第 35 卷 第 4 期 计 算 机 工 程 2009 年 2 月
Vol.35 No.4 Computer Engineering February 2009
·软件技术与数据库· 文章编号：1000—3428(2009)04—0048—03 文献标识码：A 中图分类号：TP311

基于 C 语言的轻量级高效 XML 编解码器

查 峰
(华为技术有限公司南京研究所，南京 210012)

摘 要：将树形数据结构用于最小化 DOM 模型 XML 编解码器的开发。在解析 XML 文本时，基于 Expat 解释器将 XML 字符串构造成树状结构；编码时构造 DOM 树结构，采用非递归深度优先法遍历树，将其串行化成 XML 串。本编解码器解决了国际化语言关键问题，是一个通用、高效的工业级 XML 应用支撑模块。

关键词：XML 解析器；DOM 模型；Expat 解释器

Light Effective XML Encoding and Decoding Module Based on C

ZHA Feng
(Nanjing Institute, Huawei Technology Company, Nanjing 210012)

【Abstract】 This paper designs a DOM liked tree data structure for developing a minimal DOM based eXtended Marked Language(XML) coding and encoding module. When parsing XML text, based on expat, it puts the parsed content into the tree. When encoding, it constructs a DOM tree, then use non-recursion depth-first method to access the tree and serialize it into XML text. This module also solves the key problem of international language, and forms an universal, highly effective and industrial-level-reached basic module for XML application.

【Key words】 eXtended Marked Language(XML) parser; DOM model; Expat interpreter

1 概述

扩展标记语言(eXtended Marked Language, XML)是一种能进行自我描述的协议语言[1]。目前开发人员在软件项目中采用大量 XML 技术进行软件规则的配置和模块间协议接口的开发。

在 C++软件中，Windows 环境中的 VC7.0 提供基于 COM 组件 XML 接口。基于开源模块的 Xcerse 也是一个广泛使用的的解析器。但对于很多开发人员，特别是 C 程序员来说，仍缺少一种真正能方便使用的 XML 解析器。微软提供的 XML 接口基于 COM，非常复杂，难于使用；Xcerse 解析器库大，配置参数多，处理 XML 信息时工作量较大，而且 Xcerse 不能用于标准 C 项目的开发。一些开发人员选用另一个 C 开源模块 Expat[2]来开发软件项目，Expat 最初在 Apache Web 服务器中使用，效率非常高，很多高级 XML 解析器以 Expat 为基础封装成 XML 组件。Expat 基于 SAX 流的解析模式，直接使用他们每解析一个 XML 元素，都须在标记处理回调函数中做元素标记名匹配，遇到父节点和子节点同名时很难处理，而在解析文本型元素值时，如遇到转移字符，则会将一段文本截断成几个元素，同时 Expat 只能处理 UTF-8 或 UNICODE 等有限编码的字符串，对 GB2312 和 GBK 等格式的中文 XML 文本的解析不支持。

本文介绍的 XML 模块 MiniDom 是基于 Expat，按 DOM 接口标准设计数据结构，并进行了相应的算法处理，提供完备的 API。解析前进行国际化语言预处理，解析中进行结构化处理，解析后又进行 2 次处理，将树形数据结构完整呈现给用户，并提供类 DOM 的 API，使用户可方便地对 XML 文档进行访问或修改，并快速构建 XML 应用。

2 预备知识

2.1 XML 结构

XML 可定义为文档对象模型(DOM)。每个文档可用如下树形结构来表示：(1)文档总是从根节点(元素)开始，每个 XML 文档至少有 1 个根节点；(2)每个节点有 0 个或多个属性，每个属性以“属性名=属性值”的值对形式表示；(3)每个节点有 0 个或多个孩子节点，主要有元素型节点、文本型节点、CDMA 和注释型节点。元素型标记对象只有标记名称和属性；文本节点没有标记名称，只有文本值；CDMA 型节点类似文本型，但其值包含于“<![CDATA[“和”]]>”之间，内容不经过字符转义处理；注释型节点不作为 DOM 文档正式内容，仅作说明用。

XML 文档中可包含 DTD，用来规定 XML 文档的结构和标记内容的约束关系。DTD 用正则式定义了组成文档的元素、不同层次元素的关系、元素可能出现的顺序和数目、元素的属性组成。但 DTD 是可选的，它是对 XML 文档的模式说明，可写在每个 XML 实例文本中。在定义 XML 文档时，一般单独将 DTD 片段写在设计文档中，作为 XML 文档的规约，这是涉及 XML 文档的开发依据。在解析 XML 串时，XML 按 DTD 规约对内容进行合法验证。有些开发人员在编写 XML 文档实例子时，并未将 DTD 片段写入，解析完成后才用编码来检验 XML 标记内容的合法性。因为用 DTD 验证解析器必须建立复杂的哈希结构树进行信息匹配，会消耗大量资源。实验结果表明，使用 DTD 验证，解析效率随 XML 模

