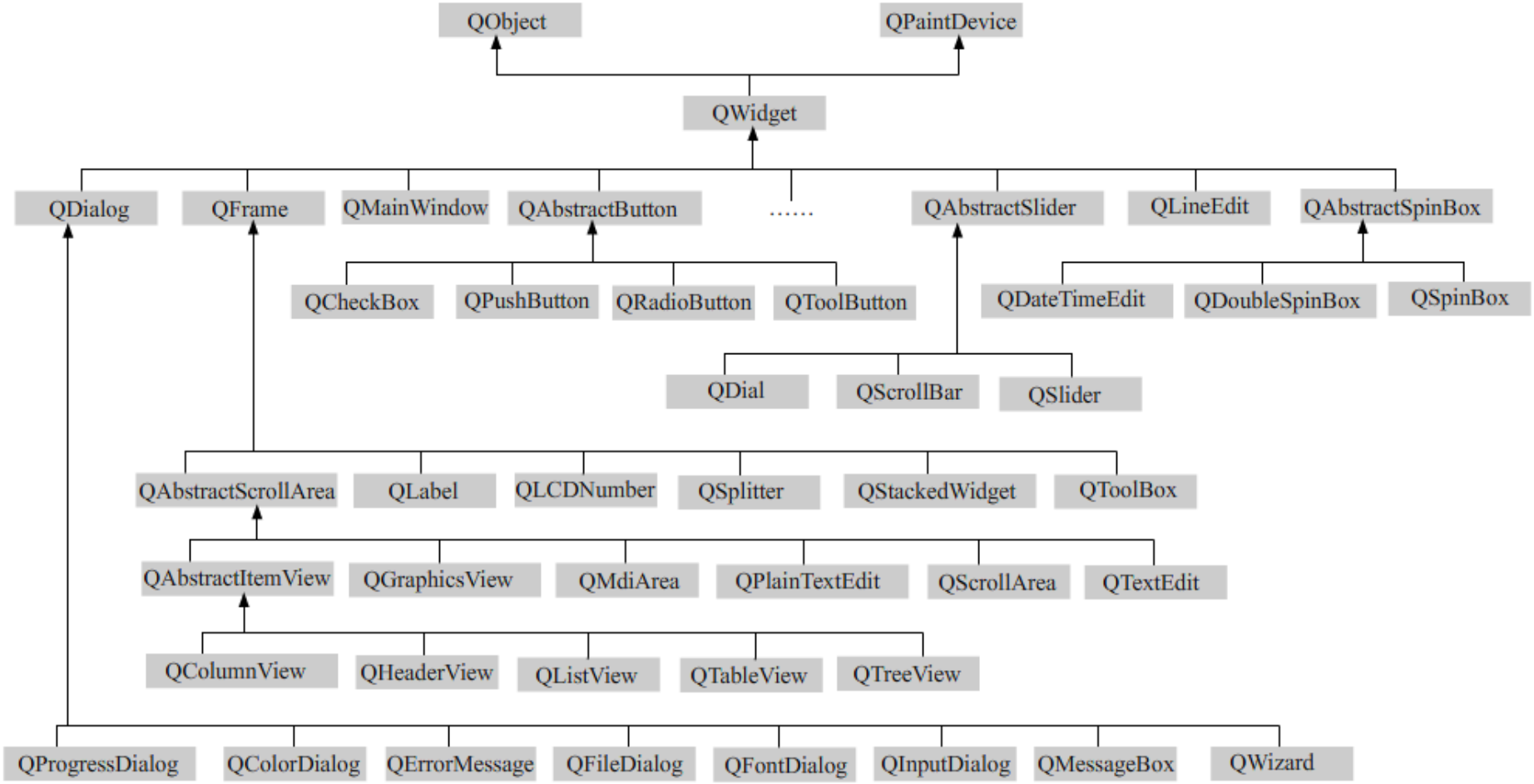
窗口部件

南开大学软件学院
马玲 张圣林

前1节中第一次建立helloworld程序时，曾看到Qt Creator提供的默认基类只有QMainWindow、QWidget和QDialog三种。这三种窗体也是以后用的最多的，QMainWindow是带有菜单栏和工具栏的主窗口类，QDialog是各种对话框的基类，而它们二者全部继承自QWidget。不仅如此，其实所有的窗口部件都继承自QWidget。



主 要 内 容

- 基础窗口部件QWidget
- 对话框QDialog
- 其他窗口部件
- 小结

基础窗口部件QWidget

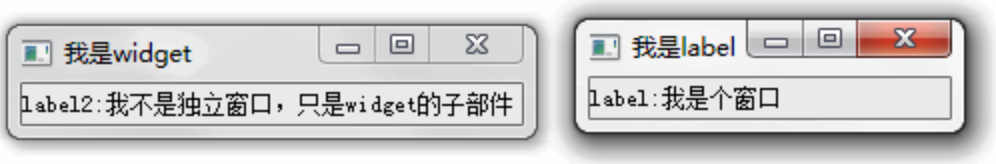
QWidget类是所有用户界面对象的基类，被称为基础窗口部件。QWidget继承自QObject类和QPaintDevice类，其中QObject类是所有支持Qt对象模型（Qt Object Model）的Qt对象的的基类，QPaintDevice类是所有可以绘制的对象的基类。本节内容：

- 窗口、子部件以及窗口类型
- 窗口几何布局
- 程序调试

窗口、子部件以及窗口类型

来看一个代码片段：

```
// 新建QWidget类对象，默认parent参数是0，所以它是个窗口
QWidget *widget = new QWidget();
// 设置窗口标题
widget->setWindowTitle(QObject::tr("我是widget"));
// 新建QLabel对象，默认parent参数是0，所以它是个窗口
QLabel *label = new QLabel();
label->setWindowTitle(QObject::tr("我是label"));
// 设置要显示的信息
label->setText(QObject::tr("label:我是个窗口"));
// 改变部件大小，以便能显示出完整的内容
label->resize(180, 20);
// label2指定了父窗口为widget，所以不是窗口
QLabel *label2 = new QLabel(widget);
label2->setText(QObject::tr("label2:我不是独立窗口，只是widget的子部件"));
label2->resize(250, 20);
// 在屏幕上显示出来
label->show();
widget->show();
```



