# 第3章
# 整体布局(上)——标签尺寸处理

## 3.1 整体布局与整体布局中使用的标签

### 3.1.1 整体布局

了解基本规则,做好前期开发准备之后,就要真正进入“代码书写”阶段了。网页整体布局,是整个网站具体代码实现时的第一步,该步骤是针对网页进行分析,划分为多个模块,将一个网页解读成几个组成部分,并针对这些组成部分进行位置、大小的控制。第 3 章和第 4 章所涉及的具体知识如图 3.1 所示。

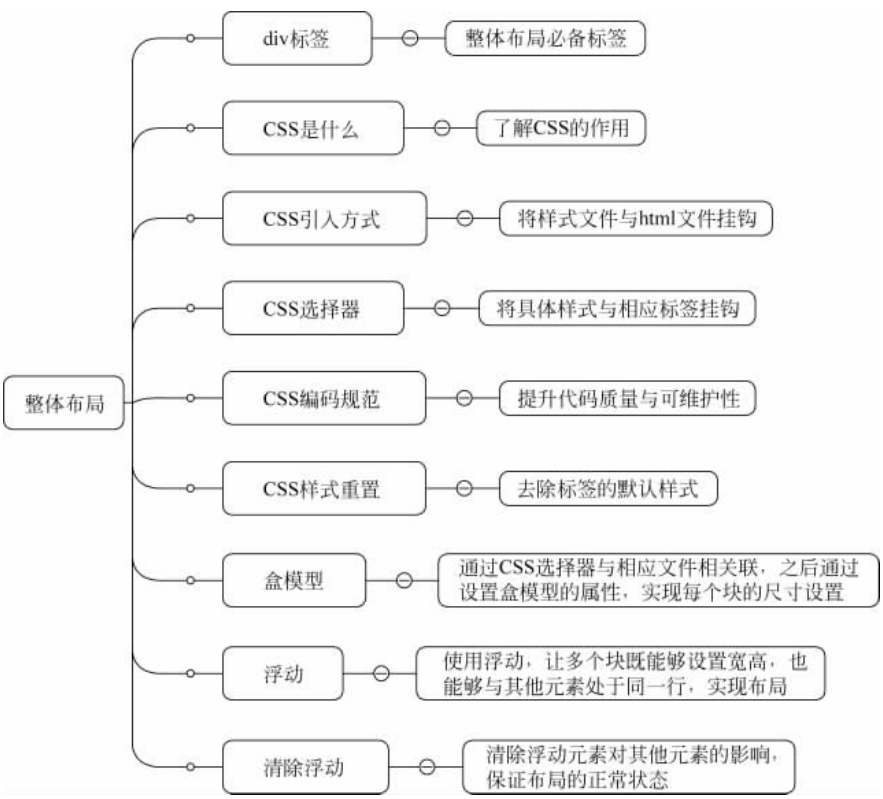



图 3.1 第 3 章和第 4 章涉及的知识

在整体布局的过程中，首先会接触并使用到< body >当中的第一种 HTML 元素——div。这种元素也是在整体布局当中会使用到的唯一的 HTML 元素。

然后要涉及关于 CSS 布局方面的一些知识，让一个个 div 能够按照开发工程师的需求和想法，呈现出相应的大小，出现在网页中适当的位置上，如图 3.2 所示。

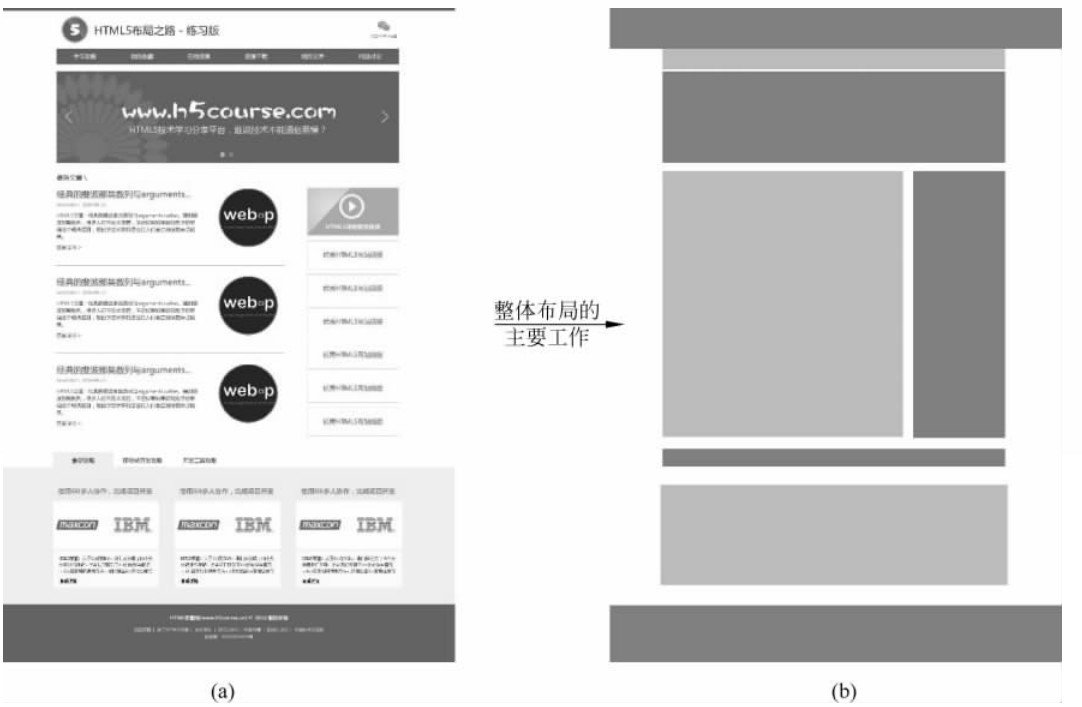



**图 3.2　整体布局的主要工作**

**注**：图 3.2(a)为网页的 PSD 设计图，图 3.2(b)为整体布局完成后，HTML 文件在浏览器中的显示效果。

### 3.1.2　div 元素

**标签含义：**

div 元素没有任何具体含义，在网站开发当中，主要用于网页的布局工作。通过一个一个的 div，将网站划分为不同的部分，之后再使用其他元素针对每个具体部分进行开发。

div 元素是一个块元素，默认情况下占据父级元素的宽度，由内容撑开高度；可以设置宽度和高度，设置后，依旧独自占据父元素的一行。

**注意**：关于块元素、行元素的概念，会在第 5 章中详细讲解，此处请先掌握 div 的呈现特点。

**代码实例：**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - div 元素实例</title>
    <style>
```

```
        /* 可以通过如下代码查看到每个 div 的边界 */
        div {
            border: 5px solid black;        /* 5 像素黑色实线边框 */
        }
    </style>
</head>
<body>
    <div>div 元素中的内容,该内容会显示在浏览器中</div>
    <div>
        <div>嵌套在父级 div 中的 div 元素 1</div>
        <div>嵌套在父级 div 中的 div 元素 2</div>
    </div>
</body>
</html>
```

## 3.2　什么是 CSS



### 3.2.1　没有 CSS 时代的网页

HTML 比 CSS 早几年诞生,在最初还没有 CSS 的年代,网页的样式显示如图 3.3 所示。

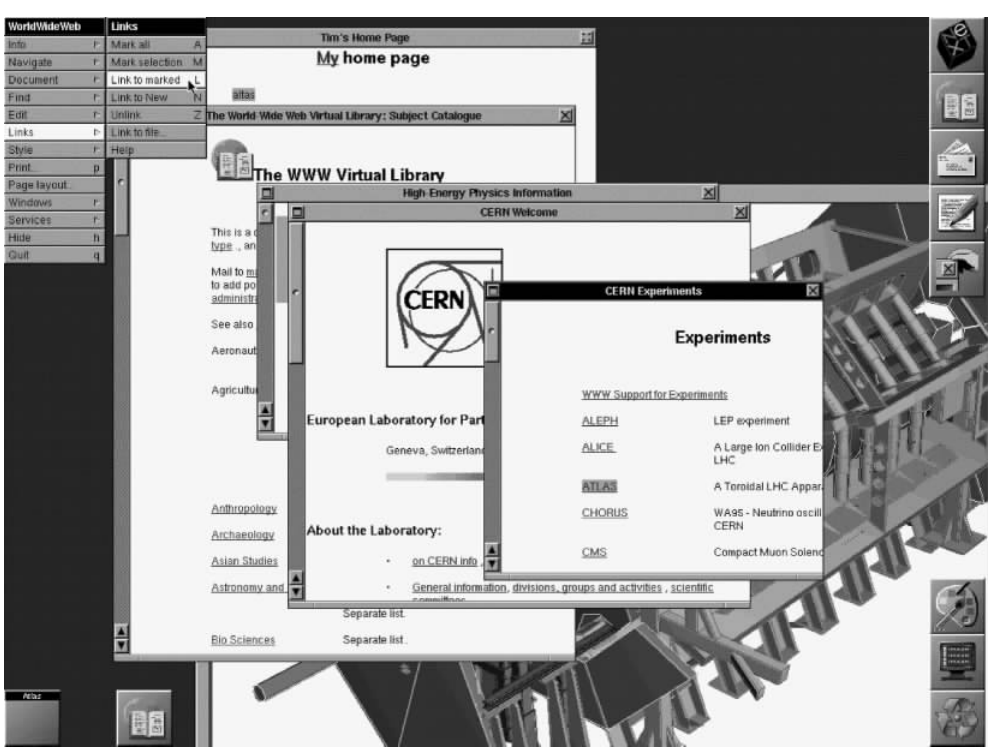



图 3.3　没有 CSS 时代的网页

在当前可以浏览的网站当中，都存在着 CSS 样式代码，可以通过谷歌等浏览器的控制台，去除网页中的 CSS 代码，来查看没有 CSS 时代网页的显示效果。其中，图 3.4 为存在 CSS 样式代码时，HTML5 学堂的页面效果，图 3.5 为去掉 CSS 样式代码之后，HTML5 学堂的页面效果。


图 3.4　存在 CSS 样式时，网页的页面效果

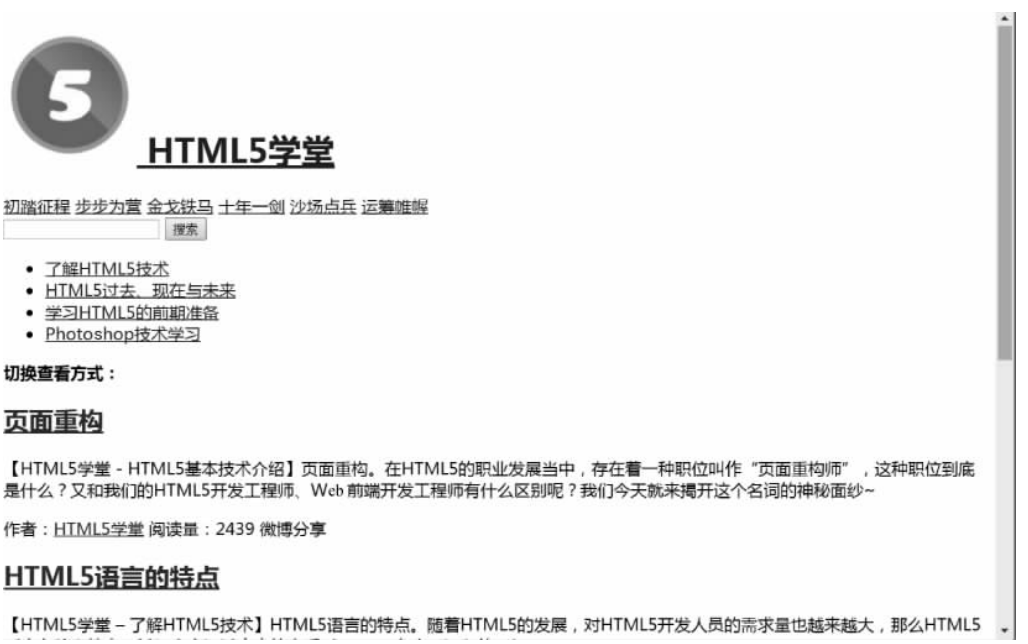



图 3.5　去掉 CSS 样式后，网页的页面效果

### 3.2.2　什么是 CSS

CSS(Cascading Style Sheets),可译为层叠样式表或级联样式表,是一组格式设置规则,主要用于控制 Web 页面的外观。通过使用 CSS 样式设置页面的风格,可将页面的内容与表现形式分离。

## 3.3　CSS 引入方式

### 3.3.1　行内书写——最简单的样式操作方法

#### 1. 最简单样式的操作方法

一个网页中存在结构标签,也存在具体的样式,只有“某些结构标签”应用“某种具体样式”时样式才能够生效,那么如何实现这个关联过程呢?

将 CSS 样式与 HTML 结构“关联”起来的方法有多种,一种最为原始也是最容易理解,看上去似乎是最为简单的书写方式,是直接将“样式”作为标签的“属性”(style)来书写。



**基本语法:**

