第2章

在 HTML 中使用 JavaScript

本章内容

- 使用<script>元素
- 嵌入脚本与外部脚本
- 文档模式对 JavaScript 的影响
- 考虑禁用 JavaScript 的场景

只要一提到把 JavaScript 放到网页中，就不得不涉及 Web 的核心语言——HTML。在当初开发 JavaScript 的时候，Netscape 要解决的一个重要问题就是如何做到让 JavaScript 既能与 HTML 页面共存，又不影响那些页面在其他浏览器中的呈现效果。经过尝试、纠错和争论，最终的决定就是为 Web 增加统一的脚本支持。而 Web 诞生早期的很多做法也都保留了下来，并被正式纳入 HTML 规范当中。

2.1 <script>元素

向 HTML 页面中插入 JavaScript 的主要方法，就是使用<script>元素。这个元素由 Netscape 创造并在 Netscape Navigator 2 中首先实现。后来，这个元素被加入到正式的 HTML 规范中。HTML 4.01 为<script>定义了下列 6 个属性。

- async：可选。表示应该立即下载脚本，但不应妨碍页面中的其他操作，比如下载其他资源或等待加载其他脚本。只对外部脚本文件有效。
- charset：可选。表示通过 src 属性指定的代码的字符集。由于大多数浏览器会忽略它的值，因此这个属性很少有人用。
- defer：可选。表示脚本可以延迟到文档完全被解析和显示之后再执行。只对外部脚本文件有效。IE7 及更早版本对嵌入脚本也支持这个属性。
- language：已废弃。原来用于表示编写代码使用的脚本语言（如 JavaScript、JavaScript1.2 或 VBScript）。大多数浏览器会忽略这个属性，因此也没有必要再用了。
- src：可选。表示包含要执行代码的外部文件。
- type：可选。可以看成是 language 的替代属性；表示编写代码使用的脚本语言的内容类型（也称为 MIME 类型）。虽然 text/javascript 和 text/ecmascript 都已经不被推荐使用，但人们一直以来使用的都还是 text/javascript。实际上，服务器在传送 JavaScript 文件时使用的 MIME 类型通常是 application/x-javascript，但在 type 中设置这个值却可能导致脚本被忽略。另外，在非 IE 浏览器中还可以使用以下值：application/javascript 和 application/

ecmascript。考虑到约定俗成和最大限度的浏览器兼容性，目前 type 属性的值依旧还是 text/javascript。不过，这个属性并不是必需的，如果没有指定这个属性，则其默认值仍为 text/javascript。

使用<script>元素的方式有两种：直接在页面中嵌入 JavaScript 代码和包含外部 JavaScript 文件。

在使用<script>元素嵌入 JavaScript 代码时，只须为<script>指定 type 属性。然后，像下面这样把 JavaScript 代码直接放在元素内部即可：

```
<script type="text/javascript">
    function sayHi(){
        alert("Hi!");
    }
</script>
```

包含在<script>元素内部的 JavaScript 代码将被从上至下依次解释。就拿前面这个例子来说，解释器会解释到一个函数的定义，然后将该定义保存在自己的环境当中。在解释器对<script>元素内部的所有代码求值完毕以前，页面中的其余内容都不会被浏览器加载或显示。

在使用<script>嵌入 JavaScript 代码时，记住不要在代码中的任何地方出现"</script>"字符串。例如，浏览器在加载下面所示的代码时就会产生一个错误：

```
<script type="text/javascript">
    function sayScript(){
        alert("</script>");
    }
</script>
```

因为按照解析嵌入式代码的规则，当浏览器遇到字符串"</script>"时，就会认为那是结束的</script>标签。而通过把这个字符串分隔为两部分可以解决这个问题，例如：

```
<script type="text/javascript">
    function sayScript(){
        alert("<\/script>");
    }
</script>
```

像这样分成两部分来写就不会造成浏览器的误解，因而也就不会导致错误了。

如果要通过<script>元素来包含外部 JavaScript 文件，那么 src 属性就是必需的。这个属性的值是一个指向外部 JavaScript 文件的链接，例如：

