# 1、QT的安装

## 什么是QT

QT是基于C++的跨平台的用户图形界面开发框架

## QT的安装

### Qtcreator

QT的集成开发环境（IDE），编码、编译、运行、调试。。。。

### ubuntu下有线安装与卸载

ubuntu包管理工具：apt-get install qt-sdk

CentOS包管理工具：yum install XXX

卸载：

|  |
| --- |
| apt-get purge qt-sdk  apt-get autoremoveqt-sdk |

apt-get 下载下来的包在什么地方？

|  |
| --- |
| /var/cache/apt/archives |

apt-get install 下载源

|  |
| --- |
| /etc/apt/sources.list |

### ubuntu下无线安装与卸载

两种方式：

1. 获取源代码：编译、安装
2. 获取安装包。

apt-get download qt-sdk，仅仅只能下载这一个安装包，大多数时候并不能安装成功，软件本身要依赖一些其他的软件，所以我们要下载其他的依赖包。

进入安装包目录，安装：

|  |
| --- |
| 安装：dpkg -i\*.deb |
| 卸载： |

安装的时候要求的是后缀名.deb的安装包，卸载的时候不能通过安装包卸载，要找到那个程序的名字进行卸载。

一般包以下划线 \_进行组织包名，一般第一个代表程序名称。

awk命令介绍：

awk以行方式处理数据，默认以空格对行数据进行分割

|  |
| --- |
| ll | awk '{print $8}' |

我们可以用-F这个参数来指定以什么进行分割行数据

|  |
| --- |
| 单个字符直接跟在F的后面：ll | awk-F: '{print $2}' |
| 如果是字符串，则要单引号：ll | awk -F'4096' '{print $1}' |
| ll | awk '{print $NF}'； $NF代表最后一列 |

卸载：

|  |
| --- |
| for i in `ls | awk -F\_ '{print $1}'`; do dpkg -P $i;done |

### Windows下安装

安装包下载地址：[www.qt-project.org](http://www.qt-project.org)

# 2、第一个QT程序

## 2.1在终端上编写QT程序

1、编写源代码：main.cpp

|  |
| --- |
| #include <QApplication>  #include <QWidget>  int main(int argc, char \*\*argv)  {  QApplication app(argc, argv);  QWidget w;  w.show();  return 0;  } |

2、生成一个工程管理文件：qmake -project

会生成一个XXX.pro文件

1. 生成Makefile文件：qmake
2. 生成可执行文件：make

如果仅仅这样做，会出错：

怎么解决？？？

在工程管理里面增加两个库，gui、widgets

|  |
| --- |
| ##########################################################  # Automatically generated by qmake (3.0) ?? 4? 5 21:42:37 2017  ##########################################################  TEMPLATE = app  TARGET = 01  INCLUDEPATH += .  # Input  SOURCES += main.cpp  QT += gui widgets |

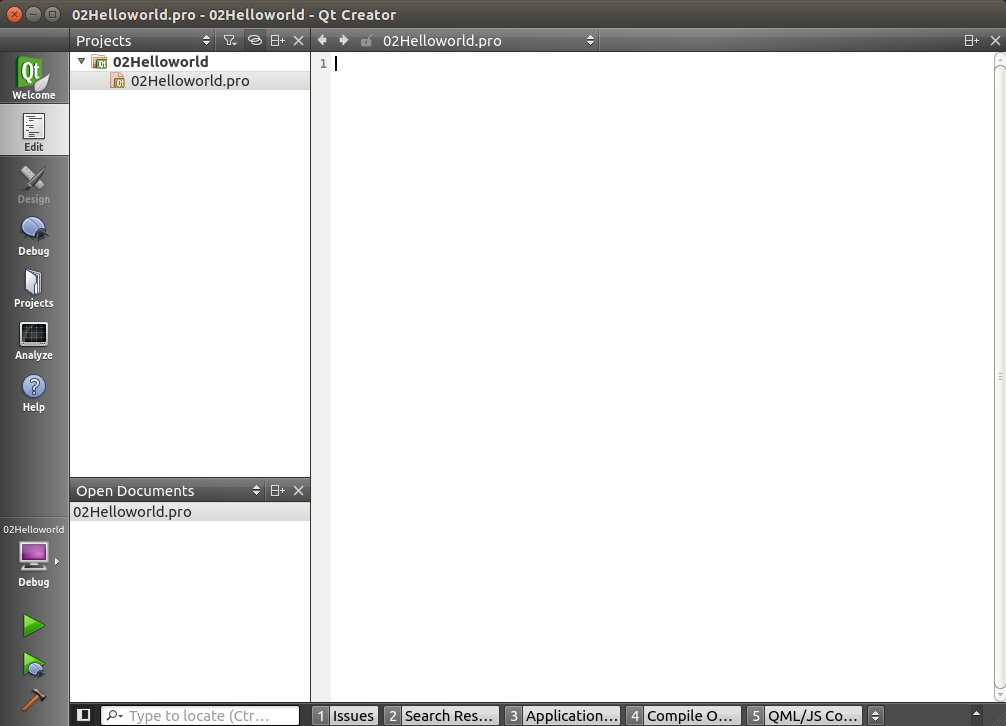
这句话是什么意思呢，就是说这个QT程序要额外包含两个库，这两个库呢，一个叫gui，一个叫widgets。以前4.8的时候生成的工程文件自动将这两个库加进去，但是4.8以后为了让程序更加轻量级，有些库就不会自动包含了，要手动包含。