```
<标签名 style = "具体样式属性: 属性值">内容</标签名>
```

**代码实例:**

```
<div style = "border: 1px solid red"> 1 </div>
```

**代码解析:**

style 是 div 标签的属性,表示的是“样式”。

style 中的 border 指的是边框这种样式,而“1px solid red”表示的是边框样式为 1 像素、实线、红色。

**备注:** 关于边框部分,在后面的盒模型部分当中会做详细讲解,因此先不要急,当前只需要了解标签名、style 属性以及具体的属性名、属性值的关系即可。

#### 2. 实际案例解析

下面是一个实际案例:

在网页(HTML 文件)中存在 10 个 div 标签,前 5 个 div 元素添加“1 像素红色实线边框”的样式,后 5 个 div 添加“1 像素蓝色实线边框”的样式。

**代码实例:**

```
<!doctype html>
<html>
<head>
    <meta charset = "UTF - 8">
    <title> HTML5 布局之路 - 行内书写 CSS </title>
</head>
<body>
```

```
    <div style="border: 1px solid red">1</div>
    <div style="border: 1px solid red">2</div>
    <div style="border: 1px solid red">3</div>
    <div style="border: 1px solid red">4</div>
    <div style="border: 1px solid red">5</div>
    <div style="border: 1px solid blue">6</div>
    <div style="border: 1px solid blue">7</div>
    <div style="border: 1px solid blue">8</div>
    <div style="border: 1px solid blue">9</div>
    <div style="border: 1px solid blue">10</div>
</body>
</html>
```

### 3. 客户需求带来的麻烦事

将制作好的网页提交给客户，客户查看网页效果时，感觉红色边框并不好看，提出了修改的要求："能不能将所有的红色边框全部改为绿色?"于是，修改代码如下。



```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 行内书写 CSS</title>
</head>
<body>
    <div style="border: 1px solid green">1</div>
    <div style="border: 1px solid green">2</div>
    <div style="border: 1px solid green">3</div>
    <div style="border: 1px solid green">4</div>
    <div style="border: 1px solid green">5</div>
    <div style="border: 1px solid blue">6</div>
    <div style="border: 1px solid blue">7</div>
    <div style="border: 1px solid blue">8</div>
    <div style="border: 1px solid blue">9</div>
    <div style="border: 1px solid blue">10</div>
</body>
</html>
```

将修改后的网页再次提交给客户，客户觉得："绿色的边框的确比较好看，能不能将所有的蓝色边框也调整为绿色?"

于是，继续调整代码：

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 行内书写 CSS</title>
</head>
<body>
    <div style="border: 1px solid green">1</div>
    <div style="border: 1px solid green">2</div>
```

```
    <div style = "border: 1px solid green"> 3 </div>
    <div style = "border: 1px solid green"> 4 </div>
    <div style = "border: 1px solid green"> 5 </div>
    <div style = "border: 1px solid green"> 6 </div>
    <div style = "border: 1px solid green"> 7 </div>
    <div style = "border: 1px solid green"> 8 </div>
    <div style = "border: 1px solid green"> 9 </div>
    <div style = "border: 1px solid green"> 10 </div>
</body>
</html>
```

当再次修改代码之后,客户查看着绿色的边框说:“或许把边框都变成黑色更好一些。”再一次修改代码:

```
<!doctype html>
<html>
<head>
    <meta charset = "UTF - 8">
    <title> HTML5 布局之路 - 行内书写 CSS </title>
</head>
<body>
    <div style = "border: 1px solid black"> 1 </div>
    <div style = "border: 1px solid black"> 2 </div>
    <div style = "border: 1px solid black"> 3 </div>
    <div style = "border: 1px solid black"> 4 </div>
    <div style = "border: 1px solid black"> 5 </div>
    <div style = "border: 1px solid black"> 6 </div>
    <div style = "border: 1px solid black"> 7 </div>
    <div style = "border: 1px solid black"> 8 </div>
    <div style = "border: 1px solid black"> 9 </div>
    <div style = "border: 1px solid black"> 10 </div>
</body>
</html>
```

客户看过修改后的代码说:“看了这么多种颜色,我还是觉得没有边框最好。”最后,之前书写、修改的样式代码都被删掉了,之前的工作似乎是在消磨时间,作为开发工程师这样的遭遇又何止一次……

### 3.3.2　内部书写——简化样式操作

#### 1. 简化样式操作的“内部书写”

经过几次需求更改,作为开发工程师的你,有没有感觉到“style 属性”的书写方式产生了大量重复代码?一旦涉及修改,操作起来也异常麻烦?

发明 CSS 的人与我们有着类似的感受,于是就提供了另外两种方法,来解决如此麻烦的问题。

**代码实例:**

```
<!doctype html>
<html>
```

```
<head>
    <meta charset = "UTF - 8">
    <title>HTML5 布局之路 - 内部书写 CSS 样式</title>
    <style>
        div {
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <div>1 </div>
    <div>2 </div>
    <div>3 </div>
    <div>4 </div>
    <div>5 </div>
    <div>6 </div>
    <div>7 </div>
    <div>8 </div>
    <div>9 </div>
    <div>10 </div>
</body>
</html>
```



在这段代码中，将 CSS 的属性、属性值放在了一个< style ></style >标签当中，这种书写方法叫作 CSS 样式的“内部书写”。

**代码解析：**

< style ></style >标签当中，div 表示选择器，这个选择器能够找到的标签就是 body 元素中的所有 div 元素；大括号{}包含具体的样式，这些样式将应用于选择器选择到的所有标签；“border：1px solid black;”表示的是具体样式代码。

**2.“内部书写”方法的原理与优势**

原理：将样式代码统一放置于 style 标签当中，然后通过“选择器”，将规定的样式和相应的标签建立联系。

优势：使用“内部书写”的方法来操作样式时更加方便快捷。在进行 CSS 样式修改时，并不需要翻阅 HTML 代码，也不需要去针对每个 HTML 标签处理样式，只需要调整 style 标签中的样式，就可以针对所有相应标签进行样式的修改。

**3. 你可能在思考的问题**

可能你会考虑：这种改变让所有的 div 都发生样式变化，但是最初的需求是一半红色一半蓝色，这种书写方式无法满足之前的需求。

如果能够想到这个问题，说明你更深入地进行了思考。不过，先把这个问题放一放，本章稍后会讲解各种各样的选择器，通过各种选择器，可以很灵活地选择想要的标签，自然能够轻松地解决这个问题了。

### 3.3.3 外部引入——控制多页面样式

**1. 内部书写在网站中的不足**

掌握在 HTML 文件中书写 style 标签进行相应元素的样式控制之后，还要面对这样的

场景：一个网站当中拥有 10 个甚至更多的 HTML 页面，在这 10 个 HTML 页面中，必然会拥有相同的部分，例如，网页的头部、底部、导航区、某些模块等。对于开发工程师来说，如果希望能够一次性控制这些不同 HTML 文件中相同模块的样式，内部书写的方法就无法实现这个需求了。

### 2. link 标签

link 标签的作用：将外部样式表文件链接到 HTML 文档中。

link 较完整写法：< link rel="stylesheet" type="text/css" href="css/index. css">

(1) href：定义外部样式表(CSS)文件的地址，该属性值可以使用 URL 地址的格式进行设置，可以使用相对路径或绝对路径。

(2) type：定义链接文档的类型。type="text/css"表示当前标签包含内容的文本类型为 CSS 样式代码。

(3) rel、rev：定义链接外部文件的关联类型，也就是设置文件默认应用的是哪种类型的链接文件。rel="stylesheet"表示浏览器在默认状态下应该先执行样式表的文件类型。

(4) 除此之外，link 还包含 title、media 等属性，由于较少在开发中使用，在此不再讲解。

### 3. 字符编码

外部引入的 CSS 文件，需要进行字符编码的设置：

```
@charset"UTF - 8";
```

@charset 命令用于定义外部引入的 CSS 文件的字符编码，该命令需要在 CSS 文件中的最前面定义，且在当前的 CSS 文件当中只允许出现一次。如果开发工程师没有为 CSS 文件定义编码格式，这个文件将按照被引入到的 HTML 文件的编码，来确定自己的编码。

字符编码的代码要放在 CSS 文件的最前端，由于网页代码的加载顺序自上而下，放在最前面能够让后面所有的代码都按照这种字符编码格式进行编码和解析。

### 4. 路径

1) 什么是路径

从字面上可以将路径理解为通向某个目标的道路。

如果需要在 HTML 文件当中引入一个 CSS 文件，那么从当前的 HTML 文件出发，怎样才能够找到这个 CSS 文件呢，是找当前文件级别下的某个文件，还是找当前位置的父级的某个文件，或者找当前位置的子级的某个文件？

以当前 HTML 文件为起点，以要引入的文件为终点，构成的道路就是“路径”。

2) 何处需要使用路径

在前端开发中，除了在 HTML 中引入 CSS、JS 文件时会使用路径之外，还会在 HTML 或 CSS 文件引入图片、字体、多媒体文件等情况下用到路径。应该说，只要是需要引入的文件，都需要使用到路径。

3) 路径的种类

路径有两类，一种是相对路径，一种是绝对路径。

4) 相对路径

相对路径：以引用文件的网页所在位置为参考基础，而建立出的目录路径。当保存于

不同目录的网页文件引用同一个文件时，所使用的路径将不相同，故称之为相对。例如：

```
../images/h5course.jpg
```

5）绝对路径

绝对路径：以 Web 站点根目录为参考基础的目录路径。之所以称为绝对，意指当所有网页引用同一个文件时，所使用的路径都是一样的。例如：

```
G:/2016/h5course/images/h5course.jpg
```

6）开发时的选用

对于绝对路径和相对路径的选择，在开发中通常使用相对路径。这主要是因为当开发工程师在客户端书写好代码，并上传到服务端时，服务器端的盘符、具体的文件位置与客户端并不相同，如果使用绝对路径，会在路径上出现错误。

7）路径的特殊符号

../表示当前文件所在层级的上一层级。

./表示当前文件所在的层级。

/表示根目录。

8）相对路径书写实例

图 3.6 展示了一个相对路径书写实例。

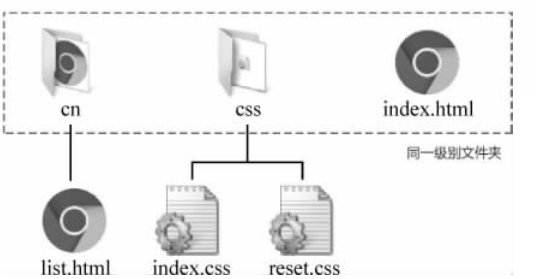



**图 3.6　相对路径书写实例**

当前的文件夹结构中存在 cn 文件夹、css 文件夹以及 index.html 文件。在 cn 文件夹中有一个 list.html 文件，css 文件夹当中有两个 CSS 文件 index.css 和 reset.css。

代码实例 1：index.html 中引用 reset.css 文件：首先找到同级的 css 文件夹，之后找到这个文件夹中的 reset.css 文件。

```
css/reset.css
```

代码实例 2：list.html 中引用 reset.css 文件：首先找到 list.html 文件的父级（cn 文件夹），然后找到 cn 文件夹的同级 css 文件夹，再从 css 文件夹当中找到 reset.css 文件。

```
../css/reset.css
```

### 5. 外部引入的书写方法

以上面案例中的 10 个 div 为例，假设存在多个 HTML 文件，这 10 个 div 在这些 HTML 文件中属于公共模块，即多个 HTML 文件当中存在这 10 个 div 元素，且在这些文件中的 10 个 div 元素样式相同。

一旦客户需求发生变更，开发工程师就需要打开所有包含“公共模块”的 HTML 文件，依次进行编辑和修改，这时的操作变得异常复杂和麻烦。第三种样式引入方式——“外部引入”能够很好地解决这个问题。

外部引入的原理：外部引入是将 CSS 代码书写在一个 CSS 文件当中，然后通过 link 标签，使用正确的路径，将这个 CSS 文件引入到 HTML 文件当中。在 CSS 文件中的具体代码遵循了内部书写的基本规则，也涉及选择器、具体样式等。

**代码实例**：HTML 文件中的具体内容如下。

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 外部引入 CSS</title>
    <link rel="stylesheet" href="css/index.css">
</head>
<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
    <div>5</div>
    <div>6</div>
    <div>7</div>
    <div>8</div>
    <div>9</div>
    <div>10</div>
</body>
</html>
```

**代码实例**：CSS 文件中的具体内容如下。

