如果你的操作系统是中文系统，

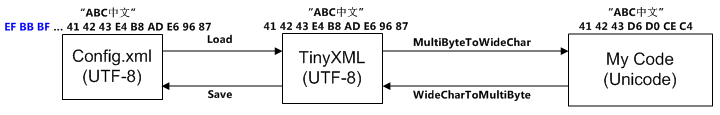
1.你新建xml文档，默认是gb2312编码的。  
<?xml version="1.0" encoding="gb2312" ?> 绝对可以显示中文。

2，你是改的别人的xml文档，但之前该xml文档是UTF-8/16编码的，  
<?xml version="1.0" encoding="gb2312" ?>  
不能显示中文，因为该文档不是gb2312编码的，虽然你强制申明它要用gb2312编码转换，但gb2312编码机制处理不了UTF-8/16的编码。必须用UTF-8/16编码机制转换。也就是  
<?xml version="1.0" encoding="UTF-8" ?> or 干脆不声明encoding属性。

PS：用windows自带的记事本的另存为功能，可以进行编码转换。

之前陈超给的xml模板是utf-8编码的。通过用记事本转化后，使用模板的文件也是可以写入并显示中文的。

TinyXML是完全支持UTF-8的，可以尝试load一个UTF-8格式的xml文件，完全没有问题，那现在问题是load了一个UTF-8文件后，文件中的字符串，在TiXmlDocument对象中，到底是什么格式？TinyXML在load它之后会不会直接把UTF-8格式转为Unicode格式？——不会，为什么？因为TinyXML在编写的时候是把自己定位为一个比较单纯的C++ Lib，它没有使用任何Windows相关的API，包括很重要的两个编码转换函数WideCharToMultiByte和MultiByteToWideChar，它都没用，对不同平台，编码转换函数可能是不同的，因此TinyXML不会画蛇添足地提供这个转换，也就是说，如果你的xml文件（UTF-8格式）中如果有中文的话，用TinyXML加载进来之后，依旧是UTF-8的，不经过转换直接print出来的话，就是乱码了，所以要经过一个UTF-8到Unicode的转换，转换代码我这里就不写了。再说一下，TinyXML声称对UTF-8的支持表示它认识UTF-8的xml文件，但并不表示它自动对UTF-8的文本进行转换。

好，再回到保存的问题上，其实你差不多想到了，应该怎么做，因为整个TinyXML其实并不计较内存中的文本究竟是什么内容，它都可以把文本当做是UTF-8（当然了，实际上把文本转为UTF-8然后交给TinyXML的工作是我们的程序去完成），那我们可不可以说保存的xml文件其实就已经是UTF-8格式的？——答案是肯定的。而UTF-8的格式标志其实就是文件头的三个字节：0xEF，0xBB，0xBF。有了这个头，用UltraEdit打开这个xml，UltraEdit就把它认为是一个UTF-8的文本文件了。  


这个改动非常简单，稍微浏览下TinyXML的代码就知道怎么改了，我下载的TinyXML是2.6.1版本的（我没记错的话），在tinyxml.cpp文件中搜索“useMicrosoftBOM”，把“useMicrosoftBOM = false;”这个语句改为“useMicrosoftBOM = true;”就可以了，好像一共才三处地方，十分简单，代码一看就懂，不多说了。

说说我用tinyXML遇到的一个小问题： 用tinyXML的SaveFile方法生成的XML文件中有中文字符，初始化时声明的encodeing为utf-8，用IE打开就报错，但是记事本打开不会显示乱码。

    在网上搜了半天。这个牵涉到对于XML文件编码方式的确认，就是说怎么认定某个XML文件的编码方式，可以参见这个文档<http://www.cnblogs.com/azol/articles/1137035.html>，简单来说就是先看BOM，没有就看文件里写的encodeing的值，再没有就默认utf-8。所以我遇到的IE打开报错的问题就是保存时没有写BOM（tinyXML默认是不写BOM的，useMicrosoftBOM在LoadFile时被置为false），而声明了encodeing为utf-8，则保存的这个XML文件的编码格式就是utf-8。所以IE解析的时候认为XML文件是utf-8编码的，就把中文按UTF-8来解释，而实际中文的编码是GB2312（因为我在写入元素的属性或文本时没有做编码转换，我使用的是多字节字符集），自然就报错了。而为什么记事本打开不是乱码呢，那就又要涉及到记事本程序对于文本文件的编码认定，参见<http://kuyur.info/blog/archives/618>，简单来说就是先判断BOM，没有再依据UTF-8字符串检测算法判断是否UTF-8，如果不是则最后默认是ANSI。所以以记事本打开SaveFile方法生成的XML文件时，首先是没有写BOM的，而检测UTF-8字符串也检测不到（我设置元素文本或属性时并没有将中文转为UTF-8格式），那最后就默认的认定为ANSI（这个和GB2312什么关系搞不太清楚），总之记事本认为打开的文件编码不是UTF-8，自然就不会乱码了。

