# Qt简介

Q\_PROPERTY

Q\_PROPERTY(bool focus READ hasFocus)

Q\_PROPERTY(bool enabled READ isEnabled WRITE setEnabled)

Q\_PROPERTY(QCursor cursor READ cursor WRITE SetCursor RESET unsetCursor)

setProperty可以在运行时为类定义一个新的属性，称之为动态属性

# 第2章 GUI应用程序设计基础

# 第3章 Qt类库概述

## 信号和槽

QMetaObject::Connection QObject::connect(

const QObject \*sender,

const char \*signal,

const QObject \*receiver,c

onst char \*member,

Qt::ConnectionType=Qt::AutoConnection)

/如果信号和槽函数带有参数，需要注明参数类型connect(spinNum,SIGNAL(valueChanged(int)),this,SLOT(updQt eStQt us(int));

QMetaObject::Connection QObject::connect(

cosnt QObject \*sender,

const QMetaMethod &signal,

const QObject \*receivers,

const QMetaMethod &method,

Qt::ConnectionType=Qt::AutoConnection)

/对于信号名称唯一的，没有参数不同而不同的信号函数 connect(lineEdit,&QLineEdit::textChanged,this,&widget::on\_textChanged) |最后一个参数：Qt：:ConnectionType表示了信号与槽槽之间的关联方式

最后一个参数：Qt:ConnectionType表示了信号与槽槽之间的关联方式Qt:AutoConnection(缺省值）：自动确定关联方式。

Qt::DirectConnection:信号被发射时，槽立即执行，槽函数与信号在同一线程Qt::QueuedConnection:事件循环回到接收者线程后执行槽，槽与信号在不同线程Qt::BlockingQueueConnection:与Qt:QueuedConnection相似，信号线程会被阻塞直到槽执行完毕。当槽函数与信号在同一线程，会造成死锁。

## Qt全局定义

<QtGlobal>头文件包含了Qt类库的一些全局定义：

·基本数据类型、函数、宏 Qt类的头文件都会包含该文件

全局变量定义确保在各个平台上各数据类型都有统一确定的长度

qreal 缺省是8字节 double 类型浮点数，如果Qt使用-qreal floQt 选项进行配置，就是4字节floQt 类型的浮点数

qfloQt 16是Qt5.9.0中新增的一个类，用于表示16位的浮点数，要使用qfloQt 16,需要包含头文件<QFloQt 16>

|  |  |  |
| --- | --- | --- |
| Qt数据类型 | 等效定义 | 字节数 |
| qint8 | signed char | 1 |
| qint16 | signed short | 2 |
| qint32 | signed int | 4 |
| qint64 | long long int | 8 |
| qlonglong | long long int | 8 |
| quint8 | unsigned char | 1 |
| quint16 | unsigned short | 2 |
| quint32 | unsigned int | 4 |
| quint64 | unsigned long long int | 8 |
| uchar | unsigned char | 1 |
| ushort | unsigned short | 2 |
| uint | unsigned int | 4 |
| ulong | unsigned long | 8 |
| qreal | double | 8 |
| qfloQt 16 |  | 2 |

## 容器类

顺序容器通过元素在容器中的位置顺序存储和访问

关联容器通过键（key)存储和读取元素

Qt的容器类分为顺序容器和关联容器。

顺序容器：QList,QLinkedList，QVector，QStack和QQueue

关联容器：QMap，QMultiMap，QHash,QMultiHash和QSet

QLinkedList:除了不提供下标索引的数据访问，和QList其他函数接口基本相同。

QVector：函数接口与QList几乎完全相同，访问性能更高，因为是连续存储。

|  |  |  |
| --- | --- | --- |
| 容器类 | 只读迭代器 | 读写迭代器 |
| QList<T>,QQueue<T> | QList<T>::sonst\_iterQt or | QList<T>::iterQt or |
| QLinkedListc<T> | QLinkedList<1>::const\_iterQt or | QLinkedList<T>::iterQt or |
| QVector<T>,QStack<T> | QVector<T>::const\_ilerQt or | QVector<T>::iterQt or |
| QMap<Key,P> QMultiMap<Kcy,T> | QMapsKey<T>const iterQt or | QMapsKey<T>::iterQt or |
| QSet<T> | QSet<T>::const\_iterQt or | QSet<T>::iterQt or |

可以使用const\_reverse\_iterQt or和reverse\_iterQt or定义相应的反向迭代器

STL类型的迭代器是数组的指针，所以“++”运算符使迭代器指向下一个数据项，运算符返回数据项内容。

Qt使用了隐式共享只有sizes发生了数据修改时，才会将共享对象的数据复制给sizes

对于STL类型的迭代器，隐式共享还涉及另外一个问题，即当有一个迭代器在操作一个容器变量时，不要去复制这个容器变量。

## Qt类库的模块

分为以下几大类：

·Qt 基本模块（Qt Essentials)：提供了Qt 在所有平台上的基本功能。

·Qt附加模块（QtAdd-Ons)：实现一些特定功能的提供附加价值的模块。

·增值模块（Value-AddModules)：单独发布的提供额外价值的模块或工具。

·技术预览模块（Technology Preview Modules)：一些处于开发阶段，但是可以作为技术预览使用的模块。

·Qt工具（Qt Tools)：帮助应用程序开发的一些工具。

模块 描述QtCore其他模块都用到的核心非图形类Qt Gul设计GUl界面的基础类，包括OpenGLQt Multimedia音频、视频、摄像头和广播功能的类Qt Multimedia Widgets实现多媒体功能的界面组件类

|  |  |  |
| --- | --- | --- |
| 基本模块 | | |
| 模块 | 描述 | |
| QtCore | 其他模块都用到的核心非图形类 | |
| Qt Gul | 设计GUl界面的基础类，包括OpenGL | |
| Qt Multimedia | 音频、视频、摄像头和广播功能的类 | |
| Qt MultimediaWidgets | 实现多媒体功能的界面组件类 | |
| Qt Network | 使网络编程更简单和轻便的类 | |
| Qt QML | 用于QML和lavaScript语言的类 | |
| QtQuick | 用于构建具有定制用户界面的动态应用程序的声明框架 | |
| Qt QuickControls | 创建桌面样式用户界面，基于Qt Quick的用户界面控件 | |
| Qt QuickDialogs | 用于Qt Quick的系统对话框类型 | |
| Qt QuickLayouts | 用于Qt Quick2界面元素的布局项 | |
| Qt sQL | 使用SQL用于数据库操作的类 | |
| Qt Test | 用于应用程序和库进行单元测试的类 | |
| Qt Widgets | 用于构建GUI界面的C++图形组件类 | |
|  |  | |
| Qt附加模块 | | |
| 模块 | 描述 | |
| Active Qt | 用于开发使用Activex和COM的Windows应用程序 | |
| Qt 3D | 支持20和3D渲染,提供用于开发近实时仿真系统的功能 | |
| Qt AndroidExtras | 提供Android平台相关的APl | |
| Qt Bluetooth | 提供访问蓝牙硬件的功能 | |
| Qt Concurrent | 提供一些类，无需使用底层的线程控制就可以编写多线程程序 | |
| Qt D-Bus | 使进程间通过D-Bus协议通信的一些类 | |
| Qt Gamepad | 使Qt应用程序支持游戏手柄硬件的使用 | |
| Qt lmageFormQt s | 支持附加图片格式的插件，包括TIFF、MNG、TGA、WBMP | |
| Qt MacExtras | 提供macOS平台相关的API | |
| Qt NFC | 提供访问NFC(近场通信)硬件的功能 | |
| Qt Positioning | 提供一些类，用于通过GPS卫星、WiFi等定位 | |
| Qt PrintSupport | 提供一些用干打印控制的类 | |
| Qt Purchasing | 提供一些类，在Qt应用程序内实现应用内购买的功能 | |
| Qt Sensors | 提供访问传感器硬件的功能，以识别运动和手势 | |
| Qt SerialBus | 访问串行工业总线的功能，目前只支持CAN和Modbus协议 | |
| Qt SVG | 提供显示SVG图片文件的类 | |
| Qt Webchannd | 用于实现服务器端（QML或C+应用程序）与客户端（HTML/lavaScript或QML应用程序)之间的P2P通信 | |
| Qt WebEngine | 提供类和函数，实现在应用程序中嵌入网页内容 | |
| Qt Websocket | 提供兼容于RFC6455的Websocket通信，是实现客户端程序与远端主机进行双向通信的基于Web的协议 | |
| Qt WindowsExtras | 提供Windows平台相关的API | |
| Qt XML | 该模块不再维护了，应使用QtCore中的QXmlStreamReader和QtXML QXml Stream Writer提供对XPath、XQuery、XSLT和XML等的支持 | |
| Qt Charts | 用于数据显示的二维图表组件 | |
| Qt Data Visualization | 用于3D数据可视化显示的界面组件 | |
| Qt Virtual Keyboard | 实现不同输入法的虚拟键盘框架 | |
| Qt附加模块可以实现一些特定目的。可能只在某些开发平台上有，或只能用于某些操作系统，或只是为了向后兼容。 | | |
|  | | |
| Qt的增值模块 | | |
| 特性 | 描述 | |
| Qt forDevice | 高效、易用、全集成的嵌入式设备应用程序开发工具，包括很多其Creation他增值特性Qt | |
| Qt Qauck Compiler | 编译.qml源文件生成二进制应用程序的编译器，提高载入时间和代 | |
| 这些模块只在商业版许可的Qt里才有 | | |
|  | | |
| 技术预览模块 | | |
| 模块 | | 描述 |
| Qt Network Authorization | | 基于QAuth协议，为应用程序提供网络账号验证的功能 |
| Qt Speech | | 提供文字转语音（text-to-speech)功能支持 |
| Qt Remote Objects | | 进程间或设备间通信，共享QObject的APl |
| 就是一些还处于开发和测试阶段的模块，一般技术预览模块经过几个版本的发布后会变成正式的模块 | | |

# 第4章 常用界面设计组件

## QString常用用法

(1）使用QString的sprintf)函数 例如：

QString str；

str.sprintf("%s%.1f%%”，"Value"，100.0)；

结果str被赋值为“Value 100.0%”。

(2）使用arg（） 例如：

QString str；

str=QString("%1，%2"）.arg("aa").arg("bb")；

结果str为“aa,bb”

数字转换为字符串

（1）使用静态函数QString:number() 例如：

str=Qstring::number(38.5)；

（2）使用setNum（）函数

str.setNum(38.5）；

字符串转换为数字 使用的是tolnt(），toLongLong(），toDouble(）...等等。

例如：

QString str="12"； int i=str.toInt()；

(1） mid()函数返回一个给定起始位置（第一个参数）和长度（第二个参数）的字串。 例如：

QString str = "white man"; QString str2 = str.mid(6, 2);

结果str2为“ma”

（2）如果省略第二个参数，mid()返回从指定的起始位置到这个字符串结尾的子串。

QString str = "white man"; QString str2 = str.mid(6);