- 在程序中定义了一个QWidget类对象的指针widget和两个QLabel对象指针label与label2，其中label没有父窗口，而label2在widget中，widget是其父窗口。

- 窗口部件（Widget）这里简称部件，是Qt中建立用户界面的主要元素。像主窗口、对话框、标签，还有以后要介绍到的按钮、文本输入框等都是窗口部件。

- 在Qt中，把没有嵌入到其他部件中的部件称为窗口，一般的，窗口都有边框和标题栏，就像程序中的widget和label一样。

- QMainWindow和大量的QDialog子类是最一般的窗口类型。窗口就是没有父部件的部件，所以又称为顶级部件（top-level widget）。与其相对的是非窗口部件，又称为子部件（child widget）。在Qt中大部分部件被用作子部件，它们嵌入在别的窗口中，例如程序中的label2。

窗口类型

前面讲到窗口一般都有边框和标题栏，其实这也不是必需的：

- QWidget的构造函数有两个参数：QWidget * parent = 0和Qt::WindowFlags f = 0；
- 前面的parent就是指父窗口部件，默认值为0，表明没有父窗口；
- 而后面的f参数是Qt::WindowFlags类型的，它是一个枚举类型，分为窗口类型（WindowType）和窗口标志（WindowFlags。前者可以定义窗口的类型，比如我们这里f=0，表明使用了Qt::Widget一项，这是QWidget的默认类型，这种类型的部件如果有父窗口，那么它就是子部件，否则就是独立的窗口。

例如：使用其中的Qt::Dialog和Qt::SplashScreen，更改程序中的新建对象的那两行代码：

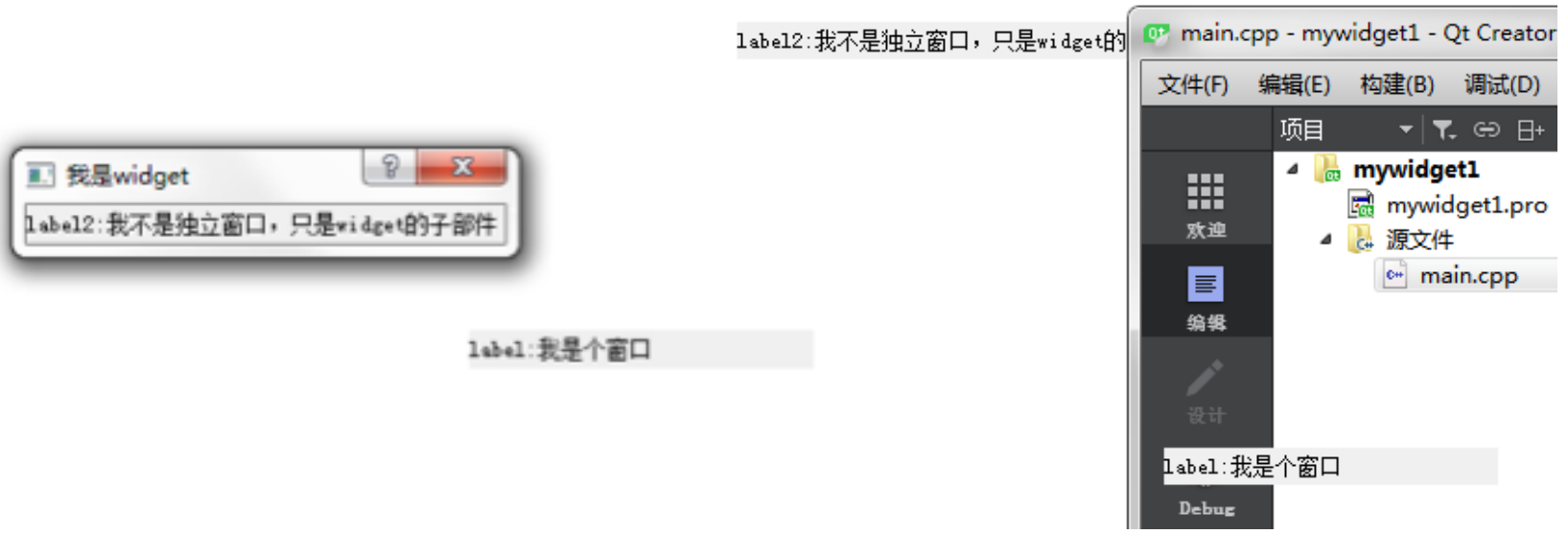
```
QWidget *widget = new QWidget(0, Qt::Dialog);
QLabel *label = new QLabel(0, Qt::SplashScreen);
```

当更改窗口类型后，窗口的样式发生了改变，一个是对话框类型，一个是欢迎窗口类型。

而对于窗口标志，它主要的作用是更改窗口的标题栏和边框，而且它们可以和窗口类型进行位或操作。下面再次更改那两行代码：

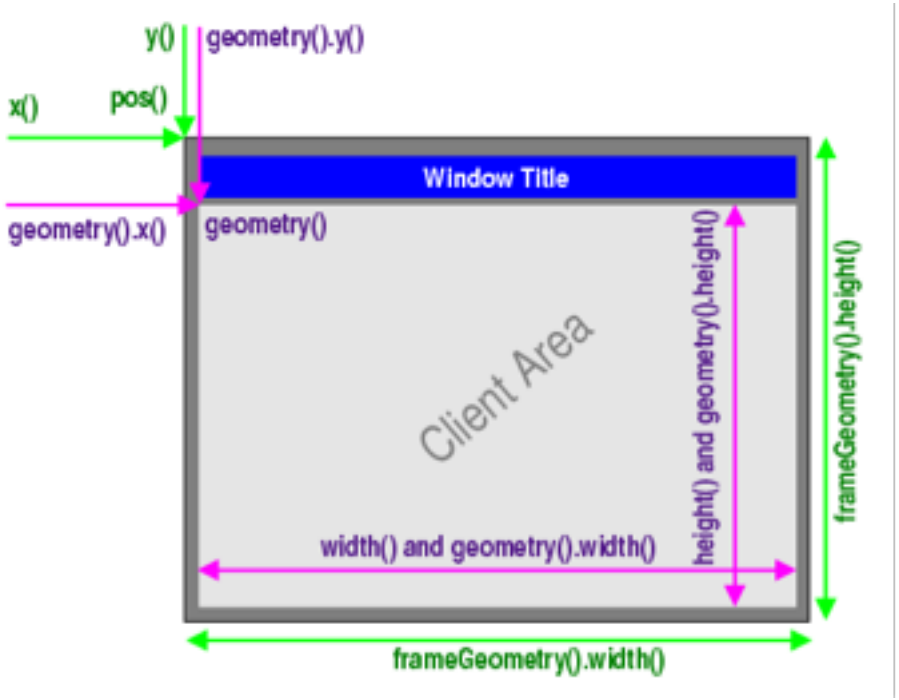
```
QWidget *widget = new QWidget(0, Qt::Dialog | Qt::FramelessWindowHint);
QLabel *label = new QLabel(0, Qt::SplashScreen | Qt::WindowStaysOnTopHint);
```

