# Xml && Json

数据打包，传输与存储

TinyXML & JsonCpp & rapidJson


Auth : 王桂林

Mail : guilin_wang@163.com

Org : 能众软件

Web : http://edu.nzhsoft.cn

版本信息：

| 版本 | 修订人 | 审阅人 | 时间 | 组织 |
|---|---|---|---|---|
| v1.0 | 王桂林 | | 2016.04.02 | |
| v2.0 | 王桂林 | | 2016.12.01 | 山东能众软件科技有限公司 |
| v2.1 | 王桂林 | | 2018.12.31 | 山东能众软件科技有限公司 |

更多学习：

# 1.XML

## 1.1.什么是 XML

### 1.1.1.定义

XML 是 Extensible Markup Language(可扩展标记语言)首字母的缩写词。它是这样一种语言，类似于 HTML(Hypertext Markup Language,超文本标记语言)，但拥有更加严格的语法，且标签没有关联任何的语意。

### 1.1.2.HTML 对比 XML

HTML 的标签是有格式和语意的，且不可以随便的定义，而 XML 则可以任意定义自己的标签。以下是对比。

```
<html>
    <head>
        <title>baidu.com</title>
    </head>
    <body>
        <h1>China</h1>
        <h2>America</h2>
        <h3>Canada</h3>
    </body>
</html>
```

```
<lmth>
    <tail>
        <title>baidu.com</title>
    </tail>
    <subject>
        <h1>China</h1>
        <h2>America</h2>
        <h3>Canada</h3>
    </subject>
</lmth>
```

- XML 是一种标记语言，很类似 HTML。
- XML 的设计宗旨是传输数据，而非显示数据，HTML 被设计用来显示数据，其焦点是数据的外观。
- XML 被设计用来结构化、存储以及传输信息。
- XML 标签没有被预定义，需要自行定义标签，且标签无语义。

### 1.1.3.应用（序列化与反序）

#### 1.1.3.1.存储应用

一种自解释的数据存储。Qt 中的 UI 文件本，本质就是 XML,Camtasia 中的工程文件都是 XML 的典型应用。

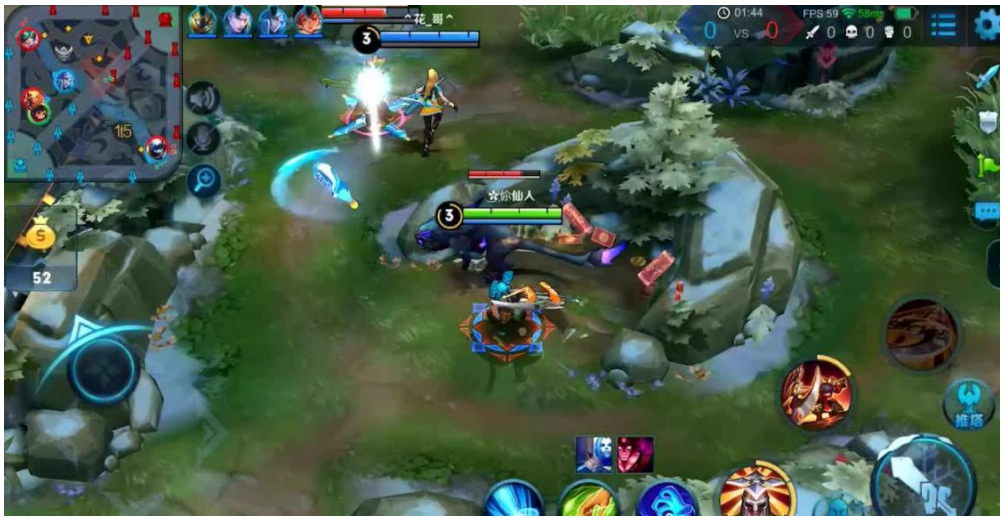
```
<ui version="4.0">
    <class>MainWindow</class>
    <widget class="QMainWindow" name="MainWindow">
        <property name="geometry">
            <rect>
                <x>0</x>
                <y>0</y>
                <width>400</width>
```

```
            <height>300</height>
        </rect>
    </property>
    <property name="windowTitle">
        <string>MainWindow</string>
    </property>
    <widget class="QWidget" name="centralWidget">
        <widget class="QPushButton" name="pushButton">
            <property name="geometry">
                <rect>
                    <x>80</x>
                    <y>210</y>
                    <width>75</width>
                    <height>23</height>
                </rect>
            </property>
            <property name="text">
                <string>Generator</string>
            </property>
        </widget>
        <widget class="QPushButton" name="pushButton_2">
            <property name="geometry">
                <rect>
                    <x>230</x>
                    <y>210</y>
                    <width>75</width>
                    <height>23</height>
                </rect>
            </property>
            <property name="text">
                <string>Send</string>
            </property>
        </widget>
        <widget class="QPlainTextEdit" name="plainTextEdit">
            <property name="geometry">
                <rect>
                    <x>60</x>
                    <y>10</y>
                    <width>281</width>
                    <height>171</height>
                </rect>
            </property>
        </widget>
```

```
        </widget>
        <widget class="QMenuBar" name="menuBar">
            <property name="geometry">
                <rect>
                    <x>0</x>
                    <y>0</y>
                    <width>400</width>
                    <height>23</height>
                </rect>
            </property>
        </widget>
        <widget class="QToolBar" name="mainToolBar">
            <attribute name="toolBarArea">
                <enum>TopToolBarArea</enum>
            </attribute>
            <attribute name="toolBarBreak">
                <bool>false</bool>
            </attribute>
        </widget>
        <widget class="QStatusBar" name="statusBar"/></widget>
    <layoutdefault spacing="6" margin="11"/>
    <resources/>
    <connections/>
</ui>
```

#### 1.1.3.2. 数据传输

网络应用的前后台数据交互。



## 1.2.格式解析

### 1.2.1.一个 XML 文案实例

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This is a XML comment -->
<note>
    <to>George</to>
    <from>John</from>
    <heading>Reminder</heading>
    <body>Don't forget the meeting!</body>
</note>
```

XML 使用简单的具有自我描述性的语法：

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。下一行描述文档的根元素(像在说："本文档是一个便签");接下来 4 行描述根的 4 个子元素(to, from, heading 以及 body);最后一行定义根元素的结尾。

### 1.2.2.XML 文档结构

XML 文档形成一种树结构,XML 文档必须包含根元素(有且只有一个)。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。

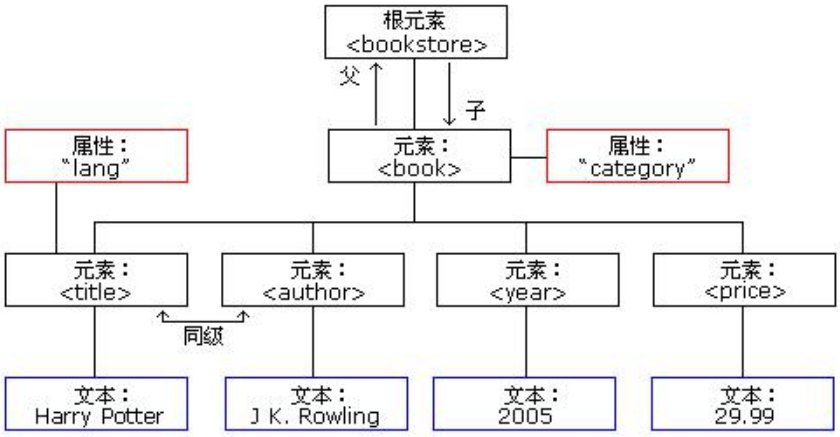
#### 1.2.2.1. 案例

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This is a XML comment -- >

<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
    <book/>
</bookstore>
```

