# 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

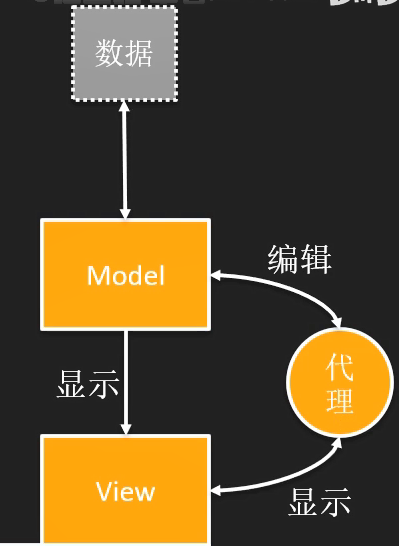
## 5.1 Model/View结构

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。

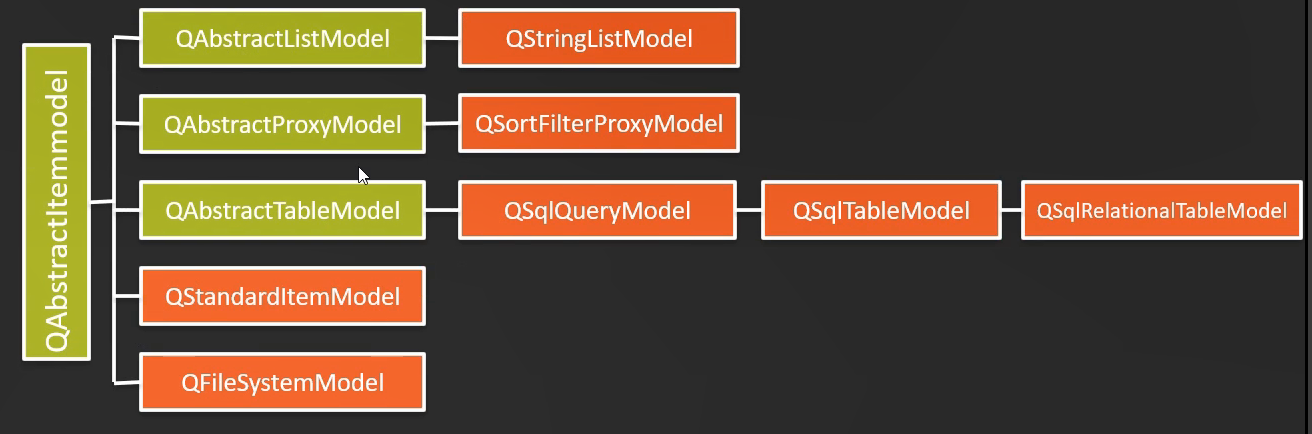
View:是屏幕上的界面组件，视图从数据模型获得每个数据项的模型索引（model index），通过模型索引获取数据

·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。

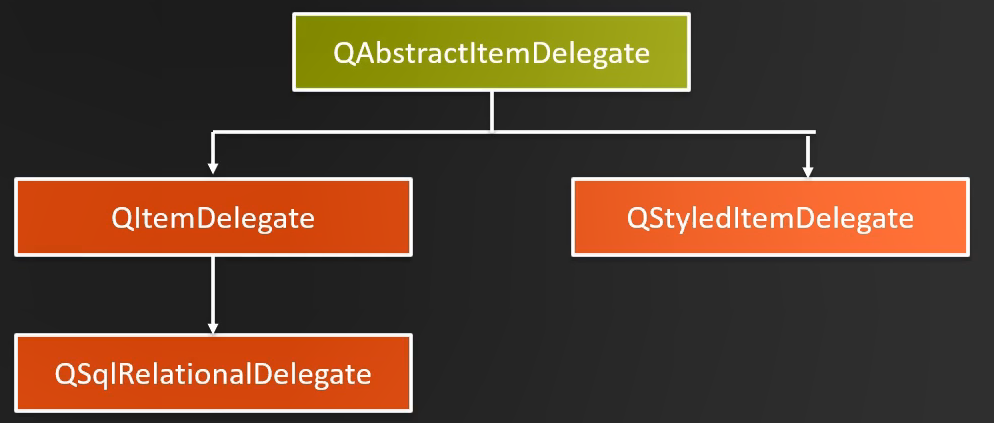
QltemSelectionModel:跟踪视图组件的单元格选择状态的类，通过QltemSelectionModel可以获得选中的单元格的模型索引。