只要更改工程文件Makefile文件会自动跟着变。

## 2.2在qtcreator上编写QT程序

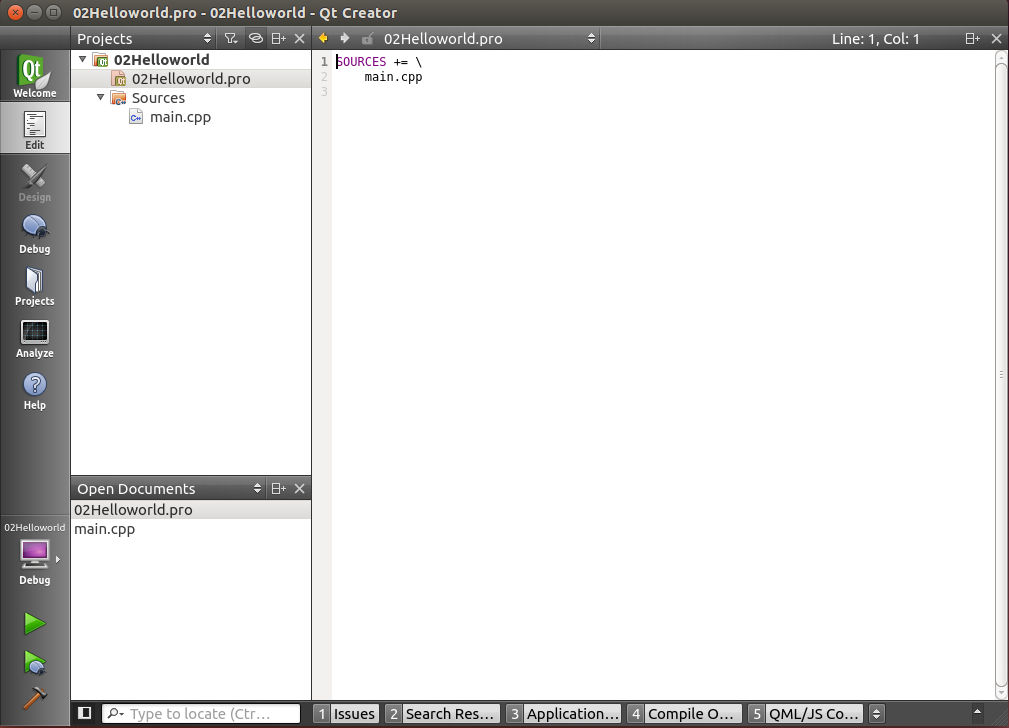
1、在终端上输入qtcreator命令打开qtcreator。

2、创建一个空的项目。



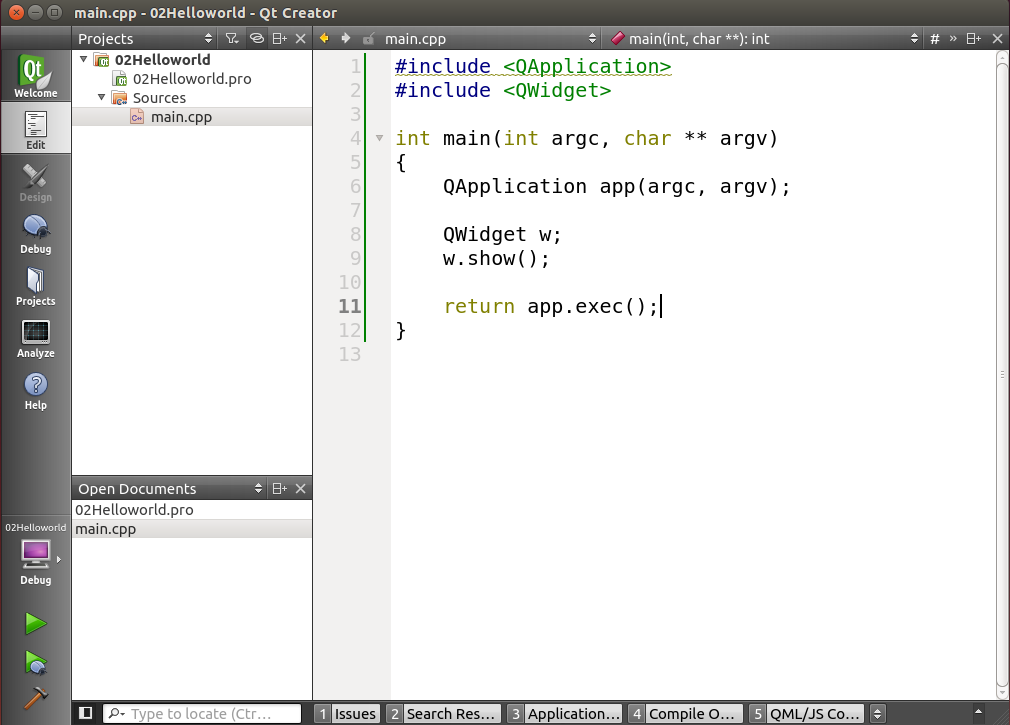
里面只有一个工程管理文件

3、添加一个源文件main.cpp

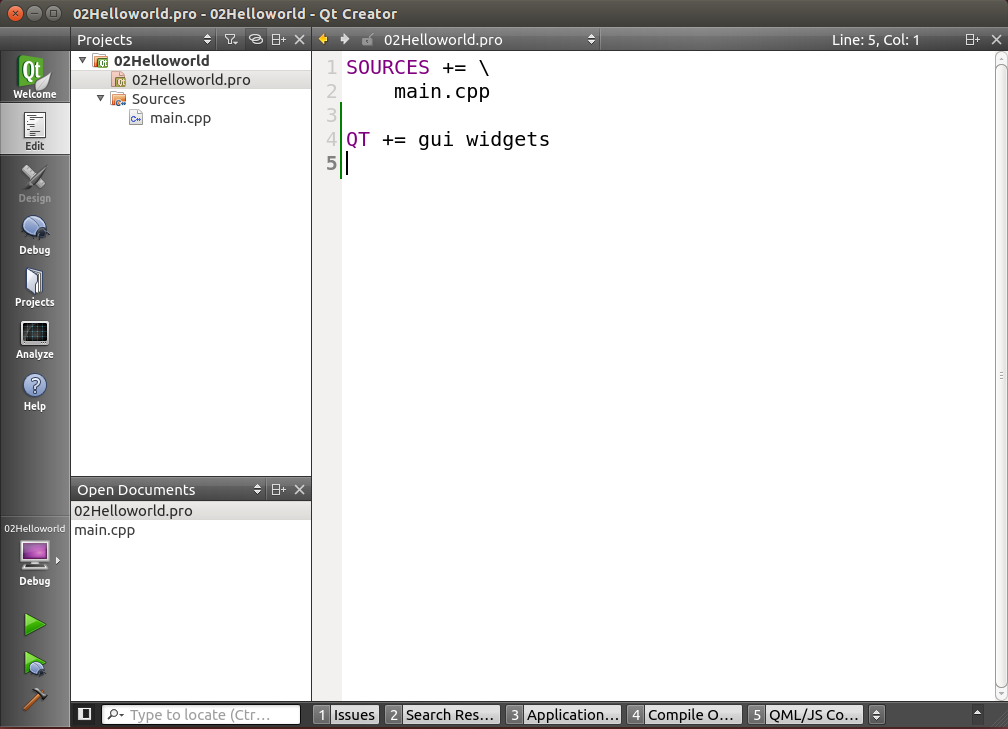


工程文件会自动生成相应内容

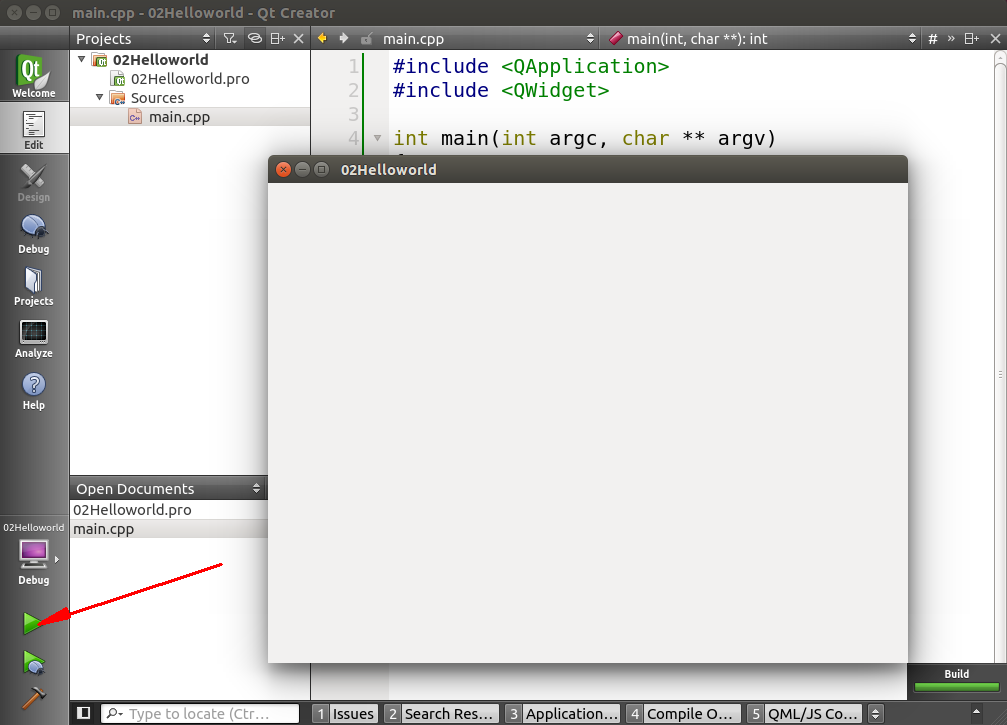
4、编辑源文件，修改工程文件



在工程文件中添加 QT+=gui widgets



5、编译运行



点击左下角三角形或者使用 Ctrl+R进行运行程序。

## 2.3程序分析

|  |
| --- |
| //应用程序抽象类，抽象的是一个应用程序，表示一个应用程序  #include<QApplication>  //窗口类，代表窗口  #include<QWidget>  intmain(intargc,char\*\*argv)  {  //定义了一个应用程序对象app,app就代表当前应用程序  QApplicationapp(argc,argv);  //定义了一个窗口对象，代表一个窗口  QWidgetw;  //显示窗口  w.show();  //设置窗口标题  w.setWindowTitle("ABC");  //app的exec里面有一个消息循环，死循环，处理消息（事件）  returnapp.exec();  } |

## 2.4添加一个按钮

1、包含按钮的头文件：#include<QPushButton>

2、创建按钮的对象：QPushButtonbutton;

3、设置按钮显示内容：

可以有两种方式设置：

1. 创建对象的时候直接设置：QPushButtonbutton("Button");
2. 通过内部成员函数设置：

4、设置对象的父子关系，将当前窗口对象设置为该button的父对象：

button.setParent(&w);

