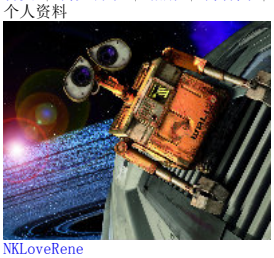

TinyXML Tutorial 中文指南 (2008-01-01 17:19)
分类： C/C++编程

TinyXML 指南

注： 本文是 TinyXML 2.5.3 版本 Document 中的《TinyXML Tutorial》的翻译文档，由本人 Dennis.Gao 翻译，版权归原作者所有，转载本文档请注明出处。原文出自 TinyXML

Author : Dennis.Gao

Date : 2008.01.01

这是什么？

本指南就如何有效的使用 TinyXML 提供一些窍门和建议。

这里也会包括一些 C++ 的窍门，像如何在字符串和整数之间进行转换。这和 TinyXML 本身并没有任何关系，但是它会对你的工程有所帮助，所以我把它写了进来。

如果你不知道 C++ 的基本概念，那么本指南对你没有任何用处。同样，如果你不知道 DOM 是什么，先在别的地方学习一下吧。

开始之前

一些 XML 数据集/文件将会被用到：

example1.xml:

```
<?xml version="1.0" ?>
<Hello>World</Hello>
```

example2.xml:

```
<?xml version="1.0" ?>
<poetry>
	<verse>
		Alas
		  Great World
			Alas (again)
	</verse>
</poetry>
```

example3.xml:

```
<?xml version="1.0" ?>
<shapes>
	<circle name="int-based" x="20" y="30" r="50" />
	<point name="float-based" x="3.5" y="52.1" />
</shapes>
```

example4.xml:

```
<?xml version="1.0" ?>
<MyApp>
    <!-- Settings for MyApp -->
    <Messages>
        <Welcome>Welcome to MyApp</Welcome>
        <Farewell>Thank you for using MyApp</Farewell>
    </Messages>
    <Windows>
        <Window name="MainFrame" x="5" y="15" w="400" h="250" />
    </Windows>
    <Connection ip="192.168.0.1" timeout="123.456000" />
</MyApp>
```

开始起步

从文件加载 XML

将一个文件加载到 TinyXML DOM 中的最简单方法：

```
TiXmlDocument doc( "demo.xml" );
doc.LoadFile();
```

下面是一个更实际的用法。它会加载文件并在标准输出中显示文件内容：

```
// load the named file and dump its structure to STDOUT

void dump_to_stdout(const char* pFilename)
{
	TiXmlDocument doc(pFilename);
	bool loadOkay = doc.LoadFile();
	if (loadOkay)
	{
		printf("\n%s:\n", pFilename);
		dump_to_stdout( &doc ); // defined later in the tutorial

	}
	else
	{
		printf("Failed to load file \"%s\"\n", pFilename);
	}
}
```

这是在 main() 函数中使用这个函数的简单示例：

```
int main(void)
{
	dump_to_stdout("example1.xml");
	return 0;
}
```

Example 1 的 XML 是：

```
<?xml version="1.0" ?>
<Hello>World</Hello>
```

运行这个程序就可以将 XML 文件显示在控制台或 DOS 窗口中：

```
DOCUMENT
+ DECLARATION
+ ELEMENT Hello
  + TEXT[World]
```

在本指南的后面会给出 dump_to_stdout() 函数的定义，它对你了解如何递归遍历一个 DOM 十分有用。

通过编程建立 XML 文档

下面的函数可以建立 Example 1 文档：

```
void build_simple_doc( )
{
	// Make xml: <?xml ..><Hello>World</Hello>

	TiXmlDocument doc;
	TiXmlDeclaration * decl = new TiXmlDeclaration( "1.0", "", "" );
	TiXmlElement * element = new TiXmlElement( "Hello" );
	TiXmlText * text = new TiXmlText( "World" );
	element->LinkEndChild( text );
	doc.LinkEndChild( decl );
	doc.LinkEndChild( element );
	doc.SaveFile( "madeByHand.xml" );
}
```