## 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 |

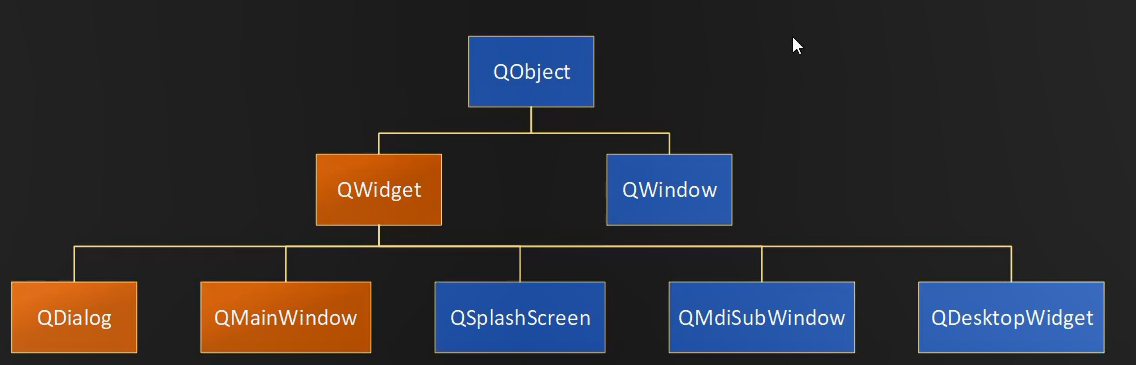
## 6.2　自定义对话框及其调用

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 |

## 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

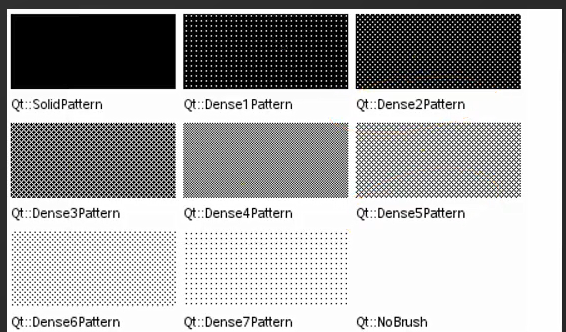
painter.setPen(QPen(..))；

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

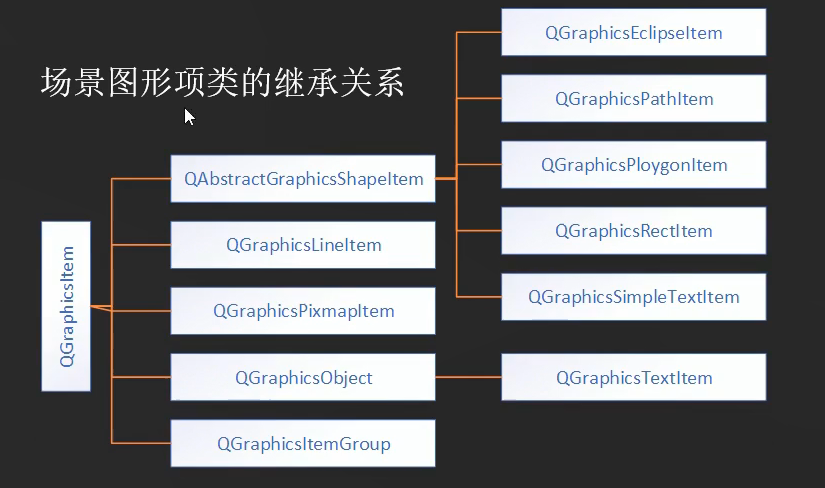
QGraphicsltem(图形项）：也就是图元，支持鼠标事件响应。比如：矩形的图形项QGraphicsView(视图）：关联场景可以让场景中的所有图形项可视化