#### 1.2.2.2. 结构

例子中的根元素是 <bookstore>。文档中的所有 <book> 元素都被包含在 <bookstore> 中。

<book> 元素有 4 个子元素：<title>、< author>、<year>、<price>。




### 1.2.3.XML 语法特点

#### 1.2.3.1..起始与注释

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

*<!-- This is a XML comment -- >*

*此两者均不属于XML 文档内容的一部分。提供给参阅者或是阅读器使用。*

#### 1.2.3.2. XML 文档必须有根元素

*<bookstore> </bookstore> 根元素*

#### 1.2.3.3. XML 标签对大小写敏感

*<bookstore> </bookstore>*

*<Bookstore> </Bookstore> 此属于两类不同的标签*

#### 1.2.3.4. XML 命名规则

*XML 元素必须遵循以下命名规则:*

*名称可以含字母、数字以及其他的字符*

*名称不能以数字或者标点符号开始*

*名称不能以字符 "xml"(或者 XML、Xml)开始*

*名称不能包含空格*

#### 1.2.3.5. 所有 XML 元素都须有关闭标签

*<bookstore> </bookstore>*

*<book category="COOKING"> </book>*

*<book/>*

*元素必须有起始和结束标签，或者自结束*

#### 1.2.3.6. XML 必须正确地嵌套

*<author>Erik T. Ray <year>2003</year> </author> 正确*

*This text is bold and italic 错误*

*This text is bold and italic 正确*

*由于  元素是在  元素内打开的，那么它必须在  元素内关闭*

#### 1.2.3.7. XML 的属性值须加引号

*<book category="COOKING"> category 是标签 book 的属性，"COOKING"是属性值。*

*<book category="WEB"> 属性值必须用引号引起*

#### 1.2.3.8. 实体引用(转义)

*如果你把字符 "<" 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。*

*<message>if salary < 1000 then</message>，会报错，此种情况，XML 提供了如下转义列表。*

| 符号 | 语义 | 转义 |
|---|---|---|
| < | 小于 | < |
| > | 大于 | > |
| & | 和号 | & |
| ' | 单引号 | ' |
| " | 引号 | " |

*所以上述写法，应该改为，<message>if salary > 1000 then</message>*

#### 1.2.3.9. XML 元素是可扩展的

*随时可以增加新的元素。*

### 1.2.4.术语解析与英文对照

| XML | 术语/英文 | 解析 |
|---|---|---|
| xx.xml | 文档<br>Document | xml 文档，通常以 xml 作为结尾，可以包含 Declaration/Comment/Element |
| <?xml version="1.0" encoding="utf-8"?> | 声明<br>Declaration | 声明语句，**至今版本只有一个**，编码通常置为 utf-8 |
| <!-- xxxxx --> | 注释<br>Comment | 提供给开发者使用 |
| <bookstore> </bookstore> | 元素<br>Element | 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。<br>元素含:Element/Attribute/Text/Comment。 |
| <bookstore> | 标签<br>Tag | 用于表示素的起始与结束 |

| | | |
|---|---|---|
| <book category="COOKING"> | 属性<br>Attribute | 用于注解分类标签的。属性是由属性名称和属性值构成，属性值必须有两(单)引号引起。 |
| <year>2005</year> | 文本<br>Text | 用于陈述标签主体，如果说属性用于划分门类，Text 则用于描述主体。 |

### 1.2.5.实例理解

#### 1.2.5.1. sample-BREAKFAST

```
<breakfast_menu>
    <food>
        <name>Belgian Waffles</name>
        <price>$5.95</price>
        <description>
            Two of our famous Belgian Waffles with plenty of real maple syrup
        </description>
        <calories>650</calories>
    </food>
    <food>
        <name>Strawberry Belgian Waffles</name>
        <price>$7.95</price>
        <description>
            Light Belgian waffles covered with strawberries and whipped cream
        </description>
        <calories>900</calories>
    </food>
</breakfast_menu>
```

#### 1.2.5.2. sample-CD

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This is a XML comment -- >

<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>Romanza</TITLE>
```

```
        <ARTIST>Andrea Bocelli</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>Polydor</COMPANY>
        <PRICE>10.80</PRICE>
        <YEAR>1996</YEAR>
    </CD>

    <CD>
        <TITLE>Soulsville</TITLE>
        <ARTIST>Jorn Hoel</ARTIST>
        <COUNTRY>Norway</COUNTRY>
        <COMPANY>WEA</COMPANY>
        <PRICE>7.90</PRICE>
        <YEAR>1996</YEAR>
    </CD>
    <CD>
        <TITLE>Midt om natten</TITLE>
        <ARTIST>Kim Larsen</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>Medley</COMPANY>
        <PRICE>7.80</PRICE>
        <YEAR>1983</YEAR>
    </CD>
</CATALOG>
```

#### 1.2.5.3. sample-PLANT

```
<CATALOG>
    <PLANT>
        <COMMON>Bloodroot</COMMON>
        <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
        <ZONE>4</ZONE>
        <LIGHT>Mostly Shady</LIGHT>
        <PRICE>$2.44</PRICE>
        <AVAILABILITY>031599</AVAILABILITY>
    </PLANT>
    <PLANT>
        <COMMON>Columbine</COMMON>
        <BOTANICAL>Aquilegia canadensis</BOTANICAL>
        <ZONE>3</ZONE>
        <LIGHT>Mostly Shady</LIGHT>
        <PRICE>$9.37</PRICE>
        <AVAILABILITY>030699</AVAILABILITY>
    </PLANT>
```

```
<PLANT>
    <COMMON>Ginger, Wild</COMMON>
    <BOTANICAL>Asarum canadense</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$9.03</PRICE>
    <AVAILABILITY>041899</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Mayapple</COMMON>
    <BOTANICAL>Podophyllum peltatum</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$2.98</PRICE>
    <AVAILABILITY>060599</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Phlox, Woodland</COMMON>
    <BOTANICAL>Phlox divaricata</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Sun or Shade</LIGHT>
    <PRICE>$2.80</PRICE>
    <AVAILABILITY>012299</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Phlox, Blue</COMMON>
    <BOTANICAL>Phlox divaricata</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Sun or Shade</LIGHT>
    <PRICE>$5.59</PRICE>
    <AVAILABILITY>021699</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Spring-Beauty</COMMON>
    <BOTANICAL>Claytonia Virginica</BOTANICAL>
    <ZONE>7</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$6.59</PRICE>
    <AVAILABILITY>020199</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Trillium</COMMON>
    <BOTANICAL>Trillium grandiflorum</BOTANICAL>
```

