XML

Author williamfan 2017-07-01

website pengfeifan.github.io

XML
XML
什么是 XML？
XML 树结构
XML 语法规则
XML 元素
XML 属性
XML 验证
XMLHttpRequest 对象
XML Parser
XML DOM
XML DOM 高级
XML 命名空间
XML CDATA
XML 编码
XML 相关技术
XML 编辑器
XML - E4X
XML 总结
XML DOM（Document Object Model）
XSLT（XML 样式表语言转换）
XML DTD（文档类型定义）
XML Schema

XML 实例

XML

XML 指可扩展标记语言（**eXtensible Markup Language**）
XML 被设计用来传输和存储数据。
HTML 被设计用来显示数据。

什么是 XML？

- XML 指可扩展标记语言（EXtensible Markup Language）。
- XML 是一种很像HTML的标记语言。
- XML 的设计宗旨是传输数据，而不是显示数据。
- XML 标签没有被预定义。需要自行定义标签。
- XML 被设计为具有自我描述性。
- XML 是 W3C 的推荐标准。

XML 树结构

XML 文档形成了一种**树结构**，它从`根部`开始，然后扩展到`枝叶`。
一个 XML 文档实例，XML 文档使用简单的具有自我描述性的语法：

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

第一行是`XML 声明`。它定义 XML 的版本（1.0）和所使用的编码（UTF-8：万国码, 可显示各种语言）。
下一行描述文档的`根元素`（像在说："本文档是一个便签"）：
`<note>`
接下来 4 行描述根的 4 个`子元素`（to, from, heading 以及 body）：

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

最后一行定义根元素的结尾：
`</note>`
您可以假设，从这个实例中，XML 文档包含了一张 Jani 写给 Tove 的便签。
XML 具有出色的自我描述性

XML 文档形成一种树结构

- XML 文档必须包含根元素。该元素是所有其他元素的父元素。
- XML 文档中的元素形成了一棵**文档树**。这棵树从根部开始，并扩展到树的最底端。
  所有的元素都可以有子元素：

```
<root>
<child>
<subchild>.....</subchild>
</child>
</root>
```

父、子以及同胞等术语用于描述元素之间的关系。父元素拥有子元素。相同层级上的子元素成为同胞（兄弟或姐妹）。
所有的元素都可以有文本内容和属性（类似 HTML 中）。

XML 语法规则

1. XML 文档必须有根元素

XML 必须包含根元素，它是所有其他元素的父元素，比如以下实例中 root 就是根元素：

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

2. XML 声明

XML 声明是文件的**可选部分**，如果存在需要放在文档的第一行，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
```

以上实例包含 XML 版本（
UTF-8 也是 HTML5, CSS, JavaScript, PHP, 和 SQL 的默认编码。
所有的 XML 元素都必须有一个关闭标签
注释：**XML 声明没有关闭标签。这不是错误。声明不是 XML 文档本身的一部分，它没有关闭标签。**

3. XML 标签对大小写敏感

XML 标签对大小写敏感。标签 `<Letter>` 与标签 `<letter>` 是不同的。
必须使用相同的大小写来编写打开标签和关闭标签：
`<Message>这是错误的</message>`
`<message>这是正确的</message>`
注释：**打开标签和关闭标签通常被称为开始标签和结束标签。**

4. XML 必须正确嵌套

在 XML 中，所有元素都必须彼此正确地嵌套：
`<b><i>This text is bold and italic</i></b>`
在上面的实例中，正确嵌套的意思是：由于`<i>`元素是在`<b>`元素内打开的，那么它必须在`<b>`元素内关闭。

5. XML 属性值必须加引号

与 HTML 类似，XML 元素也可拥有属性（**名称/值的对**）。
在 XML 中，**XML 的属性值必须加**`引号`。
请研究下面的两个 XML 文档。 第一个是错误的，第二个是正确的：

