# 0 前言说明

1. **下载说明：由于可执行文件比较大，如有需要请到网盘下载。**
2. **网店地址：https://shop244026315.taobao.com/**
3. **联系方式：QQ（517216493）微信（feiyangqingyun）推荐加微信。**
4. **以下项目已经全部支持Qt4/5/6所有版本以及后续版本**
5. 监控作品体验：https://pan.baidu.com/s/1d7TH_GEYl5nOecuNlWJJ7g 提取码：01jf
6. 其他作品体验：https://pan.baidu.com/s/1ZxG-oyUKe286LPMPxOrO2A 提取码：o05q
7. 监控系统在线文档：https://feiyangqingyun.gitee.io/QWidgetDemo/video_system/
8. 大屏系统在线文档：https://feiyangqingyun.gitee.io/QWidgetDemo/bigscreen/
9. 物联网系统在线文档：https://feiyangqingyun.gitee.io/QWidgetDemo/iotsystem/

# 1 开发经验

## 01：001-010

1. 当编译发现大量错误的时候，从第一个看起，一个一个的解决，不要急着去看下一个错误，往往后面的错误都是由于前面的错误引起的，第一个解决后很可能都解决了。比如我们可能就写错了一行代码，编译提示几百个错误，你只要把这一行纠正了，其他错误也就没了。
2. 定时器是个好东西，学会好使用它，有时候用QTimer::singleShot单次定时器和QMetaObject::invokeMethod可以解决意想不到的问题。比如在窗体初始化的时候加载一个耗时的操作，很容易卡主界面的显示，要在加载完以后才会显示界面，这就导致了体验很卡不友好的感觉，此时你可以将耗时的加载（有时候这些加载又必须在主线程，比如用QStackWidget堆栈窗体加载一些子窗体），延时或者异步进行加载，这样就会在界面显示后去执行，而不是卡住主界面。

```
//异步执行load函数
QMetaObject::invokeMethod(this, "load", Qt::QueuedConnection);
//延时10毫秒执行load函数
QTimer::singleShot(10, this, SLOT(load()));
```

3. 默认QtCreator是单线程编译，可能设计之初考虑到尽量不过多占用系统资源，而现在的电脑都是多核心的，默认msvc编译器是多线程编译的不需要手动设置，而对于其他编译器，需要手动设置才行。

- 方法一：在每个项目的构建设置中（可以勾选一个 shadow build 的页面地方）的build步骤，make arguments增加一行 -j16 即可，此设置会保存在pro.user文件中，一旦删除就需要重新设置，不建议此方法；
- 方法二：在构建套件的环境中增加，工具->选项->构建套件(kits)->选中一个构建套件->environment->右侧change按钮->打开的输入框中填入 MAKEFLAGS=-j4 ， 这样就可以不用每次设置多线程编译，只要是应用该构件套件的项目都会加上这个编译参数；
- 注意：-j后面接的是电脑的核心数，写多了不会有效果，要自己看下电脑的参数，或者填个-j4就行，毕竟现在电脑4核心应该是最基本的；
- 大概从2019年开始的新版本的QtCreator默认已经会根据电脑的核心自动设置多线程编译，比如识别到你的电脑是16核心的就会默认设置-j16参数进行编译；

4. 如果你想顺利用QtCreator部署安卓程序，首先你要在 Android Studio 里面配置成功，编译一个程序能够在手机上或者模拟器中跑起来，把坑全部趟平。
5. 很多时候找到Qt对应封装的方法后，记得多看看该函数的重载，多个参数的，你会发现不一样的世界，有时候会恍然大悟，原来Qt已经帮我们封装好了，比如QString、QColor的重载参数极其丰

富，很多你做梦都想要的功能就在里面。

6. 可以在pro文件中写上版本号、程序图标、产品名称、版权所有、文件说明等信息（Qt5才支持），其实在windows上就是qmake的时候会自动将此信息转换成rc文件。对于早期的Qt4版本你可以手动写rc文件实现。

```
#程序版本
VERSION  = 2025.10.01
#程序图标
RC_ICONS = main.ico
#产品名称
QMAKE_TARGET_PRODUCT = quc
#版权所有
QMAKE_TARGET_COPYRIGHT = feiyangqingyun
#文件说明
QMAKE_TARGET_DESCRIPTION = QQ: 517216493  WX: feiyangqingyun
```

7. 管理员运行程序，限定在MSVC编译器，在项目pro文件中增加如下代码。

```
QMAKE_LFLAGS += /MANIFESTUAC:"level='requireAdministrator' uiAccess='false'"
#以管理员运行
QMAKE_LFLAGS += /SUBSYSTEM:WINDOWS,"5.01" #VS2013 在XP运行
```

8. 运行文件附带调试输出窗口，这个非常有用，很多时候当我们发布程序阶段，我们会遇到程序双击无法运行也不报错提示（开发机器上一切正常），都不知道发生了什么，甚至任务管理器可以看到运行了但是没有界面弹出来，此时就需要在项目的pro文件中加上一行CONFIG += console，带界面的程序也会自动弹出调试窗口打印输出信息，方便找问题，一般没法正常运行的程序都会打印一些提示信息缺啥之类的。

```
TEMPLATE    = app
MOC_DIR     = temp/moc
RCC_DIR     = temp/rcc
UI_DIR      = temp/ui
OBJECTS_DIR = temp/obj
#就是下面这行用来设置运行文件附带调试输出窗口
CONFIG      += console
```

9. 绘制平铺背景QPainter::drawTiledPixmap，绘制圆角矩形QPainter::drawRoundedRect()，而不是QPainter::drawRoundRect()，这两个函数非常容易搞混。
10. 指定控件移除旧的样式。

```
//移除原有样式
style()->unpolish(ui->btn);
//必须要有下面这行不然还是不会卸载
ui->btn->setStyleSheet("");
//重新设置新的该控件的样式。
style()->polish(ui->btn);
```

## 02：011-020

11. 获取类的属性和方法

```
//拿到控件元对象
const QMetaObject *metaObject = widget->metaObject();

//所有属性的数量
int propertyCount = metaObject->propertyCount();
//propertyOffset是自定义的属性开始的位置
int propertyOffset = metaObject->propertyOffset();
//循环取出控件的自定义属性, int i = 0 表示所有属性
for (int i = propertyOffset; i < propertyCount; ++i) {
    QMetaProperty metaProperty = metaObject->property(i);
    const char *name = metaProperty.name();
    const char *type = metaProperty.typeName();
    QVariant value = widget->property(name);
    qDebug() << name << type << value;
}

//所有方法的数量
int methodCount = metaObject->methodCount();
//methodOffset是自定义的方法开始的位置
int methodOffset = metaObject->methodOffset();
//循环取出控件的自定义方法, int i = 0 表示所有方法
for (int i = methodOffset; i < methodCount; ++i) {
    QMetaMethod metaMethod = metaObject->method(i);
    const char *name = metaMethod.name();
    const char *type = metaMethod.typeName();
    qDebug() << name << type;
}
```

12. Qt内置图标封装在QStyle中，大概七十多个图标，可以直接拿来用。

```
SP_TitleBarMenuButton,
SP_TitleBarMinButton,
SP_TitleBarMaxButton,
SP_TitleBarCloseButton,
SP_MessageBoxInformation,
SP_MessageBoxWarning,
SP_MessageBoxCritical,
SP_MessageBoxQuestion,
...
//下面这样取出来使用就行
QPixmap pixmap = this->style()->standardPixmap(QStyle::SP_TitleBarMenuButton);
ui->label->setPixmap(pixmap);
```

13. 根据操作系统位数判断加载

```
win32 {
    contains(DEFINES, WIN64) {
        DESTDIR = $$PWD/../bin64
    } else {
        DESTDIR = $$PWD/../bin32
    }
}
```

14. Qt5增强了很多安全性验证，如果出现setGeometry: Unable to set geometry，请将该控件的可见移到加入布局之后。
15. 可以将控件A添加到布局，然后控件B设置该布局，这种灵活性提高了控件的组合度，比如可以在文本框左侧右侧增加一个搜索按钮，按钮设置图标即可。

```
QPushButton *btn = new QPushButton;
btn->resize(30, ui->lineEdit->height());
QHBoxLayout *layout = new QHBoxLayout(ui->lineEdit);
layout->setMargin(0);
layout->addStretch();
layout->addWidget(btn);
```

16. 对QLCDNumber控件设置样式，需要将QLCDNumber的segmentstyle设置为flat，不然你会发现没效果。
17. 巧妙的使用 findChildren 可以查找该控件下的所有子控件。 findChild 为查找单个。

```
//查找指定类名objectName的控件
QList<QWidget *> widgets = fatherWidget.findChildren<QWidget *>
("widgetname");
//查找所有QPushButton
QList<QPushButton *> allPButtons = fatherWidget.findChildren<QPushButton *>
();
//查找一级子控件,不然会一直遍历所有子控件
QList<QPushButton *> childButtons = fatherWidget.findChildren<QPushButton *>
(QString(), Qt::FindDirectChildrenOnly);
```

18. 巧妙的使用inherits判断是否属于某种类。

```
QTimer *timer = new QTimer;          // QTimer inherits QObject
timer->inherits("QTimer");           // returns true
timer->inherits("QObject");          // returns true
timer->inherits("QAbstractButton"); // returns false
```

19. 使用弱属性机制，可以存储临时的值用于传递判断。可以通过widget->dynamicPropertyNames()列出所有弱属性名称，然后通过widget->property("name")取出对应的弱属性的值。
20. 在开发时, 无论是出于维护的便捷性, 还是节省内存资源的考虑, 都应该有一个 qss 文件来存放所有的样式表, 而不应该将 setStyleSheet 写的到处都是。如果是初学阶段或者测试阶段可以直接UI上右键设置样式表，正式项目还是建议统一到一个qss样式表文件比较好，统一管理。

## 03：021-030

21. 如果出现Z-order assignment: is not a valid widget.错误提示，用记事本打开对应的ui文件，找到为空的地方，删除即可。
22. 善于利用QComboBox的addItem的第二个参数设置用户数据，可以实现很多效果，使用itemData取出来。特别注意的是第二个参数是QVariant类型，这就不要太灵活了，意味着可以附带万能的数据比如结构体，这样就可以带一堆数据了，而不是一个数据。比如下拉框选择学号，对应元素可以附带该学生的姓名、班级、成绩等。很多人以为只能附带QString、int之类的数据，因为通常的用法也是那两种。

```
QStringList listVideoOpenInterval, listVideoOpenIntervalx;
listVideoOpenInterval << "0.0 秒" << "0.1 秒" << "0.3 秒" << "0.5 秒" << "1.0 秒" << "2.0 秒";
listVideoOpenIntervalx << "0" << "100" << "300" << "500" << "1000" << "2000";
for (int i = 0; i < listVideoOpenInterval.count(); ++i) {
    ui->cboxVideoOpenInterval->addItem(listVideoOpenInterval.at(i), listVideoOpenIntervalx.at(i));
}
//取出对应的值
int indexVideoOpenInterval = ui->cboxVideoOpenInterval->currentIndex();
indexVideoOpenInterval = ui->cboxVideoOpenInterval->itemData(indexVideoOpenInterval).toInt();
```

23. 如果用了webengine模块，发布程序的时候带上QtWebEngineProcess.exe、translations文件夹、resources文件夹，不然无法正常运行。
24. 在MFC程序或者VB/C#等窗体程序中，每个控件都有一个句柄，而且用句柄工具移过去会自动识别，但是在Qt程序中默认Qt是一个窗体一个句柄，如果要让每个控件都拥有独立的句柄，在main函数中要做如下设置。

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    a.setAttribute(Qt::AA_NativeWindows);
}
```

25. Qt编写的Android程序防止程序被关闭。

```
#if defined(Q_OS_ANDROID)
QAndroidService a(argc, argv);
return a.exec()
#else
QApplication a(argc, argv);
return a.exec();
#endif
```

26. 可以对整体的指示器设置样式，而不需要单独对每个控件的指示器设置，

```
*::down-arrow{}
*::menu-indicator{}
*::up-arrow:disabled{}
*::up-arrow:off{}
```

27. 可以指定位置设置背景图片。

```
QMainWindow > .QWidget {
    background-color: gainsboro;
    background-image: url(:/images/xxoo.png);
    background-position: top right;
    background-repeat: no-repeat
}
```

28. 嵌入式linux运行Qt程序

```
//Qt4写法
./HelloQt -qws &

//Qt5写法 xcb 可以改成 linuxfb eglfs vnc wayland 等,有哪个就用哪个挨个测试
./HelloQt --platform xcb
./HelloQt --platform linuxfb
./HelloQt --platform wayland
```

29. 如果发现QtCreator中的构建套件不正常了或者坏了（比如不能正确识别环境中的qmake或者编译器、打开项目不能正常生成影子构建目录），请找到两个目录（C:\Users\Administrator\AppData\Local\QtProject、C:\Users\Administrator\AppData\Roaming\QtProject）删除即可，删除后重新打开QtCreator进行构建套件的配置就行。
30. QMediaPlayer是个壳（也可以叫框架），依赖本地解码器，视频这块默认基本上就播放个MP4甚至连MP4都不能播放，如果要支持其他格式需要下载k-lite或者LAV Filters安装即可（k-lite或者LAV Filters是指windows上的，其他系统上自行搜索，貌似嵌入式linux上依赖GStreamer，并未完整验证，报错提示 Your GStreamer installation is missing a plug-in，需要命令安装 sudo apt-get install ubuntu-restricted-extras）。如果需要做功能强劲的播放器，初学者建议用vlc、mpv，终极万能大法用ffmpeg（解码出来的视频可以用QOpenGLWidget走GPU绘制或者转成QImage绘制，音频数据可以用QAudioOutput播放）。

## 04：031-040

31. 判断编译器类型、编译器版本、操作系统。

```
//GCC编译器
#ifdef __GNUC__
#if __GNUC__ >= 3   // GCC3.0  以上

//MSVC编译器
#ifdef _MSC_VER
#if _MSC_VER >=1000 // VC++4.0 以上
#if _MSC_VER >=1100 // VC++5.0 以上
#if _MSC_VER >=1200 // VC++6.0 以上
#if _MSC_VER >=1300 // VC2003  以上
#if _MSC_VER >=1400 // VC2005  以上
#if _MSC_VER >=1500 // VC2008  以上
#if _MSC_VER >=1600 // VC2010  以上
#if _MSC_VER >=1700 // VC2012  以上
#if _MSC_VER >=1800 // VC2013  以上
#if _MSC_VER >=1900 // VC2015  以上

```

```
//Visual Studio版本与MSVC版本号的对应关系
MSC    1.0   _MSC_VER == 100
MSC    2.0   _MSC_VER == 200
MSC    3.0   _MSC_VER == 300
MSC    4.0   _MSC_VER == 400
MSC    5.0   _MSC_VER == 500
MSC    6.0   _MSC_VER == 600
MSC    7.0   _MSC_VER == 700
MSVC++ 1.0   _MSC_VER == 800
MSVC++ 2.0   _MSC_VER == 900
MSVC++ 4.0   _MSC_VER == 1000 (Developer Studio 4.0)
MSVC++ 4.2   _MSC_VER == 1020 (Developer Studio 4.2)
MSVC++ 5.0   _MSC_VER == 1100 (Visual Studio 97 version 5.0)
MSVC++ 6.0   _MSC_VER == 1200 (Visual Studio 6.0 version 6.0)
MSVC++ 7.0   _MSC_VER == 1300 (Visual Studio .NET 2002 version 7.0)
MSVC++ 7.1   _MSC_VER == 1310 (Visual Studio .NET 2003 version 7.1)
MSVC++ 8.0   _MSC_VER == 1400 (Visual Studio 2005 version 8.0)
MSVC++ 9.0   _MSC_VER == 1500 (Visual Studio 2008 version 9.0)
MSVC++ 10.0  _MSC_VER == 1600 (Visual Studio 2010 version 10.0)
MSVC++ 11.0  _MSC_VER == 1700 (Visual Studio 2012 version 11.0)
MSVC++ 12.0  _MSC_VER == 1800 (Visual Studio 2013 version 12.0)
MSVC++ 14.0  _MSC_VER == 1900 (Visual Studio 2015 version 14.0)
MSVC++ 14.1  _MSC_VER == 1910 (Visual Studio 2017 version 15.0)
MSVC++ 14.11 _MSC_VER == 1911 (Visual Studio 2017 version 15.3)
MSVC++ 14.12 _MSC_VER == 1912 (Visual Studio 2017 version 15.5)
MSVC++ 14.13 _MSC_VER == 1913 (Visual Studio 2017 version 15.6)
MSVC++ 14.14 _MSC_VER == 1914 (Visual Studio 2017 version 15.7)
MSVC++ 14.15 _MSC_VER == 1915 (Visual Studio 2017 version 15.8)
MSVC++ 14.16 _MSC_VER == 1916 (Visual Studio 2017 version 15.9)
MSVC++ 14.2  _MSC_VER == 1920 (Visual Studio 2019 Version 16.0)
MSVC++ 14.21 _MSC_VER == 1921 (Visual Studio 2019 Version 16.1)
MSVC++ 14.22 _MSC_VER == 1922 (Visual Studio 2019 Version 16.2)

//Borland C++
#ifdef __BORLANDC__

//Cygwin
#ifdef __CYGWIN__
#ifdef __CYGWIN32__

//mingw
#ifdef __MINGW32__

//windows
#ifdef _WIN32           //32bit
#ifdef _WIN64           //64bit
#ifdef _WINDOWS         //图形界面程序
#ifdef _CONSOLE         //控制台程序

//Windows（95/98/Me/NT/2000/XP/Vista）和Windows CE都定义了
#if (WINVER >= 0x030a)  // Windows 3.1以上
#if (WINVER >= 0x0400)  // Windows 95/NT4.0以上
#if (WINVER >= 0x0410)  // Windows 98以上
#if (WINVER >= 0x0500)  // Windows Me/2000以上
#if (WINVER >= 0x0501)  // Windows XP以上
```

```
#if (WINVER >= 0x0600)  // Windows Vista以上

//_WIN32_WINNT 内核版本
#if (_WIN32_WINNT >= 0x0500) // Windows 2000以上
#if (_WIN32_WINNT >= 0x0501) // Windows XP以上
#if (_WIN32_WINNT >= 0x0600) // Windows Vista以上
```

32. 在pro中判断Qt版本及构建套件位数

```
#打印版本信息
message(qt version: $$QT_VERSION)
#判断当前qt版本号
QT_VERSION = $$[QT_VERSION]
QT_VERSION = $$split(QT_VERSION, ".")
QT_VER_MAJ = $$member(QT_VERSION, 0)
QT_VER_MIN = $$member(QT_VERSION, 1)
#下面是表示 Qt5.5及以上版本
greaterThan(QT_VER_MAJ, 4) {
greaterThan(QT_VER_MIN, 4) {
#自己根据需要做一些处理
}}

#QT_ARCH是Qt5新增的,在Qt4上没效果
#打印当前Qt构建套件的信息
message($$QT_ARCH)
#表示arm平台构建套件
contains(QT_ARCH, arm) {}
#表示32位的构建套件
contains(QT_ARCH, i386) {}
#表示64位的构建套件
contains(QT_ARCH, x86_64) {}

#其实Qt内置了主版本号和子版本号变量
#判断当前qt版本号
message($$QT_ARCH : $$QT_VERSION -> $$QT_MAJOR_VERSION . $$QT_MINOR_VERSION)

#下面的含义是如果版本 < 4.8
lessThan(QT_MAJOR_VERSION, 5) {
lessThan(QT_MINOR_VERSION, 8) {
#这里放要做的处理
}}

#下面的含义是如果版本 < 5.12.0
REQ_QT_MAJOR = 5
REQ_QT_MINOR = 12
REQ_QT_PATCH = 0
lessThan(QT_MAJOR_VERSION, $$REQ_QT_MAJOR)|lessThan(QT_MINOR_VERSION,
$$REQ_QT_MINOR)|lessThan(QT_MINOR_VERSION, $$REQ_QT_PATCH) {
#这里放要做的处理
}

#下面的含义是如果版本 >= 5.5
greaterThan(QT_MAJOR_VERSION, 4) {
greaterThan(QT_MINOR_VERSION, 4) {
#这里放要做的处理
```

```
}}

//代码中判断版本不要太简单
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
//这里放要做的处理
#endif

//下面表示 >= 5.0.0
#if QT_VERSION >= 0x050000
...
#endif

//下面表示 < 5.12.10
#if QT_VERSION < 0x050C0A
...
#endif
```

33. Qt最小化后恢复界面可能会出现假死冻结现象，加上代码

```
void showEvent(QShowEvent *e)
{
    setAttribute(Qt::WA_Mapped);
    QWidget::showEvent(e);
}
```

34. 获取标题栏高度：style()->pixelMetric(QStyle::PM_TitleBarHeight); PM_TitleBarHeight点进去你会发现新大陆，有一堆玩意在里面。
35. 设置高分屏属性以便支持2K4K等高分辨率，尤其是手机app。必须写在main函数的QApplication a(argc, argv);的前面。

```
#if (QT_VERSION >= QT_VERSION_CHECK(5,6,0))
    QGuiApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
#endif
    QApplication a(argc, argv);
```

36. 如果运行程序出现 Fault tolerant heap shim applied to current process. This is usually due to previous crashes. 错误。

- 第一步：输入命令 regedit 打开注册表；
- 第二步：找到节点 HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers\；
- 第三步：选中Layers键值，从右侧列表中删除自己的那个程序路径即可。

37. Qt内置了QFormLayout表单布局用于自动生成标签+输入框的组合的表单界面，设置布局用的很少，一般用的最多的是横向布局、垂直布局、表格布局。
38. qml播放视频在linux需要安装 sudo apt-get install libpulse-dev。
39. 可以直接继承QSqlQueryModel实现自定义的QueryModel，比如某一列字体颜色，占位符，其他样式等，重写QVariant CustomSqlModel::data(const QModelIndex &index, int role) const。
40. Qt5以后提供了类QScroller直接将控件滚动。

```
//禁用横向滚动条
ui->listWidget->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
//禁用纵向滚动条
ui->listWidget->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
//设置横向按照像素值为单位滚动
ui->listWidget->setHorizontalScrollMode(QListWidget::ScrollPerPixel);
//设置纵向按照像素值为单位滚动
ui->listWidget->setVerticalScrollMode(QListWidget::ScrollPerPixel);
//设置滚动对象以及滚动方式为鼠标左键拉动滚动
QScroller::grabGesture(ui->listWidget, QScroller::LeftMouseButtonGesture);
//还有个QScrollerProperties可以设置滚动的一些参数
```

## 05：041-050

41. 如果使用sqlite数据库不想产生数据库文件，可以创建内存数据库。

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
db.setDatabaseName(":memory:");
```

42. 清空数据表并重置自增ID，sql = truncate table table_name。
43. QtChart模块从Qt5.7开始自带，最低编译要求Qt5.4。在安装的时候记得勾选，默认不勾选。使用该模块需要引入命名空间。

```
#include <QChartView>
QT_CHARTS_USE_NAMESPACE
class CustomChart : public QChartView
```

44. QPushButton左对齐文字，需要设置样式表QPushButton{text-align:left;}
45. QLabel有三种设置文本的方法，掌握好Qt的属性系统，举一反三，可以做出很多效果。

```
//常规办法
ui->label->setText("hello");
//取巧办法
ui->label->setProperty("text", "hello");
//属性大法
ui->label->setStyleSheet("qproperty-text:hello;");
```

46. 巧妙的用QEventLoop开启事件循环，可以使得很多同步获取返回结果而不阻塞界面。查看源码得知，原来QEventLoop内部新建了线程执行。

```
QEventLoop loop;
connect(reply, SIGNAL(finished()), &loop, SLOT(quit()));
loop.exec();
```

47. 多种预定义变量 #if (defined webkit) || (defined webengine)，去掉生成空的debug和release目录，在pro文件中加一行 CONFIG -= debug_and_release。
48. 新版的Qtcreator增强了语法检查，会弹出很多警告提示等，可以在插件列表中关闭clang打头的几个即可，Help》About Plugins。也可以设置代码检查级别，Tools》Options 》C++ 》Code Model。
49. QSqlTableModel的rowCount方法，默认最大返回256，如果超过256，可以将表格拉到底部，会自动加载剩余的，每次最大加载256条数据，如果需要打印或者导出数据，记得最好采用sql语句去

查询，而不是使用QSqlTableModel的rowCount方法。不然永远最大只会导出256条数据。
如果数据量很小，也可以采用如下方法：

```
//主动加载所有数据,不然获取到的行数<=256
while(model->canFetchMore()) {
    model->fetchMore();
}
```

50. 如果需要指定无边框窗体，但是又需要保留操作系统的边框特性，比如自由拉伸边框，可以使用setWindowFlags(Qt::CustomizeWindowHint)，这样会保留一个系统白边框。

## 06：051-060

51. 在某些http post数据的时候，如果采用的是&字符串连接的数据发送，中文解析乱码的话，需要将中文进行URL转码。

```
QString content = "测试中文";
QString note = content.toUtf8().toPercentEncoding();
```

52. Qt默认不支持大资源文件，比如添加了字体文件，需要pro文件开启。
CONFIG += resources_big
53. Qt中继承QWidget之后，样式表不起作用，解决办法有三个。强烈推荐方法一。

- 方法一：设置属性 this->setAttribute(Qt::WA_StyledBackground, true);
- 方法二：改成继承QFrame，因为QFrame自带paintEvent函数已做了实现，在使用样式表时会进行解析和绘制。
- 方法三：重新实现QWidget的paintEvent函数时，使用QStylePainter绘制。

```
void Widget::paintEvent(QPaintEvent *)
{
    QStyleOption option;
    option.initFrom(this);
    QPainter painter(this);
    style()->drawPrimitive(QStyle::PE_Widget, &option, &painter, this);
}
```

54. 有时候在界面上加了弹簧，需要动态改变弹簧对应的拉伸策略，对应方法为changeSize，很多人会选择使用set开头去找，找不到的。
55. 在使用QFile的过程中，不建议频繁的打开文件写入然后再关闭文件，比如间隔5ms输出日志，IO性能瓶颈很大，这种情况建议先打开文件不要关闭，等待合适的时机比如析构函数中或者日期变了需要重新变换日志文件的时候关闭文件。不然短时间内大量的打开关闭文件会很卡，文件越大越卡。
56. 在很多网络应用程序，需要自定义心跳包来保持连接，不然断电或者非法关闭程序，对方识别不到，需要进行超时检测，但是有些程序没有提供心跳协议，此时需要启用系统层的保活程序，此方法适用于TCP连接。

```
int fd = tcpSocket->socketDescriptor();
int keepAlive = 1;      //开启keepalive属性,缺省值:0(关闭)
int keepIdle = 5;       //如果在5秒内没有任何数据交互,则进行探测,缺省值:7200(s)
int keepInterval = 2;   //探测时发探测包的时间间隔为2秒,缺省值:75(s)
int keepCount = 2;      //探测重试的次数,全部超时则认定连接失效,缺省值:9(次)
setsockopt(fd, SOL_SOCKET, SO_KEEPALIVE, (void *)&keepAlive,
sizeof(keepAlive));
setsockopt(fd, SOL_TCP, TCP_KEEPIDLE, (void *)&keepIdle, sizeof(keepIdle));
setsockopt(fd, SOL_TCP, TCP_KEEPINTVL, (void *)&keepInterval,
sizeof(keepInterval));
setsockopt(fd, SOL_TCP, TCP_KEEPCNT, (void *)&keepCount, sizeof(keepCount));
```

57. 如果程序打包好以后弹出提示 This application failed to start because it could not find or load the Qt platform plugin 一般都是因为platforms插件目录未打包或者打包错了的原因导致的。
58. 非常不建议tr中包含中文，尽管现在的新版Qt支持中文到其他语言的翻译，但是很不规范，也不知道TMD是谁教的（后面发现我在刚学Qt的时候也发布了一些demo到网上也是tr包含中文的，当时就狠狠的打了自己一巴掌），tr的本意是包含英文，然后翻译到其他语言比如中文，现在大量的初学者滥用tr，如果没有翻译的需求，禁用tr，tr需要开销的，Qt默认会认为他需要翻译，会额外进行特殊处理。
59. 很多人Qt和Qt Creator傻傻分不清楚，经常问Qt什么版本结果发一个Qt Creator的版本过来，Qt Creator是使用Qt编写的集成开发环境IDE，和宇宙第一的Visual Studio一样，他可以是msvc编译器的（WIN对应的Qt集成安装环境中自带的Qt Cerator是msvc的），也可以是mingw编译的，还可以是gcc的。如果是自定义控件插件，需要集成到Qt Creator中，必须保证该插件的动态库文件（dll或者so等文件）对应的编译器和Qt版本以及位数和Qt Creator的版本完全一致才行，否则基本不大可能集成进去。特别注意的是Qt集成环境安装包中的Qt版本和Qt Creator版本未必完全一致，必须擦亮眼睛看清楚，有些是完全一致的。
60. 超过两处相同处理的代码，建议单独写成函数。代码尽量规范精简，比如 if(a == 123) 要写成 if (123 == a)，值在前面，再比如 if (ok == true) 要写成 if (ok)，if (ok == false) 要写成 if (!ok)等。