结果str2为“man”

（3）left()函数返回最前面的n个字符

QString str = "white man"; QString str2 = str.left(5);

结果str2为“white”

（4）right()函数返回最后面的n个字符

QString str = "white man"; QString str2 = str.right(3);

结果str2为“man”

判断字符串是否包含

(5)如果我们希望找出一个字符串是否包含某个字符，字符串，或者正则表达式，我们可以使用QString的indexOf()函数。

QString str = "white man";

int i = str.indexOf("man");

结果，i为6，若不存在为-1。

(6)判断字符串是否以某物开始或者结束

（1）以某物开始，使用startsWith()函数

QString str = "http:www.baidu.com";int i = str.startsWith("http:");

结果为i = 1；

（2）以某物结束，使用endsWith()函数

QString str = "http:www.baidu.com";int i = str.endsWith("com");

结果为i = 1；

(7)字符串替换

（1）用一个字符串替换另一个字符串的某一部分，可以用replace()函数

QString str = "white man";str.replace(0 , 5, "Black");

结果str为“Black man”

（2）删除从位置n开始的m个字符，使用remove(n,m)函数

QString str = "white man";str.remove(6,3);

结果str为“white”

（3）在位置n插入某个字符串，使用insert(int, QString)函数

QString str = "white man";str.insert(6,"strong ");

结果str为“white strong man”

(8)分割字符串

一个字符串能被分成为一个装有子串的QStringList，通过使用QString::split()

QString str = "white black yellow"; QStringList color = str.split(" ");

结果color.at(1)为“white”，color.at(2)为“black”，color.at(3)为“yellow”

判断字符串是否为空

调用isEmpty()或检查length()是否为0就可以达到目的。

if(str.isEmpty()){}if(str.length() == 0){}

(9)const char \* 与 QString 互转

（1）隐式转换（自动） 例如：

QString str = "( 121 )";

（2）显式转换

简单地使用一个QString cast，或者调用fromAscii()或fromLatin1()函数

QString 转为 const char \*

使用toAscii()或toLatin1()。 这些函数返回一个QByteArray，它能被转换为一个const char \*，通过使用QByteArray::data()或QByteArray::constData()。

例如：

QString = "(1178)"str.toAscii().data();

## SpinBox的使用

QSpinBox和QDoubleSpinBox都是QAbstractSpinBox的子类

|  |  |
| --- | --- |
| 属性名称 | 描述 |
| prefix | 数字显示的前缀，例如“$” |
| suffix | 数字显示的后缀，例如“Kg” |
| minimum | 数值范围的最小值，如0 |
| maximum | 数值范围的最大值，如255 |
| singlestep | 单击右侧上下调整按钮时的单步改变值，如设置为1，或0.1 |
| value | 当前显示的值 |
| displaylntegerBase | QSpinBox特有属性，显示整数使用的进制，例如2就表示二进制 |
| decimals | QDoubleSpinBox特有属性，显示数值的小数位数，例如2就显示两位小数 |

## 其他数字输入和显示组件

QSlider:滑动条，通过滑动设置数值，可用于数值输入。

QScrollBar:卷滚条，与aSlider功能类似，还可以用于卷滚区域域

QProgressBar:进度条，一般用于显示任务进度，可用于数值的百分比显示

QDial:表盘式数值输入组件，通过转动表针获取输入值

QLCDNumber:模仿LCD数字的显示组件，可以显示整数或浮点数，显示整数可以使用不同进制数值



## 时间日期与定时器

QTime：时间数据类型，如15:23:13。

QDate：日期数据类型，如2017-4-5。

QDateTime：日期时间数据类型，如2017-03-2308：12:43。

QDateEdit：编辑和显示日期的组件。

QDateTimeEdit:编辑和显示日期时间的组件。

QCalendarWidget:一个用日历形式选择日期的组件。

QTimer：定时器，如果周期为100，那么每100毫秒会发射一次timeout信号

|  |  |
| --- | --- |
| 字符 | 意义 |
| d | 天，不补零显示，1-31 |
| dd | 天，补零显示，01-31 |
| M | 月，不补零显示，1-12 |
| MM | 月，补零显示，01-12yy年，两位显示，00-99yyyy年，4位数字显示，如2016 |
| h | 小时，不补零，0-23或1-12（如果显示AM/PM) |
| hh | 小时，补零2位显示，00-23或01-12（如果显示AM/PM) |
| H | 小时，不补零，0-23（即使显示AM/PM) |
| HH | 小时，补零显示，00-23（即使显示AM/PM) |
| m | 分钟，不补零，0-59 |
| mm | 分钟，补零显示，00-59 |
| z | 毫秒，不补零，0-999 |
| zzz | 毫秒，补零3位显示，000-999 |
| AP或A | 使用AM/pm显示 |
| ap或a | 使用am/pm显示 |

## QComBox和QPlain TextEdit

QPlain TextEdit的文字内容以QTextDocument类型储存，函数document返回这个文档对象的指针。

QTextDocument是内存中的文本对象，以文本块的方式储存，每个段落以换行符结束QTextDocument提供一些函数实现对文本内容的存取

·int blockCount()，返回文本块个数

·QTextBlock findBlockByNumber(int),读取一个文本块，序号从0开始

## QListWidget和QToolButton

Qt中用于项（ltem)处理的组件有两类：

·ltem Views：包括QListView、 QTreeView、QTableView、 QColumnView等；

·ltem Widgets：包括QListWidget、 QTreeWidget和QTable Widget。

# 第5章 Model/View结构

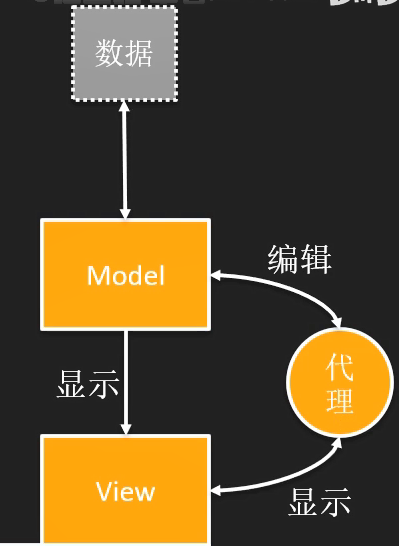
## 5.1 Model/View结构

数据：如数据库的一个数据表或SQL查询结果，内存中的一个StringList，或磁盘文件结构等。

Model:与数据通信，并为视图组件提供数据接口。

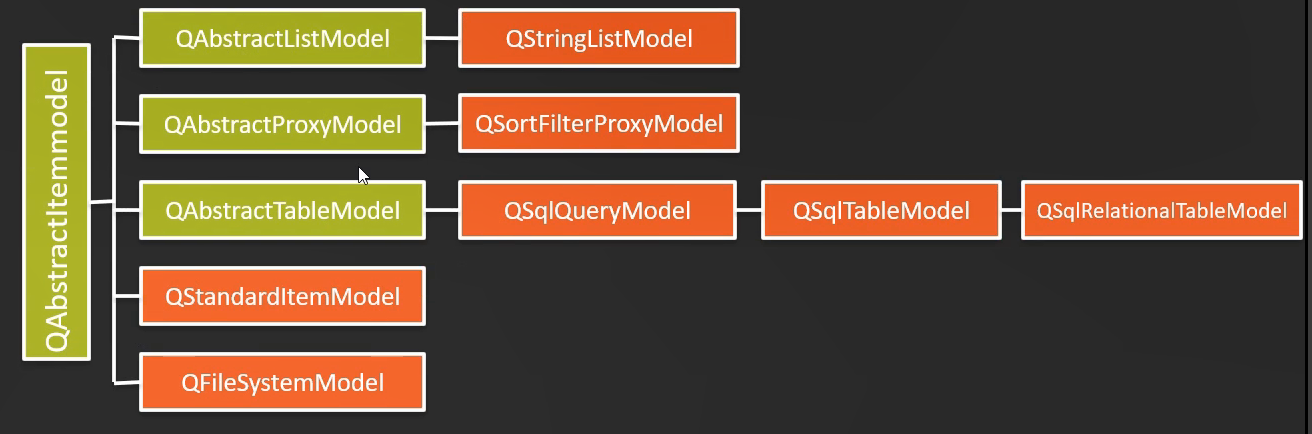
View:是屏幕上的界面组件，视图从数据模型获得每个数据项的模型索引（model index），通过模型索引获取数据

·代理：定制数据的界面显示和编辑方式。在标准的视图组件中，代理功能显示一个数据，当数据被编辑时，提供一个编辑器，一般是QLineEdit。



模型、视图和代理之间使用信号和槽通信。

|  |  |
| --- | --- |
| Model类 | 用途 |
| QStringListModel | 用于处理字符串列表数据的数据模型类 |
| QStandardltemModel | 标准的基于项数据的数据模型类，每个项数据可以是任何数据类型. |
| QFileSystemModel | 计算机上文件系统的数据模型类 |
| QSortFilterProxyModel | 与其他数据模型结合，提供排序和过滤功能的数据模型类 |
| QSqlaueryModel | 用于数据库SQL查询结果的数据模型类 |
| QSqlTableModel | 用于数据库的一个数据表的数据模型类Q |
| SqlRelationalTableModel | 用于关系型数据表的数据模型类 |



## 5.2 QFileSystemModel

如同Widnows的资源管理器一样。使用QFileSystemModel提供的接口函数，可以创建目录、删除目录、重命名目录，可以获得文件名称、目录名称、文件大小等参数，还可以获得文件的详细信息。

QFileSystemModel \*model=new QFileSystemModel;

model->setRootPath(QDir:currentPath()；

## 5.3QStringListModel

setStringList()函数可以初始化数据模型的字符串列表的内容

提供编辑和修改字符串列表数据的函数，如insertRows（）、removeRows()、setData()等

theModel->insertRow(theModel->rowCount())；∥在尾部插入一空行 //QModellndex index;

QModellndex index=theModel->index(theModel->rowCount()-1,0);/最后一行 theModel->setData(index,"new item",Qt:DisplayRole);/设置显示文字 ui->listView->setCurrentlndex(index);∥设置当前选中的行

