<https://blog.csdn.net/taiyang1987912/article/details/46695245>

# 一、简介

        QtXml模块提供了一个读写XML文件的流，解析方法包含DOM和SAX。DOM（Document ObjectModel）：将XML文件表示成一棵树，便于随机访问其中的节点，但消耗内存相对多一些。SAX（Simple APIfor XML）：一种事件驱动的XML API，接近于底层，速度较快，但不便于随机访问任意节点。      
       使用XML模块，在.pro文件中添加QT += xml，并加如相应的头文件#include <QDomDocument>或者#include <QXmlStreamReader>。  
       分析的xml文件：

**[html]** [view plain](https://blog.csdn.net/taiyang1987912/article/details/46695245) [copy](https://blog.csdn.net/taiyang1987912/article/details/46695245)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<COMMAND>**
3. **<OBJECT>**USER**</OBJECT>**
4. **<ACTION>**LOGIN**</ACTION>**
5. **<DATA>**
6. **<USER** NAME="root" PASSWORD="123456"**/>**
7. **</DATA>**
8. **</COMMAND>**

# 二、详解

## 1、QXmlStreamReader

（1）streamparsexml.h

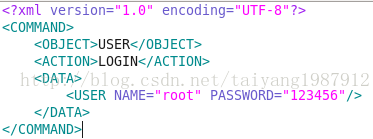
**[html]** [view plain](https://blog.csdn.net/taiyang1987912/article/details/46695245) [copy](https://blog.csdn.net/taiyang1987912/article/details/46695245)

1. #ifndef STREAMPARSEXML\_H
2. #define STREAMPARSEXML\_H
3. #include **<QXmlStreamWriter>**
4. #include **<QXmlStreamReader>**
5. #include **<QFile>**
6. #include **<QMessageBox>**
8. class StreamParseXml
9. {
10. public:
11. StreamParseXml();
12. ~StreamParseXml();
13. int writeXml();
14. int readXml();
16. private:
17. void parseUserInformation();
18. QString getValue(const QString &name);
19. QString getAttribute(const QString &name);
21. private:
22. QString fileName;
23. QXmlStreamReader \*reader;
24. };
26. #endif // STREAMPARSEXML\_H

（2）streamparsexml.cpp

**[html]** [view plain](https://blog.csdn.net/taiyang1987912/article/details/46695245) [copy](https://blog.csdn.net/taiyang1987912/article/details/46695245)

1. #include **<QDebug>**
2. #include "streamparsexml.h"
4. StreamParseXml::StreamParseXml()
5. {
6. fileName = "streamparse.xml";
7. }
9. StreamParseXml::~StreamParseXml()
10. {
12. }
14. int StreamParseXml::writeXml()
15. {
16. QFile file(fileName);
17. if(file.open(QIODevice::WriteOnly | QIODevice::Text)) {
18. QXmlStreamWriter writer(&file);
19. writer.setAutoFormatting(true);
20. writer.writeStartDocument();
21. writer.writeStartElement("COMMAND");
22. writer.writeTextElement("OBJECT", "USER");
23. writer.writeTextElement("ACTION", "LOGIN");
24. writer.writeStartElement("DATA");
25. writer.writeStartElement("USER");
26. writer.writeAttribute("NAME", "root");
27. writer.writeAttribute("PASSWORD", "123456");
28. writer.writeEndElement();
29. writer.writeEndElement();
30. writer.writeEndElement();
31. file.close();
32. }
33. return 0;
34. }
36. int StreamParseXml::readXml()
37. {
38. if(fileName.isEmpty()) return -2;
39. QFile \*file = new QFile(fileName);
40. if(!file-**>**open(QFile::ReadOnly | QFile::Text)) {
41. QMessageBox::information(NULL, QString("title"), QString("open error!"));
42. return -1;
43. }
44. reader = new QXmlStreamReader(file);
45. while(!reader-**>**atEnd() && !reader-**>**hasError()) {
46. QXmlStreamReader::TokenType token = reader-**>**readNext();
47. if(token == QXmlStreamReader::StartDocument) {
48. continue;
49. }
50. if (reader-**>**isStartElement() && reader-**>**name() == "OBJECT") {
51. QString elementText = reader-**>**readElementText();
52. if (elementText == "USER") {
53. parseUserInformation();
54. break;
55. }
56. }
57. }
58. if (reader-**>**hasError()) {
59. qDebug() **<<** **reader->**errorString();
60. //QMessageBox::information(NULL, QString("parseXML"), reader-**>**errorString());
61. }
62. reader-**>**clear();
63. delete reader;
64. reader = NULL;
65. return 0;
66. }
68. void StreamParseXml::parseUserInformation()
69. {
70. QString elementString = getValue("ACTION");
71. if (elementString == "LOGIN") {
72. while(!reader-**>**atEnd()) {
73. reader-**>**readNext();
74. if (reader-**>**name() == "USER") {
75. QXmlStreamAttributes attributes = reader-**>**attributes();
76. if(attributes.hasAttribute("NAME")) {
77. qDebug() **<<** "USER=" **<<** **attributes.value**("NAME").toString();
78. }
79. if(attributes.hasAttribute("PASSWORD")) {
80. qDebug() **<<** "PASSWORD=" **<<** **attributes.value**("PASSWORD").toString();
81. }
82. }
83. }
84. }
85. }
86. QString StreamParseXml::getValue(const QString &name)
87. {
88. while(!reader-**>**atEnd()) {
89. reader-**>**readNext();
90. if (reader-**>**isStartElement() && reader-**>**name() == name) {
91. return reader-**>**readElementText();
92. }
93. }
94. return "";
95. }