可以通过下面的调用来加载文件并将它显示在控制台上：

```
dump_to_stdout("madeByHand.xml"); // this func defined later in the tutorial
```

你会发现它和 Example 1 是完全一样的：

```
madeByHand.xml:
Document
+ Declaration
+ Element [Hello]
  + Text: [World]
```

下面这段代码通过节点的建立和连接的不同顺序生成一个完全相同的 XML DOM ：

```
void write_simple_doc2( )
{
	// same as write_simple_doc1 but add each node

	// as early as possible into the tree.


	TiXmlDocument doc;
	TiXmlDeclaration * decl = new TiXmlDeclaration( "1.0", "", "" );
	doc.LinkEndChild( decl );

	TiXmlElement * element = new TiXmlElement( "Hello" );
	doc.LinkEndChild( element );

	TiXmlText * text = new TiXmlText( "World" );
	element->LinkEndChild( text );

	doc.SaveFile( "madeByHand2.xml" );
}
```

这两段代码生成的是同一段 XML ，如下：

```
<?xml version="1.0" ?>
<Hello>World</Hello>
```

也就是这种结构形式：

```
DOCUMENT
+ DECLARATION
+ ELEMENT Hello
  + TEXT[World]
```

属性

很简单就可以给一个已经存在的节点设置属性：

```
window = new TiXmlElement( "Demo" );
window->SetAttribute("name", "Circle");
window->SetAttribute("x", 5);
window->SetAttribute("y", 15);
window->SetDoubleAttribute("radius", 3.14159);
```

你也可以使用 TiXmlAttribute 对象来做这件事。

下面这段代码给出了一种（并非仅此一种）如何得到一个元素的所有属性，然后打印出名字和字符串值的方法，并且，如果这些值可以转换成整形或浮点型，那么把他们也打印出来：

```
// print all attributes of pElement.

// returns the number of attributes printed

int dump_attribs_to_stdout(TiXmlElement* pElement, unsigned int indent)
{
    if ( !pElement ) return 0;

    TiXmlAttribute* pAttrib=pElement->FirstAttribute();
    int i=0;
    int ival;
    double dval;
    const char* pIndent=getIndent(indent);
    printf("\n");
    while (pAttrib)
    {
        printf( "%s%s: value=[%s]", pIndent, pAttrib->Name(), pAttrib->Value());

        if (pAttrib->QueryIntValue(&ival)==TIXML_SUCCESS) printf( " int=%d", ival);
        if (pAttrib->QueryDoubleValue(&dval)==TIXML_SUCCESS) printf( " d=%1.1f", dval);
        printf( "\n" );
        i++;
        pAttrib=pAttrib->Next();
    }
    return i;
}
```

将文档写入文件

将建立好的 DOM 写入一个文件很简单：

```
doc.SaveFile( saveFilename );
```

回想一下 example 4：

```
<?xml version="1.0" ?>
<MyApp>
    <!-- Settings for MyApp -->
    <Messages>
        <Welcome>Welcome to MyApp</Welcome>
        <Farewell>Thank you for using MyApp</Farewell>
    </Messages>
    <Windows>
```

```
    <Windows>
        <Window name="MainFrame" x="5" y="15" w="400" h="250" />
    </Windows>
    <Connection ip="192.168.0.1" timeout="123.456000" />
</MyApp>
```