## 5.4QStandardItemModel

以项数据为基础的标准数据模型类，通常与QTableView组合成Model/View结构，实现通用的二维数据的管理功能。

主要用到以下3个类：

·QStandardltemModel:可以处理二维数据。每个项是一个QStandardltem类的变量，用于存储项的数据、字体格式、对齐方式等。

·QTableView:一个单元格显示QStandardltemModel数据模型中的一个项。

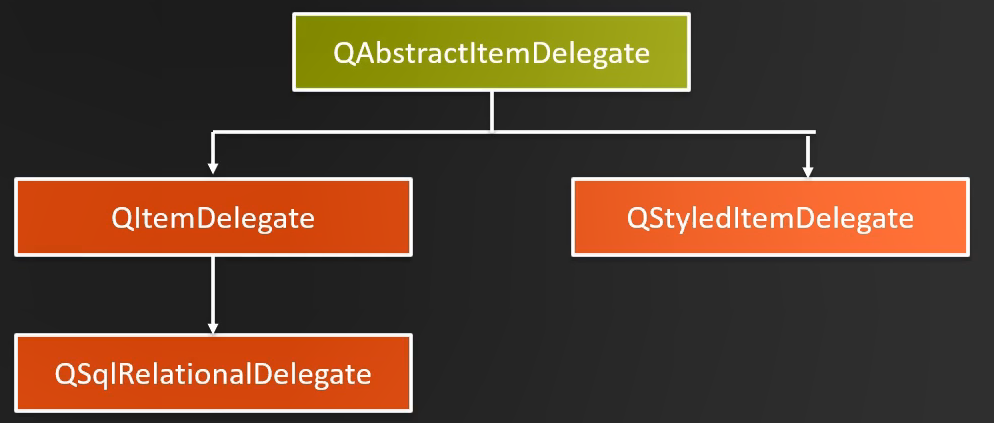
QltemSelectionModel:跟踪视图组件的单元格选择状态的类，通过QltemSelectionModel可以获得选中的单元格的模型索引。

## 5.5自定义代理

定制数据的界面显示和编辑方式。在标准的视图组件中，代理功能显示一个数据，当数据被编辑时，提供一个编辑器，一般是QLineEdit。

QAbstractltemDelegate是所有代理类的抽象基类。

QStyledltemDelegate是视图组件使用的缺省的代理类，QltemDelegate也是类似功能的类。区别在于，QStyledltemDelegate可以使用当前的样式表设置来绘制组件，因此建议使用QStyledltemDelegate。



必须实现以下几个函数：

createEditor():创建用于编辑模型数据的widget组件，如QSpinBox,QComboBox。

setEditorData()：从数据模型获取数据，供widget组件进行编辑。

setModelData()：将widget上的数据更新到数据模型。

updateEditorGeometry()：用于给widget组件设置一个合适的大小。

# 第6章 对话框与多窗体设计

## 6.1　标准对话框

Qt提供了一些常用的标准对话框，如打开文件对话框、选择颜色对话框、信息提示和确认选择对话框、标准输入对话框等

Qt预定义标准对话框：

|  |  |  |
| --- | --- | --- |
| 对话框 | 常用静态函数名称 | 函数功能 |
| QFileDialog文件对话框 | QString getOpenFileName()  QString ListgetOpenFileNames()  QString getSaveFileName()  QString getExistingDirectory()  QUrl getOpenFileUrl() | 选择打开一个文件选择打开多个文件选择保存一个文件选择一个己有的目录选择打开一个文件，可选择远程网络文 |
| QColorDialog颜色对话框 | QColor getColor() | 选择颜色 |
| QFontDialog字体对话框 | QFont getFont() | 选择字体 |
| QinputDialog输入对话框 | QString getText()  Int getlnt()  Double getDouble()  QString getltem() | 输入单行文字输入整数输入浮点数从一个下拉列表框中选择输入输入多行字符串 |

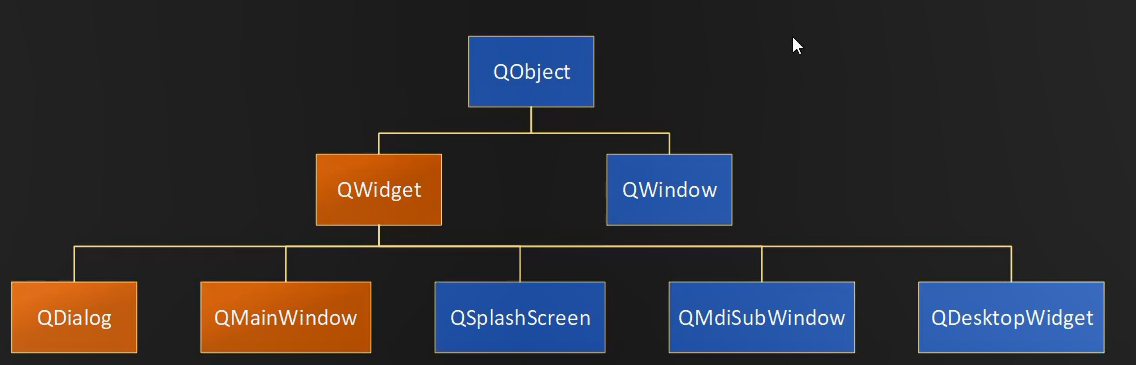
## 6.2　自定义对话框及其调用

QWDialogSize·设置表格行列数对话框

QWDialogHeaders·设置表头标题对话框

QWDialogLocate·单元格定位与文字设置对话框

## 6.3　多窗体应用程序设计



QWidget:在没有指定父容器时可作为独立的窗口，指定父容器后可以作为容器的内部组件。

QDialog:用于设计对话框，以独立窗口显示。

QMainWindow:用于设计带有菜单栏、工具栏、状态栏的主窗口，一般以独立窗口显示。

QSplashScreen:用作应用程序启动时的splash窗口，没有边框。

QMdiSubWindow:用于为QMdiArea提供一个子窗体，用于MDI（多文档）的设计。

QDesktopWidget:具有多个显卡和多个显示器的系统具有多个桌面，这个类提供用户桌面信息，如屏幕个数、每个屏幕的大小等。

·QWindow:通过底层的窗口系统表示一个窗口的类，一般作为一个父容器的嵌入式窗体，不作为独立窗体。

|  |  |
| --- | --- |
| Void QWidget:setWindowState(Qt:WindowStateswindowstate) | |
| 常量 | 意义 |
| Qt:NonModal | 无模态，不会阻止其他窗口的输入 |
| Qt::WindowModal | 窗口对于其父窗口、所有的上级父窗口都是模态 |
| Qt::ApplicationModal | 窗口对整个应用程序是模态的，阻止所有窗口的输入 |

void QWidget::setWindowOpacity(qreal level)

参数level是1.0（完全不透明）至0.0（完全透明）之间的数。窗口透明度缺省值是1.0，即完全不透明。

|  |  |
| --- | --- |
| Void QWidget::setAttribute(Qt:WidgetAttribute attribute,bool on=true) | |
| Qt:WA\_AcceptDrops | 允许窗体接收拖放来的组件 |
| Qt::WA\_DeleteOnClose | 窗体关闭时删除自己，释放内存 |
| Qt::WA\_Hover | 鼠标进入或移出窗体时产生paint事件 |
| Qt::WAAcceptTouchEvents | 窗体是否接受触屏事件 |

|  |  |  |
| --- | --- | --- |
| Void QWidget::setWindowFlags(Qt:WindowFlagstype) | | |
| 表示窗体类型的常量 | | |
| at:Widget | 这是QWidget类的缺省类型。这种类型的窗体，如果它有父窗体，就作为父窗体的子窗体；否则就作为一个独立的窗口 | |
| at:Window | 表明这个窗体是一个窗口，通常具有窗口的边框、标题栏，而不管它是否有父窗体 | |
| at:Dialog | 表明这个窗体是一个窗口，并且要显示为对话框（例如在标题栏没有最小化、最大化按钮。这是QDialog类的缺省类型 | |
| Qt:Popup | 表明这个窗体是用作弹出式菜单的窗体表明这个窗体是工具窗体，具有更小的标题栏和关闭按钮，通常作为工具栏的窗体 | |
| Qt:ToolTip | 表明这是用于Tooltip消息提示的窗体 | |
| at:SplashScreen | 表明窗体是splash屏幕，是QSplashScreen类的缺省类型 | |
| at::Desktop | 表明窗体是桌面，这是QDesktopWidget类的类型 | |
| at::SubWindow | 表明窗体是子窗体，例如QMdisubWindow就是这种类型 | |
|  |  | |
| 控制窗体显示效果的常量 | | |
| at:MSWindowsFixedSizeialogHint | | 在Wndous平台上使窗口具有更窄的边框，用于固定大小的对话框 |
| at::FramelessWindowHint | | 创建无边框窗口 |
|  | |  |
| WindowHint要定义窗体外观定制窗体外观的常量，需要先设置Qt:Customize | | |
| Qt:CustomizeWindowHlint | | 关闭缺省的窗口标题栏 |
| at::WindowTitleHint | | 窗口有标题栏 |
| at:WindowSystemMenulint | | 有窗口系统菜单 |
| at:WindowMinimizeButtonHint | | 有最小化按钮 |
| at::WindowMaximizeButtonHint | | 有最大化按钮 |
| at::WindowMinMaxButtonsHint | | 有最小化、最大化按钮 |
| Qt::WindowCloseButtonHint | | 有关闭按钮 |
| Qt:WindowContextHelpButtonHint | | 有上下文帮助按钮 |
| Qt::WindowStaysOnTopHint | | 窗口总是处于最上层 |
| Qt:WindowStaysOnBottomHint | | 窗口总是处于最下层 |
| at:WindowTransparentForlnput | | 窗口只作为输出，不接受输入 |

## 6.4　MDI应用程序设计

QMdiArea（Multiple Document Interface Area)提供了一个可以同时显示多个文档窗口的区域。

区域本身是一个框架，每一个窗口都是是一个QMdisubWindow对象

设置MDI视图模式用setViewMode（)函数，有两种模式可以选择：

1.QMdiArea:SubWindowView是传统的子窗口模式

2.QMdiArea::TabbedView是多页的显示模式

## 6.5　Splash与登录窗口

Qt有一个QSplashScreen类可以实现Splash窗口的功能，它提供了载入图片，自动设置窗口无边框效果等功能。

1.event->globalPos()获取的鼠标位置是鼠标偏离电脑屏幕左上角的位置；

2.pos()获取的位置是主窗口（widget窗口）左上角（边框的左上角，外左上角）相对于电脑屏幕的左上角的偏移位置

move(event->globalPos()-m\_lastPos)；

m\_lastPos=event->globalPos()-pos()；

# 第7章 文件系统和文件读写

## 7.1　文本文件读写

|  |  |
| --- | --- |
| 枚举值 | 描述 |
| QIODevice::ReadOnly | 以只读方式打开 |
| QloDevice::WriteOnly | 以只写方式打开 |
| QloDevice:ReadWrite | 以读写方式打开 |
| QloDevice::Append | 新增加的内容将被追加到文件末尾 |
| QloDevice::Truncate | 以重写的方式打开，原有内容会被删除 |
| QIODevice::Text | 在读取时，将行结束符转换成\n;在写入时，将行结束符转换成本地格式，例如Win32平台上是\r\n |

//解决汉字乱码问题

1.在main中添加

#include <QTextCodec>

QTextCodec \*codec = QTextCodec::codecForName("UTF-8");

QTextCodec::setCodecForLocale(codec); //解决汉字乱码问题

QTextCodec::setCodecForTr(codec);  
    QTextCodec::setCodecForCStrings(codec);

