第一篇

HTML与HTML 5

第 1 章

◂ 网页的架构基础 ▸

本书开篇先介绍 HTML 网页的架构基础。相信读者对于 HTML 网页一定很熟悉，那么是否清楚关于 HTML 的准确定义呢？HTML（Hyper Text Markup Language，超文本标记语言）规定了一系列的语法规则，用来表示比“文本”更丰富的内容，譬如链接、表格、图形、照片、动画、音视频等。

市面上的主流浏览器（包括 Google Chrome、Mozilla Firefox、Microsoft Internet Explorer & Microsoft Edge、Opera 及其他第三方浏览器）均可运行基于 HTML 定义的语法并用来显示 HTML 网页文档。因此可以讲，我们日常在互联网上浏览的绝大部分网页都是基于 HTML 语法编写而成的，HTML 实际上就是一个互联网语言标准。

本章主要包括以下内容：

- HTML 网页的基础构成
- 基底网址标记 base 元素
- 用 CSS 定义网页的样式
- 使用脚本让网页动起来
- 利用 noscript 元素检测网页对脚本的支持
- 在网页代码中添加注释，方便后期代码的阅读
- 判断浏览器对 HTML 5 属性的支持

1.1 HTML 网页的基础构成

我们常见的网页都是 HTML 网页，HTML 既然是一种标准，网页就有基础的结构形式。本节的目的就是让读者了解一个 HTML 网页的基础构成。

1.1.1 从一个空白的 HTML 网页说起

一个标准的 HTML 网页必须包括一些固定的标签，譬如 DOCTYPE 标签、html 标签、head 标签和 body 标签等。这些固定元素构成一个 HTML 网页的基础骨架，倘若缺少其中任何一项，都有可能造成 HTML 网页运行时出现不可预知的错误。所以，读者首先要清楚 HTML 网页的基础构成，并理解每个标签的含义与作用。

【示例 1-1】　空白的 HTML 网页

市面上有很多款创建、编辑、运行 HTML 网页的工具（较轻量级的有 EditPlus、UltraEdit、Sublime 等，较重量级的有 Eclispe、Visual Studio、Dreamweaver 等，至于选择哪款，主要看个人喜好），本书选择 EditPlus 编辑工具自动生成一个基本的 HTML 网页框架（参见源代码 chapter01/ch01-htmlcomp-init.html 文件），主要代码如下：

```
01  <!doctype html>
02  <html>
03  <head>
04      <meta charset="UTF-8">
05      <meta name="Generator" content="EditPlus®">
06      <meta name="Author" content="">
07      <meta name="Keywords" content="">
08      <meta name="Description" content="">
09      <title>Blank HTML Page</title>
10  </head>
11  <body>
12  <!-- 添加文档内容 -->
13  </body>
14  </html>
```

在这段空白的 HTML 网页代码中，我们看到上文中提到的几个标签均使用了，如第 01 行中的<!doctype>标签、第 02 行中的<html>标签、第 03 行中的<head>标签以及第 11 行中的<body>标签。

注意第 10 行、第 13 行与第 14 行中再次使用了</head>、</body>和</html>标签，不同之处是增加了一个“/”斜杠字符。HTML 标准将第 02 行、第 03 行和第 11 行的标签定义为开始标记，而将第 10 行、第 13 行与第 14 行的标签定义为结束标记。其中，第 02 行的<html>标签与第 14 行的</html>标签相互对应，称为一组标记。同理，第 03 行与第 10 行为一组标记，第 11 行与第 13 行为一组标记。读者可以将一组标记之间的内容理解为一个整体，代表这个标签所定义的内容，完成一组统一的功能。

代码 ch01-htmlcomp-init.html 实际上没有定义任何实际内容，在浏览器中运行后显示的是一个空白的页面，如图 1.1 所示。

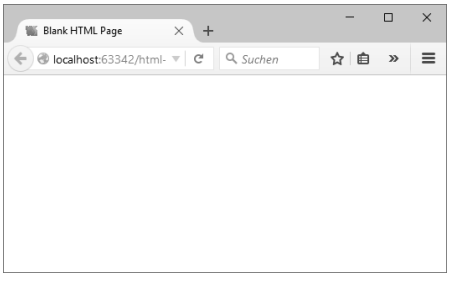

图 1.1　空白的 HTML 网页

1.1.2 通过网页中的 DOCTYPE 标签识别文档类型

DOCTYPE 标签是一种标准通用标记语言的文档类型声明，其存在的意义是通知标准通用标记语言解析器应该使用什么样的文档类型定义（Document Type Definition，DTD，由 W3C 标准化组织定义）来解析文档。

回顾 1.1.1 小节中的代码段，第 01 行使用的<!doctype html>标签就是对文档类型的声明，根据具体定义可以知道声明 ch01-htmlcomp-init.html 文件为一个 HTML 网页。

DOCTYPE 标签在使用时不区分大小写。

这里有几点是读者在使用 DOCTYPE 标签时必须注意的：

（1）<!doctype>标签必须声明在 HTML 网页的第一行，位于<html>标签之前。

（2）严格意义上讲，<!doctype>不是一个 HTML 标签，而是定义浏览器运行 HTML 网页时使用哪个 HTML 版本来进行解释的指令。

（3）在 HTML 5 标准下，必须使用<!doctype html>这样的方式来定义，可见 ch01-htmlcomp-init.html 文件是一个标准 HTML 5 网页。而 HTML 4.01 标准是基于 SGML（Standard Generalized Markup Language，标准通用标记语言）的，所以必须引用 DTD，这样浏览器才能正确地显示 HTML 网页内容。

<!doctype>标签声明时是没有结束标记的。

这里比较复杂的是 HTML 4.01 标准，其规定了 3 种文档类型（DTD），分别为 Strict、Transitional 和 Frameset，具体的声明方法如下：

（1）HTML Strict DTD

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

如果想避免表现层的混乱，那么建议使用此类型，并与层叠样式表（CSS）配合使用。

（2）HTML Transitional DTD

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

如果使用了不支持层叠样式表（CSS）的浏览器而又不得不使用 HTML 网页来展现内容，那么可以使用此类型。

（3）Frameset DTD

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" " http://www.w3.
```

```
org/TR/html4/frameset.dtd">
```

此类型用于带有框架的 HTML 网页，除了使用<frameset>标签替代<body>标签外，也可以用 Frameset DTD 替代 Transitional DTD。

这里顺便提一句，DOCTYPE 标签除了可以声明 HTML 网页外，还可以用于声明其他类型的文件。譬如，<!doctype math>用于声明数学标记语言，<!doctype tmx>用于声明翻译存储交换标记语言，<!doctype wml>用于声明无线标记语言，等等。可见 DOCTYPE 标签的内容是非常丰富的，感兴趣的读者可以参考相关文档进一步深入了解 DOCTYPE 标签的内容。

1.1.3　html 标签声明这是一个网页

html 标签用于通知浏览器这是一个 HTML 网页。一般来讲，<html>与</html>标签限定了 HTML 网页的开始标记和结束标记，标记之间的内容是 HTML 网页的头部和主体。至于头部和主体后面马上会介绍，也就是读者熟悉的 head 标签和 body 标签。

HTML 网页中的内容通常需要放置在<html>标签中。在 HTML 网页的头部可以放置标题、兼容性、语言、字符格式、关键字和描述等重要信息，而 HTML 网页需要向用户展示的具体内容可以统统放置在主体中。

如下所示为一个关于 html 标签的代码段（参见源代码 chapter01/ch01-htmlcomp-html.html 文件）。

【示例 1-2】　html 标签

```
01  <!DOCTYPE html>
02  <html lang="en">
03  <head>
04      <!-- 添加文档头部内容 -->
05  </head>
06  <body>
07      <!-- 添加文档主体内容 -->
08  </body>
09  </html>
```

在这段 HTML 代码中，第 02 行与第 09 行就是对<html>和</html>标签的使用。读者注意到在<html>标签中添加了一个字段 lang="en"，一般称其为属性字段，lang 关键字代表规定标签内容的语言，"en"关键字代表英文，如果读者想定义中文，那么可以使用"zh"关键字。

1.1.4　head 标签定义网页的头部

head 标签用于定义 HTML 网页的头部，可以说 head 是所有头部标签的容器。在 head 标签中可以实现描述元信息（meta）、添加层叠样式表（CSS）、引用外部脚本文件（JavaScript，简称 JS）、定义 HTML 网页标题以及与其他文档的关系等功能。另外，绝大部分在 HTML 网页头

部定义的数据都不会作为页面的具体内容显示给用户。

head 标签位于 HTML 网页的头部，以<head>作为开始标记，并以</head>作为结束标记。一般来讲，head 中包含 meta、base、link、script、title 等常用标签，这些标签详细说明如下：

- meta 标签可以定义的内容十分广泛，譬如 HTML 网页介绍、HTML 网页关键字、HTML 网页编码、页面作者、自动跳转定义以及 robots 协议等内容均可以放置在其中。
- base 标签是定义 HTML 网页默认打开方式的声明。
- link 标签用于定义目标文件链接，包括对外部层叠样式表（CSS）文件的引用、对外部脚本（JS）文件的引用以及对 favicon.ico 图标的引用等。
- script 标签既可以用于引入外部脚本（JS）文件，也可以定义嵌入 HTML 网页内部的脚本代码。
- style 标签用于定义直接嵌入网页的层叠样式表（CSS）代码。
- title 标签用于定义 HTML 网页的唯一标题。

下面向读者介绍使用 head 标签的方法（参见源代码 chapter01/ch01-htmlcomp-head.html 文件）。

【示例 1-3】 head 标签

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="zh-cn" />
    <meta name="author" content="king">
    <meta name="revised" content="king,10/5/2019">
    <meta name="generator" content="EditPlus">
    <meta name="description" content=" HTML 网页 head 标签使用">
    <meta name="keywords" content="HTML, CSS, JavaScript">
    <meta http-equiv="Refresh" content="1;url=http://localhost:8080" />
    <base href="http://localhost:8080" target="_blank" />
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <style type="text/css">
        h1 {font: bold 24px/2.4em arial,verdana;}
        h3 {font: bold 18px/1.8em arial,verdana;}
        p {font: italic 13.5px/1.2em arial,verdana;}
    </style>
    <script type="text/javascript">
        document.write("<h1>HTML 之 head 标签</h1>");
    </script>
    <title>HTML 网页标题</title>
</head>
```

```
24  <body>
25      <!-- 添加文档主体内容 -->
26      <h3>header</h3>
27      <p>Paragraph.</p>
28      <!-- 添加文档主体内容 -->
29      <p>
30      <a href="ch01-htmlcomp-init.html">尝试在本窗口打开链接</a><br>
31      说明：正常应该在当前窗口打开该链接，但实际是在新窗口中打开了；<br>
32      原因就是在本 HTML 网页头部中，base 标签中设置了 target="_blank"属性。<br>
33      </p>
34  </body>
35  </html>
```

在关于 head 标签的这段 HTML 代码中，第 03～24 行就是使用 head 标签的方法。为了让读者全面了解 head 标签的使用方法，该代码段尽可能将 HTML 网页头部可能用到的标签全部包含进去。下面我们逐一对这些标签进行介绍。

第 04～11 行是对 meta 标签的使用，meta 是 HTML 网页头部的一个辅助性标签。meta 标签共有两个属性，分别是 http-equiv 属性和 name 属性。其中，http-equiv 属性相当于 HTTP 协议文件头的作用，通过该属性可以向浏览器回传数据信息，以帮助准确地显示网页内容，其属性值放置在与之对应的 content 属性中；而 name 属性主要用于描述网页，包括分类信息的内容以及便于搜索引擎 robots 协议查找的内容，其属性值同样放置在与之对应的 content 属性中。

第 04 行中的 http-equiv 属性定义为"Content-Type"，Content-Type 表示设定的显示字符集。本代码段中对应的 content 属性定义为"text/html; charset=utf-8"，表明本 HTML 网页设定的显示字符集为"utf-8"，为通用的 Unicode 编码格式（utf-8 编码支持中英文字符，比传统 gb2312 中文编码更通用）。

第 05 行中的 http-equiv 属性定义为"Content-Language"，Content-Language 表示给 HTML 网页设定的页面语言。本代码段中对应的 content 属性定义为"zh-cn"，表明本 HTML 网页设定的页面语言为简体中文（如果是繁体中文就为"zh-tw"，而"en-us"表示英语（美国））。

第 06 行中的 name 属性定义为"author"，author 表示网页作者。本代码段中对应的 content 属性定义为"king"。

第 07 行中的 name 属性定义为"revised"，revised 表示网页最后一次更改的作者及时间。本代码段中对应的 content 属性定义为"king,10/5/2019"。

第 08 行中的 name 属性定义为"generator"，generator 表示创建和编辑网页使用的工具软件。本代码段中对应的 content 属性定义为"EditPlus"。

第 09 行中的 name 属性定义为"description"，description 表示对网页功能、内容的相关描述，属于比较重要的一个 meta 属性。本代码段中对应的 content 属性定义为" HTML 网页 head 标签使用"。

第 10 行中的 name 属性定义为"keywords"，keywords 表示网页的关键词。本代码段对应的 content 属性定义为"HTML, CSS, JavaScript"。

第 11 行中的 http-equiv 属性定义为"Refresh"，Refresh 是一个非常有用的功能，可以对网页自动刷新并重定向指向新页面。本代码段中对应的 content 属性定义为"1;url=http://localhost:8080"，其代表两个内容，并使用分号进行分割，分号前面的数值 1 表示时间间隔为 1 秒，分号后面的 url 代表重定向链接地址，合在一起的含义就是在间隔 1 秒后刷新，重新跳转到新的链接地址。因为 Refresh 功能是在 HTML 网页头部定义的，所以在该页面初次打开后就会计算时间间隔并执行重定向操作。第 11 行测试后的结果如图 1.2 所示。

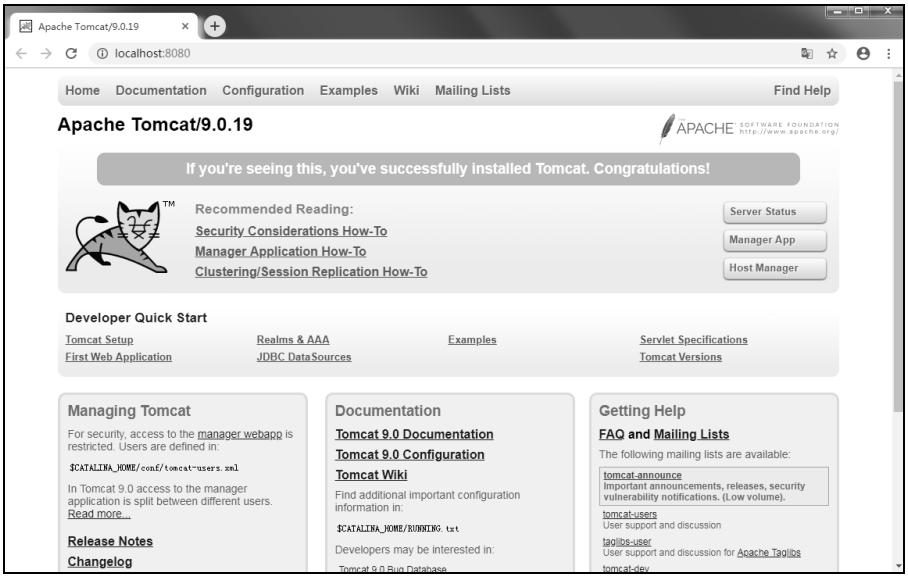

图 1.2　HTML 网页 Refresh 功能测试

从图 1.2 中可以看到，运行 ch01-htmlcomp-head.html 文件后，并没有看到 HTML 网页主体中定义的代码内容，而是很快跳转到图 1.2 显示的页面地址（http://localhost:8080），这个页面其实是本机安装的 Web 服务器——Tomcat 9.0.19 的主页。

第 12 行为对 base 标签的使用，base 的功能是为页面上的所有链接规定默认地址或默认目标。一般情况下，浏览器会从当前文档的 URL 地址中获取相应的数据来填写相对 URL 地址中的空白，而使用 base 标签则可以使用指定的 URL 地址来解析所有的相对 URL 地址。该 base 标签有两个属性，分别是 href 属性和 target 属性。其中，href 属性用于规定页面中所有相对链接的基准 URL 地址；而 target 属性用于指定在何处打开页面中所有的链接，其共有 4 个属性值：

- _blank 代表在新的窗口打开链接。
- _self 代表自身窗口，一般可以不用定义。
- _parent 代表在父窗口或超链接引用框架的框架集中打开链接。
- _top 表示会清除所有被包含的框架并将文档载入整个浏览器窗口。

在本行中，href 属性值定义为"http://localhost:8080"，target 属性值定义为"_blank"。

HTML 网页主体中的第 29～33 行是对第 12 行的测试，第 30 行定义了一个超链接，指向前面创建的一个 HTML 网页 ch01-htmlcomp-init.html，并且尝试在本窗口中打开该页面。下面我们

先行注销第 11 行的重定向功能，运行本页面的结果如图 1.3 所示。单击图 1.3 页面中定义的超链接，尝试在本窗口打开链接，测试后的结果如图 1.4 所示。

图 1.3　HTML 网页之 base 标签（1）

图 1.4　HTML 网页之 base 标签（2）

从图 1.4 可以看到，ch01-htmlcomp-init.html 页面在新窗口中被打开了，而且该页面似乎没有被正确地显示出来。在新窗口中被打开是因为第 12 行中 base 标签的 target 属性值定义为"_blank"，而页面没有正确显示是因为 ch01-htmlcomp-init.html 页面并没有被正确找到。这是什么原因造成的呢？我们看第 12 行中 base 标签的 href 属性值定义为"http://localhost:8080"，这样全部相对 URL 链接地址都会以"http://localhost:8080"为基地址，就是指向本机的 Tomcat 服务器，而 ch01-htmlcomp-init.html 页面并不存在于该服务器下，自然就无法找到该页面并正确显示。

第 13 行是对 link 标签的使用，link 用于定义文档与外部资源的关系，常见的用法就是定义外部链接层叠样式表（CSS）。本行中定义了保存在 css 目录下的外部样式表，文件名称为"style.css"。

第 14～18 行是对 style 标签的使用，script 用于定义直接嵌入网页的层叠样式表（CSS）代码。我们先不管第 15～17 行具体实现了什么功能，只要知道这 3 行是对<h1>、<h3>和<p>这 3 个 HTML 标签进行样式定义就可以了。在页面主体的第 26～27 行中，使用<h3>和<p>定义了两行页面内容，页面运行后的效果如图 1.5 所示。

第 19～21 行是对 script 标签的使用，script 用于引入外部脚本（JS）文件或定义内部脚本（JS）代码。其中，第 20 行定义了一行脚本代码，用于在页面主体输出一行文本信息，页面运行后的效果如图 1.6 所示。

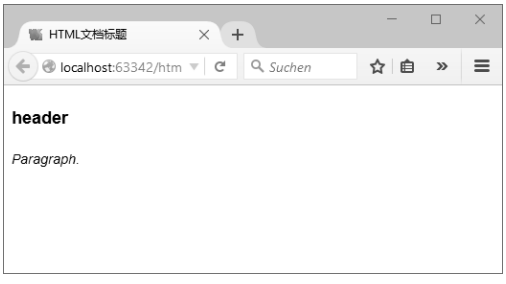

图 1.5　HTML 网页之 style 标签

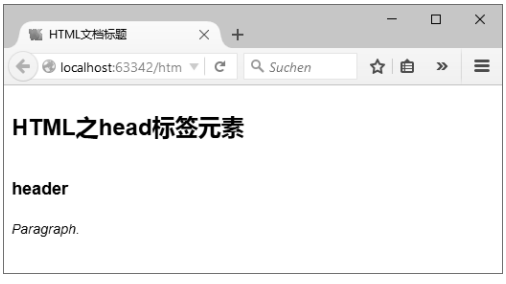

图 1.6　HTML 网页之 script 标签

另外，http-equiv 还有几个属性读者可能了解不多，但也是非常重要的功能，在这里简单介绍一下。

- Expires（期限）

功能描述：用于设定网页的过期时间，如果网页过期就必须连接服务器进行重新传输。

使用方法：<meta http-equiv="expires" content="Tue,31 Dec 2019 23:59:59 GMT">。

注意事项：必须使用 GMT 格式时间。

- Pragma（Cache 模式）

功能描述：禁止浏览器从本地计算机的缓存中访问 HTML 网页的内容。

使用方法：<meta http-equiv="Pragma" content="no-cache">。

注意事项：如果这样设置，用户就无法脱机浏览网页。

- Set-Cookie（Cookie 过期设置）

功能描述：如果网页过期，那么保存在本机的全部 Cookie 将被自动删除。

使用方法：<meta http-equiv="Set-Cookie" content="cookie-value=xxx; expires= Tue,31 Dec 2019 23:59:59 GMT; path=/ ">。

