QT

# 1.开发环境搭建

## 1.1.Windows下开发环境搭建

在如下网址：<https://www.qt.io/download-open-source/> 下载得到了如下文件：qt-unified-windows-x86-online.exe，该文件是一个安装程序，可以在线下载安装QT开发需要的环境，包括：MinGW，QT运行需要的库文件，QT Creator等；

本安装方式，需要实时的网络下载！

## 1.2.Ubuntu16.04下安装开发环境

下载QT：**http://download.qt.io/archive/qt/**，选中了最新的5.11.1版本的源码包；

# 2.HelloWorld

点击“New Project”，创建一个新的项目“helloworld”，创建时选择“模板”为“Application”，具体为“Qt widgets application”。类名等信息做相应修改。

项目创建完成后，自动进入的是“编辑”模式，选择“界面文件”中的“mainwindow.ui”文件，双击后将进入该ui的“设计”模式。

在“设计”模式下，选取一个“label”控件，直接拉到ui上，修改内容为“helloworld”，之后点击左下角的“运行”按钮，就可以将该显示。

如果要发布release版本的程序，需要在编译时选择release方式，这样编译完成后就可以在workspace中找到对应的release目录，例如我这里有一个：build-helloworld-Desktop\_Qt\_5\_7\_0\_MinGW\_32bit-Release 目录，其中包含了release目录，该目录下游需要的helloworld.exe文件。

需要发布的时候，除该exe文件外，还需要包含所需要的dll文件，这里我们需要的是：

1. libwinpthread-1.dll；
2. Qt5Core.dll；
3. Qt5Gui.dll；
4. Qt5Widgets.dll；

这几个文件，这几个文件在安装目录下可以找到，我这里的安装目录是：E:\QT\InstallFiles\5.6\mingw49\_32\bin；

将exe文件和这几个dll文件打包在同一个文件夹中，在任何位置都可以直接点击exe文件，弹出helloworld对话框了。

## 2.1.更换应用程序图标

如果要给生成的exe文件更换一个自定义的图标，可以如下操作：

1. 在项目中增加一个myapp.rc文件(文件自定义)，然后再里面输入内容：IDI\_ICON1               ICON    DISCARDABLE     "appico.ico" (这里的appico.ico就是自定义需要使用的图标);
2. 在.pro文件中增加下边一条代码：RC\_FILE = myapp.rc
3. 重新编译程序；

需要注意的是，需要使用一个符合icon文件格式的图标，否则编译失败，将提示：windres: icon file `./../helloworld/appico.ico' does not contain icon data；

## 2.2.无需每次复制dll文件的方法

每次对dll文件拷贝显然是麻烦的，针对开发者，还可以有如下两种方式，无需每次拷贝文件：

1. 设置PATH环境变量，这里我的设置是：E:\QT\InstallFiles\5.6\mingw49\_32\bin；
2. 产出成静态文件(文件体积增加，不推荐)；

# 3.快捷键

## 3.1.更改QT Creator的自动补全快捷键

“工具(tools)”—“选项(options)”，选择“键盘”后，输入CompleteThis，将自动过滤出现在的快捷键，可以看到是“ctrl+space”，但是这个经常与输入法的中英文切换冲突，改成eclipse里用习惯了的“Alt+/”；

注意这里修改的时候，是让QT Creator自动捕获键盘动作，而不是手动输入的。

## 3.2.快速切换已经打开的文件

Ctrl+tab

## 3.3.注释 / 取消选中的代码

Ctrl+/

## 3.4.自动缩进选中代码

Ctrl+i

## 3.5.添加固定格式的整段注释的快捷方式

1. “工具”—“选项”—“文本编辑器”；
2. 选中“片段”中的“添加”按钮，新建一个“触发”，名字自己定为了“funcComment”，触发种类同样适用到了这个名字；
3. 填写自己的注释内容：

/\*\*

\* @brief

\* @param

\* @author

\* @date

\*/

执行如上设置后，在qtcreator中输入“funcComment”，就会提示该段落；

## 3.6.改变编辑器中字体大小

Ctrl + +

Ctrl + -

## 3.7.编辑 信号/槽

在“设计”模式下，按下F4可以进入编辑“信号/槽”的界面；

# 4.设置字体编码格式

可以通过如下方式改变编码格式，一般在中文乱码显示时需要进行设置：

#include <QTextCodec>

QTextCodec::setCodecForLocale(QTextCodec::codecForName("GB2312"));

需要注意的是，一定要在主函数执行逻辑动作之前就进行设置才能全部执行过程中都生效；

# 5.两种捕获按钮按下动作的方法

## 5.1.槽回调函数

选中要设置动作的按钮，选择“选中槽”后，选择某一个动作(例如左键单击对应的动作就是clicked())，函数将跳转到一个函数处(例如on\_pushButton\_clicked())，这个就是自动生成的槽，已经在头文件中做了声明，只需要修改其函数实现即可。