注意；  
 1、父子关系说的是什么对象的父子关系，不是类的父子关系

2、窗口的show()的前后顺序不同，显示会不一样。

## 2.5父子对象关系

父子关系影响显示位置

没有父对象的窗口，叫主窗口。

## 2.6按钮的事件响应：信号与槽机制（Signal&Slot）

QT提供信号与槽的机制，来让任意两个对象之间进行消息处理。作用就是让一个对象产生的信号能够被另一个对象接受并处理。

QT基本所有的对象都继承QObject对象，在这个对象中有一个静态函数connect(..),这个函数可以让一个对象产生的信号能够被另一个对象接受并处理。

QObject::connect(&button,SIGNAL(clicked()),&w,SLOT(close()));

第一个参数：要发送消息的对象（一定要是QObject子类对象），这里为按钮对象button

第二个参数：要发送的信号，用SIGNAL宏将其转化为char\*类型，这里为button的点击事件

第三个参数：接收信号的对象（也是QObject子类对象），这里是窗口对象w

第四个参数：接收信号的处理方式，用SLOT将其转化为char\*类型，这里为窗口对象的槽函数close

前两个参数称为信号，后两个参数称为槽。经过这样的连接之后，按钮button的clicked函数和窗口w的close函数就进行了绑定，调用button的clicked函数就相当于调用了窗口对象w的close函数。 这种方式的好处是将两个独立的模块，可以通过第三方连接起来，降低了设计的耦合性。

# 3、控件

## 3.1文本输入框

### 3.1.1输入模式

头文件：QLineEdit

使用：

|  |
| --- |
| QWidgetw;  QLineEditlineEdit;  lineEdit.setParent(&w);  w.show(); |

这是一个基本的输入框，我们可以更改输入框的一些属性，以适应不同的场合

QT中有一些宏定义了输入框的不同行为

enumEchoMode{Normal,NoEcho,Password,PasswordEchoOnEdit};

Normal：正常输入模式

NoEcho：输入的时候不显示

Password：密码方式显示

PasswordEchoOnEdit：输入的时候为明文，切换焦点以后就变为密码模式

可以通过设置输入模式方法更改输入模式：

lineEdit.setEchoMode(QLineEdit::Password);

### 3.1.2获取文本

lineEdit.text();

### 3.1.3输入提示

通过text方法获取输入框的文本内容，返回的是一个QString字符串

lineEdit.text();

### 3.1.4自动补全

QCompleter\*completer=newQCompleter(QStringList()<<"123"<<"1234"<<"1abc"<<"wang"<<"li"<<"zhang");

lineEdit.setCompleter(completer);

通过设置completer的匹配模式能进行不同模式下的字符匹配

completer->setFilterMode(Qt::MatchFlag::MatchContains);

## s3.2坐标系统

窗体类中有一个setGeometry方法可以设置窗体的位置。

控件也是窗体，控件也可以通过这个函数设置自己的位置

***button.setGeometry(30,30,100,30);***

这个按钮坐标的位置为（30，30），宽度为100，高度为30

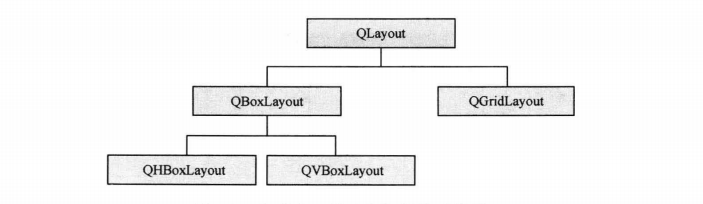
注意：

1. UI坐标是以左上角为原点，往左为X轴正向，往下为Y轴正向，控件位置是相对于原点的位置。
2. 某个对象在设置位置的时候用的坐标体系是其父对象的坐标体系，简而言之，对象设置的位置是相对于其父对象而言的。

## 3.3布局管理

在设计复杂的GUI用户界面时，仅通过指定父子对象之间的位置关系以期达到加载和排列窗口部件的方法是行不通的。我们可以使用布局管理相关的类帮我们自动管理对象之间的关系。

QT提供了QHBoxLayout类、QVBoxLayout类及QgridLayout类等的基本布局管理，分别是水平排列布局、垂直排列布局和网格排列布局。它们之间的继承关系如下：



以QVBoxLayout使用为例：

1、包含头文件：#include<QVBoxLayout>

2、构造对象：QVBoxLayout layout;

