

## 《Qt 6 C++开发指南》示例源程序——qmake V2.0 for Qt6.5.1

《Qt 6 C++开发指南》全书示例代码 V1.0 是针对 Qt 6.2.3 测试的。

Qt 6 的第二个 LTS 版本是 Qt 6.5。将本书 qmake V1.0 版本的代码使用 Qt 6.5.1 进行编译测试，有少数的项目有编译警告或错误，有一个项目代码有 bug，所做修改整理为本文档。



王维波

2023-12-17 发布

## 第 2 章 GUI 程序设计基础

所有示例项目编译和运行没有任何问题，代码无需任何修改

## 第 3 章 Qt 框架功能概述

所有示例项目编译和运行没有任何问题，代码无需任何修改

## 第 4 章 常用界面组件的使用

### 4.3 QString 字符串操作

#### 示例项目 samp4\_02

编译时提示下面一段代码中的 `QString::count()` 这个函数已经被淘汰，不要再使用了（`deprecated`），建议使用 `size()`、`length()` 函数。

```
void Widget::on_pushButton_4_clicked()
{ //count(),length(),size() 函数
    ui->plainTextEdit->appendPlainText("\n===size(),count(),length() 函数测试");

    QString str1=ui->comboBox1->currentText();
    ui->plainTextEdit->appendPlainText(str1);

    int N=str1.size();
    ui->plainTextEdit->appendPlainText(QString::asprintf("size()=%d",N));

    N=str1.count(); //在 Qt 6.5 中，QString::count() 函数是 deprecated
    ui->plainTextEdit->appendPlainText(QString::asprintf("count()=%d",N));

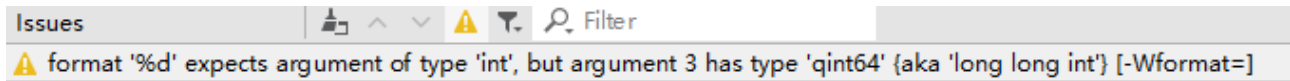
    N=str1.length();
    ui->plainTextEdit->appendPlainText(QString::asprintf("length()=%d",N));
}
```

这段代码就是为了测试这 3 个函数，所以无需修改代码，在实际编程中不要使用 `QString::count()` 这个函数即可。

## 4.7 日期时间数据

### 示例项目 samp4\_07

编译时提示警告，警告信息如下：



是 107 行的代码导致的警告，原来的代码如下：

```
qDebug("Days between DT2 and DT1= %d",DT2.daysTo(DT1)); //Qt 6.2 中的代码，编译有警告
```

是因为 qDebug()中的格式字符“%d”与函数 QDate::daysTo()的返回类型不一致。将代码改成如下的，就可以解除警告，且测试运行输出数据正常。

```
qDebug("Days between DT2 and DT1= %lld",DT2.daysTo(DT1)); //Qt 6.5 中的代码
```