下面的函数用来建立这个 DOM 并把它写进一份名为 "appsettings.xml" 的文件中：

```
void write_app_settings_doc( )
{
	TiXmlDocument doc;

	TiXmlElement* msg;
	TiXmlDeclaration* decl = new TiXmlDeclaration( "1.0", "", "" );
	doc.LinkEndChild( decl );

	TiXmlElement * root = new TiXmlElement( "MyApp" );
	doc.LinkEndChild( root );

	TiXmlComment * comment = new TiXmlComment();
	comment->SetValue(" Settings for MyApp " );
	root->LinkEndChild( comment );

	TiXmlElement * msgs = new TiXmlElement( "Messages" );
	root->LinkEndChild( msgs );

	msg = new TiXmlElement( "Welcome" );
	msg->LinkEndChild( new TiXmlText( "Welcome to MyApp" ));
	msgs->LinkEndChild( msg );

	msg = new TiXmlElement( "Farewell" );
	msg->LinkEndChild( new TiXmlText( "Thank you for using MyApp" ));
	msgs->LinkEndChild( msg );

	TiXmlElement * windows = new TiXmlElement( "Windows" );
	root->LinkEndChild( windows );

	TiXmlElement * window;
	window = new TiXmlElement( "Window" );
	windows->LinkEndChild( window );
	window->SetAttribute("name", "MainFrame");
	window->SetAttribute("x", 5);
	window->SetAttribute("y", 15);
	window->SetAttribute("w", 400);
	window->SetAttribute("h", 250);

	TiXmlElement * cxn = new TiXmlElement( "Connection" );
	root->LinkEndChild( cxn );
	cxn->SetAttribute("ip", "192.168.0.1");
	cxn->SetDoubleAttribute("timeout", 123.456); // floating point attrib


	dump_to_stdout( &doc );
	doc.SaveFile( "appsettings.xml" );
}
```

通过 dump_to_stdout() 函数可以显示这个结构：

```
Document
+ Declaration
+ Element [MyApp]
 (No attributes)
  + Comment: [ Settings for MyApp ]
  + Element [Messages]
 (No attributes)
    + Element [Welcome]

 (No attributes)
      + Text: [Welcome to MyApp]
    + Element [Farewell]
 (No attributes)
      + Text: [Thank you for using MyApp]
  + Element [Windows]
 (No attributes)
    + Element [Window]
      + name: value=[MainFrame]
      + x: value=[5] int=5 d=5.0
      + y: value=[15] int=15 d=15.0
      + w: value=[400] int=400 d=400.0
      + h: value=[250] int=250 d=250.0
      5 attributes
  + Element [Connection]
    + ip: value=[192.168.0.1] int=192 d=192.2
    + timeout: value=[123.456000] int=123 d=123.5
    2 attributes
```

很高兴在默认的情况下， TinyXml 用其他的 API 所谓的"优美"格式来写 XML ，它修改元素文字的空白，然后用嵌套层次的方式显示这棵树。

我没有注意到在写文件的时候是否有办法关掉缩排，但是这肯定很容易。

[Lee：在 STL 模式下很容易，只要使用 cout << myDoc 就可以了。非 STL 模式通常是"优美"格式的。加入一个"开关"会是一个不错的特性，并且已经这么做了。]

XML 与 C++ 对象之间的转换

介绍

这个示例假定你正要加载你的应用设置并把它们保存在 XML 文件中，就像 example4.xml 那样。

有许多办法可以完成这件事。例如，看一看 TinyBind 这个工程，你可以在这里找到它：http://sourceforge.net/projects/tinybind

本节给出一个使用 XML 来加载并保存一个基本的对象结构的浅显易懂的方法。

建立你的对象类

先从下面的这些基类入手：

```
#include <string>
#include <map>
using namespace std;

typedef std::map<std::string,std::string> MessageMap;

// a basic window abstraction - demo purposes only

class WindowSettings
{
public:
	int x,y,w,h;
	string name;

	WindowSettings()
		: x(0), y(0), w(100), h(100), name("Untitled")
	{
	}

	WindowSettings(int x, int y, int w, int h, const string& name)
	{
		this->x=x;
		this->y=y;
		this->w=w;
		this->h=h;
		this->name=name;
	}
};

class ConnectionSettings
{
public:
	string ip;
	double timeout;
};

class AppSettings
{
public:
	string m_name;
	MessageMap m_messages;
	list<WindowSettings> m_windows;
	ConnectionSettings m_connection;

	AppSettings() {}

	void save(const char* pFilename);
	void load(const char* pFilename);

	// just to show how to do it

	void setDemoValues()
	{
		m_name="MyApp";
		m_messages.clear();
		m_messages["Welcome"]="Welcome to "+m_name;
		m_messages["Farewell"]="Thank you for using "+m_name;
		m_windows.clear();
		m_windows.push_back(WindowSettings(15,15,400,250,"Main"));
		m_connection.ip="Unknown";
		m_connection.timeout=123.456;
	}
};
```