3、在布局管理对象layout中加载窗口部件：

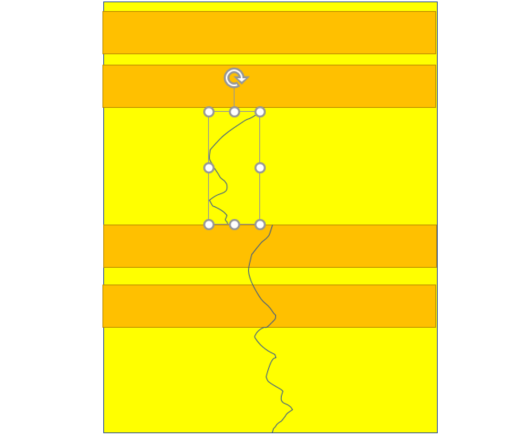
layout.addWidget(&button);

layout.addWidget(&lineEdit);

4、在窗口中设置布局：w.setLayout(&layout);

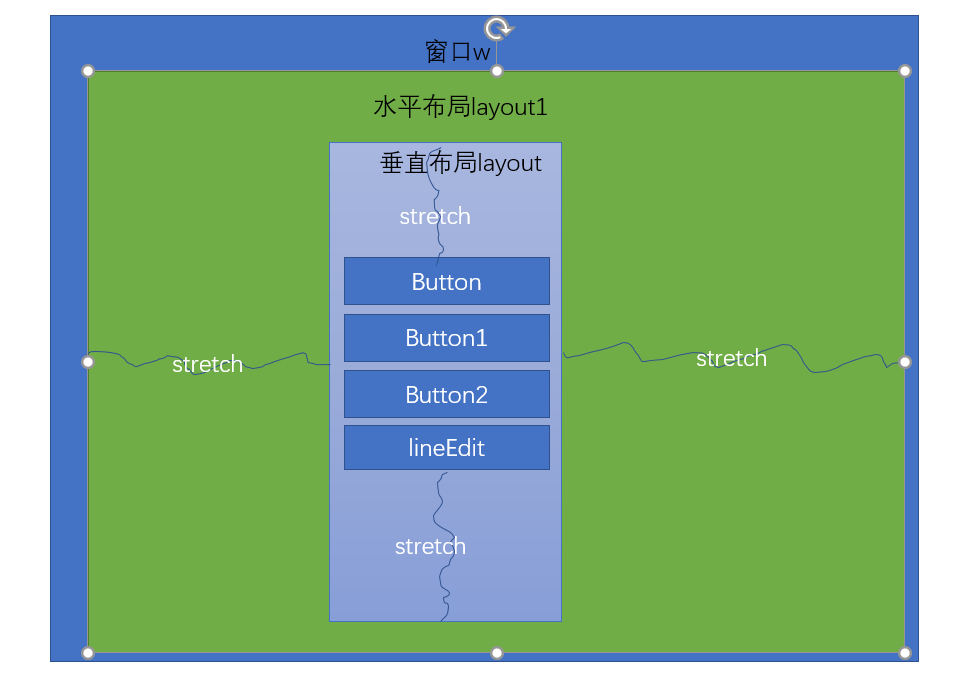
现在层次关系变为当前窗体中有一个垂直布局layout，这个layout中有一个按钮和一个文本输入框。

|  |
| --- |
| vlayout.addWidget(&button1);  vlayout.addWidget(&button2);  vlayout.addStretch(1);  vlayout.addWidget(&button3);  vlayout.addWidget(&lineEdit);  //增加一个弹簧，防止往下进行伸缩  vlayout.addStretch(1); |



布局嵌套：

|  |
| --- |
| QVBoxLayoutlayout;  layout.addStretch(1);  layout.addWidget(&button);  layout.addWidget(&button1);  layout.addWidget(&button2);  layout.addWidget(&lineEdit);  layout.addStretch(1);  QHBoxLayoutlayout1;  layout1.addStretch(1);  layout1.addLayout(&layout);  layout1.addStretch(1);  w.setLayout(&layout1); |

布局分析：

窗口中有一个水平布局layout1，水平布局中有一个垂直布局layout，垂直布局中有3个button和一个文本输入框。水平布局的两边和垂直布局的上下都设置了stretch，可以将它们想象为弹簧，将控件王中间挤压。

