<http://blog.csdn.net/liang19890820/article/details/52767153>

Qt5 中包含了处理 JSON 的类，均以 QJson 开头（例如：QJsonDocument、QJsonArray、QJsonObject），在 QtCore 模块中，不需要额外引入其它模块。

* [简述](http://blog.csdn.net/liang19890820/article/details/52767153#简述)
* [常用的 JSON 库](http://blog.csdn.net/liang19890820/article/details/52767153#常用的-json-库)
* [JSON 常用类](http://blog.csdn.net/liang19890820/article/details/52767153#json-常用类)
* [简单的 JSON 对象](http://blog.csdn.net/liang19890820/article/details/52767153#简单的-json-对象)
* [简单的 JSON 数组](http://blog.csdn.net/liang19890820/article/details/52767153#简单的-json-数组)
* [复杂的 JSON](http://blog.csdn.net/liang19890820/article/details/52767153#复杂的-json)
* [更多参考](http://blog.csdn.net/liang19890820/article/details/52767153#更多参考)

# 常用的 JSON 库

[json.org](http://json.org/) 中介绍了 JSON 在各种语言中的应用，在 C/C++ 中比较常用的JSON 库主要有以下几个：

* JsonCpp   
  JsonCpp 是一个 C++ 用来处理 JSON 数据的开发包。   
  网址：<http://jsoncpp.sourceforge.net/>
* cJSON   
  cJSON 是一个超轻巧，携带方便，单文件，简单的可以作为 ANSI-C 标准的 JSON 解析器。   
  网址：<http://sourceforge.net/projects/cjson/>
* QJson   
  QJson 是一个基于 Qt 的开发包用来将 JSON 数据解析成 QVariant 对象，JSON 的数组将被映射为QVariantList 实例，而其他对象映射为 QVariantMap 实例。   
  网址：<http://qjson.sourceforge.net/>

关于 Qt 中对 JSON 的生成与解析，Qt5 以前的版本，可以使用 QJson 库，需要单独下载、编译，才能使用。到了 Qt5，提供了专门的 QJsonDocument 及其相关类来读和写 JSON 文档。

# JSON 常用类

QJsonDocument

QJsonDocument 类用于读和写 JSON 文档。

一个 JSON 文档可以使用 QJsonDocument::fromJson() 从基于文本的表示转化为 QJsonDocument， toJson() 则可以反向转化为文本。解析器非常快且高效，并将 JSON 转换为 Qt 使用的二进制表示。

已解析文档的有效性，可以使用 !isNull() 进行查询。

如果要查询一个 JSON 文档是否包含一个数组或一个对象，使用 isArray() 和 isObject()。包含在文档中的数组或对象可以使用 array() 或 object() 检索，然后读取或操作。

也可以使用 fromBinaryData() 或 fromRawData() 从存储的二进制表示创建来 JSON 文档。

QJsonArray

QJsonArray 类封装了一个 JSON 数组。

JSON 数组是值的列表。列表可以被操作，通过从数组中插入和删除 QJsonValue 。

一个 QJsonArray 可以和一个 QVariantList 相互转换。可以使用 size() 来查询条目的数量，通过 insert() 在指定索引处插入值，removeAt() 来删除指定索引的值。

QJsonObject

QJsonObject 类封装了一个 JSON 对象。

一个 JSON 对象是一个“key/value 对”列表，key 是独一无二的字符串，value 由一个 QJsonValue 表示。

一个 QJsonObject 可以和一个 QVariantMap 相互转换。可以使用 size() 来查询“key/value 对”的数量，通过 insert() 插入“key/value 对”， remove() 删除指定的 key。

QJsonValue

QJsonValue 类封装了一个值。

JSON 中的值有 6 种基本数据类型：

* bool（QJsonValue::Bool）
* double（QJsonValue::Double）
* string（QJsonValue::String）
* array（QJsonValue::Array）
* object（QJsonValue::Object）
* null（QJsonValue::Null）

一个值可以由任何上述数据类型表示。此外，QJsonValue 有一个特殊的标记来表示未定义的值，可以使用 isUndefined() 查询。

值的类型可以通过 type() 或 isBool()、isString() 等访问函数查询。同样地，值可以通过 toBool()、toString() 等函数转化成相应的存储类型。

