17.1.3 延迟执行脚本

为了避免脚本在执行时影响页面的构造，HTML 为<script>标签定义了 defer 属性。defer 属性能够迫使脚本被延迟到整个页面都解析完毕后再运行。因此，在<script>标签中设置 defer 属性，相当于告诉浏览器虽然可以立即下载 JavaScript 代码，但延迟执行。

【示例】在下面示例中，虽然把<script>标签放在文档的<head>标签中，但其中包含的脚本将延迟到浏览器遇到</html>标签后再执行。

```
<!doctype html>
<html>
<head>
<script type="text/javascript" defer src="test1.js"></script>
<script type="text/javascript" defer src="test2.js"></script>
</head>
<body>
<!-- 网页内容 -->
</body>
</html>
```

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 事件执行。在实际应用中，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。

【提示】

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。IE4~1E7 还支持对嵌入脚本的 defer 属性，但 1E8 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。因此，把延迟脚本放在页面底部仍然是最佳选择。

【注意】

在 XHTML 类型的文档中，defer 属性应该定义为 defer="defer"。

17.1.4 异步响应脚本

HTML5 为<script>标签定义了 async 属性。这个属性与 defer 属性类似，都用于改变外部脚本的行为。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。

【示例】在下面代码中，第二个脚本文件 test2.js 可能会在第一个脚本文件 test1.js 之前执行。因此，用户要确保两个文件之间没有逻辑顺序的关联，互不依赖是非常重要的。

```
<!doctype html>
<html>
<head>
<script type="text/javascript" async src="test1.js"></script>
<script type="text/javascript" async src="test2.js"></script>
</head>
<body>
<!-- 网页内容 -->
</body>
</html>
```

指定 async 属性的目的是不让页面等待两个脚本文件下载完后再执行，从而异步加载页面其他内容。

【提示】

异步响应的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。异步脚本不要在加载期间修改 DOM。

支持异步脚本的浏览器包括 Firefox 3.6+、Safari 5 和 Chrome。在 XHTML 文档中，要把 async 属性设为 async="async"。

17.1.5 在 XHTML 中使用 JavaScript 脚本

XHTML（EXtensible HyperText Markup Language）表示可扩展超文本标记语言，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。

【提示】

将页面的 MIME 类型指定为 application/xhtml+xml.的情况下会触发 XHTML 模式，当然并不是所有浏览器都支待以这种方法。

【示例】下面代码块虽然在 HTML 中是有效的，但在 XHTML 中则是无效的。

```
<script type="text/javascript">
function test(a,b){
    if(a < b){
        alert(a + "<" + b);
    }
    else if(a > b){
        alert(a + ">" + b);
    }
    else{
        alert(a + "=" + b);
    }
}
test(1,3);
</script>
```

在 HTML 中，有特殊的规则用以确定<script>标签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。例如，上面代码中比较语句 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。

避免在 XHTML 中出现类似语法错误的方法：

方法一，使用相应的 HTML 转码(&1t;)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 test(a,b){
    if(a &lt; b){
        alert(a + "<" + b);
    }
    else if(a > b){
        alert(a + ">" + b);
    }
    else{
        alert(a + "=" + 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。

方法二，使用一个 CData 片段来包含 JavaScript 代码。在 XHTML 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，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 test(a,b){
    if(a < b){
```

```
        alert(a + "<" + b);
    }
    else if(a > b){
        alert(a + ">" + b);
    }
    else{
        alert(a + "=" + 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。这时可以使用 JavaScript 注释将 CData 标记注释掉就可以了：

```
<script type="text/javascript">
//<![CDATA[
function test(a,b){
    if(a < b){
        alert(a + "<" + b);
    }
    else if(a > b){
        alert(a + ">" + b);
    }
    else{
        alert(a + "=" + b);
    }
}
test(1,3);
//]]>
</script>
```

这样在所有现代浏览器中都可以正常使用，它能通过 XHTML 验证，而且能够兼容 XHTML 之前的浏览器。

17.1.6 兼容不支持 JavaScript 的浏览器

最早引入<script>标签时，与传统 HTML 的解析规则存在冲突。由于要对这个标签应用特殊的解析规则，因此在那些不支持 JavaScript 的浏览器中就会导致问题，即会把<script>标签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>标签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。

【示例】下面代码给脚本加上 HTML 注释后，不支持 JavaScript 的浏览器就会忽略<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。

```
<script><!--
function test(a,b){
    alert("ok");
}
//--></script>
```

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，也不再推荐使用这种方法。

在 XHTML 模式下，因为脚本包含在 XML 注释中，所以脚本会被忽略。