```
@charset 'UTF-8';
div{
    border: 1px solid black;
}
```

**代码注意事项：**

将 HTML 文件命名为 index.html，CSS 文件名称为 index.css(index 为文件的名称，.html、.css 均表示文件的后缀名)。

CSS 文件处于 HTML 文件所在文件夹中的 css 文件夹内。即某文件夹包含 index.html 和 css 文件夹，而 css 文件夹包含 index.css。

文件夹名与文件名的大小写不要写错，此处 link 标签中的 href 表示路径，如果文件位置放置错误或文件名写错，CSS 文件并不能够被引入到 HTML 文件中，样式不会生效。

外部引入这种方式将 HTML 结构与 CSS 样式完全分离成了两个文件，实现了结构与样式的分离，增强了网页结构的扩展性，提高了代码的可维护性。在多个 HTML 文件当中都引入了同一个 CSS 文件时，这个 CSS 文件中的代码修改，就能够引发这些 HTML 文件中相应部分的变化，代码的操作与维护就变得更加快捷与轻松了。

### 3.3.4　CSS 三种常见引入方式比较

页面中使用 CSS 的方式主要有三种，分别是行内书写(行内添加定义 style 属性值)、内

部书写(页面当中使用< style >标签调用样式)和外部引入(通过外部链接,引入 CSS 文件)。

下面比较一下这三种 CSS 的引入方式,看看在实战当中,它们分别应用于什么情况。

### 1. 控制性

控制性比较如表 3.1 所示。

**表 3.1 CSS 三种引入方式——控制性比较**

| 引入方式 | 行内书写 | 内部书写 | 外部引入 |
|---|---|---|---|
| 控制性 | 最弱 | 中等 | 最强 |

标签内书写:只控制当前标签。

头部书写:通过一个选择器能够控制当前页面中所有符合条件的标签。

外部引入:一个 CSS 文件能够控制多个页面,实现了结构与样式相分离。

### 2. 维护性

维护性比较如表 3.2 所示。



**表 3.2 CSS 三种引入方式——维护性比较**

| 引入方式 | 行内书写 | 内部书写 | 外部引入 |
|---|---|---|---|
| 维护性 | 不便于维护 | 维护性一般 | 便于维护 |

标签内书写:修改起来最不方便。

头部书写:对于多个网页中都存在的公共样式,修改起来并不方便,需要多次重复修改。

外部引入:对于多个网页中都存在的公共样式,只需要修改一次就可以完成所有的修改。

### 3. 代码量

代码量比较如表 3.3 所示。

**表 3.3 CSS 三种引入方式——代码量的比较**

| 引入方式 | 行内书写 | 内部书写 | 外部引入 |
|---|---|---|---|
| 代码量 | 存在大量的冗余代码 | 对于单页面来说,没有冗余代码<br>对于整站,依旧存在大量冗余代码 | 对于整个网站来说,不存在冗余代码,代码量较少,有效地利用了浏览器有缓存机制<br>对于单页面来说,存在冗余代码 |

标签内书写:文件大小变大,从而导致加载速度变慢,用户体验变差。

头部书写:对于单页面来说,所有的 CSS 代码都是有用的,加载速度较快。但是,由于各个网页中存在相似模块,因此对于整站来说,存在冗余代码,代码量较大。

外部引入:访问站点时,会下载 CSS 文件,当整站 CSS 文件下载完毕之后,再去加载其他页面,速度会非常快(涉及浏览器缓存)。但是,对于单页面来说,特别是第一个页面,浏览器需要下载的 CSS 文件当中,除了包含当前页面的样式之外,也会包含其他页面的 CSS 代码,因此,针对这一个页面来说,就出现了冗余代码,加载速度相对较慢。

#### 4. 服务器请求压力

服务器请求压力的比较如表 3.4 所示。

**表 3.4　CSS 三种引入方式——服务器请求压力的比较**

| 引 入 方 式 | 行内书写 | 内部书写 | 外 部 引 入 |
|---|---|---|---|
| 服务器请求压力 | 无 | 无 | 有可能造成 |

外部引入：数据(图片、文件)能够呈现在网页中，需要两步——由客户端向服务器发出申请，再由服务器将数据发回。当客户端的用户多次向服务器索要数据，同时服务器还很忙时，就会面临“服务器请求压力”的问题。

#### 5. 应用场景

CSS 三种引入方式的应用场景如表 3.5 所示。

**表 3.5　CSS 三种引入方式的应用场景**

| 引入方式 | 行 内 书 写 | 内 部 书 写 | 外 部 引 入 |
|---|---|---|---|
| 应用场景 | 特殊情况(基于其优先级特点) | 比较大型网站的首页 | 大部分小型网站 |

标签内书写：特殊情况指的是“找不到哪里出现问题，但是项目又着急上线，需要立即修改某个部分的样式”。

头部书写：比较大型网站的首页，需要以最快速度加载出首页，把样式代码书写在网页头部必然是最好的选择。

外部引入：小型网站没有太多的页面，也不会生成太大量的代码，因此可以将整站的代码放置在同一个文件当中。

#### 6. 综述

在实际的开发当中，通常以外部引入为主，在必要的时候选用内部书写和行内书写的方法。切忌背定义，要根据具体的开发要求灵活变通。

### 3.3.5　外部引入 CSS 的扩展知识

#### 1. @import 的基本语法

除了介绍到的 link 标签能够外部引入 CSS 文件之外，还可以使用@import。

使用 link 方法引入的代码实例：

```
<link rel = "stylesheet" rev = "stylesheet" href = "CSS 文件" type = "text/css" media = "all" />
```

使用@import 的方法引入的代码实例：

```
<style type = "text/css" media = "screen">
    @import url("CSS 文件");
</style>
```

2. link 与@import 的区别与使用原则

这两种方法虽然都能够实现外部引入 CSS,但是存在一定的区别。也正是由于这些区别,才导致多数人在实际开发中放弃了使用@import 的引入方式。

(1) link 是 XHTML 标签,除了加载 CSS 外,还可以定义 RSS 等其他事务;@import 属于 CSS 范畴,只能加载 CSS。

(2) link 引用 CSS 时,在页面载入时同时加载;@import 需要页面网页完全载入以后加载。

(3) link 是 XHTML 标签,无兼容问题;@import 是在 CSS2.1 中提出的,低版本的浏览器不支持。

(4) link 支持使用 JavaScript 控制 DOM 去改变样式;而@import 不支持。

### 3.3.6 CSS 引入方式的问题区

温馨提醒:对于理论类的问题请先思考,实战类的问题可以先动手输代码,得出基本结论,再看答案,学习方法和基本知识同等重要。



(1) 如何理解< div style="border: 1px solid red"> 代码中 style 与 div 的关系?

① 代码的基本含义。

div 是一个标签元素,style 是标签的一种属性,而“border:1px solid red”是 style 这种属性的具体属性值。对于一个标签来说,能够拥有多种属性,另外,具体的属性值可能会再细分。

```
<标签名 标签的某一种属性名字 = "具体属性值" 标签的某一种属性名字 = "具体属性值"></标签名>
```

② 借助生活的类似实例加深理解。

可以借助生活中的实际情况来理解这段代码:将把元素理解成人,每个人都有基本特征这种属性,而基本特征当中又可以细化分为性别、身高、体重等(当然,每个人也可以有爱好、教育经历、工作经验这些属性等)。于是这段代码就成了这个样子:

```
<人 基本特征属性 = "性别: 男; 身高: 180cm; 体重:80kg;" 教育经历 = "本科: ……; 高中: ……; 初中: ……; 小学: ……;" 工作经验 = "……" 个人爱好 = "……"></人>
```

③ 回看 link 标签。

此时,应该能够更深入地理解外部引入 CSS 时使用的 link 标签了。

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

本段代码当中,link 是一个标签,拥有 rel、href、type 三种属性,其中,rel 属性的属性值是“stylesheet”;href 属性的属性值是 index.css 这个文件的文件路径,即“../css/index.css”;type 属性的属性值是“text/css”。

(2) 网页中有冗余代码有何影响?

代码冗余度大,通常是由于在书写代码时,采用了错误的 CSS 引入方式,或者没有将相同模块的样式合并起来进行书写。大量的冗余代码会让工程师的维护工作变得麻烦,也会

增加文件的大小，从而对加载速度造成影响。

① 对于工程师来说，代码变得难于维护。

开发工程师书写完成基本的网页代码，并不是就结束了，后期需要大量的修改和维护。换言之，工程师还要多次面对自己写过的代码，并针对其进行修改。

如果在网页制作过程中，开发工程师采用的是冗余度较高的 CSS 行内书写的方式，一旦进行修改，就不得不查看很多文件，并一一排查，确定是否需要修改。一个很简单的、很小的修改操作，就可能会耗费掉工程师非常大量的时间。如果工程师采用< style >标签在 HTML 文件中内部书写 CSS，对于一些修改也会比较繁杂，虽然不用一个个地排查标签，但是还是需要打开各个 HTML 文件，进行一些重复操作，在操作过程中也比较容易造成文件的遗忘和丢失，出现比较低级的错误。

② 用户体验下降。

冗余的代码，也就意味着文件增大，当文件增大之后，整个网站的加载速度会下降，而用户体验会变差。网站的每一个浏览者，都希望能够快速打开一个网站，而不是等待三五秒之后网站才逐渐呈现在眼前。用户体验变差，很有可能造成用户流失，从而对企业造成致命性的影响。毕竟，对于广大的用户来说，只要一个网站不是唯一的、必须使用的，那么他们完全可以抛弃当前平台，去找另一个功能类似但用户体验更好的网站。



(3) style 标签必须书写在头部吗？放在其他位置会不会无法实现样式效果？

可以在编辑器当中尝试如下代码：

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - style 位置的影响</title>
</head>
<body>
    <div>第一个 div</div>
    <style>
        div {
            border: 1px solid red;
        }
    </style>
    <div>第二个 div</div>
</body>
</html>
```

在浏览器显示效果当中，两个 div 均存在 1 像素红色实线边框。也就是说，style 标签书写在 body 标签中时代码生效，div 得到了 style 标签中设置的具体样式效果。

style 并不仅仅可以放置在 body 元素之内，还可以放在某个具体标签之内，也可以和其他标签同级，还可以放置在< html >标签之内，< body ></body >之后，甚至可以放在</html >标签的后面。

虽然 style 标签的位置随意，但是并不推荐将它放置在< head ></head >标签之外。主要原因有以下两点。

① 不便于维护：当 style 标签放置到 body 标签中，工程师在后期维护时，就需要花额

外的时间进行寻找，在密密麻麻的代码当中寻找某种样式并进行修改，或者调试是否因为某种样式而引发的问题，绝对是一件让人心烦的事情。

② 会引起页面回流与重绘，从而降低加载速度。网页是由 DOM Tree（也就是各个标签）和样式结构体组合后构建成的 render tree。render tree 构建完成后，浏览器就会根据 render tree 来进行页面的绘制。

- 页面回流：当 render tree 中的一部分（或全部）因为元素的规模尺寸、布局、隐藏等改变而引起的页面重新渲染（或者叫作重新构建绘制）。
- 页面重绘：当 render tree 中的一些元素需要更新属性，但这些属性只会影响元素的外观、风格，而不会影响到元素的布局，此类的页面渲染叫作页面重绘。

由于网页文档是自上而下加载并解析的，假设将 style 标签书写在了</body >的后面，在读取到 style 标签之前，整个网页文档已经渲染得差不多了，这时候突然发现存在一个 style 标签，并且 style 标签中的内容针对网页中某些元素的样式进行了设置，于是浏览器只好重新为每个元素计算样式，再加载出来。这会导致网页的加载速度变慢。

（4）对于引入 CSS 文件的 link 标签，可不可以书写在网页文档的其他位置？



与上面提到的< style ></style >类似，link 标签也可以放置在网页文档的其他地方。出于与< style >标签同样的原因，会将< link >标签放置在 head 标签当中，先于 HTML 结构加载。

## 3.4 CSS 选择器

通过引入方式，在 HTML 页面中“放入”了 CSS 样式，接下来要做的操作就是让相应的标签应用定义好的 CSS 样式，这里会涉及“选择器”的基本知识。

### 3.4.1 生活中的“选择器”——找人

今天学校举办了一场“舞会”，邀请所有的同学出席，现在你和你宿舍的好友们已经把要穿的衣服、饰品准备好了，之后就是要“选择”谁“穿着”什么样的“衣服或饰品”出席“舞会”。

例如，我们希望：

（1）“身份证号码”是 XXX 的人穿着“长裙”；

（2）“班级”是“HTML5-1 班”的人，手上戴着“腕表”；

（3）“男性”打“领带”。

将“希望”书写成需求，并进行合理拆解，如表 3.6 所示。

**表 3.6 找人的基本需求拆解**

| 需　　求 | 选 择 条 件 | 服装要求（样式） |
|---|---|---|
| “身份证号码”是 XXX 的人穿着“长裙” | 身份证号是 XXX | 长裙 |
| “HTML5-1 班”的人，手上戴着“腕表” | HTML5-1 班 | 腕表 |
| “男性”打“领带” | 男性 | 领带 |

每个人最初都没有“服装要求”（样式），所有的“服装要求”（样式）是根据需求已确定的，而“选择条件”将人和“服装要求”（样式）关联到了一起。

### 3.4.2　CSS 选择器的基本语法

```
选择器名{
    属性名: 属性值;
    属性名: 属性值;
    ⋮
}
```

### 3.4.3　CSS 基本选择器

在生活的“舞会”当中,通过“选择条件”,让不同的人和不同的服装要求建立关系。在代码世界当中,HTML 标签就相当于参加舞会的每个人,而 CSS 就是“服装要求”(样式),连接这个人和礼服的桥梁,就是“选择器”。

CSS 当中存在三种基本选择器,分别被称为:ID 选择器、类名选择器与标签名选择器。也就是说,CSS 为我们提供了三种最基本的方法,来为“标签”与“样式”建立联系。

