = 5050
1024-2048的累加和 = 1574400
```

函数实现输出 i 行 j 列由自定义字符组成的图案

```
function print_chars(row, col, c) {
    var str = "";
    for(var i = 0; i < row; i++) {
        for(var j = 0; j < col; j++) {
            str += c;
        }
        str += "\n";
    }
    console.log(str);
}

print_chars(5, 10, '?');
```

运行结果

```
??????????
??????????
??????????
??????????
??????????
```

13.2 局部变量与全局变量

13.2.1 局部变量（Local Varaible）

JS 的局部变量是在函数里面被声明的，这些变量的作用域在本地，也就是说这些变量只能在函数内部可用。本地变量在函数调用时被创造，在函数结束时被销毁。

在函数中，函数的每次调用就会产生一个独立的空间，在这个空间中的变量，是函数的这次运行所独有的，函数的参数也是局部变量。

局部变量

```
1 function test(a) {
2     a = 2;
3     console.log("a = " + a);
4 }
5
6 var a = 1;
7 console.log("a = " + a);
8 test(a);
9 console.log("a = " + a);
```

运行结果

```
a = 1
a = 2
a = 1
```

13.2.2 全局变量（Global Varaible）

JS 的全局变量就是在函数外被声明的变量，作用域为全局，所有的在页面上的脚本和函数都可以获取这些变量。全局变量在其被声明时创建，在页面被关闭时被销毁。

全局变量的优先级低于局部变量，当全局变量与局部变量重名的时候，起作用的是局部变量，全局变量会被暂时屏蔽掉。

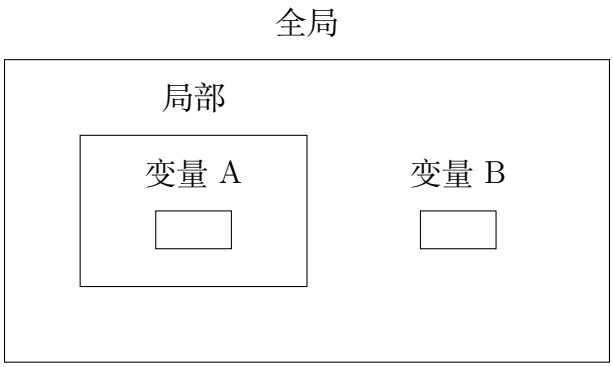


图 13.3: 全局变量

全局变量

```
1 var a = 1;          // 全局变量
2
3 function test() {
4     var a = 2;      // 本地变量
5     console.log("a = " + a);
6 }
7
8 test();
```

运行结果