Qt::FramelessWindowHint用来产生一个没有边框的窗口，而Qt::WindowStaysOnTopHint用来使该窗口停留在所有其它窗口上面。



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它们的数值。



这里的函数分为两类，一类是包含框架的，一类是不包含框架的：

- 包含框架：x()、y()、frameGeometry()、pos()和move()等函数；
- 不包含框架：geometry()、width()、height()、rect()和size()等函数。

程序调试

下面在讲解窗口几何布局的几个函数的同时，讲解一下程序调试方面的内容。

将主函数内容更改如下：

```
#include <QApplication>
#include <QWidget>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QWidget widget;
    int x = widget.x();
    int y = widget.y();
    QRect geometry = widget.geometry();
    QRect frame = widget.frameGeometry();
    return a.exec();
}
```

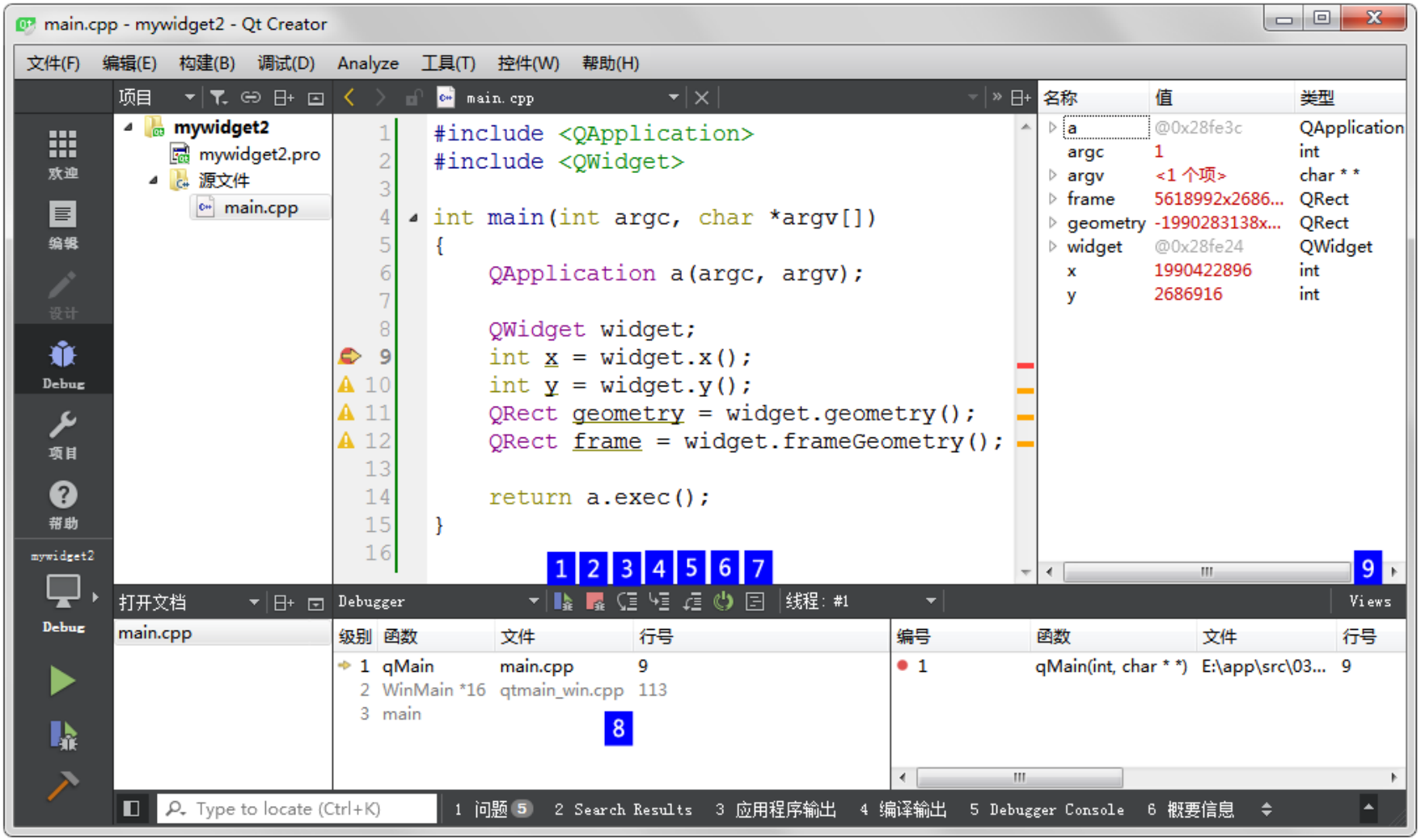
x()、y()分别返回部件的位置坐标的x、y值，它们的默认值为0。

而geometry()和frameGeometry()函数分别返回没有边框和包含边框的窗口框架矩形的值，其返回值是QRect类型的，就是一个矩形，它的形式是（位置坐标，大小信息），也就是（x，y，宽，高）。

下面在int x = widget.x(); 一行代码的标号前面点击鼠标左键来设置断点。

所谓断点，就是程序运行到该行代码时会暂停下来，从而可以查看一些信息，如变量值等。要取消断点，只要在那个断点上再点击一下就可以了。设置好断点后便可以按下F5或者左下角的调试按钮开始调试。这时程序会先进行构建再进入调试模式，这个过程可能需要一些时间。在程序构建时可能会出现警告，那是因为我们定义了变量却没有使用造成的，不用管它。