## 07：061-070

61. 很多人问Qt嵌入式平台用哪个好，这里统一回答（当前时间节点2018年）：imx6+335x比较稳定，性能高就用RK3288 RK3399，便宜的话就用全志H3，玩一玩可以用树莓派香橙派。
62. 对于大段的注释代码，建议用 #if 0 #endif 将代码块包含起来，而不是将该段代码选中然后全部双斜杠注释，下次要打开这段代码的话，又需要重新选中一次取消，如果采用的是 #if 0则只要把0改成1即可，开发效率提升很多。
63. Qt打包发布，有很多办法，Qt5以后提供了打包工具windeployqt（linux上为linuxdeployqt，mac上为macdeployqt）可以很方便的将应用程序打包，使用下来发现也不是万能的，有时候会多打包一些没有依赖的文件，有时候又会忘记打包一些插件尤其是用了qml的情况下，而且不能识别第三方库，比如程序依赖ffmpeg，则对应的库需要自行拷贝，终极大法就是将你的可执行文件复制到Qt安装目录下的bin目录，然后整个一起打包，挨个删除不大可能依赖的组件，直到删到正常运行为止。
64. Qt中的动画，底层用的是QElapsedTimer定时器来完成处理，比如产生一些指定规则算法的数据，然后对属性进行处理。
65. 在绘制无背景颜色只有边框颜色的圆形时候，可以用绘制360度的圆弧替代，效果完全一致。

```
QRect rect(-radius, -radius, radius * 2, radius * 2);
//以下两种方法二选一,其实绘制360度的圆弧=绘制无背景的圆形
painter->drawArc(rect, 0, 360 * 16);
painter->drawEllipse(rect);
```

66. 不要把d指针看的很玄乎，其实就是在类的实现文件定义了一个私有类，用来存放局部变量，个人建议在做一些小项目时，没有太大必要引入这种机制，会降低代码可读性，增加复杂性，新手接受项目后会看的很懵逼。
67. 很多人在绘制的时候，设置画笔以为就只可以设置个单调的颜色，其实QPen还可以设置brush，这样灵活性就提高不知道多少倍，比如设置QPen的brush以后，可以使用各种渐变，比如绘制渐变颜色的进度条和文字等，而不再是单调的一种颜色。
68. 很多控件都带有viewport，比如QTextEdit/QTableWidget/QScrollArea，有时候对这些控件直接处理的时候发现不起作用，需要对其viewport()设置才行，比如设置滚动条区域背景透明，需要使用scrollArea->viewport()->setStyleSheet("background-color:transparent;");而不是scrollArea->setStyleSheet("QScrollArea{background-color:transparent;}");
69. 有时候设置了鼠标跟踪setMouseTracking为真，如果该窗体上面还有其他控件，当鼠标移到其他控件上面的时候，父类的鼠标移动事件MouseMove识别不到了，此时需要用到HoverMove事件，需要先设置 setAttribute(Qt::WA_Hover, true);
70. Qt封装的QDateTime日期时间类非常强大，可以字符串和日期时间相互转换，也可以毫秒数和日期时间相互转换，还可以1970经过的秒数和日期时间相互转换等。

```
QDateTime dateTime;
QString dateTime_str = dateTime.currentDateTime().toString("yyyy-MM-dd hh:mm:ss");
//从字符串转换为毫秒（需完整的年月日时分秒）
datetime.fromString("2011-09-10 12:07:50:541", "yyyy-MM-dd hh:mm:ss:zzz").toMSecsSinceEpoch();
//从字符串转换为秒（需完整的年月日时分秒）
datetime.fromString("2011-09-10 12:07:50:541", "yyyy-MM-dd hh:mm:ss:zzz").toTime_t();
//从毫秒转换到年月日时分秒
datetime.fromMSecsSinceEpoch(1315193829218).toString("yyyy-MM-dd hh:mm:ss:zzz");
//从秒转换到年月日时分秒（若有zzz，则为000）
datetime.fromTime_t(1315193829).toString("yyyy-MM-dd hh:mm:ss[:zzz]");
```

## 08：071-080

71. 在我们使用QList、QStringList、QByteArray等链表或者数组的过程中，如果只需要取值，而不是赋值，强烈建议使用 at() 取值而不是 [] 操作符，在官方书籍《C++ GUI Qt 4编程（第二版）》的书中有特别的强调说明，此教材的原作者据说是Qt开发的核心人员编写的，所以还是比较权威，至于使用 at() 与使用 [] 操作符速度效率的比较，网上也有网友做过此类对比。原文在书的212页，这样描述的：Qt对所有的容器和许多其他类都使用隐含共享，隐含共享是Qt对不希望修改的数据决不进行复制的保证，为了使隐含共享的作用发挥得最好，可以采用两个新的编程习惯。第一种习惯是对于一个（非常量的）向量或者列表进行只读存取时，使用 at() 函数而不用 [] 操作符，因为Qt的容器类不能辨别 [] 操作符是否将出现在一个赋值的左边还是右边，他假设最坏的情况出现并且强制执行深层赋值，而 at() 函数则不被允许出现在一个赋值的左边。
72. 如果是dialog窗体，需要在exec以后还能让其他代码继续执行，请在dialog窗体exec前增加一行代码，否则会阻塞窗体消息。

```
QDialog dialog;
dialog.setWindowModality(Qt::WindowModal);
dialog.exec();
```

73. 安全的删除Qt的对象类，强烈建议使用deleteLater而不是delete，因为deleteLater会选择在合适的时机进行释放，而delete会立即释放，很可能会出错崩溃。如果要批量删除对象集合，可以用

qDeleteAll，比如 qDeleteAll(btns);

74. 在QTableView控件中，如果需要自定义的列按钮、复选框、下拉框等其他模式显示，可以采用自定义委托QItemDelegate来实现，如果需要禁用某列，则在自定义委托的重载createEditor函数返回0即可。自定义委托对应的控件在进入编辑状态的时候出现，如果想一直出现，则需要重载paint函数用drawPrimitive或者drawControl来绘制。
75. 将 QApplication::style() 对应的drawPrimitive、drawControl、drawItemText、drawItemPixmap等几个方法用熟悉了，再结合QStyleOption属性，可以玩转各种自定义委托，还可以直接使用paint函数中的painter进行各种绘制，各种牛逼的表格、树状列表、下拉框等，绝对屌炸天。QApplication::style()->drawControl 的第4个参数如果不设置，则绘制出来的控件不会应用样式表。
76. 心中有坐标，万物皆painter，强烈建议在学习自定义控件绘制的时候，将qpainter.h头文件中的函数全部看一遍、试一遍、理解一遍，这里边包含了所有Qt内置的绘制的接口，对应的参数都试一遍，你会发现很多新大陆，会一定程度上激发你的绘制的兴趣，犹如神笔马良一般，策马崩腾遨游代码绘制的世界。
77. 在使用setItemWidget或者setCellWidget的过程中，有时候会发现设置的控件没有居中显示而是默认的左对齐，而且不会自动拉伸填充，对于追求完美的程序员来说，这个可不大好看，有个终极通用办法就是，将这个控件放到一个widget的布局中，然后将widget添加到item中，这样就完美解决了，而且这样可以组合多个控件产生复杂的控件。

```
//实例化进度条控件
QProgressBar *progress = new QProgressBar;
//增加widget+布局巧妙实现居中
QWidget *widget = new QWidget;
QHBoxLayout *layout = new QHBoxLayout;
layout->setSpacing(0);
layout->setMargin(0);
layout->addWidget(progress);
widget->setLayout(layout);
ui->tableWidget->setCellWidget(0, 0, widget);
```

78. 很多时候需要在已知背景色的情况下，能够清晰的绘制文字，这个时候需要计算对应的文字颜色。

```
//根据背景色自动计算合适的前景色
double gray = (0.299 * color.red() + 0.587 * color.green() + 0.114 *
color.blue()) / 255;
QColor textColor = gray > 0.5 ? Qt::black : Qt::white;
```

79. 对QTableView、QTableWidget、QTreeView、QTreeWidget禁用列拖动。

```
#if (QT_VERSION < QT_VERSION_CHECK(5,0,0))
    ui->tableView->horizontalHeader()->setResizeMode(0, QHeaderView::Fixed);
    ui->treeView->header()->setResizeMode(0, QHeaderView::Fixed);
#else
    ui->tableView->horizontalHeader()->setSectionResizeMode(0,
QHeaderView::Fixed);
    ui->treeView->header()->setSectionResizeMode(0, QHeaderView::Fixed);
#endif
```

80. 从Qt4转到Qt5，有些类的方法已经废弃或者过时了，如果想要在Qt5中启用Qt4的方法，比如QHeadVew的setMovable，可以在你的pro或者pri文件中加上一行即可：DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0

## 09：081-090

81. Qt中的QColor对颜色封装的很完美，支持各种转换，比如rgb、hsb、cmy、hsl，对应的是toRgb、toHsv、toCmyk、toHsl，还支持透明度设置，颜色值还能转成16进制格式显示。

```
QColor color(255, 0, 0, 100);
qDebug() << color.name() << color.name(QColor::HexArgb);
//输出 #ff0000 #64ff0000
```

82. QVariant类型异常的强大，可以说是万能的类型，在进行配置文件的存储的时候，经常会用到QVariant的转换，QVariant默认自带了toString、toFloat等各种转换，但是还是不够，比如有时候需要从QVariant转到QColor，而却没有提供toColor的函数，这个时候就要用到万能办法。

```
if (variant.typeName() == "QColor") {
    QColor color = variant.value<QColor>();
    QFont font = variant.value<QFont>();
    QString nodeValue = color.name(QColor::HexArgb);
}
```

83. Qt中的QString和const char *之间转换，最好用toStdString().c_str()而不是toLocal8Bit().constData()，比如在setProperty中如果用后者，字符串中文就会不正确，英文正常。
84. Qt的信号槽机制非常牛逼，也是Qt的独特的核心功能之一，有时候我们在很多窗体中传递信号来实现更新或者处理，如果窗体层级比较多，比如窗体A的父类是窗体B，窗体B的父类是窗体C，窗体C有个子窗体D，如果窗体A一个信号要传递给窗体D，问题来了，必须先经过窗体B中转到窗体C再到窗体D才行，这样的话各种信号关联信号的connect会非常多而且管理起来比较乱，可以考虑增加一个全局的单例类AppEvent，公共的信号放这里，然后窗体A对应信号绑定到AppEvent，窗体D绑定AppEvent的信号到对应的槽函数即可，干净清爽整洁。
85. QTextEdit右键菜单默认英文的，如果想要中文显示，加载widgets.qm文件即可，一个Qt程序中可以安装多个翻译文件，不冲突。
86. Qt中有个全局的焦点切换信号focusChanged，可以用它做自定义的输入法。Qt4中默认会安装输入法上下文，比如在main函数打印a.inputContext会显示值，这个默认安装的输入法上下文，会拦截两个牛逼的信号QEvent::RequestSoftwareInputPanel和QEvent::CloseSoftwareInputPanel，以至于就算你安装了全局的事件过滤器依然识别不到这两个信号，你只需要在main函数执行a.setInputContext(0)即可，意思是安装输入法上下文为空。Qt5.7以后提供了内置的输入法，可以通过在main函数最前面加上 qputenv("QT_IM_MODULE", QByteArray("qtvirtualkeyboard")); 来启用。
87. 在Qt5.10以后，表格控件QTableWidget或者QTableView的默认最小列宽改成了15，以前的版本是0，所以在新版的qt中，如果设置表格的列宽过小，不会应用，取的是最小的列宽。所以如果要设置更小的列宽需要重新设置ui->tableView->horizontalHeader()->setMinimumSectionSize(0);
88. Qt源码中内置了一些未公开的不能直接使用的黑科技，都藏在对应模块的private中，比如gui-private widgets-private等，比如zip文件解压类QZipReader、压缩类QZipWriter就在gui-private模块中，需要在pro中引入QT += gui-private才能使用。

```
#include "QtGui/private/qzipreader_p.h"
#include "QtGui/private/qzipwriter_p.h"

QZipReader reader(dirPath);
QString path("");
//解压文件夹到当前目录
reader.extractAll(path);
```

```
//文件夹名称
QZipReader::FileInfo fileInfo = reader.entryInfoAt(0);
//解压文件
QFile file(filePath);
file.open(QIODevice::WriteOnly);
file.write(reader.fileData(QString::fromLocal8Bit("%1").arg(filePath)));
file.close();
reader.close();

QZipWriter *writer = new QZipWriter(dirPath);
//添加文件夹
writer->addDirectory(unCompress);
//添加文件
QFile file(filePath);
file.open(QIODevice::ReadOnly);
writer->addFile(data, file.readAll());
file.close();
writer->close();
```

89. 理论上串口和网络收发数据都是默认异步的，操作系统自动调度，完全不会卡住界面，网上那些说收发数据卡住界面主线程的都是扯几把蛋，真正的耗时是在运算以及运算后的处理，而不是收发数据，在一些小数据量运算处理的项目中，一般不建议动用线程去处理，线程需要调度开销的，不要什么东西都往线程里边扔，线程不是万能的。只有当真正需要将一些很耗时的操作比如编码解码等，才需要移到线程处理。
90. 在构造函数中获取控件的宽高很可能是不正确的，需要在控件首次显示以后再获取才是正确的，控件是在首次显示以后才会设置好正确的宽高值，记住是在首次显示以后，而不是构造函数或者程序启动好以后，如果程序启动好以后有些容器控件比如QTabWidget中的没有显示的页面的控件，你去获取宽高很可能也是不正确的，万无一失的办法就是首次显示以后去获取。

## 10：091-100

91. 数据库处理一般建议在主线程，如果非要在其他线程，务必记得打开数据库也要在那个线程，即在那个线程使用数据库就在那个线程打开，不能打开数据库在主线程，执行sql在子线程，很可能出问题。
92. 新版的QTcpServer类在64位版本的Qt下很可能不会进入incomingConnection函数，那是因为Qt5对应的incomingConnection函数参数变了，由之前的int改成了qintptr，改成qintptr有个好处，在32位上自动是quint32而在64位上自动是quint64，如果在Qt5中继续写的参数是int则在32位上没有问题在64位上才有问题，所以为了兼容Qt4和Qt5，必须按照不一样的参数写。

```
#if (QT_VERSION > QT_VERSION_CHECK(5,0,0))
    void incomingConnection(qintptr handle);
#else
    void incomingConnection(int handle);
#endif
```

93. Qt支持所有的界面控件比如QPushButton、QLineEdit自动关联 on*控件名*信号(参数) 信号槽，比如按钮的单击信号 on_pushButton_clicked()，然后直接实现槽函数即可。
94. QWebEngineView控件由于使用了opengl，在某些电脑上可能由于opengl的驱动过低会导致花屏或者各种奇奇怪怪的问题，比如showfullscreen的情况下鼠标右键失效，需要在main函数启用软件opengl渲染。

```
#if (QT_VERSION > QT_VERSION_CHECK(5,4,0))
    //下面两种方法都可以,Qt默认采用的是AA_UseDesktopOpenGL
    QCoreApplication::setAttribute(Qt::AA_UseOpenGLES);
    //QCoreApplication::setAttribute(Qt::AA_UseSoftwareOpenGL);
#endif
    QApplication a(argc, argv);
```

另外一个方法解决 全屏+QWebEngineView控件一起会产生右键菜单无法弹出的bug,需要上移一个像素

```
QRect rect = qApp->desktop()->geometry();
rect.setY(-1);
rect.setHeight(rect.height());
this->setGeometry(rect);
```

95. QStyle内置了很多方法用处很大，比如精确获取滑动条鼠标按下处的值。

```
QStyle::sliderValueFromPosition(minimum(), maximum(), event->x(), width());
```

96. 用QFile读写文件的时候，推荐用QTextStream文件流的方式来读写文件，速度快很多，基本上会有30%的提升，文件越大性能区别越大。

```
//从文件加载英文属性与中文属性对照表
QFile file(":/propertyname.txt");
if (file.open(QFile::ReadOnly)) {
    //QTextStream方法读取速度至少快百分之30
#if 0
    while(!file.atEnd()) {
        QString line = file.readLine();
        appendName(line);
    }
#else
    QTextStream in(&file);
    while (!in.atEnd()) {
        QString line = in.readLine();
        appendName(line);
    }
#endif
    file.close();
}
```

97. 用QFile.readAll()读取QSS文件默认是ANSI格式，不支持UTF8，如果在QtCreator中打开qss文件来编辑保存，这样很可能导致qss加载以后没有效果。

```
void frmMain::initStyle()
{
    //加载样式表
    QString qss;
    //QFile file(":/qss/psblack.css");
    //QFile file(":/qss/flatwhite.css");
    QFile file(":/qss/lightblue.css");
    if (file.open(QFile::ReadOnly)) {
#if 1
```

```
        //用QTextStream读取样式文件不用区分文件编码 带bom也行
        QStringList list;
        QTextStream in(&file);
        //in.setCodec("utf-8");
        while (!in.atEnd()) {
            QString line;
            in >> line;
            list << line;
        }

        qss = list.join("\n");
#else
        //用readAll读取默认支持的是ANSI格式,如果不小心用creator打开编辑过了很可能打不开
        qss = QLatin1String(file.readAll());
#endif
        QString paletteColor = qss.mid(20, 7);
        qApp->setPalette(QPalette(QColor(paletteColor)));
        qApp->setStyleSheet(qss);
        file.close();
    }
}
```

98. QString内置了很多转换函数，比如可以调用toDouble转为double数据，但是当你转完并打印的时候你会发现精确少了，只剩下三位了，其实原始数据还是完整的精确度的，只是打印的时候优化成了三位，如果要保证完整的精确度，可以调用 qSetRealNumberPrecision 函数设置精确度位数即可。

```
QString s1, s2;
s1 = "666.5567124";
s2.setNum(888.5632123, 'f', 7);
qDebug() << qSetRealNumberPrecision(10) << s1.toDouble() << s2.toDouble();
```

99. 用QScriptValueIterator解析数据的时候，会发现总是会多一个节点内容，并且内容为空，如果需要跳过则增加一行代码。

```
while (it.hasNext()) {
    it.next();
    if (it.flags() & QScriptValue::SkipInEnumeration)
       continue;
    qDebug() << it.name();
}
```

100. setPixmap是最糟糕的贴图方式，一般只用来简单的不是很频繁的贴图，频繁的建议painter绘制，默认双缓冲，在高级点用opengl绘制，利用GPU。

## 11：101-110

101. 如果需要在尺寸改变的时候不重绘窗体，则设置属性即可 this->setAttribute(Qt::WA_StaticContents, true); 这样可以避免对已经显示区域的重新绘制。
102. 默认程序中获取焦点以后会有虚边框，如果看着觉得碍眼不舒服可以去掉，设置样式即可：setStyleSheet("*{outline:0px;}");

103. Qt表格控件一些常用的设置封装，QTableWidget继承自QTableView，所以下面这个函数支持传入QTableWidget。

```
void QUIHelper::initTableView(QTableView *tableView, int rowHeight, bool headVisible, bool edit)
{
    //奇数偶数行颜色交替
    tableView->setAlternatingRowColors(false);
    //垂直表头是否可见
    tableView->verticalHeader()->setVisible(headVisible);
    //选中一行表头是否加粗
    tableView->horizontalHeader()->setHighlightSections(false);
    //最后一行拉伸填充
    tableView->horizontalHeader()->setStretchLastSection(true);
    //行标题最小宽度尺寸
    tableView->horizontalHeader()->setMinimumSectionSize(0);
    //行标题最大高度
    tableView->horizontalHeader()->setMaximumHeight(rowHeight);
    //默认行高
    tableView->verticalHeader()->setDefaultSectionSize(rowHeight);
    //选中时一行整体选中
    tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
    //只允许选择单个
    tableView->setSelectionMode(QAbstractItemView::SingleSelection);

    //表头不可单击
#if (QT_VERSION > QT_VERSION_CHECK(5,0,0))
    tableView->horizontalHeader()->setSectionsClickable(false);
#else
    tableView->horizontalHeader()->setClickable(false);
#endif

    //鼠标按下即进入编辑模式
    if (edit) {
        tableView->setEditTriggers(QAbstractItemView::CurrentChanged | QAbstractItemView::DoubleClicked);
    } else {
        tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
    }
}
```

104. 在一些大的项目中，可能嵌套了很多子项目，有时候会遇到子项目依赖其他子项目的时候，比如一部分子项目用来生成动态库，一部分子项目依赖这个动态库进行编译，此时就需要子项目按照顺序编译或者设置好依赖规则。

```
TEMPLATE = subdirs
#设置ordered参数以后会依次编译 projA projB projC
CONFIG  += ordered
SUBDIRS += projA
SUBDIRS += projB
SUBDIRS += projC
#还可以通过设置depends指定某个项目依赖 比如下面指定projB依赖projA
projB.depends = projA
projC.depends = projA
projD.depends = projC
```

105. MSVC编译器的选择说明

- 如果是32位的Qt则编译器选择x86开头的
- 如果是64位的Qt则编译器选择amd64开头的
- 具体是看安装的Qt构建套件版本以及目标运行平台的系统位数和架构
- 一般现在的电脑默认以64位的居多，选择amd64即可
- 如果用户需要兼容32位的系统则建议选择32位的Qt，这样即可在32位也可以在64位系统运行
- 诸葛大佬补充：x86/x64都是编译环境和运行环境相同，没有或。带下划线的就是交叉编译，前面是编译环境，后面是运行环境。

| 名称 | 说明 |
| --- | --- |
| x86 | 32/64位系统上编译在32/64位系统上运行 |
| x86_amd64 | 32/64位系统上编译在64位系统上运行 |
| x86_arm | 32/64位系统上编译在arm系统上运行 |
| amd64 | 64位系统上编译在64位系统上运行 |
| amd64_x86 | 64位系统上编译在32/64位系统上运行 |
| amd64_arm | 64位系统上编译在arm系统上运行 |

106. 很多时候用QDialog的时候会发现阻塞了消息，而有的时候我们希望是后台的一些消息继续运行不要终止，此时需要做个设置。

```
QDialog dialog;
dialog.setWindowModality(Qt::WindowModal);
```

107. 很多初学者甚至几年工作经验的人，对多线程有很深的误解和滥用，尤其是在串口和网络通信这块，什么都往多线程里面丢，一旦遇到界面卡，就把数据收发啥的都搞到多线程里面去，殊不知绝大部分时候那根本没啥用，因为没找到出问题的根源。

- 如果你没有使用wait***函数的话，大部分的界面卡都出在数据处理和展示中，比如传过来的是一张图片的数据，你需要将这些数据转成图片，这个肯定是耗时的；
- 还有就是就收到的数据曲线绘制出来，如果过于频繁或者间隔过短，肯定会给UI造成很大的压力的，最好的办法是解决如何不要频繁绘制UI比如合并数据一起绘制等；
- 如果是因为绘制UI造成的卡，那多线程也是没啥用的，因为UI只能在主线程；
- 串口和网络的数据收发默认都是异步的，由操作系统调度的，如果数据处理复杂而且数据量大，你要做的是将数据处理放到多线程中；

- 如果没有严格的数据同步需求，根本不需要调用wait***之类的函数来立即发送和接收数据，实际需求中大部分的应用场景其实异步收发数据就足够了；
- 有严格数据同步需求的场景还是放到多线程会好一些，不然你wait***就卡在那边了；
- 多线程是需要占用系统资源的，理论上来说，如果线程数量超过了CPU的核心数量，其实多线程调度可能花费的时间更多，各位在使用过程中要权衡利弊；
- 再次强调，不要指望Qt的网络通信支持高并发，最多到1000个能正常工作就万事大吉，一般建议500以内的连接数。有大量高并发的需求请用第三方库比如swoole等。

108. 在嵌入式linux上，如果设置了无边框窗体，而该窗体中又有文本框之类的，发现没法产生焦点进行输入，此时需要主动激活窗体才行。

```
//这种方式设置的无边框窗体在嵌入式设备上无法产生焦点
setWindowFlags(Qt::WindowStaysOnTopHint | Qt::FramelessWindowHint |
Qt::X11BypassWindowManagerHint);

//需要在show以后主动激活窗体
w->show();
w->activateWindow();
```

109. QString的replace函数会改变原字符串，切记，他在返回替换后的新字符串的同时也会改变原字符串，我的乖乖！
110. QGraphicsEffect类的相关效果很炫，可以实现很多效果比如透明、渐变、阴影等，但是该类很耗CPU，如果不是特别需要一般不建议用，就算用也是要用在该部件后期不会发生频繁绘制的场景，不然会让你哭晕在厕所。

## 12：111-120

111. 在不同的平台上文件路径的斜杠也是不一样的，比如linux系统一般都是 / 斜杠，而在windows上都是 \ 两个反斜杠，Qt本身程序内部无论在win还是linux都支持 / 斜杠的路径，但是一些第三方库的话可能需要转换成对应系统的路径，这就需要用到斜杠转换，Qt当然内置类方法。

```
QString path = "C:/temp/test.txt";
path = QDir::toNativeSeparators(path);
//输出 C:\\temp\\test.txt

QString path = "C:\\temp\\test.txt";
path = QDir::toNativeSeparators(path);
//输出 C:/temp/test.txt
```

112. 巧用QMetaObject::invokeMethod方法可以实现很多效果，包括同步和异步执行，很大程度上解决了跨线程处理信号槽的问题。比如有个应用场景是在回调中，需要异步调用一个public函数，如果直接调用的话会发现不成功，此时需要使用 QMetaObject::invokeMethod(obj, "fun", Qt::QueuedConnection); 这种方式来就可以。

- invokeMethod函数有很多重载参数，可以传入返回值和执行方法的参数等。
- invokeMethod函数不仅支持槽函数还支持信号，而且这逼居然是线程安全的，可以在线程中放心使用，牛逼！
- 测试下来发现只能执行signals或者slots标识的方法。
- 默认可以执行private(protected/public) slots下的函数，但是不能执行private(protected/public)下的函数。
- 毛总补充：前提必须是slots或者signals标注的函数，不是标注的函数不在元信息导致无法查找，执行之后会提示No such method。

- 2021-11-06补充：如果要执行private(protected/public)下的函数，需要函数前面加上 Q_INVOKABLE 关键字，今天又学到了，必须加鸡腿。
- 其实这样看下来，就是任何方法函数都能执行了，这就超越了private(protected/public)的权限限定了，相当于一个类的私有函数用了 Q_INVOKABLE 关键字修饰也可以被 invokeMethod 执行，哇咔咔。

```
//头文件声明信号和槽函数
signals:
    void sig_test(int type,double value);
private slots:
    void slot_test(int type, double value);
private:
    Q_INVOKABLE void fun_test(int type, double value);

//构造函数关联信号槽
connect(this, SIGNAL(sig_test(int, double)), this, SLOT(slot_test(int, double)));

//单击按钮触发信号和槽,这里是同时举例信号槽都可以
void MainWindow::on_pushButton_clicked()
{
    QMetaObject::invokeMethod(this, "sig_test", Q_ARG(int, 66), Q_ARG(double, 66.66));
    QMetaObject::invokeMethod(this, "slot_test", Q_ARG(int, 88), Q_ARG(double, 88.88));
    QMetaObject::invokeMethod(this, "fun_test", Q_ARG(int, 99), Q_ARG(double, 99.99));
}

//会打印 66 66.66、88 88.88
void MainWindow::slot_test(int type, double value)
{
    qDebug() << type << value;
}

//会打印 99.99
void MainWindow::fun_test(int type, double value)
{
    qDebug() << type << value;
}
```

113. Qt5中的信号是public的，可以在需要的地方直接emit即可，而在Qt4中信号是protected的，不能直接使用，需要定义一个public函数来emit。
114. Qt5.15版本开始官方不再提供安装包，只提供源码，可以自行编译或者在线安装，估计每次编译各种版本太麻烦，更多的是为了统计收集用户使用信息比如通过在线安装，后期可能会逐步加大商业化力度。
115. 有时候我们需要判断当前Qt版本有没有某个模块可以使用qtHaveModule（Qt5新引入的判断）来判断，如果要判断自己的项目中有没有 QT += 的方式添加的模块，可以用 contains来判断。

```
qtHaveModule(webenginewidgets) {
message("当前Qt库有找到 webenginewidgets 模块")
}

```

```
!qtHaveModule(webkit) {
message("当前Qt库没有找到 webkit 模块")
}

contains(QT, network) {
message("当前项目已经引入 network 模块")
}

!contains(QT, widgets) {
message("当前项目没有引入 widgets 模块")
}
```

116. c++11新引入了原始字符串格式，用户避免在字符串中加入转义字符\，可以用于表示json字符串等场景。

```
QString s1 = R"(test\001.jpg)";
s1.replace("\\", "#");
qDebug()<< s1;
//结果 test#001.jpg
```