注意事项：必须使用 GMT 格式时间。

- Window-target（显示窗口的设置）

功能描述：强制 HTML 网页在当前窗口以独立页面方式显示。

使用方法：<meta http-equiv="Window-target" content="_top">。

注意事项：用来防止外部页面在框架里调用本页面。

1.1.5 body 标签定义网页的主体

body 标签用于定义 HTML 网页的主体，我们在网页上看到的全部内容（譬如文本、超链接、表单、照片、动画、视频等）都是放置在<body></body>标签中的。

body 标签自身可以包含几个属性，用于定义 HTML 网页主体的背景颜色、背景图片、文本颜色、链接颜色等。不过由于 CSS 样式表的存在，一般不建议直接在 body 标签自身中定义这些属性，比较标准的方法是全部定义在 CSS 样式表中。下面这几行就是直接在 body 标签自身中定义页面背景颜色（bgcolor="red"，设置为红色），读者可以参考：

```
<body bgcolor="red">
……
</body>
```

针对 body 标签比较常用的做法是定义事件属性，由于 body 标签地位的特殊性（其是

HTML 网页的主体标签，在页面初始化后 body 标签是最先加载的标签），我们可以在 body 标签定义 onload 事件属性。关于 onload 事件属性后面的章节会详细介绍，现在我们只需知道该事件属性可以完成页面加载后立即执行 JS 脚本的功能即可。

下面向读者介绍使用 body 标签的方法（参见源代码 chapter01/ch01-htmlcomp-body.html 文件）。

【示例 1-4】　body 标签

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之 body 标签</title>
</head>
<script type="text/javascript">
    /**
     * 自定义函数: init()
     */
    function init() {
        document.getElementById("id-content").innerHTML = "<p>初始化时动态添加页面内容!
     </p>";
    }
</script>
<body onload="init();">
    <!-- 添加文档主体内容 -->
    <h3>HTML 之 body 标签</h3>
    <div id="id-content"></div>
    <!-- 添加文档主体内容 -->
</body>
</html>
```

在关于 body 标签的这段 HTML 代码中，第 16 行是对 body 标签的使用。在 body 标签内，通过 onload 事件属性在页面初始化时触发了一个自定义 JS 函数（函数名称为 init()），该函数定义在第 11～14 行中，主要完成了对第 19 行中定义的一个层（div）标签进行动态更新内容的操作。页面运行后的效果如图 1.7 所示。

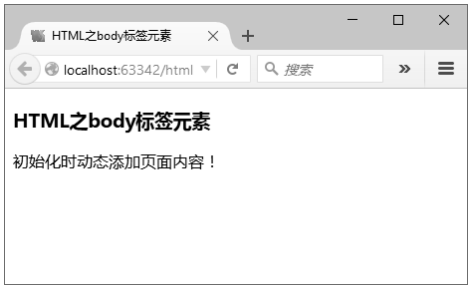

图 1.7　HTML 网页之 body 标签

在原始的 ch01-htmlcomp-body.html 文件中，body 主体内并没有定义“初始化时动态添加页面内容！”这条文本信息，不过从图 1.7 中的显示效果来看，这条文本信息没有经过用户任何操作却显示出来了，这表明该文本信息是在页面加载时动态添加进去的。

1.2 基底网址标记 base 标签

在 1.1 节向读者简要介绍过 base 标签，由于 base 标签在大型网站中比较常用，因此在这一节我们单独介绍 base 标签的用法。

一般情况下，浏览器会将当前页面的 URL 地址作为基地址，如果想改变所有链接默认的基地址，就可以使用 base 标签来实现。通过在 base 标签中设定新的基地址，浏览器将不再使用当前页面的 URL 地址作为基地址，所有的相对链接地址将使用新设定的基地址来进行解析。例如，HTML 网页中常用的 a 标签、img 标签、link 标签、form 标签等，其中的相对 URL 地址均会按照 base 标签中设定的新基地址进行解析。

下面向读者介绍使用 body 标签进行图片定位的操作方法（参见源代码 chapter01/ch01-htmlcomp-base-img.html 文件）。

【示例 1-5】 base 标签

```
01  <!DOCTYPE html>
02  <html lang="zh-cn">
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <base href="./images/" />
06      <base href="./images/png/" />
07      <title>HTML 之 base 标签</title>
08  </head>
09  <body>
10      <!-- 添加文档主体内容 -->
11      <h3>HTML 之 base 标签</h3>
12      <img src="img_eiffel200.png" alt="img_eiffel200.png" />
13      <img src="png_eiffel200.png" alt="png_eiffel200.png" />
14      <!-- 添加文档主体内容 -->
15  </body>
16  </html>
```

在这段 HTML 代码中，第 05 行与第 06 行就是对 base 标签的使用，在这两行的 body 标签内，通过 href 属性定义了两个基地址路径，这两个路径指向了我们预先保存的两组图片。

在第 12～13 行中，使用 img 标签在页面中显示了两张图片，我们预想的是第 12 行显示第 05 行定义的路径中的图片，而第 13 行显示第 06 行定义的路径中的图片。页面运行后的效果如图 1.8 所示。

从显示的效果可以看到，第 12 行定义的图片被正确显示出来了，而第 13 行定义的图片却无法正确显示（仅仅显示出图片文件名称），这是什么原因造成的呢？

我们尝试将源代码文件中的第 05 行与第 06 行调换先后顺序，页面运行后的效果发生了变化，如图 1.9 所示。

图 1.8　HTML 网页之 base 标签（1）

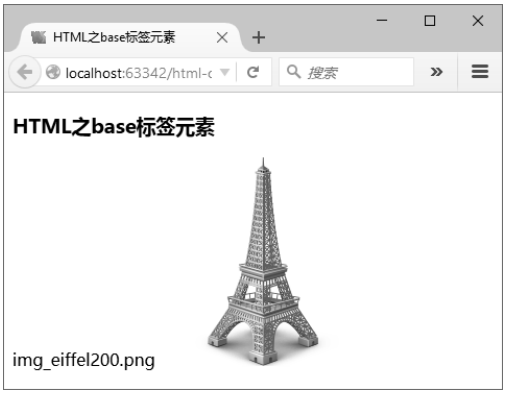

图 1.9　HTML 网页之 base 标签（2）

从图 1.9 中显示的效果可以看到，第 12 行定义的图片没有显示出来（同样仅仅显示出图片文件名称），而第 13 行定义的图片却正确显示出来了。由此可见，如果使用多个 base 标签定义基地址路径，那么只有最先定义的能生效，之后定义的会被浏览器全部无视，这点读者使用时要注意。

下面我们模拟一个包含 3 个 HTML 页面的简单网站，主要演示使用 body 标签进行基地址转换的操作方法。这是第一个页面的代码（参见源代码 chapter01/ch01-htmlcomp-base-url.html 文件），主要起到网站主页的作用。

【示例 1-6】　base 标签基地址操作（1）

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <base href="./url/" target="_blank" />
    <title>HTML 之 base 标签</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之 base 标签</h3>
    <p>当前地址：ch01-htmlcomp-base-url.html</p>
    <p>
```

```
   13       链接到：
   14    <a  href="ch01-htmlcomp-base-url-a.html">url/ch01-htmlcomp-base-url-
a.html</a>
   15       </p>
   16       <!-- 添加文档主体内容 -->
   17   </body>
   18   </html>
```

在这段 HTML 代码中，第 05 行使用 base 标签定义了一个基地址（"./url/"），该地址是 ch01-htmlcomp-base-url.html 文件所在目录下的一个子目录（名称为"url"），因此第 14 行中定义的超链接地址将会以第 05 行定义的基地址进行解析。同时，target 属性定义为"_blank"，这样该页面内所有超链接指向的页面文件均会在新窗口中打开。

下面是第二个页面的代码（参见本书源代码 chapter01/url/ch01-htmlcomp-base-url-a.html 文件），这个页面就是【示例 1-6】中第 14 行定义的超链接所引用的页面文件，该文件保存在子目录"url"中。

【示例 1-7】 base 标签基地址操作（2）

```
   01   <!DOCTYPE html>
   02   <html lang="zh-cn">
   03   <head>
   04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   05      <title>HTML 之 base 标签</title>
   06   </head>
   07   <body>
   08      <!-- 添加文档主体内容 -->
   09      <h3>HTML 之 base 标签</h3>
   10      <p>当前地址：url/ch01-htmlcomp-base-url-a.html</p>
   11      <p>
   12      链接到：
   13    <a  href="ch01-htmlcomp-base-url-b.html">url/ch01-htmlcomp-base-url-
b.html</a>
   14      </p>
   15      <!-- 添加文档主体内容 -->
   16   </body>
   17   </html>
```

在这段 HTML 代码中，没有使用 base 标签进行任何设定，这样第 13 行中超链接引用的页面文件将会在本窗口中打开。

下面是第三个页面的代码（参见源代码 chapter01/url/ch01-htmlcomp-base-url-b.html 文件），该文件就是【示例 1-7】中第 13 行定义的超链接所引用的页面文件。

【示例 1-8】　base 标签基地址操作（3）

```
01  <!DOCTYPE html>
02  <html lang="zh-cn">
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <base href="../" target="_blank" />
06      <title>HTML 之 base 标签</title>
07  </head>
08  <body>
09      <!-- 添加文档主体内容 -->
10      <h3>HTML 之 base 标签</h3>
11      <p>当前地址：url/ch01-htmlcomp-base-url-b.html</p>
12      <p>
13      返回到：<a href="ch01-htmlcomp-base-url.html">ch01-htmlcomp-base-
url.html</a>
14      </p>
15      <!-- 添加文档主体内容 -->
16  </body>
17  </html>
```

在这段 HTML 代码中，第 05 行使用 base 标签定义了一个基地址（"../"），该地址是 ch01-htmlcomp-base-url-b.html 文件所在目录的上一级目录（ch01-htmlcomp-base-url.html 文件所在的目录），因此第 13 行中定义的超链接地址将会以第 05 行定义的基地址进行解析，这样就可以跳转回主页面。同时，target 属性定义为"_blank"，这样该页面内所有超链接指向的页面文件均会在新窗口中打开。

最后，测试这个模拟的小网站。首先运行 ch01-htmlcomp-base-url.html 主页面文件，页面打开后的效果如图 1.10 所示。在图 1.10 中显示了当前页面的地址信息以及链接到新页面的超链接地址信息。

单击页面中的超链接，尝试打开 ch01-htmlcomp-base-url-a.html 文件，运行后的效果如图 1.11 所示。在图 1.11 中可以看到页面 ch01-htmlcomp-base-url-a.html 在新的窗口中打开了，并且提示当前页面路径在子目录"url"中。单击页面中的超链接尝试打开 ch01-htmlcomp-base-url-b.html 文件，运行后的效果如图 1.12 所示。

在图 1.12 中，我们看到页面 ch01-htmlcomp-base-url-b.html 在原窗口中打开了，并且提示当前页面路径仍在子目录"url"中。单击页面中的超链接尝试返回 ch01-htmlcomp-base-url.html 文件，运行后的效果如图 1.13 所示。

在图 1.13 中，我们看到主页面 ch01-htmlcomp-base-url.html 在新窗口中重新打开了，不同之处是经过一系列操作之后，浏览器中一共打开了 3 个页面；同时，由于合理地使用了 base 标签进行相对路径解析，使得这 3 个 HTML 页面在根目录和子目录（"url"）之间实现了相互跳转。

图 1.10　HTML 网页之 base 标签基地址操作（1）

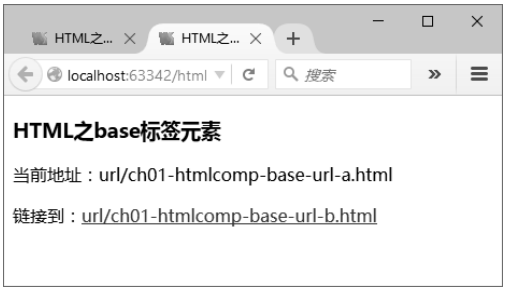

图 1.11　HTML 网页之 base 标签基地址操作（2）

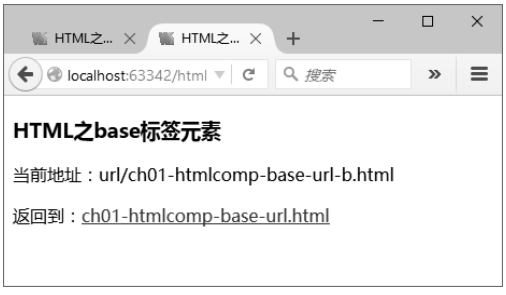

图 1.12　HTML 网页之 base 标签基地址操作（3）

图 1.13　HTML 网页之 base 标签基地址操作（4）

1.3 定义 CSS 样式表为网页排版

一个网页仅有内容是不够的，还需要进行排版，使用户看起来更舒服。CSS 样式表的目的就是为网页排版，美化网页。

1.3.1 CSS 样式表概述

CSS（Cascading Style Sheets，层叠样式表）是一种用来表现 HTML 网页样式的技术。CSS 最初是作为 W3C 的一项标准推出的，从 CSS 1 版本开始，经过 CSS 2 版本的完善，目前的 CSS 3 版本已经被广泛使用，并成为一种事实上的设计标准。

使用 CSS 设计网页的优点是能够真正做到将网页内容与表现形式进行分离，这样设计人员的分工可以更为细化，工作效率也会明显提高。具体来说，CSS 能够支持几乎全部字体风格与字号大小，能够对网页中的对象位置进行像素级别的精确定位，能够对网页对象的样式进行动态编辑，能够进行简单的人机交互设计，是目前基于网页内容展示最优秀的表现类设计语言。

在网页上使用 CSS 基本有 3 种形式，分别为外链式、嵌入式和内联式（可能不同教材上的名称有所区别），具体说明如下。

（1）外链式（Linking）：具体方法是将网页链接到外部样式表。一般页面需要很多样式的时候，外链式 CSS 是最合理的选择，使用外链式 CSS 可以通过修改一个 CSS 文件来改变整个页

面或网站的样式风格。外链式 CSS 的基本使用方法如下：

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

（2）嵌入式（Embedding）：具体方法是在网页上创建嵌入的样式表。一般单个页面需要定制的样式时，嵌入式 CSS 是很好的方法，设计人员可以在 HTML 网页头部通过<style>标签定义嵌入式 CSS。嵌入式 CSS 的基本使用方法如下：

```
<head>
<style type="text/css">
body {background-color: yellow}
p {margin: 32px}
</style>
</head>
```

（3）内联式（Inline）：具体方法是在单个页面元素中加入样式表。只有当页面中的个别元素需要单独的样式时，才推荐使用内联式 CSS。内联式 CSS 的基本使用方法如下：

```
<p style="color:black; margin:16px">
This is a inline-css paragraph.
</p>
```

HTML 网页在解析 CSS 时是有优先级的，其顺序为：内联式 CSS ＞ 嵌入式 CSS ＞ 外联式 CSS，因此设计 CSS 时要考虑优先级顺序，否则可能无法显示出预想的样式效果。

另外，目前应用 CSS 样式表推荐的方式是 DIV+CSS 布局方式，其原因很容易理解，页面结构越简单，通过修改 CSS 改变页面风格就越容易。对于大型站点来说，倘若页面中使用的标签种类繁多、结构复杂，维护 CSS 简直就是一场灾难，可能需要手动修改很多页面；而如果整个站点都使用 DIV+CSS 进行布局，可能仅仅需要修改 CSS 样式表中的一段代码就可以完成对整个站点页面风格的修改。

1.3.2　定义外链式 CSS 样式表

外链式 CSS 是大型站点首选的定义方式。下面我们使用外链式 CSS 方式创建一个页面，体会一下通过修改 CSS 文件中的几行代码就可以改变整个页面样式风格的效果。

下面是一个外链了两个 CSS 样式文件的简单 HTML 网页页面（参见源代码 chapter01/ch01-htmlcomp-css-link.html 文件）。

【示例 1-9】　外链式 CSS 样式表之 HTML 网页

```
01   <!DOCTYPE html>
02   <html lang="zh-cn">
```

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="css/position.css" >
    <link rel="stylesheet" type="text/css" href="css/font.css" >
    <title>HTML 之外链式 CSS</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h1>HTML 之外链式 CSS</h1>
    <div>
        <h3>HTML 之外链式 CSS</h3>
        <p>HTML 之外链式 CSS</p>
    </div>
    <!-- 添加文档主体内容 -->
</body>
</html>
```

在这段 HTML 代码中，第 05 行与第 06 行通过 link 标签引用了两个 CSS 样式文件，第一个样式文件（文件名称为 position.css）用于定义定位、边距等样式，第二个样式文件（文件名称为 font.css）用于定义字体大小、风格等样式。设计时将 CSS 样式按照不同的类别放置于不同的 CSS 样式文件中是比较合理的编程习惯，便于后期的修改与维护操作。

下面是第一个 CSS 样式文件（参见源代码 chapter01/css/position.css 文件），用于定义标签的定位、边距等样式。

【示例 1-10】 外链式 CSS 样式表之边距定义

```
/**
 * CSS - position.css
*/
body {
    margin: 32px;   /** 设置页面边距 */
}
/** h1 */
h1 {
    margin: 16px;   /** 设置外边距 */
    padding: 8px;   /** 设置内边距 */
}
/** div */
div {
    margin: 32px;   /** 设置外边距 */
    padding: 2px;   /** 设置内边距 */
}
/** h3 */
```

```
18  h3 {
19      margin: 8px;   /** 设置外边距 */
20      padding: 4px;   /** 设置内边距 */
21  }
22  /** p */
23  p {
24      margin: 4px;   /** 设置外边距 */
25      padding: 2px;   /** 设置内边距 */
26  }
```

在这段 CSS 代码中，主要使用 CSS 的 margin 属性和 padding 属性定义了标签的外边距与内边距数值。

下面是第二个 CSS 样式文件（参见源代码 chapter01/css/font.css 文件），用于定义文本字体等样式。

【示例 1-11】　外链式 CSS 样式表之字体定义

```
01  /**
02   * CSS - font.css
03  */
04  body {
05      font: normal 12px/1.0em arial,verdana;        /** 设置页面字体 */
06  }
07  /** h1 */
08  h1 {
09      font: bold 24px/2.4em arial,verdana;          /** 设置字体 */
10      letter-spacing: 2px;                        /** 设置字符间距 */
11  }
12  /** h3 */
13  h3 {
14      font: italic 18px/1.8em arial,verdana;        /** 设置字体 */
15      letter-spacing: -0.2em;                       /** 设置字符间距 */
16  }
17  /** p */
18  p {
19      font: bold 12px/1.2em arial,verdana;          /** 设置字体 */
20      letter-spacing: 16px;                       /** 设置字符间距 */
21  }
```

在这段 CSS 代码中，主要使用 CSS 的 font 属性和 letter-spacing 属性定义了字体样式与字符间距数值。

下面我们运行测试这个页面，效果如图 1.14 所示。我们看到了使用 letter-spacing 字符间距属性的效果，在第 15 行中定义了一个负值的字符间距，图 1.14 中的第二行文本被有效压缩了。

最后，我们向读者演示如何通过修改外链式 CSS 文件中的一行或几行达到改变页面整体风格的效果。我们在 chapter01/css/font.css 样式文件中的第 05 行后添加一行样式定义，具体如下：

```
text-decoration: underline;     /** 设置字体下画线 */
```

重新刷新 chapter01/ch01-htmlcomp-css-link.html 页面文件，运行后的效果如图 1.15 所示。我们看到页面中的全部文本被添加了下画线，这是因为上面添加的样式代码作用于 body 标签，所以页面主体中的全部文本均会被添加该样式效果。

图 1.14　外链式 CSS 样式表效果图（1）

图 1.15　外链式 CSS 样式表效果图（2）

1.3.3　定义 CSS 样式 style 标签

外链式 CSS 固然是建设大型站点所推荐的定义样式方法，但如果仅仅需要在某个单独的页面加入一些特别的样式风格，那么使用 style 标签是相对快捷的方法，维护起来也比较方便，毕竟所定义的 CSS 样式代码仅仅对本页面的内容才有效。

下面是一个通过 style 标签定义 CSS 样式风格的 HTML 网页页面（参见源代码 chapter01/ch01-htmlcomp-css-style.html 文件）。

【示例 1-12】　CSS 样式 style 标签

```
01  <!DOCTYPE html>
02  <html lang="zh-cn">
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <style type="text/css">
06          body {
07              margin: 32px;                              /** 设置页面边距 */
08              font: normal 12px/1.0em arial,verdana;     /** 设置页面字体 */
09              text-decoration: underline;                /** 设置字体下画线 */
10          }
11          /** h1 */
12          h1 {
```