```
a = 2
```

13.3 递归

13.3.1 递归（Recursion）

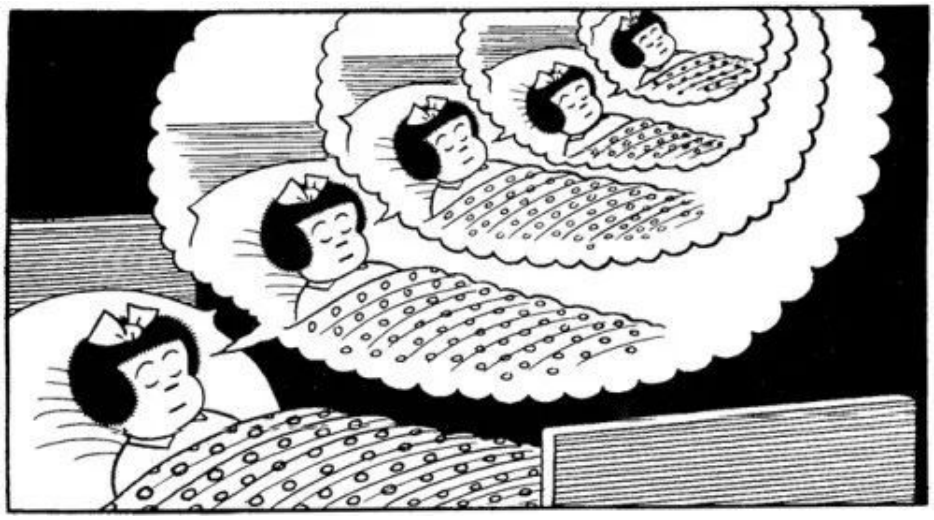
要理解递归，先得理解递归（见13.3章节）。

在函数的内部，直接或者间接的调用自己的过程就叫作递归。对于一些问题，使用递归可以简洁易懂的解决问题，但是递归的缺点是性能低，占用大量系统栈空间。

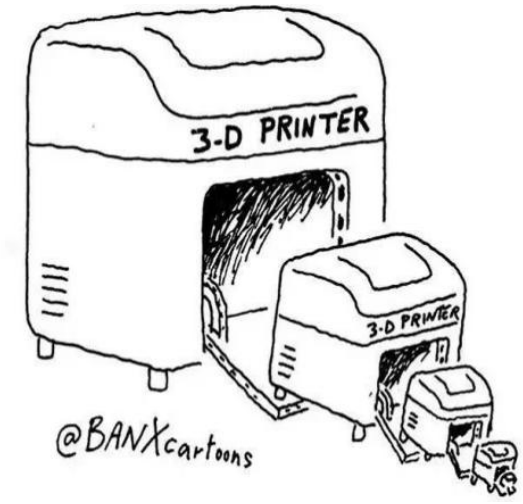
递归算法很多时候可以处理一些特别复杂、难以直接解决的问题。例如：

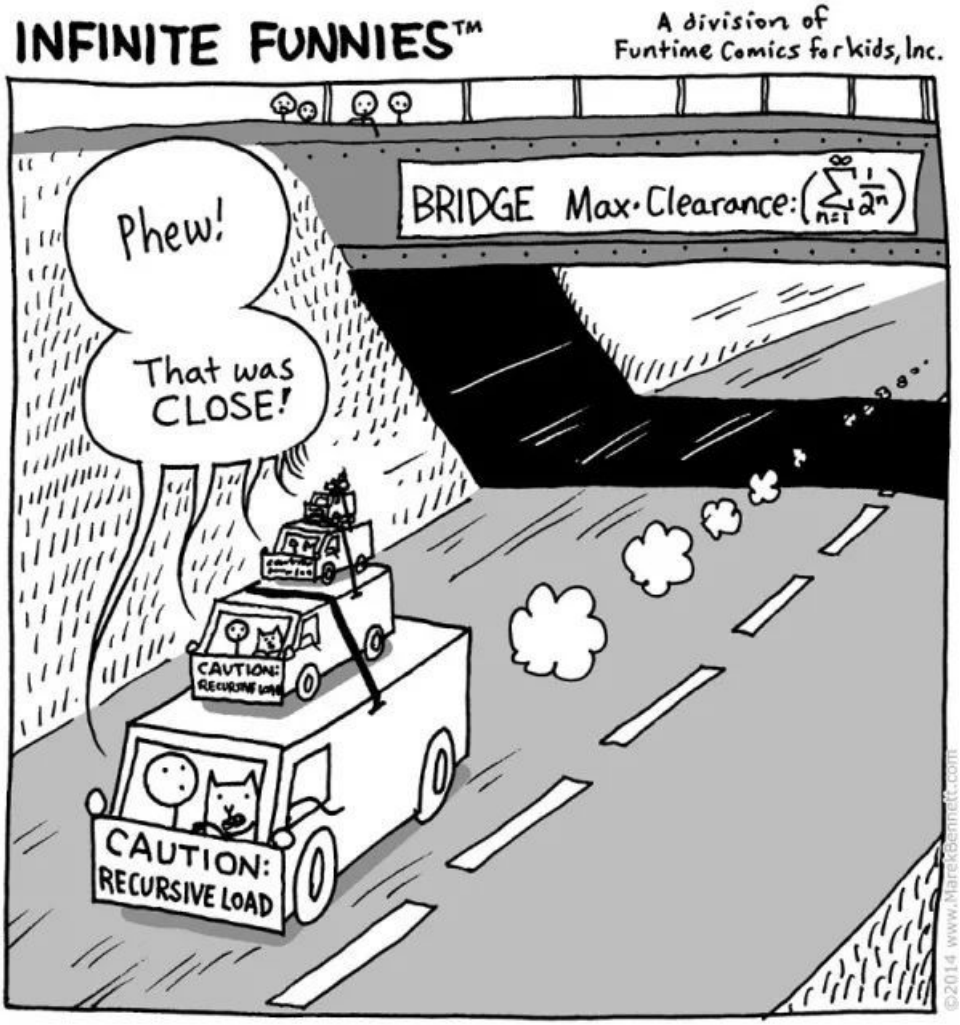
- 迷宫
- 汉诺塔
- 八皇后
- 排序
- 搜索

在定义递归函数时，一定要确定一个结束条件，否则会造成无限递归的情况，最终会导致栈溢出。




CN
COLLECTION


3-D PRINTER
3-D PRINTER
@BANXcartoons



无限递归

```
function tell_story() {
    console.log("从前有座山");
    console.log("山里有座庙");
    console.log("庙里有个老和尚和小和尚");
    console.log("老和尚对小和尚在讲故事");
    console.log("他讲的故事是：");
    tell_story();
}

tell_story();
```

运行结果

```
从前有座山
山里有座庙
庙里有个老和尚和小和尚
老和尚对小和尚在讲故事
他讲的故事是:
从前有座山
山里有座庙
庙里有个老和尚和小和尚
老和尚对小和尚在讲故事
他讲的故事是:
...
```

递归函数一般需要定义递归的出口，即结束条件，确保递归能够在适合的地方退出。

阶乘

```
1 function factorial(n) {
2     if(n == 0 || n == 1) {
3         return 1;
4     }
5     return n * factorial(n - 1);
6 }
7
8 console.log("5! = " + factorial(5));
```

运行结果