## 第 5 章 模型/视图结构

所有示例项目编译和运行没有任何问题，代码无需任何修改

## 第 6 章 事件处理

所有示例项目编译和运行没有任何问题，代码无需任何修改

## 第 7 章 对话框和多窗口程序设计

所有示例项目编译和运行没有任何问题，代码无需任何修改

## 第 8 章 文件系统操作和文件读写

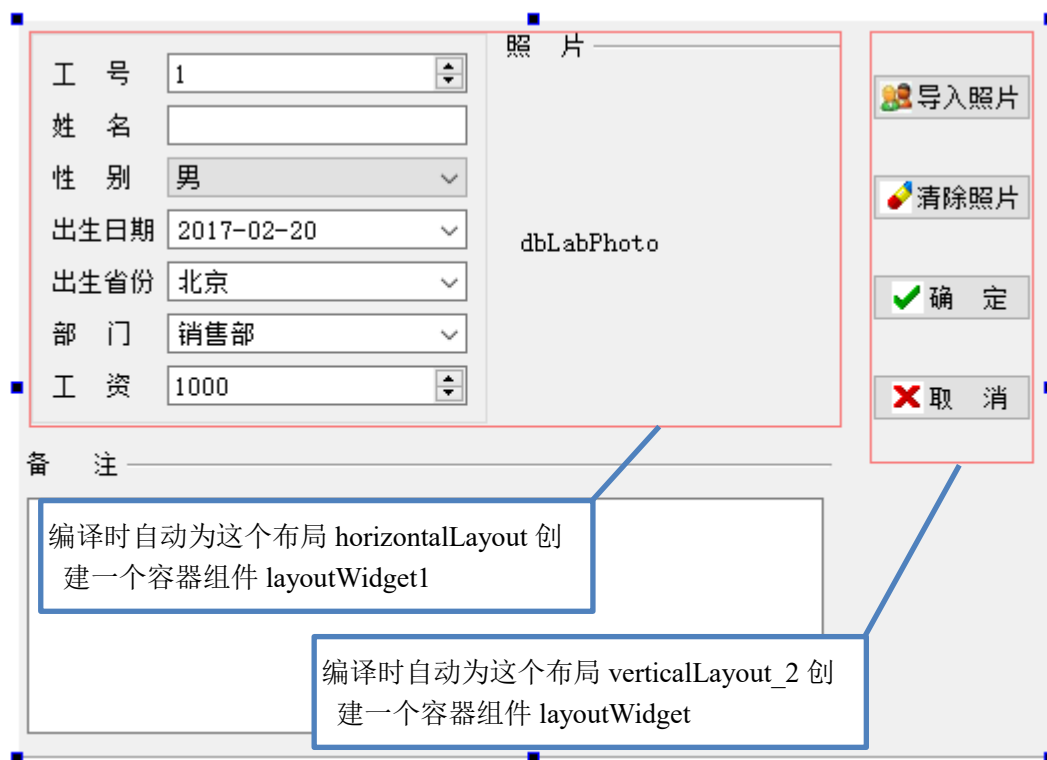
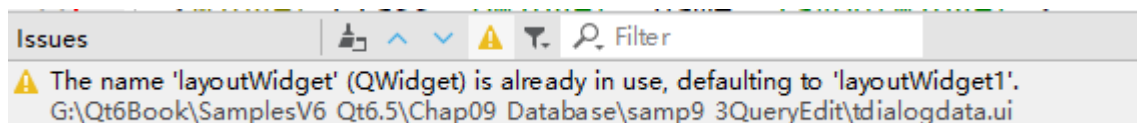
所有示例项目编译和运行没有任何问题，代码无需任何修改

# 第 9 章 数据库

## 示例项目 samp9\_3

### 问题现象和原因 (1)

编译时提示如下的警告，但是不影响运行。



这是因为在设计 `tdialogdata.ui` 的界面时，我们使用了两个布局，即上图中的 `horizontalLayout` 和 `verticalLayout_2`，这两个布局没有容器组件，所以它们显示为红色线框。在 UIC 编译此 UI 文件时，会自动为没有容器组件的布局创建一个 `QWidget` 类型的容器组件，在文件 `ui_tdialogdata.h` 中可以看到如下的代码，用于为布局 `verticalLayout_2` 创建容器组件 `layoutWidget`:

```
layoutWidget = new QWidget(TDialogData);
layoutWidget->setObjectName(QString::fromUtf8("layoutWidget"));
layoutWidget->setGeometry(QRect(425, 5, 82, 216));
verticalLayout_2 = new QVBoxLayout(layoutWidget);
verticalLayout_2->setObjectName(QString::fromUtf8("verticalLayout_2"));
verticalLayout_2->setContentsMargins(0, 0, 0, 0);
```

```
btnSetPhoto = new QPushButton(layoutWidget);
```

同样的，为 horizontalLayout 创建容器组件 layoutWidget1 的代码如下：

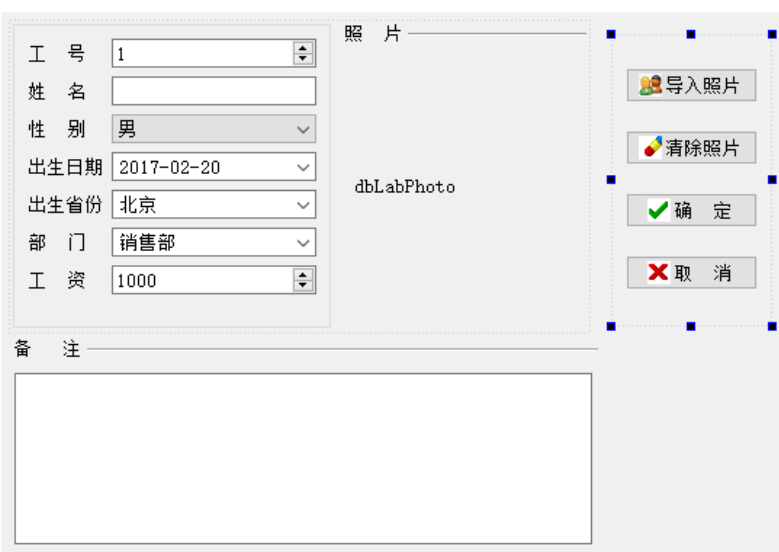
```
layoutWidget1 = new QWidget(TDialogData);
layoutWidget1->setObjectName(QString::fromUtf8("layoutWidget1"));
layoutWidget1->setGeometry(QRect(5, 5, 406, 198));
horizontalLayout = new QHBoxLayout(layoutWidget1);
horizontalLayout->setObjectName(QString::fromUtf8("horizontalLayout"));
horizontalLayout->setContentsMargins(0, 0, 0, 0);
groupBox_2 = new QGroupBox(layoutWidget1);
```