```
13              margin: 16px;                                /** 设置外边距 */
14              padding: 8px;                                /** 设置内边距 */
15              font: bold 24px/2.4em arial,verdana;         /** 设置字体 */
16              letter-spacing: 2px;                         /** 设置字符间距 */
17          }
18          /** div */
19          div {
20              margin: 32px;                                /** 设置外边距 */
21              padding: 2px;                                /** 设置内边距 */
22          }
23          /** h3 */
24          h3 {
25              margin: 8px;                                 /** 设置外边距 */
26              padding: 4px;                                /** 设置内边距 */
27              font: italic 18px/1.8em arial,verdana;       /** 设置字体 */
28              letter-spacing: -0.2em;                      /** 设置字符间距 */
29          }
30          /** p */
31          p {
32              margin: 4px;                                 /** 设置外边距 */
33              padding: 2px;                                /** 设置内边距 */
34              font: bold 12px/1.2em arial,verdana;         /** 设置字体 */
35              letter-spacing: 2px;                         /** 设置字符间距 */
36          }
37      </style>
38      <title>HTML 之 CSS 样式 style 标签</title>
39  </head>
40  <body>
41      <!-- 添加文档主体内容 -->
42      <h1>HTML 之 CSS 样式 style 标签</h1>
43      <div>
44          <h3>HTML 之 CSS 样式 style 标签</h3>
45          <p>HTML 之 CSS 样式 style 标签</p>
46      </div>
47      <!-- 添加文档主体内容 -->
48  </body>
49  </html>
```

在这段 HTML 代码中，第 05～37 行通过 style 标签定义了 CSS 样式代码，读者可以将这段样式代码与【示例 1-10】和【示例 1-11】的样式代码进行比较，会发现【示例 1-12】将定位、边距等样式和字体大小、风格等样式合并在一起了，这是通过 style 标签定义样式代码的特点。因此，通过 style 标签定义样式代码只适合单个 HTML 网页页面的场景，这样在后期修改与维护

操作时还是比较方便的，同时可以满足对单个页面定义特殊风格的 CSS 样式代码的需求。

下面我们运行测试这个页面，页面打开后的效果如图 1.16 所示。

图 1.16　定义 CSS 样式 style 标签效果图

1.3.4　定义内联式 CSS 样式表

除了外链式 CSS 样式表和嵌入式 CSS 样式表外，还有一种常用的方式就是内联式 CSS 样式表。该方式非常适合在具体的标签内部加入一段很短的样式代码，是最为快捷方便的定义方法。但对于大型站点，内联式 CSS 维护起来是比较费时费力的，毕竟设计人员需要在 HTML 页面通篇代码中定位到具体的标签后才能修改其 CSS 样式代码。当然，目前流行的集成开发工具的功能都十分强大，可以协助设计人员快速定位，不过还是建议不要在大型站点的开发中大量使用内联式 CSS 样式表。

下面是一个使用内联式 CSS 定义样式代码的 HTML 网页页面（参见源代码 chapter01/ch01-htmlcomp-css-inline.html 文件）。

【示例 1-13】　内联式 CSS 样式表

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之内联式 CSS</title>
</head>
<body style="margin:32px;font:12px/1.0em arial,verdana;text-
decoration:underline;">
    <!-- 添加文档主体内容 -->
    <h1 style="margin:16px;padding:8px;font:bold 24px/2.4em
arial,verdana;">HTML 之  内联式 CSS</h1>
    <div style="margin:32px;padding:2px;">
        <h3 style="margin:8px;padding:4px;font:italic 18px/1.8em
 arial,verdana;">HTML 之内联式 CSS</h3>
        <p style="margin:4px;padding:2px;font:bold 12px/1.2em
```

```
arial,verdana;">HTML16  之内联式 CSS</p>
    13      </div>
    14      <!-- 添加文档主体内容 -->
    15  </body>
    16  </html>
```

在这段 HTML 代码中，CSS 样式代码是嵌入每一个具体的标签内部的。例如，第 07 行对 body 标签使用 style 属性定义了外边距、字体和下画线 3 个样式，第 09、11 和 12 行分别对 h1、h3 和 p 这 3 个标签使用 style 属性定义了外边距、内边距和字体 3 个样式，第 10 行对 div 标签使用 style 属性定义了外边距和内边距两个样式。

下面我们运行测试这个页面，打开后的效果如图 1.17 所示。

图 1.17　内联式 CSS 样式表效果图

在图 1.17 中，我们看到定义的 CSS 样式效果全部显示出来了，再回过头去看【示例 1-13】中的代码，由于在每一个标签中均加入了 CSS 样式代码，整段代码显得臃肿纷乱，远没有使用外链式 CSS 样式和 style 标签的 CSS 样式那样层次分明。可以想象，当 HTML 网页页面中包含成百上千个标签且样式风格极为复杂时，使用内联式 CSS 样式简直是一场灾难，维护的难度可想而知。因此，使用内联式 CSS 样式表时务必要简短精悍，只在最需要的时候使用该方式。

1.4 添加网站 LOGO

网站 LOGO 通常是指一个标志符号，该标志符号通常使用文字、图形、色彩等手法组合而成，并表达出一定的特殊内涵。我们举一些例子，互联网巨头（譬如 Google、Facebook、Microsoft 等）的门户网站一般都使用以自己公司名称为主题设计制作的网站 LOGO；产品设备制造商（譬如汽车、服装、食品等）的公司网站一般都使用其产品注册商标为主题设计制作的网站 LOGO；而诸如提供娱乐、购物、旅游等在线服务类网站的 LOGO，设计理念更为丰富多彩，写意性和抽象性的特点更为突出。可以说，无论公司网站采用什么风格形式的 LOGO，其最突出的特征就是能够表达出公司的核心价值，能够给用户最直接、最深刻、最具想象力的视觉

效果。

网站 LOGO 设计大概可以分为以下几大类：

- 第一大类是文字 LOGO，指使用中文、英文、数字等符号经过艺术处理和美化加工后形成的文字标志，且往往是以公司名称为主体而设计的。
- 第二大类是图形 LOGO，指由点、线、面等不规则图形所组成，并配有鲜明色彩对比所创造出来的新图形，且该图形更具抽象性。
- 第三大类是图像 LOGO，指以现实生活中存在的事物为主体所制作的图像，产品类公司网站一般采用这种 LOGO。

目前，网站 LOGO 基本使用 CSS 样式代码来设计制作，使用 CSS 方式设计网站 LOGO 有多种实现方法，下面我们一一举例。

1.4.1 添加网站图像 LOGO

下面这段代码（参见源代码 chapter01/ch01-htmlcomp-logo-png.html 文件）实现一个网站图像 LOGO。

【示例 1-14】 CSS 之网站图像 LOGO

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <style type="text/css">
      body {
         margin: 32px;                          /** 设置页面边距 */
         font: normal 12px/1.0em arial,verdana;     /** 设置页面字体 */
      }
      /** h1 */
      h1 {
         margin: 16px;                          /** 设置外边距 */
         padding: 8px;                          /** 设置内边距 */
         font: bold 16px/1.6em arial,verdana;       /** 设置字体 */
         letter-spacing: 2px;                   /** 设置字符间距 */
         text-align: center;                        /** 设置字符居中 */
      }
      /** div */
      div {
         margin: 32px;                          /** 设置外边距 */
         padding: 2px;                          /** 设置内边距 */
      }
```

```
23          /** span */
24          span {
25              width: 50%;                                  /** 设置宽度 */
26              height: auto;                                /** 设置高度 */
27          }
28          /** p */
29          p {
30              margin: 4px;                                 /** 设置外边距 */
31              padding: 2px;                                /** 设置内边距 */
32              font: bold 12px/1.2em arial,verdana;         /** 设置字体 */
33              letter-spacing: 2px;                         /** 设置字符间距 */
34          }
35          /** .logo */
36          .logo {
37              width:100px;                                 /** 设置宽度 */
38              height:100px;                                /** 设置高度 */
39              background-image:url(images/logo.png);       /** 引用图像 */
40              background-position:2px 2px;
41              background-repeat:no-repeat;
42              float:left;                                  /** 设置浮动定位 */
43          }
44      </style>
45      <title>HTML 之 CSS 图像 logo</title>
46  </head>
47      <!-- 添加文档主体内容 -->
48      <div>
49          <span class="logo" title="logo"></span>
50          <h1>HTML 之 CSS 图像 logo</h1>
51      </div>
52      <hr>
53      <div>
54          <p>HTML 之 CSS 图像 logo</p>
55      </div>
56      <!-- 添加文档主体内容 -->
57  </body>
58  </html>
```

在这段 HTML 代码中，网站 LOGO 是通过 CSS 样式代码引用本地图像来实现的。下面解释这段代码。

第 49 行通过 span 标签定义了一个放置网站 LOGO 的区域，并使用 class 类属性引用了属性值为"logo"的 CSS 样式类，同时使用 title 属性定义了"logo"属性值。

而名称为"logo"的 CSS 样式类是在第 36～43 行中定义的，第 39 行通过 background-image 属

性引用了本地图像（"images/logo.png"），第 42 行通过 float 属性定位浮动位置为左侧，这样就保证了网站 LOGO 的位置居于页面左上角。

第 52 行使用 hr 标签创建了一条水平分割线，这样使得第 48～51 行定义的 div 层看上去像是页面的标题栏，第 49 行定义的网站 LOGO 和第 50 行定义的标题均位于该标题栏内。

下面我们运行测试这个页面，页面打开后的效果如图 1.18 所示。我们看到使用 CSS 样式定义的网站 LOGO 显示出来了，使用图像方式定义 LOGO 还是很容易的。

图 1.18　CSS 之图像网站 LOGO 效果图

1.4.2　添加网站文字 LOGO

我们再看一个示例，这段代码（参见源代码 chapter01/ch01-htmlcomp-logo-text.html 文件）实现一个动态文字网站 LOGO。

【示例 1-15】　CSS 之网站文字 LOGO

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        body {
            margin: 32px;                          /** 设置页面边距 */
            font: normal 12px/1.0em arial,verdana;     /** 设置页面字体 */
        }
        @import url(http://fonts.googleapis.com/css?family=fantasy);
        #logo {
            width: auto;
            height: 64px;
            font-family: fantasy;
            font-size: 24px;
            color: purple;
            margin: 16px;
            -webkit-transition: all 3s ease-in-out;
```

```
            -moz-transition: all 3s ease-in-out;
            -o-transition: all 3s ease-in-out;
            -ms-transition: all 3s ease-in-out;
            transition: all 3s ease-in-out;
            text-shadow:
            1px 1px 0 rgba(36,117,203,1),
            13px 3px 0 rgba(36,117,203,0.9),
            5px 5px 0 rgba(36,117,203,0.8),
            17px 7px 0 rgba(36,117,203,0.7),
            9px -15px 0 rgba(36,117,203,0.6),
            -11px 11px 0 rgba(36,117,203,0.5),
            7px -13px 0 rgba(36,117,203,0.4),
            15px 30px 0 rgba(36,117,203,0.3),
            -20px 17px 0 rgba(36,117,203,0.2),
            -19px 19px 0 rgba(36,117,203,0.1),
            30px -21px 0 rgba(36,117,203,0.08),
            23px 23px 0 rgba(36,117,203,0.06),
            -25px 40px 0 rgba(36,117,203,0.04),
            27px 27px 0 rgba(36,117,203,0.02),
            -29px 29px 0 rgba(36,117,203,0.0);
        }
        #logo:hover {
            -webkit-transform: rotate(3deg) scale(1.05);
            -moz-transform: rotate(3deg) scale(1.05);
            -o-transform: rotate(3deg) scale(1.05);
            -ms-transform: rotate(3deg) scale(1.05);
            transform: rotate(3deg) scale(1.05);
            text-shadow:
            10px 1px 0 rgba(36,117,203,0.1),
            13px 23px 0 rgba(36,117,203,0.2),
            15px 5px 0 rgba(36,117,203,0.03),
            17px 7px 0 rgba(36,117,203,0.04),
            9px 15px 0 rgba(36,117,203,0.2),
            -11px -11px 0 rgba(36,117,203,0.06),
            17px -13px 0 rgba(36,117,203,0.07),
            15px 30px 0 rgba(36,117,203,0.1),
            -20px -17px 0 rgba(36,117,203,0.06),
            -19px 19px 0 rgba(36,117,203,0.08),
            5px 21px 0 rgba(36,117,203,0.02),
            23px -23px 0 rgba(36,117,203,0.1),
            25px 40px 0 rgba(36,117,203,0.2),
            27px 17px 0 rgba(36,117,203,0.1),
```

```
            -29px -29px 0 rgba(36,117,203,0.1);
        }
        /** div */
        div {
            margin: 32px;   /** 设置外边距 */
            padding: 2px;   /** 设置内边距 */
        }
        /** span */
        span {
            width: 33%;                          /** 设置宽度 */
            height: auto;                        /** 设置高度 */
        }
        /** h1 */
        h1 {
            margin: 2px;                         /** 设置外边距 */
            padding: 2px;                        /** 设置内边距 */
            font: bold 16px/1.6em arial,verdana;   /** 设置字体 */
            letter-spacing: 2px;                 /** 设置字符间距 */
            text-align: right;                   /** 设置字符居中 */
        }
        /** p */
        p {
            margin: 4px;                         /** 设置外边距 */
            padding: 2px;                        /** 设置内边距 */
            font: bold 12px/1.0em arial,verdana;   /** 设置字体 */
            letter-spacing: 2px;                 /** 设置字符间距 */
        }
    </style>
    <title>HTML 之 CSS 文本 logo</title>
</head>
    <!-- 添加文档主体内容 -->
    <div>
        <span id="logo"><a href="#">CSS 文本 logo</a></span>
        <span><h1>HTML 之 CSS 文本 logo</h1></span>
    </div>
    <hr>
    <div>
        <p>HTML 之 CSS 文本 logo</p>
    </div>
    <!-- 添加文档主体内容 -->
</body>
</html>
```

在这段 HTML 代码中，网站 LOGO 是通过文字 CSS 样式代码来实现的。下面解释这段代码。

先来看第 10 行，这里使用一个不太常用的语法@import 引入了一个名称为"fantasy"的文本字体，而引用该字体时通过 url 关键字定义了一个链接地址（http://fonts.googleapis.com/css?family=fantasy），这表明其取自于 Google 公司提供的字体库。之所以要引入这样一个字体，是因为这段代码要实现一个 CSS 样式的文本 LOGO，使用一种特殊的风格字体显示出来会更吸引眼球。另外，关于@import 的语法，读者可以参阅相关文档，其使用方法类似于 link 标签，但又不尽相同。

再来看第 92～99 行，这是页面的主体部分。第 93 行定义了一个 id 值为"logo"的 span 标签，这里就是我们要实现的文字 LOGO；第 94 行定义了一段文字，相当于页面标题；第 96 行使用 hr 标签创建了一条水平分割线，这样使得第 92～95 行定义的 div 层与第 97～99 行定义的 div 层隔离开来，形成了页面标题栏和页面正文部分的效果。

而第 93 行定义 id 值"logo"，是在 CSS 样式代码部分的第 11～39 行与第 40～62 行中实现的。这里主要使用了文字渐变（"transition"）和文字阴影（"text-shadow"）两个 CSS 3 属性（注意，这两个是 CSS 3 版本才支持的属性）。使用渐变"transition"属性后，当鼠标放置于某区域中时会产生宽度上的变化效果，且针对不同的浏览器有不同的定义方法，在第 18～22 行和第 41～45 行中进行了定义；使用阴影"text-shadow"属性后会为文字增加阴影效果，在第 23～38 行和第 46～61 行中进行了定义。

下面我们运行测试这个页面，页面打开后的效果如图 1.19 所示。我们看到使用 CSS 样式定义的网站文字 LOGO 显示出来了，该 LOGO 还具有动态效果，当用户将鼠标放置于该 LOGO 区域中时，文字会产生动态旋转效果，如图 1.20 所示。

图 1.19　CSS 之网站文字 LOGO 效果图（1）

图 1.20　CSS 之网站文字 LOGO 效果图（2）

1.4.3　添加网站图形 LOGO

我们再看一个示例，这段代码（参见源代码 chapter01/ch01-htmlcomp-logo-drawing.html 文件）通过 CSS 绘制图形来实现网站 LOGO。

【示例 1-16】　CSS 之网站图形 LOGO

```
01   <!DOCTYPE html>
```

```
<html lang="zh-cn">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <style type="text/css">
      body {
         margin: 16px auto;                    /** 设置外边距 */
         font: bold 16px/1.6em arial,verdana;      /** 设置字体 */
         text-align: center;
      }
      /** drawing */
      .drawing-holder {
         float: left;                          /** 定义浮动 */
         margin: 32px;                         /** 设置外边距 */
         text-align: center;
      }
      .drawing {
         position: relative;                       /** 定义位置 */
         background: white;                    /** 定义背景色 */
         width: 100px;                      /** 定义宽度 */
         height: 100px;                     /** 定义高度 */
         border-radius: 48px;                  /** 定义圆角 */
         box-sizing: border-box;               /** 定义边框 */
         box-shadow: 0 2px 4px rgba(0, 0, 0, 0.3);   /** 定义边框阴影 */
      }
      .drawing:after {                         /** 定义伪类 after */
         content: '';
         display: block;
         clear: both;
      }
      .drawing-name {
         line-height: 36px;                    /** 定义行高度 */
         font-family: 'Oswald', sans-serif;
      }
      /* ----- logo ----- */
      .drawing.logo {                          /** 定义 logo 类 */
         padding: 24px;
      }
      .drawing.logo .wrap {
         position: relative;
         width: 100%;
         height: 100%;
      }
```

```
        .drawing.logo .logoshape {            /** 定义 logoshape 类 */
            position: absolute;                   /** 定义位置 */
            left: 18px;                    /** 定义位置"左" */
            width: 16px;                   /** 定义宽度 */
            height: 16px;                  /** 定义高度 */
            background: black;             /** 定义背景色 */
            -webkit-transform: rotate(-8deg) skew(16deg, 16deg);/** 定义旋转 */
            -moz-transform: rotate(-8deg) skew(16deg, 16deg); /** 定义旋转 */
            -ms-transform: rotate(-8deg) skew(16deg, 16deg);  /** 定义旋转 */
            transform: rotate(-8deg) skew(16deg, 16deg);      /** 定义旋转 */
        }
        .drawing.logo .logoshape:nth-child(2) {        /** 定义第2个子元素 */
            margin-top: 24px;
        }
        .drawing.logo .logoshape:nth-child(3) {        /** 定义第3个子元素 */
            margin-left: -24px;
        }
        .drawing.logo .logoshape:nth-child(4) {        /** 定义第4个子元素 */
            margin-left: 24px;
        }
        .drawing.logo .logoshape:nth-child(5) {        /** 定义第5个子元素 */
            margin-top: 48px;
        }
        .drawing.logo .logoshape:nth-child(3),
        .drawing.logo .logoshape:nth-child(4) {      /** 定义第3、4个子元素 */
            margin-top: 24px;
        }
        /** title */
        #title {
            float: right;
            margin: 32px;
            height: 100px;
            text-align: center;
        }
        /** h1 */
        h1 {
            margin: 2px;                           /** 设置外边距 */
            padding: 2px;                          /** 设置内边距 */
            font: bold 24px/2.0em arial,verdana;   /** 设置字体 */
            letter-spacing: 2px;                   /** 设置字符间距 */
            text-align: right;             /** 设置字符居中 */
        }
```

```
86      </style>
87      <title>HTML 之 CSS 图形 logo</title>
88  </head>
89      <!-- 添加文档主体内容 -->
90      <div class="drawing-holder">
91          <div class="drawing logo">
92              <div class="wrap">
93                  <div class="logoshape"></div>
94                  <div class="logoshape"></div>
95                  <div class="logoshape"></div>
96                  <div class="logoshape"></div>
97                  <div class="logoshape"></div>
98              </div>
99          </div>
100         <div class="drawing-name">LOGO</div>
101     </div>
102     <div id="title">
103         <h1>HTML 之 CSS 图形 logo</h1>
104     </div>
105     <!-- 添加文档主体内容 -->
106 </body>
107 </html>
```

在这段 HTML 代码中，网站 LOGO 是通过图形 CSS 样式代码来实现的。下面解释这段代码。

先来看第 90～101 行，这里使用嵌套的 div 层标签定义了一组图形。

第 90 行为最外面的 div 层，引用了一个 CSS 类（名称为"drawing-holder"），该 CSS 类的定义在第 12～16 行，主要用于实现一个图形 LOGO 的层（div）容器。

第 91 行为第二个 div 层，引用了一个 CSS 类（名称为"drawing logo"），注意"drawing"与"logo"之间的空格，表明"logo"是"drawing"的子类。"drawing"类的定义在第 17～25 行，定义了位置、长宽、背景、圆角和边框及其阴影等属性；"logo"类的定义在第 36～38 行，定义了内边距属性。

第 92 行为第三个 div 层，引用了一个 CSS 类（名称为"wrap"），可以将其理解为一组 LOGO 图形的小容器，"wrap"类的定义在第 39～43 行，定义了位置和长宽属性。

生成图形 LOGO 的主要代码在第 93～97 行，这里定义了一组 div 层，引用了相同的 CSS 类（名称为"logoshape"）。"logoshape"类的定义在第 44～70 行，其中第 44～54 行是对图形 LOGO 主体的定义，不但对位置、尺寸、背景色等属性进行了定义，还使用旋转变换属性美化了 LOGO 图形，其中 rotate 方法表示以图形中心进行旋转，skew 方法表示沿着 *x* 轴进行 2D 旋转；而第 55～70 行代码使用伪类选择器 nth-child 对第 2~5 个图形的位置进行了定义，这样就可以在页面上展示一组完整的图形 LOGO。

第 100 行使用 div 层标签定义了图形 LOGO 的文字描述部分，其引用了一个 CSS 类（名称为"drawing-name"），"drawing-name"类的定义在第 31～34 行，主要定义了字体属性。

下面我们运行测试这个页面，页面打开后的效果如图 1.21 所示。我们看到使用纯 CSS 样式代码定义的网站 LOGO 是一个包含 5 个小菱形及一个外部阴影的图形，5 个小菱形呈梅花形状布置，图形 LOGO 整体十分简洁美观。