```
<note date=12/11/2007>
<to>Tove</to>
<from>Jani</from>
</note>
```

```
<note date="12/11/2007">
<to>Tove</to>
<from>Jani</from>
</note>
```

在第一个文档中的错误是，note 元素中的 date 属性没有加引号。

6. 实体引用

在 XML 中，一些字符拥有特殊的意义。
如果您把字符 “<” 放在 XML 元素中，会发生错误，这是因为解析器会把它当作新元素的开始。
这样会产生 XML 错误：
`<message>if salary < 1000 then</message>`
为了避免这个错误，请用**实体引用**来代替 “<” 字符：
`<message>if salary < 1000 then</message>`
在 XML 中，有 5 个预定义的实体引用：

实体引用	字符	表示
`& lt;`	<	less than
`& gt;`	>	greater than
`& amp;`	&	ampersand
`& apos;`	‘	apostrophe
`& quot;`	“	quotation mark

注释：**在 XML 中，只有字符 “<” 和 “&” 确实是非法的。大于号是合法的，但是用实体引用来代替它是一个好习惯。**

7. XML 中的注释

在 XML 中编写注释的语法与 HTML 的语法很相似。

```
<!-- This is a comment -->
```

8. 在 *XML* 中，空格会被保留

HTML 会把多个连续的空格字符裁减（合并）为一个，
在 XML 中，文档中的空格不会被删减。

9. XML 以 *LF* 存储换行

在 `Windows` 应用程序中，换行通常以一对字符来存储：`回车符(CR)` 和 `换行符(LF)`。
在 `Unix 和 Mac OSX` 中，使用 `LF` 来存储新行。
在 `旧的 Mac 系统` 中，使用 `CR` 来存储新行。
XML 以 `LF` 存储换行。

XML 元素

XML 文档包含 XML 元素。

1. 什么是 *XML* 元素？

XML 元素指的是从（且包括）开始标签直到（且包括）结束标签的部分。
一个元素可以包含：

- 其他元素
- 文本
- 属性
- 或混合以上所有...

```
<bookstore>
    <book category="CHILDREN">
        <title>Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title>Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

在上面的实例中，`<bookstore>`和 `<book>` 都有**元素内容**，因为他们包含其他元素。`<book>` 元素也有**属性**（`category="CHILDREN"`）。`<title>`、`<author>`、`<year>` 和 `<price>` 有**文本内容**，因为他们包含文本。

2. XML 命名规则

XML 元素必须遵循以下命名规则：

- 名称可以包含`字母`、`数字`以及`其他的字符`
- 名称不能以数字或者标点符号开始
- 名称不能以字母 xml（或者 XML、Xml 等等）开始
- 名称不能包含空格
- 可使用任何名称，没有保留的字词。

3. 最佳命名习惯

使名称具有描述性。使用下划线的名称也很不错：`<first_name>`、`<last_name>`。
名称应简短和简单，比如：`<book_title>`，而不是：`<the_title_of_the_book>`。
避免`"-"`字符。如果您按照这样的方式进行命名：`"first-name"`，一些软件会认为您想要从 first 里边减去 name。
避免`"."`字符。如果您按照这样的方式进行命名：`"first.name"`，一些软件会认为“name” 是对象 “first” 的属性。
避免`":"`字符。冒号会被转换为命名空间来使用（稍后介绍）。
XML 文档经常有一个对应的数据库，其中的字段会对应 XML 文档中的元素。有一个实用的经验，即使用**数据库的命名规则**来命名 XML 文档中的元素。
在 XML 中，éòá 等非英语字母是完全合法的，不过需要留意，您的软件供应商不支持这些字符时可能出现的问题。

4. XML 元素是可扩展的

XML 元素是可扩展，以携带更多的信息。
请看下面的 XML 实例：

```
<note>
    <to>Tove</to>
    <from>Jani</from>
    <body>Don't forget me this weekend!</body>
</note>
```

让我们设想一下，我们创建了一个应用程序，可将 、 以及 元素从 XML 文档中提取出来，并产生以下的输出：

```
MESSAGE
To: Tove
From: Jani
Don't forget me this weekend!
```

想象一下，XML 文档的作者添加的一些额外信息：

```
<note>
    <date>2008-01-10</date>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

那么这个应用程序会中断或崩溃吗？
不会。这个应用程序仍然可以找到 XML 文档中的 `<to>` 、 `<from>` 以及 `<body>` 元素，并产生同样的输出。
XML 的优势之一，就是可以在不中断应用程序的情况下进行扩展。

XML 属性

XML元素具有属性，类似 HTML。
`属性(Attribute)` 提供有关元素的额外信息。

在 HTML 中，属性提供有关元素的额外信息：

```
<img src="computer.gif">
<a href="demo.html">
```

属性通常提供不属于数据组成部分的信息。在下面的实例中，文件类型与数据无关，但是对需要处理这个元素的软件来说却很重要：