（3）运行  
https://img-blog.csdn.net/20150630162337991?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdGFpeWFuZzE5ODc5MTI=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

## 2、QDomDocument

        QDomDocument类代表整个的XML文件。概念上讲：它是文档树的根节点，并提供了文档数据的基本访问方法。由于元素、文本节点、注释、指令执行等等不可能脱离一个文档的上下文，所以文档类也包含了需要用来创建这些对象的工厂方法。被创建的节点对象有一个ownerDocument()函数，它将对象与对象常见的文档上下文环境关联起来。DOM类中最常使用的是QDomNode、QDomDocument、QDomElement和QDomText。

        解析后的XML文件在内部是通过一个对象树来表示的，对象树可以使用各种QDom类进行访问。所有的QDom类只引用内部树上的对象。一旦最后一个DOM树的QDom对象和QDocument本身被删除掉时，DOM树上的所有内部对象会被删除掉。元素、文本节点等的创建是通过使用类提供的各种工厂方法完成的。使用QDom类的缺省构造函数只会生成空的对象，这些空的对象不能操作，也不能写入到文档中。

        QDomDocument::setContent()完成XML文档的设置，他从QFile对象中读取XML数据并检测XML文档的编码。setContent()有几种重载形式，可以分别从QByteArray、QString、QIODevice、QXmlInputSource中读取XML数据。  
（1）domdocument.h

**[html]** [view plain](https://blog.csdn.net/taiyang1987912/article/details/46695245) [copy](https://blog.csdn.net/taiyang1987912/article/details/46695245)

1. #ifndef DOMDOCUMENT\_H
2. #define DOMDOCUMENT\_H
3. #include **<QDomDocument>**
4. #include **<QFile>**
5. #include **<QTextStream>**
7. class DomDocument
8. {
9. public:
10. DomDocument();
11. ~DomDocument();
12. int writeXml();
13. int readXml();
14. int readXml2();
16. private:
17. QString fileName;
18. };
20. #endif // DOMDOCUMENT\_H

（2）domdocument.cpp

**[html]** [view plain](https://blog.csdn.net/taiyang1987912/article/details/46695245) [copy](https://blog.csdn.net/taiyang1987912/article/details/46695245)

