# QT

http://download.qt.io/archive/qt/

## function

### 消息响应

A类中有SIGNAL函数sendSigFun()

B类中有SLOT函数RevSigFun()

将A对象的信号函数与B对象的槽函数绑定

QObject::connect(&Aobj,SIGNAL(sendSigFun()),

&Bobj,SLOT(RevSigFun()));

消息函数的声明宏必须放在头文件中

函数绑定时一定不能写形参名，只能写类型

class AA :public QObject {

Q\_OBJECT //信号槽类需要声明宏

signals: //信号函数以signals:指示字符开始

void **sendSigFun**(const char\*);

public:

void **Proc**(const char\* str) {

emit sendSigFun(str);//信号函数调用前加emit

}

};

class BB :public QObject

{

Q\_OBJECT //信号槽类需要声明宏

private slots: //private slots：指示字符开始

void **RevSigFun**(const char\* str) {

qDebug()<<str;

}

};

QObject::connect(&Aobj,SIGNAL(sendSigFun(const char\*)),

&Bobj,SLOT(RevSigFun(const char\*)));

### 线程

class WorkerThread : public QThread

{

void run() override {

}

};

WorkerThread m\_WorkerThread;

m\_WorkerThread.start();

### <QDebug>

qDebug()<<\*\*\*\*\*

### <QString>

|  |  |
| --- | --- |
| 函数 | 功能 |
| QString()  QString(const char\* str)  QString(const QChar \*unicode, int size = -1) | 构造函数 |
| int a = str.toInt(); | 字符串转数字 |
| QString str=QString::number(12) | 数字转字符串 |
| str = QString::asprintf("%d,%.3f",12,12.123); | 格式化字符串 |
| int indexof(QString str);  int indexof(char ch); | 未找到返回-1 |

### <QList>

QList<int> list;

MyChar ch=list.first();

foreach(int, list){}遍历，每次将list中的成员放入ch

list.count()

### <QTimer>

QTimer \*timer=new QTimer(&obj);

timer->setInterval(1000);

QObject::connect(timer,SIGNAL(timeout()),&obj,SLOT(RevSigFun()));

timer->start();

## 动态库

int Q\_DECL\_EXPORT **sum**(int a,int b)

{

return a +b;

}

修改为控制台工程：

CONFIG += c++11 console

#TEMPLATE = lib

## 网络

#include <QtNetwork>

#include <QTextCodec>

### http

QNetworkAccessManager m\_manager;

connect(&m\_manager,SIGNAL(finished(QNetworkReply \*)),this,SLOT(rev(QNetworkReply \*)));

m\_manager.get(QNetworkRequest(QUrl("http:/\*\*\*\*")));

public slots:

void rev(QNetworkReply \*ry)

{

QTextCodec \*codec = QTextCodec::codecForName("utf-8");//QTextCodec::codecForLocale()

QString all = codec->toUnicode(ry->readAll());

qDebug()<<all;

ry->deleteLater();

}

### udp

#include <QtNetwork>

QT += network

#### 服务器

QUdpSocket m\_reciver;

void \*\*\*::revData()

{

while(m\_reciver.hasPendingDatagrams())

{

QByteArray dat;

dat.resize(m\_reciver.pendingDatagramSize());

m\_reciver.readDatagram(dat.data(),dat.size());

qDebug()<<dat.data();

}

}

m\_reciver.bind(1234,QUdpSocket::ShareAddress);

connect(&m\_reciver,SIGNAL(readyRead()),this,SLOT(revData()));

#### 客户端

QUdpSocket m\_client;

QByteArray dat = "aaa,bbb,ccc";

QHostAddress addr("127.0.0.1");

m\_client.writeDatagram(dat,addr,1234);

## widget

### 结构

#include <QApplication>

#include <QWidget>

int main(int argc, char \*\*argv){

QApplication app(argc,argv);

QWidget w; //继承 QWidget进行扩展

w.show();

return app.exec();