而在 UI 设计器的 object browser 窗口里是没有容器组件 layoutWidget 和 layoutWidget1 的，见下图。

Object	Class
TDialogData	QDialog
groupBox_3	QGroupBox
editMemo	QPlainTextEdit
horizontalLayout	QHBoxLayout
groupBox	QGroupBox
groupBox_2	QGroupBox
verticalLayout_2	QVBoxLayout
btnClearPhoto	QPushButton
btnClose	QPushButton
btnOK	QPushButton
btnSetPhoto	QPushButton

## 修改方法

修改的方法是在 UI 可视化设计时，打散（break）两个独立的没有父容器的布局 verticalLayout\_2 和 horizontalLayout，使用 QFrame 作为其父容器进行布局。修改后的界面如下图所示。这样修改后编译就不会出现警告了。



Object	Class
TDialogData	QDialog
frame	QFrame
btnClearPhoto	QPushButton
btnClose	QPushButton
btnOK	QPushButton
btnSetPhoto	QPushButton
frame_2	QFrame
groupBox	QGroupBox
groupBox_2	QGroupBox
groupBox_3	QGroupBox
editMemo	QPlainTextEdit

经验和提示：在可视化设计 UI 时，不要使用没有父容器的布局，一定要在一个父容器内布局。

## 问题现象和原因 (2)

编译时提示 2 处代码错误，如下图



其中的一行代码有错误

```
qryModel->query().exec(); //数据模型重新查询数据，更新 tableView 显示
```

这是因为 Qt 6.5 中 QSqlQueryModel::query()函数的输出形式变化了，下面是 Qt 6.5 中的定义

```
const QSqlQuery &query() const
```

而 Qt 6.2 中的定义如下：

```
QSqlQuery query() const
```

## 修改方法

要改正此错误，需要将错误的一行代码替换为如下的 2 行代码即可。

```
QString str= qryModel->query().executedQuery(); //获取执行过的 SQL 语句  
qryModel->setQuery(str); //重新执行 SQL 语句
```

# 第 10 章 绘图

所有示例项目编译和运行没有任何问题，代码无需任何修改

# 第 11 章 自定义插件和库

## 11.2 设计和使用 Qt Designer Widget

### 示例 BatteryPlugin

创建 Qt Custom Designer Widget 项目时，生成的项目类型自动使用 qmake 构建系统，不能使用 CMake 项目。所以，示例项目 BatteryPlugin 就是 qmake 项目，与书中介绍的完全一样。

但是要注意，创建此项目时开发套件(kit)只能选择 MSVC2019 64bit，也就是必须与构建 Qt Creator 的开发套件一致。

### 示例 BatteryUser

创建这个项目时可以选择使用 CMake 构建系统，使用时要注意以下事项。

- (1) 只能使用 MSVC2019 64bit 开发套件。
- (2) 在项目的根目录下创建文件夹 include，将项目 BatteryPlugin 中的文件 tpbattery.h 和 tpbatteryplugin.lib 复制到此文件夹下，如图 11-1 所示。