```
    <ZONE>5</ZONE>
    <LIGHT>Sun or Shade</LIGHT>
    <PRICE>$3.90</PRICE>
    <AVAILABILITY>042999</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Grecian Windflower</COMMON>
    <BOTANICAL>Anemone blanda</BOTANICAL>
    <ZONE>6</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$9.16</PRICE>
    <AVAILABILITY>071099</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Bee Balm</COMMON>
    <BOTANICAL>Monarda didyma</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$4.59</PRICE>
    <AVAILABILITY>050399</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Bergamot</COMMON>
    <BOTANICAL>Monarda didyma</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$7.16</PRICE>
    <AVAILABILITY>042799</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Black-Eyed Susan</COMMON>
    <BOTANICAL>Rudbeckia hirta</BOTANICAL>
    <ZONE>Annual</ZONE>
    <LIGHT>Sunny</LIGHT>
    <PRICE>$9.80</PRICE>
    <AVAILABILITY>061899</AVAILABILITY>
</PLANT>
<PLANT>
    <COMMON>Buttercup</COMMON>
    <BOTANICAL>Ranunculus</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$2.57</PRICE>
```

```
        <AVAILABILITY>061099</AVAILABILITY>
    </PLANT>
    <PLANT>
        <COMMON>Crowfoot</COMMON>
        <BOTANICAL>Ranunculus</BOTANICAL>
        <ZONE>4</ZONE>
        <LIGHT>Shade</LIGHT>
        <PRICE>$9.34</PRICE>
        <AVAILABILITY>040399</AVAILABILITY>
    </PLANT>

</CATALOG>
```

# 2.TinyXml

## 2.1.简介

TinyXML 是一个开源的解析 XML 的解析库，能够用于 C++，能够在 Windows 或 Linux 中编译。这个解析库的模型通过解析 XML 文件，然后在内存中生成 DOM(Document Object Model)模型，从而让我们很方便的遍历这棵 XML 树。


## 2.2.Qt+TinyXML 环境搭建

### 2.2.1.下载地址

软件下载：https://github.com/leethomason/tinyxml2

官方文档：*http://www.grinninglizard.com/tinyxml2docs/index.html*

### 2.2.2.环境搭建

下载完成后，解压找到 tinyxml2.cpp 以及 tinyxml2.h，并且添加到 c++工程目录当中，与其他源码一起编译即可。

```
/测试环境搭建
#include <iostream>
#include "tinyxml2.h"
using namespace tinyxml2;
using namespace std;

int main()
{
    XMLDocument doc;
    doc.LoadFile("baidu.xml");
    doc.Print();
    cout<<"doc status "<<doc.ErrorName()<<endl;
    return 0;
}
```

### 2.2.3.TinyXML 框架解析

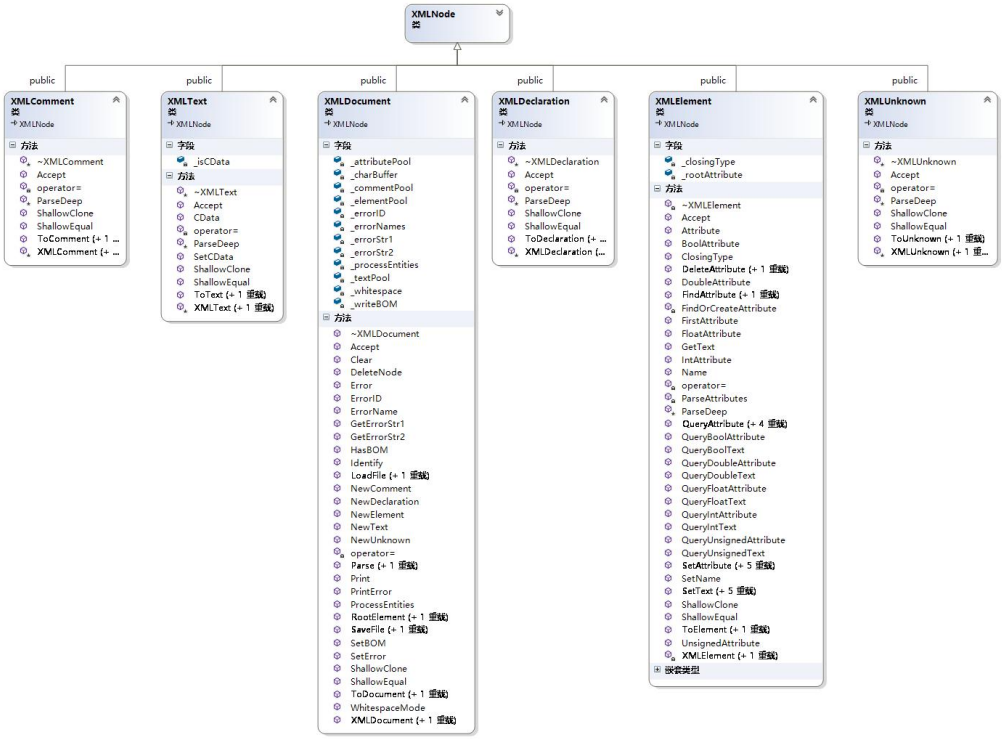
#### 2.2.3.1. DOM 对象模型

TinyXml 使用文档对象模型(DOM)来解析 xml 文件，这种模型的处理方式为在分析时，一次性的将整个 XML 文档进行分析，并在内存中形成对应的树结构，同时，向用户提供一系列的接口来访问和编辑该树结构。这种方式占用内存大，但可以给用户提供一个面向对象的访问接口，对用户更为友好，非常方便用户使用。

| | |
|---|---|
| A Document can contain: | Element(container or leaf)<br>Comment (leaf)<br>Unknown (leaf)<br>Declaration( leaf ) |
| An Element can contain: | Element (container or leaf)<br>Text(leaf)<br>Attributes (not on tree)<br>Comment (leaf)<br>Unknown (leaf) |

#### 2.2.3.2.类图关系

```
namespace tinyxml2
{
    class XMLDocument;
    class XMLElement;
    class XMLAttribute;
    class XMLComment;
    class XMLText;
    class XMLDeclaration;
    class XMLUnknown;
    class XMLPrinter;
}
```




### 2.2.3.3.常用接口

```
XMLElement* RootElement()
XMLElement* FirstChildElement( const char* name = 0 )
XMLElement*PreviousSiblingElement( const char* name = 0 )

XMLElement* NextSiblingElement( const char* name = 0 )
XMLNode* LinkEndChild( XMLNode* addThis )
XMLElement* XMLNode::FirstChildElement( const char* name )
XMLElement* XMLNode::LastChildElement( const char* name )


XMLElement* XMLDocument::NewElement( const char* name )
XMLComment* XMLDocument::NewComment( const char* str )
XMLText* XMLDocument::NewText( const char* str )
XMLDeclaration* XMLDocument::NewDeclaration( const char* str )

const char* XMLAttribute::Name() const
```

### 2.2.4.写 XML

#### 2.2.4.1. XML 原文件如下

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This is a XML comment -->

<bookstore>
	<book category="COOKING">
		<title lang="en">Everyday Italian</title>
		<author>Giada De Laurentiis</author>
		<year>2005</year>
		<price>30.00</price>
	</book>
</bookstore>
```