作者简介：查 峰(1971－)，男，硕士，主研方向：网络通信协议
收稿日期：2008-05-22 **E-mail**：fengzha@china.com

式复杂程序会有不同程度的下降，甚至低 1 倍以上。

2.2 Expat 工作原理

Expat 是开源 XML 模块，提供简单应用程序接口(Simple API for XML, SAX)的模式解析。SAX 引擎完全由事件驱动，当遇到一个标记时，它就会调用适当的函数来处理它，因此，SAX 快速、有效。Expat 模块提供很多接口及回调函数，只要掌握以下几个关键函数，即可构建功能更为强大的 XML 模块：

(1)void* XML_ParserCreate(char*encoding)

创建 XML 解析器对象。其中 encoding 是编码。

(2)int XML_SetEncoding(void*parser, char*encoding)

设定编码格式。其中 parse 是解析器对象；encoding 是编码格式。

(3)XML_SetUserData(void*parser, void*userData)

设定用户数据。其中，parser 是解析器；userData 是用户自定义数据结构的指针。该函数非常重要，本文构造的 XML 树结构根节点指针由该函数传入，在回掉函数中处理。

(4)void XML_SetElementHandler(void*parser, XML_Start ElementHandler start, XML_EndElementHandler end)

设定用户处理标记开始和结束的处理回调函数函数，该函数是核心函数。

(5)typedef void (*XML_StartElementHandler)(void*user Data, char*name, char*atts)

处理解析 XML 元素标记开始的回调函数，由用户自定义。其中，name 指向当前处理元素标记名；atts 指向解析出该元素所有属性值对；设有 n 个属性，atts[2i-2]指向第 i 个元素(1<i≤n)的属性名，atts[2i-1] 指向该属性值，atts[2n]=0；typedef void (*XML_EndElementHandler)(void *userData, const XML_Char*name)是处理解析 XML 元素标记结束的回调函数，name 是标记名。

(6)void XML_SetCharacterDataHandler(void*parser, XML_CharacterDataHandler handler)

设置文本节点的处理函数。第 2 个参数是用户自定义数据结构的指针。typedef void (*XML_CharacterDataHandler) (void *userData, const XML_ Char*s, int len)的回调函数由用户自定义，s 指向解析后的文本串指针，len 为串长度。

2.3 XML 国际语言问题

XML 规范支持多种语言编码格式，如 UTF-16, UTF8 和 GB2312 等，一般在 XML 文档首句声明中 *encoding* 的值说明了编码的格式，如没有此行声明，则默认为 UTF-8 格式。

Libconv 是国际语言开源模块，提供不同编码语言的转换的 API。有些 Unix/Linux 操作系统安装时就包含了该库，可直接利用操作系统的 API。但为提高模块的可移植性，MiniDom 将 Libconv 模块中的语言转换函数提取出来重写成静态函数，这就不会与包含 Livconv 的 Lnux 系统中发生库冲突。一般不同种国际语言不能进行相互转换，要先由宿主语言转换为 UCS-16 格式，然后再转换到目标语言格式。

3 XML 编解码的实现

3.1 数据结构

编码器在解析 XML 串时，将每个 XML 元素存入一个元素对象，组织好上下层节点和兄弟节点的关系，最终构造一棵 XML 树。根节点代表了顶层元素；解码时则先构造数结构，然后串行为字符流。元素节点的数据结构如图 1 所示。

(1)属性对象

```
struct _attr
{   char *name;   //属性名称
    char *value;  //属性值
    struct _attr * next;  //下一个元素
}_ATTR；
```

(2)元素节点对象

```
typedef enum ELEL_TYPE
{   XML_ELEMENT=0, //元素型节点
    XML_TEXT,      //文本型节点
    XML_CDATA      // CDATA 型节点
}XML_ELEL_TYPE；//元素类型
typedef struct _element
{   char *name;     //元素标记名称
    ATTR *attrs;    //元素的属性对象链表
    struct _element*parent;   //父节点元素
    struct _list *children;   //下层节点链表
    char *val;      //元素值
}_XML;
struct _list； struct _list
{   struct _element *element； //元素节点
    struct _list *next；   //下一个节点
    struct _list *prev；   //前一个节点
};
```