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

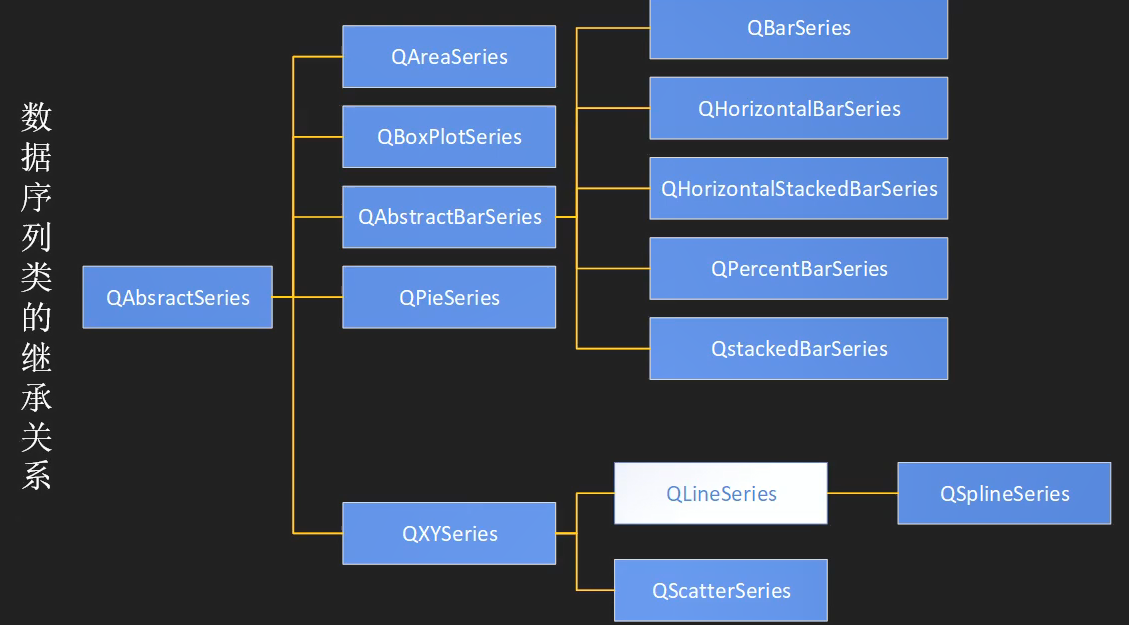
在需要使用QtCharts的类的头文件或源程序文件中，要使用如下的包含语句：

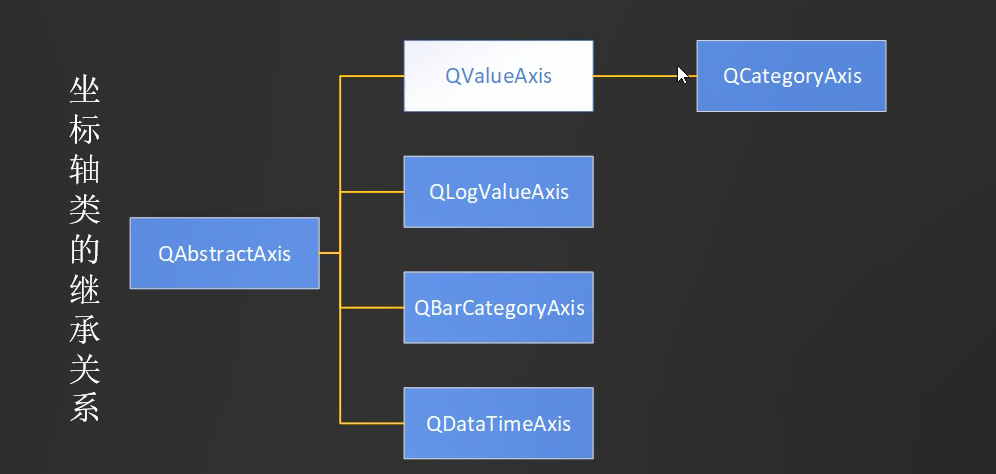
#include <QtCharts>

//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

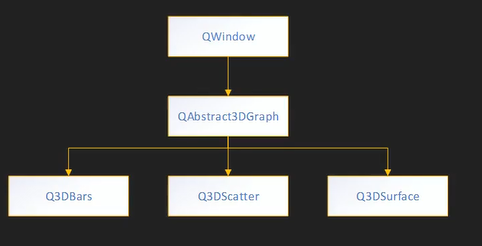
为了支持上面的功能需要自定义一个QChartView的派生类