对于 ID 选择器,可以理解成身份证;类名选择器,可以理解为班级(或一个类);标签名选择器,可以理解为性别。



**1. 通过身份证找人——ID 选择器**

**基本语法:**

```
#选择器名{
    属性名: 属性值;
    属性名: 属性值;
    ⋮
}
```

**选择器代码:**

```
#con {
    border: 1px solid red;
}
```

**代码实例:**

```
<!doctype html>
<html>
<head>
    <meta charset = "UTF - 8">
    <title>HTML5 布局之路 - ID 选择器</title>
    <style>
        #con {
            border: 1px solid red;
        }
    </style>
</head>
<body>
```

```
    <div id = "con">1 </div>
    <div>2 </div>
    <div>3 </div>
</body>
</html>
```

**代码解析：**

#con {border: 1px solid red;}这句代码，表示找到 ID 名为 con 的元素，并为元素设置 1 像素、实线、红色边框。

身份证号码，是生活中标识人们身份的一串数字，那么身份证号码（ID 选择器）有什么特点呢？

（1）属于我们每个人自己、独一无二，并不会与其他人重复。

（2）通过身份证来找人，只能找到一个人。

（3）号码需要定义。身份证号码，并不是在人出生时就规定好的，而是需要到相关部门登记，在登记后我们才拥有了这个号码。在 HTML 代码当中，设置 id 属性，就是定义了 id 名，在 CSS 中通过这个 id 名来寻找相应的元素，此时“#id 名”就是创建了一个 ID 选择器，让样式与 ID 选择器对应的标签建立联系。



### 2. 通过班级找人——类名选择器

**基本语法：**

```
.选择器名{
    属性名: 属性值;
    属性名: 属性值;
    ⋮
}
```

**选择器代码：**

```
.con {
    border: 1px solid blue;
}
```

**代码实例：**

```
<!doctype html>
<html>
<head>
    <meta charset = "UTF-8">
    <title>HTML5 布局之路 - 类名选择器</title>
    <style>
        .con {
            border: 1px solid blue;
        }
    </style>
</head>
<body>
    <div class = "con">1 </div>
```

```
    <div>2</div>
    <div class="con">3</div>
</body>
</html>
```

**代码解析：**

.con {border: 1px solid blue;}这句代码，表示找到类名(class)为 con 的元素，并为元素设置 1 像素、实线、蓝色的边框。

班级(即类名选择器)的特点如下。

(1) 有可能选择到的是一个(班里只有一个人)，也有可能选择到的是多个(每个班级可以有多个人，很多标签都可以起同一个类名)。

(2) 班级(类名选择器)，并非是天生所有的属性，而是需要后期定义的。在 HTML 代码当中，class 属性的设置，就是在定义类名；而 CSS 中的“.类名”就是创建了一个类名选择器，让其中的样式与类名选择器对应的标签建立联系。

### 3. 通过性别找人——标签名选择器

**基本语法：**

```
选择器名{
    属性名: 属性值;
    属性名: 属性值;
    ⋮
}
```

**选择器代码：**

```
div {
    border: 1px solid blue;
}
```

**代码实例：**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 标签名选择器</title>
    <style>
        div {
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
</body>
</html>
```

**代码解析：**

div{border：1px solid black;}这句代码，表示找到标签名为 div 的元素，并为元素设置 1 像素、实线、黑色的边框。

性别（即标签名选择器）的特点如下。

（1）如果说找到所有的男性，那么全世界的男性都符合这个条件，可见，这种方式找人，找到的人数有可能是一个，也有可能是多个。

（2）每个人在出生的时候就已经自带了性别属性，并不是说别人把自己定义成什么性别，就是什么性别，因此，对于性别（标签名选择器）这个特点，并不需要进行定义。在代码当中：标签名选择器，使用的就是标签，标签名就如同性别，只不过标签的类别比较多，不只有男、女，此处可以理解成动物的种类（如灵长类、两栖类）。此时，直接将“标签名”作为选择器名，就能够让样式与相应的标签建立联系。

（3）注意：HTML 当中不存在的标签，是不能够使用标签名选择器来进行书写的。

**4. 三种选择器的特点总结**

三种基本选择器的比较如表 3.7 所示，三种基本选择器的用法如表 3.8 所示。



**表 3.7　三种基本选择器的比较**

| 选择器类型 | 生活中的同类物 | 选择范围 | 精准度 | 是否需要定义 |
|---|---|---|---|---|
| ID 选择器 | 身份证号 | 最小 | 高 | 是 |
| 类选择器 | 班级号 | 较大 | 一般 | 是 |
| 标签名选择器 | 性别 | 最大 | 最差 | 否 |

**表 3.8　三种基本选择器的用法**

| 选择器类型 | 标签中的操作 | CSS 中的使用方法 |
|---|---|---|
| ID 选择器 | 定义标签的 id 属性和属性值 | #选择器名{<br>属性:属性值;<br>} |
| 类选择器 | 定义标签的 class 属性和属性值 | .选择器名{<br>属性:属性值;<br>} |
| 标签名选择器 | 无 | 标签名{<br>属性:属性值;<br>} |

## 3.4.4　样式冲突的问题

一个人，可以同时属于“女性”、“1 班”、“身份证号 XXX”，一个标签也可以被多种选择器同时选中，此时就会出现样式冲突的问题。在了解具体的样式冲突之前，先看几个代码实例。

样式冲突问题——代码实例 1：

```
<!doctype html>
<html>
```

```
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 样式冲突问题 1</title>
    <style>
        div {
            border: 1px solid red;
        }
        .con {
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <div>1</div>
    <div class="con">2</div>
    <div>3</div>
</body>
</html>
```

样式冲突问题——代码实例 2：

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 样式冲突问题 2</title>
    <style>
        div {
            border: 1px solid red;
        }
        div {
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
</body>
</html>
```

样式冲突问题——代码实例 3：

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 样式冲突问题 3</title>
    <style>
        div {
            width: 100px;
            height: 100px;
            border: 1px solid black;
```

```
        }
        #test {
            background: #39f;
        }
        .con {
            border: 10px solid black;
        }
    </style>
</head>
<body>
    <div>1</div>
    <div id="test" class="con">2</div>
    <div>3</div>
</body>
</html>
```

在如上的三个例子当中，虽然都是样式冲突，但是也各有不同。有些选择器中书写的是同样的样式；有些选择器书写的是不同的样式；有些标签能够被多个选择器选择到，而且在不同的选择器当中设置了不同的样式。

那么，在这些样式中，哪些样式才是标签最终真正展示出来的样式呢？

此时，需要先了解一下“选择器的优先级”的基本知识。



### 3.4.5 生活中的“优先级”——谁是老大

“优先级”其实可以理解成“谁说了算”，如表 3.9 所示。

表 3.9 三种基本选择器的身份设定

| ID 选择器 | 类名选择器 | 标签名选择 |
| --- | --- | --- |
| 老师 | 班长 | 组长 |

经历了一周的学习，到了周五上午，组长通知说：“明天需要过来上课”，而班长也发出了通知：“明天放假，不需要过来上课”，那么此时，你会听谁的呢？

相信你会在心里比较组长和班长这两个人哪个人的级别高，谁说话算数，就听谁的。

### 3.4.6 CSS 选择器优先级

#### 1. 行内样式与 CSS 选择器的优先级

在行内，使用 style 属性书写的样式，优先级最高（但是通常都不这么写样式）。

ID 选择器的优先级其次，类名选择器优先级再次，优先级最弱的是标签名选择器，如表 3.10 所示。

表 3.10 style 属性以及三种基本选择器的优先级

| 作为标签属性 style，设置的各类样式 | ID 选择器优先级 | 类名选择器优先级 | 标签名选择器优先级 |
| --- | --- | --- | --- |
| 1000 | 0100 | 0010 | 0001 |

### 2. 不同的选择器的优先级问题——对应样式冲突的第一个例子

实例解析，如图 3.7 所示。

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5布局之路 - 样式冲突问题1</title>
    <style>
        div {
            border: 1px solid red;
        }
        .con {
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <div>1</div>
    <div class="con">2</div>
    <div>3</div>
</body>
</html>
```




**图 3.7　不同选择器的优先级问题——实例解析图**

< style >标签中的第一个选择器“div”是标签名选择器，第二个选择器“. con”是类名选择器。

第一个选择器能够选择到 body 中的所有 div 元素，也就是为每个 div 元素均设置了 1 像素实线的红色边框。

第二个选择器选择到的是类名为 con 的元素，也就是上面代码中的第二个 div 元素。为其设置了 1 像素实线的黑色边框。

那么此时，第二个 div 元素应该是什么样式呢？边框是黑色的还是红色的呢？

上面提到，标签名选择器的优先级是 0001，而类名选择器的优先级是 0010。这两个选择器相比，很明显，0010 要大于 0001，也就意味着，“. con”这个选择器的优先级高于“div”这个选择器，因此显示出来的是黑色边框。

### 3. 同种选择器的优先级问题——对应样式冲突的第二个例子

依旧使用之前的身份设定。周五上午，班长通知大家：“明天需要过来上课”，可是到了周五的下午，班长又发出了通知：“明天放假，不需要过来上课了。”那么此时，你是遵循班长的第一个通知还是第二个通知呢？

相信你会听后面的“消息”。在 CSS 当中，由于网页的读取是自上而下的，因此，对于同种优先级的选择器，后书写的代码生效。

实例解析如图 3.8 所示。

< style >标签中有两个选择器，都是“div”这种标签名选择器，优先级均为 0001，这两个选择器均能够选择到 body 中的所有 div 元素。

第一个选择器为所有的 div 设置了 1 像素的红色实线边框。第二个选择器为所有的 div 设置了 1 像素的黑色实线边框。这两个选择器优先级别相同，但是，由于第二个选择器在后面，浏览器就会用“第二个选择器对应的边框样式”覆盖掉“第一个选择器对应的边框样

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5布局之路 - 样式冲突问题2</title>
    <style>
        div {
            border: 1px solid red;
        }
        div {
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
</body>
</html>
```

两个选择器，均选择到了所有div标签

图 3.8　同种选择器的优先级问题——实例解析图



式”。最终三个 div 显示出来的，均为黑色边框。

#### 4. 样式并不是只有覆盖——对应样式冲突的第三个例子

当不同的选择器都选择到了某个标签，并且针对这个标签设置了相同的样式，这些样式会被彼此覆盖掉。但是，当不同的选择器都选择到了某个标签，而不同选择器设置的样式各不相同时，会发生什么？

生活中的实例：

组长（标签名选择器）发布通知：“明天放假”。

班长（类名选择器）发布通知：“明天上课；明天需要携带学生证”。

老师（ID 选择器）发布通知：“明天需要携带身份证”。

综合之后，消息会变成“明天上课，需要携带学生证、身份证”。

在这些消息当中，有冲突的消息发生了覆盖（明天放假还是上课），而没有冲突的消息（携带学生证、携带身份证）并没有产生任何的覆盖，而是进行了叠加。

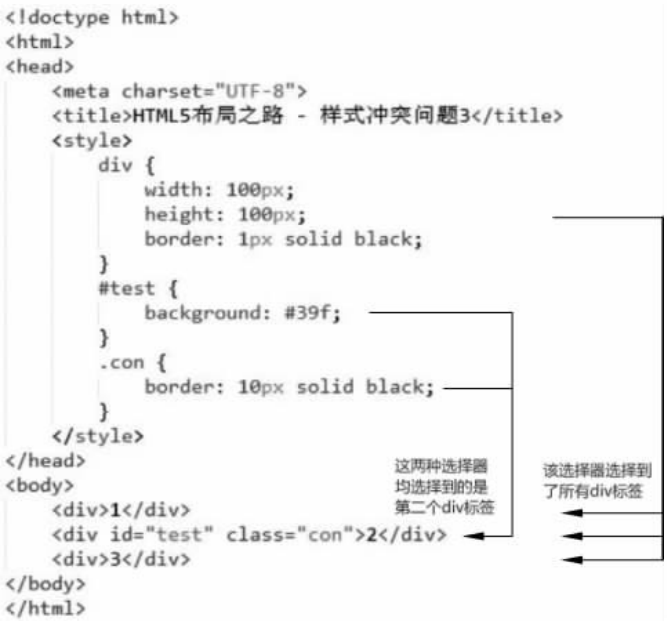
对于 CSS 样式，针对一个元素的同一种属性而设置的样式，会根据选择器的优先级进行覆盖；如果不同的选择器针对元素的不同属性进行了样式设置，这个元素会同时拥有所有符合条件的属性样式。

实例解析如图 3.9 所示。

“div”这个选择器，会选择到所有的 div 元素；“#test”这个选择器会选择到 id 名为 test 的元素；“.con”这个选择器会选择到 class（类）名为 con 的元素。

对于 body 中的第二个 div 元素，均符合这三种选择器的选择条件。那么这时，这个 div 会是什么样式呢？

同时拥有这三种选择器中的样式，但是“border”的样式发生了重复，此时需要比较哪个选择器的优先级更高。




```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5布局之路 - 样式冲突问题3</title>
    <style>
        div {
            width: 100px;
            height: 100px;
            border: 1px solid black;
        }
        #test {
            background: #39f;
        }
        .con {
            border: 10px solid black;
        }
    </style>
</head>
<body>
    <div>1</div>
    <div id="test" class="con">2</div>
    <div>3</div>
</body>
</html>
```



**图 3.9　样式并不是只有覆盖——实例解析图**

当前第二个 div 元素的样式为：