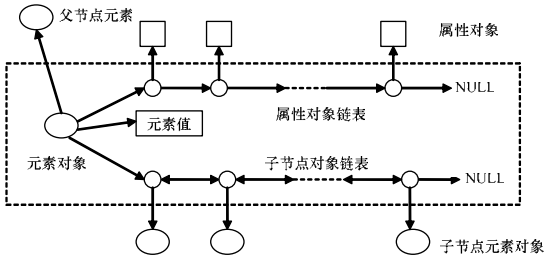

图 1 XML 元素节点数据结构

3.2 XML 解码

(1)进行 XML 的语言预处理

如果最初 2 个字节是 0xFF 和 0xFE 或 0xFE 和 0xFF，则是 unicode 编码。否则是其他编码，根据 *encoding* 的值来获取；如果最初 1 组“<?xml“和”>”内没有 *encoding*，则默认是 utf-8 编码。MiniDom 编码器统一用 utf-8 作为宿主语言。

(2)初试化

```
_XML*XML=NULL；
//先声明根节点，将该对象指针的指针设为用户数据参数
void*parser=XML_ParserCreate(NULL);
XML_SetEncoding(parser,"UTF-8");
XML_SetUserData (parser, (void *) & xml);
//设定回调函数
XML_SetElementHandler(parser, start_element, end_ element);
XML_SetCharacterDataHandler(parser, char_data);
调用解析函数 XML_Parse
```

(3)解析过程处理

在 start_element 回调函数中，进入了一个元素的标记处理过程：

1)创建元素型_XML 对象 elem,其字段 name 等于入参 1,value 设为 NULL；

2)遍历 atts，i 从 0 开始计数,若 atts[2i]为 0，执行(3)，否则创建第 i 个属性，atts[2i]和 atts[2i+1]分别为第 i 个属性的名称和值，将该属性插入到 elem 的 attrs 链表尾部，i 加 1，重复本步骤；

3)如果入参 userData 为空,则本元素为根节点；否则 userData 指针值指向父节点，将 elem 插入到父节点的 children 链表尾部；

4)userData 所指值更新为 elem，在进入下一层节点标记处理时，elem 就可为下层节点的父节点指针；

5)在 char_data 回调函数中，进入文本或 CDATA 型节点处理过程；

6)创建文本型_XML 对象 text，其 name 等于 NULL，value 等于入参 2；

7)同样用 userData 指针值获取父节点指针,将 elem 插入到父节点

的 children 链表尾部。

(4)对 XML 树再加工

当 DOM 型树结构已建立时，从根节点开始遍历树[1]，即可获取全部的 XML 信息。但此时还有很多信息不归整的问题。当 Expat 扫描到文本元素转义字符时，元素标记间一段连续文字就会分割在若干文本对象内，用户使用很不方便，还须再加工；XML 文本中如果混编了 CDATA 的字符片段，解析过程中也会作为一个文本节点处理。用户访问时，须编写大量合并节点的应用程序。

MiniDom 在将根节点返回给用户前对其进行整理，从根节点开始广度遍历 XML 树，每访问到一个元素节点 *A* 时，将其下层节点链表中所有连续的文本型节点合并为一个文本节点 *B* 后删除，最终节点值就是上述节点值的拼接；如果该链表中所有节点都是文本型节点，在合并这些节点后，将该节点的 *value* 赋给 A 的 *value*,并将 *B* 节点删除。*A* 的 *value* 就是元素标记的对间文本。此时解析过程完毕。

(5)接口

MiniDom 按 DOM 模型的接口标准提供了节点名称和节点文本，获取第一个子节点、本节点兄弟节点、本节点属性列表、任一位置的属性对象，按属性名获取属性对象等方法。

此外，MiniDom 还提供了重要方法——按路径获取节点对象列表，提供开发者在复杂的树中按路径直接定位节点的方法。通过这些 API，开发用户可访问所有 XML 信息。算法如下：

1)输入样式串 s_1，设有队列 *T, S*，元素列表 *L*，临时路径串数组 *t*，串指针 s_2, s_3，约定元素加入某路径就是该路径串尾 *strcat* 元素名后接路径分割符；