}

sudo apt -get install qt-sdk

1. 创建工程目录
2. 在该目录下编写代码
3. qmake -project (a.pro)
4. qmake Makefile
5. make

### qtdesigner消息绑定

qtcreator中可以使用qtdesigner在界面里右键控件，菜单中选择转到槽，生成对应槽函数

在界面对应类的构造函数中写入绑定

QtGuiApplication1::QtGuiApplication1(QWidget \*parent)

: QMainWindow(parent)

{

ui.setupUi(this);

QObject::connect(ui.butObj, SIGNAL(clicked()), this, SLOT(Onclick()));

}

### 鼠标

<QMouseEvent>

void mousePressEvent(QMouseEvent\*){}

容器窗口的消息响应，用虚函数重写了父类实现的

QPoint pos(); 获取鼠标的坐标

QT：：MouseButton button(); 获取键值

QT：：keyboardModifiers modifiers(); 伴随键

exp:

void mouseMoveEvent(QMouseEvent \*ev){

ev->pos()

}

### 键盘

<QKeyEvent>

if(QT::shiftModifier|QT::ControlModifier)

判断是否同时按下shift和ctrl

void keyPressEvent(QKeyEvent \*ev){}

int key();

QString modifiers();

QString text(); 键的名字

bool isAutoRepeat();按下后不放

注意键名的大小写

exp：

if(ev->key()==QT::Key\_8)

判断8键是否按下

### 控件

没有父窗口的窗口句柄为主窗口

窗口之间是对象间的关系

#### 按钮

<QPushButton>

QPushButton button;

button.setText("Button");

button.setParent(&w); w为QWidget类的对象

button.setGeometry(x,y,cx,cy);

QPushButton(QWidget \*parent=0)

QPushButton(QString text,QWidget \*parent=0)

QPushButton \*button=new QPushButton(“Button”,&w)

父窗口接管内存管理，不用通过button释放内存

#### <QLabel>

QLabel(QString str, QWidget \*parent=0)

void setText(QString str);

QString text(); 获取显示内容

#### <QLineEdit>

QLineEdit(QWidget \*parent=0)

QLineEdit(QString str, QWidget \*parent=0)

void setText(QString str)

QString text(); 获取显示内容

void setEchoMode(QLineEdit::EchoMode mode);设置显示模式

mode：

Normal

NoEcho 无显示

Password 输入时看见\*\*

PasswordEchoOnEdit 输入时看不见内容

QString str=edit.text();

int value=str.toInt();

将输入字符串数字改为数字

connect(&edit1,SIGNAL(textEdited(QString)),&edit2,SLOT(setText(QString)));

edit1变化，edit2同步显示

#### <QTextCodec>

QTextCodec \*codec=QTextCodec::codeForName(“UTF8”);

本机文本编码

修改部分文字：

codec->toUnicode(“要转换的文字”)

修改整个程序的文字：

QTextCodec::setCodecForCStrings(codec);

#### <QSlider>

<QSpinBox>

connect(slider,SIGNAL(valueChanged(int),spinbox,SLOT(setValue(int)))

slider滑块滑动，spinbox框内数字变化

slider：

QSlider slider(QT::Horizontal,parent)

QT:;Vertical

int value()

void setMaxinum(int max)

void setValue(int value)

void SetMininum(int min)

void SetRange(int min,int max)

QSpinBox:

QSpinBox(QWidget \*parent=0)

其他与QSlider一样

#### <QLCDNumber>

显示LED格式字符和数字

QLCDNumber(QWidget \*parent=0);

lcd=new QLCDNumber(&w); 如果在w类内的构造函数定义，可以

用this指针

lcd->setNumDigits(8);

lcd->setGeometry(10,10,23,23);

QTime t=QTime::currentTime(); <QTime>

lcd->display(t.toString(“hh::mm::ss”));

#### <QMenuBar>

QMenuBar(QWidget \*parent=0)