```
5! = 120
```

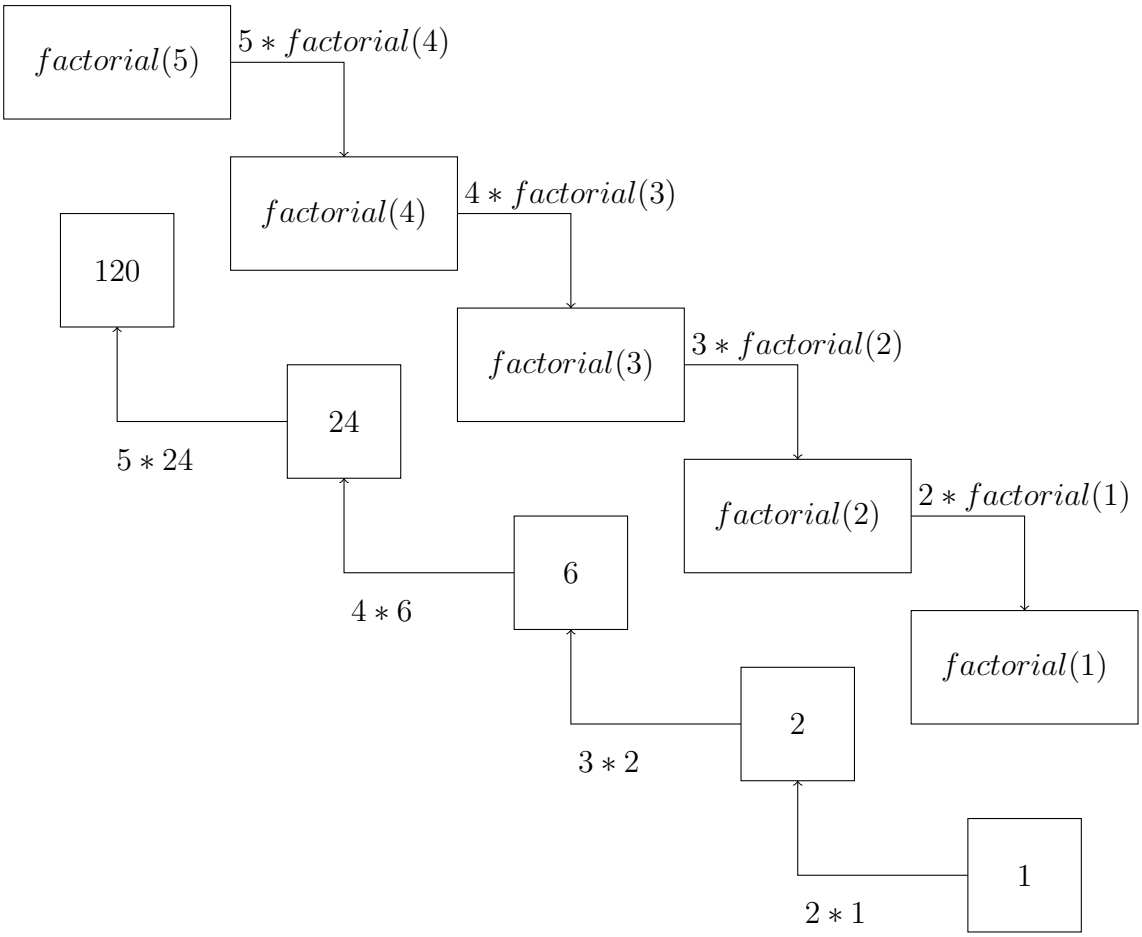


图 13.4: 阶乘

斐波那契数列（递归）

```
1  function fibonacci(n) {
2      if(n == 1 || n == 2) {
3          return 1;
4      }
5      return fibonacci(n - 2) + fibonacci(n - 1);
6  }
7
8  n = 7;
9  console.log("斐波那契数列第" + n + "位： " + fibonacci(7));
```

运行结果

斐波那契数列第7位： 13

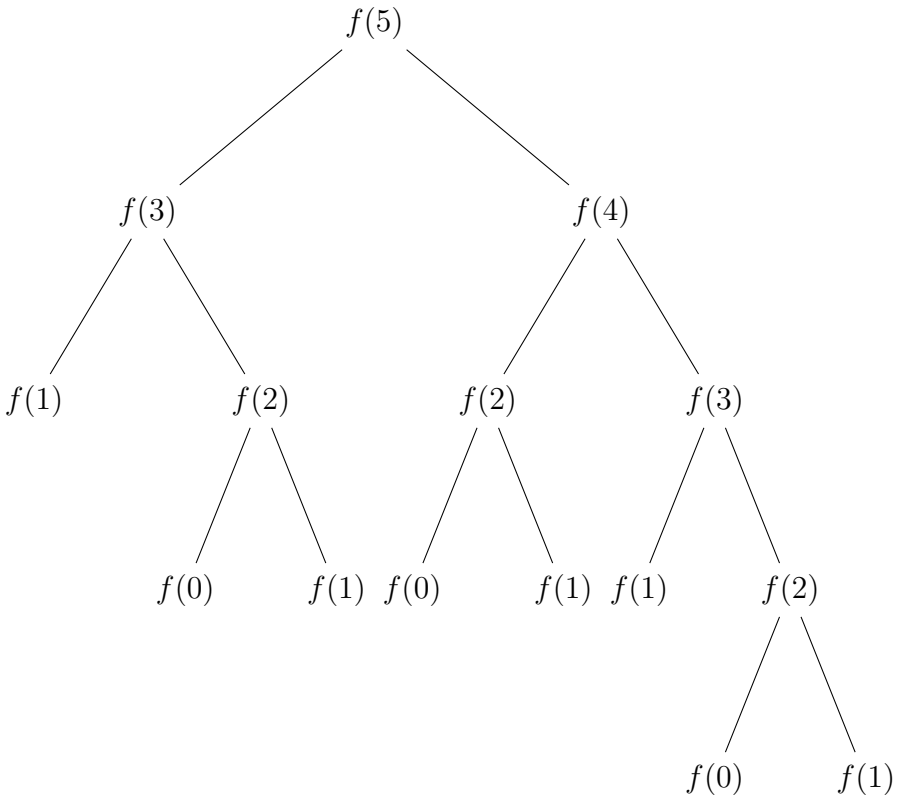


图 13.5: 递归树

斐波那契数列（迭代）

```
function fibonacci(n) {
    var f = new Array(n + 1);
    f[1] = f[2] = 1;
    for(var i = 3; i <= n; i++) {
        f[i] = f[i - 2] + f[i - 1];
    }
    return f[n];
}

n = 7;
console.log("斐波那契数列第" + n + "位： " + fibonacci(7));
```

运行结果

斐波那契数列第7位： 13

阿克曼函数

$$A(m,n) = \begin{cases} n+1 & m = 0 \\ A(m-1,1) & m > 0, n = 0 \\ A(m-1, A(m,n-1)) & m > 0, n > 0 \end{cases}$$