117. 安卓上打印信息建议使用 qInfo() 而不是 qDebug() ， qInfo()才有效果。
118. Qt的默认定时器精度不够高（比如应用场景是1分钟保存一条记录或者文件，当你用默认的定时器的时候你会发现有些时候是60秒而有些是59秒随机的，如果客户有要求这就需要设置精度了。当然我们所做的绝大部分项目也不需要精度非常高的定时器，毕竟精度越高，占用的系统资源可能越大），如果需要设置更高的精度可以设置 setTimerType(Qt::PreciseTimer)。Qt有两种定时器处理，一种是QTimer类，还有一种是QObject类就内置的timeevent事件，如果是QObject类的定时器要设置的话调用 startTimer(interval, Qt::PreciseTimer);

- Qt::PreciseTimer 精确的定时器，尽量保持毫秒精度。
- Qt::CoarseTimer 粗略的定时器，尽量保持精度在所需的时间间隔5%范围内。
- Qt::VeryCoarseTimer 很粗略的定时器，只保留完整的第二精度。
- 精度再高，也依赖对应的操作系统中断，假设中断需要 5ms，则定时器精度不可能高于5毫秒。

119. QGraphicsEffect相关类很耗CPU，甚至在绘制的时候和某些地方有冲突干扰，基本上不建议使用，情非得已只建议少量使用和非频繁触发绘制的地方使用。
120. 用QSettings设置注册表，如果不是管理员身份运行会打印 QSettings: failed to set subkey "xxx" (拒绝访问。)，你需要手动鼠标右键管理员身份运行就可以。

## 13：121-130

121. QLineEdit除了单纯的文本框以外，还可以做很多特殊的处理用途。

- 限制输入只能输入IP地址。
- 限制输入范围，强烈推荐使用 QRegExpValidator 正则表达式来处理。

```
//正在表达式限制输入
QString str = "\\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\b";
ui->lineEdit->setValidator(new QRegExpValidator(QRegExp(str)));
//用于占位
ui->lineEdit->setInputMask("000.000.000.000");

#if 0
//下面代码设置浮点数范围限制失败
```

```
ui->lineEdit->setValidator(new QDoubleValidator(20, 50, 1));
#else
//下面代码设置浮点数范围限制成功
QDoubleValidator *validator = new QDoubleValidator(20, 50, 1);
validator->setNotation(QDoubleValidator::StandardNotation);
ui->lineEdit->setValidator(validator);
#endif
//下面代码设置整数范围限制成功
ui->lineEdit->setValidator(new QIntValidator(10, 120));

//其实上面的代码缺陷很多，只能限制只输入小数，无法设定数值范围，很操蛋
//需要来个万能的牛逼的 QRegExpValidator

//限制浮点数输入范围为[-180,180]
QRegExp regexp("^-?(180|1?[0-7]?\\d(\\.\\d+)?)$");
//限制浮点数输入范围为[-90,90]并限定为小数位后4位
QRegExp regexp("^-?(90|[1-8]?\\d(\\.\\d{1,4})?)$");
QRegExpValidator *validator = new QRegExpValidator(regexp, this);
ui->lineEdit->setValidator(validator);
```

122. 在继承自QAbstractItemView的控件中，比如QTableView、QTableWidget，如果文本超过对应item的宽度，则会自动省略号显示，想要快速显示完整的文本，可以在该列和下一列分割线中间双击即可，会自动自适应显示最大宽度，如果是Qt5.14或者更高版本，你会发现显示省略号的计算规则变了，如果是rtsp、http之类的开头的英文字符串，同样的列宽下，会提前就显示省略号，比如字符串 rtmp://58.200.131.2:1935/livetv/cctv1，会显示成 rtmp://... ，而在旧版本的Qt中会显示成 rtmp://58.200.131... ，很多时候我们并不想看到烦人的省略号，可以设置取消。

```
//取消自动换行
tableView->setWordWrap(false);
//超出文本不显示省略号
tableView->setTextElideMode(Qt::ElideNone);
```

123. QVideoWidget播放视频，可能会遇到画面闪烁的情况，播放视频的窗体需要设置个属性。

```
QVideoWidget *videoWidget = new QVideoWidget;
videoWidget->setAttribute(Qt::WA_OpaquePaintEvent);
```

123. Qt bug成千上万，这个不用大惊小怪，也基本上遇不到，大部分都是特殊极端情况特定应用场景出现，甚至你会遇到有些是debug可以release报错，有些release可以debug却报错的情况，最神奇的还有先是debug报错，然后release正常，再返回去用debug又正常，需要用release激活一下！学习编程的路本来就是一条坑坑洼洼的路，不断填坑，尽量规避坑！很多时候很多看起来的坑其实是自己没有注意细节导致的。
124. Qt视图中默认排序是按照字符串的ASCII排序的，如果是IP地址的话会出现192.168.1.117排在192.168.1.2前面的情况，如果要规避这种情况，一种做法是取末尾的地址转成整型再比较大小，缺点是跨网段就歇菜了，又会出现192.168.2.65出现在192.168.1.70前面，终极大法是将IP地址转成整型再比较大小。

```
QString QUIHelper::ipv4IntToString(quint32 ip)
{
    QString result = QString("%1.%2.%3.%4").arg((ip >> 24) & 0xFF).arg((ip >> 16) & 0xFF).arg((ip >> 8) & 0xFF).arg(ip & 0xFF);
    return result;
```

```
}

quint32 QUIHelper::ipv4StringToInt(const QString &ip)
{
    int result = 0;
    if (isIP(ip)) {
        QStringList list = ip.split(".");
        int ip0 = list.at(0).toInt();
        int ip1 = list.at(1).toInt();
        int ip2 = list.at(2).toInt();
        int ip3 = list.at(3).toInt();
        result = ip3 | ip2 << 8 | ip1 << 16 | ip0 << 24;
    }
    return result;
}
```

125. 在主QWidget窗体如果直接qss设置背景图片的话，预览是可见的，运行并没有效果，你需要在这个主widget上再放个widget，在新的widget上设置qss图片就行，而如果是Dialog或者QMainWindow窗体是支持直接设置qss背景图的，预览和运行效果一致。
126. Qt提供了qDebug机制直接输出打印信息，这个弥补了QtCreator调试很鸡肋的缺点，而且无缝对接日志钩子，使得现场运行期间按照预定的打印信息输出到日志文件，有时候在开发阶段，又不想要看到一堆堆的打印信息，最笨的做法是一行行注释掉qdebug的地方，其实还可以直接pro中加上一行来禁用整个项目的qdebug输出。

```
#禁用qdebug打印输出
DEFINES += QT_NO_DEBUG_OUTPUT
```

127. 在使用 QT_NO_DEBUG_OUTPUT 关键字禁用了所有打印信息以后，可以节约不少的开销，有时候又想在禁用打印信息后，极少地方还需要看到打印信息，怎么办呢？其实 QT_NO_DEBUG_OUTPUT 禁用的 qdebug 的输出，Qt还有其他几种打印信息比如 qInfo、qWarning、qCritical，这些是不受影响的，也就是说在极少部分需要打印的地方用 qInfo 来输出信息就好。特别注意：qFatal 打印完信息程序会自动结束。

```
qDebug() << "qDebug";
qInfo() << "qInfo";
qWarning() << "qWarning";
qCritical() << "qCritical";

qDebug("qDebug");
qWarning("qWarning");
qCritical("qCritical");
```

128. Qt的pro文件可以添加各种处理来使得配置更方便，比如指定输出文件路径等，这样就不会全部在一堆编译生成的临时文件中找来找去。

```
#禁用qdebug打印输出
DEFINES     += QT_NO_DEBUG_OUTPUT

#自定义define变量 可以在整个项目中使用
#pro文件可以这样判断 contains(DEFINES, videovlc) {}
#代码文件可以这样判断 #ifdef videovlc
DEFINES     += videovlc1 videoffmpeg

```

```
#关闭编译警告提示 眼不见为净
CONFIG      += warn_off

#指定编译生成的文件到temp目录 分门别类存储
MOC_DIR     = temp/moc
RCC_DIR     = temp/rcc
UI_DIR      = temp/ui
OBJECTS_DIR = temp/obj

#指定编译生成的可执行文件到bin目录
DESTDIR     = bin
```

129. Qt对操作系统层的消息也做了很多的封装，可以直接拿到进行处理（如果需要拦截处理要用对应操作系统的API才行比如鼠标键盘钩子），比如系统休眠和唤醒做一些处理。

```
//主窗体头文件
protected:
    bool nativeEvent(const QByteArray &eventType, void *message, long *result);
#ifdef Q_OS_WIN
    bool winEvent(MSG *message, long *result);
#endif

//主窗体实现函数
#ifdef Q_OS_WIN
#include "Windows.h"
#endif

bool frmMain::nativeEvent(const QByteArray &eventType, void *message, long *result)
{
    if (eventType == "windows_generic_MSG") {
#ifdef Q_OS_WIN
        MSG *msg = static_cast<MSG *>(message);
        //qDebug() << TIMEMS << msg->message;
        if (msg->wParam == PBT_APMSUSPEND && msg->message == WM_POWERBROADCAST) {
            //系统休眠的时候自动最小化可以规避程序可能出现的问题
            this->showMinimized();
        } else if (msg->wParam == PBT_APMRESUMEAUTOMATIC) {
            //休眠唤醒后自动打开
            this->showNormal();
        }
#endif
    } else if (eventType == "NSEvent") {
#ifdef Q_OS_MACOS
#endif
    }
    return false;
}

#ifdef Q_OS_WIN
bool frmMain::winEvent(MSG *message, long *result)
{
    return nativeEvent("windows_generic_MSG", message, result);
```

```
}
#endif
```

130. Qt的pro项目管理配置文件中也可添加各种编译前后的操作及配置，主要通过 QMAKE_POST_LINK 和QMAKE_PRE_LINK，他们支持的函数以及写法，可以在QtCreator的帮助中搜索 qmake Function Reference 查看详情说明。

- QMAKE_PRE_LINK　表示编译前执行内容
- QMAKE_POST_LINK　表示编译后执行内容

```
srcFile1 = $$PWD/1.txt
srcFile2 = $$PWD/2.txt
dstDir = $$PWD/../bin
#windows上需要转换路径斜杠 其他系统不需要
srcFile1 = $$replace(srcFile1, /, \\);
srcFile2 = $$replace(srcFile2, /, \\);
dstDir = $$replace(dstDir, /, \\);

#编译前执行拷贝 多个拷贝可以通过 && 符号隔开
QMAKE_PRE_LINK += copy /Y $$srcFile1 $$dstDir && copy /Y $$srcFile2 $$dstDir
#编译后执行拷贝 多个拷贝可以通过 && 符号隔开
QMAKE_POST_LINK += copy /Y $$srcFile1 $$dstDir && copy /Y $$srcFile2
$$dstDir
```

## 14：131-140

131. Qt新版本往往会带来一些头文件的更新，比如以前使用QPainter绘制，不需要额外包含QPainterPath头文件，而5.15版本开始就需要显示主动引入#include "qpainterpath.h"才行。
132. Qt6.0发布了，是个比较大的改动版本，很多基础的类或者组件都放到单独的源码包中，需要自行官网下载并编译，默认不提供集成在开发目录下，需要手动编译并集成，比如QRegExp，QTextCodec类，需要编译集成后pro文件 QT += core5compat 才能用， 具体说明在https://doc.qt.io/qt-6/qtcore5-index.html。
133. qDebug输出打印信息，默认会完整打印转义字符，例如：\ " \t \n" 等，所以当你发现你明明设置了转义字符以后打印确还是转义前的字符，这就懵逼了，其实这是qdebug为了方便调试将各种字符都打印输出。无可否认，很多时候，我们极其兴奋的享受着Qt带来的各种轮子各种便利，但是偶尔，稍不留意，这些便利可能也会坑你一把。要做的就是擦亮眼睛，时刻谨慎，一步一个脚印踏踏实实码代码。

```
QString s1 = R"(\:device0)";
//TNND居然输出的是 \\:device0
qDebug() << s1;
//这次终于正确的输出 \:device0
qDebug().noquote() << s1;
```

134. 很多人有疑问为何qss对浏览器控件中的网页样式没法控制，其实用屁股想想也知道，那玩意是html css去控制的，和Qt一毛钱关系也没有，根本管不着，如果想要对滚动条样式设置，可以在网页代码中设置样式就行。

```
<style type="text/css">
  ::-webkit-scrollbar{width:0.8em;}
  ::-webkit-scrollbar-track{background:rgb(241,241,241);}
  ::-webkit-scrollbar-thumb{background:rgb(188,188,188);}
</style>
```

135. Qt的ini配置文件默认不支持直接读写中文，需要手动设置下编码格式才行，强烈建议统一用utf-8编码，包括代码文件。

```
//设置了编码以后配置文件内容为 Company=上海物联网技术研究中心
//没有设置编码则配置文件内容为
Company=\xe4\xb8\x8a\xe6\xb5\xb7\xe7\x89\xa9\xe8\x81\x94\xe7\xbd\x91\xe6\x8a\x80\xe6\x9c\xaf\xe7\xa0\x94\xe7\xa9\xb6\xe4\xb8\xad\xe5\xbf\x83
void App::readConfig()
{
    QSettings set(App::ConfigFile, QSettings::IniFormat);
    set.setIniCodec("utf-8");

    set.beginGroup("AppConfig1");
    App::Company = set.value("Company", App::Company).toString();
    set.endGroup();
}
void App::writeConfig()
{
    QSettings set(App::ConfigFile, QSettings::IniFormat);
    set.setIniCodec("utf-8");

    set.beginGroup("AppConfig1");
    set.setValue("Company", App::Company);
    set.endGroup();
}
```

136. 用Qt做安卓开发都会遇到权限的问题，早期的安卓版本可以直接通过 AndroidManifest.xml 配置文件来添加需要的权限，这样在安装app的时候就会提示该app需要哪些权限让用户同意，现在的安卓版本都改成了动态权限，需要在app运行的时候弹出提示让用户确认再有权限，Qt迎合了这种策略内置了动态申请权限的方法 QtAndroid::requestPermissionsSync。

```
//动态设置权限
bool checkPermission(const QString &permission)
{
#ifdef Q_OS_ANDROID
#if (QT_VERSION >= QT_VERSION_CHECK(5, 10, 0))
    QtAndroid::PermissionResult result = QtAndroid::checkPermission(permission);
    if (result == QtAndroid::PermissionResult::Denied) {
        QtAndroid::requestPermissionsSync(QStringList() << permission);
        result = QtAndroid::checkPermission(permission);
        if (result == QtAndroid::PermissionResult::Denied) {
            return false;
        }
    }
#endif
#endif
```

```
    return true;
}

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    //请求权限
    checkPermission("android.permission.READ_EXTERNAL_STORAGE");
    checkPermission("android.permission.WRITE_EXTERNAL_STORAGE");

    return a.exec();
}
```

137. Qt重载qDebug输出自定义的信息。

```
struct FunctionInfo {
    QString function;
    QString name;
    QString groupEnabled;
    QString action;
    QString group;

    friend QDebug operator << (QDebug debug, const FunctionInfo &functionInfo) {
        QString info = QString("功能: %1  名称: %2  启用: %3  方法: %4  分组: %5")

.arg(functionInfo.function).arg(functionInfo.name).arg(functionInfo.groupEnabled)
                        .arg(functionInfo.action).arg(functionInfo.group);
        debug << info;
        return debug;
    }
};
```

138. 对高分屏不同缩放比例的自适应处理方法。

```
//方法1：在main函数的最前面加上下面这句 5.6版本才开始有这个函数
#if (QT_VERSION > QT_VERSION_CHECK(5,6,0))
    QGuiApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    //开启高缩放支持以后图片可能发虚还要开启下面这个属性
    QCoreApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);
#endif

//方法2：在可执行文件同目录下新建文件 qt.conf 填入下面内容
[Platforms]
WindowsArguments = dpiawareness=0
//下面这行用来解决Qt高DPI下文字显示有锯齿的问题
WindowsArguments = fontengine=freetype

//方法3：在main函数最前面设置Qt内部的环境变量
qputenv("QT_AUTO_SCREEN_SCALE_FACTOR", "1.5");

```

```
//方法4：新版本的Qt比如Qt5.14修正了对高分屏的处理支持不是整数的缩放
qputenv("QT_ENABLE_HIGHDPI_SCALING", "1");
QGuiApplication::setHighDpiScaleFactorRoundingPolicy(Qt::HighDpiScaleFactorRoundingPolicy::PassThrough);

//禁用缩放
//测试发现AA_Use96Dpi属性在Qt5.9以上版本完全正常，以下版本比如5.7有部分控件在175%缩放不正常比如QTextEdit，需要外层套个widget才行。
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
    QApplication::setAttribute(Qt::AA_Use96Dpi);
#endif
#if (QT_VERSION >= QT_VERSION_CHECK(5,14,0))

 QGuiApplication::setHighDpiScaleFactorRoundingPolicy(Qt::HighDpiScaleFactorRoundingPolicy::Floor);
#endif
```

139. QTabWidget选项卡有个自动生成按钮切换选项卡的机制，有时候不想看到这个烦人的切换按钮，可以设置usesScrollButtons为假，其实QTabWidget的usesScrollButtons属性最终是应用到QTabWidget的QTabBar对象上，所以只要设置全局的QTabBar的这个属性关闭即可。为啥要设置全局的呢，因为如果只是对QTabWidget设置了该属性，而在QMainWindow窗体中QDockWidget合并自动形成的选项卡只有QTabBar对象导致依然是有切换按钮。

```
//对tabWidget设置无切换按钮
ui->tabWidget->setUsesScrollButtons(false);
//对tabBar设置无切换按钮
ui->tabWidget->tabBar()->setUsesScrollButtons(false);
//对整个系统的选项卡设置无切换按钮
QTabBar{qproperty-usesScrollButtons:false;}
//设置选项卡自动拉伸 这玩意居然之前自动计算来设置原来内置了哇咔咔
QTabBar{qproperty-expanding:false;}
//设置选项卡关闭按钮可见
QTabBar{qproperty-tabsClosable:true;}
//还有其他属性参见QTabBar头文件有惊喜
//依旧是万能大法所有可视化类的 Q_PROPERTY 包含的属性都可以这样设置
```

140. QMainWindow的分割线默认尺寸比较大，有时候想设置小一点或者不想要，最开始的时候以为是QSplitter，打印所有子元素找遍了也没找到影子，最后发现样式表中有对应设置的内容。

```
//真的是做梦也没想到要这样设置
QMainWindow::separator{width:1px;height:1px;margin:1px;padding:1px;background:#FF0000;}
```

## 15：141-150

141. QImage支持xpm图标，查看Qt内置的QStyle风格的代码中可以发现大量的xpm图标定义，通过代码的形式来产生图标，哇咔咔好牛逼。

```
static const char * const imgData[] = {
    "15 11 6 1",
    "   c None",
    "+  c #979797",
    "@  c #C9C9C9",
```

```
    "$  c #C1C1C1",
    "b  c None",
    "d  c None",
    " $++++++++$    ",
    "$+bbbbbbbb+$   ",
    "+b $$      +$  ",
    "+b $@       +$ ",
    "+b           +$",
    "+b           d+",
    "+b          d+$",
    "+b $$      d+$ ",
    "+b $@     d+$  ",
    "$+dddddddd+$   ",
    " $++++++++$    "};

//这样就能直接显示一个箭头的图形
QImage img(imgData);
QLabel lab;
lab.setPixmap(QPixmap::fromImage(img));
lab.show();
```

142. 在停靠窗体QDockWidget和QOpenGLWidget同时使用的时候，从嵌入状态切换到浮动状态或者浮动状态切换到嵌入状态，QOpenGLWidget的上下文会被打乱导致白屏失效，需要在main函数中开头位置设置下共享OpenGL上下文。

```
int main(int argc, char *argv[])
{
    //需要设置共享上下文不然停靠窗体从正常到浮动后QOpenGLWidget窗体会失效
#if (QT_VERSION > QT_VERSION_CHECK(5,4,0))
    QCoreApplication::setAttribute(Qt::AA_ShareOpenGLContexts);
#endif
    QApplication a(argc, argv);
    ...
}
```

143. 关于Qt中文乱码的问题，个人也稍微总结了一点，应该可以解决99%以上的Qt版本的乱码问题。

- 第一步：代码文件选择用utf8编码带bom。
- 第二步：在有中文汉字的代码文件顶部加一行（一般是cpp文件） #pragma execution_character_set("utf-8") 可以考虑放在head.h中，然后需要的地方就引入head头文件就行，而不是这行代码写的到处都是；这行代码是为了告诉msvc编译器当前代码文件用utf8去编译。
- 第三步：main函数中加入设置编码的代码，以便兼容Qt4，如果没有Qt4的场景可以不用，从Qt5开始默认就是utf8编码。

```
void QUIHelper::setCode()
{
#if (QT_VERSION <= QT_VERSION_CHECK(5,0,0))
#if _MSC_VER
    QTextCodec *codec = QTextCodec::codecForName("gbk");
#else
    QTextCodec *codec = QTextCodec::codecForName("utf-8");
#endif
    QTextCodec::setCodecForLocale(codec);
    QTextCodec::setCodecForCStrings(codec);
```

```
    QTextCodec::setCodecForTr(codec);
#else
    QTextCodec *codec = QTextCodec::codecForName("utf-8");
    QTextCodec::setCodecForLocale(codec);
#endif
}
```

144. 关于Qt众多版本（至少几百个）都不兼容的问题，在经过和Qt中国的林斌大神和其他大神（Qt非官方技术交流群）头脑风暴以后，最终得出以下的结论。

- Qt在二进制兼容这块，已经做了最大的努力，通过将各种代码细节隐藏，Q指针+D指针技巧，尽量保持了接口的统一；
- 是否兼容最主要考虑编译器的因素，毕竟任何Qt版本都是需要通过编译器编译成对应的二进制文件，由他说了算。如果两个Qt版本采用的编译器版本一样，极大概率可执行文件是兼容的，比如Qt5.10+msvc2015 32 位 和 Qt5.11+msvc2015 32位 编译出来的可执行文件，都用Qt5.11的库是可行的；
- mingw编译器的Qt版本也是如此，就是因为Qt官方安装包集成的mingw编译器一直在更新（极少附近版本没有更新mingw编译器版本除外），比如5.7用的mingw53，5.12用的mingw73，5.15用的mingw81，因为带的Qt库也是这个编译器编译出来的，所以导致看起来全部不兼容；
- 如果想要完全兼容，还有一个注意要素，那就是对应代码使用的类的头文件接口是否变了，按道理原有的接口极少会变，一般都是新增加，或者大版本才会改变，比如Qt4-Qt5-Qt6这种肯定没法兼容的，接口和模块都变了；
- 大胆的猜测：如果Qt5.6到Qt5.15你全部用一种编译器比如mingw73或者msvc2015重新编译生成对应的Qt运行库，然后在此基础上开发程序，最后生成的可执行文件用Qt5.15的库是都可以的，这样就轻松跨越了多个版本兼容；
- 大胆的建议：在附近的几个版本统一编译器，比如5.6-5.12之间就统一用mingw53或者msvc2015,5.12-5.15统一用msvc2017，要尝鲜其他编译器的可以自行源码编译其他版本，这样最起码附近的一大段版本（大概2-3年的版本周期）默认就兼容了。
- 本人测试的是widget部分，qml未做测试，不清楚是否机制一样；

145. 通过酷码大哥（Qt开发者交流群）的指点，到今天才知道，Qt设置样式表支持直接传入样式表文件路径，亲测4.7到5.15任意版本，通过查看对应函数的源码可以看到内部会检查是否是 'file:///' 开头，是的话则自动读取样式表文件进行设置，无需手动读取。

```
//以前都是下面的方法
QFile file(":/qss/psblack.css");
if (file.open(QFile::ReadOnly)) {
    QString qss = QLatin1String(file.readAll());
    qApp->setStyleSheet(qss);
    file.close();
}

//其实一行代码就行
qApp->setStyleSheet("file:///:/qss/psblack.css");
//特别说明，只支持qApp->setStyleSheet 不支持其他比如widget->setStyleSheet
```

146. Qt中自带的很多控件，其实都是由一堆基础控件（QLabel、QPushButton等）组成的，比如日历面板 QCalendarWidget 就是 QToolButton+QSpinBox+QTableView 等组成，妙用 findChildren 可以拿到父类对应的子控件集合，可以直接对封装的控件中的子控件进行样式的设置，其他参数的设置比如设置中文文本（默认可能是英文）等。

```
//打印子类类名集合
```

```
void printObjectChild(const QObject *obj, int spaceCount)
{
    qDebug() << QString("%1%2 : %3")
                .arg("", spaceCount)
                .arg(obj->metaObject()->className())
                .arg(obj->objectName());

    QObjectList childs = obj->children();
    foreach (QObject *child, childs) {
        printObjectChild(child, spaceCount + 2);
    }
}

//拿到对话框进行设置和美化
QFileDialog *fileDialog = new QFileDialog(this);
fileDialog->setOption(QFileDialog::DontUseNativeDialog, true);
QLabel *lookinLabel = fileDialog->findChild<QLabel*>("lookInLabel");
lookinLabel->setText(QString::fromLocal8Bit("文件目录："));
lookinLabel->setStyleSheet("color:red;");

//设置日期框默认值为空
QLineEdit *edit = ui->dateEdit->findChild<QLineEdit *>
("qt_spinbox_lineedit");
if (!edit->text().isEmpty()) {
    edit->clear();
}
```

147. Qt内置了各种对话框，比如文件对话框-QFileDialog，颜色对话框-QColorDialog，默认都会采用系统的对话框风格样式，这样可以保持和系统一致，如果不需要的话可以取消该特性，取消以后会采用Qt自身的对话框，这样才能进行美化和其他处理。

```
QFileDialog *fileDialog = new QFileDialog(this);
//不设置此属性根本查找不到任何子元素,因为默认采用的系统对话框
fileDialog->setOption(QFileDialog::DontUseNativeDialog, true);
qDebug() << fileDialog->findChildren<QLabel *>();
//打印输出 QLabel(0x17e2ff68, name="lookInLabel"), QLabel(0x17e35f88,
name="fileNameLabel"), QLabel(0x17e35e68, name="fileTypeLabel")
```

148. QtCreator集成开发环境，也内置了对快速添加注释的支持，比如最常用的在头文件开头添加一大段通用模板的注释，标注文件创建者、时间等信息。

- 菜单->工具->选项->文本编辑器->右侧tab页面片段(snippets);
- 组选择C++, 可以看到这里面已经内置了不少定义比如foreach，可以依葫芦画瓢;
- 添加一个片段, 比如名字是fun, 触发种类是这个片段的简单描述;
- 当我们在代码文件中键入fun时, 会自动弹出智能提醒, 选择我们的代码片段回车, 自动填充代码;
- 按tab可以在变量间切换, 输入完成后回车, 完成编辑;

```
/**
  * @brief $name$
  * @param $param$
  * @author feiyangqingyun
  * @date $date$
  */
$ret$ $name$($param$)
{
    $$
}
```

149. Qt5时代对信号槽运行机制据说有了很大的改进。

- 在Qt5之前，connect一般都只能这么写connect(sender, SIGNAL(signalFunc()), receiver, SLOT(receiveFunc()))，就是说在connect的时候，必须把信号用宏SIGNAL包裹起来，把槽函数用宏SLOT包裹起来，这样才能被Qt的Moc机制识别；
- 在编译的时候即使信号或槽不存在或者参数不正确也不会报错，但是在执行的时候无效，会打印提示，对于C++这种静态语言来说，这是不友好的，不利于调试；
- 但是Qt5之后更加推荐"取地址的写法"，采用这种写法，如果编译的时候信号或槽不存在是无法编译通过的，相当于编译时检查，不容易出错；
- 如果没有历史遗留问题需要兼容Qt4的话，还是推荐用新写法，有类型检查更严格，而且支持的写法多样非常灵活；
- 一些简单的处理逻辑强烈推荐直接lambda表达式直接处理完；

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public: MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;

private:
    void test_fun();

private slots:
    void test_slot();
};

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    //早期写法,通用Qt所有版本,只支持定义了slots关键字的函数
    //connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(test_fun()));
    connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(test_slot()));

    //新写法,支持Qt5及后期所有版本,支持所有函数,无需定义slots关键字也行
    connect(ui->pushButton, &QPushButton::clicked, this,
&MainWindow::test_fun);
```

```
    connect(ui->pushButton, &QPushButton::clicked, this,
&MainWindow::test_slot);

    //另类写法,支持lambda表达式,直接执行代码
    connect(ui->pushButton, &QPushButton::clicked, [this] {test_fun();});
    connect(ui->pushButton, &QPushButton::clicked, [this] {
        qDebug() << "hello lambda";
    });

    //lambda带参数
    connect(ui->pushButton, &QPushButton::clicked, [&] (bool isCheck) {
        qDebug() << "hello lambda" << isCheck;
    });