调试模式


main.cpp - mywidget2 - Qt Creator
文件(F) 编辑(E) 构建(B) 调试(D) Analyze 工具(T) 控件(W) 帮助(H)
项目
mywidget2
mywidget2.pro
源文件
main.cpp
main.cpp
#include <QApplication>
#include <QWidget>
int main(int argc, char *argv[])
{
QApplication a(argc, argv);
QWidget widget;
int x = widget.x();
int y = widget.y();
QRect geometry = widget.geometry();
QRect frame = widget.frameGeometry();
return a.exec();
}
名称 值 类型
a @0x28fe3c QApplication
argc 1 int
argv <1 个项> char * *
frame 5618992x2686... QRect
geometry -1990283138x... QRect
widget @0x28fe24 QWidget
x 1990422896 int
y 2686916 int
1 2 3 4 5 6 7
9
8
欢迎
编辑
设计
Debug
项目
帮助
mywidget2
Debug
打开文档
main.cpp
Debugger
线程: #1
Views
级别 函数 文件 行号
1 qMain main.cpp 9
2 WinMain *16 qtmain_win.cpp 113
3 main
编号 函数 文件 行号
1 qMain(int, char * *) E:\app\src\03... 9
Type to locate (Ctrl+K)
1 问题 5
2 Search Results
3 应用程序输出
4 编译输出
5 Debugger Console
6 概要信息

下面对调试模式的几个按钮和窗口进行简单介绍：

①继续按钮。程序在断点处停了下来，按下继续按钮后，程序便会像正常运行一样，执行后面的代码，直到遇到下一个断点，或者程序结束。

②停止调试按钮。按下该按钮后结束调试。

③单步跳过按钮。直接执行本行代码，然后指向下一行代码。

④单步进入按钮。进入调用的函数内部。

⑤单步跳出按钮。当进入函数内部时，跳出该函数，一般与单步进入配合使用。

⑥重新启动调试会话。

⑦显示源码对应的汇编指令，并可以单步调试。

⑧堆栈视图。这里显示了从程序开始到断点处，所有嵌套调用的函数所在的源文件名和行号。

⑨其它视图。这里可以选择多种视图。

单步调试

- 点击一下“单步进入”按钮，或者按下F11，这时，程序会跳转到QWidget类的x()函数的源码处，这里对这个函数不做过多讲解，下面直接按下“单步跳出”按钮回到原来的断点处。然后便开始一直按“单步跳过”按钮，单步执行程序，并查看局部变量和监视器视图中相应变量值的变化情况。等执行到最后一行代码return a.exec();时，按下“停止调试”按钮，结束调试。

- 这里要补充说明一下，我们在程序调试过程中可以进入到Qt类的源码中，其实还有一个很简单的方法也可以实现这个功能，就是在编辑器中将鼠标光标定位到一个类名或者函数上，然后按下F2键，或者点击鼠标右键，选择“跟踪光标位置的符号”，这时编辑器就会跳转到其源码处。

- 从变量监视器中可以看到x、y、geometry和frame四个变量初始值都是一个随机未知数。等到调试完成后，x、y的值均为0，这是它们的默认值。而geometry的值为640x480+0+0，frame的值为639x479+0+0。

- 现在对这些值还不是很清楚，不过，为什么x、y的值会是0呢？我们可能会想到，应该是窗口没有显示的原因，那么就更改代码，让窗口先显示出来，再看这些值。在QWidget widget;一行代码后添加一行代码：

```
widget.show();
```

- 现在再次调试程序，这时会发现窗口只显示了一个标题栏，先不管它，继续在Qt Creator中点击“单步跳过”按钮。当我们将程序运行到最后一行代码return a.exec();时，再次按下“单步跳过”按钮后，程序窗口终于显示出来了。这是因为只有程序进入主事件循环后才能接收事件，而show()函数会触发显示事件，所以只有在完成a.exe()函数调用进入消息循环后才能正常显示。这次看到几个变量的值都有了变化，但是这时还是不清楚这些值的含义。

- 注意：因为使用调试器进行调试要等待一段时间，而且步骤很麻烦，对于初学者来说，如果按错了按钮，还很容易出错。 所以，并不推荐初学者使用。

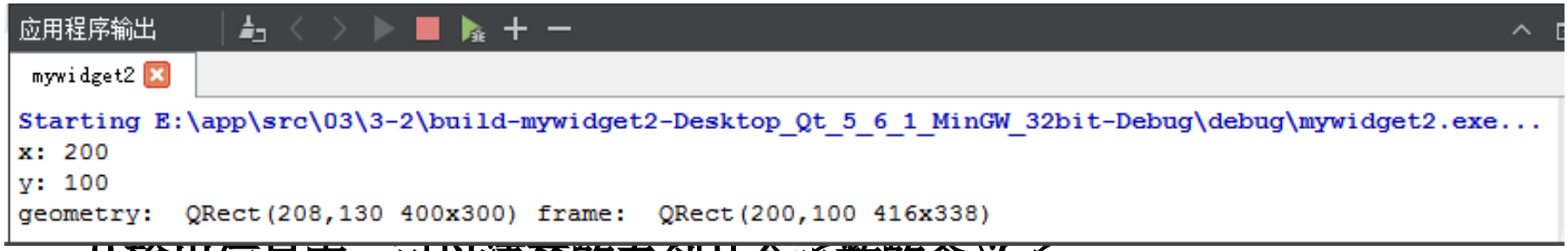
使用qDebug()函数

一般在程序调试过程中很常用的是qDebug()函数，它可以将调试信息直接输出到控制台，在Qt Creator中是输出到应用程序输出栏 。例如：

```
QWidget widget;
widget.resize(400, 300);          // 设置窗口大小
widget.move(200, 100);            // 设置窗口位置
widget.show();
int x = widget.x();
qDebug("x: %d", x);               // 输出x的值
int y = widget.y();
qDebug("y: %d", y);
QRect geometry = widget.geometry();
QRect frame = widget.frameGeometry();
qDebug() << "geometry: " << geometry << "frame: " << frame;
```

要使用qDebug()函数，就要添加#include <QDebug>头文件。然后这里使用了两种输出方式：

- 方式一：直接将字符串当做参数传给qDebug()函数，例如上面使用这种方法输出x和y的值。
- 方式二：使用输出流的方式一次输出多个值，它们的类型可以不同，例如程序中输出geometry和frame的值。
- 需要说明的是，如果只使用第一种方法，那么是不需要添加<QDebug>头文件的，如果使用第二种方法就必须添加这个头文件。因为第一种方法很麻烦，所以经常使用的是第二种方法。