QJsonParseError

QJsonParseError 类用于在 JSON 解析中报告错误。

枚举 QJsonParseError::ParseError：

该枚举描述 JSON 文档在解析过程中所发生的错误类型。

| **常量** | **值** | **描述** |
| --- | --- | --- |
| QJsonParseError::NoError | 0 | 未发生错误 |
| QJsonParseError::UnterminatedObject | 1 | 对象不正确地终止以右花括号结束 |
| QJsonParseError::MissingNameSeparator | 2 | 分隔不同项的逗号丢失 |
| QJsonParseError::UnterminatedArray | 3 | 数组不正确地终止以右中括号结束 |
| QJsonParseError::MissingValueSeparator | 4 | 对象中分割 key/value 的冒号丢失 |
| QJsonParseError::IllegalValue | 5 | 值是非法的 |
| QJsonParseError::TerminationByNumber | 6 | 在解析数字时，输入流结束 |
| QJsonParseError::IllegalNumber | 7 | 数字格式不正确 |
| QJsonParseError::IllegalEscapeSequence | 8 | 在输入时，发生一个非法转义序列 |
| QJsonParseError::IllegalUTF8String | 9 | 在输入时，发生一个非法 UTF8 序列 |
| QJsonParseError::UnterminatedString | 10 | 字符串不是以引号结束 |
| QJsonParseError::MissingObject | 11 | 一个对象是预期的，但是不能被发现 |
| QJsonParseError::DeepNesting | 12 | 对解析器来说，JSON 文档嵌套太深 |
| QJsonParseError::DocumentTooLarge | 13 | 对解析器来说，JSON 文档太大 |
| QJsonParseError::GarbageAtEnd | 14 | 解析的文档在末尾处包含额外的乱码 |

# 简单的 JSON 对象

构造一个简单的 JSON 对象：

{

"Cross Platform": true,

"From": 1991,

"Name": "Qt"

}

生成比较简单，由于是一个对象，只需要用 QJsonObject 即可。

// 构建 JSON 对象

QJsonObject json;

json.insert("Name", "Qt");

json.insert("From", 1991);

json.insert("Cross Platform", true);

// 构建 JSON 文档

QJsonDocument document;

document.setObject(json);

QByteArray byteArray = document.toJson(QJsonDocument::Compact);

QString strJson(byteArray);

qDebug() << strJson;

解析如下：

QJsonParseError jsonError;

QJsonDocument doucment = QJsonDocument::fromJson(byteArray, &jsonError); // 转化为 JSON 文档

if (!doucment.isNull() && (jsonError.error == QJsonParseError::NoError)) { // 解析未发生错误

if (doucment.isObject()) { // JSON 文档为对象

QJsonObject object = doucment.object(); // 转化为对象

if (object.contains("Name")) { // 包含指定的 key

QJsonValue value = object.value("Name"); // 获取指定 key 对应的 value

if (value.isString()) { // 判断 value 是否为字符串

QString strName = value.toString(); // 将 value 转化为字符串

qDebug() << "Name : " << strName;

}

}

if (object.contains("From")) {

QJsonValue value = object.value("From");

if (value.isDouble()) {

int nFrom = value.toVariant().toInt();

qDebug() << "From : " << nFrom;

}

}

if (object.contains("Cross Platform")) {

QJsonValue value = object.value("Cross Platform");

if (value.isBool()) {

bool bCrossPlatform = value.toBool();

qDebug() << "CrossPlatform : " << bCrossPlatform;

}

}

}

}

**注意：**在转化为 QJsonDocument 后，首先需要根据 QJsonParseError 的值判定是否转化成功，然后在进行相应的转化解析。

# 简单的 JSON 数组

构造一个简单的 JSON 对象：

[

"Qt",

5.7,

true

]

生成比较简单，由于是一个数组，只需要用 QJsonArray 即可。

// 构建 JSON 数组

QJsonArray json;

json.append("Qt");

json.append(5.7);

json.append(true);

// 构建 JSON 文档

QJsonDocument document;

document.setArray(json);

QByteArray byteArray = document.toJson(QJsonDocument::Compact);

QString strJson(byteArray);

qDebug() << strJson;

需要注意的是，和上面不同的是，这里使用的是 QJsonDocument 的 setArray() 函数，因为是数组嘛！

解析如下：

QJsonParseError jsonError;

QJsonDocument doucment = QJsonDocument::fromJson(byteArray, &jsonError); // 转化为 JSON 文档

if (!doucment.isNull() && (jsonError.error == QJsonParseError::NoError)) { // 解析未发生错误

if (doucment.isArray()) { // JSON 文档为数组

QJsonArray array = doucment.array(); // 转化为数组

int nSize = array.size(); // 获取数组大小

for (int i = 0; i < nSize; ++i) { // 遍历数组

QJsonValue value = array.at(i);

if (value.type() == QJsonValue::String) {

QString strName = value.toString();

qDebug() << strName;