```
<file type="gif">computer.gif</file>
```

XML 属性必须加引号

属性值必须被引号包围，不过单引号和双引号均可使用。比如一个人的性别，person 元素可以这样写：
`<person sex="female">`
或者这样也可以：
`<person sex='female'>`
如果属性值本身包含双引号，您可以使用单引号，就像这个实例：
`<gangster name='George "Shotgun" Ziegler'>`
或者您可以使用字符实体：
`<gangster name="George "Shotgun" Ziegler">`

XML 元素 vs. 属性

请看这些实例：

```
<person sex="female">
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

```
<person>
<sex>female</sex>
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

在第一个实例中，`sex 是一个属性`。在第二个实例中，`sex 是一个元素`。这两个实例都提供相同的信息。
在 HTML 中，属性用起来很便利，但是**在 XML 中，应该尽量避免使用属性**。如果信息感觉起来很像数据，那么请使用元素吧。
下面的三个 XML 文档包含完全相同的信息：
第一个实例中使用了 date 属性：

```
<note date="10/01/2008">
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

第二个实例中使用了 date 元素：

```
<note>
<date>10/01/2008</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

第三个实例中使用了扩展的 date 元素：

```
<note>
<date>
<day>10</day>
<month>01</month>
<year>2008</year>
</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

避免 XML 属性？

因使用属性而引起的一些问题：

- 属性不能包含多个值（元素可以）
- 属性不能包含树结构（元素可以）
- 属性不容易扩展（为未来的变化）
  属性难以阅读和维护。请尽量使用元素来描述数据。而仅仅使用属性来提供与数据无关的信息。
  不要做这样的蠢事（这不是 XML 应该被使用的方式）：

```
<note day="10" month="01" year="2008"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

针对元数据的 XML 属性

有时候会向元素分配 ID 引用。这些 ID 索引可用于标识 XML 元素，它起作用的方式与 HTML 中 id 属性是一样的。这个实例向我们演示了这种情况：

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

上面的 id 属性仅仅是一个**标识符**，用于标识不同的便签。它并不是便签数据的组成部分。
* `元数据（有关数据的数据）`应当存储为属性，而数据本身应当存储为元素。 *

XML 验证

拥有正确语法的 XML 被称为”形式良好”的 XML。
通过`DTD`验证的XML是”合法”的 XML。

形式良好的 XML 文档
`"形式良好"的 XML 文档拥有正确的语法`
在前面的章节描述的语法规则：

- XML 文档必须有一个根元素
- XML元素都必须有一个关闭标签
- XML 标签对大小写敏感
- XML 元素必须被正确的嵌套
- XML 属性值必须加引号

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

验证 XML 文档

合法的 XML 文档是"形式良好"的 XML 文档，这也符合`文档类型定义（DTD）`的规则：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

在上面的实例中，`DOCTYPE 声明`是对外部`DTD 文件`的引用。下面的段落展示了这个文件的内容。

XML DTD

DTD 的目的是`定义 XML 文档的结构`。它使用一系列合法的元素来`定义文档结构`：

```
   <!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

XML Schema

W3C 支持一种基于 XML 的 DTD 代替者，它名为 **XML Schema**：

```
<xs:element name="note">

  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

</xs:element>
```

XML 验证器

XML 错误会终止程序

W3C 的 XML 规范声明：如果 XML 文档存在错误，那么程序就不应当继续处理这个文档。
理由是，XML 软件应当轻巧，快速，具有良好的兼容性。
如果使用 HTML，创建包含大量错误的文档是有可能的（比如您忘记了结束标签）。其中一个主要的原因是 HTML 浏览器相当臃肿，兼容性也很差，并且它们有自己的方式来确定当发现错误时文档应该显示为什么样子。
使用 XML 时，这种情况不应当存在。

使用 CSS 显示 XML

使用 CSS 格式化 XML 不是常用的方法。W3C 推荐使用 XSLT

使用 XSLT 显示 XML

XSLT 是首选的 XML 样式表语言。
XSLT（eXtensible Stylesheet Language Transformations）远比 CSS 更加完善。
XSLT 是在浏览器显示 XML 文件之前，先把它转换为 HTML