1.2

QTextCodec \*codec = QTextCodec::codecForName("UTF-8");

QTextCodec::setCodecForTr(codec);

QTextCodec::setCodecForLocale(QTextCodec::codecForLocale());  
  QTextCodec::setCodecForCStrings(QTextCodec::codecForLocale());

Qt5.9以后的版本移除了setCodecForCStrings()和 codecForCStrings()

2.msvc编译器在pro中添加

msvc {

QMAKE\_CFLAGS += /utf-8

QMAKE\_CXXFLAGS += /utf-8

}

msvc10好像不行

1. Qt5.9.9调试C++程序出现输出中文乱码情况，解决方案：将程序文件转成utf8+BOM编码后，再重新导入qt,问题解决，供大家参考。转成utf8+BOM编码可通过Notepad++手动转码或在vs上安装ForceUTF8 (with BOM)插件，安装后重新导入项目文件，生成项目自动转码成功

## 7.2　二进制文件读写

使用QFile和QDataStream进行二进制数据文件的读写

·QFile负责文件的IO设备接口，即与文件的物理交互；

·QDataStream 以数据流的方式读取文件内容或写入文件内容。

·Qt预定义编码

·标准编码

## 7.3　文件目录操作

文件目录操作相关类：

QCoreApplication:用于提取应用程序路径、程序名等文件信息

QFile:除了打开文件操作外，还有复制文件、删除文件等功能

QFilelnfo:用于提取文件信息，包括路径、文件名、后缀

QDir:用于提取目录或文件信息，获取一个目录下的文件或目录列表，创建或删除目录和文件，文件重名等操作

QTemporaryDir 和QTemporaryFile:用于创建临时目录和临时文件。

QFileSystemWatcher:监听目录下文件的添加、删除等变化

# 第8章 绘图

## 8.1　QPainter基本绘图

Qt可以使用相同的API在屏幕和绘图设备上进行绘制，它主要基于QPainter、QPaintDevice和QPaintEngine 这三个类。

QPainter:用来进行绘图操作的类

QPaintDevice:抽象的二维界面，是中间媒介

QPaintEngine:提供了一些接口，可用于QPainter在不同的设备上进行绘制。

QPainter 可以在继承自QPaintDevice类的任何对象上进行绘制操作

QPainter一般在部件的绘图事件paintEvent()中进行绘制。当窗口程序需要升级或者重新绘制时，调用此成员函数。使用repaint（）和update（）后，调用函数paintEvent()。

QPainter的3个主要设置是：pen,brush,font

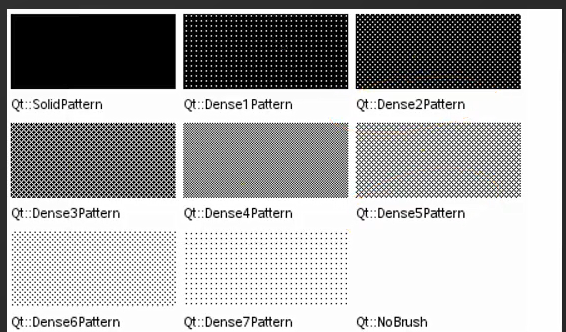
painter.setPen(QPen(..))；

painter.setBruch(QBrush(..))；

painter.setFont(QFont(..))；

painter.setRenderHint(QPainter::Antialiasing)；

设置了bursh之后，画的是效果即填充 QPainter的属性影响绘制的图形 通常使用painter.drawXXX（.）来绘制图形



## 8.2　坐标系统和坐标变换

### 8.2.1坐标系统和坐标变换

为了绘图的方便，QPainter提供了一些坐标变换的功能，通过平移，旋转，缩放等坐标变换，得到一个逻辑坐标系统，使用逻辑坐标系统在某些时候绘图更方便。

QPainter有关坐标变换操作的函数：

|  |  |  |
| --- | --- | --- |
| 分组 | 函数原型 | 功能 |
| 坐标变换 | voidtranslate(qrealdx,qrealdy)  voidrotate(qrealangle)  voidscale(qrealsx,qrealsy)  voidshear(qraelsh,qrealsy) | 坐标系统一定的偏移量，坐标原点平移到新的点  坐标系统顺时针旋转一个角度  坐标系统缩放  坐标系统做扭转变换 |
| 状态保存与恢复 | voidsave()  voidrestore()  voidresetTransform() | 保存painter当前的状态，就是将当前状态压入栈  恢复上一次状态，就是从堆栈中弹出上次的状态  复位所有的坐标变换 |

### 8.2.1视口和窗口

视口：绘图设备的任意一个矩形区域的物理坐标，可以只选取物理坐标的一个矩形区域用于绘图。视口默认情况下等于绘图设备的整个矩形区。

窗口：对应于视口的矩形区域，只不过是用逻辑坐标定义的坐标系，窗口坐标的中心在矩形中心。

使用窗口坐标绘制，不用管实际的物理大小|

## 8.3　GraphicsView绘图架构

Graphics View绘图架构

QPainter不能实现图件的选择、编辑、拖放、修改等功能。

QGraphicScene(场景）：可以管理多个图形项

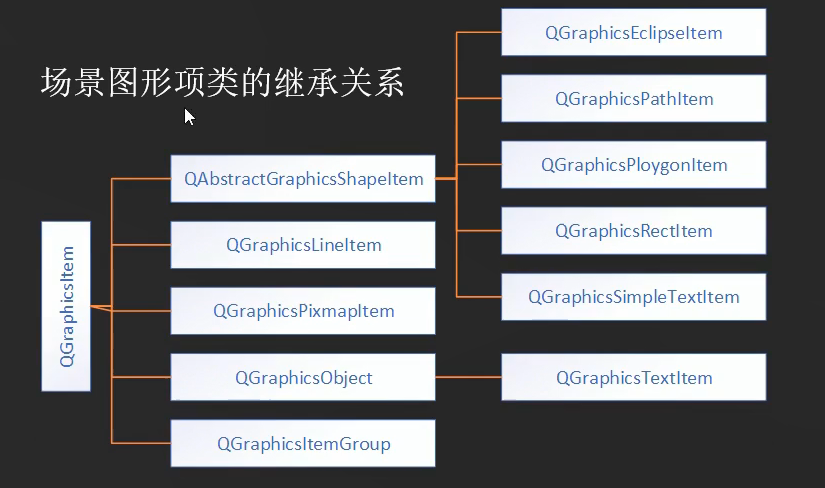
QGraphicsltem(图形项）：也就是图元，支持鼠标事件响应。比如：矩形的图形项QGraphicsView(视图）：关联场景可以让场景中的所有图形项可视化

视图坐标：坐标原点窗口左上角

场景坐标、图形项坐标：坐标原点窗口中心

QGraphicsView是Qt的图形视图组件，在ui设计器的Display Widgets分组。

QGraphicsView没有与mouseMoveEvent()相关的信号，需要继承自定义一个派生类



# 第9章 Qt Charts

## 9.1　Qt Charts概述

Qt图表模块提供如下类型：

·折线图

·样条曲线图

·面积图

·散点图

·条形图

·饼图

·方块胡须图

·蜡烛图

·极坐标图

Qt Charts基于Qt的Graphics View架构，其核心组件是QChartView和QChart

·QChartView是显示图标的视图，基类为QGraphicsView

·QChart的基类是QGraphicsltem

要在项目中使用Qt Charts模块，必须在项目的配置文件.pro文件中增加下面的一行语句：

QT+=charts

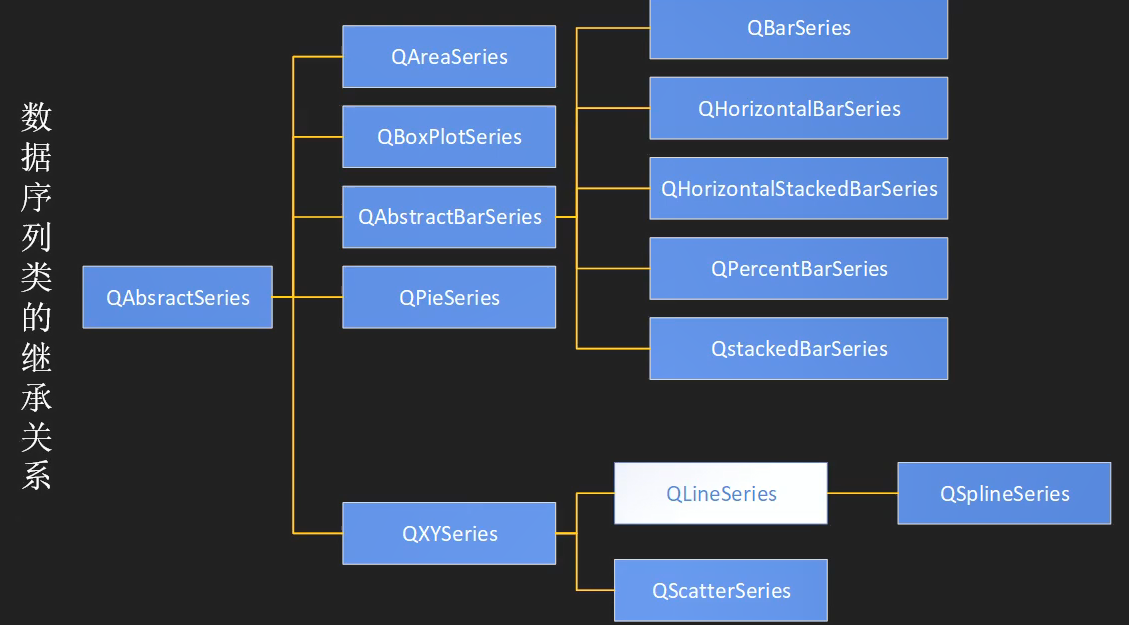
在需要使用QtCharts的类的头文件或源程序文件中，要使用如下的包含语句：

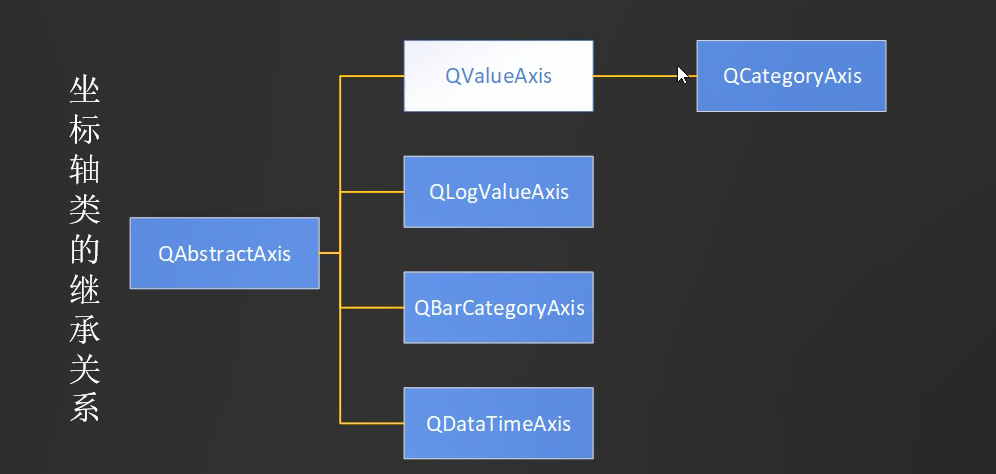
#include <QtCharts>

//using namespace QtCharts;