}

if (value.type() == QJsonValue::Double) {

double dVersion = value.toDouble();

qDebug() << dVersion;

}

if (value.type() == QJsonValue::Bool) {

bool bCrossPlatform = value.toBool();

qDebug() << bCrossPlatform;

}

}

}

}

和 JSON 对象类似，在遍历数组时，获取每个 value，首先需要判断 value 的类型（和 is\*\*\*() 函数类似，这里根据 type() 函数返回的枚举值来判断），然后再进行相应的转换。

# 复杂的 JSON

构造一个复杂的 JSON 对象：

{

"Company": "Digia",

"From": 1991,

"Name": "Qt",

"Page": {

"Developers": "https://www.qt.io/developers/",

"Download": "https://www.qt.io/download/",

"Home": "https://www.qt.io/"

},

"Version": [

4.8,

5.2,

5.7

]

}

包含了一个拥有五个“key/value 对”的对象，values 中的两个（Company、Name）是字符串，一个（From）是数字，一个（Page）是对象，一个（Version）是数组。

要生成这样一个复杂的 JSON 文档，需要分别构造对象和数组，然后将它们拼接起来：

// 构建 Json 数组 - Version

QJsonArray versionArray;

versionArray.append(4.8);

versionArray.append(5.2);

versionArray.append(5.7);

// 构建 Json 对象 - Page

QJsonObject pageObject;

pageObject.insert("Home", "https://www.qt.io/");

pageObject.insert("Download", "https://www.qt.io/download/");

pageObject.insert("Developers", "https://www.qt.io/developers/");

// 构建 Json 对象

QJsonObject json;

json.insert("Name", "Qt");

json.insert("Company", "Digia");

json.insert("From", 1991);

json.insert("Version", QJsonValue(versionArray));

json.insert("Page", QJsonValue(pageObject));

// 构建 Json 文档

QJsonDocument document;

document.setObject(json);

QByteArray byteArray = document.toJson(QJsonDocument::Compact);

QString strJson(byteArray);

qDebug() << strJson;

解析部分其实并没有看起来这么复杂，只要一步步搞明白对应的类型，然后进行相应转化即可。

QJsonParseError jsonError;

QJsonDocument doucment = QJsonDocument::fromJson(byteArray, &jsonError); // 转化为 JSON 文档

if (!doucment.isNull() && (jsonError.error == QJsonParseError::NoError)) { // 解析未发生错误

if (doucment.isObject()) { // JSON 文档为对象

QJsonObject object = doucment.object(); // 转化为对象

if (object.contains("Name")) {

QJsonValue value = object.value("Name");

if (value.isString()) {

QString strName = value.toString();

qDebug() << "Name : " << strName;

}

}

if (object.contains("Company")) {

QJsonValue value = object.value("Company");

if (value.isString()) {

QString strCompany = value.toString();

qDebug() << "Company : " << strCompany;

}

}

if (object.contains("From")) {

QJsonValue value = object.value("From");

if (value.isDouble()) {

int nFrom = value.toVariant().toInt();

qDebug() << "From : " << nFrom;

}

}

if (object.contains("Version")) {

QJsonValue value = object.value("Version");

if (value.isArray()) { // Version 的 value 是数组

QJsonArray array = value.toArray();

int nSize = array.size();

for (int i = 0; i < nSize; ++i) {

QJsonValue value = array.at(i);

if (value.isDouble()) {

double dVersion = value.toDouble();

qDebug() << "Version : " << dVersion;

}

}

}

}

if (object.contains("Page")) {

QJsonValue value = object.value("Page");

if (value.isObject()) { // Page 的 value 是对象

QJsonObject obj = value.toObject();

if (obj.contains("Home")) {

QJsonValue value = obj.value("Home");

if (value.isString()) {

QString strHome = value.toString();

qDebug() << "Home : " << strHome;

}

}

if (obj.contains("Download")) {

QJsonValue value = obj.value("Download");

if (value.isString()) {

QString strDownload = value.toString();

qDebug() << "Download : " << strDownload;

}

}

if (obj.contains("Developers")) {

QJsonValue value = obj.value("Developers");

if (value.isString()) {

QString strDevelopers = value.toString();

qDebug() << "Developers : " << strDevelopers;

}

}

}

}

}

}

基本的用法就这些，比较简单，细节很关键，建议在处理的过程中启用严格模式，例如：先通过 QJsonParseError::NoError 判断转化 JSON 文档无误，再进行解析。在解析过程中，先判断 QJsonValue 是否为对应的类型如 isObject()，再通过 toObject() 转化。