XMLHttpRequest 对象

`XMLHttpRequest` 对象用于在前台与服务器交换数据。
`XMLHttpRequest` 对象是开发者的梦想，因为您能够：

- 在不重新加载页面的情况下更新网页
- 在页面已加载后从服务器请求数据
- 在页面已加载后从服务器接收数据
- 在前台向服务器发送数据

建议:
创建一个* *XMLHttpRequest** 对象
所有现代浏览器（IE7+、Firefox、Chrome、Safari 和 Opera）都有内建的 XMLHttpRequest 对象。
创建 XMLHttpRequest 对象的语法：
`xmlhttp=new XMLHttpRequest();`
旧版本的Internet Explorer（IE5和IE6）中使用 ActiveX 对象：
`xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");`

XML Parser

所有现代浏览器都有内建的`XML 解析器`。
`XML 解析器`把 `XML 文档`转换为 `XML DOM 对象` - 可通过 `JavaScript`操作的对象。

解析 XML 文档

下面的代码片段把* *XML* 文档**解析到** *XML DOM** 对象中：

```
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","books.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
```

解析 XML 字符串

下面的代码片段把 **XML 字符串**解析到 **XML DOM** 对象中：

```
txt="<bookstore><book>";
txt=txt+"<title>Everyday Italian</title>";
txt=txt+"<author>Giada De Laurentiis</author>";
txt=txt+"<year>2005</year>";
txt=txt+"</book></bookstore>";

if (window.DOMParser)
{
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(txt,"text/xml");
}
else // Internet Explorer
{
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async=false;
  xmlDoc.loadXML(txt);
}
```

注释：Internet Explorer 使用 `loadXML()` 方法来解析 XML 字符串，而其他浏览器使用 `DOMParser` 对象。

跨域访问

XML DOM

`DOM（Document Object Model 文档对象模型）` 定义了访问和操作文档的标准方法。

`XML DOM（XML Document Object Model）` 定义了访问和操作 XML 文档的标准方法。
XML DOM 把 XML 文档作为 `树结构` 来查看。
所有元素可以通过 DOM 树来访问。可以修改或删除它们的内容，并创建新的元素。元素，它们的文本，以及它们的属性，都被认为是节点。

HTML DOM

`HTML DOM` 定义了访问和操作 HTML 文档的标准方法。
所有 HTML 元素可以通过 HTML DOM 来访问。

加载一个 XML 文件 - 跨浏览器实例

下面的实例把 XML 文档（”note.xml”）解析到 XML DOM 对象中，然后通过 JavaScript 提取一些信息：
实例

```
<html>
<body>
<h1>W3Schools Internal Note</h1>
<div>
  <b>To:</b> <span id="to"></span><br />
  <b>From:</b> <span id="from"></span><br />
  <b>Message:</b> <span id="message"></span>
</div>

<script>
  if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
      xmlhttp=new XMLHttpRequest();
  }
  else
  {// code for IE6, IE5
      xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }

  xmlhttp.open("GET","note.xml",false);
  xmlhttp.send();
  xmlDoc=xmlhttp.responseXML;

  document.getElementById("to").innerHTML=
        xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValu
e;
  document.getElementById("from").innerHTML=
        xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeVa
lue;
  document.getElementById("message").innerHTML=
        xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeVa
lue;
</script>
</body>
</html>
```

重要注释！
如需从上面的 XML 文件（"note.xml"）的 `<to>` 元素中提取文本 "Tove"，语法是：
`getElementsByTagName("to")[0].childNodes[0].nodeValue`
请注意，即使 XML 文件只包含一个 `<to>` 元素，仍然必须指定数组索引 [0]。这是因为 `getElementsByTagName()` 方法返回一个数组。

加载一个 XML 字符串 - 跨浏览器实例

下面的实例把 XML 字符串解析到 XML DOM 对象中，然后通过 JavaScript 提取一些信息：
实例

```
<html>
<body>
<h1>W3Schools Internal Note</h1>
<div>
  <b>To:</b> <span id="to"></span><br />
  <b>From:</b> <span id="from"></span><br />
  <b>Message:</b> <span id="message"></span>
</div>

<script>
txt="<note>";
txt=txt+"<to>Tove</to>";
txt=txt+"<from>Jani</from>";
txt=txt+"<heading>Reminder</heading>";
txt=txt+"<body>Don't forget me this weekend!</body>";
txt=txt+"</note>";

if (window.DOMParser)
{
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(txt,"text/xml");
}
else // Internet Explorer
{
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async=false;
  xmlDoc.loadXML(txt);
}

document.getElementById("to").innerHTML=
    xmlDoc.getElementsByTagName("to")[0].childNodes[0].nodeValue;
document.getElementById("from").innerHTML=
    xmlDoc.getElementsByTagName("from")[0].childNodes[0].nodeValue;
document.getElementById("message").innerHTML=
    xmlDoc.getElementsByTagName("body")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

在 HTML 页面中显示 XML 数据

在下面的实例中，打开一个 XML 文件（”cd_catalog.xml”），然后遍历每个 CD 元素，并显示HTML 表格中的 ARTIST 元素和 TITLE 元素的值：
实例