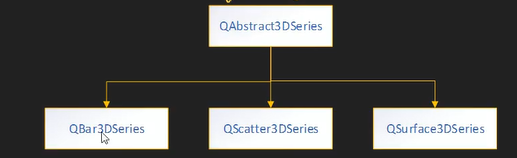
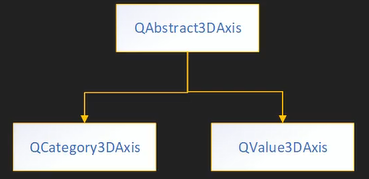
# 第10章 Data Visualization

## 10.1　DataVisualization模块概述

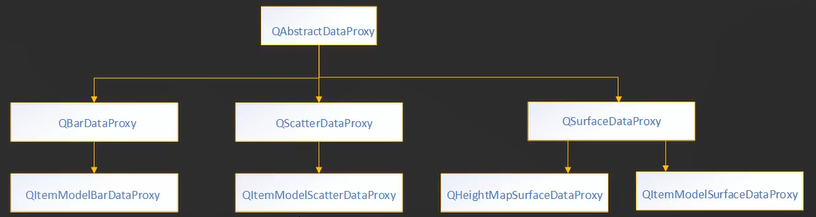
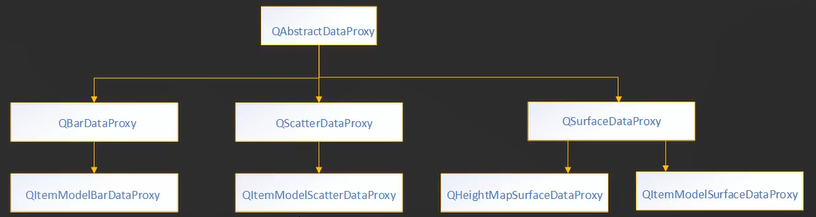
Data Visualization的三维显示功能主要由三种三维图形来实现，分别是三维柱状图Q3DBars，三维空间散点Q3DScatter，三维曲面Q3DSurface。这三个类的父类都是QAbstract3DGraph，从QWindow继承而来。