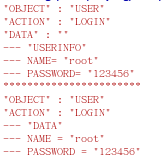
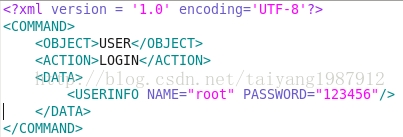
1. #include **<QDebug>**
2. #include "domdocument.h"
4. DomDocument::DomDocument()
5. {
6. fileName = "domparse.xml";
7. }
9. DomDocument::~DomDocument()
10. {
12. }
14. int DomDocument::writeXml()
15. {
16. QFile file(fileName);
17. if (!file.open(QIODevice::WriteOnly | QIODevice::Truncate))
18. return -2;
19. QTextStream out(&file);
20. QDomDocument doc;
21. QDomText text;
22. QDomElement element;
23. QDomAttr attr;
24. QDomProcessingInstruction instruction;
25. instruction = doc.createProcessingInstruction( "xml", "version = \'1.0\' encoding=\'UTF-8\'" );
26. doc.appendChild( instruction );
28. QDomElement root = doc.createElement( "COMMAND" );
29. doc.appendChild(root);
30. element = doc.createElement( "OBJECT" );
31. text = doc.createTextNode( "USER" );
32. element.appendChild(text);
33. root.appendChild(element);
35. element = doc.createElement( "ACTION" );
36. text = doc.createTextNode( "LOGIN" );
37. element.appendChild(text);
38. root.appendChild(element);
40. element = doc.createElement( "DATA" );
41. root.appendChild(element);
43. QDomElement userElement = doc.createElement( "USERINFO" );
44. attr = doc.createAttribute( "NAME" );
45. attr.setValue("root");
46. userElement.setAttributeNode(attr);
47. attr = doc.createAttribute( "PASSWORD" );
48. attr.setValue("123456");
49. userElement.setAttributeNode(attr);
50. element.appendChild(userElement);
52. doc.save(out, 4);       //each line space of file is 4
53. return 0;
54. }
56. int DomDocument::readXml()
57. {
58. QDomDocument doc;
59. QFile file(fileName);
60. QString error = "";
61. int row = 0, column = 0;
62. if (!file.open(QIODevice::ReadOnly)) return -2;
64. if(!doc.setContent(&file, false, &error, &row, &column)){
65. qDebug() **<<** "parse file failed:" **<<** **row** **<<** "---" **<<** **column** **<<**":" **<<error**;
66. file.close();
67. return -1;
68. }
70. file.close();
71. QDomElement root = doc.documentElement();
72. QDomNode node = root.firstChild();
73. while(!node.isNull()) {
74. QDomElement element = node.toElement(); // try to convert the node to an element.
75. if(!element.isNull()) {
76. qDebug()**<<element.tagName**() **<<** ":" **<<** **element.text**();
77. QDomNode nodeson = element.firstChild();
78. while(!nodeson.isNull()) {
79. QDomElement elementson = nodeson.toElement();
80. if(!elementson.isNull()) {
81. qDebug()**<<** "---" **<<elementson.tagName**();
82. if (elementson.hasAttribute("NAME")) {
83. qDebug()**<<** "---" **<<** "NAME=" **<<** **elementson.attributeNode**("NAME").value();
84. }
85. if (elementson.hasAttribute("PASSWORD")) {
86. qDebug()**<<** "---" **<<** "PASSWORD=" **<<** **elementson.attributeNode**("PASSWORD").value();
87. }
88. }
89. nodeson = nodeson.nextSibling();
90. }
91. }
92. node = node.nextSibling();
93. }
94. return 0;
95. }
96. int DomDocument::readXml2()
97. {
98. QDomDocument doc;
99. QFile file(fileName);
100. QString error = "";
101. int row = 0, column = 0;
102. if (!file.open(QIODevice::ReadOnly)) return -2;
104. if(!doc.setContent(&file, false, &error, &row, &column)){
105. qDebug() **<<** "parse file failed:" **<<** **row** **<<** "---" **<<** **column** **<<**":" **<<error**;
106. file.close();
107. return -1;
108. }
110. file.close();
111. QDomElement root = doc.documentElement();
112. QDomNode node = root.firstChildElement();
113. while(!node.isNull()) {
114. QDomElement element = node.toElement(); // try to convert the node to an element.
115. if(!element.isNull()) {
116. if (element.tagName() == "DATA") {
117. qDebug()**<<** "---" **<<element.tagName**();
118. QDomNodeList list = element.childNodes();
119. for(int index = 0; index **<** **list.count**(); index++) {
120. QDomNode list\_node = list.item(index);
121. QDomElement list\_element = list\_node.toElement();
122. if (list\_element.hasAttribute("NAME")) {
123. qDebug()**<<** "---" **<<** "NAME =" **<<** **list\_element.attributeNode**("NAME").value();
124. }
125. if (list\_element.hasAttribute("PASSWORD")) {
126. qDebug()**<<** "---" **<<** "PASSWORD =" **<<** **list\_element.attributeNode**("PASSWORD").value();
127. }
128. }
129. }
130. else {
131. qDebug()**<<element.tagName**() **<<** ":" **<<** **element.text**();
132. }
133. }
134. node = node.nextSibling();
135. }
136. return 0;
137. }

（3）main.cpp

**[html]** [view plain](https://blog.csdn.net/taiyang1987912/article/details/46695245) [copy](https://blog.csdn.net/taiyang1987912/article/details/46695245)