<QMenu>菜单项目

按顺序加入：

QMenu \*menu=menubar->addMenu(“&file”);

必须用指针方式

menuaddAction(“&open”,&w,SLOT(ab()));

menuaddSeparator()；加上分隔线

#### <QPainter>

调用时间：

1. 窗口显示时（程序开始运行时）
2. 调用QWidget的update()成员函数
3. 系统认为需要重绘时

void paintEvent(QPaintEvent \*){}

QPainter p(&w); 一般定义在窗口类中，用this

p.drawLine(QLine(...))

p.drawRect(QRect(...))

p.drawEllipse(QRect(...)) 外切矩形的椭圆

p.drawEllipse(QPoint(中心点x，y，半径1，半径2))

p.drawText(QRect(....),QT::AlignCenter,”xx”);

p.drawImage(x,y,QImage(“带路径的文件名”)) <QImage>

1. 绘制动作：drawLine，drawRect.....
2. 选择工具动作：

QPen 影响开放性几何图形和封闭性集合图形的边缘

QBrush 只影响封闭图形内部

<QPen>

<QBrush>

<QFont>

p.setPen(QPen(QT::red));

p.setBrush(QBrush(QT::yellow));

p.setFont(QFont(“字体”,”画笔大小”，高度，bool表示是否斜体));

#### 坐标体系

QPoint(int x, int y)

QLine(int x1, int y1, int x2, int y2)

QLine(QPoint p1, QPoint p2)

QRect(int x,int y, int wide, int height)

QSize(int wide, int height)

#### <QHBoxLayout>

horizontal

QHBoxLayout layout(&w);

创建一个layout指向主窗口

layout.addWidget(&button);

layout.addWidget(&button1);

将需要被layout控制的控件放入，layout绑在主窗口，控件绑在layout上

QHBoxLayout(QWidget \*parent=0);

void addWidget(QWidget \*widget, int stretch=0)

第二个参数用来放大窗口占的总水平线的比例

void addLayout(QLayout \*layout, int stretch=0)

void addStrech(int stretch=0)

void addSpacing(int size)

layout.addStretch(1) 最左边，向右挤

layout.......... 按顺序加入排列内容

layout.addStretch(1) 最右边，向左挤

#### <QVBoxLayout>

与水平排列类相同，QH改为QV即可

QHBoxLayout layout(&w);

QVBoxLayout v;

可使用： layout.addLayout(&r)

#### <QGridLayout>

QGridLayout layout(&w);

void addLayout(QLayout \*layout, int row, int col)

void setRowStretch(int row, int stretch)

void setColumnStretch(int col , int stretch)

exp:两行两列排

layout.addWidget(&b1,0,0);

layout.addWidget(&b2,0,1);

layout.addWidget(&b3,1,0);

layout.addWidget(&b4,0,0);

### 设置窗体位置

setGeometry(0, 0, 400, 400);

### 消息框

<QMessageBox>

QMessageBox::warning(&w,”warning”,”信息字符串”)；

## QML

### 结构

引用对象重命名必须大写字母开头

import QtQuick 2.9 as Mm

import "Aa.js" as Jobj //引用JavaScript代码

Window {//实例化一个Window类的对象

visible: true

width: 640;height: 480

title: *qsTr*("title")

color: "#FF0000FF"//ARGB

Mm.Component.onCompleted: console.debug("finish") //初始化完成时调用

Mm.MouseArea{

anchors.fill: *parent*

onClicked: *console*.debug("颜色：" + *color*) //调试输出

onReleased: *console*.debug(Jobj.fun(10))

}

Mm.Rectangle{//窗口中创建一个矩形区域

id: *aB3C* //id必须小写字母或者下划线开头

visible: true

color: "#FF00FF00"

width: 600

height: 200

}

Mylable{//默认可以直接访问到同目录下对象类型，需要其他对象类型时需要import

Mm.Text{ text:"xxxx"} //为Mylable中的Text对象默认属性赋值

}

}

Mylable.qml:

Text {

default property var someText//默认属性

readonly property int intVal: 10 //只读属性

text: "val:" + intVal + someText.text

}

Aa.js:

function fun(val){

return val \* 12;

}

### 与c++交互

#### 对象注入

类中声明属性调用函数

Q\_PROPERTY(QString name READ getName WRITE setName)

//代码中将对象加入上下文

#include <QQmlContext>

QQmlApplicationEngine engine;

MyData data;

QQmlContext \*ptx = engine.rootContext();

ptx->setContextProperty("msg",&data);

//qml调用

msg.name = "abc";//调用setName 函数

var val = msg.name;//调用getName 函数

#### 注册qml对象

class MyData:public QObject

{

Q\_OBJECT

Q\_PROPERTY(QString name READ getName WRITE setName NOTIFY nameChanged)

public:

void setName(const QString &name)

{

if(m\_name != name)

{

m\_name = name;

emit nameChanged(name);

}

}

QString getName()const{

return m\_name;

}

signals:

void nameChanged(const QString &name);

//函数首字母必须小写，对应槽函数onNameChanged

private:

QString m\_name;

//注册枚举类

Q\_ENUMS(Status)

public:

enum Status

{

Status\_A,

Status\_B,

};

Q\_INVOKABLE Status getStatus() //注册可以被qml调用的函数，函数首字母必须小写

{

return Status\_A;

}

};

//代码中注册

qmlRegisterType<MyData>("jyl.data", 1, 0, "MyData");

//qml调用

import jyl.data 1.0

MyData{

id:myd

onNameChanged: function(name)

{//可以不写function(name)

console.debug(name);

}

}

myd.name = "abc";//调用setName 函数

var val = myd.name;//调用getName 函数

//调用getStatus()函数

var ss = myd.getStatus();

if(ss === 0) console.log("Status\_A");

### 布局

|  |  |
| --- | --- |
| 类型 | 说明 |
| anchors | anchors.centerIn: *parent*  将元素位置固定到父元素中间 |
| anchors.right: *r1*.left  anchors.rightMargin: 30  将元素位置固定到父元素左侧（window无right，可以在window上覆盖一个Rectangle）,且设置间距为30像素 |
| Column {  spacing: 10  } | 将所有visible属性为true的项目排成一列，数据列间距设置为10 |
| Row {} | 排成一行 |
| Grid {  columns: 3  rows:3  } | 定义一个3行3列的排列网格，如果其中的元素数量超出9个，则会从第一个格子开始覆盖。 |

### 控件

|  |  |
| --- | --- |
| 控件类型 | 使用 |
| Item | 所有项目都继承自Item，常用与项目分组  Item  {  opacity: 0.2//不透明度，1表示完全不透明。该属性应用于该项目下所有项目  ｝ |
| Rectangle | 使用纯色或渐变色绘制矩形区域 |
| Text | Text{  font.pointSize: 30  text:"123"  color: "blue"  } |
| Button | import QtQuick.Controls 1.1  Button{text: "开始";y:0;onClicked: fun()} |

### 粒子系统

import QtQuick.Particles 2.0

import QtQuick.Controls 1.1

如果将渲染器和发射器放入粒子系统类，则不需要定义id

ParticleSystem{

id:*par*//定义粒子系统par

running: false

}

ItemParticle{

system: *par*

delegate://为par定义渲染器

Rectangle{

id:*rect*;width: 10;height: 10;color: "red";radius: 10

}

}

Emitter{////为par定义发射器

system: *par*

emitRate: 50//每秒发射粒子数，默认10

x:0;width: *parent*.width //从0开始宽度为父对象宽度的直线上发射粒子

velocity: PointDirection{y:800;x:400;yVariation: 100;}//发射方向（x与y组合的向量），yVariation表示粒子间y方向不同但是差异不超过该值

}

//按钮控制par的开始

Button{text: "开始";y:0;onClicked: *par*.start()}