```
width: 100px;
height: 100px;
border: 10px solid black;
background: #39f;
```

### 3.4.7　行内的 style 属性

除了上述几种基本选择器外，如前所述可以使用行内的 style 属性进行样式的设置。行内 style 属性中定义的样式，优先级要比所有的选择器都高，在实际开发当中，由于这种方式没有实现样式和结构的相分离，并不推荐使用，在此也不再详细讲解。

**基本语法：**

```
<标签名 style="属性名: 属性值; 属性名: 属性值;">具体内容</标签名>
```

### 3.4.8　选择器的使用原则

在网站实战当中，如何合理地选用“三种基本选择器”呢？

(1) 在此强烈推荐新手使用“类名选择器”。之所以选择“类名选择器”，主要原因在于：类名选择器可以同时选择多个元素，类似的元素可以使用同一个类名，比 ID 选择器操作起来更灵活；由于类名选择器必须定义，只有定义类名的标签才能够应用样式，并不会对网页中其他元素造成不必要的影响。

(2) 标签名选择器，由于其选择范围过广，会导致网页当中所有的同种类型标签都会被

选中，此时很容易造成样式影响。不推荐使用。

(3) ID选择器，优先级比较高，并且只能够选择到一个，适合其使用的“环境”并不多。另外，ID选择器，通常是给原生JavaScript预留，一旦看到ID选择器，就能够想到“在这里应该是有JS功能存在的”。不推荐使用。

**注意**：可能你会觉得这三种选择器的操作并不足够灵活，不要紧，CSS还存在更多功能更强的“选择器”。但是，由于当前的“整体布局”功能并不需要太复杂的选择器，因此在第5章中再详细讲解。

### 3.4.9 CSS选择器的问题区

(1) 0010的优先级数字是怎么回事？

当前在学习的是网页的整体布局，所需要处理的布局结构相对比较简单，后期的一些模块布局当中，会遇到更复杂的网页结构，也会讲解各类选择器，当一个标签符合多个选择器的选择条件时，它的最终样式应当如何计算，就成了一个必须要解决的问题。由于当前处于整体布局阶段，并不需要使用到复杂的选择器，因此请先记住这三种基本选择器的优先级，对于0010所涉及的计算，在后面第5章当中会做详细解释。



(2) 开发当中是不是不能使用id选择器和标签名选择器？

当然不是！

样式重置方面，会使用到标签名选择器(样式重置会在3.6节当中讲到)。

在网页的其他地方，id选择器和标签名选择器也是能够使用的。

在前面“选择器选用”时，之所以提醒大家不要随意使用，主要是由于这两种选择器很容易出现样式覆盖，导致标签的最终显示效果出现问题。

选择器种类的具体选用，要根据网页的实际情况而定。例如，如果整个网站当中，只存在一处表格，需要使用到表格类元素，那么可以使用标签名选择器，因为它并不会对其他任何地方造成影响。但是，即便如此，依旧不推荐使用标签名选择器(除样式重置外)。在开发时，需要考虑后期的维护和调整，以刚才表格需求为例，一旦后期网站发生调整，在网站中添加了几处表格，就需要调整原有的样式代码，维护工作就会变得很不方便。

## 3.5 CSS编码规范

同样都能够实现代码，为何还要考虑编码规范？

(1) 从维护角度来说，编码在绝大多数情况下是拿给人来看的，只是偶尔让机器读一下。了解基本的编码规范，对自己日后的维护，或者自己小伙伴的维护是有推进作用的。

(2) 优秀的代码会让团队工作事半功倍，不同的编码习惯，其实也直接影响着薪资。

### 3.5.1 CSS注释

CSS代码当中，使用/**/来表示注释，在/*与*/之间书写具体的注释内容，被注释的内容并不会被浏览器解读，因此可以使用中文注释。在书写代码时，有时需要花几个工作日来完成某一个模块或页面；有时一些CSS代码只是测试使用、后期需要删除；有时书写的一些CSS代码只是出于某种考虑添加到原有代码当中的。合理的注释能够让开发者更

清晰某段代码的意义，大幅提升代码的可读性。

CSS 注释，可以针对单行文本进行注释，也可以针对多行文本进行注释。

**代码实例：**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - CSS 代码注释</title>
    <style>
        div {
            width: 100px;                    /* 此处书写的是代码注释 */
            height: 100px;                   /* 一段 CSS 代码注释,
            可以是单行的,也可以是多行的,
            并不会受到换行等问题的影响*/
            border: 1px solid black;
            /* 在书写时,建议将注释内容与符号之间添加一个空格,方便查看 */
        }
    </style>
</head>
<body>
    <div>/* CSS 代码注释在 HTML 代码中不生效 */</div>
</body>
</html>
```



需要注意的是，当浏览器读取到第一个/*时即为注释开始，之后寻找到第一个*/即为注释结束，因此，如果在原有注释外再进行注释，建议先将原有注释删除，防止出现问题。CSS 注释如图 3.10 所示。

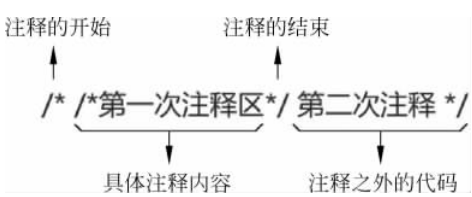



**图 3.10　CSS 注释**

## 3.5.2　书写风格

(1) 每一条规则的大括号{前后加空格。

(2) 每一条规则结束的大括号}前加空格。

(3) 属性名冒号之前不加空格，冒号之后加空格。

(4) 每一个属性值后必须添加分号；单行模式下，分号后加空格。

### 1. 单行形式

```
div { width: 100px; height: 200px; }
```

### 2. 多行形式

```
div {
    width: 200px;
    font-size: 16px;
}
```

**注意：**

(1) 空格部分的添加与否，并不会影响到代码的读取。合理地书写空格，能够提升代码的可读性。

(2) 大括号、冒号、分号，则必须使用英文半角状态下的标点符号，并且不能缺失，否则会影响代码读取，页面有可能无法渲染成功。

(3) 在{}中，最后一组的“属性：属性值；”中结尾的分号可以省略，但是并不推荐省略，特别是对于新手来说，省略之后再添加某些样式时，可能忘记添加分号，很容易出现代码问题。

(4) 单行形式和多行形式相比，更加推荐使用多行形式，在阅读方面会更快捷清晰。有些人可能考虑到单行形式的代码量会小一些，的确如此。但是，在实际开发当中，工程师开发时使用的CSS文件，需要经过压缩之后，才上传到服务器端，压缩的过程中就有这种“多行形式”到“单行形式”的自动转换。

### 3.5.3 关于类名命名



(1) 不建议使用中文进行命名；

(2) 不建议使用汉语拼音进行命名；

(3) 不建议使用box1、2、3、4、…、10等英文＋数字的形式进行命名；

(4) 不建议使用过长的英文单词进行命名；

(5) 建议使用英文单词进行类名的命名；

(6) 建议根据模块具体含义选用英文单词，类名要有具体含义；

(7) 建议英文单词可以进行适当的缩写；

(8) 建议不同单词之间，可以使用连字符或下画线进行分隔，如：.list-tit、.list_tit；

(9) 建议尽量避免class与id重名(id通常是为JavaScript服务的)。

### 3.5.4 样式书写顺序

从本章开始之后的几章会介绍各类样式属性。这些样式属性可以简单地划分为4大类，分别是“显示样式”、“自身样式”、“文本样式”和“CSS3新样式”。

(1) 显示样式：控制元素展示方式的属性，主要包括浮动(float)、定位(position)、展示方式(display)、超出状态以及可视化(overflow、visibility)等。在第4、5、6、8章当中会进行介绍。

(2) 自身样式：关于元素自身的样式属性，主要就是本章涉及的5种属性(宽度width、高度height、外边距margin、内边距padding、边框border)以及第9章涉及的最大最小宽高。

(3) 文本样式：用于处理背景图片、段落文章、文字字体的样式。第7章中的各类样式均属于文本样式。

(4) CSS3新样式：CSS3新增的属性，第13～16章当中讲解的属性。

在进行编码时，建议遵循“显示样式”→“自身样式”→“文本样式”→“兼容与CSS3新样式”，一方面便于开发者查看，另一方面，浏览器对一个标签样式的解析，是按照“显示样式”→“自身样式”→“文本样式”的顺序执行的。

在此给出比较完整的代码书写顺序(推荐):

- display;
- position;
- position 相关的 left、top、right、bottom、z-index;
- float;
- clear;
- width;
- height;
- margin;
- padding;
- border;
- background;
- color;
- font;
- text-decoration;
- text-align;
- vertical-align;
- white-space;
- text-XXX(其他的 text 类属性);
- CSS3 类。

关于如上的各类样式代码,会在后面的章节当中逐渐讲解到。对于本书当中的各类案例,代码的书写规范也尽可能地遵循了这个顺序。在此需要掌握的是,明确 CSS 的代码书写顺序,并在实际开发当中养成良好的代码编写习惯。

### 3.5.5　CSS 编码规范的问题区

(1) 不遵循编码规范会影响效果实现吗?

大部分情况下都不会。编码规范类似于一个行业标准,是行业人达成的共识,并非浏览器渲染的标准。

开发工程师要掌握编码规范,并在开发中实际应用的主要目的,在于要使自己的代码优秀,从而使自身水平更贴近于行业标准。使用 CSS 给一个标签书写样式并不难,难的是要高质量地实现样式。CSS 最核心的地方在于,如何将代码写得优秀!

(2) 为何不能随便起名字?

类的命名有各种各样的方法,如使用中文类名、汉语拼音、英文单词,各种类名都能够实现想要的样式。

**代码实例:**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
```

```
    <title>HTML5 布局之路 - 各式各样的命名方式</title>
    <link rel = "stylesheet" href = "../css/reset.css">
    <style>
        .布局之路 {
            width: 200px;
            height: 100px;
            border: 5px solid black;
        }
        .bujuzhilu {
            width: 200px;
            height: 100px;
            border: 1px solid blue;
        }
        .the-road-of-layout {
            width: 100px;
            height: 80px;
            border: 3px solid red;
        }
    </style>
</head>
<body>
    <div class = "box">
        <div class = "布局之路">第一个 div</div>
        <div class = "bujuzhilu">第二个 div</div>
        <div class = "the-road-of-layout">第三个 div</div>
    </div>
</body>
</html>
```

显示效果如图 3.11 所示。

对于新手，在进行网站开发时，很容易陷入命名的泥潭当中，虽然各种命名都能够实现，但是在代码量与可维护性方面都存在一定的问题。

使用中文命名，有可能造成编码解读的问题，使用汉语拼音，相对来说比较冗长，而 box1、box2、box3 等让工程师看到类名之后并没有解读出代码段对应的是哪些模块，过长的英语单词会增加代码量，书写以及修改起来都不是很方便，很容易写错，从而出现不必要的问题。

(3) 为何在本书中出现了 con1、con2 等类似的类名？

本书的部分案例使用了 con1、con2 等命名方式，其主要原因是在书写 demo(代码案例)，本身标签没有语义性，而在实际的网站开发当中，每个模块都有各自的语义性，不能使用此类命名。因此，无论是本书还是自己进行代码的编写，请区分开测试 demo 与正规开发。


**图 3.11　各类 CSS 类名效果图**

(4) 关于下画线、连字符、驼峰。

① 三种不同的书写方式。

在命名时,如果需要使用到多个英文单词,书写方式有三种。

假设当前需要为一个列表的标题命名。列表的英文单词是 list,标题的英文单词是 title(可以缩写为 tit)。

书写方式 1:

```
.list-tit。该方法使用连字符进行分隔。
```

书写方式 2:

```
.list_tit。该方法使用下画线进行分隔。
```

书写方式 3:

```
.listTit。该方法叫作小驼峰,也就是从第二个单词(含第二个单词)之后的每个单词首字母大写。
```



② 书写方式的优劣势。

首先,并不建议小驼峰的书写方法,在 JavaScript(行为代码)当中涉及 id 的命名,id 的命名规则为“驼峰命名法”。为了将 id 与类名区分开来,并不会选择驼峰的方式命名类名。

对于连字符和下画线,行业中各有说辞,两者各自有各自的优势。

含义方面:连字符相对更好。从代码含义上来说,连字符是将几个独立单词连接到了一起;下画线,只是在视觉上分隔单词,而在代码含义上来说,类名整体是一个完整的单词。

书写方面:下画线会更方便。从代码的编写方面来说,在编辑器中双击类名,如果是下画线连接的类名,能够选中整个类名;如果是连字符连接的类名,只能够选择到其中的一个类名,如图 3.12 所示。

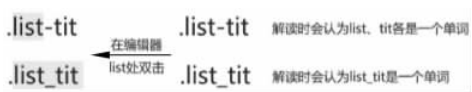



图 3.12　连字符与下画线的区别

③ 连字符、下画线,何去何从?

在真正的开发当中,这两者都可以使用。在当前行业当中,有些公司推荐使用连字符,有些公司推荐使用下画线。根据自己所在公司,与团队成员保持使用的一致性即可。个人在此更加推荐连字符,相对语义性会更好一些。

## 3.6　CSS 样式重置

### 3.6.1　什么是样式重置

HTML 与 CSS 语言,并不是同期出现的,HTML 要出现的更早一些。那么,在原来没有 CSS 语言的时候,如何控制网页当中的样式呢?

HTML 的发明者在发明标签之时,为这些标签设置了一些样式,这些样式就被称为标签的“默认样式”,“清除标签的默认样式”称为“样式重置”。

### 3.6.2 为何需要样式重置

#### 1. 防止默认样式对 CSS 书写带来的不便

随着时代的发展，CSS 语言的出现，让开发工程师完全可以通过 CSS 语言控制各个标签的样式，那么之前标签的默认样式反而成了“累赘”。

例如，书写一个 h1 标签，希望它的外边距是 0px，但是默认情况下 h1 元素存在外边距，此时还需要额外设置一行代码，如图 3.13 所示。

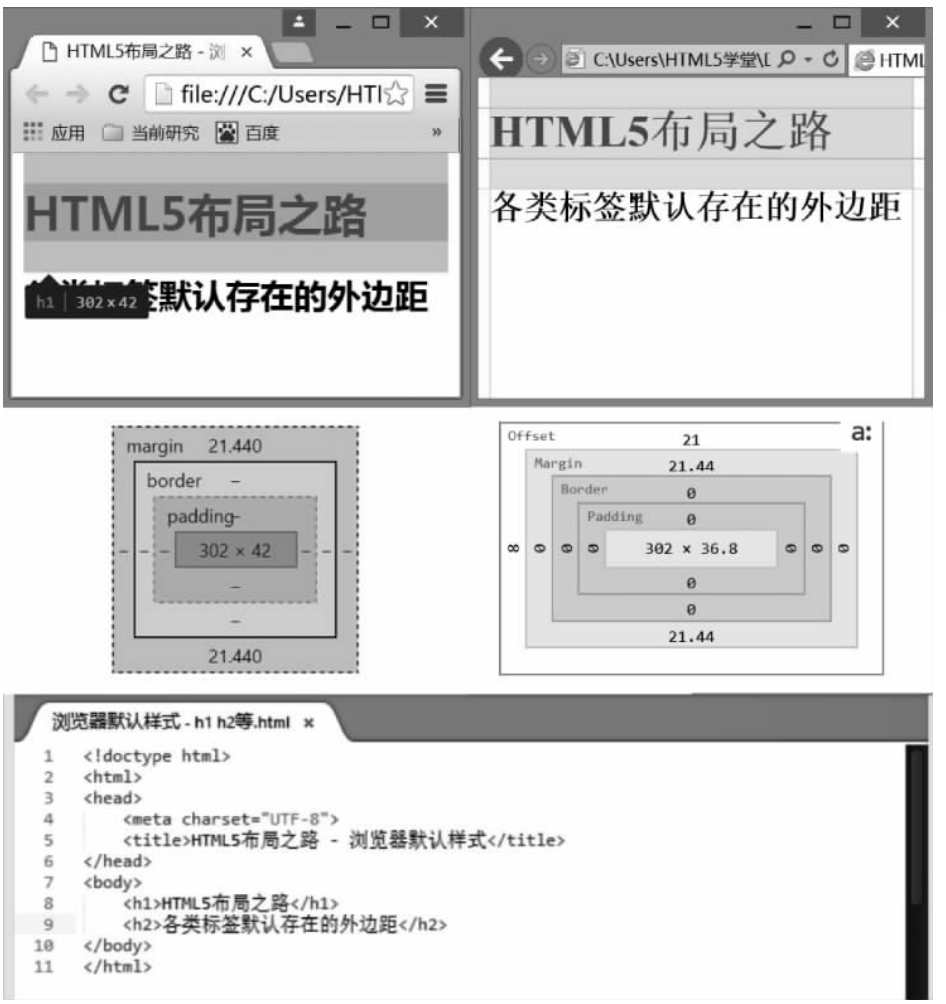



**图 3.13 <h1>在各个浏览器中的默认样式**

#### 2. 保证网页在各个浏览器表现的一致性

浏览器中样式的不同表现，也是“清除默认样式”的一个重要因素。在 IE、谷歌、火狐等不同的浏览器中，同样的标签默认样式并不相同，如图 3.14 所示。作为开发工程师，必须保证网页在各个浏览器中表现的一致性，因此，需要在开发之前清除掉这种不同的默认样式。


**图 3.14　各个浏览器默认样式有所不同**

## 3.6.3　样式重置文件

### 1. 样式重置文件内容

下列代码即清除浏览器各个标签(比较常用的大部分标签)默认样式的代码。

```
@charset 'utf-8';
html{color:#000;background:#FFF;font-family:'Microsoft YaHei',sans-serif,Arial;}
body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,code,form,fieldset,legend,input,button,
textarea,p,blockquote,th,td,strong{padding:0;margin:0;font-family:'Microsoft YaHei',sans
-serif,Arial;}
table{border-collapse:collapse;border-spacing:0;}
img{border:0;}
a{text-decoration:none; color:#333; outline:none;}
a:hover{text-decoration:underline;}
var,em,strong{font-style:normal;}
em,strong,th,var{font-style:inherit;font-weight:inherit;}
li{list-style:none;}
caption,th{text-align:left;}
h1,h2,h3,h4,h5,h6{font-size:100%;font-weight:normal;}
input,button,textarea,select,optgroup,option{font-family:inherit;font-size:inherit;font
-style:inherit;font-weight:inherit;}
input,button,textarea,select{*font-size:100%;}
```

2. 下载方式

可以在本书提供的电子资料的案例当中，下载 reset.css 文件，然后使用上面讲到的“外部引入”的方式，将 reset.css 文件引入到 HTML 文件当中。

3. 引入样式重置文件后的 HTML 文件

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 引入重置文件后的 HTML 文件</title>
    <link rel="stylesheet" href="../css/reset.css">
</head>
<body>
    <!-- 此处书写网页内容 -->
</body>
</html>
```



**注意**：外部引入文件时，文件路径不要写错。

4. 样式重置文件内容相关解释

在重置文件当中的 CSS 代码，主要有两种功能。一类代码是用来清除掉标签的默认样式，另一类代码则是根据当前的网站制作需求，修改一些默认样式，以方便开发工程师进行网站代码的开发与书写。

关于具体文件当中的内容是什么含义，在学过所有的标签与样式之后，完全可以自己解读。当前请先理解“样式重置的重要性”以及“如何进行样式重置”这两点。

## 3.7 盒模型

### 3.7.1 生活中的“盒模型”——鱼缸

假设一个“卖鱼缸”的商家，当前新进了两个鱼缸，需要存放在库房当中。

库房存放基本需求(俯视状态，单位 cm)：

(1) 鱼缸大小：100×100。

(2) 包装纸箱容量：110×110。

(3) 包装纸箱厚度：5(4 个方向)。

(4) 放入纸箱与鱼缸之间泡沫：5(4 个方向)。

(5) 纸箱与墙壁、其他纸箱间隔：5(4 个方向)。

示意图如图 3.15 所示。

### 3.7.2 盒模型基本知识

生活中与代码的盒模型对比如图 3.16 所示。

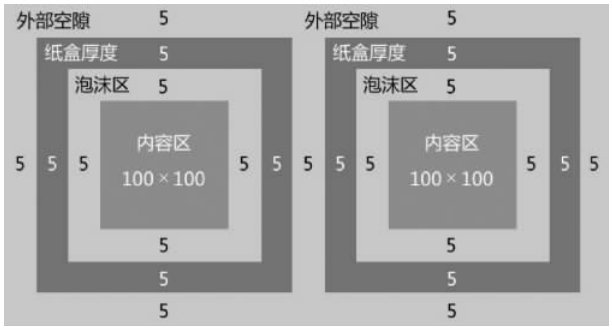



图 3.15　生活中的盒模型

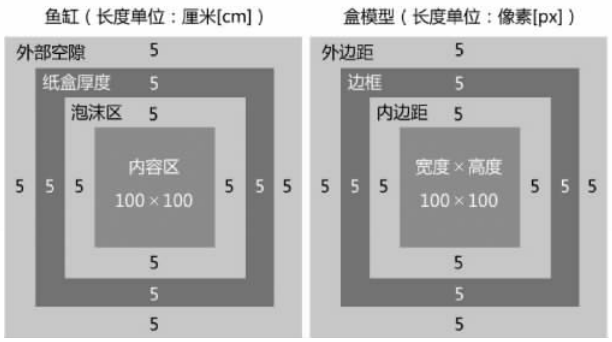



图 3.16　生活中与代码的盒模型对比图

在 HTML 当中，每个元素在浏览器中的解析，都可以被看作一个“盒子”，拥有盒子一样的外形和平面空间。完整的盒模型是由 width、height（宽度和高度构成—>内容）、border（边框）、padding（内边距）、margin（外边距）这几部分属性组成，如表 3.11 所示。

表 3.11　生活与代码中的盒模型对比

| 鱼　缸 | 盒　模　型 | 设 置 代 码 |
|---|---|---|
| 单位：cm | 单位：px | width：100px； |
| 鱼缸区 100×100 | width、height | height：100px； |
| 泡沫区 10cm | padding | margin：5px； |
| 边界区 2cm | border | padding：10px； |
| 外部空隙区 5cm | margin | border：2px solid ＃39f； |

## 3.7.3　盒模型——width 与 height 属性

盒模型的宽度与高度属性如图 3.17 所示。

**属性功能：**

width 用于设置元素的宽度；height 用于设置元素的高度。

**基本语法：**

```
width: 100px;
height: 100px;
```

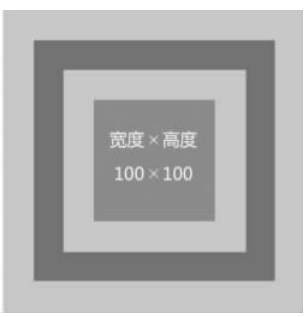



图 3.17　盒模型宽度与高度属性

**代码解析：**

设置元素宽度为 100 像素，高度为 100 像素。

属性值如表 3.12 所示。

**表 3.12　width 属性的属性值**

| 值 | 描　　述 |
|---|---|
| auto | 默认值。浏览器可计算出实际的宽度/高度 |
| length | 使用 px、em、cm 等单位定义宽度/高度 |
| % | 定义基于包含块(父元素)宽度的百分比宽度/高度 |
| inherit | 规定应该从父元素继承 width/height 属性的值 |

### 3.7.4　盒模型——margin 属性

盒模型的外边距属性如图 3.18 所示。

**属性功能：**

设置一个元素外边距的宽度。外边距，可以理解为当前元素与父级或其他兄弟级元素之间的距离。

**基本语法：**

```
margin: 5px 5px 5px 5px;
```

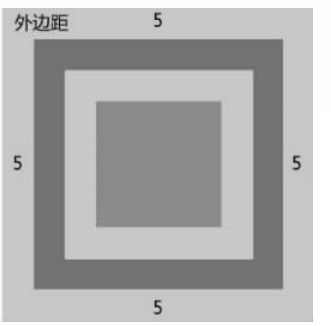



**图 3.18　盒模型的外边距属性**

**代码解析：**

设置元素外边距为 5 像素(4 个方向)。

属性值(数量：1～4 个)如表 3.13 所示。

**表 3.13　margin 属性的属性值**

| 值 | 描　　述 |
|---|---|
| auto | 浏览器计算外边距 |
| length | 规定以具体单位计的外边距值，如 px、em、cm 等 |
| % | 规定基于父元素的宽度的百分比的外边距 |
| inherit | 规定应该从父元素继承外边距 |

#### 1. 为何会有 1～4 个 margin 值

从图片当中能够看出来，一个标签在平面内存在 4 个方向，也就有 4 个外边距值与之相对应。而一个、两个、三个属性值的书写方法，只在于我们希望代码能够书写的更“少”一些。于是，就出现了这三种缩写的方法。

#### 2. 1～4 个 margin 属性值的含义

1) 4 个属性值

设置 margin 为 4 个值时，值与方向的对应顺序为“上-右-下-左”(从顶部，顺时针转下来)。

margin 有 4 个属性值的代码：

```
margin: 10px 5px 20px 0px;
```

**代码解析：**

设置元素的上外边距为 10px，右侧外边距为 5px，下外边距为 20px，左侧外边距为 0px。

2）三个属性值

设置 margin 为三个值时，值与方向的对应顺序为“上-左右-下”。

margin 有三个属性值的代码：

```
margin: 10px 20px 0px;
```

**代码解析：**

设置元素的上外边距为 10px，左侧和右侧外边距均为 20px，下外边距为 0px。

3）两个属性值

设置 margin 为两个值时，值与方向的对应顺序为“上下-左右”。

margin 有两个属性值的代码：

```
margin: 0 20px;
```



**代码解析：**

设置元素的上、下外边距均为 0px，左侧和右侧外边距均为 20px。

4）一个属性值

设置 margin 为一个值时，该值表示 4 个方向的外边距均设置为这个属性值。

margin 有一个属性值的代码：

```
margin: 20px;
```

**代码解析：**

设置元素的上、下、左、右 4 个方向外边距均为 20px。

### 3. margin 的分写

margin 的分写属性如表 3.14 所示。

**表 3.14　margin 分写属性**

| 属　性　名 | 属 性 含 义 |
| --- | --- |
| margin-left \| right \| top \| bottom | 左侧/右侧/顶部/底部的外边距 |