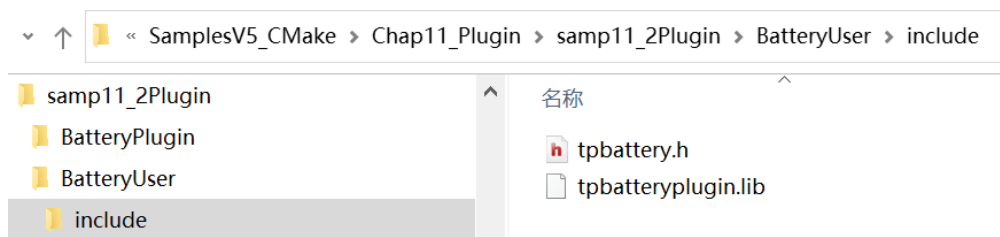


图 11-1

- (3) 编辑文件 CMakeLists.txt，添加头文件和库文件的搜索路径，也就是项目源程序目录下的子文件夹 include

```
INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include) #设置头文件搜索路径
LINK_DIRECTORIES(${PROJECT_SOURCE_DIR}/include) #设置需要连接的库文件搜索路径
```

在文件 CMakeLists.txt 中再添加需要连接的动态库的名称，即 tpbatteryplugin.dll

```
target_link_libraries(BatteryUser PUBLIC tpbatteryplugin)    #添加需要链接的动态库
```

- (4) 构建此项目生成可执行文件 `BatteryUser.exe`，将示例项目 `BatteryPlugin` 中生成的动态链接库文件 `tpbatteryplugin.dll` 复制到可执行文件 `BatteryUser.exe` 所在的文件夹，才可以正常运行项目，否则提示找不到 DLL 文件。

注意，这些修改是针对 `Release` 模式的，使用的都是 `Release` 版本的 `lib` 和 `dll` 文件。

## 第 12 章 Qt Charts

本章所有示例项目编译和运行没有任何问题，代码无任何修改

## 第 13 章 Qt Data Visualization

本章所有示例项目编译和运行没有任何问题，代码无任何修改



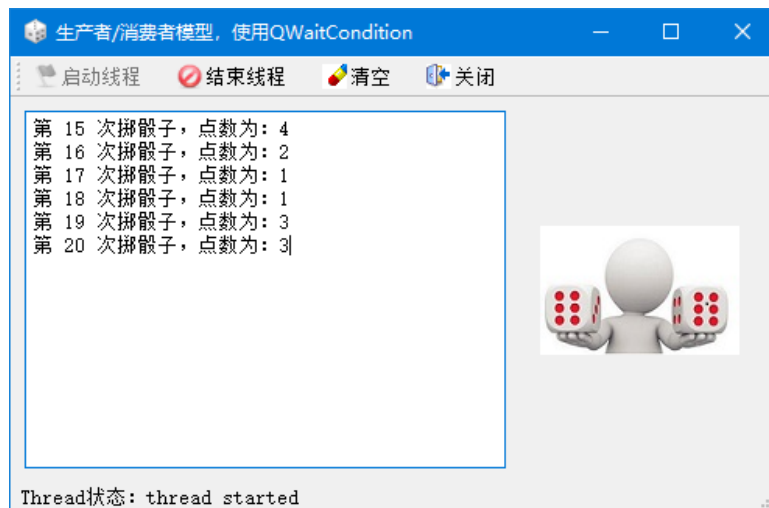
# 第 14 章 多线程

## 示例 samp14\_3

### 问题现象和原因

这个示例编译时没有警告和错误，但是启动线程后，文本框中有输出，但是右边的骰子点数的图片不变，在 Qt Creator 的 Application Output 窗口输出如下的提示信息

```
qt.core.qobject.connect: QObject::connect: Cannot queue arguments of type 'QString&'
(Make sure 'QString&' is registered using qRegisterMetaType().)
```



问题：启动线程后，文本框内点数变化，但是右边的图片不变

从提示信息可见是使用 connect()函数时，使用的参数类型 `QString&` 出现问题。查看项目内的代码，在 MainWindow 类中自定义的槽函数如下，只有 do\_newPicture()使用了 `QString&`类型的参数。

```
private slots:
    void do_newPicture(QString &picName);
```

在 MainWindow 类的构造函数中有如下的 connect()函数语句：

```
connect(threadPic, &TPictureThread::newPicture, this, &MainWindow::do_newPicture);
```

threadPic 的信号 newPicture()与 MainWindow 的 do\_newPicture()槽函数关联。threadPic 是 TPictureThread 类的对象实例。TPictureThread 类中的信号 newPicture()定义如下：

```
class TPictureThread : public QThread
{
    Q_OBJECT
protected:
    void run(); //线程的任务函数
public:
    explicit TPictureThread(QObject *parent = nullptr);
```

```
signals:
    void newPicture(QString &picName);
};
```