QT CHARTS USE NAMESPACE/使用宏定义

## 9.2　QChart绘制折线图





## 9.3　各种常见图表的绘制

柱状图相关的主要类包括：

QBarSet:数据集

·例如：数学、语文、英文，每个科目有一个数据集

QBarSeries:一个序列可以包含多个QBarSet

·例如：包含所有同学的各科成绩。

QBarCategoryAxis:

分类坐标轴

平均分采用的是QLineSeries，需要点数据（QPointF）

柱状图的一种变体

堆叠柱状图相关的主要类包括：

·QBarSet:数据集

·QStackedBarSeries:一个序列可以包含多个QBarSet

·QBarCategoryAxis:分类坐标轴

散点图和光滑曲线图

相关的主要类包括：

QSplineSeries、QLineSeries:

QValueAxis:分类坐标轴

## 9.4　图表的其他操作

·鼠标移动时，状态栏显示坐标值

·图例赋予类似QCheckBox的功能

·通过按钮或按键进行缩放操作

·通过键盘移动序列的显式位置

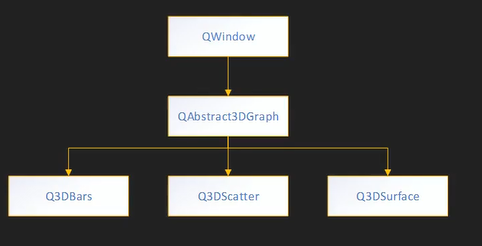
为了支持上面的功能需要自定义一个QChartView的派生类

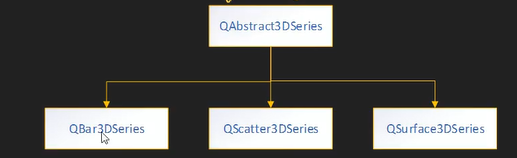
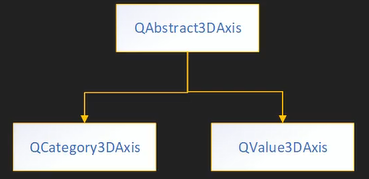
# 第10章 Data Visualization

## 10.1　DataVisualization模块概述

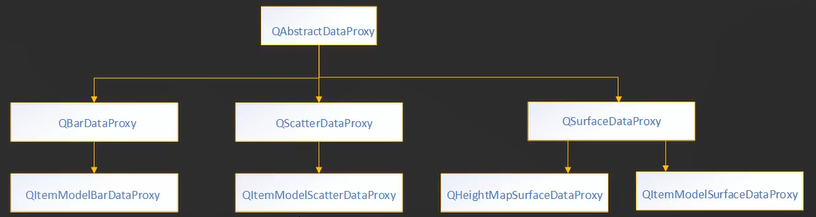
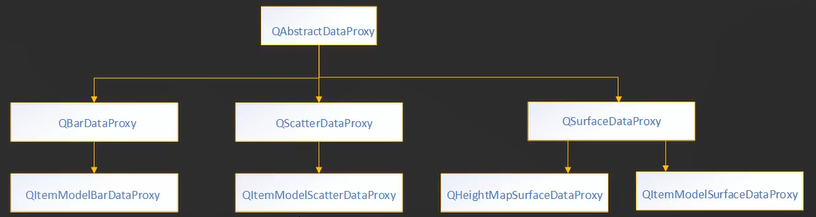
Data Visualization的三维显示功能主要由三种三维图形来实现，分别是三维柱状图Q3DBars，三维空间散点Q3DScatter，三维曲面Q3DSurface。这三个类的父类都是QAbstract3DGraph，从QWindow继承而来。

Data Visualization与Qt Charts类似都是基于Qt Graphics View的图形视图结构

Q3DBars、Q3DScatter、Q3DSurface相当于Qt Charts中的QChart。每一种三维图表对应一种三维序列

Data Visualization模块中还有数据代理（Data Proxies)，数据代理类就是与序列对应，用于存储序列的数据的类。

 图形不一样，存储的数据结构也不一样

在使用时，需要在Qt工程.pro文件中加入：

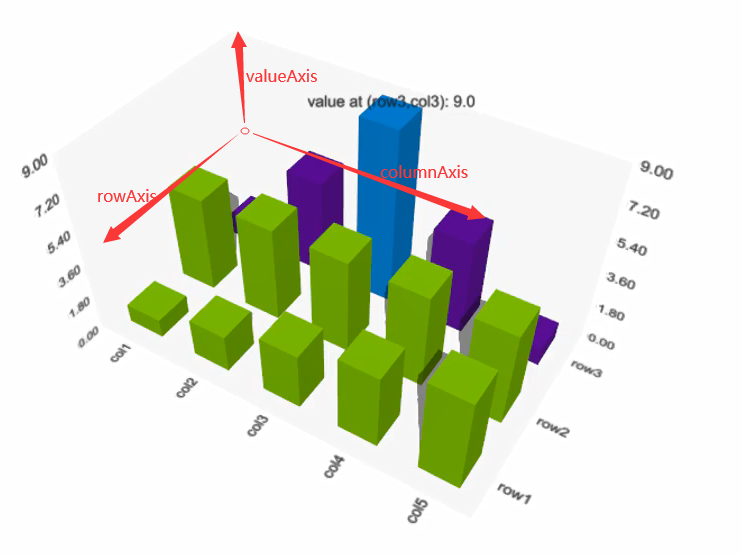
QT+=datavisualization 在调用时源码中添加：

#include <QtDataVisualization>

using namespace QtDataVisualization;

## 10.2　三维柱状图

当主题为Retro时无法改变项的颜色



三维图表类都是从QWindow继承而来，需要使用QWidget：:createWindowContainer()动态创建QWidget作为三维图表的容器。

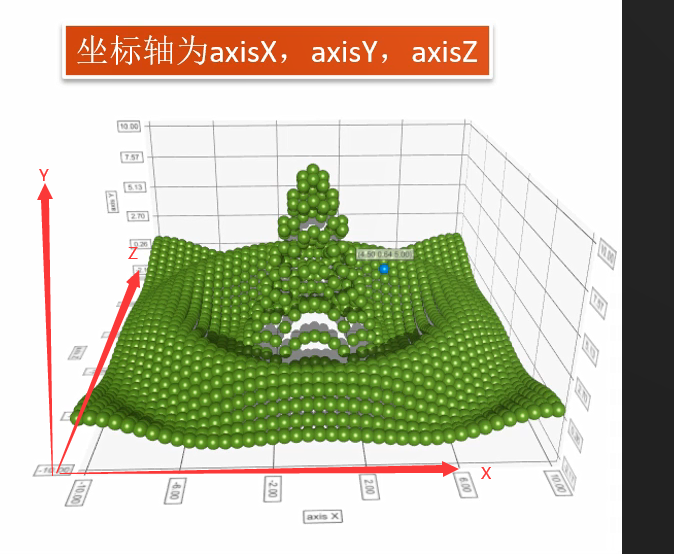
namespace QtDataVisualization{

typedef QVector<QBarDataltem>QBarDataRow;

typedef QList<QBarDataRow \*> QBarDataArray;

}

## 10.3　三维散点图



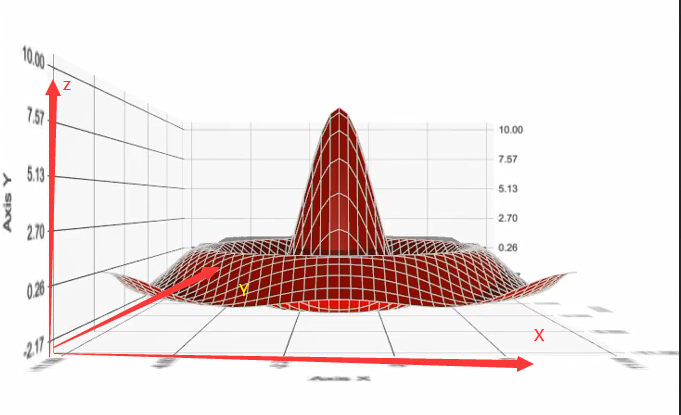
namespace QtDataVisualization{

typedef QVector<QScatterDataltem>QScatterDataArray;

}

## 10.4　三维曲面绘图

### 3D曲面图

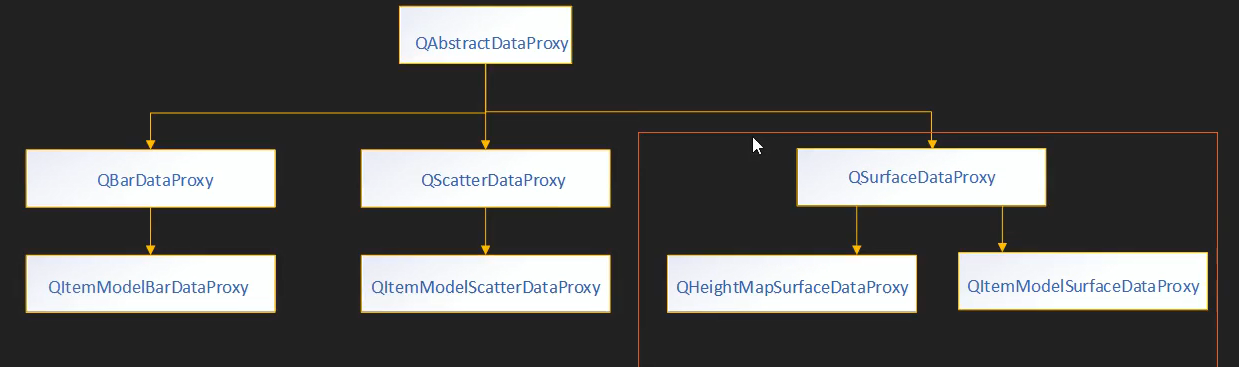


使用QSurfaceDataProxy

namespace QtDataVisualization{

typedef QVector<QSurfaceDataltem>QSurfaceDataRow;

typedef QList-QSurfaceDataRow \*>QSurfaceDataArray;