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

在这个例子中，外部文件 example.js 将被加载到当前页面中。外部文件只须包含通常要放在开始的<script>和结束的</script>之间的那些 JavaScript 代码即可。与解析嵌入式 JavaScript 代码一样，在解析外部 JavaScript 文件（包括下载该文件）时，页面的处理也会暂时停止。如果是在 XHTML 文档中，也可以省略前面示例代码中结束的</script>标签，例如：

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

但是，不能在 HTML 文档使用这种语法。原因是这种语法不符合 HTML 规范，而且也得不到某些浏览器（尤其是 IE）的正确解析。

按照惯例，外部 JavaScript 文件带有.js 扩展名。但这个扩展名不是必需的，因为浏览器不会检查包含 JavaScript 的文件的扩展名。这样一来，使用 JSP、PHP 或其他服务器端语言动态生成 JavaScript 代码也就成为了可能。但是，服务器通常还是需要看扩展名决定为响应应用哪种 MIME 类型。如果不使用.js 扩展名，请确保服务器能返回正确的 MIME 类型。

需要注意的是，带有 src 属性的<script>元素不应该在其<script>和</script>标签之间再包含额外的 JavaScript 代码。如果包含了嵌入的代码，则只会下载并执行外部脚本文件，嵌入的代码会被忽略。

另外，通过<script>元素的 src 属性还可以包含来自外部域的 JavaScript 文件。这一点既使<script>元素倍显强大，又让它备受争议。在这一点上，<script>与<img>元素非常相似，即它的 src 属性可以是指向当前 HTML 页面所在域之外的某个域中的 URL，例如：

```
<script type="text/javascript" src="http://www.somewhere.com/afile.js"></script>
```

这样，位于外部域中的代码也会被加载和解析，就像这些代码位于加载它们的页面中一样。利用这一点就可以在必要时通过不同的域来提供 JavaScript 文件。不过，在访问自己不能控制的服务器上的 JavaScript 文件时则要多加小心。如果不幸遇到了怀有恶意的程序员，那他们随时都可能替换该文件中的代码。因此，如果想包含来自不同域的代码，则要么你是那个域的所有者，要么那个域的所有者值得信赖。

无论如何包含代码，只要不存在 defer 和 async 属性，浏览器都会按照<script>元素在页面中出现的先后顺序对它们依次进行解析。换句话说，在第一个<script>元素包含的代码解析完成后，第二个<script>包含的代码才会被解析，然后才是第三个、第四个……

2.1.1　标签的位置

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

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

这种做法的目的就是把所有外部文件（包括 CSS 文件和 JavaScript 文件）的引用都放在相同的地方。可是，在文档的<head>元素中包含所有 JavaScript 文件，意味着必须等到全部 JavaScript 代码都被下载、解析和执行完成以后，才能开始呈现页面的内容（浏览器在遇到<body>标签时才开始呈现内容）。对于那些需要很多 JavaScript 代码的页面来说，这无疑会导致浏览器在呈现页面时出现明显的延迟，而延迟期间的浏览器窗口中将是一片空白。为了避免这个问题，现代 Web 应用程序一般都把全部 JavaScript 引用放在<body>元素中页面的内容后面，如下例所示：

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

这样，在解析包含的 JavaScript 代码之前，页面的内容将完全呈现在浏览器中。而用户也会因为浏览器窗口显示空白页面的时间缩短而感到打开页面的速度加快了。

2.1.2 延迟脚本

HTML 4.01 为<script>标签定义了 defer 属性。这个属性的用途是表明脚本在执行时不会影响页面的构造。也就是说，脚本会被延迟到整个页面都解析完毕后再运行。因此，在<script>元素中设置 defer 属性，相当于告诉浏览器立即下载，但延迟执行。