2)*t* 初试化为路径分割符，将 *t*+1 赋予 s_2, s_3，XML 树根元素插入 *T* 表尾；

3)如果 *T* 不为空，从 T 删除第一个元素并得其值 *e*，*e* 加入路径 s_2；

如果 s_1 等于 *t*，将 *e* 加入 L，否则如果 s_1 包含 *t*,将 *e* 的所有元素型孩子节点加入 *S* 队尾；

如果 *T* 不为空，将 e 加入路径 s_3，将 s_3 串尾指针加 1 赋予 s_3，此时如果 *S* 不为空，将 *S* 中所有元素自队头逐个取出，加入 *T* 队尾；转至 3)；

4)销毁 *T,S*，返回 *L*，*L* 中包含了所有符合样式 *s*1 的元素。

3.3 XML 编码

XML 字符串的编码是解析的逆过程，须理解 XML 文件的模式构造 XML 树，算法如下：

1)一般从根节点开始构造树，按深度优先[3]的原则。

2)根节点的父节点 parent 域为 NULL。

3)为每个 XML 节点创建一个_XML 结构对象，name 对应标记名称，value 是标记元素的值 如果是文本或 CDATA 型节点，则 name 为 NULL。

4)为每个元素的每个属性值对创建_ATTR 对象，name 为属性名，value 为属性值，将_ATTR 对象插入元素对象 attrs 所指链表尾部。

5)将新生成非根节点对象插入父节点对象 children 链表尾部，并将本对象的 parent 域置为指向父节点，如 XML 树未构造完，转 3)。

6)将 XML 树结构串行化，这是类似遍历广义表的过程：

①设有队列 L、T，s 为输出流,先取得根节点指针插入 L。

②如果 L 不为空，取得并访问 L 队首元素 A。

情形 1：节点是元素型且未访问过,构造开始标记，依次将标记名写入 s，从元素的属性链表中对象依次取出，每个属性按“ 对属性名=属性值 ”模式写入 s，最后将元素值写入 s，给 A 打上已访问标志。将 A 的所有子节点依次出入 T，然后将 T 中元素依次从尾部取出插入到 L 头部。

情形 2：节点是元素型且已访问过,将 A 元素名填入结束的标记写入 s。

情形 3：节点是文本型,直接元素值写入 s。

情形 4：节点是 CDATA 型,直接将元素值写入 s。

情形 2~情形 4 执行完须将 A 从 L 中删除。

清空 T 后转②。

通过递归方法实现，在串行化非 CDATA 元素值时，遇到需要对“ & ”、“ < ” 、“ > ”、“ \ ”、“ \ ” 5 个转义字符时，要分别用“ &amp； ”、“ &lt； ”、“ &gt； ”、“ &apos； ”、“ &quot； ”替换写入输出流；否则左右分别连接“ <![CDATA["和]]> ”写入输入流。

7)将 XML 串转换为目标语言编码,加入 XML 报文声明行，并将编码格式在“ <?xml ”行的“ encode= ”后给予编码说明，如果是 UTF-8 可免去写入此行。此串就是要构造的 XML 流。

8)释放 XML 树，算法结束。

3.4 XML 树的其他方法

不论编码或解码，都将获得以_XML 为节点的树，本文还提供了对给定元素节点按名称删除属性，删除或复制指定节点为根的子树，通过这些方法实现对 XML 的裁减或扩充，实现对 XML 树的重构后再串行化，在很多项目中此功能非常有效。

4 结束语

本文介绍的 MiniDom 是一个遵循最小化 DOM 标准的 XML 编码解码器，其编码构建于 Expat 之上，经数据结构处理，形成 DOM 树型结构；编码时则先结构化，再串行化为 XML 流。XML 是递归的自我描述结构，本文算法全部用非递归方法实现，减少了函数对栈空间的占用。功能包含了 DOM 所有的接口,是个小而全的基础软件部件。经测试证明，MiniDom 编码器 1 s 能解析 2 000 个~5 000 个节点的 XML 流，优于其他流行的解析器。在几十个大型网关项目中使用多年，未出现过任何故障，编解成功所有合法的 XML 包。在 Unix/Linux/windows/winCE/vxWorks 多种平台中承担了支撑的功能，也证明了其优异的跨平台性。

本文设计的整个 MiniDom 模块基于 C 语言,移植方面只需对编译脚本稍做修改即可。进一步研究工作是设计高效内存管理器，对 MiniDom 编码器做进一步优化。

参考文献

[1] G W3C XML Working Group. XML1.0 Recommendation[Z]. (2000-10-01). http://www.w3c.org.

[2] Clark J. Expatopen Source Project[Z]. (2004-07-22). http://expat.sourceforge.net.

[3] 严蔚敏. 数据结构[M]. 北京: 清华大学出版社, 1996.