## 5.2.信号/槽

在“设计”模式下，按下F4进入“信号/槽”的编辑模式，选中需要设置按下动作的按钮，并选择某一个动作和对应的接收对象即可。

QT Creator快速入门

对《QT Creator快速入门》第二版的读书笔记；

# 3.窗口部件

QT Creator提供的默认基类，有QMainWindow、QWidget和QDialog三种；

* QWidget是所有窗口部件的基类；
* QDialog是各种对话框的基类；
* QMainWindow是带有菜单栏和工具栏的主窗口类；

## 3.1.基础窗口部件QWidget

### 3.1.1.窗口、子部件以及窗口类型

#### 窗口与子部件

一段示例代码：

#include <QtGui>

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

{

QApplication a(argc, argv);

//代码中支持中文

QTextCodec::setCodecForTr(QTextCodec::codecForName(“UTF-8”));

//新建QWidget对象，默认parent设置为空，是一个窗口

QWidget \* widget = new QWidget();

//新建label对象，默认parent设置为空，是一个窗口

QLabel \* label = new QLabel();

//label2也是一个label对象，但是指定父窗口为widget，所以不是一个窗口

QLabel \* label2 = new QLabel(widget);

//在显示器上显示

label->show();

widget->show();

int ret = a.exec();

//释放对象，注意，label2不执行delete

delete label;

delete widget;

//结束

return ret;

}

需要注意的点：

1. 头文件包含的是QtGui，因为用到的类：QApplication、QWidget都包含在这个模块里；
2. 因为label没有父窗口，所以需要手动释放它申请的空间；而label2的父窗口是widget，**QT中销毁父对象的时候，会自动销毁子对象**，因此label2不需要手动释放空间，因为delete widget的时候已经将它释放了；

#### 窗口类型

默认的窗口，一般都存在边框和标题栏，但其实这并不是必须的；

QWidget的构造函数有两个参数：QWidget \* parent = 0 和 Qt::WindowFlags f = 0；

* 第一个参数“parent”指定其父窗口部件，默认值为0表示不存在父窗口；
* 第二个参数f的类型Qt::WindowFlags是一个枚举，分为：窗口类型(WindowType)和窗口标志(WindowFlags)。

简单的例子，对于窗口类型来说，Qt::Dialog可以指定窗口的样式是对话框类型；Qt::SplashScreen指定窗口类型是欢迎窗口类型；Qt::Widget是默认值，如果有父窗口，就是其子部件，否则就是一个独立窗口；

对于窗口标志来说，作用主要就是更改窗口的标题栏和边框，而且可以和窗口类型进行或操作，例如：Qt::FramelessWindowHint用来产生一个没有边框的窗口；Qt::WindowStaysOntTopHint用来使本窗口停留在所以其他窗口上面；

### 3.1.2.窗口几何布局

对于窗口的大小和位置，根据是否包含边框和标题栏两种情况，要用不同的函数来获取。例如：

* 包含框架的：x(), y(), frameGeometry(), pos(), move()等；
* 不包含框架的：geometry(), width(), height(), rect(), size()等；

### 3.1.3.程序调试

#### 设置断点

将光标移动到要调试的函数上，按下F1键，可以查看到函数的帮助文档；

设置好断电后，按下F5键可以开始调试；

按下F11，可以进入单步调试；

将光标移动到一个类名或者函数上，按下F2键，或者右击后选择“跟踪光标位置的符号”，编辑器会跳转到源码处；

#### 使用qDebug函数

使用qDebug函数，可以将调试信息直接输出到控制台；

示例：

1. qDebug(“ret = %d”, ret);
2. qDebug << “ret = ” << ret << enl << “desc : ” << desc;

如果使用方式1输出，使用类似于printf的输出格式，无需新增头文件；

方式2输出，无需手动指定format，更加方便，但是需要包含头文件：#include <QDebug>；

方式2中，endl与std中的标准输出换行作用相同；

## 3.2.对话框Dialog

### 3.2.1.模态和非模态对话框

QDialog类是所有对话框窗口类的基类。

对话框窗口是一个经常用来玩成一个短小任务或者和用户有简单交互的顶层窗口；

模态对话框就是没有关闭它之前，不能再与同一个应用程序的其他窗口进行交互，像新建项目时弹出的对话框就是这种；

非模态对话框就是既可以与它交互，也可以与同一个程序中的其他窗口交互，例如word中查找字符串的功能；

最简单的，想让一个对话框作为模态存在，只需要使用其exec()函数来执行；而想做为一个非模态的，只需要使用show()函数的默认方式来执行；

需要注意的是，使用show函数也可以创建一个模态对话框：

QDialog \* pD = new Dialog(this);

pD->setModal(true); //默认该属性为false

pD->show();