这是一个简化的 main() 函数，它演示了如何建立一个默认设置的树的对象，保存它以及重新载入它：

```
int main(void)
{
	AppSettings settings;

	settings.save("appsettings2.xml");
	settings.load("appsettings2.xml");
	return 0;
```

```
}
```

下面这个 main() 函数演示了如何建立，修改，保存，还有加载一个结构体：

```
int main(void)
{
	// block: customise and save settings

	{
		AppSettings settings;
		settings.m_name="HitchHikerApp";
		settings.m_messages["Welcome"]="Don't Panic";
		settings.m_messages["Farewell"]="Thanks for all the fish";
		settings.m_windows.push_back(WindowSettings(15,25,300,250,"BookFrame"));
		settings.m_connection.ip="192.168.0.77";
		settings.m_connection.timeout=42.0;

		settings.save("appsettings2.xml");
	}

	// block: load settings

	{
		AppSettings settings;
		settings.load("appsettings2.xml");
		printf("%s: %s\n", settings.m_name.c_str(),
			settings.m_messages["Welcome"].c_str());
		WindowSettings & w=settings.m_windows.front();
		printf("%s: Show window '%s' at %d,%d (%d x %d)\n",
			settings.m_name.c_str(), w.name.c_str(), w.x, w.y, w.w, w.h);
		printf("%s: %s\n", settings.m_name.c_str(), settings.m_messages["Farewell"].c_str());
	}
	return 0;
}
```

当 save() 函数和 load() 函数完成后（在下面），运行这个 main() 函数，在控制台会显示：

```
HitchHikerApp: Don't Panic
HitchHikerApp: Show window 'BookFrame' at 15,25 (300 x 100)
HitchHikerApp: Thanks for all the fish
```

编码为 XML

有许多不同的方法来解决如何将它（一个结构体）保存在文件中。比如：

```
void AppSettings::save(const char* pFilename)
{
	TiXmlDocument doc;
	TiXmlElement* msg;
	TiXmlComment * comment;
	string s;
	 TiXmlDeclaration* decl = new TiXmlDeclaration( "1.0", "", "" );
	doc.LinkEndChild( decl );

	TiXmlElement * root = new TiXmlElement(m_name.c_str());
	doc.LinkEndChild( root );

	comment = new TiXmlComment();
	s=" Settings for "+m_name+" ";
	comment->SetValue(s.c_str());
	root->LinkEndChild( comment );

	// block: messages

	{
		MessageMap::iterator iter;

		TiXmlElement * msgs = new TiXmlElement( "Messages" );
		root->LinkEndChild( msgs );

		for (iter=m_messages.begin(); iter != m_messages.end(); iter++)
		{
			const string & key=(*iter).first;
			const string & value=(*iter).second;
			msg = new TiXmlElement(key.c_str());
			msg->LinkEndChild( new TiXmlText(value.c_str()));
			msgs->LinkEndChild( msg );
		}
	}

	// block: windows

	{
		TiXmlElement * windowsNode = new TiXmlElement( "Windows" );
		root->LinkEndChild( windowsNode );

		list<WindowSettings>::iterator iter;
```

```
		for (iter=m_windows.begin(); iter != m_windows.end(); iter++)
		{
			const WindowSettings& w=*iter;

			TiXmlElement * window;
			window = new TiXmlElement( "Window" );
			windowsNode->LinkEndChild( window );
			window->SetAttribute("name", w.name.c_str());
			window->SetAttribute("x", w.x);
			window->SetAttribute("y", w.y);
			window->SetAttribute("w", w.w);
			window->SetAttribute("h", w.h);
		}
	}

	// block: connection

	{
		TiXmlElement * cxn = new TiXmlElement( "Connection" );
		root->LinkEndChild( cxn );
		cxn->SetAttribute("ip", m_connection.ip.c_str());
		cxn->SetDoubleAttribute("timeout", m_connection.timeout);
	}

	doc.SaveFile(pFilename);
}
```