```
function A(m, n) {
    if(m == 0) {
        return n + 1;
    } else if(m > 0 && n == 0) {
        return A(m-1, 1);
    } else if(m > 0 && n > 0) {
        return A(m-1, A(m, n-1));
    }
}

console.log(A(3, 4));
```

运行结果

125

m \ n	0	1	2	3	4	n
0	1	2	3	4	5	$n+1$
1	2	3	4	5	6	$2+(n+3)-3$
2	3	5	7	9	11	$2(n+3)-3$
3	5	13	29	61	125	$2^{n+3}-3$
4	13	65533	$2^{65536}-3$	$A(3,2^{65536}-3)$	$A(3,A(4,3))$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3\ twos}-3$
5	65533	$A(4,65533)$	$A(4,A(5,1))$	$A(4,A(5,2))$	$A(4,A(5,3))$	...
6	$A(5,1)$	$A(5,A(5,1))$	$A(5,A(6,1))$	$A(5,A(6,2))$	$A(5,A(6,3))$	...

表 13.1: 阿克曼函数

汉诺塔

给定三根柱子，其中 A 柱子从大到小套有 n 个圆盘，问题是如何借助 B 柱子，将圆盘从 A 搬到 C。

规则：

- 一次只能搬动一个圆盘
- 不能将大圆盘放在小圆盘上面

递归算法求解汉诺塔问题：

1. 将前 n-1 个圆盘从 A 柱借助于 C 柱搬到 B 柱。
2. 将最后一个圆盘直接从 A 柱搬到 C 柱。
3. 将 n-1 个圆盘从 B 柱借助于 A 柱搬到 C 柱。

```
var move = 0;       // 移动次数

/**
 * @brief   汉诺塔算法
 * @note    把 n 个盘子从 src 借助 mid 移到 dst
 * @param   n: 层数
 * @param   src: 起点柱子
 * @param   mid: 临时柱子
 * @param   dst: 目标柱子
 */
function hanoi(n, src, mid, dst) {
    if(n == 1) {
        console.log(n + "号盘: " + src + " -> " + dst);
        move++;
    } else {
        // 把前 n-1 个盘子从 src 借助 dst 移到 mid
```

```
        hanoi(n-1, src, dst, mid);
        // 移动第 n 个盘子
        console.log(n + "号盘: " + src + " -> " + dst);
        move++;
        // 把刚才的 n-1 个盘子从 mid 借助 src 移到 dst
        hanoi(n-1, mid, src, dst);
    }
}

hanoi(4, 'A', 'B', 'C');
console.log("步数 ==> " + move);
```

运行结果

```
1号盘: A -> B
2号盘: A -> C
1号盘: B -> C
3号盘: A -> B
1号盘: C -> A
2号盘: C -> B
1号盘: A -> B
4号盘: A -> C
1号盘: B -> C
2号盘: B -> A
1号盘: C -> A
3号盘: B -> C
1号盘: A -> B
2号盘: A -> C
1号盘: B -> C
步数 ==> 15
```

Chapter 14 事件

14.1 事件

14.1.1 事件（Event）

JS 创建动态页面时，事件是可以被 JS 侦测到的行为。网页中的每个元素都可以产生某些可以出发 JS 函数或程序的事件。例如当用户单击或者提交表单数据时，就发生一个鼠标单击事件，需要浏览器做出处理，返回给用户一个结果。

事件	描述
onclick	鼠标单击事件
onmouseover	鼠标经过事件
onmouseout	鼠标移开事件
onchange	文本框内容改变事件
onselect	文本框内容被选中事件
onfocus	光标聚集
onblur	光标失焦
onload	网页导入
onunload	关闭网页

14.2 鼠标单击事件

14.2.1 鼠标单击事件 onclick

当在网页上单击鼠标时，就会发生该事件，同时 onclick 事件调用的程序块就会被执行，onclick 通常与按钮一起使用。

onclick

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>鼠标单击事件onclick</title>
    <script type="text/JavaScript">
        var cnt = 0;
        function feedback() {
            cnt++;
            console.log("我被点击了"+ cnt + "次");
        }
    </script>
</head>
<body>
    <form action="get/post">
        <input type="button" value="点击" onclick="feedback();">
    </form>
</body>
</html>
```

14.3 鼠标经过/移开事件

14.3.1 鼠标经过事件 onmouseover

当鼠标移到一个对象上时，该对象就会触发 onmouseover 事件，并执行 onmouseover 事件调用的程序。

onmouseover

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>鼠标经过事件onmouseover</title>
    <script type="text/JavaScript">
        var cnt = 0;
        function feedback() {
            alert("卸载之前再想想吧...");
        }
    </script>
</head>
<body>
    <form action="get/post">
        <input type="button" value="卸载"
            onmouseover="feedback();">
    </form>
</body>
</html>
```

14.3.2 鼠标移开事件 onmouseout

当鼠标移开当前对象时，执行 onmouseout 事件调用的程序。

onmouseout

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>鼠标移开事件onmouseout</title>
    <script type="text/JavaScript">
        var cnt = 0;
        function feedback() {
            alert("不要离开！只要输入密码，再点击登录就OK啦！");
        }
    </script>
</head>
<body>
    <form action="get/post">
        密码：<input type="password"
                onmouseout="feedback();">
        <input type="button" value="登录">
    </form>
</body>
</html>
```