终于对编码有一定的认识，一说编码，就tmd的恶心。

关于编码ansi、GB2312、unicode与utf-8的区别

先做一个小小的试验：

在一个文件夹里，把一个txt文本（文本里包含“今天的天气非常好”这句话）分别另存为ansi、unicode、utf-8这三种编码的txt文件。然后，在该文件夹上点击右键，选择“搜索(E)…”。

搜索“天气”二字，可以搜索出ansi和unicode这两种编码的txt文件，搜索不出utf-8编码的文件。

原因：

1.中文操作系统默认ansi编码，生成的txt文件默认为ansi编码，所以，可以搜索出来。

2.unicode是国际通用编码，所以，可以搜索出来。

3.utf-8编码是unicode编码在网络之间（主要是网页）传输时的一种“变通”和“桥梁”编码。utf-8在网络之间传输时可以节约数据量。所以，使用操作系统无法搜索出txt文本。

按照utf-8创始人的愿望：

端（unicode）——传输（utf-8）——端（unicode）

但是，后来，许多网站开发者在开发网页时直接使用utf-8编码。

端（utf-8）——传输（utf-8）——端（utf-8）

所以，在浏览器上看到的编码是：unicode（utf-8）。正因为在浏览器上这么并列地列出unicode（utf-8），造成许多网友（甚至不少程序员）误认为unicode=utf-8。其实，按照utf-8创始人的原意，在开发网页时使用utf-8编码是错误的做法，并且，早期的浏览器也不支持解析utf-8编码。但是，众人的力量是巨大的，微软不得不“趋炎附势”，在浏览器上支持解析utf-8编码。

问题是：utf-8编码影响了网站开发者，或者说，网站开发者“扩展”了utf-8编码的使用范围。但是，网站开发者仍然无法影响各类文档的开发者，所以，word文档和一些国际通用的文档仍然使用unicode编码而不使用utf-8编码。

比如：“严”的Unicode码是4E25，UTF-8编码是E4B8A5，两者是不一样的。

在中文和日文操作系统里生成的（txt和xml）文件的编码虽然都是ansi，但是，在简体中文系统下，ansi 编码代表 GB2312 编码，在日文操作系统下，ansi 编码代表 JIS 编码。不同 ansi 编码之间互不兼容，当信息在国际间交流时，无法将属于两种语言的文字，存储在同一段 ansi 编码的文本中。

结论：国际文档（txt和xml）使用unicode编码是正宗做法；操作系统和浏览器都能够“理解”unicode编码。浏览器“迫于压力”才“理解”utf-8编码。但是，操作系统有时只认unicode编码。

Unicode与Unicode big endian的区别：你吃鸡蛋时先吃小头还是先吃大头？Unicode与Unicode big endian的区别就是在编码时小头优先与大头优先的区别。“随波逐流”使用Unicode就OK了。

我（不是程序员）这几年一直因为编码问题，感到非常困惑，查了许多资料，在国际文档的实际应用中也遇到过许多问题，所以，“感性”地总结了上述观点，不一定准确（或者说，不一定正确）。

**unicode和utf-8是什么关系**

Unicode的最初目标，是用1个16位的编码来为超过65000个字符提供映射。但这还不够，它不能覆盖全部历史上的文字，也不能解决传输的问题(implantation head-ache's)，尤其在那些基于网络的应用中。已有的软件必须做大量的工作来实现16位的数据。  
　　因此，Unicode用一些基本的保留字符制定了三套编码方式。它们分别是UTF-8,UTF-16和UTF-32。正如名字所示，在UTF－8中，字符是以8位序列来编码的，用一个或几个字节来表示一个字符。这种方式的最大好处，是UTF－8保留了ASCII字符的编码做为它的一部分，例如，在UTF－8和ASCII中，“A”的编码都是0x41. UTF－16和UTF－32分别是Unicode的16位和32位编码方式。考虑到最初的目的，通常说的Unicode就是指UTF-16。在讨论Unicode时，搞清楚哪种编码方式非常重要。