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

在这个例子中，虽然我们把<script>元素放在了文档的<head>元素中，但其中包含的脚本将延迟到浏览器遇到</html>标签后再执行。HTML5 规范要求脚本按照它们出现的先后顺序执行，因此第一个延迟脚本会先于第二个延迟脚本执行，而这两个脚本会先于 DOMContentLoaded 事件（详见第 13 章）执行。在现实当中，延迟脚本并不一定会按照顺序执行，也不一定会在 DOMContentLoaded 事件触发前执行，因此最好只包含一个延迟脚本。

前面提到过，defer 属性只适用于外部脚本文件。这一点在 HTML5 中已经明确规定，因此支持 HTML5 的实现会忽略给嵌入脚本设置的 defer 属性。IE4～IE7 还支持对嵌入脚本的 defer 属性，但 IE8 及之后版本则完全支持 HTML5 规定的行为。

IE4、Firefox 3.5、Safari 5 和 Chrome 是最早支持 defer 属性的浏览器。其他浏览器会忽略这个属性，像平常一样处理脚本。为此，把延迟脚本放在页面底部仍然是最佳选择。

在 XHTML 文档中，要把 defer 属性设置为 defer="defer"。

2.1.3 异步脚本

HTML5 为<script>元素定义了 async 属性。这个属性与 defer 属性类似，都用于改变处理脚本

的行为。同样与 defer 类似，async 只适用于外部脚本文件，并告诉浏览器立即下载文件。但与 defer 不同的是，标记为 async 的脚本并不保证按照指定它们的先后顺序执行。例如：

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

在以上代码中，第二个脚本文件可能会在第一个脚本文件之前执行。因此，确保两者之间互不依赖非常重要。指定 async 属性的目的是不让页面等待两个脚本下载和执行，从而异步加载页面其他内容。为此，建议异步脚本不要在加载期间修改 DOM。

异步脚本一定会在页面的 load 事件前执行，但可能会在 DOMContentLoaded 事件触发之前或之后执行。支持异步脚本的浏览器有 Firefox 3.6、Safari 5 和 Chrome。

在 XHTML 文档中，要把 async 属性设置为 async="async"。

2.1.4　在 XHTML 中的用法[①]

可扩展超文本标记语言，即 XHTML（Extensible HyperText Markup Language），是将 HTML 作为 XML 的应用而重新定义的一个标准。编写 XHTML 代码的规则要比编写 HTML 严格得多，而且直接影响能否在嵌入 JavaScript 代码时使用<script/>标签。以下面的代码块为例，虽然它们在 HTML 中是有效的，但在 XHTML 中则是无效的。

```
<script type="text/javascript">
    function compare(a, b) {
        if (a < b) {
            alert("A is less than B");
        } else if (a > b) {
            alert("A is greater than B");
        } else {
            alert("A is equal to B");
        }
    }
</script>
```

在 HTML 中，有特殊的规则用以确定<script>元素中的哪些内容可以被解析，但这些特殊的规则在 XHTML 中不适用。这里比较语句 a < b 中的小于号（<）在 XHTML 中将被当作开始一个新标签来解析。但是作为标签来讲，小于号后面不能跟空格，因此就会导致语法错误。

① HTML5 正快速地被前端开发人员采用，建议读者在学习和开发中遵循 HTML5 标准，本节内容可以跳过。

避免在 XHTML 中出现类似语法错误的方法有两个。一是用相应的 HTML 实体（<）替换代码中所有的小于号（<），替换后的代码类似如下所示：

```
<script type="text/javascript">
    function compare(a, b) {
        if (a &lt; b) {
            alert("A is less than B");
        } else if (a > b) {
            alert("A is greater than B");
        } else {
            alert("A is equal to B");
        }
    }
</script>
```