#### 2.2.4.2. 代码实现

```
#include <iostream>
#include "tinyxml2.h"
using namespace tinyxml2;
using namespace std;


int main()
{
    XMLDocument  doc;
    doc.LinkEndChild(
              doc.NewDeclaration(
              "xml version=\"1.0\" encoding=\"UTF-8\"")
              );
    doc.LinkEndChild(doc.NewComment("This is a XML comment"));

    XMLElement *pEBookStore = doc.NewElement("bookstore");
       XMLElement *pEBook  = doc.NewElement("book");
       pEBook->SetAttribute("category","COOKING");
          XMLElement *pETitle = doc.NewElement("title");
          pETitle->SetAttribute("lang","en");
          pETitle->SetText("Everyday Italian");
          pEBook->LinkEndChild(pETitle);
          XMLElement *pEAuthor = doc.NewElement("author");
          pEAuthor->SetText("Giada De Laurentiis");
          pEBook->LinkEndChild(pEAuthor);
```

```
            XMLElement *pEYear= doc.NewElement("year");
            pEYear->SetText("2005");
            pEBook->LinkEndChild(pEYear);
            XMLElement *pEPrice= doc.NewElement("price");
            pEPrice->SetText("30.00");
            pEBook->LinkEndChild(pEPrice);

        pEBookStore->LinkEndChild(pEBook);

    doc.LinkEndChild(pEBookStore);

    doc.Print();
    doc.SaveFile("test.xml");

    return 0;
}
```

### 2.2.5.读 XML

```
#include <iostream>
#include "tinyxml2.h"
using namespace  tinyxml2;
using namespace std;


int main()
{
    XMLDocument doc;
    doc.LoadFile("bookstore.xml");

    XMLElement * pEBookStore = doc.RootElement();
    cout<<pEBookStore->Name()<<endl;

    XMLElement *pEBook = pEBookStore->FirstChildElement();
    cout<<pEBook->Attribute("category")<<endl;

    XMLElement *pEBookChild = pEBook->FirstChildElement();
    while(pEBookChild)
    {
        cout<<pEBookChild->Value()<<":"
          <<pEBookChild->GetText()<<endl;
        pEBookChild = pEBookChild->NextSiblingElement();
    }
```

```
    doc.Print();
    return 0;
}
```

### 2.2.6.XMLPrinter

```
XMLPrinter printer;
doc.Print( &printer );
cout<<printer.CStr();
```

### 2.2.7.综合操作

#### 2.2.7.1.增加一个元素

#### 2.2.7.2.修改一个属性

#### 2.2.7.3.修改一个值

## 2.3.课上练习

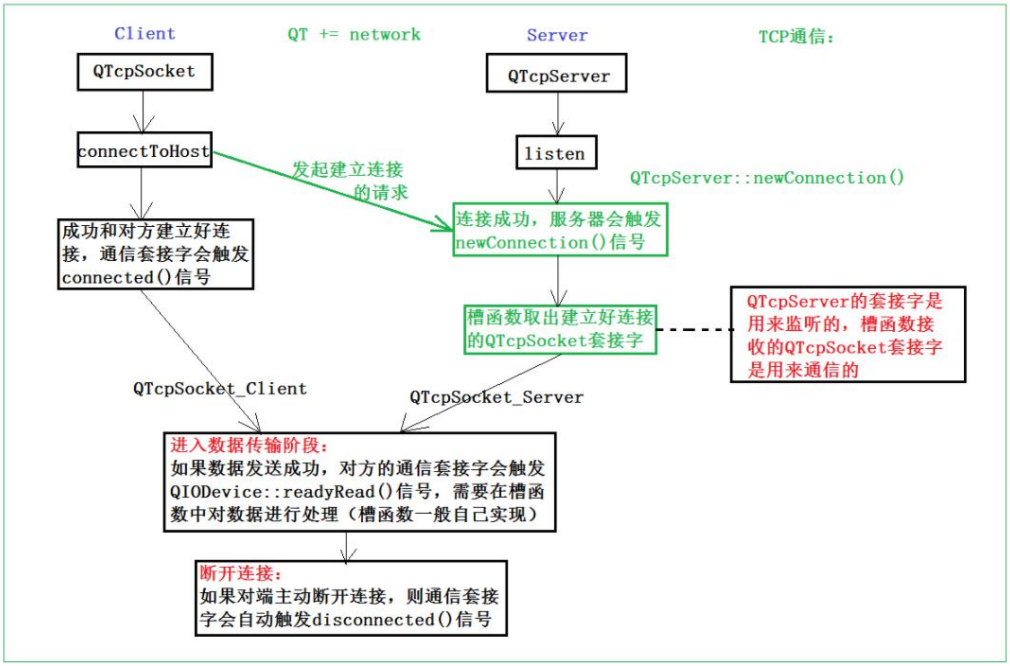
### 2.3.1.将如下 xml 写入文件

```
<?xml version="1.0" encoding="utf-8"?>
<country>
    <name>中国</name>
    <province>
        <name>黑龙江</name>
        <cities>
            <city>哈尔滨</city>
            <city>大庆</city>
        </cities>
    </province>
    <province>
        <name>广东</name>
        <cities>
            <city>广州</city>
            <city>深圳</city>
            <city>珠海</city>
        </cities>
    </province>
    <province>
        <name>台湾</name>
        <cities>
            <city>台北</city>
            <city>高雄</city>
        </cities>
    </province>
    <province>
        <name>新疆</name>
        <cities>
            <city>乌鲁木齐</city>
        </cities>
    </province>
</country>
```