    //头文件 signals:void sig_test(int i);
    connect(this, &MainWindow::sig_test, [] (int i) {
        qDebug() << "hello lambda" << i;
    });
    emit sig_test(5);
}

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

void MainWindow::test_fun()
{
    qDebug() << "test_fun";
}

void MainWindow::test_slot()
{
    qDebug() << "test_slot";
}
```

150. Qt样式表有多种运行机制，主要是考虑到各种需求场景，继承自QWidget的类和qApp类都支持setStyleSheet方法，还可以统一将样式表放在文件，或者将样式文件加入到资源文件。

- 斗气：qss内容写得到处都是，哪里需要就写在哪里，各种控件调用 setStyleSheet方法传入样式表内容，或者直接对应控件鼠标右键弹出菜单选择改变样式表填入内容；
- 斗者：qss内容放在文件，读取文件内容设置样式表，程序发布的时候带上qss文件；
- 斗师：qss文件作为资源文件放到qrc文件，直接编译到可执行文件中，防止篡改；
- 斗灵：在qss文件中自定义一些标志充当变量使用，读取以后替换对应的变量为颜色值，类似动态换肤；
- 斗王：放在文件容易被篡改，集成到可执行文件不够灵活，一旦样式表更新需要重新编译文件，如何做到既能只更新样式表文件，又不需要重新编译可执行文件，又能防止被篡改：采用rcc命令将资源文件编译生成二进制，只需要替换该二进制文件即可；
- 斗皇：继承qstyle类自己实现完成所有样式接口，统一整体风格，大名鼎鼎的UOS系统默认规则就是如此，不允许用样式表，全部painter绘制；

## 16：151-160

151. 当Qt中编译资源文件太大时，效率很低，或者需要修改资源文件中的文件比如图片、样式表等，需要重新编译可执行文件，这样很不友好，当然Qt都给我们考虑好了策略，此时可以将资源文件转化为二进制的rcc文件，这样就将资源文件单独出来了，可在需要的时候动态加载。

```
//Qt中使用二进制资源文件方法如下
//将qrc编译为二进制文件rcc，在控制台执行下列命令
rcc -binary main.qrc -o main.rcc
//在应用程序中注册资源，一般在main函数启动后就注册
QResource::registerResource(qApp->applicationDirPath() + "/main.rcc");
```

152. 关于设置字体，大概都会经历一个误区，本来是打算设置整个窗体包括子控件的字体大小的，结果发现只有主窗体自己应用了字体而子控件没有。

```
//假设窗体中有子控件，默认字体12px，父类类型是QWidget，父类类名是Widget

//下面几种方法只会设置主窗体的字体，子控件不会应用，需要按个调用setFont
QFont font;
font.setPixelSize(20);
this->setFont(font);
this->setStyleSheet("{font:26px;}");
this->setStyleSheet("QWidget{font:26px;}");
this->setStyleSheet("Widget{font:26px;}");

//下面才是通过样式表设置整个控件+子控件的字体
this->setStyleSheet("font:26px;");
this->setStyleSheet("*{font:26px;}");
this->setStyleSheet("QWidget>*{font:26px;}");
this->setStyleSheet("Widget>*{font:26px;}");

//下面设置全局字体
qApp->setFont(font);
```

153. Qt中封装的QImage异常的强大，提供了各种图片格式的转换，还可以对每个像素的颜色值进行替换，有时候我们需要将单色的图片换成另外一种颜色，要注意的是如果带有透明值的颜色需要进行格式转化，比如转成Format_ARGB32或者Format_RGBA8888。

```
//pixel      函数获取像素点的颜色 setPixel      函数设置像素点的颜色 此函数任意Qt版本都有
//pixelColor 函数获取像素点的颜色 setPixelColor 函数设置像素点的颜色 此函数Qt5.6以后才有
//pixel函数取出来的是QRgb格式需要用 qRed qGreen qBlue qAlpha 进行转换
QImage image("1.png");
image = image.convertToFormat(QImage::Format_ARGB32);
int width = image.width();
int height = image.height();
//遍历图像的每一个像素
for (int x = 0; x < width; ++x) {
    for (int y = 0; y < height; ++y) {
        QString name = image.pixelColor(x, y).name();
        //将白色以外的颜色全部替换成红色
        if (name != "#ffffff") {
```

```
            image.setPixelColor(x, y, Qt::red);
        }
    }
}

//保存文件
image.save("2.png");
```

154. 在数据库相关的应用中，如果仅仅是单机版本，没有特别的需要（比如领导指定，或者需要远程存放数据），强烈建议使用sqlite数据库，这是本人经过无数次的对比测试和N个商业项目应用得出的结论。

- Qt天生内置了sqlite数据库，只需要发布的时候带上插件就行（可以看到插件动态库文件比其他几种都要大，那是因为直接将数据库的源码都编译进去了，而其他只编译了中间通信交互的插件源码），其他数据库要么还要带上动态库，要么还需要创建数据源；
- 速度上，绝对无与伦比的出类拔萃，同样的数据库结构（表结构、索引等完全一致），查询速度和批量更新速度、数据库事务等，速度都是其他几种的至少3倍以上，而且随着数据量的增大对比越发明显；
- 几千万的数据量完全没问题，而且速度和性能都还可以，不要以讹传讹网上部分菜鸡说的不支持百万以上的数据量，本人亲测亿级别，数据量建议千万级别以下，着重注意数据库表和索引的设计；
- 其他数据库还要注意版本的区别，ODBC数据源形式还容易出错和执行失败；
- sqlite数据库也有几个重大缺点：不支持加密，不支持网络访问，不支持部分数据库高级特性，不支持海量数据（亿级别以上），但是对于绝大部分Qt项目还是足够；
- 数据库支持友好度大致是 sqlite > postgresql > mysql > odbc ；
- 以上都是在Qt环境中个人测试得出的结论，结果未必正确，作为参考即可，其他编程环境比如C#、JAVA请忽略，也许差别可能在中间通信的效率造成的；

155. Qt5.10以后提供了新的类 QRandomGenerator QRandomGenerator64 管理随机数，使用更方便，尤其是取某个区间的随机数。

```
//早期处理办法 先初始化随机数种子然后取随机数
qsrand(QTime::currentTime().msec());
//取 0-10 之间的随机数
qrand() % 10;
//取 0-1 之间的浮点数
qrand() / double(RAND_MAX);

//新版处理办法 支持5.10以后的所有版本包括qt6
QRandomGenerator::global()->bounded(10);       //生成一个0和10之间的整数
QRandomGenerator::global()->bounded(10.123);   //生成一个0和10.123之间的浮点数
QRandomGenerator::global()->bounded(10, 15);   //生成一个10和15之间的整数

//兼容qt4-qt6及以后所有版本的方法 就是用标准c++的随机数函数
srand(QTime::currentTime().msec());
rand() % 10;
rand() / double(RAND_MAX);

//通用公式 a是起始值,n是整数的范围
int value = a + rand() % n;
//(min, max)的随机数
int value = min + 1 + (rand() % (max - min - 1));
//(min, max]的随机数
int value = min + 1 + (rand() % (max - min + 0));
//[min, max)的随机数
```

```
int value = min + 0 + (rand() % (max - min + 0));
//[min, max]的随机数
int value = min + 0 + (rand() % (max - min + 1));

//如果在线程中取随机数，线程启动的时间几乎一样，很可能出现取到的随机数一样的问题，就算设置随机数为当前时间啥的也没用，电脑太快很可能还是一样的时间，同一个毫秒。
//取巧办法就是在run函数之前最前面将当前线程的id作为种子设置。时间不可靠，线程的id才是唯一的。
//切记 void * 转换到数值必须用 long long，在32位是可以int但是在64位必须long，确保万一直接用quint64最大
srand((long long)currentThreadId());
qrand((long long)currentThreadId());
```

156. Qt的UI界面在resize以后有个BUG，悬停样式没有取消掉，需要主动模拟鼠标动一下。

```
void frmMain::on_btnMenu_Max_clicked()
{
    ......

    //最大化以后有个BUG,悬停样式没有取消掉,需要主动模拟鼠标动一下
    QEvent event(QEvent::Leave);
    QApplication::sendEvent(ui->btnMenu_Max, &event);
}
```

157. 项目中启用c++11语法支持。

```
greaterThan(QT_MAJOR_VERSION, 4): CONFIG += c++11
lessThan(QT_MAJOR_VERSION, 5): QMAKE_CXXFLAGS += -std=c++11
```

158. Qt的文本控件比如QTextEdit默认加载大文本比如10MB的文本，很容易卡死甚至崩溃，那是因为默认一个属性开启了，需要屏蔽掉就好很多。

```
ui->textEdit->setUndoRedoEnabled(false);
```

159. 其他几点常规小经验，本人在这几个地方摔跤过很多次。

- 有返回值的函数，一定要主动return返回值，有部分编译器在没有返回值的情况下也能正常编译通过，但是运行的时候会出问题，得不到想要的结果，因为没有return对应的值。
- 定义的局部变量，主动给定个初始值，是个必须养成的好习惯，不然编译器给的初始值很可能不是你想要的，比如int变量默认0，有时候随机变成一个很大的数值，bool变量的初始值不同编译器不同值，有些是true有些是false，主动给一个初始值更可靠。
- 某些函数参数很多，而且后期可能还会修改和增加，这就导致了源头修改以后，关联信号槽的地方也要修改，参数类型和位置必须保持完全一致，对应槽函数处理也要修改等，改动的工作量非常大而且极不友好，所以对于非固定参数的函数，建议用结构体，这样非常容易增加其他的参数，而且不用修改信号槽关联和信号槽函数定义等，比如学生信息表、商品信息表作为参数传输，最佳方案就是结构体。

160. QTabWidget选项卡控件，生成的tabbar选项卡宽度是按照文本自动设置的，文本越长选项卡的宽度越大，很多时候，我们需要的是一样的宽度或者等分填充，

```
//方法1：字符串空格填充
ui->tabWidget->addTab(httpClient1, "测    试");
ui->tabWidget->addTab(httpClient1, "人员管理");
```

```
ui->tabWidget->addTab(httpClient1, "系统设置");

//方法2：识别尺寸改变事件自动设置最小宽度
void MainWindow::resizeEvent(QResizeEvent *e)
{
    int count = ui->tabWidget->tabBar()->count();
    int width = this->width() - 30;
    QString qss = QString("QTabBar::tab{min-width:%1px;}").arg(width / count);
    this->setStyleSheet(qss);
}

//方法3：设置全局样式，不同选项卡个数的设置不同的宽度
QStringList list;
list << QString("QTabWidget[tabCount=\"2\"]>QTabBar::tab{min-width:%1px;}").arg(100);
list << QString("QTabWidget[tabCount=\"3\"]>QTabBar::tab{min-width:%1px;}").arg(70);
qApp->setStyleSheet(list.join(""));
//设置了tabCount弱属性自动去找对应的宽度设置
ui->tabWidget->setProperty("tabCount", 2);
ui->tabWidget->setProperty("tabCount", 3);

//方法4：强烈推荐-》使用内置的方法 setExpanding setDocumentMode 两个属性都必须设置
//Qt4的tabBar()是propected的，所以建议还是通过样式表设置
ui->tabWidget->tabBar()->setDocumentMode(true);
ui->tabWidget->tabBar()->setExpanding(true);
//样式表一步到位不用每个都单独设置
QString("QTabBar{qproperty-usesScrollButtons:false;qproperty-documentMode:true;qproperty-expanding:true;}");
//在5.9以前开启这个设置后，貌似选项卡个数按照真实个数+1计算宽度，也就是永远会留空一个tab的占位。
//5.9以后貌似修复了这个BUG，按照理想中的拉伸填充等分设置tab的宽度。
```

## 17：161-170

161. 经常有人说Qt垃圾，说用Qt在1毫秒绘制几千个数据点卡成屎。其实显示器最高刷新频率一般才60帧，1毫秒就绘制一次有意义吗？不仅显示器没刷新过来，人肉眼也看不过来（有人可能又要抬杠说这是老板要求的，显示归显示，至于人看不看那是另外一回事，我想说的是显示不就是给人看的吗？给程序看可以直接后台绘制图片让程序识别啊没必要显示的），程序中要做的应该是尽量降低程序的绘制刷新频率到显示器的频率（其实一秒钟30帧都足够），一次搞多一点的数据一次性绘制（数据量很大还可以考虑重采样，比如平均值法等，毕竟要考虑显示器的分辨率就那么大，搞个几十万的数据点挤一块没啥意思，可以将一整块区域内的数据点换成一个点），而不是绘制多次，尽管两种办法都可以将收到的数据绘制完成，但是效率相差的不是一点点，信号也是如此，不建议太频繁的发送信号，Qt内部1秒钟处理信号的个数也是有限制的，太频繁高并发的信号，很可能会丢失或者合并一部分，比如网络请求接收到的学生信息表，应该是在该应答数据内的所有学生信息解析完一次性发送，而不是解析一条发送一条。
162. Qt提供了N种窗体属性比如无边框属性FramelessWindowHint、不在任务栏显示属性Tool等，有时候我们需要对窗口的属性进行动态设置，比如增加一个属性或者移除一个属性，Qt5.9以前需要拿到原有的窗体属性做运算，后面可以用新的方法。

```
//增加一个无边框属性
setWindowFlags(windowFlags() | Qt::FramelessWindowHint);
//移除无边框属性
setWindowFlags(windowFlags() & ~Qt::FramelessWindowHint);

//下面是5.9以后新增的方法
//增加一个无边框属性到窗体属性链表
setWindowFlag(Qt::FramelessWindowHint, true);
//从窗体属性链表中移除无边框属性
setWindowFlag(Qt::FramelessWindowHint, false);
```

163. 如果对窗体设置了固定尺寸，窗体会变得大小不可拉伸，如果需要重新还原可拉伸，必须重新设置最小尺寸和最大尺寸。

```
setMinimumSize(0, 0);
setMaximumSize(QWIDGETSIZE_MAX, QWIDGETSIZE_MAX);
```

164. Qt内置了很多全局的对象参数可以直接获取，这样在使用的时候方便的不要不要的，比如判断当前鼠标左键还是右键可以直接用qApp->mouseButtons()，全局的鼠标坐标可以用QCursor::pos()。

```
//在鼠标右键的地方弹出菜单，如果菜单是QMenu而不是QAction则只能通过下面的方式弹出
if (qApp->mouseButtons() == Qt::RightButton) {
    videoMenu->exec(QCursor::pos());
}

//全局剪切板
qApp->clipboard();
//顶层控件对象集合
qApp->topLevelWidgets()
//当前焦点所在控件
qApp->focusWidget()
//当前平台名称
qApp->platformName()
//调用系统蜂鸣器
qApp->beep()
//打印当前Qt版本信息
qApp->aboutQt()
//设置全局的鼠标样式
qApp->setOverrideCursor()
//不使用系统的标准颜色字体等
QGuiApplication::setDesktopSettingsAware(bool on);
QApplication app(argc, argv);

//更多的全局对象属性等可以查阅 qguiapplication.h 头文件，你会发现新大陆。
```

165. Qt对区分不同的编译器也做了非常细致的处理。

```
#pro文件可以这样判断
msvc {
//要做的处理
}

mingw {
```

```
//要做的处理
}

//代码中可以这样判断
#ifdef Q_CC_MINGW
//mingw编译器
#elif Q_CC_MSVC
//msvc编译器
#endif

//判断编译器和编译器版本
#if defined Q_CC_MSVC && _MSC_VER < 1300
#if defined(Q_CC_GNU) && (__GNUC__ < 4)

//代码中判断ARM平台
#ifdef QT_ARCH_ARM
//多个条件判断
#if defined(QT_ARCH_ARM) || defined(QT_ARCH_WINDOWSCE)
```

166. 有时候需要暂时停止某个控件发射信号（比如下拉框combobox添加数据的时候会触发当前元素改变信号），有多种处理，推荐用 blockSignals 方法。

```
//方法1：先 disconnect 掉信号，处理好以后再 connect 信号，缺点很明显，很傻，如果信号很多，每个型号都要这么来一次。
disconnect(ui->cbox, SIGNAL(currentIndexChanged(int)), this, SLOT(on_cbox_currentIndexChanged(int)));
for (int i = 0; i <= 100; i++) {
    ui->cbox->addItem(QString::number(i));
}
connect(ui->cbox, SIGNAL(currentIndexChanged(int)), this, SLOT(on_cbox_currentIndexChanged(int)));

//方法2：先调用 blockSignals(true) 阻塞信号，处理号以后再调用 blockSignals(false) 恢复所有信号。
//如果需要指定某个信号进行断开那就只能用 disconnect 来处理。
ui->cbox->blockSignals(true);
for (int i = 0; i <= 100; i++) {
    ui->cbox->addItem(QString::number(i));
}
ui->cbox->blockSignals(false);
```

167. 项目代码文件数量如果很多的话，全部包含在pro项目文件中会显得非常凌乱，甚至滚动条都要拉好久，有两个方法可以处理的更好，推荐方法2。

```
//方法1：pro文件直接全部引入，而不是每个都添加一次，省心省力。
HEADERS += *.h
SOURCES += *.cpp

//方法2：分模块文件夹存放，不同模块用pri包含代码文件，比如界面可以放在ui文件夹，下面搞个ui.pri，然后pro项目文件只需要引入这个pri文件即可。
include($$PWD/ui/ui.pri)
//还可以加上一句包含路径这样可以省去在使用代码的时候不用写文件夹
INCLUDEPATH += $$PWD/ui
//加上上面这行，在使用头文件的时候可以直接 include "form.h"，没有加则需要 include "ui/form.h"。
```

168. 在网络通信中，无论是tcp客户端还是udp客户端，其实都是可以绑定网卡IP和端口的，很多人只知道服务端可以指定网卡监听端口。客户端如果没有绑定通信端口则由客户端所在的操作系统随机递增分配的，这里为啥这么强调，因为无数人，甚至不乏一些多年经验的新时代农名工，以为客户端的端口是服务端分配的，因为他们看到在服务端建立连接后可以打印出不同的端口号。网络通信的双方自己决定自己要用什么端口，服务器端只能决定自己监听的是哪个端口，不能决定客户端的端口，同理客户端也只能决定自己的端口。端口随机分配一般是按照顺序递增的，比如先是45110端口，连接重新建立就用45111端口，只要端口没被占用就这样递增下去，所以很多人会问是否可以复用一些端口，不然端口一直这样频繁的分配下去不妥，甚至有些特定的场景和需求也是会要求客户端绑定网卡和端口来和服务器通信的。

```
//tcp客户端
QTcpSocket *socket = new QTcpSocket(this);
//断开所有连接和操作
socket->abort();
//绑定网卡和端口
socket->bind(QHostAddress("192.168.1.2"), 6005);
//连接服务器
socket->connectToHost("192.168.1.3", 6000);

//打印通信用的本地绑定地址和端口
qDebug() << socket->localAddress() << socket->localPort();
//打印通信服务器对方的地址和端口
qDebug() << socket->peerAddress() << socket->peerPort() << socket->peerName();

//udp客户端
QUdpSocket *socket = new QUdpSocket(this);
//绑定网卡和端口,没有绑定过才需要绑定
//采用端口是否一样来判断是为了方便可以直接动态绑定切换端口
if (socket->localPort() != 6005) {
    socket->abort();
    socket->bind(QHostAddress("192.168.1.2"), 6005);
}
//指定地址和端口发送数据
socket->writeDatagram(buffer, QHostAddress("192.168.1.3"), 6000);

//上面是Qt5可以使用bind，Qt4中的QTcpSocket的对应接口是protected的没法直接使用，需要继承类重新实现把接口放出来。
//Qt4中的QUdpSocket有bind函数是开放的，奇怪了，为何Qt4中独独QTcpSocket不开放。
TcpSocket *socket = new TcpSocket(this);
socket->setLocalAddress(QHostAddress("192.168.1.2"));
socket->setLocalPort(6005);
```

169. 关于网络通信，tcp和udp是两种不同的底层的网络通信协议，两者监听和通信的端口互不相干的，不同的协议或者不同的网卡IP地址可以用相同的端口。之前有个人说他的电脑居然可以监听一样的端口进行通信，颠覆了他以前的认知，书上说的明明是不可以相同端口的，后面远程一看原来选择的不同的网卡IP地址，当然可以的咯。

- tcp对网卡1监听了端口6000，还可以对网卡2监听端口6000。
- tcp对网卡1监听了端口6000，udp对网卡1还可以继续监听端口6000。
- tcp对网卡1监听了端口6000，在网卡1上其他tcp只能监听6000以外的端口。
- udp协议也是上面的逻辑。

170. 开源的图表控件QCustomPlot很经典，在曲线数据展示这块性能彪悍，总结了一些容易忽略的经验要点。

- 可以将XY轴对调，然后形成横向的效果，无论是曲线图还是柱状图，分组图、堆积图等，都支持这个特性。
- 不需要的提示图例可以调用 legend->removeItem 进行移除。
- 两条曲线可以调用 setChannelFillGraph 设置合并为一个面积区域。
- 可以关闭抗锯齿 setAntialiased 加快绘制速度。
- 可以设置不同的线条样式（setLineStyle）、数据样式（setScatterStyle）。
- 坐标轴的箭头样式可更换 setUpperEnding。
- 可以用 QCPBarsGroup 实现柱状分组图，这个类在官方demo中没有，所以非常容易忽略。
- V2.0开始支持数据排序设置，默认是交给QCustomPlot排序，也可以设置setData第三个参数为true表示已经排序过，这样可以绘制往回走的曲线。

```
//对调XY轴，在最前面设置
QCPAxis *yAxis = customPlot->yAxis;
QCPAxis *xAxis = customPlot->xAxis;
customPlot->xAxis = yAxis;
customPlot->yAxis = xAxis;

//移除图例
customPlot->legend->removeItem(1);

//合并两个曲线画布形成封闭区域
customPlot->graph(0)->setChannelFillGraph(customPlot->graph(1));

//关闭抗锯齿以及设置拖动的时候不启用抗锯齿
customPlot->graph()->setAntialiased(false);
customPlot->setNoAntialiasingOnDrag(true);

//多种设置数据的方法
customPlot->graph(0)->setData();
customPlot->graph(0)->data()->set();

//设置不同的线条样式、数据样式
customPlot->graph()->setLineStyle(QCPGraph::lsLine);
customPlot->graph()->setScatterStyle(QCPScatterStyle::ssDot);
customPlot->graph()->setScatterStyle(QCPScatterStyle(shapes.at(i), 10));

//还可以设置为图片或者自定义形状
customPlot->graph()->setScatterStyle(QCPScatterStyle(QPixmap("./sun.png")));
QPainterPath customScatterPath;
for (int i = 0; i < 3; ++i) {
```

```
    customScatterPath.cubicTo(qCos(2 * M_PI * i / 3.0) * 9, qSin(2 * M_PI * i / 3.0) * 9, qCos(2 * M_PI * (i + 0.9) / 3.0) * 9, qSin(2 * M_PI * (i + 0.9) / 3.0) * 9, 0, 0);
}
customPlot->graph()->setScatterStyle(QCPScatterStyle(customScatterPath, QPen(Qt::black, 0), QColor(40, 70, 255, 50), 10));

//更换坐标轴的箭头样式
customPlot->xAxis->setUpperEnding(QCPLineEnding::esSpikeArrow);
customPlot->yAxis->setUpperEnding(QCPLineEnding::esSpikeArrow);

//设置背景图片
customPlot->axisRect()->setBackground(QPixmap("./solarpanels.jpg"));
//画布也可以设置背景图片
customPlot->graph(0)->setBrush(QBrush(QPixmap("./balboa.jpg")));
//整体可以设置填充颜色或者图片
customPlot->setBackground(QBrush(gradient));
//设置零点线条颜色
customPlot->xAxis->grid()->setZeroLinePen(Qt::NoPen);
//控制是否鼠标滚轮缩放拖动等交互形式
customPlot->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom | QCP::iSelectPlottables);

//柱状分组图
QCPBarsGroup *group = new QCPBarsGroup(customPlot);
QList<QCPBars*> bars;
bars << fossil << nuclear << regen;
foreach (QCPBars *bar, bars) {
    //设置柱状图的宽度大小
    bar->setWidth(bar->width() / bars.size());
    group->append(bar);
}
//设置分组之间的间隔
group->setSpacing(2);

//绘制往回走的曲线
QVector<double> keys, values;
keys << 0 << 1 << 2 << 3 << 4 << 5 << 4 << 3;
values << 5 << 4 << 6 << 7 << 7 << 6 << 5 << 4;
customPlot->graph(0)->setData(keys, values, true);
```

## 18：171-180

171. 在Qt编程中经常会遇到编码的问题，由于跨平台的考虑兼容各种系统，而windows系统默认是gbk或者gb2312编码，当然后期可能msvc编译器都支持utf8编码，所以在部分程序中传入中文目录文件名称的时候会发现失败，因为可能对应的接口用了早期的fopen函数而不是fopen_s函数，比如fmod中也是这个情况。这个时候就需要转码处理。

```
QString fileName = "c:/测试目录/1.txt";
//如果应用程序main函数中没有设置编码则默认采用系统的编码，可以直接通过toLocal8Bit转成正确的数据
const char *name = fileName.toLocal8Bit().constData();

//如果设置过了下面两句则需要主动转码
```

```
QTextCodec *codec = QTextCodec::codecForName("utf-8");
QTextCodec::setCodecForLocale(codec);

QTextCodec *code = QTextCodec::codecForName("gbk");
const char *name = code->fromUnicode(fileName).constData();

//推荐方式2以防万一保证绝对的正确，哪怕是设置过主程序的编码
//切记一旦设置过QTextCodec::setCodecForLocale会影响toLocal8Bit

//有时候可能还有下面这种情况
#ifdef Q_OS_WIN
#if defined(_MSC_VER) && (_MSC_VER >= 1400)
    QTextCodec *code = QTextCodec::codecForName("utf-8");
#else
    QTextCodec *code = QTextCodec::codecForName("gbk");
#endif
    const char *name = code->fromUnicode(fileName).constData();
#else
    const char *name = fileName.toUtf8().constData();
#endif
```

172. 在查阅和学习Qt源码的过程中，发现了一些趋势和改变。

- 数据类型这块尽量用Qt内部的数据类型，哪怕是重定义过的比如quint8其实unsigned char，qreal就是double，以前翻看源码的时候可能还有些是double，现在慢慢改成了qreal。
- 循环结构用 for(;;) 替代 while(1)，因为转成汇编指令后 for(;;) 只有一条指令而 while(1) 确有4条，指令少不占用寄存器而且不用跳转，理论上速度要更快。
- 其实Qt中就重定义了 forever 关键字表示 for(;;)，我的乖乖，想的真周到。
- 自动c++11以及后续的标准都支持auto万能数据类型，发现Qt的源码中也慢慢的改成了auto，这样加快了编写代码的效率，不用自己去指定数据类型而是让编译器自己推导数据类型。而且其实也不影响编译器编译的速度，因为无论指定和没有指定数据类型，编译器都要推导右侧的数据类型进行判断。不过有个缺点就是影响了阅读代码的成本，很多时候需要自己去理解推导。
-

173. Qt中设置或者打开加载本地文件需要用到QUrl类，本地文件建议加上 file:/// 前缀。

```
QString url = "file:///c:/1.html";
//浏览器控件打开本地网页文件
webView->setUrl(QUrl(url));
//打开本地网页文件，下面两种方法都可以
QDesktopServices::openUrl(QUrl::fromLocalFile(url));
QDesktopServices::openUrl(QUrl(url, QUrl::TolerantMode));
```

174. 在网络请求中经常涉及到超时时间的问题，因为默认是30秒钟，一旦遇到网络故障的时候要等好久才能反应过来，所以需要主动设置下超时时间，超过了就直接中断结束请求。从Qt5.15开始内置了setTransferTimeout来设置超时时间，非常好用。

```
//局部的事件循环,不卡主界面
QEventLoop eventLoop;

//设置超时 5.15开始自带了超时时间函数 默认30秒
#if (QT_VERSION >= QT_VERSION_CHECK(5,15,0))
manager->setTransferTimeout(timeout);
#else
```

```
QTimer timer;
connect(&timer, SIGNAL(timeout()), &eventLoop, SLOT(quit()));
timer.setSingleShot(true);
timer.start(timeout);
#endif

QNetworkReply *reply = manager->get(QNetworkRequest(QUrl(url)));
connect(reply, SIGNAL(finished()), &eventLoop, SLOT(quit()));
eventLoop.exec();

if (reply->bytesAvailable() > 0 && reply->error() == QNetworkReply::NoError)
{
    //读取所有数据保存成文件
    QByteArray data = reply->readAll();
    QFile file(dirName + fileName);
    if (file.open(QFile::WriteOnly | QFile::Truncate)) {
        file.write(data);
        file.close();
    }
}
```

175. Qt中基本上有三大类型的项目，控制台项目对应QCoreApplication、传统QWidget界面程序对应QApplication、quick/qml项目程序对应QGuiApplication。有很多属性的开启需要在main函数的最前面执行才有效果，比如开启高分屏支持、设置opengl模式等。不同类型的项目需要对应的QApplication。

