## 常用

**大小设置**

QObject->resize(x , y);

setFixeSize(x,y);//固定大小

**窗口标题**

setWindowTitle("text");

**布局**

在.ui中使用控件布局

使用widget、弹簧等规划布局

合理利用水平布局，垂直布局，栅格布局

## 控件

**QPushButton**

创建

QPushButton \* btn = new QPushButton;

信号

&QPushButton::clicked; &QPushButton::

icon属性负责按钮图标

**按钮控件组**

QToolButton

QRadioButton //单选按钮

QCheckBox //多选按钮

**QListWidget控件组**

QListWidget

使用方式

QListWidgetItem \* item = new QListWidgetItem("text");

QListWidget->addItem(item); //添加一行item

QStringList list;

list << "text" <<"text"...<<;

QListWidget->addItems(list); //添加多行item

QTreeWidget

使用方式

tree->setHeaderLabels(QString()<<"text"<<....);

//每一行是一个QTreeWidgeItem类型

QTreeWidgeItem \*item = new QTreeWidgeItem(QString()<<"text");

//加载顶层节点

tree->addTopLevelItem(item);

//追加子节点

QStringList list;

list<<"text"<<....;

QTreeWidgeItem \*item2 = new QTreeWidgeItem(list);

item->addChild(item2 );

QTableWidget

使用方式

table->setColumnCount(int) //设置列数

//设置表头

table->setHorizontalHeaderLabels(QString()<<"text"<<...)

table->setRowCount(int)//设置行数

//设置正文

table->setItem(row,column, new QTableWidgetItem("text"));

//"text"，可用 QList<QString>、QStringList存储

**QMenuBar**

菜单栏

创建

QMenuBar \* bar= menuBar( );

setMenuBar(bar);

创建菜单

QMenu \*editmenu = bar->addMenu("text");

创建菜单项

editmenu->addAction("text");

editmenu->addSeparator();

**QToolBar**

工具栏

创建

QToolBar \* toolBar = new QToolBar(this);

addToolBar(toolBar);

设置内容

toolBar->addAction(QAction \*);

toolBar->addSeparator();

添加控件

toolBar->addWidget(Qwidget \*);

**QStatusbar**

创建

QStatusbar \* stBar = statusBar();

setStatusbar(stBar);

创建标签

stBar->addWidget(QLabel \* );

**QDockWidget**

创建

QDockWidget \* dock = new QDockWidget();

addDockWidget(dock);

**QTextEdit**

创建

QTextEdit \*edit = new QTextEdit (this);

setCentralWidget(edit);

**QDialog**

创建

QDialog dlg(this);

dlg.exec();//模态方式运行

dlg.show();//非模态

//由于不阻塞，只会一闪而过，在函数中

在堆上创建

QDialog \*dlg = new QDialog(this);

释放

dlg->setAttribute(Qt::WA\_DeleteOnClose);

//关闭时，释放空间

**QMessageBox**

创建

QMessageBox::critical(this, "title", "text");

QMessageBox::waring(this, "title", "text");

QMessageBox::infomation(this, "title", "text");

QMessageBox::question(this, "title", 参数/\*支持位运算\*/);

//返回值 standButton

**QColorDialog**

QColorDialog::getColor(QColor(int r,int g,int b));

//返回值是QColor类型

**QFileDialog**

QFileDialog::getOpenFileName(this, "text", "path");

//返回值是一个Qstring 路径

**QFontDialog**

QFontDialog::getFont(&bool, QFont("type", size) );

//返回值是一个QFont类型指针

## 特性

**对象树**

当父节点被释放，其子节点也被释放

使用方式，设置某一对象父节点 QObject->setParent(QObject \*);

**信号和槽**

1、使用方式

connect(QObject1\*, &QObject::singal, QObject2\*, &QObject::slots);

//发送 信号 接收 槽函数

disconnect(QObject1\*, &QObject::singal, QObject2\*, &QObject::slots);

//断开信号

2、自定义信号和槽

1. 新建类，在新建类中自定义，并且注意选择基类
2. 信号需要写在 signals 下
3. 返回值void 需要声明，不需要实现，可以有参数，可以重载
4. 早期QT写在Public slots下，现在可以写在public下
5. 返回值void 需要声明， 需要实现，可以有参数，可以重载
6. 触发信号 emit->signal();
7. 参数应当一一对应;

3、如何解决重载函数名冲突？

fun(int a); fun(char b);

(void )( QObject:: \* fPointer )(int a) = &QObject::fun;

1. 信号连接信号

connect(QPushButton\* , &QPushButton::clicked, QObject1\*, &QObject1::signal);

connect(QObject1\*, &QObject1::signal, QObject2\*, &QObject2::signal);

//点击按钮，一次发多个信号

**去掉字符串引号**

string "text" ————>> string.toUtf8(),data()

**Lambda表达式**

c++11里才能使用，CONFIG += c++11

一种匿名函数对象

[ ] ( 函数参数 ) {

/\*

[ ]中的参数

&:全引用

=:全值传递,?但不可修改值

\*/

};



mutable:

返回值 int ret = [ ]( ) ->int{return x; } ();

## QMainWindow

**层次**

菜单栏，唯一， QMenuBar

工具栏，QToolBar

可以设置默认位置

可以限定可移动位置

setMovable

setAllowAreas

set\*\*\*()

状态栏,唯一，QStatusbar

锚接部件

中心部件，唯一

## 对话框

模态：阻塞，不可与其他窗口交互

非模态：非阻塞，可操作其他窗口

## 资源文件添加

在项目下添加文件

Qt Resoure File

在.qrc里新增资源

## 事件

事件类继承于QEvent类，在事件对象创建成功后会被传递给QObject的event()函数。

event函数不处理事件，而是按照事件对象类型分派给特定的事件处理函数。

许多事件回调函数都是protected virtual（QWidget）所以可以在子类中重写这些虚函数 且必须重写。

**如何重写？**

例如QLabel，需要在自定义的label控件(创建类，提升为控件)中实现重写。

在新增label.h中声明，label.c中定义

**鼠标事件**

**定时器事件**

**事件分发器**

用于事件分发的event函数也可以重写，即在事件分发之前的操作。

原理：对于传入的事件已经被处理则返回true，否则返回false。返回true后则不会 对此次事件进行分发。

注意：对于不关心的事件，用父类的事件分发器处理

virtual bool event(QEvent \*e)

利用 e->type()判断事件类型 并return true

不关心的事件 return QLabel:;event(e); //QLabel视情况而定

**事件过滤器**

event()函数是protected的，需要在继承的基础上重写，当有很多类的时候，需要重 写很多event()。

过滤器在QEvent之前拦截。

不想让他继续转发就返回true，否则返回false

installEventFilter(QObject \*filterQbj) 加载事件过滤器

eventFilter(QObject \*watched, QEvent \*event) 重写事件过滤器

注意：

1、这种全局的事件过滤器将会在所有其它特性对象的事件过滤器之前调用。尽管 很强大，但这种行为会严重降低整个应用程序的事件分发效率。

2、事件过滤器和被安装过滤器的组件必须在同一线程，否则，过滤器将不起作用。 另外，如果在安装过滤器之后，这两个组件到了不同的线程，那么，只有等到二 者重新回到同一线程的时候过滤器才会有效。

**文件读写**