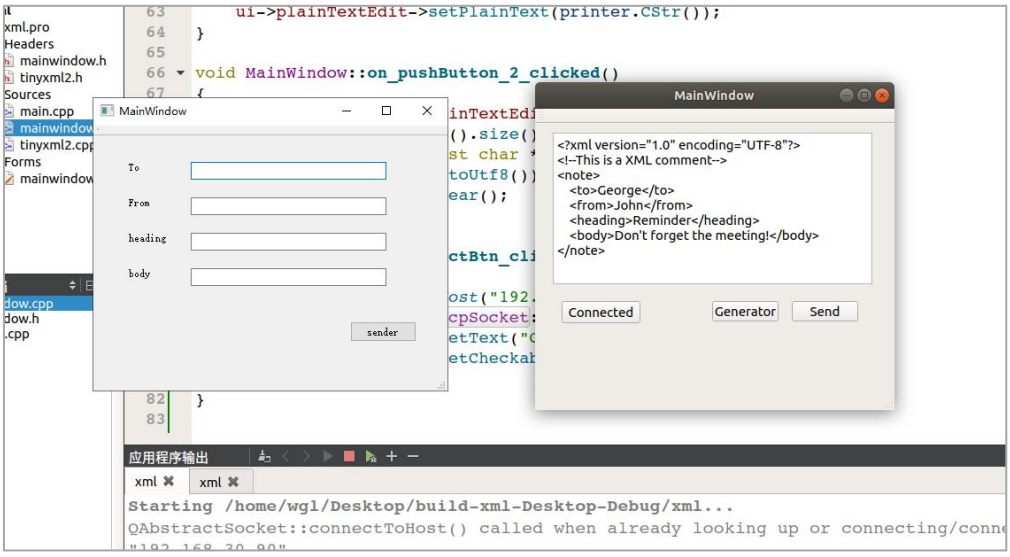
### 2.3.2.然后读出显示

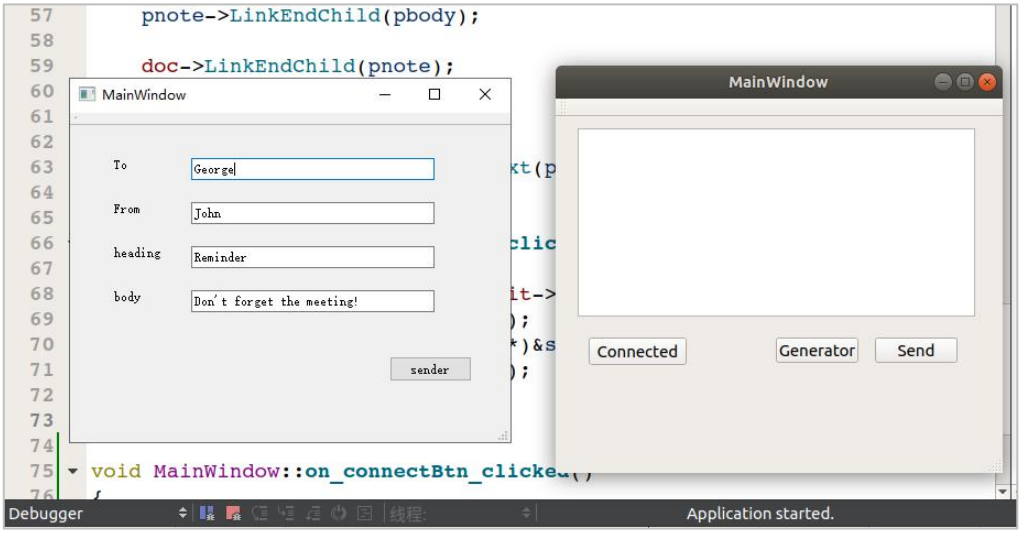
## 2.4.前后台传输

### 2.4.1.QTcpSocket/QTcpServer




### 2.4.2.成果展示






### 2.4.3.前端实现

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "qdebug.h"
#include "iostream"
#include "QHostAddress"


using namespace std;

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    tcpSocket = new QTcpSocket(this);
    connect(tcpSocket, &QTcpSocket::disconnected,
            [=](){
        ui->connectBtn->setText("Connect");
        ui->connectBtn->setCheckable(true);
    });

    doc = new XMLDocument;

};


MainWindow::~MainWindow()
```

```
{
    delete ui;
    delete doc;
}

void MainWindow::on_pushButton_clicked()
{
    doc->Clear();
    doc->LinkEndChild(
              doc->NewDeclaration("xml version=\"1.0\"
encoding=\"UTF-8\""));
    doc->LinkEndChild(
              doc->NewComment("This is a XML comment"));
    XMLElement * pnote =  doc->NewElement("note");

    XMLElement * pto = doc->NewElement("to");
    pto->SetText("George");
    pnote->LinkEndChild(pto);

    XMLElement * pfrom = doc->NewElement("from");
    pfrom->SetText("John");
    pnote->LinkEndChild(pfrom);

    XMLElement * pheading = doc->NewElement("heading");
    pheading->SetText("Reminder");
    pnote->LinkEndChild(pheading);


    XMLElement *pbody = doc->NewElement("body");
    pbody->SetText("Don't forget the meeting!");
    pnote->LinkEndChild(pbody);

    doc->LinkEndChild(pnote);

    XMLPrinter  printer;
    doc->Print(&printer);
    ui->plainTextEdit->setPlainText(printer.CStr());
}

void MainWindow::on_pushButton_2_clicked()
{
    QString str = ui->plainTextEdit->toPlainText();
    int size = str.toUtf8().size();
```

```
    tcpSocket->write((const char *)&size,4);
    tcpSocket->write(str.toUtf8());
    ui->plainTextEdit->clear();
}

void MainWindow::on_connectBtn_clicked()
{
    tcpSocket->connectToHost("192.168.30.90", 8888);
    connect(tcpSocket,&QTcpSocket::connected,[&](){
        ui->connectBtn->setText("Connected");
        ui->connectBtn->setCheckable(false);
    });
}
```

### 2.4.4.后端实现

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
#include <iostream>
using namespace  std;

#define PORT_SERVER  8888
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    tcpServer = new QTcpServer(this);
    tcpServer->listen(QHostAddress::Any, PORT_SERVER);

    doc = new XMLDocument;

    connect(tcpServer,&QTcpServer::newConnection,[&](){

        tcpSocket = tcpServer->nextPendingConnection();

        connect(tcpSocket,&QTcpSocket::readyRead,[&](){
            int len = 0;
            tcpSocket->read((char*)&len,4);
            char buf[2048] = {0};
            tcpSocket->read(buf,len);
```

```
            buf[len] = '\0';

            for(int i=0; i<len; i++)
                cout<<buf[i];
            cout.flush();

            if (doc->Parse(buf) == XML_SUCCESS)
            {
                auto pnode =  doc->RootElement();
                auto pto = pnode->FirstChildElement();
                ui->toLineEdit->setText(pto->GetText());

                auto pfrom = pto->NextSiblingElement();
                ui->fromLineEdit->setText(pfrom->GetText());

                auto pheading = pfrom->NextSiblingElement();
                ui->headingLineEdit->setText(pheading->GetText());

                auto pbody = pheading->NextSiblingElement();
                ui->bodyLineEdit->setText(pbody->GetText());

                qDebug()<<buf;
                for(int i=0; i<len; i++)
                    cout<<buf[i];
            }
        });
    });
}

MainWindow::~MainWindow()
{
    delete ui;
    delete doc;
}

void MainWindow::on_pushButton_clicked()
{
    doc->Clear();

}
```

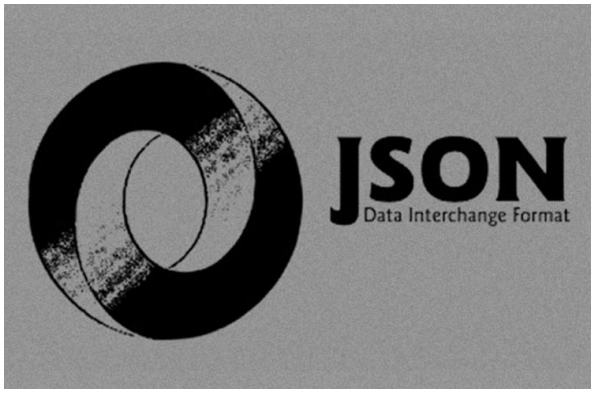
# 3.JSON

## 3.1.什么是 JSON

### 3.1.1.定义

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它使得人们很容易的进行阅读和编写。同时也方便了机器进行解析和生成。

它是基于 JavaScript Programming Language , Standard ECMA-262 3rd Edition - December 1999 的一个子集。 JSON 采用完全独立于程序语言的文本格式，但是也使用了类 C 语言的习惯(包括 C, C++, C#, Java, JavaScript, Perl, Python 等)。这些特性使 JSON 成为理想的数据交换语言。




### 3.1.2.官网及文档

官　　网：http://json.org/

在线解析器：http://json.cn/　http://www.bejson.com/

## 3.2.格式解析

### 3.2.1.一个 json 实例

```
{
"animals":
{
    "dog":
     [
        {"name": "Rufus", "age": 15},

        {"name": "Marty","age": null}
      ],


    "cat":
     [
         {"name": "bosi","age": 15},

         {"name": "maowang","age": null}

      ]
    }
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<animals>
    <dog>
        <name>Rufus</name>
        <age>15</age>
    </dog>
    <dog>
        <name>Marty</name>
        <age />
    </dog>
    <cat>
        <name>bosi</name>
        <age>15</age>
    </cat>
    <cat>
        <name>maowang</name>
        <age />
    </cat>
</animals>
```