如上进行设置后，就可以将该对话框设置为模态的了。

但是与exec函数的效果还是不同的：exec函数一旦被执行，该对话框不消失的情况下，其他对话框(包括父窗口)都将无法显示；而show函数不同，其他窗口依然存在，只是无法交互了；

与stModal()函数类似的还有一个函数：setWindowModality函数。

该函数功能更加丰富，它专门有一个参数设置模态对话框要阻塞的类型：

* Qt::NonModal:不阻塞任何窗口，就是非模态对话框；
* Qt::WindowModal:阻塞它的父窗口和所有祖先窗口，以及它们的子窗口；
* Qt::ApplicationModal:阻塞整个应用程序的所有窗口；

setModal函数的功能，其实与Qt::ApplicationModal相同；

### 3.2.2.多窗口切换

#### 信号和槽

QT中使用信号和槽，完成对象之间的协同操作；

声明槽函数：

public slots:

void showChildDialog();

槽函数的声明，必须使用“slots”关键字；这里使用public，说明这个槽可以在类外被调用。

实现该函数，可以通过快捷键实现：选中槽函数，按下Alt+enter后，选中“在\*\*\*.cpp中添加定义”，编辑器会自动创建该函数的声明，只需要增加代码即可；该方法同样适用于先在源文件中增加了定义，然后自动在头文件中增加声明的情况；

槽函数声明后，想在控件点击或其他动作时触发这个槽，还需要进行关联：connect(ui->showChildBtn, SIGNAL(clicked()), this, SLOT(showChildDialog())); 这里使用了connect函数进行关联，connect函数的四个参数分别是：发送信号的对象、发送的信号、接收信号的对象和要执行的槽；信号和槽分别使用SIGNAL和SLOT宏包起来。

信号和槽的关联，还有一种方式是自动关联。

自动关联就是将将关联函数整合到槽的命名中，但是必须使用QT部件已经提供的信号。例如：on\_showChildBtn\_clicked()，由“on”+“要发射信号的部件对象名”+“信号名”组成。这样就可以不适用connect函数进行关联了。

快捷方式修改所有某一个函数出现的地方：选中该函数，右键，弹出的菜单中选择“重构🡪Rename Symbol Under Cursor”；

#### Demo

主窗口、登录窗口，可以互相切换；

### 3.2.3.标准对话框

#### 颜色对话框

头文件：#include <QColorDialog>

颜色对话框类 QColorDialog提供了一个可以获取指定颜色的对话框部件。两种显示颜色对话框的调用方式：

1. 静态调用：QColor color = QColorDialog::getColor(Qt::red, this, tr(“颜色对话框”), QColorDialog::ShowAlphaChannel);四个参数的含义分别是：设置初始颜色、父窗口、对话框标题、显示alpha的设置内容；
2. 动态调用：QColorDialog dialog(Qt::red, this); dialog.setOption(QColorDialog::ShowAlphaChannel); dialog.exec(); QColor color = dialog.currentColor();

方式1的优点是不用创建对象，写起来更方便；但如果想更灵活的设置颜色对话框的属性，就需要用到方式2，也就是先创建对象，之后进行设置；

#### 文件对话框

头文件：#include <QFileDialog>

文件对话框类QFileDialog提供了一个允许用户选择文件或者文件夹的对话框。同样分为静态和动态两种调用方式：

1. 静态调用：QString filename = QFileDialog::getOpenFileName(this, tr(”文件对话框”), “F: ”, tr(“图片文件(\*png \*jpg)”));四个参数的含义分别是：父窗口、对话框名字、默认打开的文件路径、文件类型过滤器。
2. 动态调用：创建对象、设置属性；

设置文件类型过滤器，可以使打开的文件对话框中只显示这些类型的文件；如果不指定该值，默认打开所有类型的文件；**注意，\*png和\*jpg之间有一个空格**！

如果要显示更多种类的文件，**需要以两个分号隔开**！例如：QFileDialog::getOpenFileName(this, tr(”文件对话框”), “F: ”, tr(“图片文件(\*png \*jpg);;文本文件(\*txt)”))；

如果允许一次选中多个文件，需要使用QFileDialog类提供的getSaveFileNames函数实现，其返回值类型是QStringList，所有文件名置于其中；