14.4 光标聚焦/失焦事件

14.4.1 光标聚焦事件 onfocus

当网页中的对象获得聚点时，执行 onfocus 事件调用的程序。

onfocus

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>光标聚焦事件onfocus</title>
    <script type="text/JavaScript">
        var flag = true;
        function feedback() {
            if(flag) {
                alert("不要填错啦！ ");
                flag = false;
            }
        }
    </script>
</head>
<body>
    <form action="get/post">
        密码：<input type="password" onfocus="feedback();">
        <input type="button" value="登录">
    </form>
</body>
</html>
```

14.4.2 失焦事件 onblur

onblur 事件与 onfocus 事件是相对事件，当光标离开当前获得聚焦对象的时候，就会触发 onblur 事件，同时执行被调用的程序。

onblur

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>失焦事件onblur</title>
    <script type="text/JavaScript">
        function feedback() {
            alert("确定输对了再点登录哟！ ");
        }
    </script>
</head>
<body>
    <form action="get/post">
        密码：<input type="password" onblur="feedback();">
        <input type="button" value="登录">
    </form>
</body>
</html>
```

14.5 内容选中/改变事件

14.5.1 内容选中事件 onselect

当文本框或者文本域中的文本被选中时，触发 onselect 事件，同时调用的程序就会被执行。

onselect

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>内容选中事件onselect</title>
    <script type="text/JavaScript">
        function feedback() {
            console.log("文本内容被选中");
        }
    </script>
</head>
<body>
    <form action="get/post">
        <textarea rows="10" cols="30"
            onselect="feedback();">填写个人信息</textarea>
    </form>
</body>
</html>
```

14.5.2 内容改变事件 onchange

通过改变文本框的内容可以触发 onchange 事件，同时执行被调用的程序。

onchange

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>内容改变事件onchange</title>
    <script type="text/JavaScript">
        function feedback() {
            console.log("文本内容被修改");
        }
    </script>
</head>
<body>
    <form action="get/post">
        <textarea rows="10" cols="30"
            onchange="feedback();">填写个人信息</textarea>
    </form>
</body>
</html>
```

14.6 加载/卸载事件

14.6.1 加载事件 onload

加载事件会在页面加载完成后立即发生，同时执行被调用的程序。注意，加载事件需要写在 <body> 内。

onload

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>加载事件onload</title>
    <script type="text/JavaScript">
        function feedback() {
            alert("页面加载完成");
        }
    </script>
</head>
<body onload="feedback();">
    <p>Hello World!</p>
</body>
</html>
```

14.6.2 卸载事件 onunload

当用户退出页面时（页面关闭、页面刷新等），就会触发 onunload 事件，同时执行被调用的程序。注意，不同浏览器对 onunload 事件的支持不同。

Chapter 15 对象

15.1 对象

15.1.1 对象（Object）

JS 中的所有事物都是对象，如字符串、数值、数组、函数等，每个对象带有属性和方法。对象的属性反映了该对象的某些特定性质，如字符串的长度、图像的长宽等，对象的方法指的是能够在对象上执行的动作，如表单的提交、时间的获取等。

JS 提供了多个内建对象，如 String、Date、Array 等，使用对象前需要先使用 new 关键字进行定义。

访问对象属性

```
1 var arr = new Array(1, 2, 3, 4, 5);
2 console.log(arr.length);
```

运行结果

```
5
```

访问对象方法

```
1 var str = "Hello World";
2 console.log(str.toUpperCase());
```

运行结果

```
HELLO WORLD
```

15.2 Date

15.2.1 Date

Date 对象可以存储任意一个日期，并且可以精确到毫秒数（1/1000 秒）。使用默认构造函数创建的日期对象有初始值，为当前电脑系统时间。

定义 Date 对象

```
var date1 = new Date();
console.log(date1);

var date2 = new Date(2021, 3, 19);       //此处月份从0开始
console.log(date2);
```

运行结果

```
2021-03-19T05:09:47.713Z
2021-04-18T16:00:00.000Z
```

方法名称	描述
getDate() / setDate()	返回/设置日期
getFullYear() / setFullYear()	返回/设置年份，用四位数表示
getYear() / setYear()	返回/设置年份
getMonth() / setMonth()	返回/设置月份，月份从 0 开始
getHours() / setHours()	返回/设置小时，24 小时制
getMinutes() / setMinutes()	返回/设置分钟数
getSeconds() / setSeconds()	返回/设置秒钟数
getTime() / setTime()	返回/设置时间（毫秒为单位）
getDay()	返回 0-6 的数字表示星期，0 表示星期天

表 15.1: Date 方法

获取今日星期

```
var date = new Date();
var weekday = [
    "星期天",
    "星期一",
    "星期二",
    "星期三",
    "星期四",
    "星期五",
    "星期六"
];
console.log("今天是" + weekday[date.getDay()]);
```

运行结果

今天是星期五

15.3 String

15.3.1 计算字符串长度

定义 String 对象后就可以访问它的属性和方法。

计算字符串长度

```
1 var str = "Hello World!"
2 console.log(str.length);
```

运行结果

```
12
```

15.3.2 大小写转换

使用 String 对象的 toUpperCase() 和 toLowerCase() 可以将字符串进行大小写字母转换。

大小写字母转换

```
1 var str = "Hello World!"
2 console.log(str.toUpperCase());
3 console.log(str.toLowerCase());
```

运行结果

```
HELLO WORLD!
hello world!
```

15.3.3 返回指定的字符

使用 charAt() 可返回指定位置的字符，返回的字符是长度为 1 的字符串。

charAt()

```
var str = "Hello World!"
console.log(str.charAt(6));
```

运行结果

```
W
```

15.3.4 返回指定的字符串首次出现的位置

indexOf() 可以返回某个指定的字符串值在字符串中首次出现的位置。

```
stringObj.indexOf(substring, startPos);
```

该方法将从头到尾地检索字符串，检查是否含有需检索的子串。参数 startPos 为可选参数，用于规定开始查找的位置，如果没有设置此参数将从头开始查找。如果找到了子串，在返回子串的第一次出现位置。如果要检索的字符串值没有出现，则该方法返回-1。

indexOf()