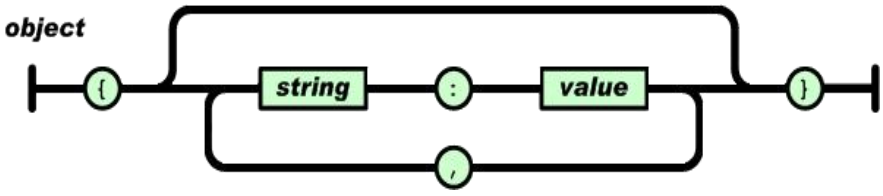
### 3.2.2.文档结构

json 简单说就是 javascript 中的对象和数组，所以这两种结构就是对象和数组两种结构，通过这两种结构可以表示各种复杂的结构。

#### 3.2.2.1.对象(Object)

对象是一个无序的" '名称/值'对"集合。一个对象以"{"（左括号）开始，"}"（右括号）结束。每个"名称"后跟一个":"（冒号）；" '名称 /值' 对"之间使用","（逗号）分隔。

一个对象中，key 不可以重复，若有重复，后者覆盖前者。




示范：

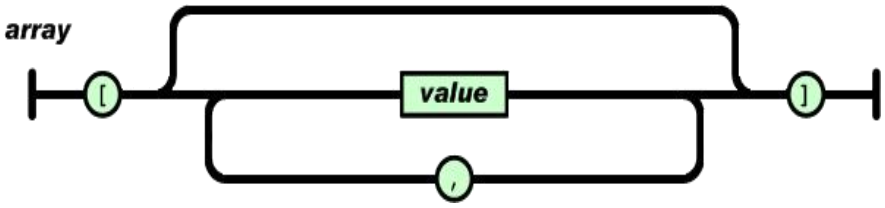
●空对象 {}

●含有一个 k-v 的对象 {"name":"zhangsan"}

●含有多个 k-v 的对象 {"name":"zhangsan","sex":"f","age":23}

#### 3.2.2.2.数组(Array)

数组是值（value）的有序集合。一个数组以"["（左中括号）开始，"]"（右中括号）结束。值之间使用","（逗号）分隔。




示范：

●空数组 []

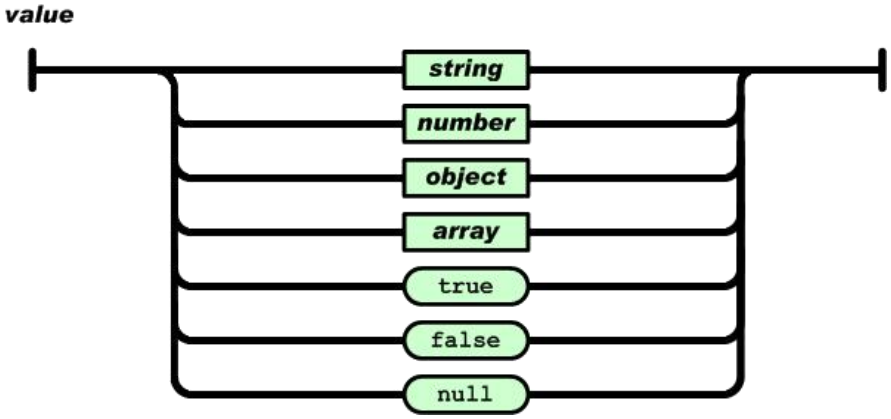
●含有一个元素的数组 [1] ["abc"]

●含有多个元素的数组 [1,"china",null,33,4,{"q":"a"}]

### 3.2.3.数据类型

#### 3.2.3.1.值(Value)

值(value) 可以是双引号括起来的 string、number、true、false、 null、对象(object)或者数组(array)

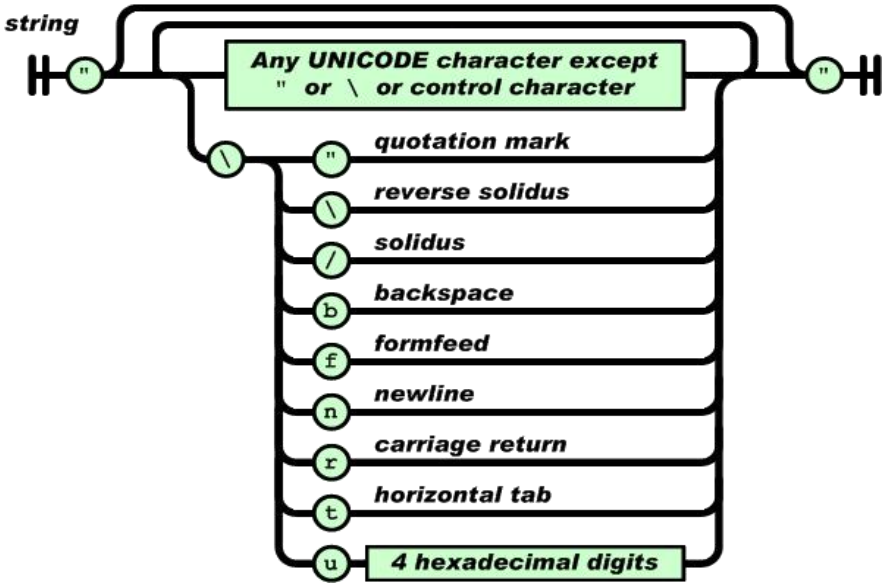



示范：

- 数字(整数或浮点数) 123
- 字符串(在双引号中) "abc"
- 逻辑值(true 或 false) true false
- 数组(在方括号中) [1,2,3]
- 对象(在花括号中) {"abc":123}
- null null

#### 3.2.3.2.字符串(String)

字符串(string) 是由双引号包围的任意数量 Unicode 字符的集合，使用反斜线转义。一个字符(character)即一个单独的字符串(character string)。

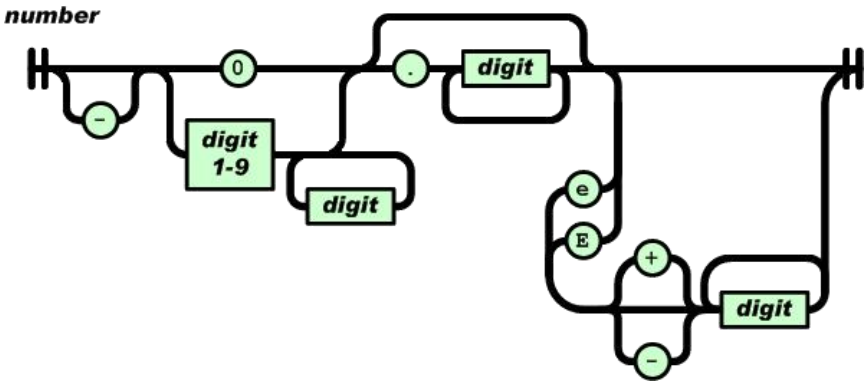



示范：

- 不含控制字符 "abc"
- 含有控制字符 "a\\bc"