图 1.21　CSS 之网站图形 LOGO 效果图

1.5 使用脚本元素 script 标签

脚本存在的意义是可以在网页中放入一段动态代码，让网页能够动起来，这就是我们常说的动态网页，这些动态代码一般称为脚本语言代码。动态网页可以实现网页与用户的交互。

1.5.1 HTML 网页内嵌脚本让网页动起来

HTML 网页中的 script 标签用于定义脚本语言，主要包括 JavaScript、VBScript、ECMAScript 等脚本语言，目前最流行的当然是 JavaScript 脚本语言了，所以我们提到的嵌入脚本指的就是嵌入 JavaScript 脚本。

在前文 1.1 节中，我们向读者介绍了 script 标签的基本使用方法，这里先明确使用 script 标签嵌入脚本的语法，看下面几行：

```
<script type="text/javascript">
……
</script>
```

如果读者想在 HTML 网页中嵌入脚本，那么一般需要遵循上面的写法，并将脚本语言写在两个 script 标签之间。

下面我们看一段代码，让读者体会使用 script 标签嵌入脚本的方法。这段代码（参见源代码 chapter01/ch01-htmlcomp-script-embed.html 文件）是一个在 HTML 网页内不同位置嵌入脚本的应用，这个应用稍微有点复杂，后面我们会详细讲解。

【示例 1-17】 HTML 网页内嵌脚本

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        div {
            margin: 2px;
            padding: 2px;
            width: auto;
            height: auto;
            border: 1px solid #000;
        }
    </style>
    <script type="text/javascript">
        document.write("<h5>log: header - script</h5>");
        document.close();
 document.getElementById("id-div-header").innerHTML+="<br><h3>header-
script</h3>";
    </script>
    <title>HTML 之嵌入式 script</title>
 </head>
 <body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之嵌入式 script</h3>
    <hr>
    <!-- 添加文档主体内容 -->
    <div id="id-div-header">header script log:</div>
    <!-- 添加脚本 -->
    <script type="text/javascript">
        document.write("<h5>log: dynamic - script</h5>");
        document.write("<div id='id-div-dynamic'>dynamic script log:</div>");
        document.close();
 document.getElementById("id-div-dynamic").innerHTML+="<h3>dynamic-script</h3>";
    </script>
    <!-- 添加文档主体内容 -->
    <div id="id-div-body">body script log:</div>
    <!-- 添加脚本 -->
    <script type="text/javascript">
        document.write("<h5>log: body - script</h5>");
        document.close();
 document.getElementById("id-div-body").innerHTML += "<br><h3>body -
script</h3>";
    </script>
 </body>
 </html>
```

在这段 HTML 代码中使用的全部是内嵌脚本。下面解释这段代码。

先来看第 14～18 行，这里是一段内嵌在 head 标签中的脚本代码。其中，第 15 行使用 document.write()方法尝试向 HTML 网页中写入一行日志，因为调用 document.write()方法将会打开一个输出流，所以在第 16 行使用 document.close()方法对其进行了关闭（建议读者予以重视）；第 17 行使用 document.getElementById()方法尝试在一个 div 标签（id 值为"id-div-header"，在第 26 行中定义）内插入一段文本。

然后来看第 28～33 行，这里是一段内嵌在 body 标签中的脚本代码。其中，第 29～30 行使用 document.write()方法打开一个输出流，尝试向 HTML 网页中写入一行日志并动态创建了一个 div 标签（id 值为"id-div-dynamic"），并在第 31 行使用 document.close()方法关闭了该输出流；第 32 行使用 document.getElementById()方法尝试在这个动态创建的 div 标签内插入一段文本。

最后来看第 37～41 行，这里同样是一段内嵌在 body 标签中的脚本代码。其中，第 38 行使用 document.write()方法打开一个输出流，尝试向 HTML 网页中写入一行日志，并在第 39 行使用 document.close()方法关闭了该输出流；第 40 行使用 document.getElementById()方法尝试在一个 div 标签（id 值为"id-div-body"，在第 35 行中定义）内插入一段文本。

下面我们运行测试这个页面，效果如图 1.22 所示。

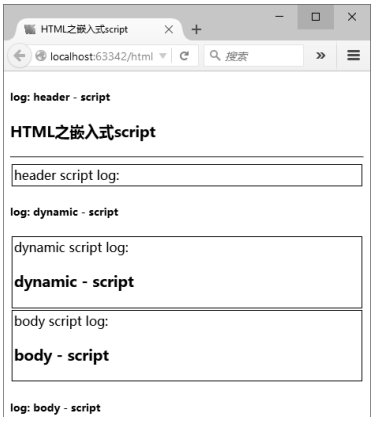

图 1.22　HTML 网页内嵌脚本效果图

从图 1.22 中看到页面中输出的信息比较多，下面我们对图中输出的结果进行详细的解释。

最先输出的是一行日志信息，通过查看【示例 1-17】的代码知道是第 15 行中输出的。而第 2 行输出的信息是第 23 行定义的，可见在 head 标签之间定义的第 14～18 行脚本代码最先被编译执行了，在 HTML 网页 DOM 树被解析之前就执行了，这完全是基于在 HTML 网页定义的 JavaScript 脚本语言编译执行一项基本原则，即“按顺序载入，载入即执行”。而第 17 行脚本代码尝试在 div 标签（id 值为"id-div-header"）中插入文本信息的操作却没有显示。第 3 行输出的 div 标签（带细线边框风格）没有发生预期的改变，这是因为该标签是在第 26 行定义的，是在脚本操作之后，相当于操作时还没有被定义。

然后，页面第 4 行输出的也是一行日志信息，通过查看【示例 1-17】的代码知道是第 29 行输出的。页面第 5 行显示的是第 30 行动态创建的 div 标签（id 值为"id-div-dynamic"，带细线边

框风格），而第 32 行尝试在动态创建的 div 标签中插入文本的操作也成功显示了，见页面第 6 行的文本输出。

最后，页面中第 7 行显示的是第 35 行定义的 div 标签（id 值为"id-div-body"，带细线边框风格），而第 40 行尝试在该 div 标签中插入文本的操作也成功显示了，见页面中第 8 行的文本输出。页面第 9 行输出的也是一行日志信息，通过查看【示例 1-17】的代码知道是第 38 行中输出的，虽然第 38 行定义在第 40 行之前，会先于第 40 行执行，但由于 id 值为"id-div-body"的 div 标签是在第 35 行中定义的，所以第 38 行定义的文本信息是最后被显示输出的。

通过上面的分析，读者会对 HTML 网页中定义的 JavaScript 脚本语言的编译执行顺序有一个初步了解。JavaScript 脚本执行顺序需要遵循以下几个大的原则：

- 按顺序载入。
- 载入即执行。
- 执行时会阻塞后续内容。

所以，为了避免第 17 行没有被有效执行的情况出现，一般建议将自定义脚本放在 body 标签的最后（或 HTML 网页最后），这样可以保证全部 HTML 网页 DOM 树载入后再执行脚本。当然，JavaScript 语言实际运用时非常复杂，上面的几个原则是最基本的，读者可以自行深入研究。

1.5.2 载入外部脚本库

这一小节我们介绍载入外部脚本库的方法，这里先明确使用 script 标签载入外部脚本库的语法：

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

如果读者想在 HTML 网页中载入外部脚本库，一般需要遵循上面的写法，并建议放在 head 标签内。但是根据 1.5.1 小节中所介绍的 JavaScript 脚本执行的原则，有时脚本引入需放在标签声明之后，才能被正常执行。

下面我们看一段载入外部脚本库的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-script-src.html 文件）是一个在 HTML 网页内载入外部脚本库的应用。

【示例 1-18】 HTML 网页载入外部脚本

```
01  <!DOCTYPE html>
02  <html lang="zh">
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <style type="text/css">
06          div {
07              margin: 2px;
08              padding: 2px;
```

```
09              width: auto;
10              height: auto;
11              border: 1px solid #000;
12          }
13      </style>
14
15      <title>HTML 之载入外部脚本</title>
16  </head>
17  <body>
18      <!-- 添加文档主体内容 -->
19      <h3>HTML 之载入外部脚本</h3>
20      <hr>
21      <!-- 添加文档主体内容 -->
22      <div id="id-div-body">body script log:</div>
23      <script type="text/javascript" src="js/src.js"></script>
24  </body>
25  </html>
```

在这段 HTML 代码中使用的是载入外部脚本的方法。下面解释这段代码。

第 23 行使用 script 标签载入了一个外部脚本，其中通过 src 属性定义了该脚本的路径为"js/src.js"，该路径是一个基于本 HTML 网页的相对路径，将其转换成绝对路径为"chapter01/js/src.js"。

下面这段脚本代码（参见源代码 chapter01/js/src.js 文件）是在【示例 1-18】的 HTML 代码中载入外部脚本库。

【示例 1-19】　HTML 网页载入外部脚本库

```
01  window.onload = function() {
02  document.getElementById("id-div-body").innerHTML += "<br><h3>body -
script</h3>";
03  }
```

在这段 HTML 代码中使用的全部是内嵌脚本。下面解释这段代码。

第 02 行使用 document.getElementById()方法尝试在一个 div 标签（id 值为"id-div-body"，在【示例 1-18】第 22 行中定义）内插入一段文本。

下面我们运行测试这个页面，页面打开后的效果如图 1.23 所示。第 02 行脚本代码定义的操作成功显示出来了，可见外部脚本与嵌入脚本的执行效果是完全一样的。

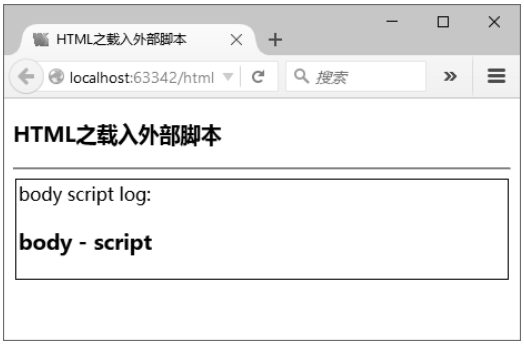

图 1.23　HTML 网页载入外部脚本效果图

1.5.3　推迟脚本执行

在前面的 1.5.1 小节中介绍过脚本语言定义在标签前面而导致操作失效的代码，这一小节我们给出推迟脚本执行的方法。通过在 script 标签中增加一个 defer 属性就可以实现推迟脚本执行的功能。使用 defer 属性的语法如下：

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

如果读者想在 HTML 网页中推迟脚本执行，一般需要遵循上面的写法，并与 src 属性结合使用。

下面我们来看一段推迟脚本执行的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-script-defer.html 文件）是一个在 HTML 网页内测试推迟脚本执行的应用。

【示例 1-20】　HTML 网页推迟脚本执行

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        div {
            margin: 2px;
            padding: 2px;
            width: auto;
            height: auto;
            border: 1px solid #000;
        }
    </style>
    <title>HTML 之载入外部脚本</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
```

```
   18      <h3>HTML 之载入外部脚本</h3>
   19      <hr>
   20      <!-- 添加脚本 -->
   21      <script id="id-script-nodefer" type="text/javascript"
src="js/nodefer.js">
   22      </script>
   23      <!-- 添加文档主体内容 -->
   24      <div id="id-div-nodefer">no defer script log:</div>
   25      <!-- 添加脚本 -->
   26      <script id="id-script-defer" type="text/javascript" src="js/defer.js"
defer>
   27      </script>
   28      <!-- 添加文档主体内容 -->
   29      <div id="id-div-defer">defer script log:</div>
   30      <!-- 添加脚本 -->
   31      <script type="text/javascript">
   32         document.getElementById("id-div-nodefer").innerHTML +=
   33               "<h3>defer value is " +
   34               document.getElementById("id-script-nodefer").defer +
   35               "</h3>";
   36         document.getElementById("id-div-defer").innerHTML +=
   37               "<h3>defer value is " +
   38               document.getElementById("id-script-defer").defer +
   39               "</h3>";
   40      </script>
   41   </body>
   42   </html>
```

在这段 HTML 代码中主要载入了两个外部脚本，其中一个是使用 defer 属性定义的，另一个则没有使用 defer 属性。下面我们分析这段代码。

第 21 行使用 script 标签载入了一个外部脚本，其中通过 src 属性定义了该脚本的路径为"js/nodefer.js"，注意并没有使用 defer 属性。

第 26 行同样使用 script 标签载入了一个外部脚本，其中通过 src 属性定义了该脚本的路径为"js/defer.js"，注意其添加了 defer 属性。

第 31～40 行主要获取了两个 script 标签的 defer 属性值，测试定义与不定义 defer 属性的 script 标签的取值。

下面这段脚本代码（参见源代码 chapter01/js/nodefer.js 文件）实现在【示例 1-20】第 21 行代码中载入外部脚本库。

【示例 1-21】　HTML 网页推迟脚本执行脚本代码（1）

```
document.getElementById("id-div-nodefer").innerHTML +=
```

```
    "<h3>no defer added.</h3>";
```

这段脚本代码尝试在 div 标签（id 值为"id-div-nodefer"，在【示例 1-20】中第 24 行定义）中插入文本信息。

下面这段脚本代码（参见源代码 chapter01/js/defer.js 文件）实现在【示例 1-20】第 26 行代码中载入外部脚本库，其主要代码如下：

【示例 1-22】 HTML 网页推迟脚本执行脚本代码（2）

```
document.getElementById("id-div-defer").innerHTML +=
    "<h3>defer added.</h3>";
```

这段脚本代码尝试在 div 标签（id 值为"id-div-defer"，在【示例 1-20】中第 29 行定义）中插入文本信息。

下面我们运行测试这个页面，效果如图 1.24 所示。【示例 1-22】中脚本代码的操作成功显示出来了，而【示例 1-21】中脚本代码的操作却没有显示出来，可见定义了 defer 属性的外部脚本是可以推迟执行的。同时，定义了 defer 属性的 script 标签取值为 true，而没有定义 defer 属性的 script 标签取值为 false，通过判断该值可以判定脚本在何时被执行。

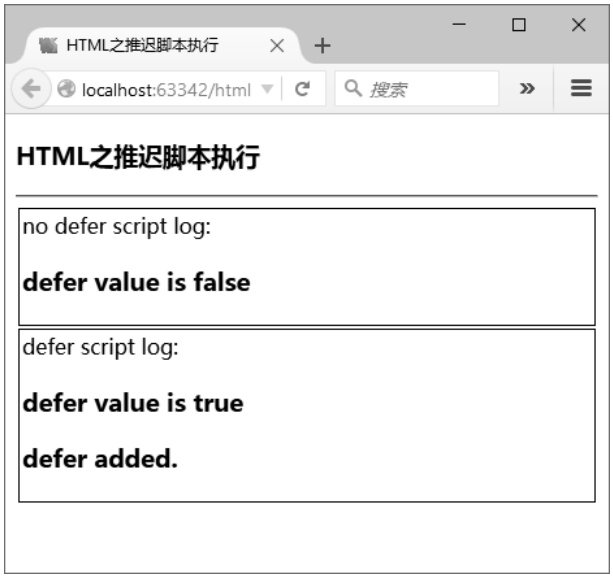

图 1.24 HTML 网页推迟脚本执行效果图

1.5.4 异步执行脚本

在前面的 1.5.1 小节中介绍过脚本会以阻塞方式运行，有些时候这并不是我们想要的效果。这一小节我们给出异步执行脚本的方法，通过在 script 标签中增加一个 async 属性就可以实现异步执行脚本的功能。使用 async 属性的语法如下：

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

如果读者想在 HTML 网页中异步执行脚本，一般需要遵循上面的写法，并与 src 属性结合使用。

下面我们看一段异步执行脚本的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-script-async.html 文件）是一个在 HTML 网页内异步执行脚本的应用。

【示例 1-23】　HTML 网页异步执行脚本

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        /** h3 */
        h3 {
            margin: 8px;                        /** 设置外边距 */
            padding: 4px;                       /** 设置内边距 */
            font: italic 18px/1.8em arial,verdana;     /** 设置字体 */
            letter-spacing: 1.0em;              /** 设置字符间距 */
        }
        /** p */
        p {
            margin: 4px;                        /** 设置外边距 */
            padding: 2px;                       /** 设置内边距 */
            font: bold 12px/1.2em arial,verdana;       /** 设置字体 */
            letter-spacing: 2px;                /** 设置字符间距 */
        }
    </style>
    <script type="text/javascript" src="js/noasync.js"></script>
    <script type="text/javascript" src="js/async.js" async></script>
    <title>HTML 之异步执行脚本</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之异步执行脚本</h3>
    <hr>
    <!-- 添加文档主体内容 -->
    <p>说明: async 属性规定一旦脚本可用,则会异步执行.</p><br>
    <p>备注: async 属性仅适用于外部脚本（只有在使用 src 属性时）.</p><br>
</body>
</html>
```

在这段 HTML 代码中主要载入了两个外部脚本，其中一个是使用 async 属性定义的，另一个则没有使用 async 属性。下面我们向读者分析这段代码。

第 21 行使用 script 标签载入了一个外部脚本，其中通过 src 属性定义了该脚本的路径为"js/noasync.js"，注意并没有使用 async 属性。

第 22 行同样使用 script 标签载入了一个外部脚本，其中通过 src 属性定义了该脚本的路径为"js/async.js"，注意其添加了 async 属性。

下面这段脚本代码（参见源代码 chapter01/js/noasync.js 文件）实现在【示例 1-23】第 21 行代码中载入外部脚本库。

【示例 1-24】 HTML 网页异步执行脚本代码（1）

```
alert("no async js");
```

下面这段脚本代码（参见源代码 chapter01/js/async.js 文件）实现在【示例 1-23】中第 22 行代码中载入外部脚本库。

【示例 1-25】 HTML 网页异步执行脚本代码（2）

```
alert("async js");
```

下面我们运行测试这个页面，页面打开后的效果如图 1.25 所示。

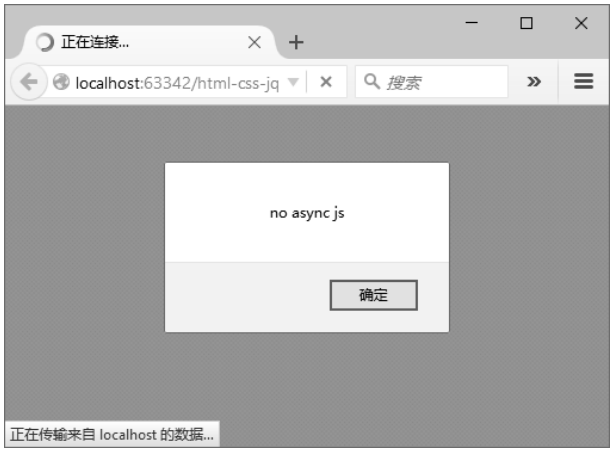

图 1.25 HTML 网页异步执行脚本效果图（1）

从图 1.25 中看到，【示例 1-24】中脚本代码的警告消息框先被显示出来，而【示例 1-23】中第 25～32 行定义的页面内容却没有显示出来，可见【示例 1-23】中第 21 行定义的外部脚本是以阻塞方式执行的。

下面单击消息框中的“确定”按钮，接着运行这个页面，之后的效果如图 1.26 所示。

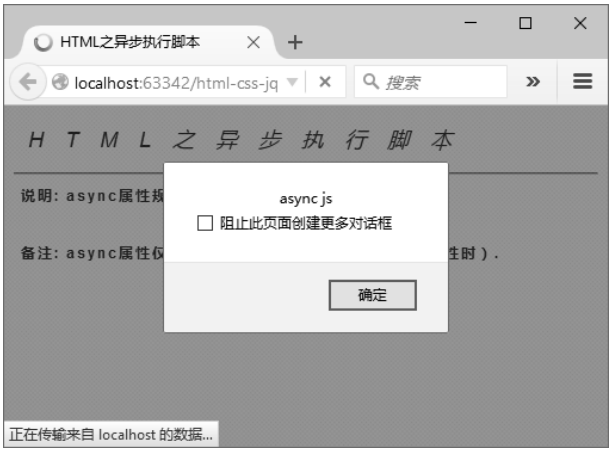

图 1.26　HTML 网页异步执行脚本效果图（2）

从图 1.26 中看到，【示例 1-25】中脚本代码的警告消息框先被显示出来，同时【示例 1-23】中第 25～32 行定义的页面内容也被显示出来，可见【示例 1-23】中第 22 行定义的外部脚本是以异步方式执行的。

下面继续单击消息框中的“确定”按钮，接着运行这个页面，之后的效果如图 1.27 所示。

图 1.27　HTML 网页异步执行脚本效果图（3）

对比图 1.26 与图 1.27 可以看到，通过 async 属性定义脚本是以异步方式加载执行的，使用 async 属性可以大大丰富网页脚本的设计手段。

另外，async 属性与 defer 属性具有一定的关联性，根据 HTML 4 官方文档的解释说明，二者存在以下关系：

- 如果定义 async，那么脚本相对于页面的其余部分异步执行（当页面继续进行解析时，脚本将被执行）。
- 如果不定义 async，但是定义 defer，那么脚本将在页面完成解析时执行。
- 如果既不定义 async 又不定义 defer，那么在浏览器继续解析页面之前立即以阻塞方式读取并执行脚本。

读者可以参考相关文档进行深入的学习。

1.6 使用 noscript 标签判断浏览器是否支持脚本

本节我们介绍 noscript 标签，它用于判断浏览器是否支持脚本语言功能，如果浏览器不支持脚本语言，浏览器就会将 noscript 标签中的内容显示出来。

我们先来看一下使用 noscript 标签的语法：