```
//如果是控制台程序则下面的QApplication换成QCoreApplication
//如果是quick/qml程序则下面的QApplication换成QGuiApplication
int main(int argc, char *argv[])
{
    //可以用下面这行测试Qt自带的输入法 qtvirtualkeyboard
    qputenv("QT_IM_MODULE", QByteArray("qtvirtualkeyboard"));

    //设置不应用操作系统设置比如字体
    QApplication::setDesktopSettingsAware(false);

#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
    //设置高分屏缩放舍入策略

 QApplication::setHighDpiScaleFactorRoundingPolicy(Qt::HighDpiScaleFactorRoundingPolicy::Floor);
#endif
#if (QT_VERSION > QT_VERSION_CHECK(5,6,0))
    //设置启用高分屏缩放支持
    //要注意开启后计算到的控件或界面宽度高度可能都不对,全部需要用缩放比例运算下
    QApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    //设置启用高分屏图片支持
    QApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);
#endif
#if (QT_VERSION > QT_VERSION_CHECK(5,4,0))
    //设置opengl模式 AA_UseDesktopOpenGL(默认) AA_UseOpenGLES AA_UseSoftwareOpenGL
    //在一些很旧的设备上或者对opengl支持很低的设备上需要使用AA_UseOpenGLES表示禁用硬件加速
    //如果开启的是AA_UseOpenGLES则无法使用硬件加速比如ffmpeg的dxva2
```

```
    //QApplication::setAttribute(Qt::AA_UseOpenGLES);
    //设置opengl共享上下文
    QApplication::setAttribute(Qt::AA_ShareOpenGLContexts);
#endif

    QApplication a(argc, argv);
    QWidget w;
    w.show();
    return a.exec();
}
```

176. QCamera中获取设备的配置参数比如支持的分辨率集合等，需要先调用load后才能正确获取，或者关联stateChanged信号中判断状态是否是ActiveState，然后再读取。

```
//方法1：调用load后获取
camera = new QCamera(this);
//先需要载入才能获取到对应参数
camera->load();
//输出当前设备支持的分辨率
QList<QSize> sizes = camera->supportedViewfinderResolutions();
emit resolutions(sizes);
//重新设置分辨率
QCameraViewfinderSettings set;
set.setResolution(cameraWidth, cameraHeight);
camera->setViewfinderSettings(set);
//获取完成后卸载
camera->unload();

//方法2：通过事件信号获取
camera = new QCamera(this);
connect(camera, SIGNAL(stateChanged(QCamera::State)), this,
SLOT(stateChanged(QCamera::State)));
void CameraThread::stateChanged(QCamera::State state)
{
    if (state == QCamera::ActiveState) {
        //输出当前设备支持的分辨率
        QList<QSize> sizes = camera->supportedViewfinderResolutions();
        emit resolutions(sizes);
        //重新设置分辨率
        QCameraViewfinderSettings set;
        set.setResolution(cameraWidth, cameraHeight);
        camera->setViewfinderSettings(set);
    }
}

//QCamera没有指定设备名称的时候则采用默认的摄像机
camera = new QCamera(this);
//cameraName = @device:pnp:\\\\?
\\usb#vid_046d&pid_0825&mi_00#6&212eebd3&0&0000#{65e8773d-8f56-11d0-a3b9-
00a0c9223196}\\global
//可以通过设备描述符来查找设备名称(唯一标识)
camera = new QCamera(cameraName.toUtf8(), this);
```

177. 很多时候需要在窗体首次显示的时候加载一些东西，而且只加载一次，当窗体再次显示的时候不加载。为什么不是在构造函数呢？因为很多玩意都是要在显示后才能确定，比如控件的尺寸，部分样式表的应用。

```
void Widget::showEvent(QShowEvent *)
{
    static bool isLoad = false;
    if (!isLoad) {
        isLoad = true;
        //执行对应的处理
    }
}
```

178. Qt获取当前所用的Qt版本、编译器、位数等信息。

```
//详细的Qt版本+编译器+位数
QString compilerString = "<unknown>";
{
#if defined(Q_CC_CLANG)
    QString isAppleString;
#if defined(__apple_build_version__)
    isAppleString = QLatin1String(" (Apple)");
#endif
    compilerString = QLatin1String("Clang ") +
QString::number(__clang_major__) + QLatin1Char('.') +
QString::number(__clang_minor__) + isAppleString;
#elif defined(Q_CC_GNU)
    compilerString = QLatin1String("GCC ") + QLatin1String(__VERSION__);
#elif defined(Q_CC_MSVC)
    if (_MSC_VER > 1999) {
        compilerString = QLatin1String("MSVC <unknown>");
    } else if (_MSC_VER >= 1920) {
        compilerString = QLatin1String("MSVC 2019");
    } else if (_MSC_VER >= 1910) {
        compilerString = QLatin1String("MSVC 2017");
    } else if (_MSC_VER >= 1900) {
        compilerString = QLatin1String("MSVC 2015");
    } else if (_MSC_VER >= 1800) {
        compilerString = QLatin1String("MSVC 2013");
    } else if (_MSC_VER >= 1700) {
        compilerString = QLatin1String("MSVC 2012");
    } else if (_MSC_VER >= 1600) {
        compilerString = QLatin1String("MSVC 2010");
    } else {
        compilerString = QLatin1String("MSVC <old>");
    }
#endif
}

//拓展知识 查看 QSysInfo 类下面有很多好东西
// qVersion() = QT_VERSION_STR
QString version = QString("%1 %2
%3").arg(qVersion()).arg(compilerString).arg(QString::number(QSysInfo::WordS
ize));
```

179. QDateTime可以直接格式化输出星期几周几，Qt6默认按照英文输出比如 ddd = 周二 Tue  dddd = 星期二 Tuesday ，此时如果只想永远是中文就需要用到QLocale进行转换。

```
//格式化输出受到本地操作系统语言的影响

//英文操作系统
//这样获取到的是Mon到Sun，英文星期的3个字母的缩写。
QDateTime::currentDateTime().toString("ddd");
//这样获取到的是Monday到Sunday，英文星期完整单词。
QDateTime::currentDateTime().toString("dddd");

//中文操作系统
//这样获取到的是周一到周日。
QDateTime::currentDateTime().toString("ddd");
//这样获取到的是星期一到星期日。
QDateTime::currentDateTime().toString("dddd");

//主动指定语言转换
//如果没有指定本地语言则默认采用系统的语言环境。
QLocale locale;
//QLocale locale = QLocale::Chinese;
//QLocale locale = QLocale::English;
//QLocale locale = QLocale::Japanese;

//下面永远输出中文的周一到周日
locale.toString(QDateTime::currentDateTime(), "ddd");
//下面永远输出中文的星期一到星期日
locale.toString(QDateTime::currentDateTime(), "dddd");
```

180. QSqlTableModel大大简化了对数据库表的显示、添加、删除、修改等，唯独对数据库分页操作有点绕弯。

```
//实例化数据库表模型
QSqlTableModel *model = new QSqlTableModel(this);
//指定表名
model->setTable("table");
//设置列排序
model->setSort(0, Qt::AscendingOrder);
//设置提交模式
model->setEditStrategy(QSqlTableModel::OnManualSubmit);
//立即查询一次
model->select();
//将数据库表模型设置到表格上
ui->tableView->setModel(model);

//测试发现过滤条件中除了可以带where语句还可以带排序及limit等
model->setFilter("1=1 order by id desc limit 100");

//如果在过滤条件中设置了排序语句则不可以再使用setSort方法
//下面的代码结果是执行出错，可能因为setSort又重新增加了order by语句导致多个order by语句冲突了。
model->setSort(0, Qt::AscendingOrder);
model->setFilter("1=1 order by id desc limit 100");

```

```
//通过setFilter设置单纯的where语句可以不用加1=1
model->setFilter("name='张三'");
//如果还有其他语句比如排序或者limit等则需要最前面加上1=1
//下面表示按照id升序排序，查询结果显示第5-15条记录。
model->setFilter("1=1 order by id asc limit 5,10");

//多个条件用and连接
//建议任何时候用了setFilter则最前面写1=1最末尾加上 ; 防止有些地方无法正确执行。
model->setFilter("1=1 and name='张三' and result>=70;");

//下面表示查询姓名是张三的记录，按照id字段降序排序，结果从第10条开始100条，相当于从第10条到110条记录。
model->setFilter("1=1 and name='张三' order by id desc limit 10,100;");

//在第3行开始添加一条记录
model->insertRow(2);
//立即填充刚刚新增加的行，默认为空需要用户手动在表格中输入。
model->setData(model->index(2, 0), 100);
model->setData(model->index(2, 1), "张三");
//提交更新
model->submitAll();

//删除第4行
model->removeRow(3);
model->submitAll();

//总之有增删改操作后都需要调用model->submitAll();来真正执行，否则仅仅是数据模型更新了数据，并不会更新到数据库中。

//撤销更改
model->revertAll();
```

## 19：181-190

181. Qt天生就是linux的，从linux开始发展起来的，所以不少Qt程序员经常的开发环境是linux，比如常用的ubuntu等系统，整理了一点常用的linux命令。

| 命令 | 功能 |
| --- | --- |
| sudo -s | 切换到管理员，如果是 sudo -i 切换后会改变当前目录。 |
| apt install g++ | 安装软件包（要管理员权限），另一个派系的是 yum install。 |
| cd /home | 进入home目录。 |
| ls | 罗列当前所在目录所有目录和文件。 |
| ifconfig | 查看网卡信息包括IP地址，windows上是 ipconfig。 |
| tar -zxvf bin.tar.gz | 解压文件到当前目录。 |
| tar -jxvf bin.tar.xz | 解压文件到当前目录。 |
| tar -zxvf bin.tar.gz -C /home | 解压文件到/home目录，记住是大写的C。 |
| tar -zcvf bin.tar.gz bin | 将bin目录压缩成tar.gz格式文件（压缩比一般）。 |
| tar -jcvf bin.tar.xz bin | 将bin目录压缩成tar.xz格式文件（压缩比高，推荐）。 |
| tar -... | j z 表示不同的压缩方法，x表示解压，c表示压缩。 |
| gedit 1.txt | 用记事本打开文本文件。 |
| vim 1.txt | 用vim打开文件，很多时候可以缩写用vi。 |
| ./configure make -j4 make install | 通用编译源码命令，第一步./configure执行配置脚本，第二步make -j4启用多线程编译，第三步make install安装编译好的文件。 |
| ./configure -prefix /home/liu/Qt-5.9.3-static -static -sql-sqlite -qt-zlib -qt-xcb -qt-libpng -qt-libjpeg -fontconfig -system-freetype -iconv -nomake tests -nomake examples -skip qt3d -skip qtdoc | Qt通用编译命令。 |
| ./configure -static -release -fontconfig -system-freetype -qt-xcb -qt-sql-sqlite -qt-zlib -qt-libpng -qt-libjpeg -nomake tests -nomake examples -prefix /home/liu/qt/Qt5.6.3 | Qt静态带中文。 |
| ./configure -prefix /home/liu/Qt-5.9.3-static -static -release -nomake examples -nomake tests -skip qt3d | 精简编译命令。 |
| ./configure --prefix=host --enable-static --disable-shared --disable-doc | ffmpeg编译命令。 |

182. Qt自带的日志重定向机制非常简单好用，自从用了以后再也不用什么断点调试啥的了，在需要的地方支持qdebug输出对应的信息，而且发布程序以后也可以开启调试日志将其输出查看等。

```
//Qt5开始提供了日志上下文信息输出，比如输出当前打印消息所在的代码文件、行号、函数名等。
//如果是release还需要在pro中加上 DEFINES += QT_MESSAGELOGCONTEXT 才能输出上下文，默认release关闭的。
//切记不要在日志钩子函数中再写qdebug之类的，那样就死循环了。
//日志重定向一般就三种处理
//1: 输出到日志文件比如txt文本文件。
//2: 存储到数据库，可以分类存储，以便相关人员查询分析。
//3: 重定向到网络，对方用小工具连接程序后，所有打印信息通过tcp发过去。

//日志重定向
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
void Log(QtMsgType type, const QMessageLogContext &context, const QString &msg)
#else
void Log(QtMsgType type, const char *msg)
#endif
{
    //加锁,防止多线程中qdebug太频繁导致崩溃
    static QMutex mutex;
    QMutexLocker locker(&mutex);
    QString content;

    //这里可以根据不同的类型加上不同的头部用于区分
    switch (type) {
        case QtDebugMsg:
            content = QString("%1").arg(msg);
            break;

        case QtWarningMsg:
            content = QString("%1").arg(msg);
            break;

        case QtCriticalMsg:
            content = QString("%1").arg(msg);
            break;

        case QtFatalMsg:
            content = QString("%1").arg(msg);
            break;
    }

    //加上打印代码所在代码文件、行号、函数名
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
    if (SaveLog::Instance()->getUseContext()) {
        int line = context.line;
        QString file = context.file;
        QString function = context.function;
        if (line > 0) {
            content = QString("行号: %1  文件: %2  函数: %3\n%4").arg(line).arg(file).arg(function).arg(content);
        }
    }
```

```
#endif

    //将内容传给函数进行处理
    SaveLog::Instance()->save(content);
}

//安装日志钩子,输出调试信息到文件,便于调试
void SaveLog::start()
{
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
    qInstallMessageHandler(Log);
#else
    qInstallMsgHandler(Log);
#endif
}

//卸载日志钩子
void SaveLog::stop()
{
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
    qInstallMessageHandler(0);
#else
    qInstallMsgHandler(0);
#endif
}
```

183. 自从c++11标准以后，各种语法糖层出不穷，其中lambda表达式用的最广，基本上从Qt5以后就支持lambda表达式。对于习惯了c99的老一辈的程序员来说，这玩意是个新鲜事物，这里特意做个小理解笔记。

- 代码格式：capture mutable ->return-type {statement}
- [capture]：捕捉列表，捕捉列表总是出现在Lambda函数的开始处，实际上，[]是Lambda引出符，编译器根据该引出符判断接下来的代码是否是Lambda函数，捕捉列表能够捕捉上下文中的变量以供Lambda函数使用。
- (parameters)：参数列表，与普通函数的参数列表一致，如果不需要参数传递，则可以连同括号 () 一起省略。
- mutable：mutable修饰符，默认情况下，Lambda函数总是一个const函数，mutable可以取消其常量性。在使用该修饰符时，参数列表不可省略（即使参数为空）。
- ->return-type：返回类型，用追踪返回类型形式声明函数的返回类型，我们可以在不需要返回值的时候也可以连同符号 -> 一起省略。此外，在返回类型明确的情况下，也可以省略该部分，让编译器对返回类型进行推导。
- {statement}：函数体，内容与普通函数一样，不过除了可以使用参数之外，还可以使用所有捕获的变量。

捕捉列表有以下几种形式：

- [var]表示值传递方式捕捉变量var。
- [=]表示值传递方式捕捉所有父作用域的变量（包括this）。
- [&var]表示引用传递捕捉变量var。
- [&]表示引用传递方式捕捉所有父作用域的变量（包括this）。
- [this]表示值传递方式捕捉当前的this指针。

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
```

```
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //按钮单击不带参数
    connect(ui->pushButton, &QPushButton::clicked, [] {
        qDebug() << "hello lambda";
    });

    //按钮单击带参数
    connect(ui->pushButton, &QPushButton::clicked, [] (bool isCheck) {
        qDebug() << "hello lambda" << isCheck;
    });

    //自定义信号带参数
    connect(this, &MainWindow::sig_test, [] (int i, int j) {
        qDebug() << "hello lambda" << i << j;
    });

    emit sig_test(5, 8);
}
```

184. 由于Qt版本众多，有时候为了兼容多个版本甚至跨度Qt4/Qt5/Qt6的兼容，有些头文件或者类名等变了或者新增了，需要用到Qt版本的判断。需要注意的是如果在头文件中使用 QT_VERSION_CHECK 需要先引入#include "qglobal.h"不然编译失败，因为 QT_VERSION_CHECK 这个函数在 qglobal.h 头文件中。

```
//至少要包含 qglobal.h，理论上Qt所有的类都包含了这个头文件，所以你引入Qt的其他头文件也行比如 qobject.h
#include "qglobal.h"
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
#include "qscreen.h"
#else
#include "qdesktopwidget.h"
#endif
```

185. 在使用QString转换到char *或者const char *的时候，务必记得分两步来完成，血的教训，在一个场景中，就因为没有分两步走，现象是msvc的debug异常release正常，mingw和gcc的debug和release都正常，这就很无语了，找问题找半天，对比法排除法按道理要么都有问题才对。

- 转换前QString的内容无关中文还是英文，要出问题都一样。
- 转换中QByteArray无关具体类型，toUtf8、toLatin1、toLocal8Bit、toStdString等方法，要出问题都一样。
- 转换后无关char *还是const char *，要出问题都一样。
- 出问题的随机性的，概率出现，理论上debug的概率更大。
- 根据酷码大佬分析可能的原因(不确定)是msvc为了方便调试，debug会在内存释放后做填充，release则不会。

```
QString text = "xxxxx";
//下面这样转换很可能会有问题
char *data = text.toUtf8().data();
//分两步转换肯定不会有问题
QByteArray buffer = text.toUtf8();
char *data = buffer.data();
const char *data = buffer.constData();
```

186. 关于是使用QList还是QVector的问题，一直是众多Qter的选择问题，主要是这两个玩意提供的的接口函数基本一致，比如插入、删除、取值等。

- 大多数情况下可以用QList。像append、prepend、insert这种操作，通常QList比QVector快的多。
- QList是基于index标签存储它的元素项在内存中，比那种依赖iterator迭代的更快捷，而且你的代码也更少。
- 如果你需要一个真正的连接着的list，且需要保证一个固定插入耗时。那就用迭代器，而不是标签。使用QLinkedList()。
- 如果你需要开辟连续的内存空间存储，或者你的元素远比一个指针大，这时你需要避免个别插入操作，出现堆栈溢出，这时候用QVector。
- 如果更在意取值的速度则用QVector，QCustomPlot用的就是QVector，需要频繁大量的取出数据进行绘制。
- 如果更在意更新数据（添加、删除等）的速度则用QList（对应操作是[]=值），但是因为QChart主要用的是QList访问数据（对应操作是at()），也是导致大数据量卡顿的原因之一，一直被诟病。
- 曲线图表这类的基本上绝大部分时间都是在访问数据，拿到设置好的数据进行绘制。
- 总之：QList更新（插入、追加等）数据速度快，QVector取数据速度快。
- 在数据量很小的情况下两者几乎没啥性能区别。
- 貌似Qt6对这两个类合并了（选择困难症的Qter解放了），QVector=QList即QVector是QList的别名，可能底层改了代码以便发挥两者的优势。

187. 关于mouseTracking鼠标追踪和tabletTracking平板追踪的几点官方说明。

- mouseTracking属性用于保存是否启用鼠标跟踪，缺省情况是不启用的。
- 没启用的情况下，对应部件只接收在鼠标移动同时至少一个鼠标按键按下时的鼠标移动事件。
- 启用鼠标跟踪的情况下，任何鼠标移动事件部件都会接收。
- 部件方法hasMouseTracking()用于返回当前是否启用鼠标跟踪。
- setMouseTracking(bool enable)用于设置是否启用鼠标跟踪。
- 与鼠标跟踪相关的函数主要是mouseMoveEvent()。
- tabletTracking属性保存是否启用部件的平板跟踪，缺省是不起用的。
- 没有启用平板跟踪的情况下，部件仅接收触控笔与平板接触或至少有个触控笔按键按下时的触控笔移动事件。
- 如果部件启用了平板跟踪功能，部件能接收触控笔靠近但未真正接触平板时的触控笔移动事件。
- 这可以用于监视操作位置以及部件的辅助操作功能（如旋转和倾斜），并为图形界面提供这些操作的信息接口。
- 部件方法hasTabletTracking()用于返回当前是否启用平板跟踪。
- setTabletTracking(bool enable)用于设置是否启用平板跟踪。
- 与平板跟踪相关的函数主要是tabletEvent()。

188. 关于QTableWidget等控件调用自带的removeRow、clearContents、clear函数删除了里面的item和内容，会自动调用item或者cellwidget的析构函数进行资源释放，不用自己手动再去释放。

```
//每次调用 clearContents 都会自动清理之前的item
ui->tableWidget->clearContents();
for (int i = 0; i < count; ++i) {
    ui->tableWidget->setItem(i, 0, new QTableWidgetItem("aaa"));
    ui->tableWidget->setItem(i, 1, new QTableWidgetItem("bbb"));
    ui->tableWidget->setCellWidget(i, 2, new QPushButton("ccc"));
}
```

189. 对于QListView（QListWidget）、QTreeView（QTreeWidget）、QTableView（QTableWidget）这种类型的控件，可以通过setChecked来让对应的item产生复选框效果，很多人（包括曾经的自己）误以为这就是复选框控件，其实不是的，他是对应控件的indicator指示器，所以想要更换样式，不能说设置了QCheckBox的样式就有效果，而要单独对齐indicator指示器设置样式才行。

```
QCheckBox::indicator,QGroupBox::indicator,QTreeWidget::indicator,QListWidget::indicator{
width:13px;
height:13px;
}

QCheckBox::indicator:unchecked,QGroupBox::indicator:unchecked,QTreeWidget::indicator:unchecked,QListWidget::indicator:unchecked{
image:url(:/qss/flatwhite/checkbox_unchecked.png);
}

QCheckBox::indicator:unchecked:disabled,QGroupBox::indicator:unchecked:disabled,QTreeWidget::indicator:unchecked:disabled,QListWidget::indicator:disabled{
image:url(:/qss/flatwhite/checkbox_unchecked_disable.png);
}

QCheckBox::indicator:checked,QGroupBox::indicator:checked,QTreeWidget::indicator:checked,QListWidget::indicator:checked{
image:url(:/qss/flatwhite/checkbox_checked.png);
}

QCheckBox::indicator:checked:disabled,QGroupBox::indicator:checked:disabled,QTreeWidget::indicator:checked:disabled,QListWidget::indicator:checked:disabled{
image:url(:/qss/flatwhite/checkbox_checked_disable.png);
}

QCheckBox::indicator:indeterminate,QGroupBox::indicator:indeterminate,QTreeWidget::indicator:indeterminate,QListWidget::indicator:indeterminate{
image:url(:/qss/flatwhite/checkbox_parcial.png);
}

QCheckBox::indicator:indeterminate:disabled,QGroupBox::indicator:indeterminate:disabled,QTreeWidget::indicator:indeterminate:disabled,QListWidget::indicator:indeterminate:disabled{
image:url(:/qss/flatwhite/checkbox_parcial_disable.png);
}
```

190. 关于QTableView（采用model数据源）、QTableWidget列名列宽设置，有时候发现没有起作用，原来是对代码设置的顺序有要求，比如setColumnWidth前必须先setColumnCount，不然列数都没有，哪来的列宽，包括setHorizontalHeaderLabels设置列标题集合也是，前提都要先有列。

```
void frmSimple::initForm()
{
    //实例化数据模型
    model = new QStandardItemModel(this);

    //设置行数列数
    row = 100;
    column = 10;
    //设置列名列宽
    for (int i = 0; i < column; ++i) {
        columnNames << QString("列%1").arg(i + 1);
        columnWidths << 60;
    }
}

void frmSimple::on_btnLoad1_clicked()
{
    //先设置数据模型,否则 setColumnWidth 不起作用
    ui->tableView->setModel(model);

    //设置列数及列标题和列宽
    model->setColumnCount(column);
    //简便方法设置列标题集合
    model->setHorizontalHeaderLabels(columnNames);
    for (int i = 0; i < column; ++i) {
        ui->tableView->setColumnWidth(i, columnWidths.at(i));
    }

    //循环添加行数据
    QDateTime now = QDateTime::currentDateTime();
    model->setRowCount(row);
    for (int i = 0; i < row; ++i) {
        for (int j = 0; j < column; ++j) {
            QStandardItem *item = new QStandardItem;
            //最后一列显示时间区别开来
            if (j == column - 1) {
                item->setText(now.addSecs(i).toString("yyyy-MM-dd
HH:mm:ss"));
            } else {
                item->setText(QString("%1_%2").arg(i + 1).arg(j + 1));
            }
            model->setItem(i, j, item);
        }
    }
}

void frmSimple::on_btnLoad2_clicked()
{
    //设置列标题和列数及列宽
    ui->tableWidget->setColumnCount(column);
    //简便方法设置列标题集合
```

```
    ui->tableWidget->setHorizontalHeaderLabels(columnNames);
    for (int i = 0; i < column; ++i) {
        ui->tableWidget->setColumnWidth(i, columnWidths.at(i));
    }

    //添加数据
    QDateTime now = QDateTime::currentDateTime();
    ui->tableWidget->setRowCount(row);
    for (int i = 0; i < row; ++i) {
        for (int j = 0; j < column; ++j) {
            QTableWidgetItem *item = new QTableWidgetItem;
            //最后一列显示时间区别开来
            if (j == column - 1) {
                item->setText(now.addSecs(i).toString("yyyy-MM-dd
HH:mm:ss"));
            } else {
                item->setText(QString("%1_%2").arg(i + 1).arg(j + 1));
            }
            ui->tableWidget->setItem(i, j, item);
        }
    }
}
```

## 20：191-200

191. 关于QList队列的处理中，我们最常用的就是调用append函数添加item，往前插入item很多人第一印象就是调用insert(0,xxx)来插入，其实QList完全提供了往前追加item的函数prepend、push_front。

```
QStringList list;
list << "aaa" << "bbb" << "ccc";

//往后追加 等价于 append
list.push_back("ddd");
//往前追加 等价于 prepend
list.push_front("xxx");

//往后追加
list.append("ddd");
//往前追加
list.prepend("xxx");

//指定第一个位置插入 等价于 prepend
list.insert(0, "xxx");

//输出 QList("xxx", "aaa", "bbb", "ccc", "ddd")
qDebug() << list;
```

192. Qt内置了一些QList、QMap、QHash相关的类型，可以直接用，不用自己写个长长的类型。

```
//qwindowdefs.h
typedef QList<QWidget *> QWidgetList;
typedef QList<QWindow *> QWindowList;
typedef QHash<WId, QWidget *> QWidgetMapper;
typedef QSet<QWidget *> QWidgetSet;

//qmetatype.h
typedef QList<QVariant> QVariantList;
typedef QMap<QString, QVariant> QVariantMap;
typedef QHash<QString, QVariant> QVariantHash;
typedef QList<QByteArray> QByteArrayList;
```

193. Qt的布局的边距间隔，如果在没有改动过的情况下，是会根据系统分辨率以及缩放比来决定对应的默认值，是变化的，比如在1080P分辨率是9px，在2K分辨率又变成了11px，所有你会发现你在1080P电脑编译的程序，明明看到的是6px、9px，怎么到2K、4K分辨率下间隔和边距就变得好大，如果要保持无论何种分辨率都一样，你需要手动重新设置这些值，这里有个坑，比如默认是是9，你想其他分辨率也是9，你必须先把9改成其他值比如10，然后再改成9，这样才表示真的改动，你直接9改成9是不会变化的，在属性设计器中右侧有个小箭头恢复值的，也是灰色，只有加深显示，并且出现了恢复默认值箭头，才表示你确实是改过了值。
194. Qt对高分屏以及dpi缩放的支持越来越成熟，在Qt4时代默认的策略就是跟随系统的缩放，从Qt5.6开始提供了 AA_EnableHighDpiScaling 的属性设置开启高分屏，到了5.14以后还可以指定缩放的策略 HighDpiScaleFactorRoundingPolicy 比如支持浮点数的缩放比而不是之前的整数倍，从Qt6开始默认永远开启了 AA_EnableHighDpiScaling 属性，没法取消。很多时候我们需要两种模式，一种就是永远不应用高分屏及缩放，一种就是自动应用高分屏及缩放。

```
//永远不应用高分屏及缩放
int main(int argc, char *argv[])
{
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
    QApplication::setAttribute(Qt::AA_Use96Dpi);
#endif
#if (QT_VERSION >= QT_VERSION_CHECK(5,14,0))

 QApplication::setHighDpiScaleFactorRoundingPolicy(Qt::HighDpiScaleFactorRou
ndingPolicy::Floor);
#endif

    QApplication a(argc, argv);
    ....
    return a.exec();
}

//自动应用高分屏及缩放
//方法很多，综合对比下来还是采用配置文件指定缩放策略最适中。
//新建qt.conf文件放到可执行文件同一目录
[Platforms]
WindowsArguments = dpiawareness=0

//有时候想让用户去选择何种策略，需要开启高分屏的之后只需要将qt.conf文件放到可执行文件同一目录即可，就算代码中设置了不应用高分屏及缩放，也无效，也是优先取qt.conf文件的策略。
```