网格布局的伸缩设置：

|  |
| --- |
| QGridLayoutlayout2;  layout2.addWidget(&button,1,1);  layout2.addWidget(&button1,1,2);  layout2.addWidget(&button2,2,1);  layout2.addWidget(&lineEdit,2,2);  layout2.setColumnStretch(0,1);  layout2.setColumnStretch(3,1);  layout2.setRowStretch(0,1);  layout2.setRowStretch(3,1); |

布局分析

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | button | button1 |  |
| button2 | lineEdit1 |
|  | | | |

做一个登陆框

|  |
| --- |
| QGridLayoutlayout2;  QLineEdit\*password=newQLineEdit();  password->setEchoMode(QLineEdit::Password);  layout2.addWidget(newQLabel("username"),1,1);  layout2.addWidget(newQLineEdit,1,2);  layout2.addWidget(newQLabel("password"),2,1);  layout2.addWidget(password,2,2);  QHBoxLayout\*box=newQHBoxLayout();  box->addStretch(1);  box->addWidget(newQPushButton("登陆"));  layout2.addLayout(box,3,2);  layout2.setColumnStretch(0,1);  layout2.setColumnStretch(3,1);  layout2.setRowStretch(0,1);  layout2.setRowStretch(4,1);  w.setLayout(&layout2); |

## 3.4常用控件介绍

### 3.4.1常用控件需要的头文件加载

|  |
| --- |
| #include<QWidget>  #include<QLabel>//标签页  #include<QLineEdit>//单行文本输入框  #include<QVBoxLayout>  #include<QTextEdit>//多行文本输入框  #include<QPushButton>//按钮  #include<QRadioButton>//单选按钮  #include<QCheckBox>//选择框  #include<QGroupBox>//组合框  #include<QComboBox>//下拉框  #include<QSlider>//滑动器  #include<QSpinBox>//选值框  #include<QDateEdit>//日期输入框  #include<QTimeEdit>//时间输入框  #include<QDateTimeEdit>//日期时间输入框  #include<QLCDNumber>//数码显示器 |
| 控件变量定义： |
| QVBoxLayout\*\_vBox;  QLabel\*\_label;  QLineEdit\*\_lineEdit;  QTextEdit\*\_textEdit;  QPushButton\*\_button;  QRadioButton\*\_radio;  QCheckBox\*\_checkbox;  QGroupBox\*\_groupBox;  QComboBox\*\_combo;  QSlider\*\_slider;  QSpinBox\*\_spinBox;  QDateEdit\*\_dateEdit;  QTimeEdit\*\_timeEdit;  QDateTimeEdit\*\_DTEdit;  QLCDNumber\*\_lcd; |
| 控件初始化： |
| \_vBox=newQVBoxLayout(this);  \_label=newQLabel("label",this);  \_lineEdit=newQLineEdit(this);  \_textEdit=newQTextEdit(this);  \_button=newQPushButton("button",this);  \_radio=newQRadioButton("RadioButton",this);  \_checkbox=newQCheckBox("checkBox",this);  \_groupBox=newQGroupBox("group",this);  \_combo=newQComboBox(this);  \_slider=newQSlider(this);  \_spinBox=newQSpinBox(this);  \_dateEdit=newQDateEdit(this);  \_timeEdit=newQTimeEdit(this);  \_DTEdit=newQDateTimeEdit(this);  \_lcd=newQLCDNumber(this); |
| 在垂直布局种加载控件： |
| \_vBox->addWidget(\_label);  \_vBox->addWidget(\_lineEdit);;  \_vBox->addWidget(\_textEdit);  \_vBox->addWidget(\_button);  \_vBox->addWidget(\_radio);  \_vBox->addWidget(\_checkbox);  \_vBox->addWidget(\_groupBox);  \_vBox->addWidget(\_combo);  \_vBox->addWidget(\_slider);  \_vBox->addWidget(\_spinBox);  \_vBox->addWidget(\_dateEdit);  \_vBox->addWidget(\_timeEdit);  \_vBox->addWidget(\_DTEdit);  \_vBox->addWidget(\_lcd); |