该方法书写起来代码较多，使用较少，通常都使用合写方法。

### 4. margin 的特殊应用

将元素的水平方向 margin 值设置为 auto，能够让块元素在父级当中水平居中。

**代码实例：**

```
<!doctype html>
<html>
<head>
```

```
    <meta charset = "UTF - 8">
    <title>HTML5 布局之路 - 元素在父元素中水平居中</title>
    <link rel = "stylesheet" href = "../css/reset.css">
    <style>
        .box {
            width: 100px;
            height: 100px;
            margin: 0 auto;
            background: #39f;
        }
    </style>
</head>
<body>
    <div class = "box"></div>
</body>
</html>
```

**代码解析：**

为一个 class(类名)为 box 的元素(div)，设置宽度 100 像素、高度 100 像素，上下外边距为 0 像素，左右外边距为 auto(自动)，背景颜色为蓝色(#39f)。



### 3.7.5 盒模型——padding 属性

盒模型的内边距属性如图 3.19 所示。

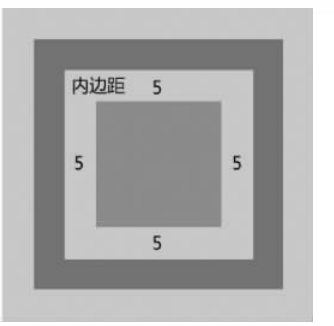



图 3.19 盒模型的内边距属性

**属性功能：**

设置一个元素内边距的宽度。外边距，可以理解为当前元素与元素边框之间的距离。

**基本语法：**

```
padding: 5px 5px 5px 5px;
```

**代码解析：**

设置元素内边距为 5 像素(4 个方向)。

属性值(数量：1～4 个)如表 3.15 所示。

表 3.15 padding 属性的属性值

| 值 | 描　述 |
|---|---|
| auto | 浏览器计算内边距 |
| length | 规定以具体单位计的内边距值，比如像素、厘米等。默认值是 0px |
| % | 规定基于父元素的宽度的百分比的内边距 |
| inherit | 规定应该从父元素继承内边距 |

**注意：**

(1) padding 值与 margin 值类似，都有 1～4 个值，计算的方式方法也相同。

(2) padding 值也有 padding-left 等 4 个方向的分写方法,分写方式使用很少。

(3) padding 值并没有负值,设置负值时相当于 0。

(4) padding 值的单位设置为百分比时,也是按照父级宽度进行计算的。

### 3.7.6　盒模型——border 属性

盒模型的边框属性如图 3.20 所示。

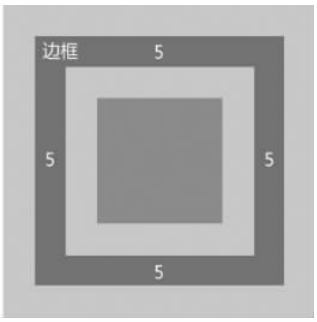



图 3.20　盒模型的边框属性

#### 1. 复合属性

border 是一个复合属性,一个边框包括边框的宽度、边框的颜色以及边框的类型。

边框宽度:以 px 等为单位。

边框颜色:十六进制颜色值或单词。

边框类型:solid(实线),dotted(点线),dashed(虚线)。

关于边框的具体属性值后面会进行详细讲解,当前请先了解以上这些知识,然后来看边框的"属性"。

#### 2. 分写方法

边框有 4 个方向,每个方向的边框都拥有三种属性(宽度、颜色和边框类型),因此就有了如表 3.16 所示的 12 种属性。

表 3.16　边框分写属性的属性值

| 属　性　名 | 属 性 含 义 |
|---|---|
| border-left \| right \| top \| bottom-width | 左侧/右侧/顶部/底部边框的宽度 |
| border-left \| right \| top \| bottom-style | 左侧/右侧/顶部/底部边框的样式 |
| border-left \| right \| top \| bottom-color | 左侧/右侧/顶部/底部边框的颜色 |

border 分写的代码:

```
border-left-width: 10px;
```

#### 3. 缩写方法

看到这里,有没有觉得上面的这种书写方法太冗余了?

于是,关于边框,就出现了如表 3.17 所示的这样几种缩写方法。

表 3.17　边框各属性合写的属性值

| 属　性　名 | 属 性 含 义 |
|---|---|
| border-width | 设置边框宽度 |
| border-style | 设置边框样式 |
| border-color | 设置边框颜色 |
| border-left \| right \| top \| bottom | 设置左侧/右侧/顶部/底部边框,需要包含边框宽度、边框样式、边框颜色 |
| border | 设置 4 个方向的边框宽度、边框样式、边框颜色 |

**注意：**

(1) border-width、border-style、border-color 均可以取 1～4 个值，属性值与方向的对应关系与“padding、margin”相同。

(2) border-left | top | right | bottom：需要包含边框宽度、样式、颜色三种属性，不同属性值之间使用空格分隔。

(3) border 与 border-left 等属性类似，均需要三种属性来组合而成，但有所不同的是，通过 border 设置的样式是应用于 4 个方向的。

**代码实例 1：**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - border 的属性合写 实例 1</title>
    <link rel="stylesheet" href="../css/reset.css">
    <style>
        .box {
            width: 100px;
            height: 100px;
            padding: 60px;
            border-width: 10px 5px;
            border-style: dotted solid dashed;
            border-color: #333 #f00 #0f0 #ff3;
            background: #39f;
        }
    </style>
</head>
<body>
    <div class="box">内容</div>
</body>
</html>
```



**代码解析：**

为类名为 box 的 div 设置如下样式：

上下边框宽度 10 像素，左右边框宽度 5 像素；

上边框样式为点线，下边框样式为虚线，左右边框样式为实线；

上边框颜色为深灰色(#333)，右侧边框颜色为红色(#f00)，下边框颜色为绿色(#0f0)，左侧边框颜色为黄色(#ff3)。

**代码实例 2：**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - border 的属性合写 实例 2</title>
```

```
    <style>
        .box {
            width: 100px;
            height: 100px;
            padding: 60px;
            border-left: 10px solid #f00;
            background: #39f;
        }
    </style>
</head>
<body>
    <div class = "box">内容</div>
</body>
</html>
```

**代码解析：**

为类名为 box 的 div 设置左侧边框；

边框宽度 10 像素，颜色红色，实线型边框。



### 4. border 属性选用原则

border(边框)所涉及的属性多达 20 种，在实际的开发当中，应该如何使用呢？

一句话来概括：越简单越好，尽可能使用缩写。

举个例子，当前要为一个元素的左侧、右侧、底部这三个方向均设置“10 像素 实线 红色”的边框，如表 3.18 所示。

**表 3.18　border 属性选用方法对比**

| 方法 1 代码与解析 | 方法 2 代码与解析 |
| --- | --- |
| 方法 1：<br>border-right：10px solid #f00；<br>border-left：10px solid #f00；<br>border-bottom：10px solid #f00； | 方法 2：(果断选择这种方法)<br>border：10px solid #f00；<br>border-top：0； |
| 代码解析：<br>使用 border-left 这三种合写方式设置样式，对于没有设置的 border-top，则默认没有边框 | 代码解析：<br>首先使用 border 将 4 个方向上均设置好边框，之后再将顶部边框清零 |

### 5. border 的属性值

在了解 border 的一系列属性之后，下面来具体说说属性值。

例如：border：10px solid #f00；

(1) 10px：边框宽度，此处可以使用 px，也可以使用 em 的相对度量单位，不允许为百分比，不允许为负值。border 属性的属性值如表 3.19 所示。

(2) solid：边框线型的一种，属于“实线”边框。边框类型除了“实线”之外，还包括点线(dotted)、虚线(dashed)。另外，在此罗列各类“线型”，如表 3.20 所示。

表 3.19　border 属性的属性值

| 值 | 描　　述 |
| --- | --- |
| thin | 定义细的边框 |
| medium | 默认。定义中等的边框 |
| thick | 定义粗的边框 |
| length | 允许自定义边框的宽度 |
| inherit | 规定应该从父元素继承边框宽度 |

表 3.20　边框样式的各类属性值

| 属性值 | 具 体 含 义 | 属性值 | 具 体 含 义 |
| --- | --- | --- | --- |
| none | 【常用】无边框 | double | 双线边框 |
| hidden | 隐藏边框 | groove | 3D 凹槽边框 |
| dotted | 【常用】点状边框 | ridge | 3D 垄状边框 |
| dashed | 【常用】虚线边框 | inset | 3D inset 边框 |
| solid | 【常用】实线边框 | outset | 3D outset 边框 |
| inherit | 从父级继承边框样式 | | |



（3）#f00：边框颜色，此处表示红色。关于颜色，可以采用多种方式方法来表示，如表 3.21 所示。

表 3.21　边框颜色的各类表示方法

| 代　　码 | 颜色含义 | 基本格式解释 |
| --- | --- | --- |
| #ff0000 | 红色 | #后面为三组十六进制的数字，第一组表示红色，第二组表示绿色，第三组表示蓝色。#ff0000 表示的是红色为 ff(16×15 + 15 = 255)，绿色和蓝色为 0 |
| #f00 | 红色 | 当第 1 和 2 位、第 3 和 4 位、第 5 和 6 位颜色值相同时，可以进行缩写，换言之，#f00 等价于#ff0000<br>对于#ffcc89 此类，有任意两位不同的颜色值，均不能缩写 |
| red | 红色 | 可以使用英语来表示颜色，国外很多开发工程师都会使用这种方法，会比较方便；但国内使用较少 |
| rgb(255, 0, 0) | 红色 | rgb 表示的是红绿蓝，括号中的三个值分别对应于红、绿、蓝，这三个值是十六进制计算出来的结果，取值范围为 0～255 |

## 3.7.7　盒模型的问题区

温馨提醒：对于此部分中的问题，大多数为实战类的问题，可以在编辑器中动手调试代码，得出基本结论，再看答案，学习方法和基本知识同等重要。

（1）margin：0 auto 中的 auto 是什么含义？

首先，margin：0 auto；这句代码当中，margin 有两个属性值，第一个属性值表示顶部和底部的外边距，第二个属性值表示左右的外边距。在 margin：0 auto；当中表示横向的外边距自动。此时，左右外边距的具体的值为：（父级元素内容区宽度 - 含边框内边距的当前元素宽度）/2。

**代码实例：**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - div 元素 margin 设置 auto 值</title>
    <link rel="stylesheet" href="../css/reset.css">
    <style>
        .box {
            width: 580px;
            height: 250px;
            padding: 0 20px;
            border: 5px solid black;
        }
        .con {
            width: 200px;
            height: 200px;
            margin: 0 auto;
            border: 2px solid black;
        }
    </style>
</head>
<body>
    <div class="box">
        <div class="con"></div>
    </div>
</body>
</html>
```

margin 水平方向设置为 auto 时的解析情况如图 3.21 所示。

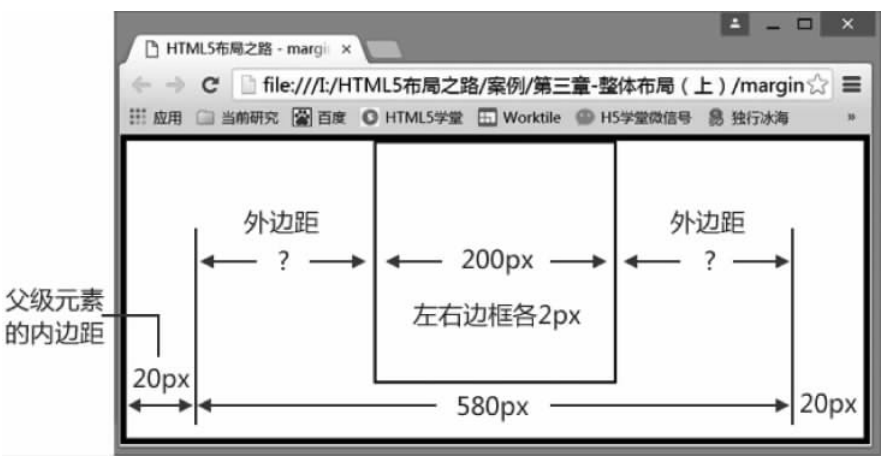



**图 3.21　margin 水平方向设置为 auto 时的解析情况**

此时子元素(类名为 con 的 div)外边距值是多少呢?

横向 margin 值＝ 580px－200px－2px×2＝376px。

左右 margin 会平分横向剩余的 margin 值，即各为 188px。

(2) 必须设置 margin：0 auto;才能够实现水平居中吗？ margin：20px auto;之类的命令可以吗？

当然可以！元素水平居中，是由横向的 margin 值控制的，与纵向 margin 值无关。

(3) 元素水平方向居中，对元素有没有什么限制？

可以尝试将类名为 con 的 div 元素修改为 span 元素，即< span class＝"con">，之后调试一下效果。

**代码实例：**