### 3.2.4.数据(Number)

数值(number) 也与 C 或者 Java 的数值非常相似。只是 JSON 的数值没有使用八进制与十六进制格式。

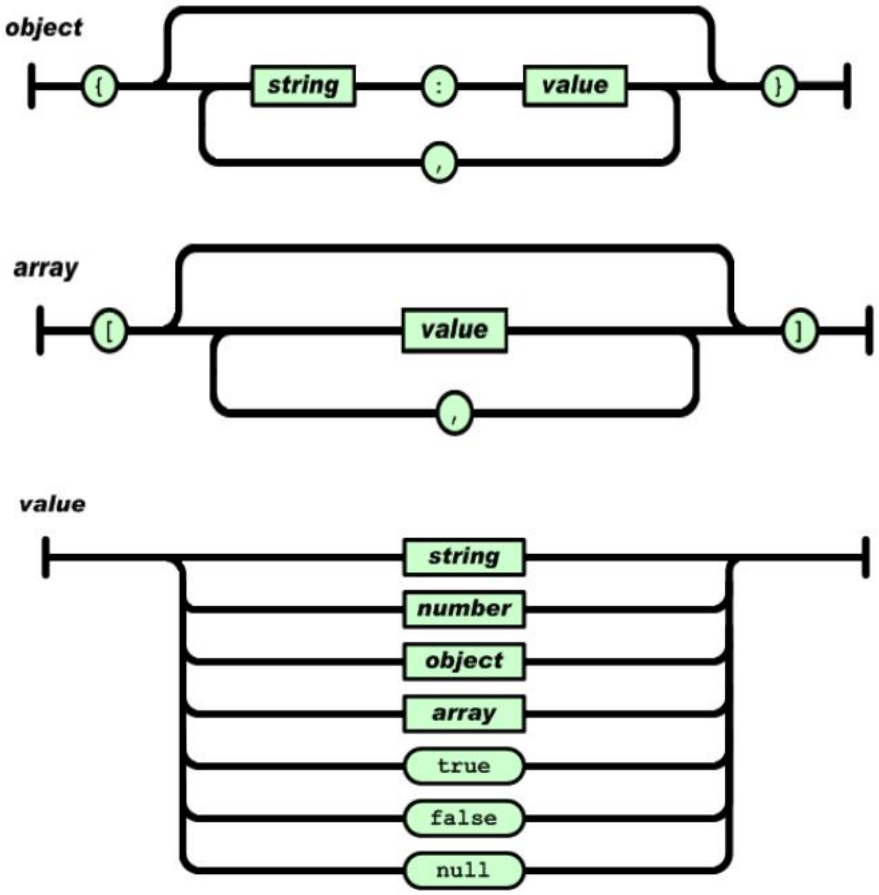



示范：

- 数字(整数或浮点数) 123

## 3.3.实现理解嵌套

### 3.3.1.Value



### 3.3.2.实例

#### 3.3.2.1. sample-PEOPLE

```
{
    "FirstName": "John",
    "LastName": "Doe",
    "Age": 43,
    "Address": {
        "Street": "Downing Street 10",
        "City": "London",
        "Country": "Great Britain"
    },
    "Phone numbers": [
        "+44 1234567",
        "+44 2345678"
    ]
}
```

#### 3.3.2.2. sample-EMPLOYEE

```
{
   "employees": [
      {
         "firstName": "Bill",
         "lastName": "Gates"
      },
      {
         "firstName": "George",
         "lastName": "Bush"
      },
      {
         "firstName": "Thomas",
         "lastName": "Carter"
      }
   ]
}
```

#### 3.3.2.3. sample-ADDRESS

```
{
   "name": "BeJson",
   "url": "http://edu.nzhsoft.com",
   "page": 88,
   "isNonProfit": true,
   "address": {
```

```
        "street": "科技园路.",
        "city": "江苏苏州",
        "country": "中国"
    },
    "links": [
        {
            "name": "Google",
            "url": "http://edu.nzhsoft.com"
        },
        {
            "name": "Baidu",
            "url": "http://edu.nzhsoft.com"
        },
        {
            "name": "nzhsoft",
            "url": "http://edu.nzhsoft.com"
        }
    ]
}
```

#### 3.3.2.4. sampe-TEACHER

```
{
    "teachers": [
        {
            "name": "黄波",
            "course": "网页高级设计"
        },
        {
            "name": "贺敏",
            "course": "Java 程序设计"
        },
        {
            "name": "毛丽娟",
            "course": "JavaScript 程序设计"
        }
    ],
    "students": [
        {
            "name": "张三",
            "age": 20
        },
        {
```

```
            "name": "李四",
            "age": 21
        },
        {
            "name": "王五",
            "age": 19
        }
    ]
}
```

# 4.JsonCpp

## 4.1.JsonCpp 简介

JSON is a lightweight data-interchange format. It can represent numbers, strings, ordered sequences of values, and collections of name/value pairs.

JsonCpp is a C++ library that allows manipulating JSON values, including serialization and deserialization to and from strings. It can also preserve existing comment in unserialization/serialization steps, making it a convenient format to store user input files.

## 4.2.JsonCpp 环境搭建

### 4.2.1.官方及下载

https://github.com/open-source-parsers/jsoncpp#generating-amalgamated-source-and-header