```
var str = "Hello World!";
console.log(str.indexOf("Hell"));
console.log(str.indexOf("o", 6));
console.log(str.indexOf("JS"));
```

运行结果

```
0
7
-1
```

15.3.5 字符串分割

split() 可以将字符串分割为字符串数组，并返回此数组。

```
stringObj.split(separator, limit);
```

其中参数 separator 必选，用于指定将字符串按某个字符切割成若干个子字符串，并以数组的形式返回。参数 limit 可选，用于指定分隔的次数。如果把空字符串作为 separator，那么字符串的每个字符之间都会被分隔。

split()

```
var str = "hello HTML hello CSS hello JavaScript";
console.log(str.split(" "));
console.log(str.split(" ", 4));
```

运行结果

```
["hello", "HTML", "hello", "CSS", "hello", "JavaScript"]
["hello", "HTML", "hello", "CSS"]
```

15.3.6 提取字符串

substring() 用于提取字符串中介于两个指定下标之间的字符。

```
stringObj.substring(startPos, stopPos);
```

该方法返回的内容是从 startPos 开始到 stopPos - 1 处的所有内容，其长度为 stopPos - startPos。如果 startPos 和 stopPos 相等，那么返回的就是一个空串

（长度为 0 的字符串）。如果 startPos 比 stopPos 大，那么该方法在提取子串之间会先交换这两个参数。

substring()

```
var str = "HelloWorld";
console.log(str.substring(2, 8));
console.log(str.substring(3, 3));
console.log(str.substring(7, 3));
```

运行结果

```
lloWor


loWo
```

15.3.7 提取指定数目的字符

substr() 用于从字符串中提取从 startPos 位置开始的指定数目的字符串。

```
stringObj.substr(startPos, length);
```

如果参数 startPos 是负数，则从字符串的尾部开始算起，如果 startPos 为负数且绝对值大于字符串长度，则 startPos 会被视为 0。

substr()

```
var str = "HelloWorld";
console.log(str.substr(2, 3));
console.log(str.substr(-5, 4));
```

运行结果

```
llo
Worl
```

15.4 Math

15.4.1 Math

Math 对象提供对数据的数学计算。需要注意的是，Math 对象是一个固有的对象，无需创建它，直接把 Math 作为对象使用就可以调用其所有属性和方法，这是它与其它对象的区别。

属性	描述
E	返回算术常量 e，即自然对数的底数（约等于 2.718）
LN2	返回 2 的自然对数（约等于 0.693）
LN10	返回 10 的自然对数（约等于 2.302）
LOG2E	返回以 2 为底 e 的对数（约等于 1.442）
LOG10E	返回以 10 为底 e 的对数（约等于 0.434）
PI	返回圆周率（约等于 3.14159）
SQRT1_2	返回 2 的平方根的倒数（约等于 0.707）
SQRT2	返回 2 的平方根（约等于 1.414）

表 15.2: Math 属性

方法	描述
sin(x)	返回 x 的正弦
cos(x)	返回 x 的余弦
tan(x)	返回 x 的正切
acos(x)	返回 x 的反余弦值
asin(x)	返回 x 的反正弦值
atan(x)	返回 x 的反正切值
ceil(x)	对 x 进行上取整
floor(x)	对 x 进行下取整
abs(x)	返回 x 的绝对值
exp(x)	返回 e 的 x 次幂
log(x)	返回 x 的自然对数（底为 e）
pow(x, y)	返回 x 的 y 次幂
max(x, y)	返回 x 和 y 中的最大值
min(x, y)	返回 x 和 y 中的最小值
round(x)	返回 x 的四舍五入最接近的整数
sqrt(x)	返回 x 的平方根
random()	返回 $0 \sim 1$ 之间的随机数

表 15.3: Math 方法

Chapter 16 浏览器对象模型 BOM

16.1 window 对象

16.1.1 window 对象

window 对象是浏览器对象模型 BOM（Browser Object Model）的核心。

方法	描述
alert()	显示带有一段消息和一个确认按钮的警告框
prompt()	显示可提示用户输入的对话框
confirm()	显示带有一段消息以及确认按钮和取消按钮的对话框
open()	打开一个新的浏览器窗口或查找一个已命名的窗口
close()	关闭浏览器窗口
print()	打印当前窗口的内容
focus()	把焦点给与一个窗口
blur()	把焦点从顶层窗口移开
moveBy()	可相对窗口的当前坐标把它移动指定的像素
moveTo()	把窗口的左上角移动到一个指定的坐标
resizeTo()	把窗口的大小调整到指定的宽度和高度
scrollBy()	按照指定的像素值来滚动内容
scrollTo()	把内容滚动到指定的坐标

表 16.1: window 方法

16.2 计时器

16.2.1 计时器

在 JS 中，可以在设定的时间间隔之后执行代码，而不是在函数被调用后立即执行。

计时器的类型分为 2 种：

1. 一次性计时器：仅在指定的延迟时间之后触发一次。
2. 间隔性触发计时器：每隔一定的时间间隔就触发一次。

方法	描述
setTimeout()	在指定的延迟时间之后来执行代码
clearTimeout()	取消 setTimeout() 的设置
setInterval()	每隔指定的时间执行代码
clearInterval()	取消 setInterval() 的设置

表 16.2: 计时器方法

16.2.2 setTimeout()

setTimeout() 计时器，在载入后延迟指定时间后，去执行一次表达式，仅执行一次。