195. 关于QSS要注意的坑。

- qss源自css，相当于css的一个子集，主要支持的是css2标准，很多网上的css3的标准的写法在qss这里是不生效的，所以不要大惊小怪。
- qss也不是完全支持所有的css2，比如text-align官方文档就有说明，只支持 QPushButton and QProgressBar，务必看清楚。
- 有时候偷懒直接来一句 *{xxx}，你会发现大部分是应用了，也有小部分或者极个别没有应用，你可能需要在对应的窗体中 this->setStyleSheet() 来设置。
- qss的执行是有优先级的，如果没有指定父对象，则对所有的应用，比如在窗体widget中{color:#ff0000;} 这样会对widget以及widget的所有子对象应用该样式，这种问题各大群每周都有人问，你会发现各种奇奇怪怪的异样不正常，怎么办呢，你需要指定类名或者对象名，比如#widget{color:#ff0000;} 这样就只会对widget对象应用该样式，另一种写法QWidget#widget{color:#ff0000;}，只想对窗体本身而不是子控件按钮标签等.QWidget{color:#ff0000;}，具体详细规则参见官方说明。
- qss整体来说还是可以的，解析速度性能在Qt5高版本后期比Qt4好很多，尤其是修复了不少qss中的解析绘制BUG。尽管有这样那样的BUG，怀着包容的心对待它。
- qss官方学习地址1：http://47.100.39.100/qtwidgets/stylesheet-reference.html
- qss官方学习地址2：http://47.100.39.100/qtwidgets/stylesheet-examples.html

196. 关于Qt延时的几种方法。

```
void QUIHelperCore::sleep(int msec)
{
    if (msec <= 0) {
        return;
    }

#if 1
    //非阻塞方式延时,现在很多人推荐的方法
    QEventLoop loop;
    QTimer::singleShot(msec, &loop, SLOT(quit()));
    loop.exec();
#else
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
    //阻塞方式延时,如果在主线程会卡住主界面
    QThread::msleep(msec);
#else
    //非阻塞方式延时,不会卡住主界面,据说可能有问题
    QTime endTime = QTime::currentTime().addMSecs(msec);
    while (QTime::currentTime() < endTime) {
        QCoreApplication::processEvents(QEventLoop::AllEvents, 100);
    }
#endif
#endif
}
```

197. 随着国产化的兴起，各种国产系统和国产数据库等逐渐进入开发者的世界，罗列几个要点。

- 中标麒麟neokylin基于centos。
- 银河麒麟kylin早期版本比如V2基于freebsd，新版本V4、V10基于ubuntu。
- 优麒麟ubuntukylin就是ubuntu的汉化版本，加了点农历控件啥的。
- deepin基于debian。
- uos基于deepin或者说是deepin的商业分支。
- ubuntu基于debian。

- linux界主要分两种发行版本，debian（ubuntu、deepin、uos、银河麒麟kylin等）和redhat（fedora、centos、中标麒麟neokylin、中兴新支点newstart等），分别对应apt-get和yum安装命令。绝大部分的linux系统都基于或者衍生自这两种发行版本。
- 理论上基于同一种系统内核的，在其上编译的程序可以换到另外的系统运行，前提是编译器版本一致，比如都是gcc4.9，在ubuntu14.04 64位用gcc4.9编译的Qt程序，是能够在uos 64位上运行的。
- 高版本编译器的系统一般能够兼容低版本的，比如你用gcc4.9编译的程序是能够在gcc7.0上运行，反过来不行。
- 意味着如果你想尽可能兼容更多的系统，尽量用低版本的编译器编译你的程序，当然要你的程序代码语法支持，比如c++11就要从gcc4.7开始才支持，如果你的代码用了c++11则必须至少选择gcc4.7版本及以上。
- 用Qt编写linux程序为了发布后的可执行文件可以兼容各种linux系统，只要在这两种内核（debian、redhat）的系统上用低版本的编译器比如gcc4.7编译qt程序发布即可。
- 2022-01-27补充：根据Qt官方安装包（关于对话框），发现基于redhat和gcc4.9（后面Qt5.14/5.15带的qtc采用gcc5.3）编译器发布的，通用各种linux系统（亲测ubuntu各个版本、fedora、centos、deepin、uos、银河麒麟kylin、中标麒麟neokylin、中兴新支点newstart等），自己按照这个版本也亲测打包发布了，亲测可用，我擦，redhat系统的也可以在debian系统跑。
- 2022-02-10补充：debian上静态编译的程序也可以在redhat系统跑，可能静态编译去掉了很多依赖吧。
- 2022-03-01补充：低调大佬补充，如果没有特定的依赖关系，高版本的编译器编译的程序也可以在低版本编译器的系统运行，比如alpine Linux下用gcc11/clang13编译生成的可执行二进制，依然可以在cenos5/ubuntu10上运行。并不是编译器版本的问题，也不是C++11特性的问题，这个问题涉及到太多，内核版本、gnu libc、ABI兼容等等，两句话说不清。
- 按照QtCreator软件采用的编译器环境规则，一般来说就是低版本的可以在高版本运行，比如Qt5可以在ubuntu14/16/18/20运行，但是高版本编译器编译的就无法在低版本编译器系统运行，会提示缺少GLBC、LIBCXX、symbol xxxxxx等，比如Qt6可以在ubuntu20运行而无法在ubuntu18/16/14等运行。
- 在uos上做开发，建议采用系统自带的Qt库环境开发，以及命令行安装开发环境，不建议使用Qt官方的安装包搭建环境，因为uos的Qt是魔改过的，用Qt官方的标准安装包的环境开发出来的程序，打包发布很可能会有依赖问题而无法运行，而用系统自带的就不存在这个问题。
- 国产人大金仓数据库用的是postgresql数据库改的，意味着你在Qt中用postgresql数据库插件也是能够连接到人大金仓数据库的。
- 以上未必完全正确，欢迎各位指正。

198. 纵观Qt的发展历史，也几乎经历着合久必分、分久必合的逻辑，比如最开始QPushButton等UI控件类都是在QtGui模块中，后面越发臃肿不方便管理和升级迭代，又分离出一个QtWidgets模块；到Qt6又将QList和QVector合并了成了一个类，搞得像分久必合；而且一些数学函数以及封装的c++标准函数库的方法，逐渐放弃了Qt自己的封装改用c++标准函数库，从开始的分到现在的合统一。
199. Qt一直在持续升级迭代，尽管新增加的代码质量明显不如诺基亚时代，但最起码有行动，慢慢完善。目前主要的升级改善在qml模块，底层也有完善，毕竟无论是widget还是qml都是公用一套底层逻辑类，底层基础一定要扎实稳固，个人这几年一直对比测试过不同Qt版本（从旧版本到新版本）很多类和函数的性能，发现官网列出来的新版本对应类和方法的性能提升改善，确实没有说谎，至于提升了多少这块有没有吹牛逼那就不清楚。

- base64算法性能提升很大。
- QStringList等凡是使用了QList相关的类，性能提升巨大。
- 对比测试大概从5.12版本开始QStringList和QMap性能相当。
- 早期版本QStringList如果查找的值先插入则时间越短，QMap则没有这个区别。

```
QStringList list1, list2;
```

```
QMap<QString, QString> map;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    for (int i = 0; i < 100000; ++i) {
        QString s1 = QString("%1").arg(i);
        QString s2 = QString("A%1").arg(i);
        list1 << s1;
        list2 << s2;
        map.insert(s1, s2);
    }
}

void MainWindow::on_pushButton_clicked()
{
    QElapsedTimer time;
    time.start();
    qDebug() << "111" << time.nsecsElapsed() <<
list2.at(list1.indexOf("9999"));
}

void MainWindow::on_pushButton_2_clicked()
{
    QElapsedTimer time;
    time.start();
    qDebug() << "222" << time.nsecsElapsed() << map.value("9999");
}
```

200. 关于QtCreator中三种不同编译版本 debug、release、profile 的区别。

- debug调试模式，编译后的可执行文件很大，带了很多调试符号信息等，方便开发阶段调试的时候进入具体的堆栈查看值。会打开所有的断言，运行阶段性能差速度慢，可能会有卡顿感觉。
- release发布模式，编译后的可执行文件很小，不带任何调试符号信息，一般用于打包发布程序。由于经过了各种优化，会关闭所有断言，运行阶段性能最好，如果有卡顿那肯定是你的程序问题。
- profile概述模式，编译后的可执行文件比debug小很多比release大一点，带有部分调试符号信息，在debug和release之间取一个平衡，兼顾性能和调试，性能更优但是又方便调试。
- 使用Qt5.7版本对应三种模式编译的空白窗体程序大小：debug（1319kb）、release（24kb）、profile（90kb）。
- debug链接的库是带d结尾的，release和profile链接的库是不带d结尾的，很多人以为profile链接的是带d结尾的其实是错误的。
- 新的Qt在线安装程序在安装的时候，可以勾选是否安装debug调试库（对应lib目录下一堆带d结尾的文件），以前的版本是默认都安装，现在可选安装以便减少体积。
- 无论是否安装了debug调试库，你都可以选择debug模式生成对应debug的文件，这个不知道怎么做到的。
- 无论是哪种模式，都可以在程序中开启日志钩子输出日志信息，方便收集运行阶段的各种信息反馈给开发人员查看问题。
- 最初的开发工具一般是具有debug和release两种模式，随着用户需求的增加和场景的需要，部分开发工具衍生出了profile模式，更有甚者比如flutter还有第四种test模式。

## 21：201-210

201. 编译生成debug版本动态库，文件末尾自动加上d结尾。

```
CONFIG(debug, debug|release) {
    win32:      TARGET = $$join(TARGET,,,d)
    mac:        TARGET = $$join(TARGET,,,_debug)
    unix:!mac:  TARGET = $$join(TARGET,,,d)
}
```

202. QtCreator中pro项目文件格式说明。

| 名称 | 说明 |
| --- | --- |
| QT += core gui | 添加本项目中需要的模块，影响后面代码文件include的时候自动弹出下拉选择，如果pro文件没有引入该模块则无法自动语法提示，一般打包发布的时候对应动态库文件比如 Qt5Core.dll。 |
| TARGET = xxx | 生成最后目标文件的名字，可以是可执行文件或者库文件。 |
| TEMPLATE = app | 项目程序的生成模式，默认是app表示生成可执行文件程序，如果是动态库项目就是 TEMPLATE = lib。 |
| CONFIG += qaxcontainer | 引入一些配置，在Qt4的时候还用来引入一些模块，其中有部分改成了QT += 方式引入，比如Qt5引入本地activex控件支持改成了QT += qaxcontainer。 |
| DEFINES += xxx | 项目中自定义的一些定义，可以在代码文件中识别，通常用来定义一些不同平台的处理，根据项目需要自己定义任何标识。 |
| HEADERS += head.h | 项目中用到的头文件，一般拓展名是.h，可以写在一行也可以分行写，分行要用 \ 斜杠结束。 |
| SOURCES += main.cpp | 项目中用到的实现文件，一般拓展名是.cpp，可以写在一行也可以分行写，分行要用 \ 斜杠结束。 |
| FORMS += Form.ui | 项目中用到的UI文件，一般拓展名是.ui，可以写在一行也可以分行写，分行要用 \ 斜杠结束。 |
| RESOURCES += main.qrc | 项目中用到的资源文件，可以多个，写代码使用对应资源文件中的文件时候务必记得资源文件中的前缀。 |
| LIBS += -L$$PWD/ -lavformat -lavcodec | 项目中编译时候链接依赖的库，一般是 .lib .a .dylib 文件，可以写在一行，省略文件名的lib打头部分，也可以分多行绝对路径和全名称。 |
| DESTDIR += $$PWD/bin | 目标生成路径，$$PWD表示当前目录，一般建议生成的最终文件重定向到另外目录存放，好找，不然一堆临时文件在里面有时候文件太多好难找。 |
| INCLUDEPATH += $$PWD/include | 工程需要的头文件，指定整个目录，写代码的时候找到的话会自动下拉。 |
| DEPENDPATH += | 工程的依赖路径，用的比较少，一般涉及到引入链接库的时候可能需要。 |
| include($$PWD/3rd.pri) | 引入pri模块文件，pri最大的好处就是分目录管理文件，通用的轮子模块可以放到一个目录下，然后用pri统一管理，可以给多个项目公用。 |

**官方详细地址**[https://doc.qt.io/qt-5/qmake-variable-reference.html](https://doc.qt.io/qt-5/qmake-variable-reference.html)

203. 如果发现之前编译正常，突然之间再编译就一直死循环的样子，停留在一行提示并疯狂不停的打印，或者提示文件时间在未来，这说明你很可能改过开发环境的时间（比如测试某个授权文件失效），导致有修改过文件的保存时间在未来，你只需要将时间调整回来，将最后更新时间不正确的代码文件重新保存下就行。Qt的增量编译是根据文件的最后修改时间来判定的，最后的修改时间比上一次的修改时间还要新则认为该文件被修改过，需要重新编译该文件。

204. Qt的构建套件一般是在安装Qt开发环境的时候自动设置的，当然也可以手动设置，手动设置的时候千万要注意编译器和Qt库必须一致，否则该构建套件是有问题的，千万不能乱设置，尤其是对构建套件命名的时候最好标明qt版本和编译器版本，最好也要一致，不要说名称叫msvc而编译器选择的确是mingw，这样尽管能正常使用该构建套件，但是会造成一种误解，还以为该套件是msvc的，其实里面是mingw的。有个qter说他的qt坏了，死活编译失败，远程一看，尼玛，构建套件名称写的qt_msvc2019 编译器选择的msvc2015（他电脑只安装了vs2015），qt库选择的mingw！差点狂扇自己八个耳光，太离谱了！
205. 当你编译Qt程序发现编译通不过提示报错，而且报错提示在Qt的头文件的时候，不要去尝试着修改Qt头文件来编译通过，那样没用的，你使用的Qt的库是已经根据原始的头文件编译好的。如果报错提示在编译生成的临时的moc等文件，你也不要尝试去修改他，那个是临时文件，这次你改好了也许编译通过了，你重新编一下又覆盖了还是旧的错误。总之你要从源头（你的代码）找问题。
206. 有时候需要对文本进行分散对齐显示，相当于无论文字多少，尽可能占满整个空间平摊占位宽度，但是在对支持对齐方式的控件比如QLabel调用 setAlignment(Qt::AlignJustify | Qt::AlignVCenter) 设置分散对齐会发现没有任何效果，这个时候就要考虑另外的方式比如通过控制字体的间距来实现分散对齐效果。

```
QString text = "测试分散对齐内容";
//计算当前文本在当前字体下占用的宽度
QFont font = ui->label->font();
int textWidth = ui->label->fontMetrics().width(text);
//显示文本的区域宽度=标签的宽度-两边的边距
int width = ui->label->width() - 12;
//需要-1相当于中间有几个间隔
int count = text.count() - 1;
//计算每个间距多少
qreal space = qreal(width - textWidth) / count;
//设置固定间距
font.setLetterSpacing(QFont::AbsoluteSpacing, space);
ui->label->setFont(font);
ui->label->setText(text);
```

207. 随着需求的不断增加，程序不断变大，用到的动态库也越来越多，到了发布程序的时候你会发现和可执行文件同一目录下文件数量真多，此时可能会考虑如何将一些库文件分门别类的存放，这样方便管理。

- Qt提供的设置动态库路径的方法setLibraryPaths是用来搜索插件动态库的，而不是程序直接依赖的动态库。
- 很多人以为这个可以设置Qt的库或者程序中依赖的第三方库的路径，其实想想也知道，因为程序依赖这个库，找不到的话根本跑不起来，程序跑不起来怎么应用执行这个代码呢?
- Qt默认是可用通过setLibraryPaths的方式设置Qt插件的动态库目录位置，比如数据库插件sqldrivers，因为这些库文件是真正在Qt程序跑起来以后通过插件形式去加载的。
- 还可以通过qt.conf文件设置 Plugins="config" 指定所有插件在可执行文件下的config目录下。
- 要想设置程序直接依赖的动态库在其他目录，找遍全宇宙也只有一个办法，那就是设置环境变量，除此别无他法。
- 至于如何设置环境变量方式很多，比如手动在电脑上设置，或者搞个批处理文件执行命令行，在程序安装的时候自动执行，或者程序打包目录下用户手动运行这个批处理。
- 大神补充：设置插件的目录还可以通过在main函数最前面写 qputenv("PATH", QString("%1;%2").arg(qgetenv("PATH"), pluginFileInfo.path()).toLocal8Bit()); 来实现。
- 网友补充：最终找插件的路径其实就是这个 QT_PLUGIN_PATH 环境变量。

208. 进度条控件如果设置的垂直方向，就算你设置了文本可见，会发现根本看不到进度文本，经过多方百折不挠的试探，以及和酷码大佬深入的探讨，发现只要设置下border样式（border:1px solid

#ff0000、border:none、border-style:solid、border-radius:0px 任意一种）就行，就可以把文本显示出来，这TM就不知道Qt为什么总是不统一规则，这个BUG通用于任何版本，这个可能是因为边框的solid样式冲突了导致无法继续绘制，确切的说这必须是BUG，这个锅Qt必须背。

209. 我们在使用QFileDialog::getOpenFileName、QFileDialog::getExistingDirectory等方法时，有时候会发现首次打开很卡，尤其是在默认目录很多文件的时候，此时你可以考虑设置这些函数最末尾的参数为QFileDialog::DontUseNativeDialog，表示不采用本地系统对话框，这样的话会采用Qt的对话框，速度快很多，估计系统的对话框在打开的时候会做很多初始化加载处理。

```
QFileDialog::getOpenFileName(this, "", "", "", 0,
QFileDialog::DontUseNativeDialog);
QFileDialog::getExistingDirectory(this, "", "",
QFileDialog::DontUseNativeDialog);
```

210. 滑块控件QSlider，如果设置的垂直样式，其进度颜色和剩余颜色，刚好和横向样式的颜色相反的，不确定这个是否是Qt的BUG，Qt456都是这个现象。

```
QSlider::groove:horizontal{
height:8px;
background:#FF0000;
}

QSlider::add-page:horizontal{
height:8px;
background:#FF0000;
}

QSlider::sub-page:horizontal{
height:8px;
background:#00FF00;
}

QSlider::handle:horizontal{
width:10px;
background:#0000FF;
}

QSlider::groove:vertical{
width:8px;
background:#FF0000;
}

QSlider::add-page:vertical{
width:8px;
background:#00FF00;
}

QSlider::sub-page:vertical{
width:8px;
background:#FF0000;
}

QSlider::handle:vertical{
height:10px;
background:#0000FF;
```

```
}
```

## 22：211-220

211. QMainWindow 在对停靠窗体进行排列的时候，有些不常用的设置容易遗忘，建议将 QMainWindow 的头文件函数过一遍一目了然。详细介绍各种停靠参数文章参见 https://zhuanlan.zhihu.com/p/388544168。

```
//设置允许各种嵌套比如上下排列左右排列非常灵活
//此设置会和下面的 setDockOptions 中的参数覆盖所以要注意顺序
//this->setDockNestingEnabled(true);

//设置停靠参数,不允许重叠,只允许拖动和嵌套
this->setDockOptions(AnimatedDocks | AllowNestedDocks);

//将底部左侧作为左侧区域，底部右侧作为右侧区域，否则底部区域会填充拉伸
this->setCorner(Qt::BottomLeftCorner, Qt::LeftDockWidgetArea);
this->setCorner(Qt::BottomRightCorner, Qt::RightDockWidgetArea);
```

212. 当我们在对QModelIndex取数据的时候，常规的角色的数据（QVariant类型支持to的比如 toString、toInt、toDouble等）可以很方便的取出来，特定的数据类型需要用的万能取值模板函数 T value() 取出来。

```
//显示文本
QString text = index.data(Qt::DisplayRole).toString();
//文本对齐
int align = index.data(Qt::TextAlignmentRole).toInt();
//文字字体
QFont font = index.data(Qt::FontRole).value<QFont>();
//前景色
QColor color = index.data(Qt::ForegroundRole).value<QColor>();
//背景色
QColor color = index.data(Qt::BackgroundRole).value<QColor>();
```

213. 很多人以为拖曳只要在dropEvent事件就可以了，其实不行的，没有效果的，需要先在 dragEnterEvent事件中执行event->accept()才行，不然根本没有效果，很多人尤其是初学者都挂在这里，我就是在这里摔了一跤，好疼！

```
void frmMain::dropEvent(QDropEvent *event)
{
    QList<QUrl> urls = event->mimeData()->urls();
}

void frmMain::dragEnterEvent(QDragEnterEvent *event)
{
    if(event->mimeData()->hasFormat("application/x-
qabstractitemmodeldatalist")) {
        event->setDropAction(Qt::MoveAction);
        event->accept();
    } else {
        event->ignore();
    }
}
```

214. Qt5.6以后内置的是webengine浏览器内核，如果需要做web交互的话必须用到 qwebchannel.js 这个文件，此文件是Qt官方提供的，所以不建议去改动其中的源码，要注意的是，由于官方对webengine的支持在不断更新，所以官方提供的对应Qt版本的 qwebchannel.js 文件也不同，意味着你要用对应提供的版本的 qwebchannel.js 文件才ok，该文件默认在 C:\Qt\Qt5.12.11\Examples\Qt-5.12.11\webchannel\shared 目录下。经过几十个Qt版本的测试发现，用高版本的 qwebchannel.js 放到低版本运行不行，低版本放到高版本可以，为了万无一失还是建议直接用对应版本的。
215. 对于QString去除空格，有多种场景，可能需要去除左侧、右侧、所有等位置的空格。

```
//字符串去空格 -1=移除左侧空格 0=移除所有空格 1=移除右侧空格 2=移除首尾空格 3=首尾清除中间留一个空格
QString QUIHelperData::trimmed(const QString &text, int type)
{
    QString temp = text;
    QString pattern;
    if (type == -1) {
        pattern = "^ +\\s*";
    } else if (type == 0) {
        pattern = "\\s";
        //temp.replace(" ", "");
    } else if (type == 1) {
        pattern = "\\s* +$";
    } else if (type == 2) {
        temp = temp.trimmed();
    } else if (type == 3) {
        temp = temp.simplified();
    }

    //调用正则表达式移除空格
    if (!pattern.isEmpty()) {
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
        temp.remove(QRegularExpression(pattern));
#else
        temp.remove(QRegExp(pattern));
#endif
    }

    return temp;
}

//测试代码
QString text = "  a  b  c d  ";
//结果：a  b  c d
QUIHelper::trimmed(text, -1);
//结果：abcd
QUIHelper::trimmed(text, 0);
//结果：  a  b  c d
QUIHelper::trimmed(text, 1);
//结果：a  b  c d
QUIHelper::trimmed(text, 2);
//结果：a b c d
QUIHelper::trimmed(text, 3);
```

216. Qt的网络库支持udp广播搜索和组播搜索，其中组播搜索可以跨网段搜索，有时候你会发现失灵，此时你可以尝试把本地的虚拟机的网卡禁用试试，估计就好了。还有就是在本地开启了代理啥的，先关掉试试。近期在使用tcpsocket连接的时候，发现在Qt4和Qt5中正常的程序，到了Qt6中就不行了，报错提示 The proxy type is invalid for this operation ，原来是本地设置了代理导致的，可能在Qt6以前会默认跳过去不处理。

```
//也可以通过代码设置跳过代理
#include <QNetworkProxy>
QNetworkProxyFactory::setUseSystemConfiguration(false);
//下面这样每次设置也可以
tcpSocket->setProxy(QNetworkProxy::NoProxy);

//查阅到文章 https://www.cnblogs.com/cppskill/p/11730452.html
//从5.8开始socket默认代理类型是DefaultProxy而不是NoProxy，不知道出于什么考虑。
```

217. 关于交叉编译，对于初学者来说是个极难跨过去的砍（一旦跨过去了，以后遇到需要交叉编译的时候都是顺水推舟、信手拈来。），因为需要搭建交叉编译环境，好在现在厂家提供的板子基本上都是测试好的环境，尤其是提供的编译器，不用自己再去折腾，按照官方手册来基本上不会有啥的的问题。

- 在linux系统上编译ffmpeg和qt都是非常简单的事情，初学者也会，前提只要本地的gcc g++编译器正常。
- 任何编译器包括嵌入式编译器，为了确保环境正常，你可以先查看对应的编译器版本是否ok，g++ -v  arm-linux-g++ -v。
- 交叉编译器查看版本 /opt/FriendlyARM/toolschain/4.5.1/bin/arm-linux-g++ -v 。
- 编译器位数和操作系统位数有关，一般32位的编译器要在32位的系统上做交叉编译，虽然32位也可以在安装依赖后，在64位系统做交叉编译，但是个人不建议，可能会出问题。64位的编译器只能在64位的系统。
- 设置了环境变量则可以省略掉长长的路径，直接打可执行文件名称即可，没有设置环境变量则需要打完整路径。
- 设置环境变量只是为了编译的时候让自动寻找编译器，其实也完全可以不用设置环境变量，使用绝对路径指定编译器位置即可。
- 在linux上编译，无论是ffmpeg还是qt还是其他，都是通用的步骤，第一步：./configure  第二步：make  第三步：make install 。
- 至于具体configure后面有哪些参数，参照对应源码包的手册就行，搜索也一大堆。当然你用默认的就不带任何参数一般也可以，自动采用默认参数进行编译。
- 交叉编译ffmpeg命令：./configure --prefix=host --enable-static --disable-shared --disable-doc --cross-prefix=/opt/FriendlyARM/toolschain/4.5.1/bin/arm-linux- --arch=arm --target-os=linux
- 交叉编译qt前提：修改mkspecs/qws/linux-arm-g++下面的qmake.conf，如果没有设置环境变量则设置对应编译器的绝对路径，并将编译器的名字改成你需要的。
- 比如修改gcc编译器：QMAKE_CC = /opt/FriendlyARM/toolschain/4.5.1/bin/arm-linux-gcc
- 交叉编译qt4.8.5命令：./configure -prefix host -embedded arm -xplatform qws/linux-arm-g++ -release -opensource -confirm-license -qt-sql-sqlite -qt-gfx-linuxfb -plugin-sql-sqlit -no-qt3support -no-phonon -no-svg -no-webkit -no-javascript-jit -no-script -no-scripttools -no-declarative -no-declarative-debug -qt-zlib -no-gif -qt-libtiff -qt-libpng -no-libmng -qt-libjpeg -no-rpath -no-pch -no-3dnow -no-avx -no-neon -no-openssl -no-nis -no-cups -no-dbus -little-endian -qt-freetype -no-opengl -no-glib -nomake demos -nomake examples -nomake docs -nomake tools
- 交叉编译qt5.9.8命令：./configure -prefix host -xplatform linux-arm-g++ -recheck-all -opensource -confirm-license -optimized-qmake -release -no-separate-debug-info -strip -

shared -static -c++std c++1z -no-sse2 -pch -compile-examples -gui -widgets -no-dbus -no-openssl -no-cups -no-opengl -linuxfb -qt-zlib -qt-libpng -qt-libjpeg -qt-freetype

- 综上所述交叉编译和常规的编译就一个区别，需要手动指定交叉编译器路径。ffmpeg是通过--cross-prefix=指定，qt比较庞大是通过更改配置文件最后通过-xplatform指定配置文件名称。
- Qt6的编译比较繁琐，默认用cmake编译，在linux上先用cmake3.19以上版本的源码，用make编译生成cmake，然后再用cmake编译qt生成qmake，最后调用qmake来编译你的qt项目。
- 编译Qt其实只是想用其中的库，至于demo、doc、tool、example等统统不用，费时费力。所以强烈建议编译的时候去掉，大大加快编译速度。
- 编译建议用普通用户编译即可，包括解压源码，因为这样编译出来的库普通用户就能用，如果是root管理员编译的则以后都需要管理员权限才行。
- 很多系统都提供了直接鼠标右键解压，其实也是可以的，就是速度慢，建议用命令行解压和删除目录。
- Qt的编译参数每个版本都可能有出入，毕竟一直在更新代码，甚至有些分类描述变了，比如之前-qt-xcb到了5.15改成了-xcb，之前-qt-sql-sqlite改成了-qt-sqlite，一定要看源码下的readme，里面约定了编译环境要求的最低版本，后面qt5开始具体的配置参数有哪些放到了qtbase目录下的config说明。
- 编译完成使用如果遇到提示 GL/gl.h 错误，需要安装 apt install libgl1-mesa-dev libglu1-mesa-dev 或者 yum install mesa-libGL-devel mesa-libGLU-devel 。
- 编译参数说明可参考 https://blog.csdn.net/xi_gua_gua/article/details/53413930。

218. 在Qt中设置图片有时候会发现不成功，很可能是因为文件的拓展名不正确导致的，比如jpg的图片拓展名是png，bmp的图片拓展名改成了jpg，QImage、QPixmap传入文件路径加载图片，是通过拓展名去调用对应的图片解析算法，比较傻，但是速度快，不用经过分析具体内部是何种图片格式。如果想要不管拓展名都能保证加载成功，则必须读取图片文件数据加载的方式处理。

```
//可以是资源文件中的图片也可以是本地文件
QString fileName = ":/test.png";

//此方式按照拓展名来区分具体格式不准确
//如果拓展名不正确就无法加载成功
ui->label->setPixmap(QPixmap(fileName));

//通过直接读取图片数据加载保证成功
QFile file(fileName);
file.open(QIODevice::ReadOnly);
QByteArray data = file.readAll();

//通过 QImage 处理
QImage img;
img.loadFromData(data);
//下面这种方式也行
//QImage img = QImage::fromData(data);
ui->label->setPixmap(QPixmap::fromImage(img));

//通过 QPixmap 处理
QPixmap pix;
pix.loadFromData(data);
ui->label->setPixmap(pix);
```

219. 总结几个Qt版本的冷知识。

- Qt4.8.7是Qt4的终结版本，是Qt4系列版本中最稳定最经典的（很多嵌入式板子还是用Qt4.8），其实该版本是和Qt5.5差不多时间发布的。参考链接 https://www.qt.io/blog/2015/05/26/qt-4-8-7-

released https://blog.qt.io/blog/2015/07/01/qt-5-5-released/

- Qt5.6.3最最后支持xp系统的长期支持版本，Qt5.7.0是最后支持xp系统的非长期支持版本（有可能有极少数函数不支持，个人没遇到过）。
- Qt5.12.3是最后提供mysql数据库插件的版本，往后的版本需要自行编译对应的mysql数据库插件，官方安装包不再提供。
- Qt5.12.5是最后样式表性能最高的版本，经过酷码大佬查阅代码发现此后版本的样式表源码中为了修复一个bug做了循环嵌套设置，导致性能急剧下降，界面越多性能暴降10倍以上。
- Qt5.14.2是最后提供二进制安装包的版本，后面的版本都需要在线安装。
- Qt5.15系列是最后支持win7的版本，后面的Qt6系列版本需要更改源码编译才能支持win7，这对于小白来说难于上青天。
- Qt6.0/6.1版本其实也是支持win7的，但是因为缺失太多模块，而且BUG成山，大佬说了狗都不用，所以使用此版本没意义。
- Qt6不支持win7，是说开发阶段和运行阶段都不支持，无论开发阶段还是运行阶段你都需要Qt的库，只要是Qt的库不支持，到哪里也不支持。
- 新版的qtc7由于采用Qt6编译，所以也只能在win10及以上运行，意味着你要用新的qtc7+Qt5做开发也必须用win10及以上。
- 欢迎各位补充，比如哪个版本以后商用需要收费之类的，貌似用Qt4，在不更改Qt本身源码，动态库发布程序，法律风险小一些?