## 解决方法

既然 QString& 是 QObject 中未注册的类型，就干脆不使用 QString& 类型参数，而直接使用 QString 类型参数。将 TPictureThread 类中的信号 newPicture() 的参数类型改为 QString 类型，即修改为如下的代码：

```
class TPictureThread : public QThread
{
    Q_OBJECT
protected:
    void run();        //线程的任务函数
public:
    explicit TPictureThread(QObject *parent = nullptr);
signals:
    void newPicture(QString picName);
};
```

把 MainWindow 类中的自定义槽函数 do\_newPicture() 的函数参数类型也修改为 QString，即（注意修改 mainwindow.c 中的相应代码）

```
void do_newPicture(QString picName);
```

MainWindow 类的构造函数中的 connect() 函数语句无需修改。这样修改后，运行时就可以正常显示骰子点数的图片了。

# 第 15 章 网络

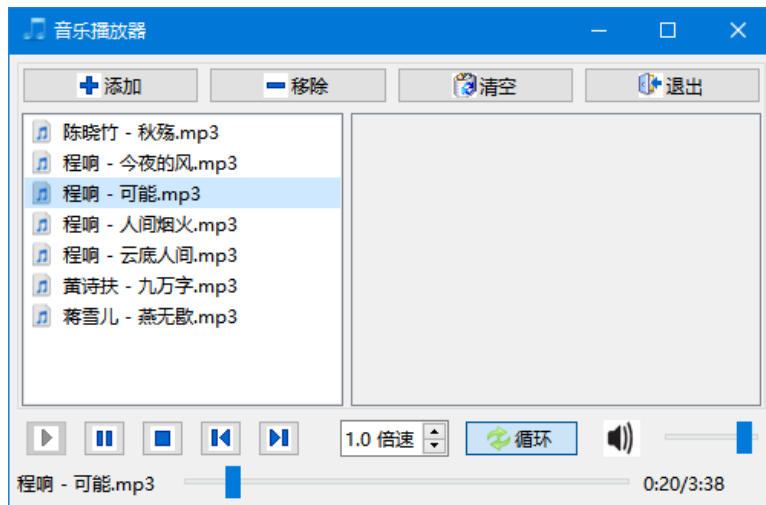
本章所有示例项目编译和运行没有任何问题，代码无任何修改

# 第 16 章 多媒体

## 示例 samp16\_1

### 问题现象和原因

项目编译时没有错误，运行时能播放音乐，但是不显示歌曲的封面图片。如图



运行时无法显示歌曲的封面图片

显示歌曲的封面图片是在槽函数 `do_metaDataChanged()` 中实现的，设置断点跟踪调试，显示变量 `metaImg` 是 `invalid`，也就是没有图片数据。查看了 Qt 帮助中 `QMediaMetaData` 类的资料，也尝试做了一些修改，但还是无法获取图片。

但是同样的代码使用 Qt 6.2 编译就可以正常显示歌曲封面图片，所以这可能是 Qt 6.5 中的一个 bug，暂时无法修改。

```
112 void MainWindow::do_metaDataChanged()
113 { //元数据变化时执行，显示歌曲图片
114     QMediaMetaData metaData=player->metaData(); //元数据对象
115     QVariant metaImg= metaData.value(QMediaMetaData::ThumbnailImage); //获取ThumbnailImage元数据
116     if (metaImg.isValid())
117     {
118         QImage metaImg (invalid) QVariant (invalid) >(); //QVariant转换为QImage
119         QPixmap musicPixmp= QPixmap::fromImage(img);
120         if (ui->scrollArea->width() < musicPixmp.width())
121             ui->labPic->setPixmap(musicPixmp.scaledToWidth(ui->scrollArea->width()-30));
122         else
123             ui->labPic->setPixmap(musicPixmp);
124     }
125     else
126         ui->labPic->clear();
127 }
```