- 从输出信息中，可以清楚的看到几个函数的含义了。

- 其实使用qDebug()函数的第二种方法时还可以让输出自动换行，下面来看一下其他几个函数的用法。在return a.exec();一行代码前添加如下代码：

```
qDebug() << "pos:" << widget.pos() << endl << "rect:" << widget.rect()
                << endl << "size:" << widget.size() << endl << "width:"
                    << widget.width() << endl<< "height:" << widget.height();
```

- 这里的“endl”就是起换行作用的。
- 根据程序的输出结果，可以很明了的看到这些函数的作用。
- 其中pos()函数返回窗口的位置，是一个坐标值，上面的x()、y()函数返回的就是它的x、y坐标值；
- rect()函数返回不包含边框的窗口内部矩形，在窗口内部，左上角是(0,0)点；
- size()函数返回不包含边框的窗口大小信息；
- width()和height()函数分别返回窗口内部的宽和高。
- 从数据可以看到，前面使用的调整窗口大小的resize()函数是设置的不包含边框的窗口大小。

对话框QDialog

本节先从对话框的介绍讲起，然后讲述两种不同类型的对话框，再讲解一个有多个窗口组成并且窗口间可以相互切换的程序，最后介绍一下Qt提供的几个标准对话框。

- 模态和非模态对话框
- 多窗口切换
- 标准对话框

模态和非模态对话框

- 模态对话框就是在我们没有关闭它之前，不能再与同一个应用程序的其他窗口进行交互，比如新建项目时弹出的对话框。要想使一个对话框成为模态对话框，只需要调用它的exec()函数：

```
QDialog dialog(this);
dialog.exec();
```

- 而对于非模态对话框，既可以与它交互，也可以与同一程序中的其他窗口交互，例如Microsoft Word中的查找替换对话框。要使一个对话框成为非模态对话框，我们就可以使用new操作来创建，然后使用show()函数来显示。

```
QDialog *dialog = new QDialog(this);
dialog->show();
```

使用show()函数也可以建立模态对话框，只需在其前面使用setModal()函数即可。例如：

```
QDialog *dialog = new QDialog(this);
dialog->setModal(true);
dialog->show();
```

现在运行程序，可以看到生成的对话框是模态的。但是，它与用exec()函数时的效果是不一样的。这是因为调用完show()函数后会立即将控制权交给调用者，那么程序可以继续往下执行。而调用exec()函数却不是这样，它只有当对话框被关闭时才会返回。

与setModal()函数相似的还有一个setWindowModality()函数，它有一个参数来设置模态对话框要阻塞的窗口类型，可以是：

- Qt::NonModal（不阻塞任何窗口，就是非模态），
- 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）。

而setModal()函数默认设置的是Qt::ApplicationModal。

多窗口切换

本节会涉及如下内容：

- 开始认识信号和槽
- 信号和槽的关联方式
- 从登陆对话框显示主界面的方法

开始认识信号和槽

在Qt中使用信号和槽机制来完成对象之间的协同操作。

简单来说，信号和槽都是函数，比如按下窗口上的一个按钮后想要弹出一个对话框，那么就可以将这个按钮的单击信号和我们定义的槽关联起来，在这个槽中可以创建一个对话框，并且显示它。这样，当单击这个按钮时就会发射信号，进而执行我们的槽来显示一个对话框。

关联方式一：使用connect()关联

- mywidget.h文件写上槽的声明：

```
public slots:
void showChildDialog();
```

- 在mywidget.cpp文件中将槽的实现：

```
void MyWidget::showChildDialog()
{
   QDialog *dialog = new QDialog(this);
   dialog->show();
}
```

- 在mywidget.cpp文件的MyWidget类的构造函数中使用connect()关联按钮单击信号和自定义的槽如下：

```
connect(ui->showChildButton, &QPushButton::clicked,
        this, &MyWidget::showChildDialog);
```

说明：

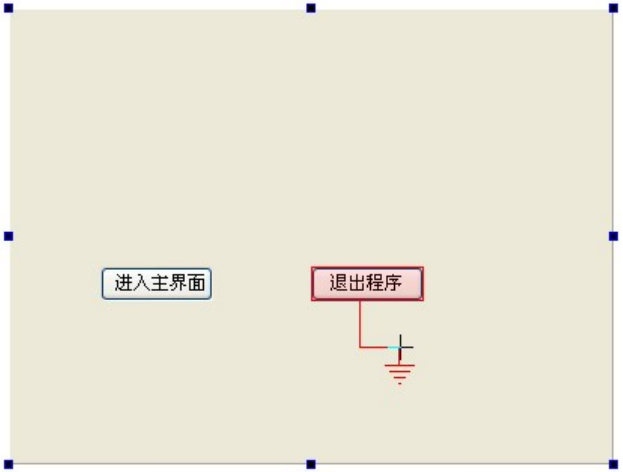
- 这里自定义了一个槽，槽一般使用slots关键字进行修饰（Qt 4中必须使用，Qt 5使用新connect语法时可以不用，为了与一般函数进行区别，建议使用），这里使用了public slots，表明这个槽可以在类外被调用。