```
<html>
<body>

<script>
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","cd_catalog.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

document.write("<table border='1'>");
var x=xmlDoc.getElementsByTagName("CD");
for (i=0;i<x.length;i++)
{
    document.write("<tr><td>");
    document.write(x[i].getElementsByTagName("ARTIST")[0].childNode
s[0].nodeValue);
    document.write("</td><td>");
    document.write(x[i].getElementsByTagName("TITLE")[0].childNode
s[0].nodeValue);
    document.write("</td></tr>");
}
    document.write("</table>");
</script>

</body>
</html>
```

`cd_catalog.xml`

```
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Unchain my heart</TITLE>
    <ARTIST>Joe Cocker</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>EMI</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1987</YEAR>
  </CD>
</CATALOG>
```

XML DOM 高级

获取元素的值

下面的实例中使用的 XML 文件：`books.xml`。
下面的实例检索第一个 `<title>` 元素的文本值：
实例

```
txt=xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
```

获取属性的值

下面的实例检索第一个 `<title>` 元素的 “lang” 属性的文本值：
实例

```
txt=xmlDoc.getElementsByTagName("title")[0].getAttribute("lang");
```

改变元素的值

下面的实例改变第一个 `<title>` 元素的文本值：
实例

```
x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.nodeValue="Easy Cooking";
```

创建新的属性

XML DOM 的 setAttribute() 方法可用于改变现有的属性值，或创建一个新的属性。
下面的实例创建了一个新的属性（edition="first"），然后把它添加到每一个 `<book>` 元素中：
实例

```
x=xmlDoc.getElementsByTagName("book");

for(i=0;i<x.length;i++)
{
    x[i].setAttribute("edition","first");
}
```

创建元素

XML DOM 的 createElement() 方法创建一个新的元素节点。
XML DOM 的 createTextNode() 方法创建一个新的文本节点。
XML DOM 的 appendChild() 方法向节点添加子节点（在最后一个子节点之后）。
如需创建带有文本内容的新元素，需要同时创建元一个新的元素节点和一个新的文本节点，然后把他追加到现有的节点。
下面的实例创建了一个新的元素 `（<edition>）`，带有如下文本：`First`，然后把它添加到第一个 `<book>` 元素：
实例

```
newel=xmlDoc.createElement("edition");
newtext=xmlDoc.createTextNode("First");
newel.appendChild(newtext);

x=xmlDoc.getElementsByTagName("book");
x[0].appendChild(newel);
```

实例解释
创建一个 `<edition>` 元素
创建值为 `First` 的文本节点
把这个文本节点追加到新的 `<edition>` 元素
把 `<edition>` 元素追加到第一个 `<book>` 元素

删除元素

下面的实例删除第一个 `<book>` 元素的第一个节点：
实例

```
x=xmlDoc.getElementsByTagName("book")[0];
x.removeChild(x.childNodes[0]);
```

XML 命名空间

XML 命名空间提供避免元素命名冲突的方法。

命名冲突
在 XML 中，元素名称是由开发者定义的，当两个不同的文档使用相同的元素名时，就会发生**命名冲突**。
这个 XML 携带 HTML 表格的信息：

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

这个 XML 文档携带有关桌子的信息（一件家具）：

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

假如这两个 XML 文档被一起使用，由于两个文档都包含带有不同内容和定义的 `<table>` 元素，就会发生命名冲突。
XML 解析器无法确定如何处理这类冲突。

使用前缀来避免命名冲突

在 XML 中的命名冲突可以通过使用名称前缀从而容易地避免。
该 XML 携带某个 HTML 表格和某件家具的信息：