220. Qt官方除了Qt库一直在升级外，对应的集成开发环境也在更新升级，一般会选用最新的Qt库编译新版本，要注意的是，有些人安装的旧版本的qtc，加载比较高版本的Qt库，很容易出现报错提示 Project ERROR: Cannot run compiler 'g++'. Maybe you forgot to setup the environment? 之类的，一般是版本跨度过大，比如用Qt5.5附带的qtc加载Qt5.9的库，导致有些环境识别不到，可能是qtc在新版本中对某些识别处理规则有变动。所以一般建议可以用新的qtc加载旧的Qt库，不建议旧的qtc加载新的Qt库。

# 2 升级到Qt6

## 00：直观总结

1. 增加了很多轮子，同时原有模块拆分的也更细致，估计为了方便拓展个管理。
2. 把一些过度封装的东西移除了（比如同样的功能有多个函数），保证了只有一个函数执行该功能。
3. 把一些Qt5中兼容Qt4的方法废弃了，必须用Qt5中对应的新的函数。
4. 跟随时代脚步，增加了不少新特性以满足日益增长的客户需求。
5. 对某些模块和类型及处理进行了革命性的重写，运行效率提高不少。
6. 有参数类型的变化，比如 long * 到 qintptr * 等，更加适应后续的拓展以及同时对32 64位不同系统的兼容。
7. 源码中的double数据类型全部换成了qreal，和Qt内部数据类型高度一致和统一。
8. 我测试的都是QWidget部分，quick部分没有测试，估计quick部分更新可能会更多。
9. 强烈建议暂时不要用Qt6.0到Qt6.2之间的版本，一些模块还缺失，相对来说BUG也比较多，推荐6.2.2版本开始正式迁移。

## 01：01-10

1. 万能方法：安装5.15版本，定位到报错的函数，切换到源码头文件，可以看到对应提示字样 QT_DEPRECATED_X("Use sizeInBytes") 和新函数。按照这个提示类修改就没错，一些函数是从Qt5.7 5.9 5.10等版本新增加的，可能你的项目还用的Qt4的方法，但是Qt6以前都兼容这些旧方法，到了Qt6就彻底需要用新方法了。**PS：如果本身就是Qt6新增的功能函数则此方法无效**
2. Qt6对core这个核心类进行了拆分，多出来core5compat，因此你需要在pro增加对应的模块已经代码中引入对应的头文件。

```
//pro文件引入模块
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
greaterThan(QT_MAJOR_VERSION, 5): QT += core5compat

//代码中引入头文件
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
#include <QtWidgets>
#endif
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
#include <QtCore5Compat>
#endif
```

3. 默认Qt6开启了高分屏支持，界面会变得很大，甚至字体发虚，很多人会不习惯，因为这种模式如果程序很多坐标计算没有采用devicePixelRatio进行运算的话，100%会出现奇奇怪怪的问题，因为坐标不准确了。要取消这种效果可以设置高分屏缩放因子。

```
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))

 QGuiApplication::setHighDpiScaleFactorRoundingPolicy(Qt::HighDpiScaleFactorRoundingPolicy::Floor);
#endif
```

4. 原有的随机数函数提示用QRandomGenerator替代，为了兼容所有qt版本，改动最小的办法是直接用c++中的随机数，比如qsrand函数换成srand，qrand函数换成rand，查看过源代码，其实封装的就是c++中的随机数，很多类似的封装比如qSin封装的sin。
5. QColor的 light 改成 lighter ， dark 改成 darker，其实 lighter、darker 这两个方法以前一直有。
6. QFontMetricsF 中的 fm.width 换成 fm.horizontalAdvance ，从5.11开始用新函数。
7. QPalette调色板枚举值，Foreground = WindowText, Background = Window，其中 Foreground 和 Background 没有了，要用 WindowText 和 Window 替代，以前就有。类似的还有 setTextColor 改成了 setForeground 。
8. QWheelEvent的 delta() 改成 angleDelta().y()，pos() 改成 position() 。
9. svg模块拆分出来了svgwidgets，如果用到了该模块则需要在pro增加 QT += svgwidgets ，同理opengl模块拆分出来了openglwidgets。
10. qlayout中的 margin() 函数换成 contentsMargins().left()，查看源码得知以前的 margin() 返回的就是 contentsMargins().left()，在四个数值一样的时候，默认四个数值就是一样。类似的还有 setMargin移除了，统统用setContentsMargins。

## 02：11-20

11. 之前 QChar c = 0xf105 全部要改成强制转换 QChar c = (QChar)0xf105，不再有隐式转换，不然编译报错提示error: conversion from 'int' to 'QChar' is ambiguous 。
12. qSort等一些函数用回c++的 std::sort 。

```
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
    std::sort(ipv4s.begin(), ipv4s.end());
#else
    qSort(ipv4s);
#endif
```

13. Qt::WA_NoBackground 改成 Qt::WA_OpaquePaintEvent 。

14. QMatrix 类废弃了没有了，换成 QTransform ， 函数功能基本一致，QTransform 类在Qt4就一直有。
15. QTime 计时去掉了，需要改成 QElapsedTimer ， QElapsedTimer 类在Qt4就一直有。
16. QApplication::desktop()废弃了， 换成了 QApplication::primaryScreen()。

```
#if (QT_VERSION > QT_VERSION_CHECK(5,0,0))
#include "qscreen.h"
#define deskGeometry qApp->primaryScreen()->geometry()
#define deskGeometry2 qApp->primaryScreen()->availableGeometry()
#else
#include "qdesktopwidget.h"
#define deskGeometry qApp->desktop()->geometry()
#define deskGeometry2 qApp->desktop()->availableGeometry()
#endif
```

17. 获取当前屏幕索引以及尺寸需要分别处理。

```
//获取当前屏幕索引
int QUIHelper::getScreenIndex()
{
    //需要对多个屏幕进行处理
    int screenIndex = 0;
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
    int screenCount = qApp->screens().count();
#else
    int screenCount = qApp->desktop()->screenCount();
#endif

    if (screenCount > 1) {
        //找到当前鼠标所在屏幕
        QPoint pos = QCursor::pos();
        for (int i = 0; i < screenCount; ++i) {
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
            if (qApp->screens().at(i)->geometry().contains(pos)) {
#else
            if (qApp->desktop()->screenGeometry(i).contains(pos)) {
#endif
                screenIndex = i;
                break;
            }
        }
    }
    return screenIndex;
}

//获取当前屏幕尺寸区域
QRect QUIHelper::getScreenRect(bool available)
{
    QRect rect;
    int screenIndex = QUIHelper::getScreenIndex();
    if (available) {
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
        rect = qApp->screens().at(screenIndex)->availableGeometry();
#else
```

```
        rect = qApp->desktop()->availableGeometry(screenIndex);
#endif
    } else {
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
        rect = qApp->screens().at(screenIndex)->geometry();
#else
        rect = qApp->desktop()->screenGeometry(screenIndex);
#endif
    }
    return rect;
}
```

18. QRegExp类移到了core5compat模块，需要主动引入头文件 #include 。

```
    //设置限制只能输入数字+小数位
    QString pattern = "^-?[0-9]+([.]{1}[0-9]+){0,1}$";
    //设置IP地址校验过滤
    QString pattern = "(2[0-5]{2}|2[0-4][0-9]|1?[0-9]{1,2})";

    //确切的说 QRegularExpression QRegularExpressionValidator 从5.0 5.1开始就有
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
    QRegularExpression regExp(pattern);
    QRegularExpressionValidator *validator = new
QRegularExpressionValidator(regExp, this);
#else
    QRegExp regExp(pattern);
    QRegExpValidator *validator = new QRegExpValidator(regExp, this);
#endif
    lineEdit->setValidator(validator);
```

19. QWheelEvent构造参数和对应的计算方位函数变了。

```
//模拟鼠标滚轮
#if (QT_VERSION < QT_VERSION_CHECK(6,0,0))
QWheelEvent wheelEvent(QPoint(0, 0), -scal, Qt::LeftButton, Qt::NoModifier);
#else
QWheelEvent wheelEvent(QPointF(0, 0), QPointF(0, 0), QPoint(0, 0), QPoint(0,
-scal), Qt::LeftButton, Qt::NoModifier, Qt::ScrollBegin, false);
#endif
QApplication::sendEvent(widget, &wheelEvent);

//鼠标滚轮直接修改值
QWheelEvent *whellEvent = (QWheelEvent *)event;
//滚动的角度,*8就是鼠标滚动的距离
#if (QT_VERSION < QT_VERSION_CHECK(6,0,0))
int degrees = whellEvent->delta() / 8;
#else
int degrees = whellEvent->angleDelta().x() / 8;
#endif
//滚动的步数,*15就是鼠标滚动的角度
int steps = degrees / 15;
```

20. qVariantValue 改成 qvariant_cast ， qVariantSetValue(v, value) 改成了 v.setValue(val)。相当于退回到最原始的方法，查看qVariantValue源码封装的就是qvariant_cast。

## 03：21-30

21. QStyleOption的init改成了initFrom。
22. QVariant::Type 换成了 QMetaType::Type ， 本身以前的 QVariant::Type 封装的就是 QMetaType::Type 。
23. QStyleOptionViewItemV2 V3 V4 之类的全部没有了，暂时可以用 QStyleOptionViewItem 替代。
24. QFont的 resolve 的一个重载函数换成了 resolveMask。
25. QSettings的 setIniCodec 方法移除了，默认就是utf8，不需要设置。
26. qcombobox 的 activated(QString) 和 currentIndexChanged(QString) 信号删除了，用int索引参数的那个，然后自己通过索引获取值。个人觉得这个没必要删除。
27. qtscript模块彻底没有了，尽管从Qt5时代的后期版本就提示为废弃模块，一致坚持到Qt6才正式废弃，各种json数据解析全部换成qjson类解析。
28. QByteArray 的 append indexOf lastIndexOf 等众多方法的QString参数重载函数废弃了，要直接传 QByteArray，就在原来参数基础上加上 .toUtf8() 。查看源码也看得到以前的QString参数也是转成.toUtf8()再去比较。
29. QDateTime的时间转换函数 toTime_t + setTime_t 名字改了，对应改成了 toSecsSinceEpoch + setSecsSinceEpoch ， 这两个方法在Qt5.8时候新增加的。
30. QLabel的 pixmap 函数之前是指针 *pixmap() 现在换成了引用 pixmap()。

## 04：31-40

31. QTableWidget的 sortByColumn 方法移除了默认升序的方法，必须要填入第二个参数表示升序还是降序。
32. qtnetwork模块中（TCP/UDP相关的socket）的错误信号error换成了errorOccurred，就改了个名字，注意websocket那块居然没统一改过来依然是叫error。

```
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
    connect(udpSocket, SIGNAL(errorOccurred(QAbstractSocket::SocketError)), this, SLOT(error()));
    connect(tcpSocket, SIGNAL(errorOccurred(QAbstractSocket::SocketError)), this, SLOT(error()));
#else
    connect(udpSocket, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));
    connect(tcpSocket, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));
#endif

//特别注意websocket中依然还是用error
connect(webSocket, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(error()));
```

33. XmlPatterns模块木有了，全部用xml模块重新解析。
34. nativeEvent的参数类型变了。

```
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
bool nativeEvent(const QByteArray &eventType, void *message, qintptr *result);
#else
bool nativeEvent(const QByteArray &eventType, void *message, long *result);
#endif
```

35. QButtonGroup的buttonClicked信号中int参数的函数全部改名字叫idClicked。

```
    QButtonGroup *btnGroup = new QButtonGroup(this);
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
    connect(btnGroup, SIGNAL(idClicked(int)), ui->xstackWidget,
SLOT(setCurrentIndex(int)));
#else
    connect(btnGroup, SIGNAL(buttonClicked(int)), ui->xstackWidget,
SLOT(setCurrentIndex(int)));
#endif
```

36. QWebEngineSettings之前是QWebEngineSettings::defaultSettings();现在改成了QWebEngineProfile::defaultProfile()->settings();通过查看之前的源码得知QWebEngineSettings::defaultSettings();封装的就是QWebEngineProfile::defaultProfile()->settings();因为Qt6去除了N多过度封装的函数。

```
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
    QWebEngineSettings *webSetting = QWebEngineProfile::defaultProfile()-
>settings();
#else
    QWebEngineSettings *webSetting = QWebEngineSettings::defaultSettings();
#endif
```

37. Qt6将enterEvent的参数QEvent改成了QEnterEvent也不打个招呼。这种改变编译也不会提示的。

```
#if (QT_VERSION >= QT_VERSION_CHECK(6,0,0))
    void enterEvent(QEnterEvent *);
#else
    void enterEvent(QEvent *);
#endif

//后面经过JasonWong大佬的指点，从父类重新实现的virtual修饰的函数，建议都加上override关键字。
//这样的话一旦父类的函数或者参数变了则会提示编译报错，而不是编译通过但是运行不正常会一脸懵逼茫然，从而把锅扣给Qt。

//下面是父类函数
virtual void enterEvent(QEvent *event);
//子类建议加上override
void enterEvent(QEvent *event) override;
```

38. Qt6中多个类进行了合并，比如现在QVector就成了QList的别名，意味着这两个类是同一个类没有任何区别，可能Qt内部对两种的优点都集中在一起，并尽量重写算法或者其他处理规避缺点。同理QStringList现在也成了 QList<QString> 的别名，是同一个类，没有单独的类。
39. 在Qt4时代默认QWidget构造函数父类是0，到了Qt5变成了Q_NULLPTR，到了Qt6居然用的是默认的c++标准中的nullptr而不是Qt自定义定义的Q_NULLPTR（同样的还有Q_DECL_OVERRIDE换成了用override等），可能是为了彻底抛弃历史包袱拥抱未来。

```
//下面依次是Qt4/5/6的写法
MainWindow(QWidget *parent = 0);
MainWindow(QWidget *parent = Q_NULLPTR);
MainWindow(QWidget *parent = nullptr);

//查阅Qt源码查看Q_NULLPTR原来是根据编译器定义来选择
#ifdef Q_COMPILER_NULLPTR
# define Q_NULLPTR         nullptr
#else
# define Q_NULLPTR         NULL
#endif

//Qt高版本兼容低版本写法比如Qt5/6都支持 *parent = 0 这种写法。
```

40. 对于委托的进度条样式QStyleOptionProgressBar类的属性，在Qt4的时候不能设置横向还是垂直样式，默认横向样式，要设置orientation需要用另外的QStyleOptionProgressBarV2。从Qt5开始新增了orientation和bottomToTop属性设置。在Qt6的时候彻底移除了orientation属性，只有bottomToTop属性，而且默认进度是垂直的，很操蛋，理论上默认应该是横向的才对，绝大部分进度条场景都是横向的。这个时候怎么办呢，原来现在的处理逻辑改了，默认垂直的，如果要设置横向的直接设置 styleOption.state |= QStyle::State_Horizontal 这种方式设置才行，而Qt6以前默认方向是通过 orientation 值取得，这个State_Horizontal从Qt4就一直有，Qt6以后要主动设置下才是横向的就是。

## 05：41-50

41. Qt6.2版本开始增加了对多媒体模块的支持，但是在mingw编译器下还是有问题，直到6.2.2才修复这个问题，官网解释是因为mingw编译器版本不支持，到6.2.2采用了新的mingw900_64，这个编译器版本才支持。所以理论上推荐从6.2.2开始使用新的Qt6。
42. QTextStream中的setCodec方法改成了setEncoding，参数变了，功能更强大。

```
QTextStream stream(&file);
#if (QT_VERSION < QT_VERSION_CHECK(6,0,0))
stream.setCodec("gbk");
#else
stream.setEncoding(QStringConverter::System);
#endif
```

43. QModelIndex的查找子节点child函数去掉了，但是查找父节点parent函数保留，查阅代码得知之前的child函数就是封装的model->index(row, column, QModelIndex)函数。

```
//下面两个函数等价 如果要兼容Qt456则用下面这个方法
QModelIndex index = indexParent.child(i, 0);
QModelIndex index = model->index(i, 0, indexParent);

//下面两个函数等价 如果要兼容Qt456则用下面这个方法
QModelIndex indexChild = index.child(i, 0);
QModelIndex indexChild = model->index(i, 0, index);
```

44. 之前QPixmap类中的静态函数grabWindow和grabWidget彻底废弃了，改成了用QApplication::primaryScreen()->grabWindow，其实这个从Qt5开始就建议用这个。

```
#if (QT_VERSION >= QT_VERSION_CHECK(5,0,0))
    QPixmap pixmap = QApplication::primaryScreen()->grabWindow(widget->winId());
#else
    QPixmap pixmap = QPixmap::grabWidget(widget->winId());
#endif
```

# 3 Qt安卓经验

## 01：01-05

1. pro中引入安卓拓展模块 QT += androidextras 。
2. pro中指定安卓打包目录 ANDROID_PACKAGE_SOURCE_DIR = $$PWD/android 指定引入安卓特定目录比如程序图标、变量、颜色、java代码文件、jar库文件等。

- AndroidManifest.xml 每个程序唯一的一个全局配置文件，里面xml格式的数据，标明支持的安卓版本、图标位置、横屏竖屏、权限等。这个文件是最关键的，如果没有这个文件则Qt会默认生成一个。
- android/res/drawable-hdpi drawable-xxxhdpi 等目录下存放的是应用程序图标。
- android/res/layout 目录下存放的布局文件。
- android/res/values/libs.xml 存储的一些变量值。
- android/libs 目录下存放的jar库文件。
- android/src 目录下存放的java代码文件，可以是根据包名建立的一层层子目录，也可以直接在src目录下。
- 其他目录自行搜索安卓目录规范。
- 后面的说明统一用的android目录举例，其实你可以改成任意目录，比如你的代码目录下是xxoo存放的安卓相关的打包文件，你就写成 ANDROID_PACKAGE_SOURCE_DIR = $$PWD/xxoo 。

3. java类名必须和文件名完全一致，区分大小写。
4. java类必须在android/src目录下不然不会打包到apk文件，可以是子目录比如 android/src/com/qt 。
5. Qt代码中的QAndroidJniObject指定传入的java包名，必须严格和java文件package完全一致，不然程序执行到此处会因为找不到而崩溃。

- android/scr/MainActivity.java 顶部 没有 package 则代码中必须是 QAndroidJniObject javaClass("MainActivity");
- android/scr/MainActivity.java 顶部 package com.qandroid; 则代码中必须是 QAndroidJniObject javaClass("com/qandroid/MainActivity");
- android/scr/com/example/MainActivity.java 顶部 package com.qandroid; 则代码中必须是 QAndroidJniObject javaClass("com/qandroid/MainActivity");
- android/scr/com/example/MainActivity.java 顶部 package com.example.qandroid; 则代码中必须是 QAndroidJniObject javaClass("com/qandroid/example/MainActivity");
- 总之这个包名是和代码中的package后面一段吻合，而不是目录路径。为了统一管理方便查找文件，建议包名和目录路径一致。

## 02：06-10

6. Qt只能干Qt内部类的事情，做一些简单的UI交互还是非常方便，如果涉及到底层操作，还是需要熟悉java会如虎添翼，一般的做法就是写好java文件调试好，提供静态方法给Qt调用，这样通过QAndroidJniObject这个万能胶水可以做到Qt程序调用java中的函数并拿到执行结果，也可以接收java中的函数。

7. pro中通过 OTHER_FILES += android/AndroidManifest.xml OTHER_FILES += android/src/JniMessenger.java 引入文件其实对整个程序的编译打包没有任何影响，就是为了方便在QtCreator中查看和编辑。
8. 在Qt中与安卓的java文件交互都是用万能的QAndroidJniObject，可以执行java类中的普通函数、静态函数，可以传类对象jclass、类名className、方法methodName、参数，也可以拿到执行结果返回值。 (I)V括号中的是参数类型，括号后面的是返回值类型，void返回值对应V，由于String在java中不是数据类型而是类，所以要用Ljava/lang/String;表示，其他类作为参数也是这样处理。

- 调用实例方法：callMethod、callObjectMethod。
- 调用静态方法：callStaticMethod、callStaticObjectMethod。
- 不带Object的函数名用来执行无返回值或者常规返回值int、float等的方法。
- 如果返回值是String或者类则需要用带Object的函数名来执行，返回QAndroidJniObject类型再转换处理拿到结果，比如toString拿到字符串。

9. 各种参数和返回值示例。

```
package org.qt;
import org.qt.QtAndroidData;

public class QtAndroidTest
{
    //需要通过实例来调用 测试发现不论 private public 或者不写都可以调用 我擦
    private void printText()
    {
        System.out.println("printText");
    }

    public static void printMsg()
    {
        System.out.println("printMsg");
    }

    public static void printValue(int value)
    {
        System.out.println("printValue:" + value);
    }

    public static void setValue(float value1, double value2, char value3)
    {
        System.out.println("value1:" + value1 + " value2:" + value2 + "
value3:" + value3);
    }

    public static int getValue()
    {
        return 65536;
    }

    public static int getValue(int value)
    {
        return value + 1;
    }

    public static void setMsg(String message)
    {
```

```
        System.out.println("setMsg:" + message);
    }

    public static String getMsg()
    {
        return "hello from java";
    }

    public static void setText(int value1, float value2, boolean value3, String message)
    {
        System.out.println("value1:" + value1 + " value2:" + value2 + " value3:" + value3 + " message:" + message);
    }

    public static String getText(int value1, float value2, boolean value3, String message)
    {
        //同时演示触发静态函数发给Qt
        QtAndroidData.receiveData("message", "你好啊 java");

        //下面两种办法都可以拼字符串
        return "value1:" + value1 + " value2:" + value2 + " value3:" + value3 + " message:" + message;
        //return "value1:" + String.valueOf(value1) + " value2:" + String.valueOf(value2) + " value3:" + String.valueOf(value3) + " message:" + message;
    }
}
```

```
#include "androidtest.h"

//java类对应的包名+类名
#define className "org/qt/QtAndroidTest"

void AndroidTest::test()
{
    jint a = 12;
    jint b = 4;
    //可以直接调用java内置类中的方法
    jint max = QAndroidJniObject::callStaticMethod<jint>("java/lang/Math", "max", "(II)I", a, b);

    //jclass javaMathClass = "java/lang/Math";
    jdouble value = QAndroidJniObject::callStaticMethod<jdouble>("java/lang/Math", "random");

    qDebug() << "111" << max << value;
}

void AndroidTest::printText()
{
    QAndroidJniEnvironment env;
    jclass clazz = env.findClass(className);
    QAndroidJniObject obj(clazz);
```

```
    obj.callMethod<void>("printText");
}

void AndroidTest::printMsg()
{
#if 0
    //查看源码得知不传入jclass类的函数中内部会自动根据类名查找jclass
    QAndroidJniEnvironment env;
    jclass clazz = env.findClass(className);
    QAndroidJniObject::callStaticMethod<void>(clazz, "printMsg");
#else
    //没有参数和返回值可以忽略第三个参数
    QAndroidJniObject::callStaticMethod<void>(className, "printMsg");
    //QAndroidJniObject::callStaticMethod<void>(classNameTest, "printMsg", "()V");
#endif
}

void AndroidTest::printValue(int value)
{
    QAndroidJniObject::callStaticMethod<jint>(className, "printValue", "(I)I", (jint)value);
}

void AndroidTest::setValue(float value1, double value2, char value3)
{
    QAndroidJniObject::callStaticMethod<void>(className, "setValue", "(FDC)V", (jfloat)value1, (jdouble)value2, (jchar)value3);
}

int AndroidTest::getValue(int value)
{
    //java类中有两个 getValue 函数 一个需要传参数
    //jint result = QAndroidJniObject::callStaticMethod<jint>(className, "getValue");
    jint result = QAndroidJniObject::callStaticMethod<jint>(className, "getValue", "(I)I", (jint)value);
    return result;
}

void AndroidTest::setMsg(const QString &msg)
{
    QAndroidJniObject jmsg = QAndroidJniObject::fromString(msg);
    QAndroidJniObject::callStaticMethod<void>(className, "setMsg", "(Ljava/lang/String;)V", jmsg.object<jstring>());
}

QString AndroidTest::getMsg()
{
    QAndroidJniObject result = QAndroidJniObject::callStaticObjectMethod(className, "getMsg", "()Ljava/lang/String;");
    return result.toString();
}

```

```
void AndroidTest::setText(int value1, float value2, bool value3, const QString &msg)
{
    QAndroidJniObject jmsg = QAndroidJniObject::fromString(msg);
    QAndroidJniObject::callStaticMethod<void>(className, "setText", "(IFZLjava/lang/String;)V", (jint)value1, (jfloat)value2, (jboolean)value3, jmsg.object<jstring>());
}

QString AndroidTest::getText(int value1, float value2, bool value3, const QString &msg)
{
    QAndroidJniObject jmsg = QAndroidJniObject::fromString(msg);
    QAndroidJniObject result = QAndroidJniObject::callStaticObjectMethod(className, "getText", "(IFZLjava/lang/String;)Ljava/lang/String;", (jint)value1, (jfloat)value2, (jboolean)value3, jmsg.object<jstring>());
    return result.toString();
}
```

10. 在原生Android开发中，不同页面会定义不同的Activity。但使用Qt Quick、Flutter等采用Direct UI方式实现的第三方开发框架则只定义了一个Activity。里面不同页面实际都是使用OpenGL等直接绘制的。https://blog.csdn.net/LCSENs/article/details/100182235