### 3.4.2常用控件使用

|  |
| --- |
| **label**: 标签、显示静态文本和图片 |
| 支持html格式:  //设置文本  QLabel\*label=newQLabel("<h1>lable</h1>");  //设置颜色  QLabel\*label=newQLabel("<fontcolor=red>lable</font>");  //设置链接  label= newQLabel("<a href=www.baidu.com>baidu</a>");  //设置图片  label= newQLabel("<imgsrc=../photo/1.gif></img>");  label->setPixmap(QPixmap("../aaa.png")); |
| 打开链接：为了不写很多槽函数，可以使用C++的lambda表达式，方法在工程管理文件中增加  CONFIG+=C++11  使用的时候相当于将槽函数写成一个匿名函数。 |
| QLabel\*label=newQLabel("<ahref=www.baidu.com>baidu</a>");  vlayout->addWidget(label);  connect(label,&QLabel::linkActivated,[](QStringstr){  qDebug()<<str;  QDesktopServices::openUrl(str);  }); |

|  |
| --- |
| **textEdit** |
| connect(\_textEdit,SIGNAL(textChanged()),this,SLOT(textEditChage())); |
| voidMyWidget::textEditChage()  {  QStringstr=\_textEdit->toPlainText();  if(str.contains("/ku"))  {  str.replace("/ku","<imgsrc=../photo/1.gif></img>");  \_textEdit->setText(str);  }  } |

|  |
| --- |
| **RadioButton、CheckBox、GroupBox** |
| **未分组的时候加载很多radioButton选择的时候只能选一个，可以通过QGroupBox进行分组** |
| QVBoxLayout\*vBox1=newQVBoxLayout(this);  vBox1->addWidget(newQRadioButton("A"));  vBox1->addWidget(newQRadioButton("B"));  vBox1->addWidget(newQRadioButton("C"));  vBox1->addWidget(newQRadioButton("D"));  QVBoxLayout\*vBox2=newQVBoxLayout(this);  vBox2->addWidget(newQRadioButton("A"));  vBox2->addWidget(newQRadioButton("B"));  vBox2->addWidget(newQRadioButton("C"));  vBox2->addWidget(newQRadioButton("D"));  QVBoxLayout\*vBox3=newQVBoxLayout(this);  vBox3->addWidget(newQRadioButton("A"));  vBox3->addWidget(newQRadioButton("B"));  vBox3->addWidget(newQRadioButton("C"));  vBox3->addWidget(newQRadioButton("D"));  QGroupBox\*groupBox=newQGroupBox("第一题",this);  groupBox->setLayout(vBox1);  QGroupBox\*groupBox1=newQGroupBox("第二题",this);  groupBox1->setLayout(vBox2);  QGroupBox\*groupBox2=newQGroupBox("第三题",this);  groupBox2->setLayout(vBox3);  \_vBox->addWidget(groupBox);  \_vBox->addWidget(groupBox1);  \_vBox->addWidget(groupBox2); |

|  |
| --- |
| **ComboBox** |
| **设置它的自动补全的时候自动补全对象初始化要用ComboBox的model()** |
| \_combo=newQComboBox(this);  \_combo->addItem("第一题");  \_combo->addItem("第二题");  \_combo->addItem("第三题");  \_combo->setEditable(true);  //\_combo->setCompleter(newQCompleter(QStringList()<<"123"<<"abc"));  \_combo->setCompleter(newQCompleter(\_combo->model()));  connect(\_combo,SIGNAL(currentIndexChanged(QString)),this,SLOT(comboChanged(QString))); |
| voidMyWidget::comboChanged(QStringstr)  {  qDebug()<<str;  if(str=="第一题")  {  \_textEdit->setText("第一题内容");  }  elseif(str=="第二题")  {  \_textEdit->setText("第二题内容");  }  elseif(str=="第三题")  {  \_textEdit->setText("第三题内容");  }  } |

|  |
| --- |
| **QSlider、QSpinBox** |
| \_slider->setMaximum(100);  \_slider->setMinimum(0);  \_spinBox->setMaximum(100);  \_spinBox->setMinimum(0); |
| connect(\_spinBox,SIGNAL(valueChanged(int)),\_slider,SLOT(setValue(int)));  connect(\_slider,SIGNAL(valueChanged(int)),\_spinBox,SLOT(setValue(int)));} |