```
<h:table>
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>

<f:table>
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>
```

在上面的实例中，不会有冲突，因为两个 `<table>` 元素有不同的名称。

XML 命名空间 - xmlns 属性

当在 XML 中使用前缀时，**一个所谓的用于前缀的命名空间必须被定义。**
命名空间是在元素的开始标签的 `xmlns 属性` 中定义的。
命名空间声明的语法如下。

xmlns:前缀="URI"。

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>

<f:table xmlns:f="http://www.w3.org/furniture">
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>

</root>
```

在上面的实例中，`<table>` 标签的 `xmlns 属性` 定义了 `h:` 和 `f:` 前缀的合格 `命名空间`。
当命名空间被定义在元素的开始标签中时，所有带有相同前缀的子元素都会与同一个命名空间相关联。
命名空间，可以在他们被使用的元素中或者在 XML 根元素中声明：

```
<root xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3cschool.cc/furniture">

<h:table>
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>

<f:table>
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>

</root>
```

注释：命名空间 URI 不会被解析器用于查找信息。其目的是赋予命名空间一个惟一的名称。
不过，很多公司常常会作为指针来使用命名空间指向实际存在的网页，这个网页包含关于命名空间的信息。
请访问 http://www.w3.org/TR/html5/。

统一资源标识符（URI，全称 Uniform Resource Identifier）

统一资源标识符（URI）是一串可以标识因特网资源的字符。
最常用的 URI 是用来标识因特网域名地址的统一资源定位器（URL）。另一个不那么常用的 URI 是统一资源命名（URN）。

默认的命名空间

为元素定义默认的命名空间可以省去在所有的子元素中使用前缀的工作。它的语法如下：
`xmlns="namespaceURI"`
这个 XML 携带 HTML 表格的信息：

```
<table xmlns="http://www.w3.org/TR/html4/">
<tr>
<td>Apples</td>
<td>Bananas</td>
</tr>
</table>
```

这个XML携带有关一件家具的信息：

```
<table xmlns="http://www.w3schools.com/furniture">
<name>African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>
```

实际使用中的命名空间

XSLT 是一种用于把 XML 文档转换为其他格式的 XML 语言，比如 HTML。
在下面的 XSLT 文档中，您可以看到，大多数的标签是 HTML 标签。
非 HTML 的标签都有前缀 xsl，并由此命名空间标识：`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`：

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
    <tr>
        <th align="left">Title</th>
        <th align="left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

XML CDATA

XML 文档中的所有文本均会被解析器解析。
只有 `CDATA 区段` 中的 `文本` 会被解析器忽略。

PCDATA - 被解析的字符数据

XML 解析器通常会解析 XML 文档中所有的文本。
当某个 XML 元素被解析时，其标签之间的文本也会被解析：
`<message>This text is also parsed</message>`
解析器之所以这么做是因为 XML 元素可包含其他元素，就像这个实例中，其中的 `<name>` 元素包含着另外的两个元素（first 和 last）：
`<name><first>Bill</first><last>Gates</last></name>`
而解析器会把它分解为像这样的子元素：

```
<name>
<first>Bill</first>
<last>Gates</last>
</name>
```

解析字符数据(PCDATA) 是 XML 解析器解析的文本数据使用的一个术语。

CDATA - （未解析）字符数据

术语 `CDATA` 是不应该由 XML 解析器解析的文本数据。
像 `"<"` 和 `"&"` 字符在 XML 元素中都是非法的。
`"<"` 会产生错误，因为解析器会把该字符解释为新元素的开始。
`"&"` 会产生错误，因为解析器会把该字符解释为字符实体的开始。
某些文本，比如 JavaScript 代码，包含大量 `"<"` 或 `"&"` 字符。
为了避免错误，可以将脚本代码定义为 `CDATA`。

CDATA 部分中的所有内容都会被解析器忽略

CDATA 部分由 `"<![CDATA["` 开始，由 `"]]>"` 结束：

`<![CDATA[ ]]>`

```
<script>
<![CDATA[
    function matchwo(a,b)
    {
        if (a < b && a < 0) then
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
]]>
</script>
```

在上面的实例中，解析器会忽略 `CDATA` 部分中的所有内容。

关于 CDATA 部分的注释：
CDATA 部分不能包含字符串 “]]>”。也不允许嵌套的 CDATA 部分
标记 CDATA 部分结尾的 “]]>” 不能包含空格或换行。

XML 编码

不同编码的体验：

```
<?xml version="1.0" encoding="us-ascii"?>
<?xml version="1.0" encoding="windows-1252"?>
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-16"?>
```