}

enum ColorStyle{

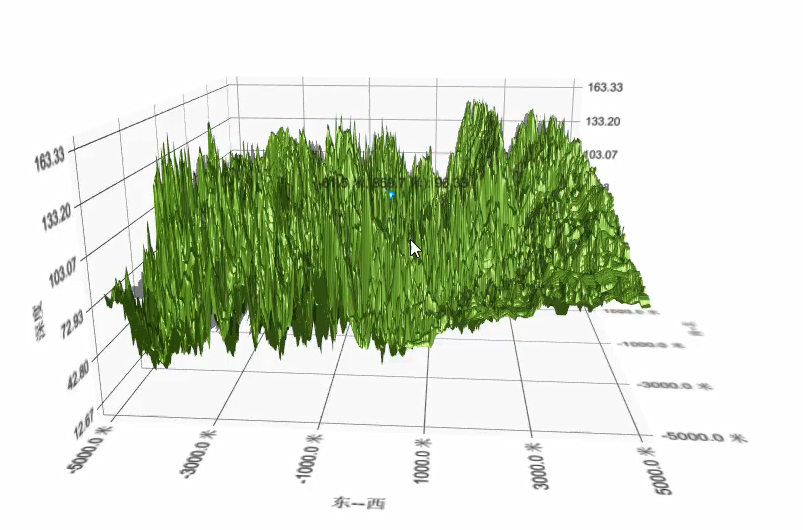
ColorStyleUniform=0,/单一颜色

ColorStyleObjectGradient,/不考虑高度渐变

ColorStyleRangeGradient /按高度渐变

}

### 3D地形图

代理为QHeightMapSurfaceDataProxy

# 第11章 数据库

## 11.1　Qt SQL模块概述

Qt SQL模块概述要包含模块类的定义，请使用以下指令：

#include <QtSql>要链接模块，

qmake.pro文件需要添加：

QT += sql

Qt SQL模块包含多个类，实现数据库的连接，SQL语句执行，数据获取与界面显示，数据与界面之间使用Model/View结构，方便的实现数据的显示和操作。

QTableView是常用的数据库内容显示组件：

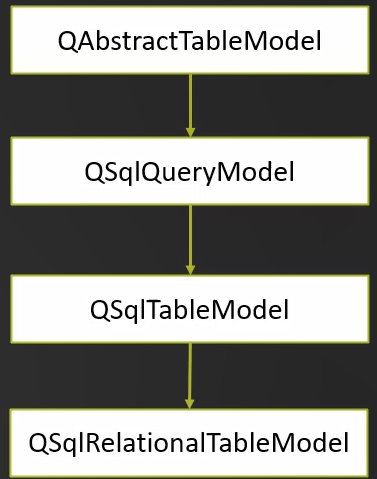
QSqlQueryModel:

通过设置SELECT语句查询获取数据库内容，但是数据是只读的。

QSqlTableModel:

直接设置一个数据表的名称，可以获取数据表的全部记录，可编辑。

QSqlRelationalTableModel:

为单张的数据库表提供了一个可编辑的数据模型，它支持外键

## 11.2　QSqlTableModel的使用

只可以显示一个指定的数据库表可在view上设置后进行增删改

## 11.3　QSqlQueryModel的使用

QSqlQueryModel是QSqlTableModel的父类

QSqlQueryModel封装了指向SELECT语句从数据库查询数据的功能

QSqlQueryModel只能读取数据，不能编辑

## 11.4　QSqlQuery的使用

QSqlQuery是能执行任意SQL语句的类，如SELECT、INSERT、UPDATE、DELETE等。

## 11.5　QSqlRelationalTableModel的使用

该类为单张的数据库表提供了一个可编辑的数据模型，它支持外键。

tableView中为外键提供了内置的代理（编辑状态为下拉菜单）

# 第12章 自定义插件和库

## 12.1　自定义Widget组件

1.采用提升法（promotion)

2·重新定义paintEvent事件

·颜色控制：

背景颜色

边框颜色

电量柱的颜色

电量短缺时电量柱的颜色

视口：绘图设备的任意一个矩形区域的物理坐标，可以只选取物理坐标的一个矩形区域用于绘图。视口默认情况下等于绘图设备的整个矩形区。

窗口：对应于视口的矩形区域，只不过是用逻辑坐标定义的坐标系，窗口坐标的中心在矩形中心。

使用窗口坐标绘制，不用管实际的物理大小

## 12.2　自定义QtDesigner插件

生成了DLL，则肯定也生成LIB文件。

源代码编译和链接，有头文件和lb就够了

·h头文件是编译时必须的

·lib是链接时需要的

程序运行起来，有DLL就够了

·dll是运行时需要的

使用Qt创建的Widget插件，编译器版本需要与Qt Creator版本一致

H文件：声明函数接口

DLL文件：函数可执行代码

当程序中使用了一个h文件里的函数，编译器怎么知道如何从DLL文件中获取呢？

LIB文件：告诉编译器，调用的函数在哪个DLL中，函数执行代码在DLL中的具体什么位置。

使用自定义插件：

在项目管理器中，右击项目名称

将编译好的DLL文件放入一下目录：

:\Qt\Qt5.9.1\Tools\QtCreator\bin\plugins\designer

:\Qt\Qt5.9.1\5.9.1\msvc2015\plugins\designer

部署插件。

编译完成后，在输出的目录下，找到对应的 \*.dl 和 \*.lib两个文件。分别拷贝到QT库的designer路径下和QT的IDE工具的designer路径下。。

以我本地5.7.0路径为例：（具体以QT安装路径为准）。

QT库的designer库路径：。

1.C:\Qt\Qt5.9.0Tools\QtCreator\bin\plugins\designer.

QT库的IDE路径：

2.C:\Qt\Qt5.9.015.9mingw53\_32\plugins\designer.

测试使用插件。

新建一个工程，在工程使用该Test自定义的插件。，

1.打开\*.ui,在左下角会出现我们前面编译的自定义插件。

2.在工程所在的文件目录中(pro同级目录)，新建include文件夹，将前面的插件头文件test.h放到该include文件夹中，新建lib文件夹，将前面生成的testplugin.lib静态库放到此lib文件夹。…

3.打开测试工程的.pro文件。添加相应的include路径和lib路径。

LIBS+=$$PWD/lib/\*.lib.

INCLUDEPATH += $$PWD/include

4.切换到 release编译模式，点击qmake，生成相应的makefile文件，并将前面的testplugin.dll 动态库添加到release输出目录中。

5.点击构建，此时会生成“\*.exe到release输出目录。此时，直接运行。

## 12.3　创建和使用静态链接库

创建一个静态库项目，设计各种需要导出的类：

·MSVC编译器生成后缀为".lib"

·MinGW编译器生成后缀为".a"

使用一个静态库：

·需要使用静态库文件和h文件

·不需要cpp文件

·release和debug模式下编译生成的都是相同的文件名

·但使用时，需要区分release还是debug版本

·需要手动修改以便区分（通常会在debug版本的文件名后添加一个‘d）

## 12.4　创建和使用共享库

#if defined(DLL\_LIBRARY)

#define DLLSHARED\_EXPORT Q DECL EXPORT

#else

#define DLLSHARED EXPORT Q DECL IMPORT

#endif

.h文件将会有两种不同的角色

·生成DLL

·生成exe

显式链接调用共享库：编译时无需共享库的任何文件

Delphi编写了一个DLL项目，里面有一个triple函数：

function triple(N:integer):interger;

在QT中，单击按钮时才加载DLL文件，使用DLL文件计算，并将结果显示在界面上。

typedef int(\*FunDef)(int);//函数原定定义

FunDef my Triple =(FunDef)myLib.resolve("triple")；/解析DLL中的函数

int V=my Triple(ui->spinlnput->value())；//调用函数

# 第13章 多线程

## 13.1　QThread创建多线程程序

对于只有一个线程的程序，操作是顺序执行的，如果有某个比较耗时的计算或操作，容易造成堵塞。

加载文件1|加载文件2|查看文件1（需要翻页等）

QThread类提供管理线程的方法：

一个对象管理一个线程·

一般从QThread继承一个自定义类，重载run函数

## 13.2　线程同步

### 基于互斥量的线程同步

假设程序有两个线程，一个负责打印订单，一个负责计算订单（每个订单有很多分项）：

X：订单号 Y：订单总额

当答复时，X或Y正在计算过程中，并且计算过程中会产生很多中间值，会发生什么？

QMutex和QMutexLocker是基于互斥量的线程同步类，QMutex定义的实例是互斥量，主要提供|三个函数：

·lock（)：锁定互斥量，如果另一个线程锁定了这个互斥量，将阻塞直到另一个解锁；

·unlock()：解锁一个互斥量

·tryLock()：试图锁定一个互斥量，如果成功返回true；失败（其他线程已经锁定了这个互斥量）返回false，不阻塞线程。

QMutexLocker简化了互斥量处理：

·构造函数接受一个互斥量作为参数，并将其锁定

·析构函数解锁该互斥量

### 基于QReadWriteLock的线程同步

·使用互斥量时存在一个问题，每次只能有一个线程获得互斥量的权限。

·如果多个线程读取某个变量，就会出现排队的现象。

·而实际上，应该允许让多个线程同时读取，互斥量会降低程序的性能。

QReadWriteLock提供以下几个主要的函数：

·lockForRead():只读方式锁定资源，如果有其他线程以写入方式锁定，这个函数会阻塞

·lockForWrite()：以写入方式锁定资源，如果本线程或其他线程以读或写模式锁定资源，则函数堵塞

·unlock()：解锁

·tryLockForRead():是lockForRead的非阻塞版本

·tryLockForWrite()：是lockForWrite的非阻塞版本

### 基于QWaitCondition的线程同步

痛点在一个线程解锁资源后，不能及时通知其他线程。

(写完了，立即通知其他线程来读）|

QwaitCondition提供如下一些函数：

·wait(QMutex \*lockedMutex):进入等待状态，解锁互斥量lockedMutex，，被唤醒后锁定lockedMutex并退出函数

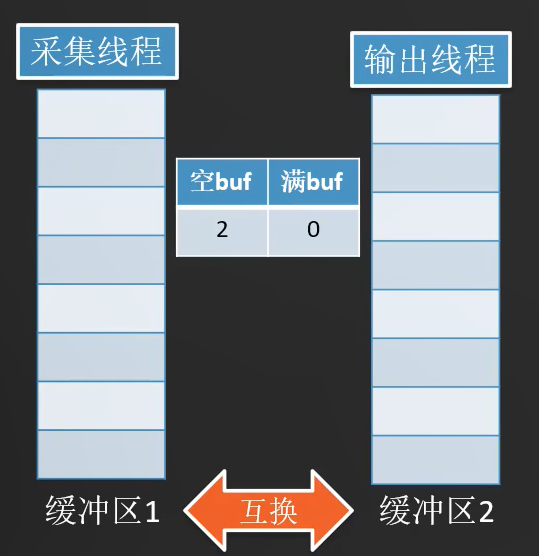
·wakeAll():唤醒所有处于等待状态的线程，线程唤醒的顺序不确定，由操作系统的调度策略决定