QFileDialog类还提供了两个函数：

* getSaveFileName函数实现了保存文件对话框和文件另存为对话框；
* getExistingDirectory函数获取一个已经存在的文件夹路径；

#### 字体对话框

头文件：#include <QFontDialog>

字体对话框提供了一个可以选择字体的对话框部件。调用方式，同样分为静态和动态两种，例如：

bool ok;

QFont font = QFontDialog::getFont(&ok, this);

if(ok)

ui->btn1->setFont(font);

else

qDebug() << tr(“没有所选择的字体”);

这里使用的就是静态的、getFont函数来获取选择的字体；

#### 输入对话框

头文件：#include <QInputDialog>

输入对话框QInputDialog类用来提供一个简单方便的对话框，从用户那里获取一个单一的数值或者字符串，例如：

* QInputDialog::getText(this, tr(“输入字符串对话框”), tr(“请输入用户名”), QLineEdit::Normal, tr(“admin”), &ok);提供了一个可以输入字符串的对话框，参数的作用分别是:
  + 父窗口；
  + 窗口标题；
  + 对话框中的标签的显示文本；
  + 字符串的显示模式，例如密码可以设置成小黑点，这里用了Normal，是显示用户输入的时机内容；
  + 输入框中的默认字符串；
  + 按下按钮的bool变量，捕获动作；
* QInputDialog::getInt(this, tr(“输入整数对话框”), tr(“请输入一个整数，范围在[-1000, 1000]”), 100, -1000, 1000, 10, &ok); 这里100表示的是对话框的默认值；-1000和1000表示的分别是允许输入的最小值和最大值；10标识箭头按钮，数值每次变化10；
* QInputDialog::getDouble(this, tr(“输入浮点数对话框”), tr(“请输入-1000到1000内的数值”), 0.00, -1000, 1000, 2, &ok); 其他参数类似，倒数第二个参数2，指定的是小数的位数；
* QStringList items; QInputDialog::getItem(this, tr(“输入条目对话框”), tr(“请选择一个条目”), items, 0, true, &ok); 需要首先提供一些条目，例如这里的items；参数0表示的是默认显示列表中的第0个条目；参数true表示的是是否允许修改条目，true就是可以修改；

如上的例子里，用到了静态函数的方式，也可以使用静态函数的方式来实现；

#### 消息对话框

头文件：#include <QMessageBox>

消息对话框提供了一个模态的对话框，来通知用户一些信息，或者向用户提出一个问题并获取答案，例如：

* QMessageBox::question(this, tr(“问题对话框”), tr(“你了解QT吗？”), QMessageBox::Yes, QMessageBox::No);
* QMessageBox::information(this, tr(“提示对话框”), tr(“学习QT吧！”), QMessageBox::Ok);
* QMessageBox::warning(this, tr(“警告对话框”), tr(“有错误！”), QMessageBox::Abort);
* QMessageBox::critical(this, tr(“严重错误对话框”), tr(“致命错误！”), QMessageBox::YesAll);
* QMessageBox::about(this, tr(“关于对话框”), tr(“这是一个说明这个工具是干什么的对话框”));

#### 进度对话框

#include <QProgressDialog>

进度对话框对一个耗时较长的操作，进行进度条的反馈，例如：

QProgressDialog dialog(tr(“文件复制进度信息”), tr(“取消”), 0, 5000, this);

dialog.setWindowTitle(“进度对话框”); //设置窗口标题

dialog.setWindowModality(Qt::WindowModal); //**设置为模态对话框**

dialog.show(); //开始显示

for(int I = 0; I < 5000; i++)

{

dialog.setValue(i); //设置进度值

QCoreApplication::processEvents(); //**避免页面冻结**

if(dialog.wasCanceled()) //用户按下了取消按钮

break;

}

dialog.setValue(5000); //这样才能显示到5000，for只能加到4999

需要注意：

1. 示例中将其设置为了模态对话框，还可以使用非模态的，但是需要定时器等其他类的配合；
2. QCoreApplication的使用，是为了避免长时间没有用户动作，导致界面冻结；
3. 构造函数的参数含义分别是：对话框的标签内容，取消按钮的显示文本，最小值，最大值，父窗口；

#### 错误信息对话框

头文件：#include <QErrorMessage>

错误消息提示框提供了一个显示错误信息的对话框。例如：

QErrorMessage \* dialog = new QErrorMessage(this);

dialog->setWindowTitle(tr(“错误消息对话框”));

dialog->showMessage(tr(“这里是出错信息”));

这种方式实现，对话框以非模态的形式体现；

在错误消息对话框中，默认有一个Show this message again的复选框，用来选择以后是否还显示相同的错误信息。

#### 向导对话框

头文件：#include <QWizard>

向导对话框提供了一个设计向导界面的框架。最典型的向导对话框的使用，就是安装、卸载软件时候的向导。

暂略。