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：

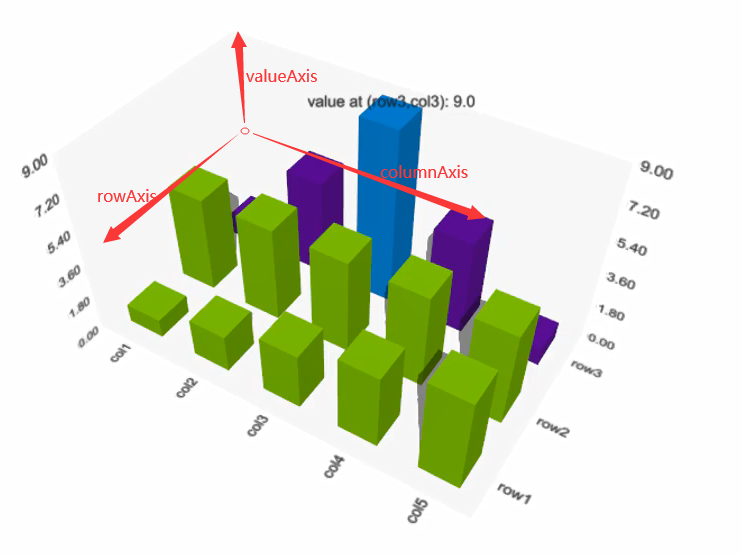
QT+=datavisualization 在调用时源码中添加：

#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。

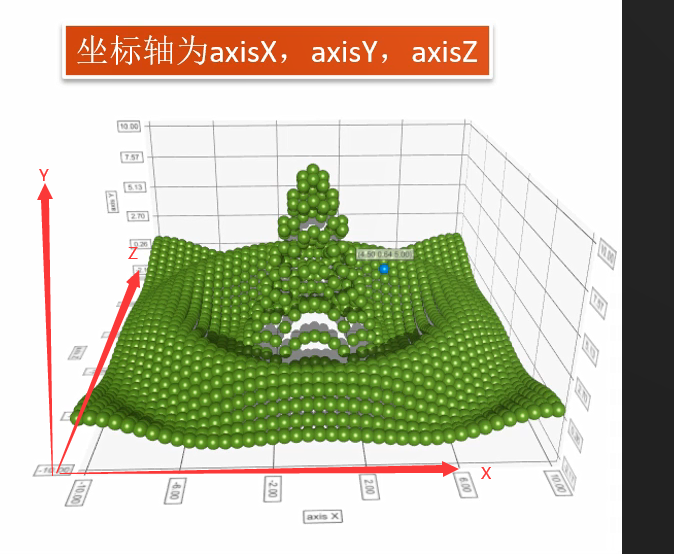
namespace QtDataVisualization{

typedef QVector<QBarDataltem>QBarDataRow;

typedef QList<QBarDataRow \*> QBarDataArray;

}

## 10.3　三维散点图



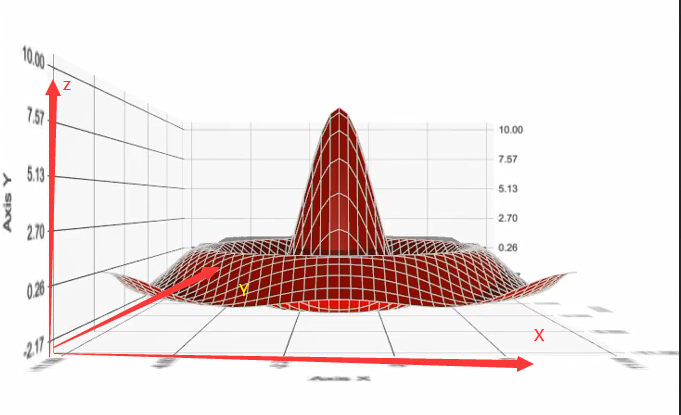
namespace QtDataVisualization{

typedef QVector<QScatterDataltem>QScatterDataArray;