·wakeOne():唤醒一个处于等待状态的线程，唤醒哪个线程不确定，由操作系统的调度策略

QWaitCondition一般用于“生产者/消费者”（producer/consumer)|模型中。“生产者”产生数据，“消费者”使用数据

### 基于信号量的线程同步

信号量（Semaphore)通常用于保护一定数量的相同的资源，例如数Data Accuisition（数据采集）时的双缓冲区。



Semaphore是实现信号量功能的类，提供以下函数：

acquire(int n):尝试获得n个资源，如果不够将堵塞线程，直到n个资源可用；

release(int n):释放资源，如果资源已经全部可用，则可扩充资源总数；

int available():返回当前信号量的资源个数；·

bool tryAcquire(int n=1):尝试获取n个资源，不成功时，不阻塞线程；

# 第14章 网络编程

## 14.1　主机信息查询

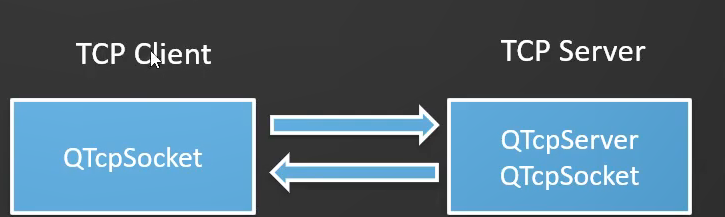
QHostlnfo: 获取主机名和IP地址

QNetWorkInterface: 获取主机的所有网络接口，包括子网掩码和广播地址等

QNetworkInterface::allAddresses()的功能与QHostlnfo：:allAddresses()函数 功能相似，指示QNetworkInterface返回地址更多（包括127.0.0.1）

## 14.2　TCP通信

TCP是一种被大多数Internet网络协议（如HTTP和FTP）用于数据传输的低级网络协议，是可靠的、面向流、面向连接的传输协议。

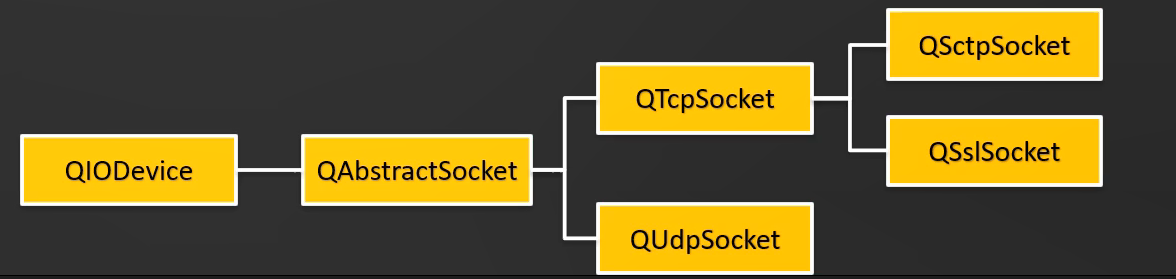


使用 QTcpServer用于端口监听

建立TCP连接

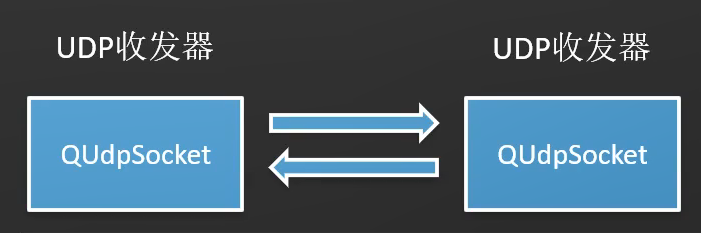
使用 QTcp Socket用于通信

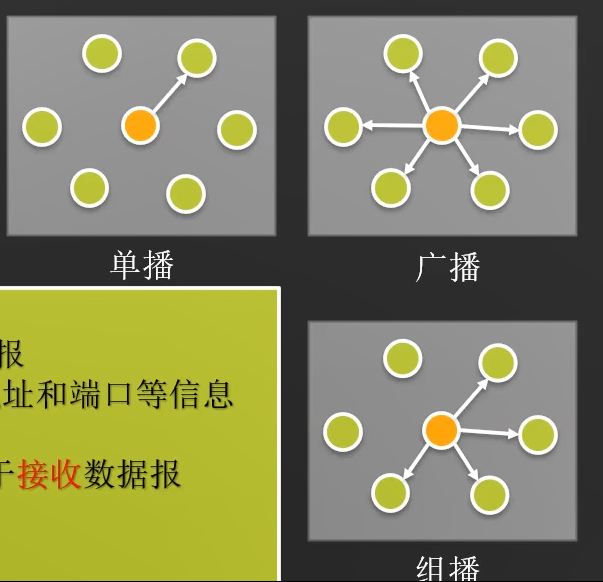
断开连接



## 14.3　QUdpSocket实现UDP通信

UDP( User Datagram Protocol,用户数据报协议)是轻量的、不可靠的、面向数据报( datagram、无连接的协议





UDP数据发送:

使用 QUdp Socket: write Datagram()发送数据报

每个数据报包含发送者和接受者的P地址和端口等信

UDP数据接收:

使用 QUdp Socket:bind()绑定一个端口,用于接收数据报

当有数据传入时,会发射 ready Read()信号

使用 readDatagram()函数来读取数据报

### QUdpSocket实现UDP组播

当多个客户端加入由一个组播地址定义的多播组之后,客户端向组播地址和端口发送∪DP数据报,组成成员都可以接收到,其功能类似于QQ群聊

D类地址的前四位是Ⅲ0,因此D类地址的范围是224.0.0.0到239.255.255.255

224.0.0.1~224.0.0.255为预留的组播地址(永久组地址),地址224.0.0.0保留不做分配,其他地址供路由协议使用;

224.0.1.0~224.0.1.255是公用组播地址,可以用于 Internet

224.0.2.0~238.255.255.255为用户可用的组播地址(临时组地址),全网范围内有效

239.0.0.0~239.255.255.255为本地管理组播地址,仅在特定的本地范围内有效

## 14.4　基于HTTP协议的网络应用程序

七层模型:OSI( Open System Interconnection)

应用层

网络服务与最终用户的一个接口。

协议有: Http Ftp Tetp Smtp Snmp dnS TelnEt Https Pop3DHcP

表示层

数据的表示、安全、压缩。(在五层模型里面已经合并到了应用层)

格式有:JPEG、 ASCI EBCDIC、加密格式等

会话层

建立、管理、终止会话。(在五层模型里面已经合并到了应用层)

对应主机进程,指本地主机与远程主机正在进行的会话

传输层

定义传输数据的协议端口号,以及流控和差错校验

协议有: TCP UDP,数据包一旦离开网卡即进入网络传输层

网络层

进行逻辑地址寻址,实现不同网络之间的路径选择

协议有: ICMP IGMP|P(PV4|PV6)

数据链路层

建立逻辑连接、进行硬件地址寻址、差错校验等功能。(由底层网络定义协议)

将比特组合成字节进而组合成帧,用MAC地址访问介质,错误发现但不能纠正。

物理层

建立、维护、断开物理连接。(由底层网络定义协议)

QNetworkRequest

通过一个URL地址发起网络协议请求,也保存网络请求信息

QNetworkAccessManager

用于协调网络操作。在 QNetwork Request发起一个网络请求后,负责发送网络请求,创建网络响应

QNetwork Reply

网络请求的响应。

# 第15章 多媒体

## 15.1　Qt多媒体模块功能概述

多媒体技术就是通过计算杋对语言文字、数据、音频、视频等各种信息进行存储和管理,使用户能够通过多种感官跟计算机进行实时信息交流的技术

要在项目中使用αt多媒体模块,需要在项目配置文件中添加:

Qt + mutimedia

如果在项目中使用视频播放功能,还需要添加:

Qt += multimediawidgets

以使用 QVideowidget或 QGraphicsVideoltem进行视频播放

## 15.2　音频播放

QMediapLayer可以播放单个文件

与 QMediaPlaylist结合,可以对一个播放列表进行播放

state Changed信号:在play、 pause(、stop()函数调用是发射

duration Changed信号:在文件的时间长度变化时发射(文件切换时)

position Changed信号:当前文件播放位置变化时发射(播放进度)

使用 QSound Effect和 QSound播放音效文件

QSound Effect用于播放低延迟的音效文件,如无压缩的WA文件,用于实现一些音效效果,如按键、提示等。

QSound Effect effect

effect. setSource(QUr: fromLocalFile(engine. wav"))

effect. setLoopCount(3) effect. setVolume(1)

effect. play ();

QSound只能播放本地WAV文件,而且是异步方式播放:

QSound: play l"mysounds/bells. wav);

## 15.3　音频输入

### 使用 QAudio Recorder录制音频

音频输入可以使用 QAudioRecordel或 QAudiolnput两个类来实现

QAudio Recorder是高层次的实现,输入的音频数据直接保存为文件

state Changed信号:状态包括“录音”、“暂停”、“停止”

duration Changed信号:录制持续时间变化

通过 QAudio Probe访问原始的音频数据

audioBuffer Probed信号:传递一个 QAudio Buffer类型的变量(缓冲区 的信息和音频原始数据)

QAudioInput是低层次的实现,直接控制音频输入设备的参数,并将音频录制数据写入一个流设备。

### 使用 QAudiolnput获取音频输入

QAudioInput与 QAudio Recorder的不同:

QAudiolnput创建时指定的 QAudioFormat将直接作用于音频输入设备也就是直接设置底层参数

QAudiolnput: start( QlODevice\* device指定一个α DEvice设备作为数据输出对象,可以是文件,也可以是其他 QlODevice继承的类(可以对输入的缓冲区数据进行处理)。

可以实现更加底层的音频输入控制

创建 QAudiolnput对象时需要两个参数

QAudio Devicelnfo:音频设备

QAudio Format:音频输入格式

## 15.4　视频播放

### 在 QVideoWidget上播放视频

QVideowidget是用于显示视频的界面组件,要在项目中使用 QVideowidget,需要在项目配置文件中添加:

Qt += multimediawidgets

没有插件报错:

Directshow Player Service::doRender

Unresolved error code 0x80040266

Qt中的多媒体播放,底层是使用 Direct show Player Service,所以安装一个 Directshow解码器,例如 LAV Filters

### 在 QGraphics Videoltem上播放视频

QMedia player解码的视频还可以在 QGraphicsVideoltem(继承自 QGraphicsltem)类组件上显示。是适用于 Graphics/View模式的图形显示组件,可以使用 QGraphicsltem类的缩放、拖拽、旋转等功能。

没有插件报错:

DirectShow Player Service: doRender

Unresolved error code 0x80040266

Qt中的多媒体播放,底层是使用 DirectShowPlayerService,所以安装一个 Directshow解码器,例如 LAV Filters

## 15.5　摄像头的使用

获取摄像头设备信息,通过摄像头进行拍照和录像:

QCameralnfo:摄像头信息类

QCAmera:摄像头控制类

setView Finder:为摄像头指定一个 QVideowidget或 QGraphicsVideoltem对象作为取景器。

setCapture Mode:用于设置摄像头的工作模式

QCamera: Capture Viewfinder:取景器模式

QCamera: CaptureStilllmage:抓取静态图片模式

QCamera: Capture video:视频录制模式

QCameralmage Capture:用于控制摄像头进行静态图片的获取

QMedia Recorder:视频和音频录制类( windows系统不支持视频录制)