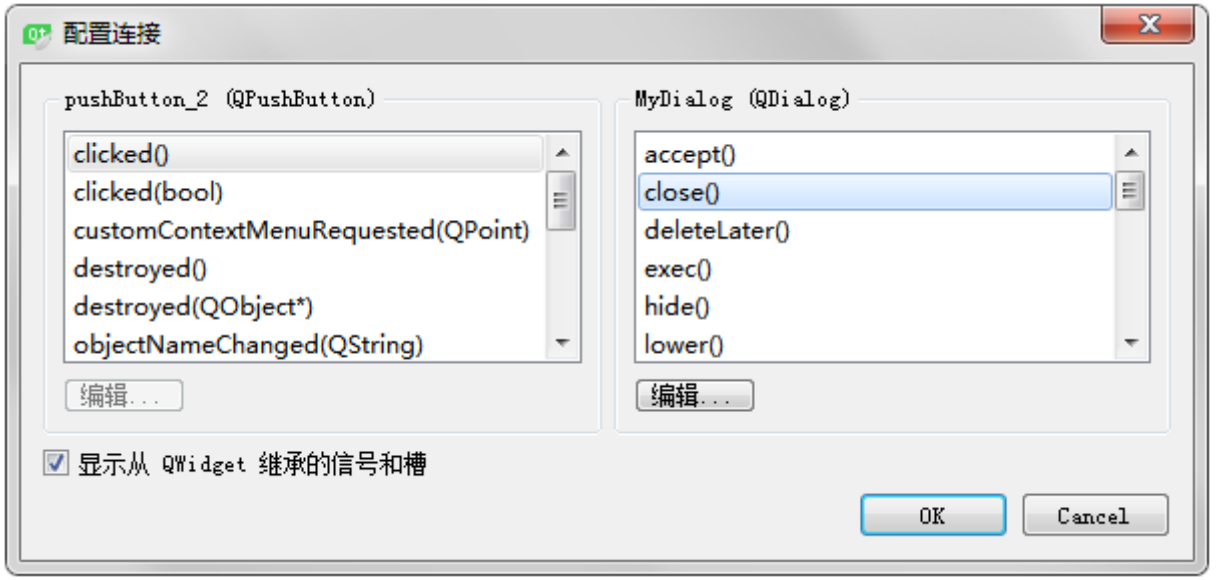
- clicked()信号在QPushButton类中进行了定义，而connect()是QObject类中的函数，因为我们的类继承自QObject，所以可以直接使用它。

- connect()函数中的四个参数分别是：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。

关联方式二：在设计模式关联

- 首先添加自定义对话框类MyDialog。在设计模式中向窗口上添加两个Push Button，并且分别更改其显示文本为“进入主界面”和“退出程序”。
- 点击设计器上方的“编辑信号/槽”图标，或者按下快捷键F4，这时便进入了部件的信号和槽的编辑模式。在“退出程序”按钮上按住鼠标左键，然后拖动到窗口界面上，这时松开鼠标左键。
- 在弹出的配置连接对话框中，选中下面的“显示从QWidget继承的信号和槽”选项，然后在左边的QPushButton栏中选择信号clicked()，在右边的QDialog栏中选择对应的槽close()，完成后按下“确定”。

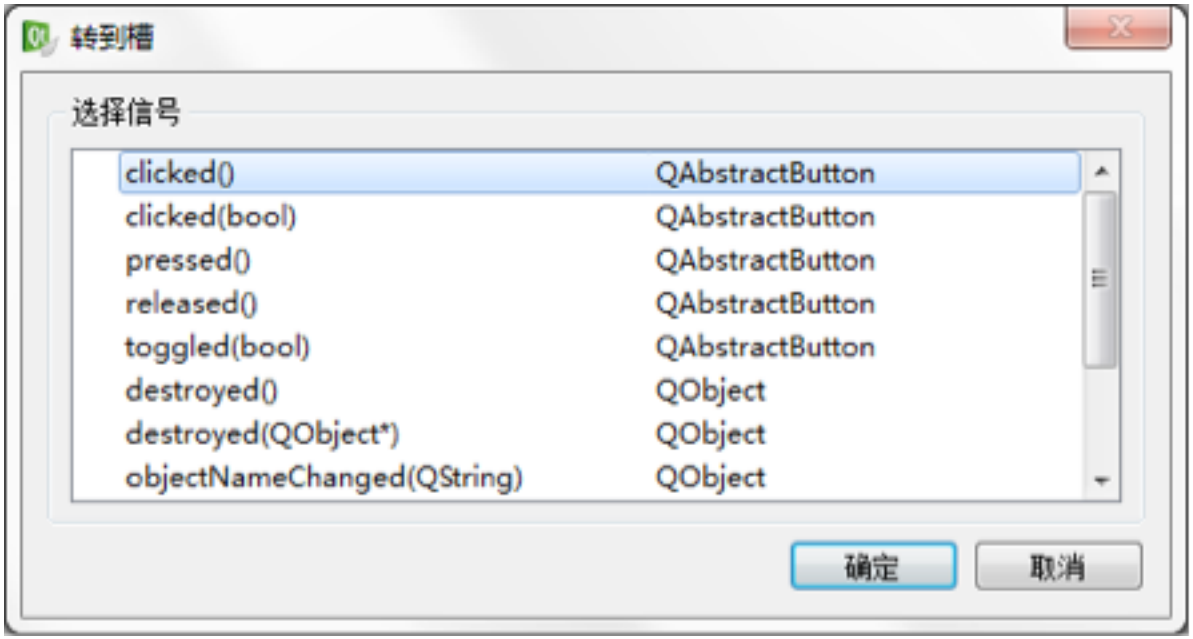




关联方式三：自动关联

在“进入主界面”按钮上右击，在弹出的菜单上选择“转到槽”，然后在弹出的对话框中选择clicked()信号，并按“确定”。这时便会进入代码编辑模式，并且定位到自动生成的on_pushButton_clicked()槽中。在其中添加代码：

```
void MyDialog::on_pushButton_clicked()
{    accept();   }
```



- 自动关联就是将关联函数整合到槽命名中。

- 例如on_pushButton_clicked()就是由字符“on”和发射信号的部件对象名，还有信号名组成。这样就可以去掉那个connect()关联函数了。每当pushButton被按下，就会发射clicked()信号，然后就会执行on_pushButton_clicked()槽。

- 这里accept()函数是QDialog类中的一个槽，对于一个使用exec()函数实现的模态对话框，执行了这个槽，就会隐藏这个模态对话框，并返回QDialog::Accepted值，我们就是要使用这个值来判断是哪个按钮被按下了。与其对应的还有一个reject()槽，它可以返回一个QDialog::Rejected值。其实，前面的“退出程序”按钮也可以关联这个槽。

使用自定义对话框登陆主界面

在main()函数中：

```
QApplication a(argc, argv);
  MyWidget w;
  MyDialog dialog;                    // 新建MyDialog类对象
  if(dialog.exec()==QDialog::Accepted){  // 判断dialog执行结果
     w.show();          // 如果是按下了“进入主界面”按钮，则显示主界面
     return a.exec();                 // 程序正常运行
  }
  else return 0;
```