}

## 10.4　三维曲面绘图

### 3D曲面图

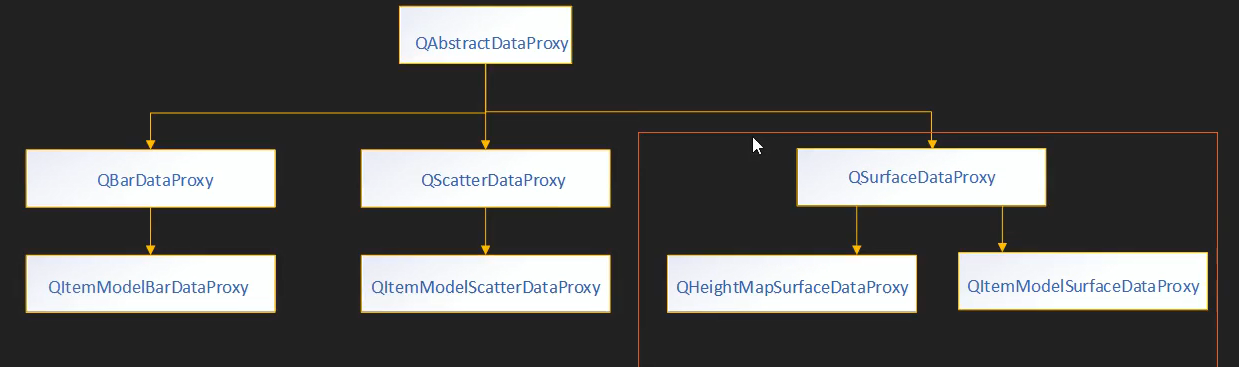


使用QSurfaceDataProxy

namespace QtDataVisualization{

typedef QVector<QSurfaceDataltem>QSurfaceDataRow;

typedef QList-QSurfaceDataRow \*>QSurfaceDataArray;

}

enum ColorStyle{

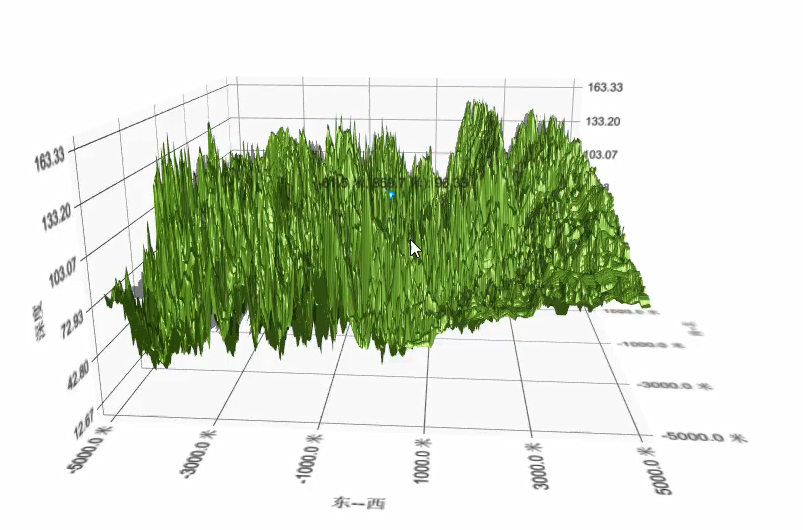
ColorStyleUniform=0,/单一颜色

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：

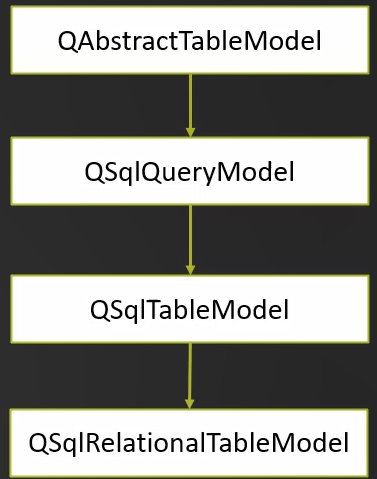
QSqlQueryModel:

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

## 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

## 13.2　线程同步

# 第14章 网络编程

## 14.1　主机信息查询

## 14.2　TCP通信

## 14.3　QUdpSocket实现UDP通信

## 14.4　基于HTTP协议的网络应用程序

# 第15章 多媒体

## 15.1　Qt多媒体模块功能概述

## 15.2　音频播放

## 15.3　音频输入

## 15.4　视频播放

## 15.5　摄像头的使用

# 第16章 应用程序设计辅助功能

## 16.1　多语言界面

## 16.2　使用样式表自定义界面

## 16.3　使用QStyle设置界面外观

## 16.4　Qt应用程序的发布