```
<noscript>…</noscript>
```

如果读者想在 HTML 网页中使用 noscript 标签判断浏览器是否支持脚本语言，一般需要遵循上面的写法。

下面我们看一段使用 noscript 标签的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-noscript.html 文件）给出了在 HTML 网页内使用 noscript 标签的方法。

【示例 1-26】在 HTML 网页内使用 noscript 标签

```
01  <!doctype html>
02  <html>
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <title>HTML 之 noscript 标签</title>
06  </head>
07  <body>
08      <!-- 添加文档内容 -->
09      <h1>HTML 之 noscript 标签</h1>
10      <hr>
11      <!-- 添加文档内容 -->
12      <noscript>Sorry, your browser does not support JavaScript!</noscript>
13      <p>注意：如果您的浏览器不支持 JavaScript 脚本语言，将显示 noscript 标签中的文本。
</p>
14  </body>
15  </html>
```

在这段 HTML 代码中，第 12 行使用 noscript 标签定义了一行文本，如果浏览器不支持脚本语言，该行文本就会被显示出来。

下面我们运行测试这个页面，效果如图 1.28 所示。

从图 1.28 中可以看到，noscript 标签并没有生效，看起来目前默认不支持脚本语言的浏览器几乎绝迹了。不过没关系，我们可以通过设置浏览器的配置或者添加禁用脚本语言的插件来实现测试效果。譬如为 FireFox 浏览器添加一款名为 NoScript 的插件就可以全面禁止脚本语言功能，

当然读者也可以使用别的方法。

下面我们再次运行测试这个页面，效果如图 1.29 所示。浏览器在禁用脚本语言功能后，noscript 标签立即生效，提示用户浏览器不支持 JavaScript 脚本语言。

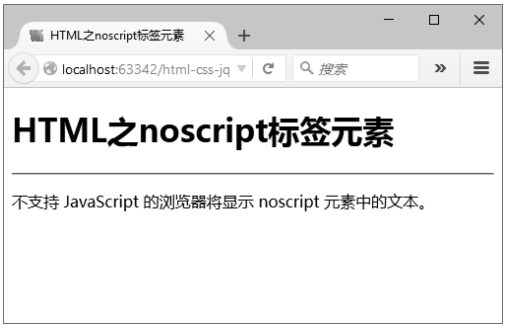

图 1.28　使用 noscript 标签效果图（1）

图 1.29　使用 noscript 标签效果图（2）

1.7 为标签添加 id、name 或 class 属性

当我们在设计一些大型网页时，网页中会有多个相同的标签，如有很多行文字时，可能会有很多个 DIV。为了识别每个 DIV，我们需要为这个 DIV 添加几个属性：id、name，如果要让这几个 DIV 具备相同的样式，就要添加 class 属性来引入 CSS 样式。

1.7.1　为标签添加 id 属性

在 HTML 规范中，标签的 id 属性必须是唯一的，id 是英文单词 identity（一般翻译为身份标识）的缩写，读者可以这样理解：既然是身份标识，就必须是唯一的，这样才可以与其他对象区分开来。

在 HTML 网页中，id 属性是非常有用的，通过 id 属性可以定义锚链接（Link Anchor），可以通过 JavaScript 脚本操作对象，还可以通过 CSS 样式表为带有指定 id 的对象添加或修改样式，等等。

下面我们来看一段操作 id 属性的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-ele-id.html 文件）给出通过标签 id 属性添加内容的操作方法，是比较常见的一种操作。

【示例 1-27】　为标签添加 id 属性

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- 添加文档头部内容 -->
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
```

```
07          div {
08              margin: 2px;
09              padding: 2px;
10              width: auto;
11              height: auto;
12              border: 1px solid #000;
13          }
14      </style>
15      <title>HTML 之标签 id</title>
16  </head>
17  <body>
18      <!-- 添加文档主体内容 -->
19      <h3>HTML 之标签 id</h3>
20      <hr>
21      <!-- 添加文档主体内容 -->
22      <div>div</div>
23      <div id="id-div">div</div>
24      <script type="text/javascript">
25          document.getElementById("id-div").innerHTML += "操作 id";
26      </script>
27  </body>
28  </html>
```

在这段 HTML 代码中，第 22 行与第 23 行定义了两个同样的 div 标签，不同的地方是第 23 行定义的 div 标签添加了 id 属性（属性值为"id-div"）作为其唯一标识，第 25 行脚本代码使用 document.getElementById()方法为第 23 行定义的 div 标签动态追加了文本“操作 id”。

下面我们运行测试这个页面，效果如图 1.30 所示。从中可以看到，如果标签定义了 id 属性，就可以对其进行操作来完成一些动态效果。

图 1.30　为标签添加 id 属性

1.7.2　为标签添加 name 属性

在 HTML 规范中，name 属性主要用于定义表单域内标签，譬如 input、select、textarea、iframe、frame、window、button 等，这些标签都与表单提交相关，一般服务器端只接收表单内具有 name 属性的元素，所以只具有 id 属性的标签是无法通过表单服务器端接收的。

另外，我们知道 radio 类标签必须在同一个分组中，且 check 操作是互斥的，即同一时间只能选中一个 radio 标签，所以这个分组就是根据相同的 name 属性来实现的。

最后，name 属性与 id 属性类似，同样可以定义锚链接，可以通过 JavaScript 脚本操作对象，通过 CSS 样式表为带有指定 name 的对象添加或修改样式，等等。

我们先来看一段操作 name 属性的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-ele-name.html 文件）给出通过标签 name 属性获取数组长度的操作方法。

【示例 1-28】为标签添加 name 属性

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- 添加文档头部内容 -->
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        input {
            margin: 2px;
            padding: 2px;
            width: 64px;
            border: 1px solid #000;
        }
    </style>
    <title>HTML 之标签 name</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之标签 name</h3>
    <hr>
    <!-- 添加文档主体内容 -->
    <label>input 数组:</label><br>
    <input type="text" name="name-input" value="input"/><br>
    <input type="text" name="name-input" value="input"/><br>
    <input type="text" name="name-input" value="input"/><br>
    <input type="text" name="name-input" value="input"/><br>
    <input type="text" name="name-input" value="input"/><br><br>
```

```
27      input 数组长度: <input type="text" id="id-input" value=""/><br>
28      <script type="text/javascript">
29          var len = document.getElementsByName("name-input").length;
30          document.getElementById("id-input").value = len;
31      </script>
32  </body>
33  </html>
```

在这段 HTML 代码中，第 22～26 行定义了一组具有相同的 name 属性的 input 标签（属性值为"name-input"）作为标识，第 29～30 行脚本代码通过 length 属性获取了这组 input 标签的长度，并将数组长度值显示在第 27 行定义的 input 标签中。

下面我们运行测试这个页面，效果如图 1.31 所示。从图中可以看到，第 22～26 行定义的一组 input 标签共有 5 个，且具有相同的 name 属性，第 29 行脚本代码通过 length 属性获取的数组长度与定义的是一致的。

图 1.31　为标签添加 name 属性

下面我们来看一段关于定义 radio 类型的 input 标签的代码。如果想定义一组具有互斥功能的选项，最好的方法就是使用 radio 类型的 input 标签，这样用户选择选项时只能选中其中一个，也就是单项选择。这段代码（参见源代码 chapter01/ch01-htmlcomp-ele-radio.html 文件）给出通过标签 name 属性定义 radio 标签的方法。

【示例 1-29】定义 radio 类型的 input 标签

```
01  <!DOCTYPE html>
02  <html lang="en">
03  <head>
04      <!-- 添加文档头部内容 -->
05      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06      <style type="text/css">
07          input {
08              margin: 2px;
```

```
09              padding: 2px;
10              width: 64px;
11              border: 1px solid #000;
12          }
13      </style>
14      <title>HTML 之标签 radio</title>
15   </head>
16   <body>
17      <!-- 添加文档主体内容 -->
18      <h3>HTML 之标签 name</h3>
19      <hr>
20      <!-- 添加文档主体内容 -->
21      <label>请选择你喜爱的编程语言:</label><br><br>
22  <input type="radio" name="name-radio" value="HTML"
checked="checked"/>HTML<br>
23      <input type="radio" name="name-radio"
value="JavaScript"/>JavaScript<br>
24      <input type="radio" name="name-radio" value="CSS"/>CSS<br>
25      <input type="radio" name="name-radio" value="jQuery"/>jQuery<br>
26  <input type="radio" name="name-radio" value="jQuery Mobile"/>jQuery
Mobile<br><br>
27  </body>
28  </html>
```

在这段 HTML 代码中，第 22～26 行定义了一组 radio 类型的 input 标签，且具有相同的 name 属性（属性值为"name-radio"）作为标识，这一组 input 标签其实就是一组单选按钮。注意，这里的 name 属性必须具有相同的属性值，否则单项选择功能是无法实现的。

下面我们运行测试这个页面，效果如图 1.32 所示。读者可以测试这个页面，每次只能选择一门编程语言，也就是说当选择另一个选项时，之前默认选择的选项会被清除。

图 1.32　定义 radio 类型的 input 标签

1.7.3 为标签添加 class 属性

在 HTML 规范中，标签的 class 属性用于定义类名，一般该类名指向 CSS 样式表中定义的类（Class）。既然 CSS 样式表中的类可以被定义用于标签，我们就可以通过 JavaScript 脚本语言对其进行增加、修改或删除操作，实现动态改变样式类的功能。

当然，读者需要注意不是全部标签均可以使用 class 属性，譬如 base、html、head、script、style、title 等标签是不支持该属性的。

下面我们来看一段定义并操作 class 属性的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-ele-class.html 文件）给出通过脚本语言定义和修改标签 class 属性改变 CSS 样式的方法，是比较常规的一种操作方法。

【示例 1-30】为标签添加 class 属性

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- 添加文档头部内容 -->
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        p.classnormal {
            margin: 4px;
            padding: 2px;
            width: 196px;
            font: normal 13.5px/1.2em arial,verdana;
        }
        p.classitalic {
            margin: 4px;
            padding: 2px;
            width: 196px;
            font: italic 13.5px/1.2em arial,verdana;
        }
        p.classbold {
            margin: 4px;
            padding: 2px;
            width: 196px;
            font: bold 13.5px/1.2em arial,verdana;
        }
    </style>
    <title>HTML 之标签 class</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
```

```
    <h3>HTML 之标签 class</h3>
    <hr>
    <!-- 添加文档主体内容 -->
    <div>
        <button type="button" id="id-button-normal"
onclick="on_normal_click();">
            change to normal style
        </button>
        <button type="button" id="id-button-italic"
onclick="on_italic_click();">
            change to italic style
        </button>
        <button type="button" id="id-button-bold"
onclick="on_bold_click();">
            change to bold style
        </button>
    </div>
    <!-- 添加文档主体内容 -->
    <p class="classnormal">HTML 之标签 class</p>
    <p class="classnormal">HTML 之标签 class</p>
    <p class="classnormal">HTML 之标签 class</p>
    <p class="classnormal">HTML 之标签 class</p>
    <p class="classnormal">HTML 之标签 class</p>
    <script type="text/javascript">
        function on_normal_click() {
            for(i=0;i<5;i++) {

document.getElementsByTagName("p")[i].className="classnormal";
            }
        }
        function on_italic_click() {
           for(i=0;i<5;i++) {
            ocument.getElementsByTagName("p")[i].className = "classitalic";
           }
        }
        function on_bold_click() {
            for(i=0;i<5;i++) {
                document.getElementsByTagName("p")[i].className =
"classbold";
            }
        }
    </script>
```

```
67    </body>
68    </html>
```

在这段 HTML 代码中，通过定义和操作标签的 class 属性实现了动态改变 CSS 样式的效果。下面具体分析。

第 07～12 行、第 13～18 行与第 19～24 行分别定义了 3 个关于 p 标签的 CSS 样式类，名称分别为 classnormal、classitalic 和 classbold。这 3 个样式类的风格基本一致，不同的地方是字体不一样，分别使用 normal、italic 和 bold 三种字体。

第 33～43 行定义了 3 个 button 标签，分别用于执行 3 个脚本函数来完成动态修改字体的功能。其中，第 34 行定义的 on_normal_click()函数在第 51～55 行的脚本代码中实现；第 37 行定义的 on_italic_click()函数在第 56～60 行的脚本代码中实现；第 40 行定义的 on_bold_click()函数在第 61～65 行的脚本代码中实现。

第 45～49 行定义了一组 p 标签，并增加了 class 属性，其初始属性值定义为"classnormal"，该样式类在前面的第 07～12 行中定义。

下面我们运行测试这个页面，效果如图 1.33 所示。从中可以看到第 45～49 行中一组 p 标签定义的 class="classnormal"属性值的效果，字体全部为 normal 样式。

单击名称为 change to italic style 的 button 标签按钮，运行测试一下，页面变化的效果如图 1.34 所示。可以看到，一组 p 标签的字体风格全部改变为 class="classitalic"属性值的效果，这是因为第 58 行脚本代码通过 className 属性改变了一组 p 标签的 class 属性值。

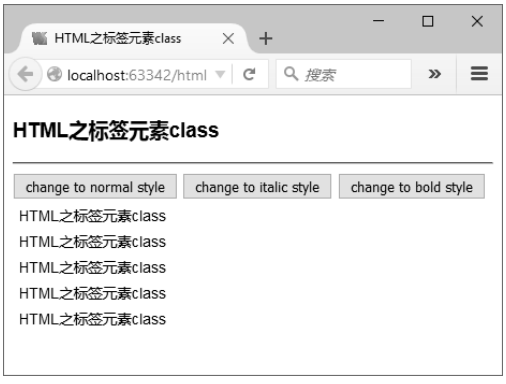

图 1.33　为标签添加 class 属性（1）

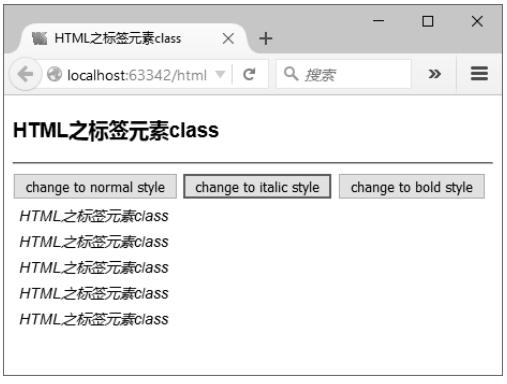

图 1.34　为标签添加 class 属性（2）

单击名称为 change to italic style 的 button 标签按钮，运行测试一下，页面变化的效果如图 1.35 所示。可以看到一组 p 标签的字体风格全部改变为 class="classbold"属性值的效果，同样，这是因为第 63 行脚本代码通过 className 属性再次改变了一组 p 标签的 class 属性值。如果再次单击名称为 change to normal style 的 button 标签按钮，页面就会返回初始的效果，如图 1.33 所示。

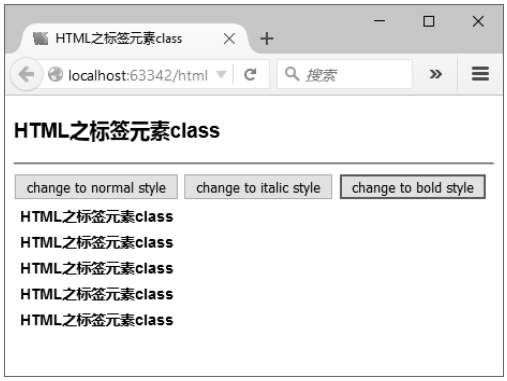

图 1.35　为标签添加 class 属性（3）

1.8 为标签添加 title 属性

本节向读者介绍 title 属性的使用方法。前文介绍 head 标签时讲过在 HTML 网页头部使用 title 标签的方法，而本节介绍的 title 属性与 title 标签是完全不同的。

在标签内使用 title 属性相当于为该标签附加上提示信息，其表现形式为当鼠标移到该标签区域时会显示一个工具条提示信息文本（Tooltip Text）。

我们先来看一下为标签添加 title 属性的语法：

```
<element title="value"></element>
```

如果读者想为标签添加 title 属性，一般需要遵循上面的写法。

下面看一段为标签添加 title 属性的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-title.html 文件）给出在 HTML 网页内为标签添加 title 属性的方法。

【示例 1-31】　在 HTML 网页内为标签添加 title 属性

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- 添加文档头部内容 -->
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        div {
            margin: 2px;
            padding: 2px;
            width: auto;
            height: auto;
            border: 0px solid #000;
            background: #f0f0f0;
```

```
14          }
15      </style>
16      <title>HTML 之为标签添加 title 属性</title>
17  </head>
18  <body>
19      <!-- 添加文档主体内容 -->
20      <h3>HTML 之为标签添加 title 属性</h3>
21      <hr>
22      <!-- 添加文档主体内容 -->
23      <p>
24    层叠样式表(<abbr title="Cascading Style Sheets">CSS</abbr>)目前最新版本为CSS 3.
25      </p>
26      <p>
27          <a href="#" title="链接到...">链接到...</a>
28      </p>
29      <!-- 添加文档主体内容 -->
30      <div>
31          表单提交：<br><br>
32          <form>
33              用户名：<input type="text" title="请输入用户名" value="" /><br><br>
34              密码：<input type="password" title="请输入密码" value="" /><br><br>
35          </form>
36      </div>
37  </body>
38  </html>
```

在这段 HTML 代码中，第 24 行为 abbr 标签添加了 title 属性，其中 abbr 标签属性用于标记一个缩写。在 HTML 标准规范中，abbr 标签使用全局的 title 属性，同时必须配合 title 属性来使用。这样，当鼠标移到 abbr 缩写区域上时，就可以提示该缩写的完整内容。

下面运行测试这个页面，并将鼠标移到 CSS 缩写上，效果如图 1.36 所示。

图 1.36　在 HTML 网页内为 abbr 标签添加 title 属性

从图 1.36 中看到，当鼠标移到 CSS 缩写上时，其完整内容 Cascading Style Sheets 就会以工具条提示信息的方式自动显示出来。

接着看【示例 1-31】这段 HTML 代码，第 27 行为超链接<a>标签添加了 title 属性。在 HTML 标准规范中，一般建议给 a 标签添加 title 属性，这是为了给用户提供更多的提示信息。

下面将鼠标移到超链接“链接到…”区域上，页面效果如图 1.37 所示。当鼠标移到“链接到…”区域上时，其提示信息“链接到…”同样以工具条提示信息的方式自动显示出来。最后，将鼠标移到表单区域中的输入框区域，页面效果如图 1.38 所示。当鼠标移到“用户名”输入框区域时，其提示信息“请输入用户名”同样以工具条提示信息的方式自动显示出来。在表单中合理使用 title 属性可以为用户在填写表单时提供更多的提示与帮助信息。

图 1.37　在 HTML 网页内为 a 标签添加 title 属性

图 1.38　在 HTML 网页内为 form 表单元素添加 title 属性

1.9 添加网页注释

本节介绍在 HTML 网页中添加注释的方法。HTML 网页中被注释的内容是不会显示在浏览器中的，这样可以有效避免页面中想隐藏的内容被显示出来，这就是注释的功能所在。

在 HTML 网页中使用注释的优点有很多：比如为代码添加注释，既可以方便自己后期修改维护，又可以方便其他程序员阅读理解并完善你写的代码；又比如将暂时不需要执行的代码先注释掉，这样以后想重新恢复代码时就很简单。当然，最关键的一点是，一段优秀的代码配上合理必要的注释才算完美，这是一个优秀的程序员必备的良好习惯之一。

如果读者想在 HTML 网页中使用注释，就需要使用“<!-- -->”符号，并遵循下面的写法：

```
<!-- comment -->
```

只有上面这种符号对 HTML 网页代码起注释作用，而像“//”和“/* */”符号也会出现在 HTML 网页中，但只会对 JavaScript 脚本代码和 CSS 样式代码起作用。

下面我们看一段使用注释的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-

comment.html 文件）给出在 HTML 网页内使用注释的方法。

【示例 1-32】在 HTML 网页中添加注释

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- 添加文档头部内容 -->
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <!-- 添加 CSS 样式代码 -->
    <style type="text/css">
        div {
            margin: 2px;
            padding: 2px;
            width: auto;
            height: auto;
            border: 1px solid #e0e0e0;
            background: #f0f0f0;
        }
        .classNormal {
            font-style: normal;     /* 定义字体风格 */
        }
        .classBold {
            font-weight: bold;      /* 定义字体风格 */
        }
        .classItalic {
            font-style: italic;         /* 定义字体风格 */
        }
        .classLarge {
            font-size: x-large;     /* 定义字体风格 */
        }
    </style>
    <script type="text/javascript">
        function funcComment() {
            alert("ok");        // TODO: 警告消息框
        }
    </script>
    <title>HTML 之使用注释</title>
</head>
<body>
    <!-- 添加文档主体标题 -->
    <h3>HTML 之为标签添加 title 属性</h3>
    <hr>
```

```
40      <!-- 添加文档 div 层 -->
41      <div>
42          <!-- 添加 classNormal 字体风格 -->
43          <p class="classNormal">class font bold</p>
44          <!-- 添加 classBold 字体风格 -->
45          <p class="classBold">class font bold</p>
46          <!-- 添加 classItalic 字体风格 -->
47          <p class="classItalic">class font italic</p>
48          <!-- 添加 classLarge 字体风格 -->
49          <p class="classLarge">class font italic</p>
50          <!-- 添加 classNormal 字体风格
51          <p class="classNormal">class font bold</p>
52      </div>
53  </body>
54  </html>
```