结论

- 始终使用编码属性
- 使用支持编码的编辑器
- 确保编辑器使用什么编码
- 编码属性中使用相同的编码

XML 相关技术

下面是一个 XML 技术的列表。

- XHTML (可扩展 HTML)
  更严格更纯净的基于 XML 的 HTML 版本。
- XML DOM (XML 文档对象模型)
  访问和操作 XML 的标准文档模型。
- XSL (可扩展样式表语言) XSL 包含三个部分：
  - XSLT (XSL 转换) - 把 XML 转换为其他格式，比如 HTML
  - XSL-FO (XSL 格式化对象)- 用于格式化 XML 文档的语言
  - XPath - 用于导航 XML 文档的语言

- XQuery (XML 查询语言)
  基于 XML 的用于查询 XML 数据的语言。
- DTD (文档类型定义)
  用于定义 XML 文档中的合法元素的标准。
- XSD (XML 架构)
  基于 XML 的 DTD 替代物。
- XLink (XML 链接语言)

在 XML 文档中创建超级链接的语言。
- XPointer (XML 指针语言)
  允许 XLink 超级链接指向 XML 文档中更多具体的部分。
- SOAP (简单对象访问协议)
  允许应用程序在 HTTP 之上交换信息的基于 XML 的协议。
- WSDL (Web 服务描述语言)
  用于描述网络服务的基于 XML 的语言。
- RDF (资源描述框架)
  用于描述网络资源的基于 XML 的语言。
- RSS (真正简易聚合)
  聚合新闻以及类新闻站点内容的格式。
- SVG (可伸缩矢量图形)
  定义 XML 格式的图形。

XML 编辑器

为什么使用 XML 编辑器?
当今，XML 是非常重要的技术，并且开发项目正在使用这些基于 XML 的技术：

- 用 XML Schema 定义 XML 的结构和数据类型
- 用 XSLT 来转换 XML 数据
- 用 SOAP 来交换应用程序之间的 XML 数据
- 用 WSDL 来描述网络服务
- 用 RDF 来描述网络资源
- 用 XPath 和 XQuery 来访问 XML 数据
- 用 SMIL 来定义图形
  为了能够编写出无错的 XML 文档，您需要一款智能的 XML 编辑器！

XMLSpy 是强大的XML 编辑器。

XML - E4X

E4X 向 JavaScript 添加了对 XML 的直接支持。
E4X 是一个 ECMAScript（JavaScript）标准
ECMAScript 是 JavaScript 的正式名称。ECMA-262（JavaScript 1.3）是在 1999 年 12 月标准化的。
E4X 是 JavaScript 的扩展，增加了对 XML 的直接支持。ECMA-357（E4X）是在 2004 年 6 月标准化的。

XML 总结

- XML 可用于交换、共享和存储数据。
- XML 文档形成 树状结构，在”根”和”叶子”的分支机构开始的。
- XML 有非常简单的 语法规则。带有正确语法的 XML 是”形式良好”的。有效的 XML 是针对 DTD 进行验证的。
- XSLT 用于把 XML 转换为其他格式，比如 HTML。
- 所有现代的浏览器有一个内建的 XML 解析器，可读取和操作 XML。
- DOM（Document Object Model）定义了一个访问 XML 的标准方式。
- XMLHttpRequest 对象提供了一个网页加载后与服务器进行通信的方式。
- XML 命名空间提供了一种避免元素命名冲突的方法。
- CDATA 区域内的文本会被解析器忽略。

XML DOM（Document Object Model）

XML DOM 定义了一种访问和处理 XML 文档的标准方式。
XML DOM 是平台和语言独立的，可用于任何编程语言，如 Java、JavaScript 和 VBScript。

XSLT（XML 样式表语言转换）

XSLT 是 XML 文件的样式表语言。
通过使用 XSLT，可以把 XML 文档转换为其他格式，比如 XHTML。

XML DTD（文档类型定义）

DTD 的目的是定义 XML 文档中合法的元素、属性和实体。
通过使用 DTD，每个 XML 文件可以随身携带它自己的格式的描述。
DTD 可以被用来确认您收到的数据和您自己的数据是否有效。

XML Schema

XML Schema 是一种基于 XML 的 DTD 替代。
不像 DTD，XML Schema 支持数据类型，且使用 XML 语法。

XML 实例

examples