```
<!doctype html>
<html>
<head>
    <meta charset = "UTF - 8">
    <title>HTML5 布局之路 - span 元素的 margin 值</title>
    <link rel = "stylesheet" href = "../css/reset.css">
    <style>
        .box {
            width: 580px;
            height: 250px;
            padding: 0 20px;
            border: 5px solid black;
        }
        .con {
            width: 200px;
            height: 200px;
            margin: 0 auto;
            border: 2px solid black;
        }
    </style>
</head>
<body>
    <div class = "box">
        <span class = "con"></span>
    </div>
</body>
</html>
```

此时，显示效果如图 3.22 所示，能够发现，span 元素不仅不支持宽高等属性，也不能够在父级元素当中水平居中。

在此可以得出一个基本结论：对于 div 等块状元素，能够通过 margin：0 auto;实现在父级元素中的水平居中，对于 span 等行元素，不能够通过 margin：0 auto;实现在父级元素中的水平居中。

目前只需要使用 div 元素实现布局，对于块元素和行元素，在后面才会逐渐使用到，详见第 5 章。

(4) 如果不为 div 元素设置宽度，但设置了 margin:0 auto;会怎么样？

当 div 没有设定固定宽度时，默认这个 div 会占据父级内容区的 100%。可以认为

图 3.22　span 元素不能够通过 margin：0 auto；水平居中

margin:0 auto;的代码是生效的，只是没有剩余的区域留给 margin 值。

(5) margin 值能否设置为负值?

margin 值可以为负值，而且 margin 负值在模块布局当中也会有很大的用途，在第 6 章当中会详细介绍 margin 负值的应用。在这里，请明确“margin 值可以设定为负值”，如图 3.23 所示。

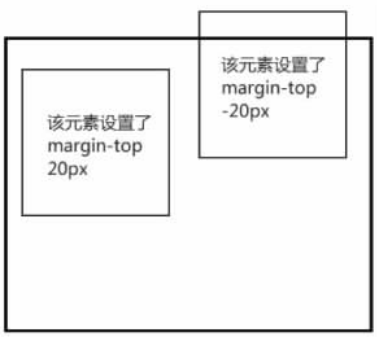



图 3.23　margin 设置负值时的效果

(6) 设置边框宽度、颜色、样式中的任意两种，会是什么效果?

设置不同边框属性时的边框效果如表 3.22 所示。

表 3.22　设置不同边框属性时的边框效果

| 设置编号 | 边框宽度 | 边框颜色 | 边框样式 | 最 终 效 果 |
|---|---|---|---|---|
| 001 | 设置 | 不设置 | 不设置 | 无效果 |
| 002 | 不设置 | 设置 | 不设置 | 无效果 |
| 003 | 不设置 | 不设置 | 设置 | 有边框效果；默认为黑色，3 像素边框 |
| 004 | 设置 | 设置 | 不设置 | 无效果 |
| 005 | 设置 | 不设置 | 设置 | 有边框效果；默认边框颜色为黑色 |
| 006 | 不设置 | 设置 | 设置 | 有边框效果；默认为黑色，3 像素边框 |

**基本结论**：默认的边框颜色为黑色，默认边框粗细为 3 像素(谷歌、火狐、IE8＋等浏览器下为 3 像素，IE7-浏览器中默认粗细为 4 像素)，必须设置“边框样式”属性，才能够触发边框效果。

(7) border、padding、margin 的缩写是否一致?

很多人在最初学习 HTML 时，会混淆盒模型三种属性的缩写。border、padding、margin 具有如下特点。

① border、padding、margin 命令均为相应属性的缩写；

② 内外边距的属性，仅存在像素大小这种属性值；

③ border 属性，包含宽度、样式、颜色三种类型的属性值；

④ border 是一个复合属性，border-width、border-color、border-style 的缩写方式和 margin、padding 相同。

(8) border：0;与 border：none;的区别。

border：0;表示设置边框,但是边框的宽度为 0,此时浏览器会正常渲染元素的边框效果,会占用内存空间；并且,所有的浏览器均能够正常使用。

border：none;表示不设置边框,浏览器不会进行任何渲染,不会占据内存空间；IE6、7不兼容。

(9) 盒模型单位如何选择?

当前,前端开发(或 HTML5 开发)当中,主要有桌端(也叫 PC 端)和移动端两个平台,桌端开发的就是台式计算机、笔记本使用的网页,而移动端就是手机使用的网站与页面。

通常,在桌端,像素(px)使用居多,如果涉及响应式、自适应等布局时会考虑使用百分比这种相对度量单位。在移动端,百分比、em、rem 这种相对度量单位使用居多,而像素(px)这种绝对度量单位使用较少。

(10) 盒模型大小与元素实际宽高值(width、height)并不相同。

这个概念至关重要,盒模型大小与元素的实际宽高,并不相同!

盒模型的宽度 = 左右外边距 + 左右边框 + 左右内边距 + width

盒模型的高度 = 上下外边距 + 上下边框 + 上下内边距 + height



(11) 使用背景颜色而非边框标识元素的原因。

在上面第 10 个问题当中能够看出,边框也是构成盒模型的宽度和高度的一部分,如果最初使用边框进行了一个 div 区域的标识,那么在实现模块具体操作之后,这个边框是需要被删除的,此时如果仅删除了边框而没有调整 width 和 height 的大小,就会导致盒模型大小发生变化。一旦开发时忘记调整或调整错误(哪怕是 1 像素的误差),都有可能引起布局错乱。

如果采用背景颜色,并不会占据盒模型的空间大小,再实现页面之后将背景颜色删掉,不需要针对当前元素进行任何其他操作,相对方便简单。

(12) 纵向外边距叠加。

当两个块状元素均设置纵向外边距时,第一个元素的下外边距和第二个元素的上外边距有可能会重合,此时,在默认情况下会出现外边距叠加的现象。

**代码实例：**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - 纵向外边距叠加</title>
    <link rel="stylesheet" href="../css/reset.css">
    <style>
        .wrap {
            width: 200px;
            height: 100px;
            margin: 50px;
            background: #39f;
        }
    </style>
</head>
<body>
```

```
    <div class = "wrap"></div>
    <div class = "wrap"></div>
</body>
</html>
```

显示效果如图 3.24 所示。

**代码解析：**

第一个 div 有 50 像素的下外边距，第二个 div 有 50 像素的上外边距，此时，两者的外边距重合，只保留 50 像素（如果两个 div 的外边距值不同，则按照较大的那个值来进行计算）。

原因以及解决办法：

之所以产生这个问题，原因在于最早的段落设置。

最初 HTML 中的 p 标签都存在默认的上下外边距，为的是和别的元素之间产生一定的距离。但是用在段落当中时，多个 p 上下排列，每两个 p 元素之间的距离就变得有些大了（上面 p 元素的下外边距＋下面 p 元素的上外边距），出于这样的考虑，浏览器解析设置了“纵向外边距叠加”的规则。

解决这个问题并不难，可以为元素设置浮动，纵向外边距就不再叠加（浮动的知识在第 4 章进行讲解）；另外一种方法则是规避掉这个问题，即为元素统一设置上或下，某一个方向的外边距，这样就不会出现重叠的现象和问题了。

（13）父子之间用 padding，兄弟之间用 margin？

行业中有这样一句话：父子之间用 padding，兄弟之间用 margin。这句话具体是什么意思呢？

先来看一个实例，模块布局的需求如图 3.25 所示。

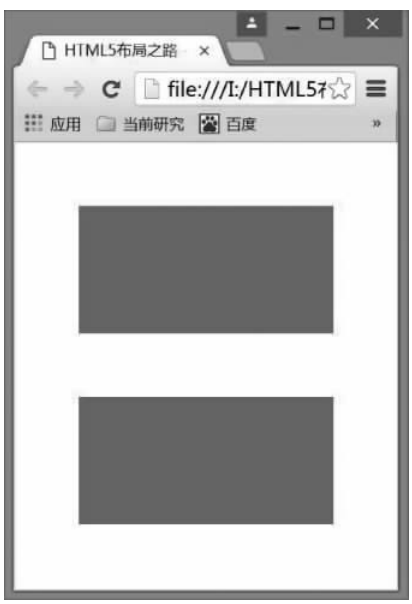



图 3.24　纵向外边距叠加

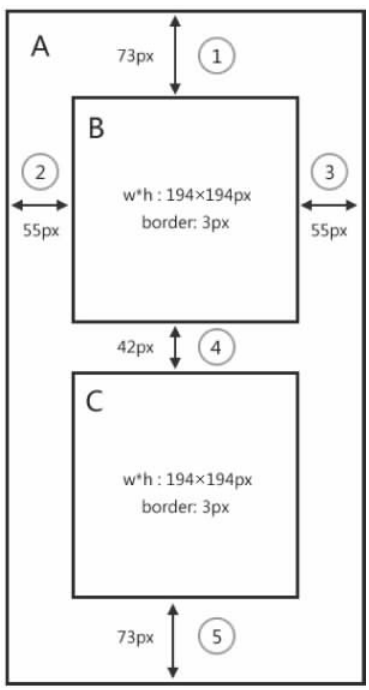
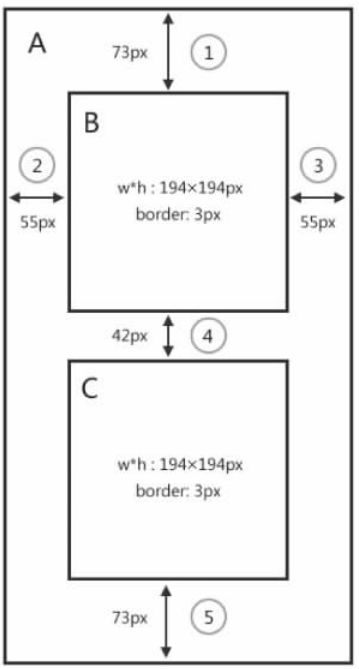



图 3.25　内外边距的选用需求图

图中外部一个 div(A),里面包含两个 div(B 和 C)。

A、B、C 之间的间距主要包括 1～5 这 5 个地方。其中,①、②、③、⑤均为父子元素之间的间距,④为两个同级元素(兄弟元素)之间的间距。

图 3.26 中,第一套方案遵循了“父子内边距、兄弟外边距”的理念;第二套方案中采用的都是外边距。




方案1

①②③⑤,父子元素之间的空隙,均使用padding值撑开。④,兄弟级之间的空隙,使用margin值撑开

样式代码如下:

```
A元素
width: 200px;
padding: 73px 55px;
border: 3px solid #555;

B元素
height: 194px;
margin-bottom: 42px;
border: 3px solid #555;

C元素
height: 194px;
border: 3px solid #555;
```

方案2

①~⑤,均使用margin值撑开

样式代码如下:

```
A元素
width: 310px;
border: 3px solid #555;

B元素
width: 194px;
height: 194px;
margin: 73px 55px 42px;
border: 3px solid #555;

C元素
width: 194px;
height: 194px;
margin: 0px 55px 73px;
border: 3px solid #555;
```

**图 3.26　内外边距的选用方案对比图**

对比两套方案,不难看出,第一套方案的代码要精简很多,相对操作的属性比较少,另外,在 IE6 等浏览器中,第一套方案也不会出现什么问题(第二套方案当中,会触发 IE6 的双倍边距 bug)。外边距在布局当中比较容易触发一些布局问题。

(14) 盒模型可能引发的布局错乱。

当子元素的盒模型大小超出了父级元素的内容区大小,会引发问题。一起来看如下这个例子。

**结构代码:**

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

**样式代码:**

```
.box {
    width: 200px;
    height: 200px;
```

```
}
.con {
    width: 180px;
    height: 180px;
    margin: 0 auto;
    padding: 10px;
    border: 1px solid #f00;
}
```

**代码解析：**

“类名为 con 的 div”是“类名为 box 的 div”的子级。

父级 div 的宽度只有 200 像素。

子级的实际宽度为：180＋10＋10＋1＋1＝202 像素。

子级的宽度已经超出父级宽度，在这种情况下，在某些浏览器下，很容易引发布局错乱等问题。

在谷歌等容错性比较强的浏览器当中，虽然并不会布局错乱，但是依旧建议修改，防止引发其他问题。

图 3.27　4 个三角形组成的矩形效果

(15) 趣味的边框样式。

请思考如图 3.27 所示效果的制作方法。

一个矩形，包含 4 个三角形，4 个三角形颜色各不相同。此处可以使用边框来实现这个效果，通过这个效果，也能够更好地理解边框的显示样式。

**代码实例：**

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5 布局之路 - border 的特殊效果</title>
    <link rel="stylesheet" href="../css/reset.css">
    <style>
        .box {
            width: 0px;
            height: 0px;
            border-width: 50px;
            border-style: solid;
            border-color: red black green yellow;
        }
    </style>
</head>
<body>
    <div class="box"></div>
</body>
</html>
```

**代码解析：**

并不为元素设置宽高，所有的部分均由边框来组成，4 个方向各设置了 50 像素不同颜色的实线边框。

边框是从两个角，45°延伸，最后两条线的延伸线会交叉成一点，如图 3.28 所示。当为 div 元素设置了宽度和高度时（宽高各 100 像素），效果会更加明显。

**图 3.28　边框的交叉点**