在这段 HTML 代码中，有多处使用到了注释。下面我们详细介绍。

第 04 行的注释说明下面的代码用于添加头部内容。

第 06 行的注释说明下面一段代码用于定义 CSS 样式。

第 17 行、第 20 行、第 23 行和第 26 行后面的 CSS 注释使用的是“/* */”符号。

第 31 行脚本代码后面的 JavaScript 注释使用的是“// TODO:”符号。

第 42 行、第 44 行、第 46 行和第 48 行的注释说明其后的代码添加了 4 种不同风格的 CSS 样式字体。

而第 50～51 行原本想完成与第 42～43 行同样的功能，但第 50 行注释代码中注释的结尾符号“-->”不小心没写完，结果将第 51～54 行全部注释了。

下面运行测试这个页面，效果如图 1.39 所示。从图中看到，由于第 50 行的错误，第 51 行的内容没有显示出来；同时，虽然第 52～54 行被注释掉了，但并没有影响 HTML 网页的正常输出。

图 1.39　在 HTML 网页中添加注释的效果

1.10 测试浏览器对 HTML 5 属性的支持

对于 HTML 开发设计人员来讲，浏览器的兼容性是一个既复杂又不可回避的问题。随着技术的进步，目前市面上的主流浏览器对 HTML 的支持已经很完善了，不像早期浏览器的兼容性让开发设计人员伤透了脑筋。

本节介绍浏览器对 HTML 属性的支持问题，包括对最新的 HTML 5 属性的支持。HTML 5 是一个全新的标准，增加了很多新的特性，对多媒体的支持更全面。因此，浏览器对 HTML 5 属性的支持是判断其兼容性的重要指标。

下面我们来看一段判断浏览器是否支持 HTML 某个属性的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-support-prop.html 文件）给出判断 HTML 常用属性的方法。

【示例 1-33】判断 HTML 常用属性

```
<!doctype html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        div {
            margin: 2px;
            padding: 2px;
            width: auto;
            height: auto;
            border: 1px solid #e0e0e0;
            background: #f0f0f0;
        }
    </style>
    <script type="text/javascript">
        function isSupport(prop) {
            return prop in document.createElement('div');
        }
    </script>
    <title>HTML 之判断支持属性</title>
</head>
<body>
    <!-- 添加文档内容 -->
    <h1>HTML 之判断支持属性</h1>
    <hr>
    <!-- 添加文档内容 -->
    <div id="id-div"></div>
    <!-- 添加脚本代码 -->
    <script type="text/javascript">
        var prop = ["id", "name", "value", "type", "style", "title"];
```

```
31          for(var I in prop) {
32              isSupportProp(prop[i]);
33          }
34          function isSupportProp(v) {
35              if(isSupport(v)) {
36                  document.getElementById("id-div").innerHTML +="层(div)标签支持"
+ v + "属性" + "<br>";
37              } else {
38                  document.getElementById("id-div").innerHTML +="层(div)标签不支
持" + v + "属性" + "<br>";
39              }
40          }
41      </script>
42  </body>
43  </html>
```

在这段 HTML 代码中，判断了 div 标签是否支持一些常用的 HTML 属性。下面我们详细介绍。

第 15～19 行定义了一个 JavaScript 脚本函数 isSupport(prop)，第 17 行通过 createElement()函数创建一个 div 标签，并使用 in 方法将属性判断结果返回。

第 30 行定义了一个包含 6 个常用的 HTML 属性的数组 prop。

第 31～33 行通过 for 循环语句依次判断数组 prop 中的属性是否为 div 标签所支持的，具体通过 isSupportProp()函数来判断。

第 34～42 行是 isSupportProp()函数的实现过程，在该函数内部通过调用第 15～19 行定义的 isSupport(prop)函数来实现判断。

下面运行测试这个页面，效果如图 1.40 所示。从图中看到，层（div）标签支持 id、style 和 title 属性，不支持 name、type 和 value 属性。

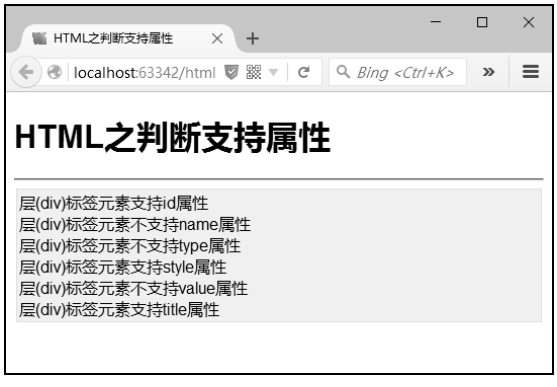

图 1.40　判断 HTML 常用属性的结果

下面我们看一段判断浏览器是否支持 HTML 5 属性的代码，这段代码（参见源代码 chapter01/ch01-htmlcomp-support-HTML 5.html 文件）给出判断 HTML 5 属性的方法。

【示例 1-34】判断浏览器是否支持 HTML 5 属性

```
<!doctype html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/javascript">
        if(typeof(Worker) !== "undefined") {
            // Yes! Web worker support!
            alert("正在使用的浏览器支持 HTML 5属性");
        }
        else {
            // Sorry! No Web Worker support!
            alert("正在使用的浏览器不支持 HTML 5属性");
        }
    </script>
    <title>浏览器之支持 HTML 5属性</title>
</head>
<body>
    <!-- 添加文档内容 -->
    <h1>浏览器之支持 HTML 5属性</h1>
    <hr>
</body>
</html>
```

在这段 HTML 代码中，判断浏览器是否支持 HTML 5 属性主要使用了 Web Worker 属性。第 06 行通过 typeof()方法判定 Worker 属性是否未定义（"undefined"），如果定义了，就判定浏览器支持 HTML 5 属性。

下面使用最新版的 FireFox 浏览器（v.66.0 版）运行测试这个页面，效果如图 1.41 所示。从图中看到，新版 FireFox 浏览器对 HTML 5 是支持的。

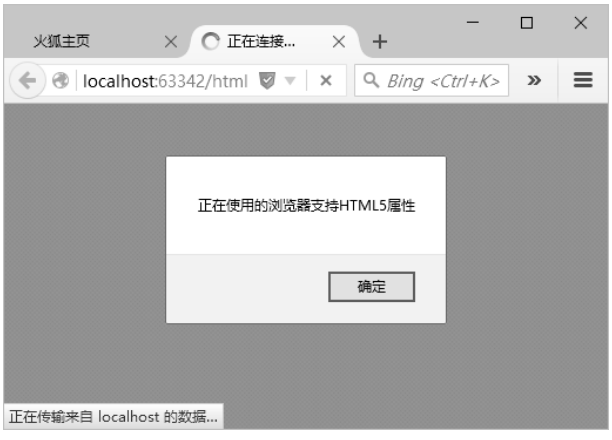

图 1.41　判断 FireFox 浏览器是否支持 HTML 5 属性

下面再使用最新版的 Microsoft Edge 浏览器（Windows 10 预览版自带）来运行测试这个页面，效果如图 1.42 所示。从图中看到，Microsoft Edge 浏览器对 HTML 5 是支持的。看来不支持 HTML 5 的浏览器只有早期 Windows XP 系统下的 IE 6、IE 7 和 IE 8 了。

图 1.42　判断 Microsoft Edge 浏览器是否支持 HTML 5 属性

1.11 案例实战：一个完整的 HTML 5 网页应用

在前面几节中，我们逐步介绍了 HTML 网页架构中方方面面的内容，让读者对于 HTML 网页有了基本的了解。

本节将向读者介绍一个完整的 HTML 网页应用，其中包含 HTML 页面、CSS 样式和 JavaScript 脚本的全部内容，是对前面内容的一个小结。另外，为了展现出一个美观的 HTML 页面，我们加入了流行的 Bootstrap 样式框架和 jQuery 脚本语言框架。读者先不用深入了解这两个框架的详细内容，只需要知道其实现了什么效果就可以了。

下面具体介绍这个 HTML 网页应用，这段代码（参见源代码 chapter01/all/index.html 文件）实现该 HTML 网页应用的主页。

【示例 1-35】　一个完整的 HTML 网页应用

```
<!DOCTYPE html>
<html lang="en">
 <head>
 <!-- meta define -->
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 <!-- css define -->
 <link rel="stylesheet" href="css/bootstrap.min.css">
 <!-- js define -->
 <script src="js/jquery.min.js"></script>
 <script src="js/bootstrap.min.js"></script>
 <title>一个完整的 HTML 网页</title>
 </head>
 <body>
```

```
<div class="container">
<header id="site-header">
<div class="row">
<div class="col-md-4 col-sm-5 col-xs-8">
<div class="logo">
<h3><a href="#"><b>HTML</b> &amp; <b>CSS</b> &amp; <b>JS</b></a></h3>
</div>
</div><!-- col-md-4 -->
<div class="col-md-8 col-sm-7 col-xs-4">
<nav class="main-nav" role="navigation">
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
    <ul class="nav navbar-nav navbar-right">
    <li class="cl-effect-11"><a href="#" data-hover="Home">主页</a></li>
    <li class="cl-effect-11"><a href="#" data-hover="Blog">博客</a></li>
    <li class="cl-effect-11"><a href="#" data-hover="Contact">联系我们
</a></li>
    <li class="cl-effect-11"><a href="#" data-hover="About">关于</a></li>
    </ul>
</div>
</nav>
</div>
</div>
</header>
</div>
<div class="content-body">
<div class="container">
<div class="row">
<main class="col-md-8">
<article class="post post-1">
<header class="entry-header">
<h1 class="entry-title">
<a href="single.html">一个 HTML + CSS + JavaScript 完整主页</a>
</h1>
<div class="entry-meta">
<span class="post-category"><a href="#">Design</a></span>
<span class="post-author"><a href="#">KING</a></span>
<span class="post-date">
    <a href="#"><time class="entry-date">May 10, 2019</time></a>
</span>
</div>
</header>
<div class="entry-content clearfix">
```

```
<p>一个 HTML + CSS + JavaScript 完整主页.</p>
<p>一个 HTML + CSS + JavaScript 完整主页.</p>
<p>一个 HTML + CSS + JavaScript 完整主页.</p>
<div class="read-more cl-effect-14">
<a href="#" class="more-link">继续浏览<span class="meta-nav">→</span></a>
</div>
</div>
</article>
</main>
<aside class="col-md-4">
<div class="widget widget-archives">
<h3 class="widget-title">目录</h3>
<ul>
    <li><a href="#">目录 2019</a></li>
    <li><a href="#">目录 2018</a></li>
    <li><a href="#">目录 2017</a></li>
</ul>
</div>
<div class="widget widget-category">
<h3 class="widget-title">分类</h3>
<ul>
    <li><a href="#">分类 2019</a></li>
    <li><a href="#">分类 2018</a></li>
    <li><a href="#">分类 2017</a></li>
</ul>
</div>
</aside>
</div>
</div>
</div>
<footer id="site-footer"></footer>
<script src="js/script.js"></script>
</body>
</html>
```

在这段 HTML 代码中，总体上应用 DIV+CSS 分层结构将页面划分为 4 个模块。下面我们详细介绍。

第一个模块是页面的头部，主要通过第 15～35 行定义了一个 header 层。其中，第 19 行定义了页面标题，第 25～30 行定义了一个水平方向的导航菜单。

第二个和第三个模块构成页面的主体部分。其中，第二个模块主要通过第 40～63 行定义了一个 main 层，实现了页面的主体内容部分；第三个模块主要通过 64～81 行定义了 aside 层，实

现了一个垂直方向的导航菜单。

第四个模块是页面的底部，主要通过 85 行定义了一个 footer 层，其中主要包括页面注册信息、版权信息和作者信息等内容。

另外，第 07 行引入了外部的 CSS 样式表（参见源代码 chapter01/all/css 文件夹中的样式表文件），也就是前面提到的 Bootstrap 样式框架；第 09～10 行引入了外部的 JavaScript 脚本库（参见源代码 chapter01/all/js 文件夹中的脚本库文件），也就是前面提到的 jQuery 脚本语言框架。

下面运行测试这个页面，效果如图 1.43 所示。从图中看到，整个 HTML 页面层次结构非常清晰，头部、主体（内容主体与垂直方向导航菜单）和底部模块依次排列在页面中，后续开发更复杂的页面内容时可以继续以此框架为基础。

图 1.43　一个完整的 HTML 网页应用效果

1.12 小结

本章主要介绍了 HTML 网页的架构基础，包括 HTML 网页页面的构成、HTML 主要标签的使用方法以及 HTML 与 CSS 样式表和 JavaScript 脚本代码结合使用的内容，并在最后介绍了一个综合的 HTML 页面应用，该应用包含 HTML 网页架构中方方面面的内容，还结合使用了流行的前端框架，可以对读者进行 HTML 网页深入开发有所启迪。

第 2 章

◂ HTML网页的基本标签 ▸

本章介绍 HTML 网页的基本标签。一般来说，HTML 网页中的基本标签包括段落、文字、符号与编号、注释、特殊符号和超链接等内容。通过这些基本的标签就可以构建出一个功能完整的 HTML 网页，并实现与用户进行基本交互的功能。

本章主要包括以下内容：

- 段落排版
- 文字效果
- 项目符号与编号
- 注释与特殊符号
- 超链接

2.1 HTML 网页段落排版

无论是视频网站还是门户网站，网页中都避免不了段落文字。门户网站就是我们常说的新浪、搜狐等，网页中多是一些图文性质的新闻，如果不进行排版，我们估计就要看花眼了。

2.1.1 设置段落样式的标记

HTML 网页中的段落是通过 标签来定义的。HTML 网页中的段落类似于我们常说的文章写作中的自然段，是 HTML 网页中一个非常重要的元素。

在浏览器中展示段落 标签时会自动为每一个段落的前后添加空行。同时，建议带上结束标签（即使不小心忘了使用结束标签，目前的浏览器也会将 HTML 网页正确显示出来），一是保持良好的代码习惯；二是浏览器有可能会出现无法正确解析 HTML 页面的问题。

下面我们看一段设置段落样式标记的代码，这段代码（参见源代码 chapter02/ch02-htmltag-p-style.html 文件）给出一种设置段落样式标记的方法。

【示例 2-1】 设置段落样式的标记

```
01 <!DOCTYPE html>
02 <html lang="zh-cn">
```

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之设置段落样式的标记</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之设置段落样式的标记</h3>
    <!-- 添加文档主体内容 -->
    <p>当前网页：ch01-htmltag-p-style.html</p>
    <!-- 添加文档主体内容 -->
    <p style="font-style: italic;font-size: larger">
        当前网页：ch01-htmltag-p-style.html
    </p>
</body>
</html>
```

在这段 HTML 代码中，对段落 p 标签设置了样式：第 11 行代码使用 p 标签定义了一个段落，其内容介绍了当前的 HTML 页面名称；第 13～15 行代码与第 11 行代码一样定义了一个段落，不同之处是在 标签内使用 style 属性定义了字体样式（font-style: italic;font-size: larger），虽然这两个段落的内容一致，但显示出来的字体风格会有差异。

运行测试这个页面，效果如图 2.1 所示。从图中看到，由于第 13～15 行代码定义了段落样式，因此页面中显示出来的字体风格产生了变化。

图 2.1　设置段落样式标记的效果

2.1.2　设置对齐与缩进的标记

在页面中使用 标签展示段落时，对齐（text-align）与缩进（text-intend）功能设置是必不可少的，就像小学生写作文一样，老师会强调作文的格式。

下面我们看一段设置对齐与缩进标记的代码，这段代码（参见源代码 chapter02/ch02-htmltag-p-intend.html 文件）给出一种设置段落对齐与缩进标记的方法。

【示例 2-2】　设置对齐与缩进的标记

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之设置段落对齐与缩进</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之设置段落对齐与缩进</h3>
    <!-- 添加文档主体内容 -->
    <p style="text-align: justify;text-indent: 2em;">
        段落的对齐和缩进是排版最常用的方法，也是要求学生重点掌握的内容。
        这部分内容有“段落对齐”、“段落缩进”等几个知识点。
    </p>
    <p style="text-align: left;text-indent: 0em;">
        “段落对齐”中对齐方式有“左对齐”、“居中”、
        “右对齐”、“两端对齐”、“分散对齐”几种。
    </p>
    <p style="text-align: right;text-indent: 4em;">
        “段落缩进”主要包括“左缩进”、“右缩进”等几种，
        使用“段落缩进”方法可以让自然段落更美观。
    </p>
</body>
</html>
```

在这段 HTML 代码中，对 3 个段落的 p 标签设置了对齐与缩进。下面我们详细介绍。

第 11～14 行代码为第一个段落，在<p>标签内使用 style 属性定义了对齐与缩进样式（text-align: justify;text-indent: 2em;），其中 justify 表示两端对齐，而缩进的尺寸为两个相对字符长度（2em）。

第 15～18 行代码为第二个段落，在<p>标签内使用 style 属性定义了对齐与缩进样式（text-align: left;text-indent: 0em;），其中 left 表示左对齐，而缩进的尺寸为零个相对字符长度（0em）。

第 19～22 行代码为第三个段落，在<p>标签内使用 style 属性定义了对齐与缩进样式（text-align: right;text-indent: 4em;），其中 right 表示右对齐，而缩进的尺寸为零个相对字符长度（4em）。

运行测试这个页面，效果如图 2.2 所示。从图中看到，两端对齐的风格是比较美观的，而左右对齐的风格可以应用到特殊的场景中。同时，缩进的长度不宜过大或过小，本段代码中设置的两个相对字符长度还是比较合适的。

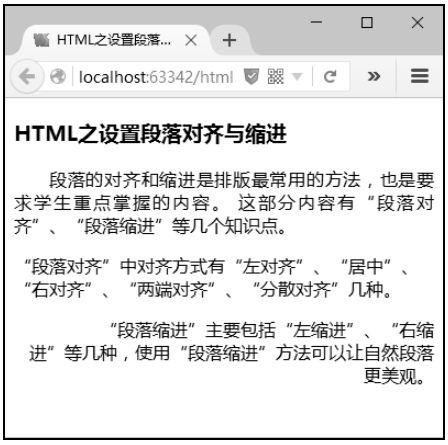

图 2.2　设置对齐与缩进标记的效果

2.1.3　添加分割线

在页面中使用 hr 分割线标签是很常见的，譬如在网页底部通常用一条分割线将公司信息、作者信息、版权信息和注册备案信息分割开来，以示和网页主体部分的区分。

下面我们看一段设置分割线的代码，这段代码（参见源代码 chapter02/ch02-htmltag-hr.html 文件）给出一种设置页面底部分割线的方法。

【示例 2-3】　添加分割线

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之设置分割线</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之设置分割线</h3>
    <!-- 添加文档主体内容 -->
    <p>页面正文</p>
    <!-- 添加文档底部内容 -->
    <hr>
    <hr style="height:2px;border:dashed;">
    <hr style="height:4px;border:double;">
    <div style="text-align: center">
        <p class="copyright">&copy; 2019 HTML CSS JavaScript
            <a href="#" target="_blank" title="KING">KING</a>
        </p>
    </div>
```

```
21 </body>
22 </html>
```

在这段 HTML 代码中，在页面底部（第 16～20 行代码）上方添加了 3 条不同风格的分割线：第 13 行代码添加第一条分割线，是没有添加任何风格的原始样式分割线；14 行代码添加第二条分割线，设置了分割线高度（2px）和虚线（dashed）边框样式；第 15 行代码添加第三条分割线，设置了分割线高度（4px）和双虚线（double）边框样式。

运行测试这个页面，效果如图 2.3 所示。

图 2.3　添加分割线的效果

2.1.4　设置段落标题

前面几个小节介绍了段落的相关内容，这个小节介绍为段落添加标题的方法，段落加上标题才会组成完整的文章。

下面这段代码（参见源代码 chapter02/ch02-htmltag-p-hx.html 文件）给出一种设置段落标题的方法。

【示例 2-4】　设置段落标题

```
01 <!DOCTYPE html>
02 <html lang="zh-cn">
03 <head>
04     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05     <title>HTML 之设置段落标题</title>
06 </head>
07 <body>
08     <!-- 添加文档主体内容 -->
09     <h3 style="text-align: center">文章大标题</h3>
10     <h4>段落标题</h4>
11     <p style="text-align: justify;text-indent: 2em;">
12         段落的对齐和缩进是排版最常用的方法，也是要求学生重点掌握的内容。
13         这部分内容有“段落对齐”、“段落缩进”等几个知识点。
14     </p>
```