```
1 setTimeout(expr, timeout);
```

- expr：要调用的函数或要执行的代码串。
- timeout：在执行代码前需等待的时间，以毫秒为单位（1s = 1000ms）。

```
1 // 网页打开2秒后弹出提示框
2 setTimeout("alert('Welcome')", 2000);
```

clearTimeout() 和 setTimeout() 一起使用，用于停止计时器。

```
clearTimeout(id_of_setTimeout);
```

id_of_setTimeout：setTimeout() 返回的 ID 值，该值标识要取消的延迟执行代码块。

计数器

counter.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>计数器</title>
    <script src="counter.js"></script>
</head>
<body>
    <form action="get/post">
        <input type="text" id="num">
        <input type="button" value="stop" onclick="stopCount();">
    </form>
</body>
</html>
```

counter.js

```
var cnt = 0;        // 计数
var counter;        // 计数器

/**
 * 每隔1000毫秒计数加1
 */
function count() {
    document.getElementById('num').value = cnt;
    cnt++;
    counter = setTimeout(count, 1000);
}

/**
```

```
 * 停止计数器
 */
function stopCount() {
    clearTimeout(counter);
}

setTimeout(count, 1000);        // 启动计数器
```

16.2.3 setInterval()

setInterval() 在执行时，从载入页面后每隔指定的时间执行代码。

```
setInterval(expr, interval);
```

- expr：要调用的函数或要执行的代码串。
- interval：周期性执行或调用表达式之间的时间间隔，以毫秒为单位（1s = 1000ms）。

clearInterval() 可取消由 setInterval() 设置的交互时间。

```
clearInterval(id_of_setInterval);
```

id_of_setInterval：setInterval() 返回的 ID 值。

实时显示当前时间

current_time.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>显示当前时间</title>
    <script src="current_time.js"></script>
</head>
<body>
```

```
    <form action="get/post">
        <input type="text" id="time" size="50">
    </form>
</body>
</html>
```

current_time.js

```
// function clock() {
//     var date = new Date();
//     document.getElementById("time").value = date;
// }

// setInterval(clock, 1000);

// 箭头函数
setInterval(() => {
    var date = new Date();
    document.getElementById("time").value = date;
}, 1000);
```

16.3 Screen 对象

16.3.1 Screen 对象

Screen 对象用于获取用户的屏幕信息。window.screen 对象在编写时可以不使用 window 前缀。

属性	描述
availHeight	窗口可以使用的屏幕高度，单位像素
availWidth	窗口可以使用的屏幕宽度，单位像素
height	屏幕的高度，单位像素
width	屏幕的宽度，单位像素

表 16.3: Screen 属性

screen.availWidth 和 screen.availHeight 属性返回访问者屏幕的宽度和高度，单位为像素，减去界面特性，比如任务栏等。不同系统的任务栏默认高度不一样，及任务栏的位置可在屏幕上下左右任何位置，所以有可能可用宽度和高度不一样。

屏幕信息

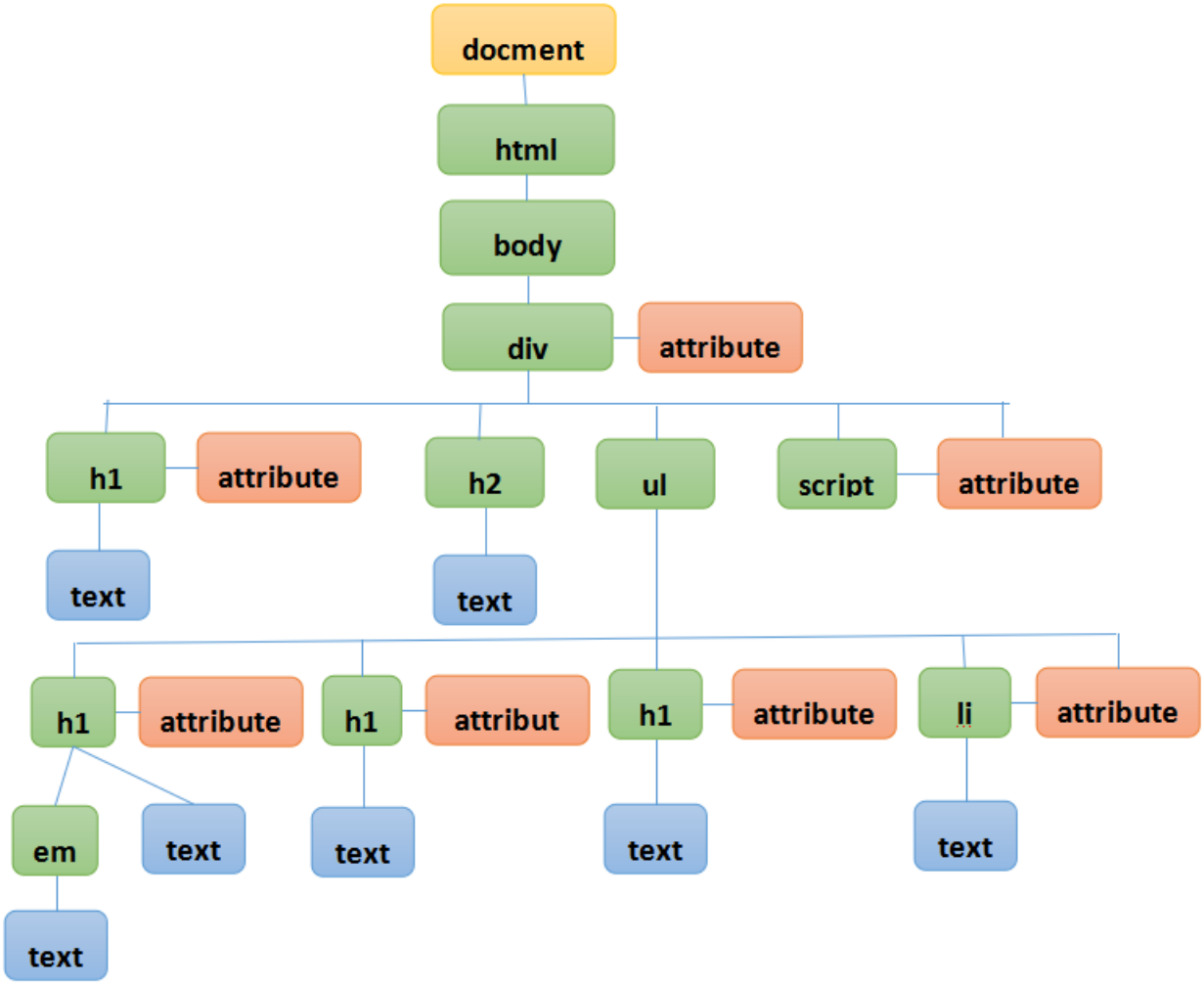
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>屏幕信息</title>
    <script type="text/JavaScript">
        console.log("屏幕分辨率： "
                    + screen.width + "*"
                    + screen.height)
        console.log("屏幕可用宽高： "
                    + screen.availWidth + "*"
                    + screen.availHeight);
    </script>
</head>
<body>
```

```
16
17 </body>
18 </html>
```