## 03：11-15

11. 安卓中一个界面窗体对应一个Activity，多个界面就有多个Activity，而在Qt安卓程序中，Qt这边只有一个Activity那就是QtActivity（包名全路径 org.qtproject.qt5.android.bindings.QtActivity），这个QtActivity是固定的写好的，整个Qt程序都是在这个QtActivity界面中。你打开AndroidManifest.xml文件可以看到对应节点有个name=org.qtproject.qt5.android.bindings.QtActivity，所以如果要让Qt程序能够更方便通畅的与对应的java类进行交互（需要上下文传递Activity的，比如震动，消息提示等），建议新建一个java类，继承自QtActivity即可，这样相当于默认Qt启动的就是你java类中定义的Activity，可以很好的控制和交互。
12. 由于AndroidManifest.xml文件每个程序都可能不一样，为了做成通用的组件，这就要求可能不能带上AndroidManifest.xml文件，这样的话每个Qt安卓程序都启动默认内置的Activity，如果依赖Activity上下文的执行函数需要传入Qt的Activity才行，这里切记Qt的Activity包名是Lorg/qtproject/qt5/android/bindings/QtActivity; 之前顺手想当然的写的 Landroid/app/Activity; 发现死活不行，原来是包名错了。
13. 一个Qt安卓程序中可以有多个Java类，包括继承自Activity的类（这样的Activity可以通过QtAndroid::startActivity函数来调用），但是只能有一个通过AndroidManifest.xml文件指定的Activity，不指定会默认一个。如果java类中不需要拿到Qt的Activity进行处理的，可以不需要继承任何Activity，比如全部是运算的静态函数。
14. 在java类中如果上面没有主动引入包名，则下面需要写全路径，引入了则不需要全路径可以直接用（包括枚举值都可以直接写，比如 VIBRATOR_SERVICE 这种枚举值引入了包名后不需要写 android.content.Context.VIBRATOR_SERVICE），建议引入包名，比如上面写了 import org.qtproject.qt5.android.bindings.QtActivity; 则下面继承类可以直接写 public class QtAndroidActivity extends QtActivity，如果没有引入则需要写成 public class QtAndroidActivity extends org.qtproject.qt5.android.bindings.QtActivity 。
15. 建议搭配 android studio 工具开发，因为在 android studio 中写代码都有自动语法提示，包名会提示自动引入，可以查看有那些函数方法等，还可以校验代码是否正确，而如果在QtCreator中手写有时候可能会写错，尤其是某个字母写错，当然这种错误是编译通不过的，会提示错误在哪行。

## 04：16-20

16. 用Qt做安卓开发最大难点两个，第一个就是传参数这些奇奇怪怪的字符（Ljava/lang/String;）啥意思，如何对应，这也不是Qt故意为难初学者啥的，因为这套定义机制是安卓系统底层要求的，系统层面定义的一套规范，其实这个在帮助文档中写的很清楚，都有数据类型对照表，用熟悉了几次就很简单了。第二个难点就是用java写对应的类，如果是会安卓开发的人来说那不要太简单，尤其是搜索那么方便一大堆，没有搞过安卓开发的人来说就需要学习下，这个没有捷径，只是希望Qt能够尽可能最大化的封装一些可以直接使用的类，比如后期版本就提供了权限申请的类QtAndroid::requestPermissionsSync 之类的，用起来就非常的爽，不用自己写个java类调来调去的。
17. 理论上来说按照Qt提供的万能大法类QAndroidJniObject，可以不用写java类也能执行各种处理，拿到安卓库中的属性和执行方法，就是写起来太绕太费劲，在java类中一行代码，这里起码三行，所以终极大法就是熟悉安卓开发，直接封装好java类进行调用。
18. 测试发现GetStringUTFChars方法对应的数据字符串中不能带有temp字样，否则解析有问题，不知什么原因。
19. 数据类型参数和返回值类型必须完全一致，否则执行会提示找不到对应的函数，有返回值一定要写上返回值。
20. jar文件对包名的命名没有要求，只要放在android/libs目录下即可，安卓底层是通过包名去查找，而不是通过文件名，你甚至可以将原来的包名重新改成也可以正常使用，比如classes.jar改成test.jar也能正常使用。

## 05：21-25

21. 关于权限设置，在早期的安卓版本，所有权限都写在全局配置文件AndroidManifest.xml中，这种叫安装时权限，就是安装的时候告诉安卓系统当前app需要哪些权限。大概从安卓6开始，部分权限需要动态申请，这种叫动态权限，这种申请到的权限也可以动态撤销，就是要求程序再次执行代码去向系统申请权限，比如拍照、存储读写等。也不是所有的权限都改成了动态申请，意味着兼容安卓6以上的系统你既要在AndroidManifest.xml中写上要求的权限，也要通过checkPermission申请你需要的权限。
22. android studio 新建并生产jar包步骤。

- 第一步：文件（File）-》新建（new）-》项目（new project）-》空白窗体（empty activity）。
- 第二步：刚才新建好的项目鼠标右键新建（new）-》模块（new module）-》安卓库（android library）。
- 说明：如果选择的不是安卓库（android library）而是java库（Java Library），则直接编译出来的就是jar文件，默认包名 com.example.lib.MyClass。推荐选择java库，编译后不用去一堆文件中找jar文件。
- 第三步：写好库名字，根据项目需要选择好最低sdk版本-》完成。
- 第四步：在刚才新建好的库项目mylibrary，依次找到子节点src/main/java/com.example.mylibrary上鼠标右键新建-》class类。切记是这个节点不是java节点或者其他节点。
- 第五步：写好你的类方法函数等。

```
package com.example.mylibrary;
public class Test {
    public static int add(int a, int b) {
        return a + b;
    }
}
```

- 第六步：选中库项目mylibrary，菜单执行编译（build）-》编译库（make module xxx）。

- 第七步：此时在mylibrary/build目录下有outputs目录和intermediates目录，其中outputs/aar目录下是生成的Android库项目的二进制归档文件，包含所有资源，class以及res资源文件全部包含。有时候我们仅仅需要jar文件，只包含了class文件与清单文件 ，不包含资源文件，如图片等所有res中的文件。需要到intermediates/aar_main_jar/debug目录下，可以看到classes.jar，将这个拷贝出来使用即可。当然你也可以对刚才的aar文件用解压缩软件解压出来也能看到classes.jar，是同一个文件。
- 其他：调用jar包非常简单，只需要将jar文件放在你的项目的libs目录下即可，对应的包名和函数一般jar包提供者会提供，没有提供的话，可以在android studio中新建空白项目，切换到project视图，找到libs目录，鼠标右键最下面作为包动态库添加到项目，导入包完成以后会自动在libs目录列出，双击刚刚导入的包然后就自动列出对应的类和函数。

23. Qt安卓使用jar包步骤。

- 第一步：将classes.jar放到android/libs目录下，为啥是这个目录？因为这是安卓的规则约定，这个目录就是放库文件，放在这个目录下的文件会自动打包编译到apk文件中。
- 第二步：调用jar文件之前，前提是你知道jar文件中的函数详细信息，这个一般jar提供者会提供好手册，如果代码没有混肴的话，你可以在android studio中双击打开查阅具体的函数。
- 第三步：如果jar文件中的函数简单，直接拿到结果不需要绕来绕去，可以直接写Qt类来调用；如果还是很复杂，建议再去新建java类处理完再交给Qt，当然也可以让jar的作者尽可能封装函数的时候就做好，尽量提供最简单的接口返回需要的数据。比如返回图片数据可以做成jar内部存储好图片，然后返回图片路径即可，不然有些数据转换也挺烦。
- 第四步：编写最终的调用函数。

```
int AndroidJar::add(int a, int b)
{
#ifdef Q_OS_ANDROID
    const char *className = "com/example/mylibrary/Test";
    jint result = QAndroidJniObject::callStaticMethod<jint>(className, "add", "(II)I", (jint)a, (jint)b);
    return result;
#endif
}
```

24. Qt6中对安卓支持部分做了大的改动，目前还不完善，如果是不涉及到与java交互的纯Qt项目，可以正常移植，涉及到的暂时不建议移植到Qt6，等所有类完善了再说。

- 移除了安卓插件androidextras，将其中部分功能类移到core模块中，不需要额外引入。
- 类名发生了变化，比如QAndroidJniObject改成了QJniObject、QAndroidJniEnvironment改成了QJniEnvironment，可能是为了统一移动开发平台类，弱化安卓的影响。
- 对应的安卓jdk要用jdk11而不是jdk1.8，Qt5.15两个都支持，建议就统一用jdk11。
- 对应封装的java类包名去掉了qt5标识，org.qtproject.qt5.android.bindings.QtActivity改成了org.qtproject.qt.android.bindings.QtActivity、org.qtproject.qt5.android.bindings.QtApplication改成了org.qtproject.qt.android.bindings.QtApplication。
- 对安卓最低sdk有要求，所以建议在配置AndroidManifest.xml文件的时候不要带上最低版本要求。
- 对AndroidManifest.xml文件内容有要求，之前Qt5安卓的不能在Qt6安卓下使用，具体内容参见示例下的文件。
- 对应示例demo在 C:\Qt\Examples\Qt-6.3.0\corelib\platform 目录下，之前是 C:\Qt\Examples\Qt-5.15.2\androidextras ，目前就一个示例，可能因为其他类还没有移植好。

25. 如果想要安卓全屏遮挡住顶部状态栏，可以在main函数中将show改成showFullScreen即可，当然也可以采用java的方式在onCreate函数中加一行 getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);

## 06：25-30

26. 横竖屏切换的识别，在Qt中会同时反映到resizeEvent事件中，你可以在这个是尺寸变化后读取下当前屏幕是横屏还是竖屏，然后界面上做出调整，比如上下排列改成左右排列。
27. 由于不同Qt版本对应的安卓配置文件 AndroidManifest.xml 内容格式不一样，高版本和低版本模板格式互不兼容，所以建议使用自己的Qt版本创建的 AndroidManifest.xml 文件，创建好以后如果使用的是自己重新定义的java文件的启动窗体则需要将 AndroidManifest.xml 文件中的 android:name="org.qtproject.qt5.android.bindings.QtActivity" 换掉就行。
28. 如果自己用android studio编译的jar文件放到Qt项目的libs目录下，导致编译通不过，提示 com.android.dx.cf.iface.ParseException: bad class file magic 之类的，那是因为jdk版本不一致导致的，你可能需要在android studio项目中找到模块编jdk版本设置的地方降低版本，比如你用的ndk是r14，则需要选择jdk1.6或者jdk1.7。一般来说高版本兼容低版本，因为ndk版本太低无法兼容jdk1.8。后面发现如果直接新建的是java库（Java Library）则不存在这个问题，如果选择的是安卓库（android library）就可能有这个问题。
29. 安卓项目配置文件是固定的名字 AndroidManifest.xml ，改成其他名字就不认识，不要想当然改成其他名字导致无法正常识别。
30. AndroidManifest.xml文件中的package="org.qtproject.example"是包名，也是整个apk程序的内部唯一标识，如果多个apk这个包名一样，则会覆盖，所以一定要注意不同的程序记得把这个包名改成你自己的。这个包名也决定了java文件中需要使用资源文件时候的引入包名 import org.qtproject.example.R; 如果包名不一样则编译都通不过。

# 4 Qt设计模式

**读《c++ Qt设计模式》书籍整理的一点经验。此书和官方的《C++ GUI Qt4编程》一起的。**

1. 通常而言，好的做法是在包含了Qt头文件之后再包含非Qt头文件，由于Qt（为编译器和预处理器）定义了许多符号，这使得避免名称冲突变得更容易，也更容易找到文件。

```
#include "frminput2019.h"
#include "ui_frminput2019.h"

#include "qdatetime.h"
#include "qdebug.h"

#include "input2019.h"
#include "inputnumber.h"
```

2. 一种好的编程实践是在代码中使用const实体而不是嵌入数字型常量（有时称他们为“幻数”）。如果以后需要修改他的值时，就可以获得这种灵活性。一般而言，将常量“孤立”出来，可提高程序的可维护性。

```
//不推荐写法
for (int i = 0; i < 100; ++i) {
    ...
}

//推荐下面的写法
const int count = 100;
for (int i = 0; i < count; ++i) {
    ...
}
```

3. 内存管理使程序员获得了强大的能力，但是，“权力越大，责任越大”。
4. 只要有可能，就应当使用列表而不是数组，比如应该使用 QList 代替 int []，在c++中数组被看成是“邪恶的”。
5. 在利用Qt编写程序的过程中，因为Qt的父子所有权继承关系，很少会用到智能指针，因为需要调用delete的情况很少。任何时候只要我们需要调用delete，或者是需要将某个指针设定为0时，应该考虑使用一个智能指针。
6. 实际上，我们不能完全确定使用多线程就一定能够真正改善程序的性能，例如，如果增加使用线程的数量，使他与系统可用的内核数量成正比，这样做或许还会降低程序的性能，因为所获得的收益会因线程竞争的剧增而消失殆尽。有时候，单线程中最有效的算法在多线程中却不一定有效。因此，如果真的是想改进程序的性能，理想的做法是，使用不同的实现方法，并与他们的性能进行比较后加以分类，当然测试对比的前提是使用完全相同的硬件和软件配置环境。
7. 在源代码中关于文件路径，使用 / 会更方便一些，因为无论是在何种平台上，Qt都能理解他，不需要对他进行转换。但是，当我们想为用户显示路径时，最好还是根据应用程序所在平台的正确形式来显示他。
8. 当我们有很多项数据需要处理时，比如成千上万或者更多，那么为每个处理都创建一个线程可能导致大量的开销，这样来依次处理数据或许更快些。一种解决办法就是创建少量的辅助线程，并让每个线程只处理一组数据。
9. 

# 5 Qt大佬专区

## 5.1 酷码大佬

**微信：Kuma-NPC**

1. 关于Qt事件传递的一个说明：

- 通常写win32程序，鼠标消息应该是直接发给指定窗口句柄的，指定窗口没有处理就会转化成透传消息，交给父窗口处理。你在一个普通文字label上点击，父窗口也能收到鼠标事件。
- Qt应该是所有消息都发给了顶层窗口，所以事件分发逻辑是自己处理，主窗口收到鼠标事件然后Qt自己分发给指定子控件，QEvent会有ignore或者accept表示自己处理了没有，例如鼠标点击事件，事件分发器发现没有被处理，数据重新计算然后分发给父窗口。这样父窗口收到的事件坐标就是基于自己窗口内的。用eventFilter就需要自己计算坐标。
- 再比如，当使用QDialog，放一个QLineEdit并设置焦点，按Esc时QDialog也会自动关闭，本质上就是因为QLineEdit并不处理Esc的按键事件，透传给了QDialog。

## 5.2 小豆君

1. 无论你是学Qt，Java，Python或其它，都需要明白一个道理：摒弃掉你的好奇心，千万不要去追求第三方类或工具是怎么实现的，这往往会让你收效甚微，其实，你只需要熟练掌握它的接口，知道类的目的即可，不可犯面向过程的毛病，刨根问底。记住，你的目标是让其它工具为你服务，你要踩在巨人的肩膀上创造世界。
2. Qt真正的核心：元对象系统、属性系统、对象模型、对象树、信号槽。往死里啃这五大特性，在你的项目中，逐渐的设法加入这些特性，多多练习使用它们，长此以往你会收获意想不到的效果。
3. 一边请教别人，一边多多重构，其实编码这条路虽然有人给你指路，但真正走下去的是你自己，当你真正走完时，你的编码水平一定会有非常大的提升。也许别人1000行的代码，在你这里几十行就搞定了，这也正事Qt的魅力。
4. 在阅读Qt的帮助文档时，要静下心来，不要放过每一句，记住在文档中没有废话，尤其是每段的开头。

# 6 其他经验

1. Qt界的中文乱码问题，版本众多导致的如何选择安装包问题，如何打包发布程序的问题，堪称Qt界的三座大山！
2. 在Qt的学习过程中，学会查看对应类的头文件是一个好习惯，如果在该类的头文件没有找到对应的函数，可以去他的父类中找找，实在不行还有爷爷类，肯定能找到的。通过头文件你会发现很多函数接口其实Qt已经帮我们封装好了，有空还可以阅读下他的实现代码。
3. Qt安装目录下的Examples目录下的例子，看完学完，月薪20K起步；Qt常用类的头文件的函数看完学完使用一遍并加以融会贯通，月薪30K起步。
4. Qt在开发阶段不支持中文目录（运行阶段可以，比如打包发布的程序放到中文目录运行是ok的），切记，这是无数人可能犯的错误，在安装Qt集成开发环境以及编译器的时候，务必记得目录必须英文，Qt项目源码也必须是英文目录，否则很可能不正常，建议尽量用默认的安装位置。
5. 如果出现崩溃和段错误，80%都是因为要么越界，要么未初始化，死扣这两点，80%的问题解决了。
6. Qt一共有几百个版本，关于如何选择Qt版本的问题，我一般保留四个版本，为了兼容Qt4用4.8.7，最后的支持XP的版本5.7.0，最新的长期支持版本比如5.15，最高的新版本比如5.15.2。强烈不建议使用4.7以前和5.0到5.3之间的版本（Qt6.0到Qt6.2之间、不含6.2的版本也不建议，很多模块还没有集成），太多bug和坑，稳定性和兼容性相比于之后的版本相当差，能换就换，不能换睡服领导也要换。如果没有历史包袱建议用5.15.2，目前新推出的6.0版本也强烈不建议使用，官方还在整合当中，好多类和模块暂时没有整合，需要等到6.2.2版本再用。
7. Qt和msvc编译器常见搭配是Qt5.7+VS2013、Qt5.9+VS2015、Qt5.12+VS2017、Qt5.15+VS2019、Qt6.2+VS2019，按照这些搭配来，基本上常用的模块都会有，比如webengine模块，如果选用的Qt5.12+msvc2015，则很可能官方没有编译这个模块，只是编译了Qt5.12+msvc2017的，如果一定要用msvc2015不想换msvc2017则只能选择Qt5.9+msvc2015套件，或者自行源码重新编译（这个难度超大，初学者绕过）。
8. Qt默认有对应VS版本，在下载对应VS插件的时候心里要有个数，官方默认提供的是原配的插件，如果想要Qt4.8+VS2015的插件，需要自行编译。一般来说是Qt4.8原配VS2010，Qt5.6原配VS2013，Qt5.9原配VS2015，Qt5.12原配VS2017，Qt5.15原配VS2019，切记：原配最好。
9. 用Qt做开发机器建议用win10，尤其是2021年以后新发布的Qt版本，比如Qt5.12.12、Qt5.15.2、Qt6.2.2等，因为很可能自带的QtCreator用的最新的版本，Qt6开始不再支持win7，或者由于其他的原因，对win7的支持不友好，会出现奇奇怪怪的问题等，所以又是没得选必须用win10。建议各位拥抱新时代的变化，这世上唯一不变的只有变化。
10. 新版本Qt安装包安装的时候需要填写注册信息，如果不想填写，先禁用网卡，在运行安装包，可以直接跳过这一步进行安装。**从Qt5.15开始不再提供离线安装包，意味着必须使用在线安装器安装Qt的后续版本，必须填写用户信息，没得选。**

11. 终极秘籍：如果遇到问题搜索Qt方面找不到答案，试着将关键字用JAVA C# android打头，你会发现别有一番天地，其他人其他语言其他领域很可能做过！
12. 如果Qt能从下面几个方面努力，相信会更有发展前景。

- QWidget支持CSS3，具有诸多的牛逼的效果，目前支持的是CSS2。
- QWidget支持GPU绘制，可选切换CPU或者GPU，提升绘制效率，利用现在强大的硬件。
- Qml无缝支持js，可以利用现在各种js轮子，指数级提升qml的项目范围。
- 支持将程序转成web运行，比如转成cgi之类的程序，目前Qt for WebAssembly很鸡肋，功能极其有限，sql/network/本地访问等都不支持，首次加载速度超慢，大部分Qt类还不支持。

13. Qt自从4.7以后引入的QML。从此以后，Qt开发就分成了两种流派，一者使用原来的C++ 语言进行开发，另外一种使用QML语言进行开发。这下搞得嘞，经常吵吵不亦乐乎，在Qt界从此就有两大阵营产生激烈的纷争，那就是选用qml还是widget好，大量初学者也会问这个问题，有以下几点总结。

- widget属于传统界面开发，和VB/VC/Delphi等拖曳控件开发类似，走CPU绘制，能最大化的兼容现有的硬件和过去的相对偏低性能的硬件。
- qml属于新时代的产物，大概从2010年开始，和flutter/Electron等web开发框架及移动开发框架类似，为了适应各种移动端开发及动画流畅性触摸丝滑体验、充分利用和“榨干”现在的GPU性能，把CPU留出来给用户最大化发挥。
- 硬件性能越好，GPU越是强劲，qml的综合性能越是完爆widget，反之对比也是指数级的。除了极其省成本的嵌入式硬件领域或者国产CPU等，其他领域的硬件性能都是暴增。
- widget主要集中在金融、军工、安防、航天、船舶、教育等领域，qml主要集中在汽车仪表、车机、直播等领域。
- 目前国内widget多于qml，国外可能偏向qml，这个不难看出，流行的移动端开发框架都是国外开发者居多。
- 可预见的十年内，这两者将长期并存，官方基本不再更新widget而是主推qml，意味着将来对qml的性能优化只增不减，未来趋势是qml。
- 没有编程经验的新手qml学习成本更低，而从VB/VC等传统软件开发转过来的从业者更适合学习widget。
- 有的时候不禁要问，既生widget何生qml，学习成本和选择又多了，其实这正是和这个世界的哲学一样：世界是简单的又是复杂的。为了适应各种需求和满足需要。
- 总之，无论qml还是widget，和找老婆一样，适合自己的就是最好的，自己擅长哪个就用哪个。
- 如果还不知道擅长哪个，有空就两个都学，学习过程中自己就会有切身感受和对比，能者多劳多多益善。能够顺利的最快的完成老板的任务给老板赚钱才是王道。
- 网友补充：如果你的软件最终是手指操作的多，就用qml，如果是鼠标操作的多，就选择widget。

14. 写程序过程中发现问题，比如有些问题是极端特殊情况下出现，最好找到问题的根源，有时候肯定多多少少会怀疑是不是Qt本身的问题，怀疑是对的，但是99.9%的问题最终证实下来还是自己的代码写的不够好导致的，如果为了赶时间老板催的急，实在不行再用重启或者复位大法，比如搞个定时器、线程、网络通信啥的去检测程序是否正常，程序中某个模块或者功能是否正常，不正常就复位程序或者重启程序，在嵌入式上还可以更暴力一点就是系统重启和断电重启。
15. 写程序过程中尤其要注意32位的库和64位的库互不兼容，比如32位的程序引用64位的库，64位的程序引用32位的库，都是编译通不过的，而在windows64位系统中是能够运行32位程序的，因为64位的系统提供了32位的运行环境，一般目录在Program Files(x86)，32位的程序在64位的环境中最终引用的还是32位的库。关于如何判断自己的Qt库是多少位，有个误区就是很多人要么看成了QtCreator的关于信息中列出的位数，要么以为自己是64位的系统就认为是64位的Qt，最终要在Qt构建套件中查看具体位数，大概从Qt5.14开始基本上很少提供32位的库，尤其是Qt6.0以后基本上默认就是只有64位的库了，这也是顺应时代潮流，毕竟不久的将来（个人预计2030年以前）基本上32位的系统占比不超过1%，放心大胆的用64位的库吧，抛弃烦人的32位以及XP系统。
16. 关于程序中动态和静态的一点个人理解：

- 在Qt程序中，分动态库版本的Qt和静态库版本的Qt。
- 官方默认提供的二进制包就是动态库版本的Qt，如果自行编译则编译的时候对应参数 -shared。
- 静态库版本的Qt需要自行编译，编译的时候对应参数 -static，（理论上无论商业非商业使用Qt静态库需要收费，因为静态编译后都看不到Qt的相关库文件）。
- 使用动态库的Qt支持编译生成动态库和静态库（CONFIG += staticlib）的程序。
- 使用动态库的Qt程序支持动态库的引用（引用的时候 LIB += ，运行的时候需要动态库文件比如 .dll .so 文件支持）。
- 使用动态库的Qt程序支持静态库的引用（引用的时候 LIB += ，运行的时候无需库文件支持，可以理解为该文件已经和可执行文件合二为一，缺点是可执行文件体积变大）。
- 通过生成文件的个数和大小可以发现，静态库相当于把运行时需要的文件也一并合并到一个文件了，而动态库是拆分成两个文件，一个用于编译，一个用于运行。
- 上述动态库的规则也通用于静态库。
- 此规则应该是通用于其他语言框架。
- 很多人有个误区包括几年前的我，以为要用Qt编写静态库就必须用静态的Qt库，其实动态库的Qt也可以编写静态的库，只是该库不会生成动态库文件。
- 如果要将Qt程序编译成静态的可执行文件（单个文件无依赖），前提是所用的Qt库必须静态的。

17. 后期的Qt版本，大致从5.15开始，就不在提供离线版本下载，需要自行通过在线安装器安装，由于默认服务器在国外，很多人反映下载的时候很慢，或者选择晚上的时候下载要快很多，为了解决这个烦人的问题，不至于时间都浪费在没有意义的等待上，有个极其简单的方法可以将速度提升几万倍，甚至冲坏你的硬盘。先下载 Fiddler5（尽量选择中文版本不然小白看不懂），双击打开程序后（可能win10自带的杀毒软件会报毒删除，临时停用杀毒软件或者恢复可信任文件即可），在底部的输入栏中输入 urlreplace download.qt.io mirrors.ustc.edu.cn/qtproject/ 回车应用，然后再去打开安装器在线安装，世界突然变得非常美好。
18. 最后一条：珍爱生命，远离编程。祝大家头发浓密，睡眠良好，情绪稳定，财富自由！

# 7 杂七杂八

## 7.1 推荐开源主页

| 名称 | 网址 |
| --- | --- |
| Qt技术交流群1 | 46679801(已满员) |
| Qt技术交流群2 | 573199610(未满员) |
| Qt高级学习群 | 951393302(未满员，推荐此群) |
| Qt交流大会群 | 853086607(已满员) |
| QtWidget开源demo集合 | https://gitee.com/feiyangqingyun/QWidgetDemo |
| QtQuick/Qml开源demo集合 | https://gitee.com/jaredtao/TaoQuick |
| QtQuick/Qml开源demo集合 | https://gitee.com/zhengtianzuo/QtQuickExamples |

## 7.2 推荐网站主页

| 名称 | 网址 |
| --- | --- |
| qtcn | http://www.qtcn.org |
| 豆子的空间 | https://www.devbean.net |
| yafeilinux | http://www.qter.org |
| feiyangqingyun | https://blog.csdn.net/feiyangqingyun |
| **Qt作品大全** | https://qtchina.blog.csdn.net/article/details/97565652 |
| Qt系列文章 | https://blog.csdn.net/feiyangqingyun/category_11460485.html |
| 一去二三里 | http://blog.csdn.net/liang19890820 |
| 乌托邦2号 | http://blog.csdn.net/taiyang1987912 |
| foruok | http://blog.csdn.net/foruok |
| jason | http://blog.csdn.net/wsj18808050 |
| 朝十晚八 | http://www.cnblogs.com/swarmbees |
| BIG_C_GOD | http://blog.csdn.net/big_c_god |
| 公孙二狗 | https://qtdebug.com/qtbook |
| 雨田哥 | https://blog.csdn.net/ly305750665 |
| 郑天佐 | https://blog.csdn.net/zhengtianzuo06 |
| 寒山-居士 | https://blog.csdn.net/esonpo |
| 前行中小猪 | http://blog.csdn.net/goforwardtostep |
| 涛哥的知乎专栏 | https://zhuanlan.zhihu.com/TaoQt |
| Qt君 | https://blog.csdn.net/nicai_xiaoqinxi |

## 7.3 推荐学习网站

| 名称 | 网址 |
| --- | --- |
| Qt老外视频教程 | http://space.bilibili.com/2592237/#!/index |
| Qt维基补充文档 | https://wiki.qt.io/Main |
| Qt源码查看网站 | https://code.woboq.org/qt5 |
| Qt官方下载地址 | https://download.qt.io |
| Qt官方下载新地址 | https://download.qt.io/new_archive/qt/ |
| Qt国内镜像下载地址 | https://mirrors.cloud.tencent.com/qt |
| Qt安装包下载地址 | http://qthub.com/download/ |
| Qt最新版二进制包 | https://build-qt.fsu0413.me/ |
| Qt版本更新内容 | https://doc-snapshots.qt.io/qt6-6.2/whatsnew62.html |
| Qt中qmake变量说明 | https://doc.qt.io/qt-5/qmake-variable-reference.html |
| Qt入门最简单教程 | http://c.biancheng.net/qt/ |
| qss学习地址1 | http://47.100.39.100/qtwidgets/stylesheet-reference.html |
| qss学习地址2 | http://47.100.39.100/qtwidgets/stylesheet-examples.html |
| qss学习地址3 | https://doc.qt.io/qt-6/qstyle.html |
| 精美图表控件QWT | http://qwt.sourceforge.net/ |
| 精美图表控件QCustomPlot | https://www.qcustomplot.com/ |
| 免费图标下载 | http://www.easyicon.net/ |
| 图形字体下载 | https://www.iconfont.cn/ |
| 漂亮界面网站 | https://www.ui.cn/ |
| 微信公众号 | **官方公众号：Qt软件　亮哥公众号：高效程序员** |

# 8 书籍推荐

1. C++入门书籍推荐《C++ primer plus》，进阶书籍推荐《C++ primer》。
2. Qt入门书籍推荐霍亚飞的《Qt Creator快速入门》，Qt进阶书籍推荐官方的《C++ GUI Qt4编程》，qml书籍推荐《Qt5编程入门》，Qt电子书强烈推荐《Qt5.10 GUI完全参考手册》。

3. 强烈推荐程序员自我提升、修养、规划系列书《走出软件作坊》《大话程序员》《程序员的成长课》《解忧程序员》，受益匪浅，受益终生！