```
15      <h4>段落标题</h4>
16      <p style="text-align: justify;text-indent: 2em;">
17          段落的对齐和缩进是排版最常用的方法，也是要求学生重点掌握的内容。
18          这部分内容有“段落对齐”、“段落缩进”等几个知识点。
19      </p>
20  </body>
21  </html>
```

在这段 HTML 代码中，添加了文章大标题和段落小标题：第 09 行代码为文章添加了居中对齐的大标题；第 10 行与第 15 行代码为两个段落添加了小标题。

运行测试这个页面，效果如图 2.4 所示。

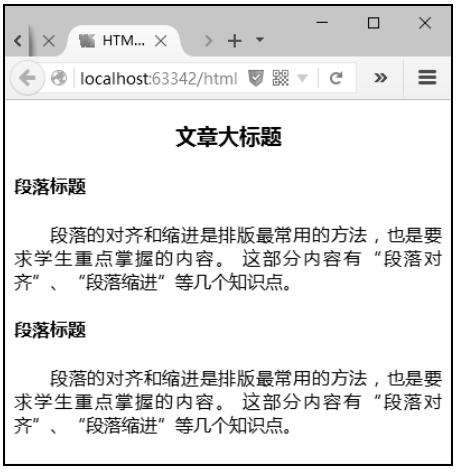

图 2.4　添加段落标题的效果

2.2 文字效果

网页中的文字效果一般包括文字的字体、字号、上标、下标等，本节主要通过小例子来演示网页中不同的文字展现形式。

2.2.1 设置字形样式的标记

在 HTML 网页中可以展现出风格多样的字形样式，一般通过设置 CSS 的 font-family 属性就可以实现。

下面我们看一段设置字形样式的代码，这段代码（参见源代码 chapter02/ch02-htmltag-font-family.html 文件）给出一种设置不同风格字形的方法。

【示例 2-5】　设置字形样式的标记

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
        p {
            text-align: justify;
            text-indent: 2em;
        }
    </style>
    <title>HTML 之设置字形样式</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之设置字形样式</h3>
    <!-- 添加文档主体内容 -->
    <p style="font-family: cursive;">
        段落的对齐和缩进是排版最常用的方法，也是要求学生重点掌握的内容。
        这部分内容有“段落对齐”、“段落缩进”等几个知识点。
    </p>
    <p style="font-family: '黑体';">
        段落的对齐和缩进是排版最常用的方法，也是要求学生重点掌握的内容。
        这部分内容有“段落对齐”、“段落缩进”等几个知识点。
    </p>
    <p style="font-family: '幼圆';">
        段落的对齐和缩进是排版最常用的方法，也是要求学生重点掌握的内容。
        这部分内容有“段落对齐”、“段落缩进”等几个知识点。
    </p>
</body>
</html>
```

在这段 HTML 代码中，对 3 个段落设置了 3 种字形样式：第 17 行代码为第 17～20 行代码的段落定义了 cursive 字形样式，该字形与 Serif 和 Sans-serif 一样为通用样式；第 21 行代码为第 21～24 行代码的段落定义了“黑体”字形样式，该字形为特定样式；第 25 行代码为第 25～28 行代码的段落定义了“幼圆”字形样式，该字形也为特定样式。

运行测试这个页面，效果如图 2.5 所示。

图 2.5　设置字形样式标记的效果

在网页代码的头部建议将字符编码设置成 utf-8 编码，这样可以避免出现一些不必要的乱码现象。

2.2.2　设置上标和下标

在 HTML 网页中有时需要定义上标字体和下标字体，譬如在引用文献时肯定要用到上标字体，而定义数理化等元素符号时下标字体是必不可少的。HTML 规范设计了 sup 标签表示上标，sub 标签表示下标。

下面我们看一段使用上标和下标的代码，这段代码（参见源代码 chapter02/ch02-htmltag-sup-sub.html 文件）给出一种设置字体上标和下标的方法。

【示例 2-6】　设置上标和下标

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之字体上下标</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之字体上下标</h3>
    <!-- 添加文档主体内容 -->
    <p style="text-align: justify;text-indent: 2em;">
        引用文献<sup>【1】</sup>是关于 HTML 上标标签的。
    </p>
    <p style="text-align: justify;text-indent: 2em;">
        H<sub>2</sub>O 是代表水分子元素符号的。
```

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

在这段 HTML 代码中，分别使用了上、下标标记：第 12 行代码使用 sup 标签定义了上标标记【1】，用于表示引用文献的序号；第 15 行代码使用 sub 标签定义了下标标记 2，用于表示水分子化学符号中氢元素（H）的分子量。

运行测试这个页面，效果如图 2.6 所示。

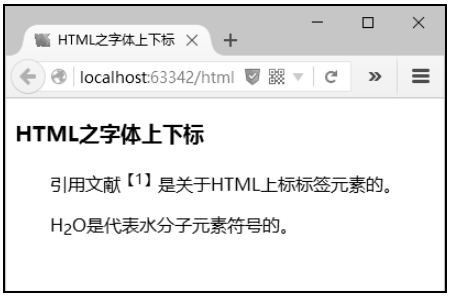

图 2.6　设置上、下标标记的效果

2.3 项目符号与编号

我们经常在网页中看到一些并列内容或有层次的文字，通常为了直观地显示这些内容，我们会用到项目符号和编号，这跟 Word 中的项目符号和编号类似。

2.3.1 符号列表

符号列表在文档中是一种比较常见的表现形式，符号可以定义为多种样式，譬如圆点符号、星型符号、箭头符号等。在 HTML 网页中，我们可以通过 ul-li 标签来实现无序的符号列表。

下面我们看一段符号列表的代码，这段代码（参见源代码 chapter02/ch02-htmltag-ul-li.html 文件）给出一种设置不同风格符号列表的方法。

【示例 2-7】　符号列表

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之符号列表</title>
</head>
<body>
```

```
08      <!-- 添加文档主体内容 -->
09      <h3>HTML 之符号列表</h3>
10      <!-- 添加文档主体内容 -->
11      <ul type="disc">
12          <li>列表1</li>
13          <li>列表2</li>
14          <li>列表3</li>
15      </ul>
16      <ul type="circle">
17          <li>列表1</li>
18          <li>列表2</li>
19          <li>列表3</li>
20      </ul>
21      <ul type="square">
22          <li>列表1</li>
23          <li>列表2</li>
24          <li>列表3</li>
25      </ul>
26  </body>
27  </html>
```

在这段 HTML 代码中，定义了 3 种样式的符号列表：第 11～15 行代码使用 ul 标签定义了 type="disc"样式的符号列表，"disc"样式表示实心圆点，同时该样式为 ul 标签的默认样式；第 16～20 行代码使用 ul 标签定义了 type="circle"样式的符号列表，"circle"样式表示空心圆圈；第 21～25 行代码使用 ul 标签定义了 type="square"样式的符号列表，"square"样式表示实心方块。

运行测试这个页面，效果如图 2.7 所示。

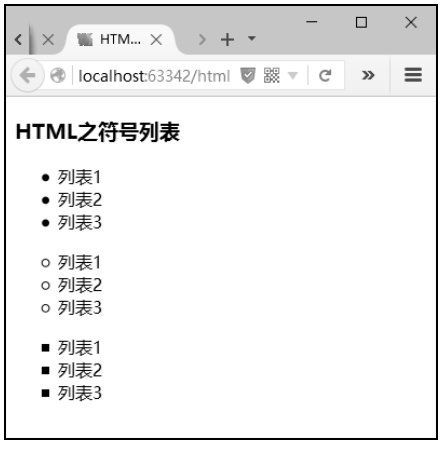

图 2.7　符号列表

另外，HTML 还支持一些特殊符号的列表，感兴趣的读者可以参阅相关文档进行进一步的学习研究。

2.3.2　编号列表

编号列表在文档中是一种比较常见的表现形式，同时编号可以定义为多种样式，譬如阿拉伯数字、罗马数字、字母等。在 HTML 网页中，我们可以通过 ol-li 标签来实现有序的编号列表。

下面我们看一段编号列表的代码，这段代码（参见源代码 chapter02/ch02-htmltag-ol-li.html 文件）给出一种设置不同风格编号列表的方法。

【示例 2-8】　编号列表

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之编号列表</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之编号列表</h3>
    <hr>
    <!-- 添加文档主体内容 -->
    <ol type="1">
        <li>列表1</li>
        <li>列表2</li>
        <li>列表3</li>
    </ol>
    <hr>
    <ol type="I">
        <li>列表 I</li>
        <ol type="i">
            <li>列表 i</li>
            <li>列表 ii</li>
        </ol>
        <li>列表 II</li>
    </ol>
    <hr>
    <ol type="A">
        <li>列表 A</li>
        <ol type="a">
            <li>列表 a</li>
            <li>列表 b</li>
        </ol>
        <li>列表 B</li>
```

```
34      </ol>
35  </body>
36  </html>
```

在这段 HTML 代码中，定义了 5 种样式的编号列表。下面我们详细介绍。

第 12～16 行代码使用 ol 标签定义了 type="1"样式的编号列表，"1"样式表示阿拉伯数字，同时该样式是 ol 标签的默认样式。

第 18～25 行代码使用 ol 标签定义了 type="I"样式的编号列表，"I"样式表示大写罗马数字；同时，第 20～23 行代码定义了 type="i"样式的二级编号列表，"i"样式表示小写罗马数字。

第 27～34 行代码使用 ol 标签定义了 type="A"样式的编号列表，"A"样式表示大写字母；同时，第 29～32 行代码定义了 type="a"样式的二级编号列表，"a"样式表示小写字母。

运行测试这个页面，效果如图 2.8 所示。

图 2.8　编号列表

另外，HTML 还支持一些复杂的编号符号的列表，感兴趣的读者可以参阅相关文档进行进一步的学习研究。

2.3.3　自定义列表

自定义列表不仅仅是一列项目，同时也是项目及其注释的组合。自定义列表以 dl 标签开始，每个自定义列表项以 dt 标签开始，每个自定义列表项的定义以 dd 标签开始。

下面我们看一段自定义列表的代码，这段代码（参见源代码 chapter02/ch02-htmltag-dl-dt-dd.html 文件）给出一种设置自定义列表的方法。

【示例 2-9】　自定义列表

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之自定义列表</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之自定义列表</h3>
    <hr>
    <!-- 添加文档主体内容 -->
    <dl>
        <dt>HTML</dt>
        <dd>HyperText Mark-up Language</dd>
        <dt>CSS3</dt>
        <dd>Cascading Style Sheets 3</dd>
        <dt>JavaScript</dt>
        <dd>一种直译式脚本语言</dd>
    </dl>
</body>
</html>
```

在这段 HTML 代码中，定义了一种自定义列表：第 12～19 行代码使用 dl 标签定义了自定义列表；第 13 行、第 15 行和第 17 行代码使用 dt 标签定义了自定义列表项；第 14 行、第 16 行和第 18 行代码使用 dd 标签定义了上面 3 个自定义列表项的注释。

运行测试这个页面，效果如图 2.9 所示。

图 2.9　自定义列表

另外，自定义列表的列表项内部可以使用段落、换行符、图片、链接以及其他列表等。

2.4 使用特殊符号

本节介绍 HTML 规范标准的特殊符号，这些特殊符号可能不是很常用，但在一些特殊情况下不得不使用。了解这些特殊符号的使用方法可以帮助我们解决很多复杂的问题。

下面这段代码（参见源代码 chapter02/ch02-htmltag-specchar.html 文件）给出一种在 HTML 文档内使用特殊符号的方法。

【示例 2-10】 在 HTML 网页内使用特殊符号

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之使用特殊符号</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 之使用特殊符号</h3>
    <!-- 添加文档主体内容 -->
    <table  border="0" align="center" cellpadding="2" cellspacing="1"
bgcolor="#F8F8F8">
        <tr>
            <td width="109" height="28" bgcolor="#E9F8E7">
                <div align="center">特殊符号</div></td>
            <td width="139" bgcolor="#F4FBF3" >
                <div align="center">命名实体</div></td>
            <td width="128" bgcolor="#F4FBF3" >
                <div align="center">十进制编码</div></td>
            <td width="109" bgcolor="#E9F8E7">
                <div align="center">特殊符号</div></td>
            <td width="139" bgcolor="#F4FBF3" >
                <div align="center">命名实体</div></td>
            <td width="129" bgcolor="#F4FBF3" >
                <div align="center">十进制编码</div></td>
        </tr>
        <tr>
            <td width="109" height="28" bgcolor="#E9F8E7">
                <div align="center">&diams;       </div></td>
            <td width="139" bgcolor="#F4FBF3" >
                <div align="center">&amp;diams; </div></td>
            <td width="128" bgcolor="#F4FBF3" >
                <div align="center">&amp;#9830; </div></td>
            <td width="109" bgcolor="#E9F8E7"><div
align="center"></div></td>
```

```
34            <td width="139" bgcolor="#F4FBF3" >
35               <div align="center">&amp;nbsp; </div></td>
36            <td width="129" bgcolor="#F4FBF3" >
37               <div align="center">&amp;#160; </div></td>
38         </tr>
39         <tr>
40            <td width="109" height="28" bgcolor="#E9F8E7">
41               <div align="center">&iexcl;       </div></td>
42            <td width="139" bgcolor="#F4FBF3" >
43               <div align="center">&amp;iexcl; </div></td>
44            <td width="128" bgcolor="#F4FBF3" >
45               <div align="center">&amp;#161; </div></td>
46            <td width="109" bgcolor="#E9F8E7">
47               <div align="center">&cent;       </div></td>
48            <td width="139" bgcolor="#F4FBF3" >
49               <div align="center">&amp;cent; </div></td>
50            <td width="129" bgcolor="#F4FBF3" >
51               <div align="center">&amp;#162; </div></td>
52         </tr>
53      </table>
54   </body>
55   </html>
```

在这段 HTML 代码中，我们分别将特殊字符及其命名实体编码和十进制编码罗列在表格中。HTML 定义的特殊字符很多，由于篇幅限制不可能一一罗列出来，上面的代码只介绍了一小部分，读者可以参阅随书源代码，里面包含全部 HTML 字符。

运行测试这个页面，打开后的效果如图 2.10 所示。从图中可以看到许多不常用的特殊字符，在 HTML 规范中均有定义，使用时可以直接编写十进制编码，也可以编写命名实体编码。

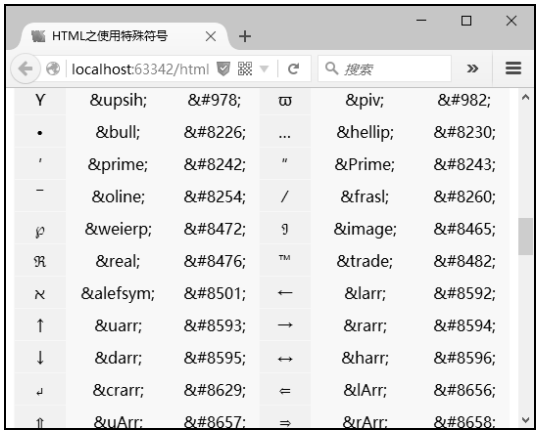

图 2.10　在 HTML 网页内使用特殊符号

2.5 创建超链接

当世界上的各种网络内容连接在一起时，我们要找某些资料就需要用到搜索引擎，而搜索引擎给出的结果都是一些链接，单击链接就会导航到具体的网站。这样的链接在网页中一般称为超链接，有时也叫超链或超级链接。

2.5.1 什么是超链接

在 HTML 网页中使用超链接可以与网络上的另一个资源（包括页面、图片、视频等）建立连接关系。理论上，HTML 网页中的绝大部分元素均可以定义超链接，譬如文字、图片、视频、表格和控件等。用户可以通过单击这些超链接跳转到新的页面，也可以跳转到当前页面的某个部分。

在 HTML 页面中，可以通过使用 a 标签定义和创建超链接，使用 a 标签的方式一般分为两种：

- 通过使用 href 属性创建指向另一个页面的链接地址。
- 通过使用 name 属性创建本页面内的书签。

定义超链接的 HTML 代码语法如下：

```
<a href="url">Link text</a>
```

其中，href 属性规定链接的目标地址，开始标签和结束标签之间的文字被作为超链接文本来显示。

下面我们分别通过示例来介绍这两种链接方式。

2.5.2 站外网页链接

所谓站外网页的超链接，就是指通过使用 href 属性创建指向另一个页面的链接地址。

下面这段代码（参见源代码 chapter02/ch02-htmltag-a-href.html 文件）给出一种创建站外网页的超链接的方法。

【示例 2-11】 站外网页链接

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTML 之站外网页链接</title>
</head>
```

```
07  <body>
08      <!-- 添加文档主体内容 -->
09      <h3>HTML 之站外网页链接</h3>
10      <!-- 添加文档主体内容 -->
11      <a href="http://www.gov.cn/" target="_self">站外网页链接：政府网站</a>
12  </body>
13  </html>
```

在这段 HTML 代码中，第 11 行代码通过 a 标签定义了一个站外网页链接（href="http://www.gov.cn/"）。运行测试这个页面，效果如图 2.11 所示。

我们尝试单击图 2.11 页面中的链接地址（站外网页链接：政府网站），由于定义了 target 属性值为“_self”（target="_self"），因此站外网页将在当前浏览器窗口中打开，其效果如图 2.12 所示。

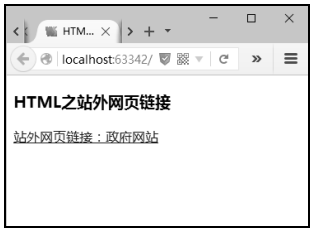

图 2.11　站外网页链接（1）

图 2.12　站外网页链接（2）

2.5.3　站内网页链接

所谓站内网页的超链接，就是指通过使用 name 属性创建本页面内的书签。

下面这段代码（参见源代码 chapter02/ch02-htmltag-a-name.html 文件）给出一种创建站内网页的超链接的方法。

【示例 2-12】　站内网页链接

```
01  <!DOCTYPE html>
02  <html lang="zh-cn">
03  <head>
04      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
05      <title>HTML 之站内网页链接</title>
06  </head>
07  <body>
```

```
08      <!-- 添加文档主体内容 -->
09      <h3>HTML 之站内网页链接</h3>
10      <!-- 添加文档主体内容 -->
11      <a href="#anchor01">HTML</a><br>
12      <a href="#anchor02">CSS</a><br>
13      <a href="#anchor03">JavaScript</a><br>
14      <!-- 添加文档主体内容 -->
15      <h4><a name="anchor01">HTML</a></h4>
16      <p>HTML5(HyperText Mark-up Language 5)是目前最流行网页设计语言。</p>
17      <br>
18      <h4><a name="anchor02">CSS</a></h4>
19      <p>CSS3(Cascading Style Sheets 3)是目前最流行样式设计语言。</p>
20      <br>
21      <h4><a name="anchor03">JavaScript</a></h4>
22      <p>JavaScript 是目前最流行前端脚本设计语言。</p>
23  </body>
24  </html>
```

在这段 HTML 代码中，定义了 3 个站内网页链接。下面我们详细介绍。

第 11 行代码通过 a 标签定义了一个站内网页链接（href="#anchor01"），其指向的是第 15 行代码定义的书签"anchor01"。

同样，第 12 行与第 13 行代码通过 a 标签定义了另外两个站内网页链接（href="#anchor02" 与 href="#anchor03"），分别指向的是第 18 行和第 21 行代码定义的书签"anchor02"与"anchor03"。

运行测试这个页面，效果如图 2.13 所示。我们尝试单击图 2.13 页面中的链接地址（CSS），效果如图 2.14 所示。从图 2.13 和图 2.14 的效果来看，当单击第 12 行代码定义的站内链接地址后，页面直接跳转到第 18 行代码定义的书签的位置。同样，当我们单击其他两个站内链接地址时，页面也会自动跳转到相应的书签位置。

图 2.13　站内网页链接（1）

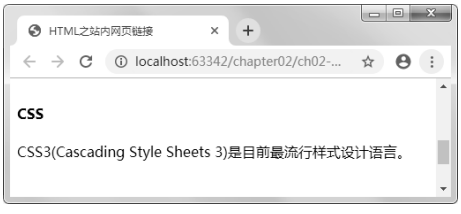

图 2.14　站内网页链接（2）

2.6 小结

本章主要介绍了 HTML 网页的基本标签，包括段落、文字、项目符号与编号、特殊符号以及超链接的内容。本章的内容比较基础，但非常重要，是 HTML 网页设计起步的必要阶段，希望能给广大读者带来帮助和启迪。

第 3 章

◂ HTML 5表单 ▸

本章介绍最新的 HTML 5 表单。在 HTML 5 表单中，为不同的标签增加了很多实用的属性，使得 HTML 网页设计开发更加快捷高效。同时，HTML 5 表单与 JavaScript 脚本语言结合得更加紧密，以前很复杂的交互功能在 HTML 5 表单中实现起来变得很简洁。

本章主要包括以下内容：

- E-Mail 和 URL 类型的输入元素
- 数值输入
- 日期选择器
- 用 datalist 来实现自动提示

3.1 各浏览器内核一览

众所周知，浏览器最重要的核心部分就是浏览器内核。其实，浏览器内核是一个通俗的称谓，比较专业的称谓是渲染引擎（Rendering Engine）。渲染引擎的功能主要是完成对网页语法的解释（包括 HTML 语法、CSS 样式表语法和 JavaScript 脚本语言语法等），并在浏览器中对页面内容进行渲染显示。因此，浏览器内核（渲染引擎）是浏览器最为核心的部分。