1. #include **<QCoreApplication>**
2. #include **<QDebug>**
3. #include "streamparsexml.h"
4. #include "domdocument.h"
6. int main(int argc, char \*argv[])
7. {
8. QCoreApplication a(argc, argv);
9. //    StreamParseXml stream;
10. //    stream.writeXml();
11. //    stream.readXml();
13. DomDocument dom;
14. dom.writeXml();
15. dom.readXml();
16. qDebug() **<<** "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";
17. dom.readXml2();
19. return a.exec();
20. }

（4）运行



## 3、其他XML解析库

        常见C/C++ XML解析器有tinyxml、XERCES、squashxml、xmlite、pugxml、libxml等等，这些解析器有些是支持多语言的，有些只是单纯C/C++的。  
（1）Xerces XML解析器  
       官方网址：http://xerces.apache.org/xerces-c/  
       Xerces前身是IBM的XML4C，XML4C也是一种功能强大的XML解析器，之后交给Apache基金会管理，遂改名为Xerces，Xerces-C++让你的程序提供读写XML数据更加容易，提供的共享库通过DOM、SAX、SAX2 API等方式对XML文档进行解析、生成、操作和验证。  
       Xerces-C++忠实于XML 1.0建议和相关标准。  
       Xerces-C++解析器高性能、模块化并且可扩展。相关开发资料也比较完善。  
       除了C++版本，Xerces同时还提供Xerces Java，Xerces Perl等版本。  
(2)TinyXML解析器  
       官方网址：http://www.grinninglizard.com/tinyxml/  
       TinyXML相比Xerces要功能简单些，正如其名Tiny，使用方法也比较简单，TinyXML也是一个开源的解析XML解析库，用于C++，支持Windows和Linux。TinyXML通过DOM模型遍历和分析XML。官方文档：  
http://www.grinninglizard.com/tinyxmldocs/index.html  
(3)squashXML解析器  
        官方地址：http://ostatic.com/squashxml  
        这个解析器在国内似乎少人使用，这个解析器也有些历史了。squashXML基于DOM Level2，也是一个XML轻量级的解析器。天缘之所以把这个写上是天缘比较看重这个解析器的目录划分及使用说明，易懂而且易上手。  
(4)XMLBooster解析器  
        官方网址：http://www.xmlbooster.com/  
        XMLBooster开发关注点比较有特色，更加关注解析性能，声称：“Application integration of XML data cannot get any simpler or any faster: instead of dealing with sophisticated api (such as DOM or SAX), use a convenient data structure, generated to suit your specific purpose, in the language of your choice. ”。  
        针对特殊需求使用更加方便的数据结构以提高性能。  
(5)LibXML解析器  
        官方地址：http://xmlsoft.org/  
        LibXML本来是为Gnome项目开发（C开发），之后被广泛使用，功能非常强大，几乎适合于常见的所有操作系统下编译和开发使用。libxml++（地址：http://libxmlplusplus.sourceforge.net/）是对libxml XML解析器的C++封装版本。此外还有各种语言封装包，参考官方链接。  
(6)补充：  
        除了上述XML解析库外，还有一些XML解析器（参考：http://www.garshol.priv.no/xmltools/platform/cpp.html），比如Berkely DBXML(BDB)等，有兴趣的读者可自行Google搜索。  
尽管XML解析器有很多种，而且功能差异很大，甚至是支持跨平台、多语言，但是对于你的应用而言，尽量选择一种相对熟悉、功能够用的即可，没必要去追求庞杂的解析器，我们只需关注：功能够用、相对稳定、适合扩展这三个功能即可。一旦有问题，修正和扩展都要更为容易。

# 四、总结

（1）注意：readElementText()函数在解析出开始标签时，就可以解析元素的文本了，直到遇到结束标签。readElementText()函数中好像有类似于readNext()这样的函数读取下一个元素并比较是否为endElement，所以当使用readElementText()读取完元素文本时，该元素的闭合标签已经被读走了。  
（2）QDomProcessingInstruction instruction;instruction = doc.createProcessingInstruction("xml","version=/"1.0/" encoding=/"UTF-8/"");用来写入XML文件的声明，这对于一个XML文件来说不可缺少。  
（3）源码已经打包上传到csdn上，可登录下载（<http://download.csdn.net/detail/taiyang1987912/8857327>）。  
（4）若有建议，请留言，在此先感谢！  