运行这个修改过的 main() 函数会生成如下的文件：

```
<?xml version="1.0" ?>
<HitchHikerApp>
    <!-- Settings for HitchHikerApp -->
    <Messages>
        <Farewell>Thanks for all the fish</Farewell>
        <Welcome>Don&apos;t Panic</Welcome>
    </Messages>
    <Windows>
        <Window name="BookFrame" x="15" y="25" w="300" h="250" />
    </Windows>
    <Connection ip="192.168.0.77" timeout="42.000000" />
</HitchHikerApp>
```

从 XML 解码

就像给一个对象编码一样，也有许多办法可以把 XML 解码为你自己的 C++ 对象结构。下面的方法使用了 TiXmlHandles 类。

```
void AppSettings::load(const char* pFilename)
{
    TiXmlDocument doc(pFilename);
    if (!doc.LoadFile()) return;

    TiXmlHandle hDoc(&doc);
    TiXmlElement* pElem;
    TiXmlHandle hRoot(0);

    // block: name

    {
        pElem=hDoc.FirstChildElement().Element();
        // should always have a valid root but handle gracefully if it does

        if (!pElem) return;
        m_name=pElem->Value();

        // save this for later

        hRoot=TiXmlHandle(pElem);
    }

    // block: string table

    {
        m_messages.clear(); // trash existing table


        pElem=hRoot.FirstChild( "Messages" ).FirstChild().Element();
        for( pElem; pElem; pElem=pElem->NextSiblingElement())
        {
            const char *pKey=pElem->Value();
            const char *pText=pElem->GetText();
            if (pKey && pText)
            {
                m_messages[pKey]=pText;
            }
        }
```

```
    }

    // block: windows

    {
        m_windows.clear(); // trash existing list


        TiXmlElement* pWindowNode=hRoot.FirstChild( "Windows" ).FirstChild().Element();
        for( pWindowNode; pWindowNode; pWindowNode=pWindowNode->NextSiblingElement())
        {
            WindowSettings w;
            const char *pName=pWindowNode->Attribute("name");
            if (pName) w.name=pName;

            pWindowNode->QueryIntAttribute("x", &w.x); // If this fails, original value is left as-is

            pWindowNode->QueryIntAttribute("y", &w.y);
            pWindowNode->QueryIntAttribute("w", &w.w);
            pWindowNode->QueryIntAttribute("hh", &w.h);

            m_windows.push_back(w);
        }
    }

    // block: connection

    {
        pElem=hRoot.FirstChild("Connection").Element();
        if (pElem)
        {
            m_connection.ip=pElem->Attribute("ip");
            pElem->QueryDoubleAttribute("timeout",&m_connection.timeout);
        }
    }
}
```