但是，现实中的情况很复杂，主流的浏览器由各大 IT 公司设计开发，不同的浏览器内核的实现有所不同，对网页语法的解释也就有所不同。这样就出现问题了，同一个 HTML 网页在不同的浏览器里的渲染显示效果可能不同，同一款浏览器对 HTML 网页与 HTML 5 网页的支持度也可能不同。因此，网页开发者必须测试浏览器的兼容性，也就是在不同内核的浏览器中测试同一个网页的显示效果。

下面简单介绍各浏览器的内核，包括其起源、演变历史及显著特性。

- Trident 内核：微软公司开发的一款网页渲染引擎，其代表产品是著名的 Internet Explorer 浏览器，因此又称为 IE 内核。另外，Netscape 8、傲游、世界之窗、腾讯 TT 等浏览器使用的也是该内核。
- Edge 内核：微软公司最新推出的网页渲染引擎，其伴随着 Windows 10 操作系统内置的 Microsoft Edge 浏览器出现。
- WebKit 内核：主要代表作品有 Safari 浏览器和 Google Chrome 浏览器。其优点是源码结构清晰、渲染速度极快；缺点是对网页代码的兼容性不高，一些编写不标准的网页

代码可能无法正常显示。

- Gecko 内核：主要代表作品为著名的 Mozilla Firefox 浏览器。Gecko 内核是一套开放源代码的、C++编写的网页渲染引擎。
- Presto 内核：主要代表作品是 Opera 浏览器。Presto 内核是由 Opera Software 开发的浏览器渲染引擎，仅支持 Opera 7.0 及以上版本使用。该内核加入了动态功能，页面可以随着 DOM 及 JavaScript 脚本语法的事件而重新进行渲染显示。

另外，还有一些知名度及应用度均不高的浏览器引擎，譬如 Tasman、KHTML、WebCore 等，感兴趣的读者可以了解一下。

3.2 E-Mail 类型的 input 标签

HTML 5 表单为 input 标签设计了几个全新的 type 类型，本节介绍 E-Mail 类型。设置成 E-Mail 类型的 input 标签可以在表单提交时自动验证其中的内容是否为合法的 E-Mail 类型地址，这样就降低了传统 JS 脚本语言验证时的复杂度。

下面我们看一段在 HTML 5 表单中使用 E-Mail 类型的代码，这段代码（参见源代码 chapter03/ch02-HTML 5form-input-email.html 文件）给出一种使用 E-Mail 类型的 input 标签的方法。

【示例 3-1】　E-Mail 类型的 input 标签的使用

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/css">
        label {
            margin: 4px;
            width: 96px;
            text-align: justify;
        }
        input {
            margin: 4px;
        }
    </script>
    <title>HTML 5表单之 E-mail 类型 input 标签</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 5表单之 E-mail 类型 input 标签</h3>
    <!-- 添加文档主体内容 -->
    <form action="#" method="get" autocomplete="on">
        <label for="name">用户名:</label><br>
```

```
23          <input type="text" id="name" /><br/>
24          <label for="pwd">密  码:</label><br>
25          <input type="password" id="pwd" /><br/>
26          <label for="email">E-mail:</label><br>
27          <input type="email" id="email" autocomplete="off" /><br/><br/>
28          <input type="submit" />
29      </form>
30      <!-- 添加文档主体内容 -->
31  </body>
32  </html>
```

在这段 HTML 代码中，定义了一个 form 表单及一些标签。下面我们详细介绍。

第 21～29 行代码通过 form 标签定义了一个 HTML 5 表单，其中 action 属性定义为提交到当前页面，method 属性定义为 get 方式，并定义了 autocomplete 自动完成属性。

第 22～23 行代码定义了一个 type="text"类型的 input 标签，用于输入用户名称。

第 24～25 行代码定义了一个 type="password"类型的 input 标签，用于输入登录密码。

第 26～27 行代码定义了一个 type="email"类型的 input 标签，用于输入电子邮箱地址。其中，type="email"类型的 input 标签就是 HTML 5 表单新增加的特性，提交时可以自动完成对电子邮件地址类型的验证。第 27 行代码定义 E-Mail 类型的 input 标签时还增加了 autocomplete="off"属性的定义，关闭了 input 标签的自动完成功能。

第 28 行代码定义了一个 type="submit"类型的 input 标签，用于提交 HTML 5 表单。

运行测试这个页面，效果如图 3.1 所示。从图中可以看到，当用户在 E-Mail 类型的 input 标签中输入不合法的电子邮件地址时，该 input 标签的边框会自动加粗变成红色，提示用户输入有误。

下面单击“提交查询”按钮运行测试这个页面，效果如图 3.2 所示。从图中可以看到，如果用户不理会提示错误，强行单击“提交查询”按钮，页面就会弹出一个 Tooltip 提示窗口，告诉用户“请输入电子邮件地址。”，这就是 HTML 5 表单的新特性，内部实现了以前使用脚本语言才能完成的验证功能，真的很强大。

图 3.1　E-Mail 类型的 input 标签的使用（1）

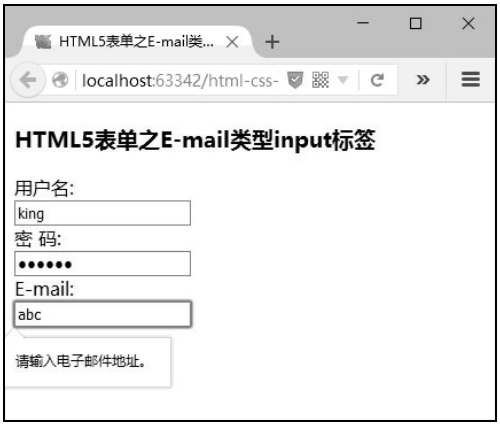

图 3.2　E-Mail 类型的 input 标签的使用（2）

在表单提交成功后，按 F5 键刷新这个页面，效果如图 3.3 所示。从图中可以看到，重新刷新页面后，Text 类型的 input 标签的内容保留下来了，而 Password 类型和 E-Mail 类型的 input 标签的内容被清空了，这就是前面提到的 HTML 5 表单的自动完成新特性。

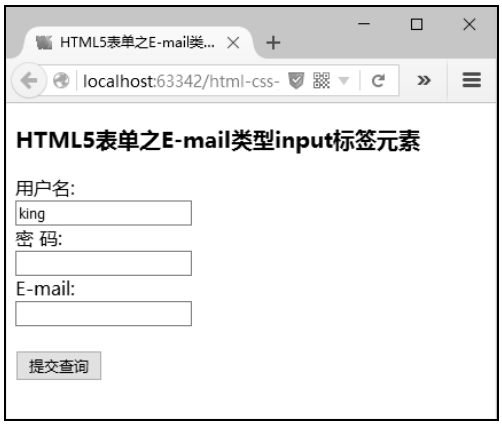

图 3.3　E-Mail 类型的 input 标签的使用（3）

3.3 URL 类型的 input 标签

本节介绍 HTML 5 表单为 input 标签设计的另一个全新的 type 类型——URL 类型。设置成 URL 类型的 input 标签可以在表单提交时自动验证其中的内容是否为合法的 URL 类型地址。

下面我们看一段在 HTML 5 表单中使用 URL 类型的代码，这段代码（参见源代码 chapter03/ch02-HTML 5form-input-url.html 文件）给出一种使用 URL 类型的 input 标签的方法。

【示例 3-2】　URL 类型的 input 标签的使用

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/css">
        label {
            margin: 4px;
            width: 96px;
            text-align: justify;
        }
        input {
            margin: 4px;
        }
    </script>
```

```
15      <title>HTML 5表单之 URL 类型 input 标签</title>
16  </head>
17  <body>
18      <!-- 添加文档主体内容 -->
19      <h3>HTML 5表单之 URL 类型 input 标签</h3>
20      <!-- 添加文档主体内容 -->
21      <form action="#" method="get">
22          <label for="url">URL:</label><br>
23          <input type="url" id="url" style="width: 256px" autocomplete="off"
/><br/><br/>
24          <input type="submit" value="提交网址" />
25      </form>
26  </body>
27  </html>
```

在这段 HTML 代码中，定义了一个 form 表单及一个 input 标签。下面我们详细介绍。

第 21～25 行代码通过 form 标签定义了一个 HTML 5 表单，其中 action 属性定义为提交到当前页面，method 属性定义为 get 方式。

第 22～23 行代码定义了一个 type="url"类型的 input 标签，用于输入 URL 类型地址。其中，type="url"类型的 input 标签就是 HTML 5 表单新增加的特性，提交时可以自动完成对 URL 地址的验证。第 23 行代码定义 URL 类型的 input 标签时还增加了 autocomplete="off"属性的定义，关闭了 input 标签的自动完成功能。

第 24 行代码定义了一个 type="submit"类型的 input 标签，用于提交 HTML 5 表单。

运行测试这个页面，效果如图 3.4 所示。从图中可以看到，当用户在 URL 类型的 input 标签中输入不合法的 URL 地址时，该 input 标签的边框会自动加粗变成红色，提示用户输入的 URL 地址有误。

下面单击“提交网址”按钮运行测试这个页面，效果如图 3.5 所示。从图中可以看到，如果用户不理会提示错误，强行单击“提交网址”按钮，页面就会弹出一个 Tooltip 提示窗口，告诉用户“请输入一个 URL。”，这就是 HTML 5 表单的新特性。

图 3.4　URL 类型的 input 标签的使用（1）

图 3.5　URL 类型的 input 标签的使用（2）

3.4 数值类型的 input 标签

本节介绍 HTML 5 表单为 input 标签设计的下一个全新的 type 类型——数值（Number）类型。设置成 Number 类型的 input 标签可以在表单提交时自动验证其中的内容是否为合法的数值类型。

下面我们看一段在 HTML 5 表单中使用数值类型的代码，这段代码（参见源代码 chapter03/ch02-HTML 5form-input-number.html 文件）给出一种使用 Number 类型的 input 标签的方法。

【示例 3-3】　数值类型的 input 标签的使用

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/css">
        label {
            margin: 4px;
            width: 96px;
            text-align: justify;
        }
        input {
            margin: 4px;
        }
    </script>
    <title>HTML 5表单之数值类型 input 标签</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 5表单之数值类型 input 标签</h3>
    <!-- 添加文档主体内容 -->
    <form action="#" method="get" autocomplete="on">
        <label for="month">月份:</label><br>
        <input type="number" id="month" min="1" max="12" step="1" value="1"
/><br/>
        <label for="day">日期:</label><br>
        <input type="number" id="day" min="1" max="31" step="1" value="1"
/><br/>
        <label for="year">年份:</label><br>
    <input type="number" id="year" min="2016" max="2024" step="4"
```

```
value="2020" />
   28     </form>
   29  </body>
   30  </html>
```

在这段 HTML 代码中，定义了一个 form 表单及 3 个 input 标签。下面我们详细介绍。

第 21～28 行代码通过 form 标签定义了一个 HTML 5 表单，其中 action 属性定义为提交到当前页面，method 属性定义为"get"方式。

第 22～23 行代码定义了第一个 type="number"类型的 input 标签，用于输入月份。第 23 行代码定义 Number 类型的 input 标签时还增加了 min、max、step 和 value 属性的定义，其中 min 属性用于定义最小数值（月份最小值为 1），max 属性用于定义最大数值（月份最大值为 12），step 属性用于定义步长（步长值为 1），value 属性用于定义初始值。

第 24～25 行代码定义了第二个 type="number"类型的 input 标签，用于输入日期。第 25 行代码定义 Number 类型的 input 标签时，min 属性用于定义最小数值（日期最小值为 1），max 属性用于定义最大数值（日期最大值为 31），step 属性用于定义步长（步长值为 1），value 属性用于定义初始值。

第 26～27 行代码定义了第三个 type="number"类型的 input 标签，用于输入年份。第 27 行代码定义 Number 类型的 input 标签时，min 属性用于定义最小数值（年份最小值为 2016），max 属性用于定义最大数值（年份最大值为 2024），step 属性用于定义步长（步长值为 4，表示只支持闰年），value 属性用于定义初始值（为 2020 年）。

运行测试这个页面，效果如图 3.6 所示。

理论上，使用 input 标签右侧的上下箭头调整数值是不会超出我们定义的数值范围的，步长也会按照设定值调整。但是，如果我们手动输入了不合法的数值，HTML 5 表单就会提醒出错了，页面效果如图 3.7 所示。从图中可以看到，当用户在数值类型的 input 标签中输入超出范围的不合法的数值时，该 input 标签的边框会自动加粗变成红色，提示用户输入的数值有误。

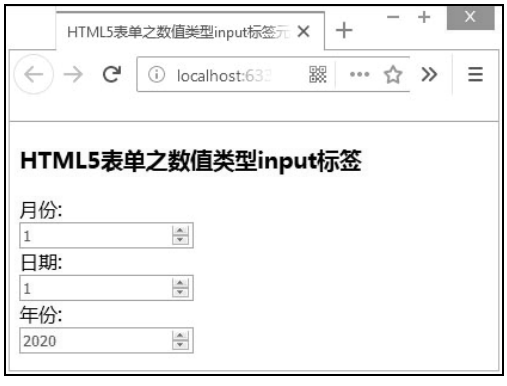

图 3.6　数值类型的 input 标签的使用（1）

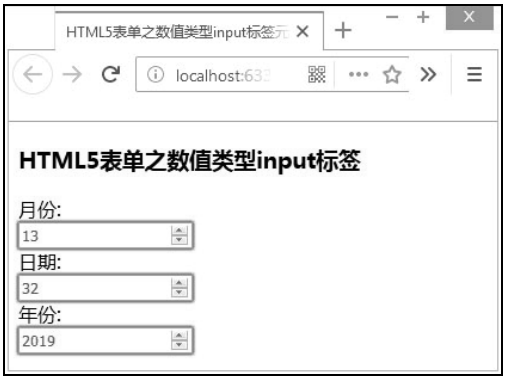

图 3.7　数值类型的 input 标签的使用（2）

3.5 使用日期选择器

本节介绍 HTML 5 表单为 input 标签设计的全新的日期选择器（DatePicker）类型。HTML 5 表单提供了大约 5 种形式的日期选择器，包括按照月份、周、时间等方式。

下面我们看一段在 HTML 5 表单中使用日期选择器的代码，这段代码（参见源代码 chapter03/ch02-HTML 5form-input-datepicker.html 文件）给出一种使用 DatePicker 类型的 input 标签的方法。

【示例 3-4】　日期选择器类型的 input 标签的使用

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/css">
        label {
            margin: 4px;
            width: 96px;
            text-align: justify;
        }
        input {
            margin: 4px;
        }
    </script>
    <title>HTML 5表单之日期选择器</title>
</head>
<body>
    <!-- 添加文档主体内容 -->
    <h3>HTML 5表单之日期选择器</h3>
    <!-- 添加文档主体内容 -->
    <form action="#" method="get" autocomplete="on">
        <label for="id-date">Date:</label><br>
        <input type="date" id="id-date" name="user_date" /><br/>
        <label for="id-month">Month:</label><br>
        <input type="month" id="id-month" name="user_date" /><br/>
        <label for="id-week">Week:</label><br>
        <input type="month" id="id-week" name="user_date" /><br/>
        <label for="id-time">Time:</label><br>
        <input type="month" id="id-time" name="user_date" /><br/>
        <label for="id-datetime">DateTime:</label><br>
```

```
31          <input type="month" id="id-datetime" name="user_date" /><br/>
32      </form>
33  </body>
34  </html>
```

在这段 HTML 代码中，定义了一个 form 表单及 5 个 input 标签：第 21～32 行代码通过 form 标签定义了一个 HTML 5 表单，其中 action 属性定义为提交到当前页面，method 属性定义为 get 方式；第 22～23 行代码定义了第一个 type="date"类型的日期选择器。

运行测试这个页面，单击 input 标签右侧的下拉箭头，Date 类型的日期选择器的效果如图 3.8 所示。

第 24～25 行代码定义了第二个 type="month"类型的日期选择器。运行测试这个页面，单击 input 标签右侧的下拉箭头，Month 类型的日期选择器的效果如图 3.9 所示。

图 3.8　Date 类型的日期选择器

图 3.9　Month 类型的日期选择器

第 26～27 行代码定义了第三个 type="week"类型的日期选择器。运行测试这个页面，单击 input 标签右侧的下拉箭头，Week 类型的日期选择器的效果如图 3.10 所示。

第 28～29 行代码定义了第四个 type="time"类型的日期选择器。运行测试这个页面，单击 input 标签右侧的下拉箭头，Time 类型的日期选择器的效果如图 3.11 所示。

图 3.10　Week 类型的日期选择器

图 3.11　Time 类型的日期选择器

第 30～31 行代码定义了第五个 type="datetime"类型的日期选择器。运行测试这个页面，单击 input 标签右侧的下拉箭头，DateTime 类型的日期选择器的效果如图 3.12 所示。

图 3.12　DateTime 类型的日期选择器

3.6 用 datalist 标签实现自动提示功能

本节介绍使用 HTML 5 表单的 datalist 标签实现自动提示功能。在 HTML 5 规范标准中，datalist 标签用于定义选项列表，并与 input 标签配合起来使用。在实际应用中，DataList 中的选项不会被显示出来，其仅提供合法的输入值列表。

下面我们看一段在 HTML 5 表单中使用 datalist 标签的代码，这段代码（参见源代码 chapter03/ch02-HTML 5form-input-datalist.html 文件）给出一种使用 datalist 标签的方法。

【示例 3-5】　datalist 标签的使用

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script type="text/css">
        label {
            margin: 4px;
            width: 96px;
            text-align: justify;
        }
        input {
            margin: 4px;
        }
    </script>
    <title>HTML 5表单之 datalist 标签</title>
```

```
16 </head>
17 <body>
18     <!-- 添加文档主体内容 -->
19     <h3>HTML 5表单之使用 datalist 标签实现自动提示</h3>
20     <!-- 添加文档主体内容 -->
21     <form action="#" method="get" autocomplete="on">
22         <label for="id-datalist">DataList:</label><br>
23         <input id="id-datalist" list="web" />
24         <datalist id="web">
25             <option value="HTML 5">
26             <option value="CSS3">
27             <option value="JavaScript">
28             <option value="jQuery">
29             <option value="Node">
30         </datalist>
31     </form>
32 </body>
33 </html>
```

在这段 HTML 代码中，定义了一个 forms 标签、一个 input 标签和一个 datalist 标签。下面我们详细介绍。

第 21～31 行代码通过 form 标签定义了一个 HTML 5 表单，其中 action 属性定义为提交到当前页面，method 属性定义为 get 方式。

第 22～23 行代码定义了一个 type="text"类型的 input 标签，并通过 list 属性（list="web"）绑定其后的 datalist 标签选项。

第 24～30 行代码定义了一个 datalist 标签，并通过 id 值（id="web"）绑定在上面的 input 标签上；第 24～29 行代码通过 option 标签定义了一组可选的选项值。

运行测试这个页面，单击 input 标签右侧的下拉按钮，效果如图 3.13 所示。如果我们在 input 标签中输入第一个字符“H”，就会自动提示首字母为“H”的选项值，效果如图 3.14 所示。

图 3.13　datalist 标签的使用（1）

图 3.14　datalist 标签的使用（2）

如果我们在 input 标签中输入第一个字符“j”，就会自动提示首字母为“j”的选项值，效果如图 3.15 所示。从图中可以看到，首字母不区分大小写，输入字母“j”后，JavaScript 和 jQuery 选项值都出现在自动提示框中了。

图 3.15　datalist 标签的使用（3）

3.7 各个浏览器对 HTML 5 表单新类型的支持

本章前面几节主要介绍了 HTML 5 表单提供的一些新特性。细心的读者一定发现了，在介绍 E-Mail 类型和 URL 类型的 input 标签时，测试页面使用的是 FireFox 浏览器，而在介绍日期选择器时，使用的是 Opera 浏览器。这是因为各个浏览器对 HTML 5 表单新特性的支持度是不一样的，有些支持得很好，有些支持得不足。该问题是设计人员常常提到的浏览器兼容性问题的一种，是一个考验开发水平的问题。

表 3-1 提供各个主流浏览器对 HTML 5 表单的支持情况（该表格部分结果引用了开源代码组织提供的研究结果），表中的内容仅供读者参考。

表 3-1　各个主流浏览器对 HTML 5 表单的支持情况

浏览器 / 类型	IE	Chrome	Firefox	Opera	Safari
E-Mail	不支持	支持	支持（4.0 版）	支持	不支持
URL	不支持	支持	支持（4.0 版）	支持	不支持
Number	不支持	支持	不支持	支持	不支持
DatePicker	不支持	支持	不支持	支持	不支持
DataList	不支持	支持	不支持	支持	不支持

3.8 小结

本章主要介绍了 HTML 5 表单的新特性，包括 E-Mail 类型、URL 类型、Number 类型和 DatePicker 类型的 input 标签的使用方法，另外还介绍了使用新的 datalist 标签实现自动提示功能的方法。希望 HTML 5 的新特性能给广大读者带来全新的体验。