虽然这样可以让代码在 XHTML 中正常运行，但却导致代码不好理解了。为此，我们可以考虑采用另一个方法。

保证让相同代码在 XHTML 中正常运行的第二个方法，就是用一个 CData 片段来包含 JavaScript 代码。在 XHTML（XML）中，CData 片段是文档中的一个特殊区域，这个区域中可以包含不需要解析的任意格式的文本内容。因此，在 CData 片段中就可以使用任意字符——小于号当然也没有问题，而且不会导致语法错误。引入 CData 片段后的 JavaScript 代码块如下所示：

```
<script type="text/javascript"><![CDATA[
    function compare(a, b) {
        if (a < b) {
            alert("A is less than B");
        } else if (a > b) {
            alert("A is greater than B");
        } else {
            alert("A is equal to B");
        }
    }
]]></script>
```

在兼容 XHTML 的浏览器中，这个方法可以解决问题。但实际上，还有不少浏览器不兼容 XHTML，因而不支持 CData 片段。怎么办呢？再使用 JavaScript 注释将 CData 标记注释掉就可以了：

```
<script type="text/javascript">
//<![CDATA[
    function compare(a, b) {
        if (a < b) {
            alert("A is less than B");
        } else if (a > b) {
            alert("A is greater than B");
        } else {
            alert("A is equal to B");
        }
    }
//]]>
</script>
```

这种格式在所有现代浏览器中都可以正常使用。虽然有几分 hack 的味道，但它能通过 XHTML 验证，而且对 XHTML 之前的浏览器也会平稳退化。

在将页面的 MIME 类型指定为"application/xhtml+xml"的情况下会触发 XHTML 模式。并不是所有浏览器都支持以这种方式提供 XHTML 文档。

2.1.5　不推荐使用的语法

在最早引入<script>元素的时候，该元素与传统 HTML 的解析规则是有冲突的。由于要对这个元素应用特殊的解析规则，因此在那些不支持 JavaScript 的浏览器（最典型的是 Mosaic）中就会导致问题。具体来说，不支持 JavaScript 的浏览器会把<script>元素的内容直接输出到页面中，因而会破坏页面的布局和外观。

Netscape 与 Mosaic 协商并提出了一个解决方案，让不支持<script>元素的浏览器能够隐藏嵌入的 JavaScript 代码。这个方案就是把 JavaScript 代码包含在一个 HTML 注释中，像下面这样：

```
<script><!--
    function sayHi(){
        alert("Hi!");
    }
//--></script>
```

给脚本加上 HTML 注释后，Mosaic 等浏览器就会忽略<script>标签中的内容；而那些支持 JavaScript 的浏览器在遇到这种情况时，则必须进一步确认其中是否包含需要解析的 JavaScript 代码。

虽然这种注释 JavaScript 代码的格式得到了所有浏览器的认可，也能被正确解释，但由于所有浏览器都已经支持 JavaScript，因此也就没有必要再使用这种格式了。在 XHTML 模式下，因为脚本包含在 XML 注释中，所以脚本会被忽略。

2.2　嵌入代码与外部文件

在 HTML 中嵌入 JavaScript 代码虽然没有问题，但一般认为最好的做法还是尽可能使用外部文件来包含 JavaScript 代码。不过，并不存在必须使用外部文件的硬性规定，但支持使用外部文件的人多会强调如下优点。

- **可维护性**：遍及不同 HTML 页面的 JavaScript 会造成维护问题。但把所有 JavaScript 文件都放在一个文件夹中，维护起来就轻松多了。而且开发人员因此也能够在不触及 HTML 标记的情况下，集中精力编辑 JavaScript 代码。
- **可缓存**：浏览器能够根据具体的设置缓存链接的所有外部 JavaScript 文件。也就是说，如果有两个页面都使用同一个文件，那么这个文件只需下载一次。因此，最终结果就是能够加快页面加载的速度。
- **适应未来**：通过外部文件来包含 JavaScript 无须使用前面提到 XHTML 或注释 hack。HTML 和 XHTML 包含外部文件的语法是相同的。

2.3　文档模式

IE5.5 引入了文档模式的概念，而这个概念是通过使用文档类型（doctype）切换实现的。最初的两

种文档模式是：混杂模式（quirks mode）[①]和标准模式（standards mode）。混杂模式会让 IE 的行为与（包含非标准特性的）IE5 相同，而标准模式则让 IE 的行为更接近标准行为。虽然这两种模式主要影响 CSS 内容的呈现，但在某些情况下也会影响到 JavaScript 的解释执行。本书将在必要时再讨论这些因文档模式而影响 JavaScript 执行的情况。