断点跟踪时显示 `metaImg` 是 `invalid`

## 解决方法

暂时没有找到解决方法。

## 示例 samp16\_3

### 问题现象和原因

选择文件格式后，录音保存的文件名后缀总是不对，会再添加一个后缀，例如再添加一个后缀.wmv。

原来的代码有错误，槽函数 on\_actRecord\_triggered()的代码有问题。

```
QMediaFormat mediaFormat;
QVariant var=ui->comboCodec->itemData(ui->comboCodec->currentIndex());
QMediaFormat::FileFormat fileFormat= var.value<QMediaFormat::FileFormat>();
mediaFormat.setFileFormat(fileFormat); //设置文件格式

var=ui->comboFileFormat->itemData(ui->comboFileFormat->currentIndex());
QMediaFormat::AudioCodec audioCodec =var.value<QMediaFormat::AudioCodec>();
mediaFormat.setAudioCodec(audioCodec); //设置编码格式

recorder->setMediaFormat(mediaFormat); //设置 mediaFormat
```

comboCodec 是“音频编码”组合框，但是其值被用来设置文件格式；comboFileFormat 是“文件格式”组合框，但是其值被用来设置编码格式。

## 解决方法

修改 actRecord 的槽函数 on\_actRecord\_triggered()代码，改正上述有问题的代码，修改后的代码如下：

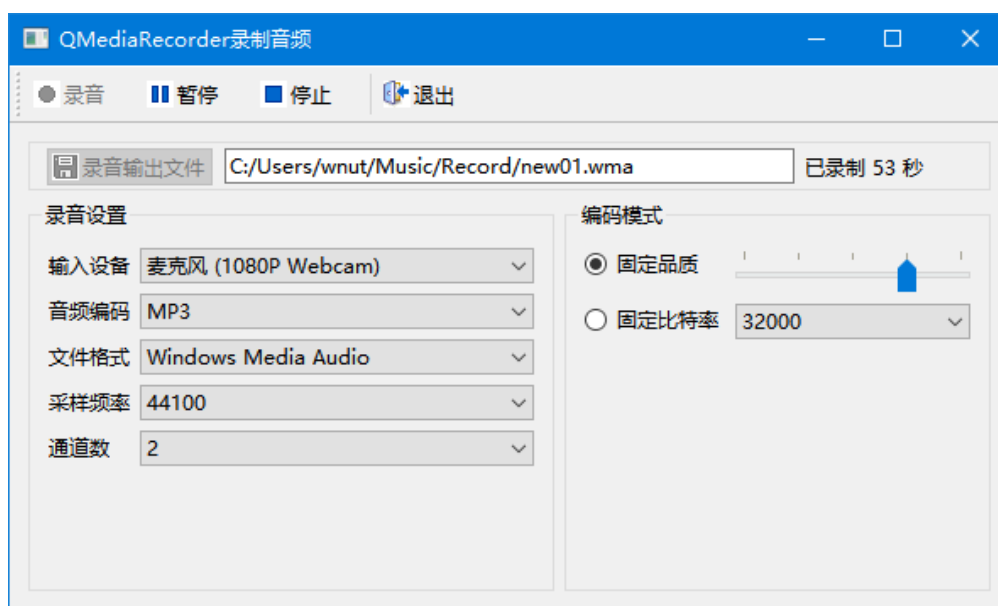
```
QMediaFormat mediaFormat;
QVariant var=ui->comboCodec->itemData(ui->comboCodec->currentIndex());
QMediaFormat::AudioCodec audioCodec =var.value<QMediaFormat::AudioCodec>();
mediaFormat.setAudioCodec(audioCodec); //设置编码格式

var=ui->comboFileFormat->itemData(ui->comboFileFormat->currentIndex());
QMediaFormat::FileFormat fileFormat= var.value<QMediaFormat::FileFormat>();
mediaFormat.setFileFormat(fileFormat); //设置文件格式

recorder->setMediaFormat(mediaFormat); //设置 mediaFormat
```

音频编码中没有 Windows Media Audio 格式选项，但是音频编码选择 MP3，文件格式选择 Windows

Media Audio 是可以录制的。



音频编码选择 MP3，文件格式选择 Windows Media Audio 是可以录制的

## 第 17 章 串口编程

本章所有示例项目编译和运行没有任何问题，代码无任何修改

## 第 18 章 其他工具软件和技术

### 示例 samp18\_3Deploy

MusicPlayer.exe 是 samp16\_1 用 Qt 6.5.1 MinGW 64-bit 套件编译的，生成这个程序的发布文件使用下面的指令

```
D:\Qt\6.5.1\mingw_64\bin\windeployqt --no-quick-import --no-translations
--no-virtualkeyboard --no-system-d3d-compiler MusicPlayer.exe
```

注意，指令中去掉了--release，因为运行指令时会报错。