官方解释: Qt Multimedia on Windows Qt Multimedia5.15.6

# 第16章 应用程序设计辅助功能

## 16.1　多语言界面

用QT开发多语言界面应用程序,主要包括以下几个步骤:

在程序设计阶段,程序代码中每一个用户可见的字符串都要用tr(函数封装,以便αt提取界面字符串用于生成翻译资源文件

在项目配置文件(pro文件)中设置需要导出的翻译文件(ts文件)名称,使用update工具扫描项目文件中需要翻译的字符串,并生成翻译文件

使用Qt的 Linguist程序打开生成的翻译文件,将程序中的字符串翻译为需要的语如将所有中文字符串翻译为英文

使用 Release工具编译翻译好的翻译文件,生成“qm文件

在应用程序中用 TRanslAtor调用不同的“qm”文件实现不同的语言界面

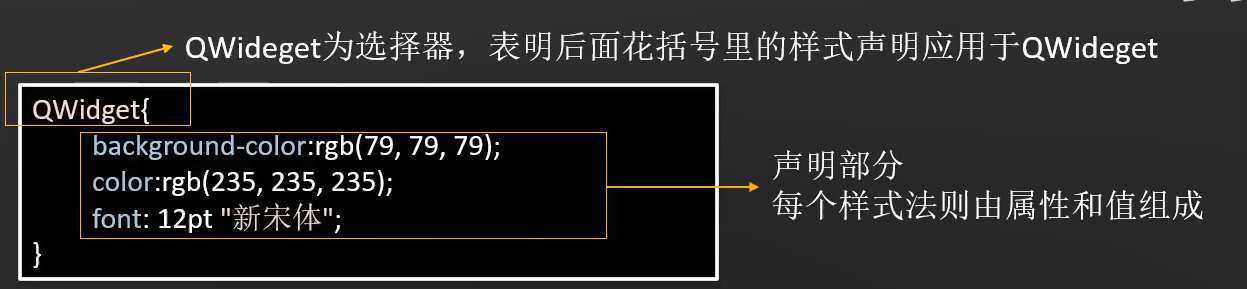
需要添加到环境变量Path:D:\QtQt5.91\ ToolsⅦ mIngw53032bin

通过 Qt Creator启动时,需要将翻译文件,放置在 makefile所在的目录下:

## 16.2　使用样式表自定义界面

无法应用于默认的标题栏

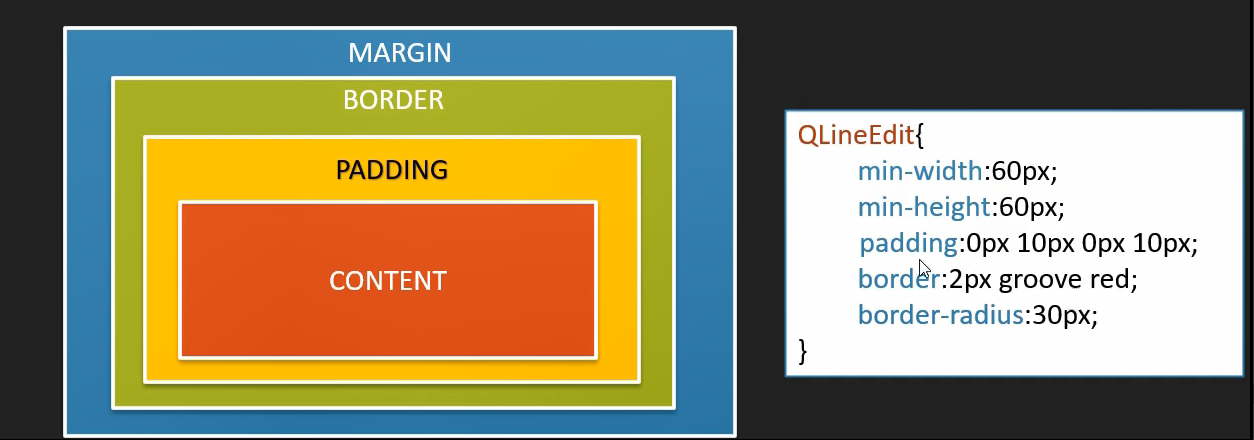
Qt样式表( style sheet)与HTML的Css( Cascading Style Sheets)类似,是纯文本的格式定义,在应用程序运行时可以加载和解析这些样式定义



选择器例子用途通用选择器所有组件类型选择器QPushButton所有 Qpush Button类及其子类的组件属性选择器QPush Button[flat=“ false”]flat为 False的 QPushButton及其子类,如果应用后组件的属性发生变化,需要重新应用样式非子类选择器 QPush Button比包含子类D选择器QPush Button#btnOKobjectName为 btnOK的 Qpush Button实例从属对象选择器 QDialog QPushButton所有从属与 DIalog的 QPushButton类的实例,子对象选择器 DIalog> QPush Button 直接从属于 DIalog的 QPushButton类的实例

属性:每个选择器可以定义多条规则,每个规则是一个“属性:值”对

可以在αt的帮助文档中查找“α t Style Sheets Reference”参看所有属性的详细说明



## 16.3　使用QStyle设置界面外观

Qt是一个跨平台的类库,相同的界面在不同的操作系统上显示效果不一样。 Qstyle是封装了GU界面组件外观的抽象类,QT定义了 Qstyle类的一些子类,应用于不同观点操作系统

QStyle Factory类管理Qt的内置样式有两个静态函数keys和 create:

keys:返回表示支持的 QStyle名称的列表

create:根据样式名称字符串创建一个 QStyle对象

QApplication有两个静态函数

style:返回当前样式

QApplication:: Style->metaObject()->className()以获取样式名称

setstyle:设置样式

## 16.4　Qt应用程序的发布

Windows发布工具: windeployqt. exe是Qt自带的 Windows平台发布工具,它可以自动为一个应用程序复制其运行所需的各种库文件、插件和翻译文件,生成可发布的目录

发布时需要选择相应的编译器的版本

D: Qt(Qt5.9. 1\5.9.1\mingw53\_32\ bin

D:QtQt5.9.1\5.9.1\ msvc2015\bin

D:Qt\Qt5.9.1\5.9.1\msvc2015\_64 bin

# 使用笔记及总结

## Qt自有类（除界面）

### QString字符串Str转换为Unicode编码的char\*字符串

QTeztcodee \* codec=QrextCodee::codeocForName("gb2312")；

QByteArray arr=codec->fromunicode (str):

string strr=arr.data()；

const char\* ch=strr.c\_str();

### 正则匹配判断str2是否含有str

QString filter=ui->lineEdite>text();

QString str="(?=";

str+=m filter; Str +="）”;

qDebug()<<str;

QRegularExpression re(str,QRegularExpression::CaseInsensitive) ;

QString str2="abc123 def";

QReqularExpressionMatch match=re.match(str2);

bool hasMatch=match.hasMatch ();// true

if(hasMatch)

qDebug()<<"true";

else

qDebug()<<"false";

## Qt 界面类

## 笔记

绘制平铺背景QPainter::drawTiledPixmap,绘制圆角矩形QPainter::drawRoundedRect(),而不是 QPainter::drawRoundRect();

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);

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。

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，设置下 a.setAttribute(Qt::AA\_NativeWindows);

获取标题栏高度：style()->pixelMetric(QStyle::PM\_TitleBarHeight); PM\_TitleBarHeight点进去你会发现新大陆。

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QVariantCustomSqlModel::data(const QModelIndex &index, int role) const。

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:");

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();

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，可以自由拉伸边框，可以使用 setWindowFlags(Qt::CustomizeWindowHint);

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

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。

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。

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。

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();

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，此时也需要做这个设置。

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);

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。

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();

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());

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();

}

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。

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); 这样可以避免可以避免对已经显示区域的重新绘制。

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;}");

### 关于Qt中文乱码的问题

应该可以解决绝大多数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。

一些其他的方法

1.在main中添加

#include <QTextCodec>

QTextCodec \*codec = QTextCodec::codecForName("UTF-8");

QTextCodec::setCodecForLocale(codec); //解决汉字乱码问题

QTextCodec::setCodecForTr(codec);  
    QTextCodec::setCodecForCStrings(codec);

1.2

QTextCodec \*codec = QTextCodec::codecForName("UTF-8");

QTextCodec::setCodecForTr(codec);

QTextCodec::setCodecForLocale(QTextCodec::codecForLocale());  
  QTextCodec::setCodecForCStrings(QTextCodec::codecForLocale());

Qt5.9以后的版本移除了setCodecForCStrings()和 codecForCStrings()

就像 std::string 做的一样， QString 可以在必要的时候由 const char\* 隐 式 或 显 式 转 换 而 成 。 这 个 过 程 中 使 用 的 编 码 ， 就 是 QTextCodec::codecForCStrings 。

Qt 有一个函数， QObject::tr ，也可以将 const char\* 转换为 QString 。 这个过程中使用的编码方式由 QTextCodec::codecForTr 指定。

另外 QTextCodec::codecForLocale 表明当前系统所用编码。一般中文 Windows 应该是 GBK 。中文 Linux 则一般为 UTF-8

但是要特别注意的是，修改 codecForLocale 要慎重！ 在 Qt里面， codecForLocale 的作用主要有两个，一个 是与外部文件读写的时候使用的默认编码，一个是向命 令行输出信息（ qDebug ）使用的编码。

2.msvc编译器在pro中添加

msvc {

QMAKE\_CFLAGS += /utf-8

QMAKE\_CXXFLAGS += /utf-8

}

msvc10好像不行

1. Qt5.9.9调试C++程序出现输出中文乱码情况，解决方案：将程序文件转成utf8+BOM编码后，再重新导入qt,问题解决，供大家参考。转成utf8+BOM编码可通过Notepad++手动转码或在vs上安装ForceUTF8 (with BOM)插件，安装后重新导入项目文件，生成项目自动转码成功
2. Qt 读取文件可以用任何一种编码，只是默认以 codecForLocale 而已。 使用 QTextStream 可以通过文本方式读取 QFile ，而 QTextStream 则可以设定使用的编码方式。 就算文件名有中文也不怕。只要正确转换为 QString ，剩 下的就交给 Qt 吧！

QFile file(QString::fromUtf8(" 中文名.txt"));

if (file.open(QIODevice::ReadOnly)) {

QTextStream fin(&file);

fin.setCodec(QTextCodec::codecForName("utf-8"));

fin.readLine(); }

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

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；

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。

1：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；

2：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；

3：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；

4：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；

5：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；

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 widgets = fatherWidget.findChildren ("widgetname");

//查找所有

QPushButton QList allPButtons = fatherWidget.findChildren ();

//查找一级子控件,不然会一直遍历所有子控件

QList childButtons = fatherWidget.findChildren (QString(), Qt::FindDirectChildrenOnly);