在主函数中建立了MyDialog对象，然后判断其exec()函数的返回值，如果是按下了“进入主界面”按钮，那么返回值应该是QDialog::Accepted，就显示主界面，并且正常执行程序。如果不是，则直接退出程序。

标准对话框

Qt提供了一些常用的对话框类型，它们全部继承自QDialog类，并增加了自己的特色功能，比如获取颜色、显示特定信息等。

- 颜色对话框
- 文件对话框
- 字体对话框
- 输入对话框
- 消息对话框
- 进度对话框
- 错误信息对话框
- 向导对话框

颜色对话框

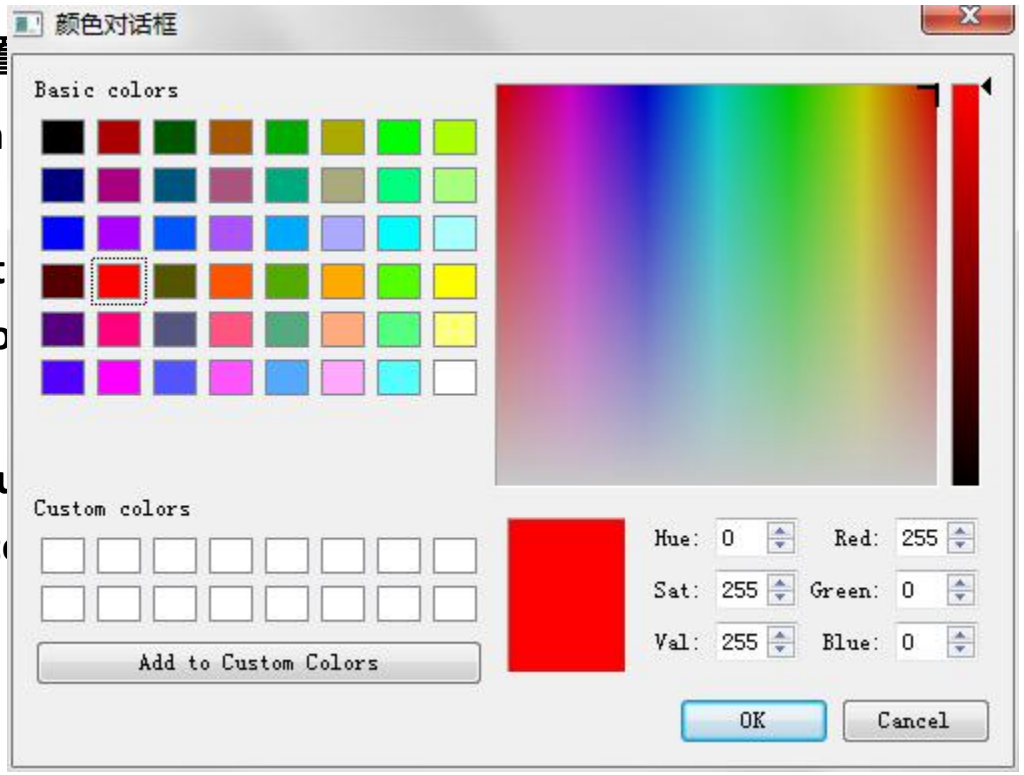
例如：

```
QColor color = QColorDialog::getColor(Qt::red, this, tr("颜色对话框"));
qDebug() << "color: " << color;
```

这里使用了QColorDialog的静态函数getColor()来获取颜色，它的三个参数的作用分别是：设置初始颜色、父窗口和对话框标题。这里的Qt::red，是Qt预定义的颜色对象。

- 如果想要更灵活的设置

```
void MyWidget::on_push
{
    QColorDialog dialog(Qt
    dialog.setOption(QColo
    dialog.exec();
    QColor color = dialog.cu
    qDebug()<<"color:"<<c
}
```