Chapter 17 文档对象模型 DOM

17.1 DOM

17.1.1 DOM (Document Object Model)



17.2 获取结点对象

17.2.1 getElementById()

getElementById() 可返回拥有指定 ID 的第一个对象的引用。如果没有指定 ID 的元素则返回 null，如果存在多个指定 ID 的元素则返回第一个。

定时变换颜色

random_color.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>随机颜色</title>
    <script src="random_color.js"></script>
</head>
<body>
    <div id="square" style="width: 100px; height: 100px"></div>
</body>
</html>
```

random_color.js

```
/**
 * 随机生成RGB颜色代码
 * @returns rgb颜色
 */
function randomRGB() {
    var r = Math.floor(Math.random() * 256);
    var g = Math.floor(Math.random() * 256);
    var b = Math.floor(Math.random() * 256);
    return "rgb(" + r + ", " + g + ", " + b + ")";
}

/**
 * 获取元素结点，设置背景颜色
 */
```

```
function changeColor() {
    var obj = document.getElementById("square");
    obj.style.background = randomRGB();
}

// 每隔300ms改变颜色
setInterval(function () {
    changeColor();
}, 300);
```

17.2.2 getElementsByClassName()

getElementByClassName() 返回文档中所有指定类名的元素集合，作为 NodeList 对象。NodeList 对象代表一个有顺序的结点列表，可以通过索引来访问列表中的结点。使用 NodeList 的 length 属性可以确定指定类名的元素个数，并循环各个元素来获取某个元素。

```
document.getElementsByClassName(className);
```

17.2.3 getElementByName()

getElementByName() 返回带有指定名称的结点对象的集合。

```
document.getElementsByName(name);
```

getElementByName() 通过元素 name 属性查询元素，文档中的 name 属性可能不唯一，所以 getElementByName() 返回的是元素的数组，而不是一个元素。

17.2.4 getElementsByTagName()

getElementsByTagName() 返回带有指定标签名的结点对象的集合，返回元素的顺序是它们在文档中的顺序。

```
document.getElementsByTagName(tagName);
```

17.3 结点操作

17.3.1 结点属性

getAttribute() 可以通过元素结点的属性名称获取属性的值。

```
elementNode.getAttribute(name);
```

其中，elementNode 可以使用 getElementById()、getElementsByTagName() 等方法获取到元素结点，参数 name 为需要查询的元素结点的属性名称。

setAttribute() 可以增加一个指定名称和值的新属性，或者把一个现有的属性设定为指定的值。

```
elementNode.setAttribute(name, value);
```

设置结点属性值

getAttribute.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>设置结点属性</title>
    <script src="getAttribute.js"></script>
</head>
<body>
    <a class="link" href="https://www.baidu.com">百度</a>
    <a class="link" href="https://www.bilibili.com">哔哩哔哩</a>
</body>
</html>
```

getAttribute.js

```
window.onload = function() {
    var links = document.getElementsByClassName("link");
```

```
3     for(var i = 0; i < links.length; i++) {
4         console.log(links[i].getAttribute("href"));
5         links[i].setAttribute("target", "_blank");
6     }
7 };
```

运行结果

```
https://www.baidu.com
https://www.bilibili.com
```

17.3.2 结点操作

createElement() 用于创建结点元素，此方法可返回一个 Element 对象。

createElement() 要与 appendChild() 或 insertBefore() 联合使用，将元素显示在页面中。

```
1 document.createElement(tagName);
```

appendChild() 用于在指定结点的最后一个子结点列表之后添加一个新的子结点。

```
1 elementNode.appendChild(newNode);
```

insertBefore() 用于在已有的子结点前插入一个新的子结点。

```
1 elementNode.insertBefore(newNode, node);
```

removeChild() 用于从子结点列表中删除某个结点，如删除成功返回被删除的结点，如失败则返回 null。

```
1 elementNode.removeChild(node);
```

replaceChild()：实现子结点的替换，返回被替换对象的引用。当 oldNode 被替换时，所有与之相关的属性内容都将被移出。

```
1 elementNode.replaceChild(newNode, oldNode);
```