dump_to_stdout() 函数的完整代码

下面是一份复制粘贴过来的演示程序：加载任意一份 XML 文件，然后使用上面所说的递归遍历的方式将 XML 结构输出到标准输出中。

```
// tutorial demo program

#include "stdafx.h"
#include "tinyxml.h"

// ----------------------------------------------------------------------

// STDOUT dump and indenting utility functions

// ----------------------------------------------------------------------

const unsigned int NUM_INDENTS_PER_SPACE=2;

const char * getIndent( unsigned int numIndents )
{
    static const char * pINDENT="                                      + ";
    static const unsigned int LENGTH=strlen( pINDENT );
    unsigned int n=numIndents*NUM_INDENTS_PER_SPACE;
    if ( n > LENGTH ) n = LENGTH;

    return &pINDENT[ LENGTH-n ];
}

// same as getIndent but no "+" at the end

const char * getIndentAlt( unsigned int numIndents )
{
    static const char * pINDENT="                                      ";
    static const unsigned int LENGTH=strlen( pINDENT );
    unsigned int n=numIndents*NUM_INDENTS_PER_SPACE;
    if ( n > LENGTH ) n = LENGTH;

    return &pINDENT[ LENGTH-n ];
}

int dump_attribs_to_stdout(TiXmlElement* pElement, unsigned int indent)
{
    if ( !pElement ) return 0;

    TiXmlAttribute* pAttrib=pElement->FirstAttribute();
    int i=0;
    int ival;
    double dval;
    const char* pIndent=getIndent(indent);
    printf("\n");
    while (pAttrib)
```

```
    {
        printf( "%s%s: value=[%s]", pIndent, pAttrib->Name(), pAttrib->Value());

        if (pAttrib->QueryIntValue(&ival)==TIXML_SUCCESS) printf( " int=%d", ival);
        if (pAttrib->QueryDoubleValue(&dval)==TIXML_SUCCESS) printf( " d=%1.1f", dval);
        printf( "\n" );
        i++;
        pAttrib=pAttrib->Next();
    }
    return i;
}

void dump_to_stdout( TiXmlNode* pParent, unsigned int indent = 0 )
{
    if ( !pParent ) return;

    TiXmlNode* pChild;
    TiXmlText* pText;
    int t = pParent->Type();
    printf( "%s", getIndent(indent));
    int num;

    switch ( t )
    {
    case TiXmlNode::DOCUMENT:
        printf( "Document" );
        break;

    case TiXmlNode::ELEMENT:
        printf( "Element [%s]", pParent->Value() );
        num=dump_attribs_to_stdout(pParent->ToElement(), indent+1);
        switch(num)
        {
            case 0: printf( " (No attributes)"); break;
            case 1: printf( "%s1 attribute", getIndentAlt(indent)); break;
            default: printf( "%s%d attributes", getIndentAlt(indent), num); break;
        }
        break;

    case TiXmlNode::COMMENT:
        printf( "Comment: [%s]", pParent->Value());
        break;

    case TiXmlNode::UNKNOWN:
        printf( "Unknown" );
        break;

    case TiXmlNode::TEXT:
        pText = pParent->ToText();
        printf( "Text: [%s]", pText->Value() );
        break;

    case TiXmlNode::DECLARATION:
        printf( "Declaration" );
        break;
    default:
        break;
    }
    printf( "\n" );
    for ( pChild = pParent->FirstChild(); pChild != 0; pChild = pChild->NextSibling())
    {
        dump_to_stdout( pChild, indent+1 );
    }
}

// load the named file and dump its structure to STDOUT

void dump_to_stdout(const char* pFilename)
{
    TiXmlDocument doc(pFilename);
    bool loadOkay = doc.LoadFile();
    if (loadOkay)
    {
        printf("\n%s:\n", pFilename);
        dump_to_stdout( &doc ); // defined later in the tutorial

    }
    else
    {
        printf("Failed to load file \"%s\"\n", pFilename);
    }
}

// ----------------------------------------------------------------------

// main() for printing files named on the command line
```

```
// ----------------------------------------------------------------------

int main(int argc, char* argv[])
{
    for (int i=1; i<argc; i++)
    {
        dump_to_stdout(argv[i]);
    }
    return 0;
}
```

在命令行或 DOS 窗口中这样运行它，比如：

```
C:\dev\tinyxml> Debug\tinyxml_1.exe example1.xml

example1.xml:
Document
+ Declaration
+ Element [Hello]
  (No attributes)
   + Text: [World]
```

作者以及变动

- 由 *Ellers* 于*2005*年*4*月~*6*月编写
- 由 *Lee Thomason* 于*2005*年*9*月整理入 *doc* 文档中
- *2005*年*10*月由 *Ellers* 进行更新

因翻译仓促，水平有限，文中如有疏漏或错误，请不吝赐教。联系作者：

Email / MSN： gao.hongda@gmail.com

blog： http://LoveRene.cublog.cn/