### 4.2.2.Qt+JsonCpp 环境搭建

①解压 jsoncpp-master.zip 包

②在根目录下，运行 python amalgamate.py

③在根目录中生成 dist 文件夹包含三个文件 dist/json/json-forwards.h dist/json/json.h dist/json.cpp

④在 Qt 工程目录下，生成 json 文件夹，并拷贝 json.h 到 json 目录下。

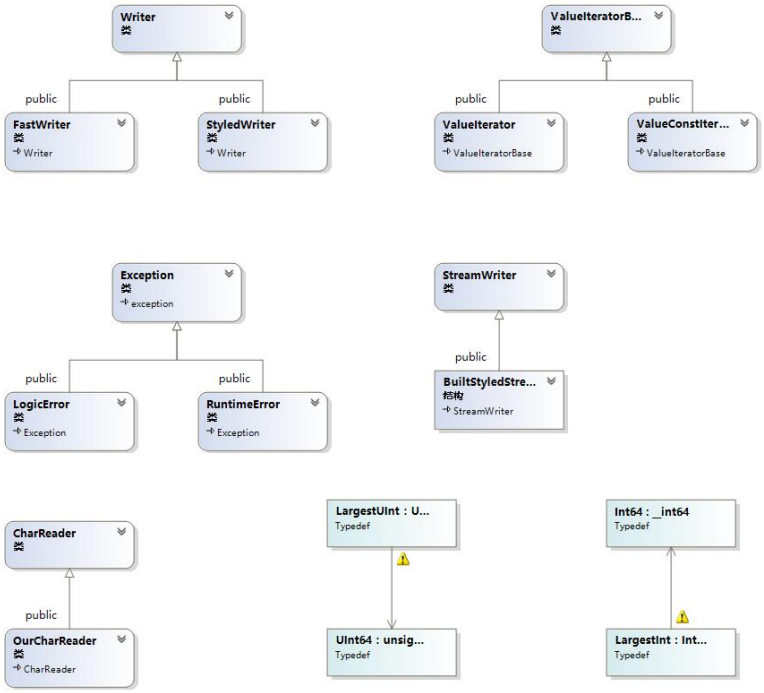
⑤在 Qt 工程中添加现有文件即可。

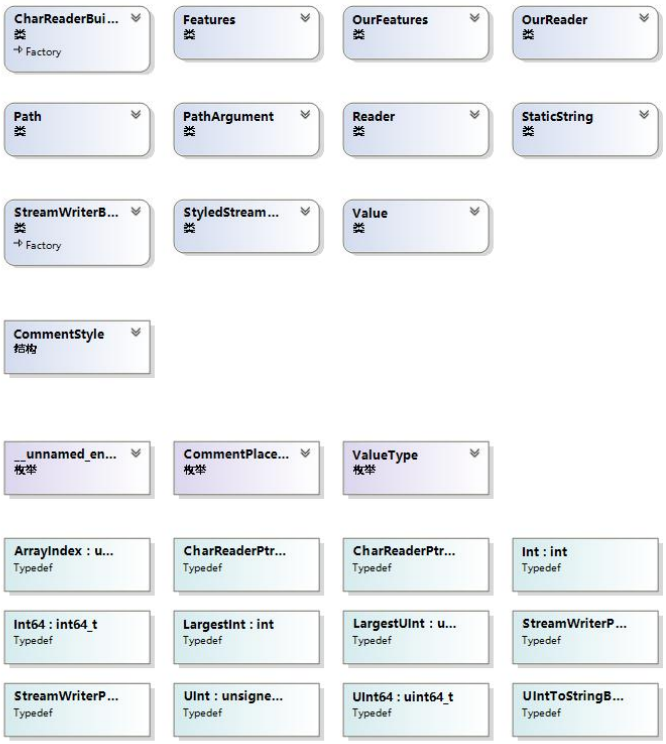
## 4.3.框架解析

### 4.3.1.类图关系

```
namespace Json {

    // writer.h
    class FastWriter;
    class StyledWriter;

    // reader.h
    class Reader;

    // features.h
    class Features;

    // value.h
    typedef unsigned int ArrayIndex;
```

```
    class StaticString;
    class Path;
   class PathArgument;
   class Value;
   class ValueIteratorBase;
   class ValueIterator;
   class ValueConstIterator;
} // namespace Json
```






### 4.3.2.常用接口

| 对象 | 释意 |
| --- | --- |
| Json::Value | 是 jsoncpp 中最基本、最重要的类，用于表示各种类型的对象。 |
| Json::Reader | 用来将内存或文件中的 json 数据转换成 Json::Value 类型。 |
| Json::Writer | 用来将 Json::Value 类型转换成内存或文件中的 json 数据。 |

## 4.4.读写 JSON

### 4.4.1.序列化

#### 4.4.1.1.写对象

```
#include <iostream>
#include "json/json.h"
using namespace Json;
using namespace std;

int main()
{
    Value rootObj;
    rootObj["name"] = "zhangsan";
    rootObj["sex"]  = "f";
    rootObj["age"]  = 23;
```

```
    string str = rootObj.toStyledString();

    cout<<str<<endl;
    return 0;
}
```

#### 4.4.1.2. 写数组

```
#include <iostream>
#include "json/json.h"
#include "fstream"
using namespace Json;
using namespace std;


int main()
{
    Value array;
    for(int i=0; i<10; i++)
    {
        array.append(i);
    }
    array.append("abc");

    Value obj;
    obj["name"] = "zhangsan";
    obj["sex"]  = "f";
    obj["age"]  = 23;

    array.append(obj);


    string str = array.toStyledString();
    cout<<str<<endl;

    fstream  fs("text.json",ios::out);

    fs<<str;
    fs.close();

    return 0;
}
```

#### 4.4.1.3.写入 json

```
#include <iostream>
#include <fstream>
#include "json/json.h"
using namespace  Json;
using namespace std;


//{
//     "FirstName": "John",
//     "LastName": "Doe",
//     "Age": 43,
//     "Address": {
//         "Street": "Downing Street 10",
//         "City": "London",
//         "Country": "Great Britain"
//     },
//     "Phone numbers": [
//         "+44 1234567",
//         "+44 2345678"
//     ]
// }
int main()
{
    Value  obj;
    obj["FirstName"] = "John";
    obj["LastName"] = "Doe";
    obj["Age"] = 43;
    Value objAddr;
    objAddr["Street"] = "Downing Street 10";
    objAddr["City"] = "London";
    objAddr["Country"] = "Great Britain";

    obj["Address"] = objAddr;

    Value arrPhone;
    arrPhone.append("+44 1234567");
    arrPhone.append("+44 2345678");
    obj["Phone numbers"] = arrPhone;


    string str = obj.toStyledString();
    cout<<str<<endl;
```

```
    fstream  fs("my.json",ios::out);

    return 0;
}
```

### 4.4.2.FastWriter/StyledWriter

#### 4.4.2.1. fstream

```
string str = obj.toStyledString();
cout<<str<<endl;
fstream  fs("my.json",ios::out);
fs<<str;
fs.close();
```

#### 4.4.2.2. styled/fast

```
    StyledWriter sw;
    string str = sw.write(obj);
    cout<<str;

    FastWriter  fw;
    str = fw.write(obj);
    cout<<str;
```

### 4.4.3.反序列化

#### 4.4.3.1. 读对象 Reader

```
fstream fs("your.json",ios::in);
if(!fs)
    cout<<"open error"<<endl;

Value  val;

Reader reader;
if(reader.parse(fs,val))
{
//        cout<<val.toStyledString()<<endl;

        cout<<val["address"]["city"].asCString()<<endl;
        cout<<val["address"]["street"].asCString()<<endl;
```

```
        cout<<val["address"]["country"].asCString()<<endl;
```

#### 4.4.3.2. 读数组

```
Value & v = val["links"];
for(int i=0; i<v.size(); i++)
{
   cout<<v[i]["name"]<<endl;
   cout<<(v[i]["url"] = "abcdefg")<<endl;
}
```

#### 4.4.3.3. 读 Json

```
#include <iostream>
#include <fstream>
#include "json/json.h"

using namespace std;
using namespace Json;


//{
//    "name": "BeJson",
//    "url": "http://www.bejson.com",
//    "page": 88,
//    "isNonProfit": true,
//    "address": {
//        "street": "科技园路.",
//        "city": "江苏苏州",
//        "country": "中国"
//    },
//    "links": [
//        {
//            "name": "Google",
//            "url": "http://www.google.com"
//        },
//        {
//            "name": "Baidu",
//            "url": "http://www.baidu.com"
//        },
//        {
//            "name": "SoSo",
```

```
//          "url": "http://www.SoSo.com"
//       }
//    ]
//}


int main()
{
    fstream fs("your.json",ios::in);
    if(!fs)
       cout<<"open error"<<endl;

    Value  val;

    Reader reader;
    if(reader.parse(fs,val))
    {
//        cout<<val.toStyledString()<<endl;

       cout<<val["address"]["city"].asCString()<<endl;
       cout<<val["address"]["street"].asCString()<<endl;
       cout<<val["address"]["country"].asCString()<<endl;

       Value & v = val["links"];

       for(int i=0; i<v.size(); i++)
       {
          cout<<v[i]["name"]<<endl;
          cout<<(v[i]["url"] = "abcdefg")<<endl;
       }

    }

    cout<<val.toStyledString()<<endl;

    return 0;
}
```

## 4.5.JSON Support in Qt

# 5.ProtoBuf

# 6.配置文件

## 6.1.数据

### 6.1.1.数据

### 6.1.2.配置文件

当某个属性需要修改的时候只需要改动配置而不是文件，可以极大的减少开发期在编译中消耗的时间，特别是原来的宏定义的方式，假如很多地方需要而被放在了头文件中，每次的更改对于大型工程来说简直就是程序员的恶梦.....

## 6.2.配置格式

### 6.2.1.json

### 6.2.2.xml

## 6.3.实战读配置文件

### 6.3.1.读 json

### 6.3.2.读 xml

# 7.协议远程传输

## 7.1.协议

## 7.2.Client

## 7.3.Server