在 IE 引入文档模式的概念后，其他浏览器也纷纷效仿。在此之后，IE 又提出一种所谓的准标准模式（almost standards mode）。这种模式下的浏览器特性有很多都是符合标准的，但也不尽然。不标准的地方主要体现在处理图片间隙的时候（在表格中使用图片时问题最明显）。

如果在文档开始处没有发现文档类型声明，则所有浏览器都会默认开启混杂模式。但采用混杂模式不是什么值得推荐的做法，因为不同浏览器在这种模式下的行为差异非常大，如果不使用某些 hack 技术，跨浏览器的行为根本就没有一致性可言。

对于标准模式，可以通过使用下面任何一种文档类型来开启：

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

<!-- XHTML 1.0 严格型 -->
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- HTML 5 -->
<!DOCTYPE html>
```

而对于准标准模式，则可以通过使用过渡型（transitional）或框架集型（frameset）文档类型来触发，如下所示：

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

<!-- HTML 4.01 框架集型 -->
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">

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

<!-- XHTML 1.0 框架集型 -->
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

准标准模式与标准模式非常接近，它们的差异几乎可以忽略不计。因此，当有人提到“标准模式”时，有可能是指这两种模式中的任何一种。而且，检测文档模式（本书后面将会讨论）时也不会发现什么不同。本书后面提到标准模式时，指的是除混杂模式之外的其他模式。

① 这里 quirks mode 的译法源自 Firefox 3.5.5 中文版。

2.4　<noscript>元素

早期浏览器都面临一个特殊的问题，即当浏览器不支持 JavaScript 时如何让页面平稳地退化。对这个问题的最终解决方案就是创造一个<noscript>元素，用以在不支持 JavaScript 的浏览器中显示替代的内容。这个元素可以包含能够出现在文档<body>中的任何 HTML 元素——<script>元素除外。包含在<noscript>元素中的内容只有在下列情况下才会显示出来：

- 浏览器不支持脚本；
- 浏览器支持脚本，但脚本被禁用。

符合上述任何一个条件，浏览器都会显示<noscript>中的内容。而在除此之外的其他情况下，浏览器不会呈现<noscript>中的内容。

请看下面这个简单的例子：

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

这个页面会在脚本无效的情况下向用户显示一条消息。而在启用了脚本的浏览器中，用户永远也不会看到它——尽管它是页面的一部分。

2.5　小结

把 JavaScript 插入到 HTML 页面中要使用<script>元素。使用这个元素可以把 JavaScript 嵌入到 HTML 页面中，让脚本与标记混合在一起；也可以包含外部的 JavaScript 文件。而我们需要注意的地方有：

- 在包含外部 JavaScript 文件时，必须将 src 属性设置为指向相应文件的 URL。而这个文件既可以是与包含它的页面位于同一个服务器上的文件，也可以是其他任何域中的文件。
- 所有<script>元素都会按照它们在页面中出现的先后顺序依次被解析。在不使用 defer 和 async 属性的情况下，只有在解析完前面<script>元素中的代码之后，才会开始解析后面<script>元素中的代码。
- 由于浏览器会先解析完不使用 defer 属性的<script>元素中的代码，然后再解析后面的内容，所以一般应该把<script>元素放在页面最后，即主要内容后面，</body>标签前面。
- 使用 defer 属性可以让脚本在文档完全呈现之后再执行。延迟脚本总是按照指定它们的顺序执行。
- 使用 async 属性可以表示当前脚本不必等待其他脚本，也不必阻塞文档呈现。不能保证异步脚本按照它们在页面中出现的顺序执行。

另外，使用<noscript>元素可以指定在不支持脚本的浏览器中显示的替代内容。但在启用了脚本的情况下，浏览器不会显示<noscript>元素中的任何内容。