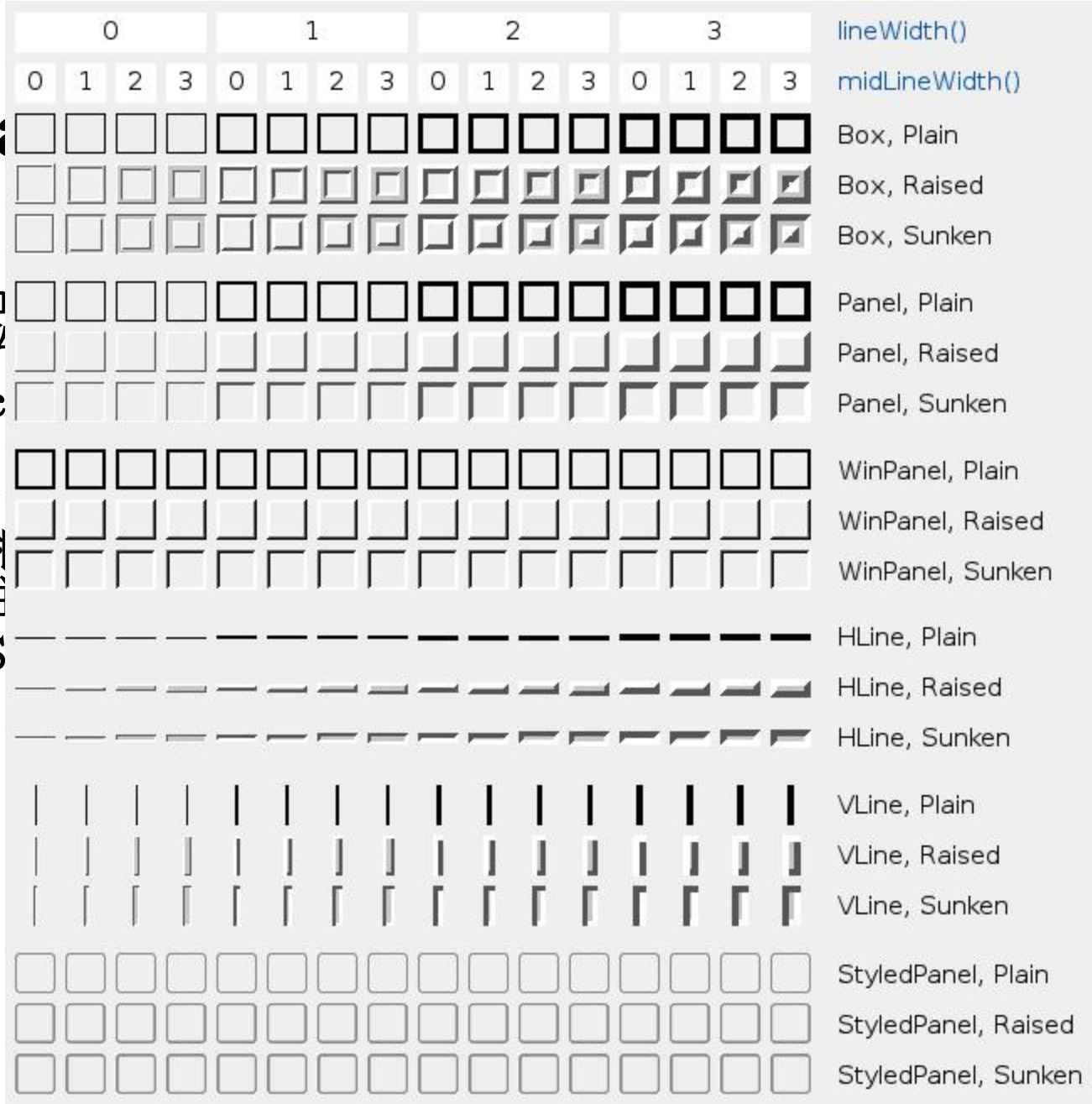
其他窗口部件

Qt提供了一些常用的窗口部件：

- QFrame类族
- 按钮部件
- 行编辑器
- 数值设定框
- 滑块部件

QFra

- QFram 的标
  签部件
  QStac
- 带边框 Frame
  类的主 框形
  状（S



例：QLabel

- 除了最常用的显示文本外，还可以显示图片：

```
ui->label->setPixmap(QPixmap("F:/logo.png"));
```

- 还可以显示gif动态图片：

```
QMovie *movie = new QMovie("F:/donghua.gif");
ui->label->setMovie(movie);        // 在标签中添加动画
movie->start();
```

按钮部件

QAbstractButton类是按钮部件的抽象基类，提供了按钮的通用功能。它的子类包括：

- 复选框QCheckBox
- 标准按钮QPushButton
- 单选框按钮QRadioButton
- 工具按钮QToolButton。

例：QPushButton

在下面代码里为三个按钮改变了显示文本，这里在一个字母前加上“&”符号，那么就可以将这个按钮的加速键设置为Alt加上这个字母。如果我们要在文本中显示“&”符号本身，那么可以使用“&&”。我们也可以使用setIcon()函数来给按钮添加图标，这里图片文件使用了相对路径（当然这个也可以在设计模式通过更改icon属性来实现）。对于pushBtn3，我们为其添加了下拉菜单，当然，现在这个菜单什么功能也没实现。

```
ui->pushBtn1->setText(tr("&nihao"));      // 这样便指定了Alt+N为加速键
ui->pushBtn2->setText(tr("帮助(&H)"));
ui->pushBtn2->setIcon(QIcon("../image/help.png"));
ui->pushBtn3->setText(tr("z&oom"));
QMenu *menu = new QMenu(this);
menu->addAction(QIcon("../ima
ui->pushBtn3->setMenu(menu);
```

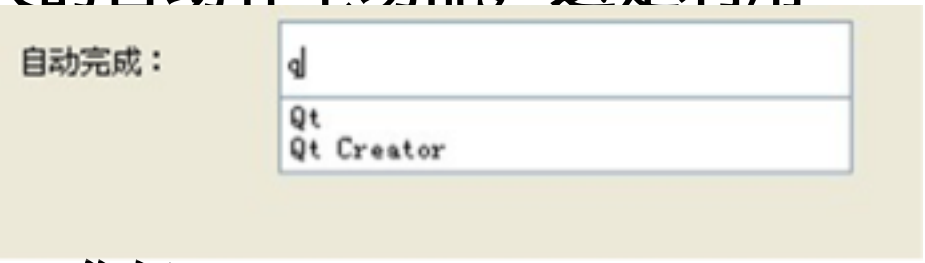
行编辑器

行编辑器QLineEdit部件是一个单行的文本编辑器，它允许用户输入和编辑单行的纯文本内容，而且提供了一系列有用的功能，包括撤销与恢复、剪切和拖放等操作。 例如：

- **显示模式**
- **输入掩码**
- **输入验证**
- **自动补全**

例：自动补全功能

在QLineEdit中也提供了强大的自动补全功能，这是利用QCompleter类实现的：




```
QStringList wordList;
wordList << "Qt" << "Qt Creator" << tr("你好");
// 新建自动完成器
QCompleter *completer = new QCompleter(wordList, this);
// 设置大小写不敏感
completer->setCaseSensitivity(Qt::CaseInsensitive);
ui->lineEdit4->setCompleter(completer);
```

数值设定框

QAbstractSpinBox类是一个抽象基类，它提供了一个数值设定框和一个行编辑器来显示设定值。它有三个子类：

- QDateTimeEdit（日期时间设定）
- QSpinBox（整数设定）
- QDoubleSpinBox（浮点数的设定）

例： QDateTimeEdit

下面的代码设置了dateTimeEdit中的日期和时间。这里简单说明一下：y表示年；M表示月；d表示日，而ddd表示星期；H表示小时，使用24小时制显示，而h也表示小时，如果最后有AM或者PM的，则是12小时制显示，否则使用24小时制；m表示分；s表示秒；还有一个z可以用来表示毫秒。

```
// 设置时间为现在的系统时间
ui->dateTimeEdit->setDateTime(QDateTime::currentDateTime());
// 设置时间的显示格式
ui->dateTimeEdit->setDisplayFormat(
                    tr("yyyy年MM月dd日ddd HH时mm分ss秒"));
```

滑块部件

QAbstractSlider类提供了一个区间内的整数值，它有一个滑块，可以定位到一个整数区间的任意值。这个类是一个抽象基类，它有三个子类QScrollBar，QSlider和QDial。其中：

- 滚动条QScrollBar更多的是用在QScrollArea类中来实现滚动区域；
- 而QSlider是我们最常见的音量控制或多媒体播放进度等滑块；
- QDial是一个刻度表盘。

小结

本节讲述了众多常用的窗口部件的使用方法，其中还涉及了程序调试和信号和槽等知识。学习本章，一定要多实践，多运用各个部件进行编程，才能真正掌握！