Using a Designer UI File in Your Application

Qt 设计师 UI 文件表示 XML 格式的表单控件树。这些表格可以进行处理：

- 在编译时，可以转换为可被编译的 C ++代码。
- 在运行时，文件可被 QUiLoader 类处理，解析 XML 文件时该类动态地构造插件树。

编译时表单处理

您可以使用 Qt Designer 创建用户界面组件，用 Qt 集成构建工具 qmake 和 UIC 生成代码。生成的代码包含表单的用户界面对象。它是一个包含一个 C ++结构：

- 指针到表单控件，布局，布局项目，按钮组，和动作。
- 调用 setupUi()函数在父控件上创建控件树。
- 调用 retranslateUi()成员函数处理表单字符串属性的转换。欲了解更多信息，请参见 Reacting to Language Changes 文档。

你的应用程序包含生成的代码，并可以直接使用。或者，你可以用它来扩展标准控件的子类。

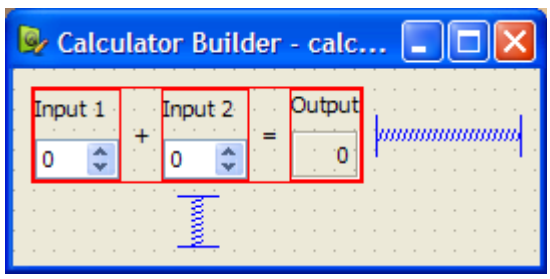
编译时处理形式可以和使用下列方法之一一起使用：

- 直接方法：构造一个 widget 作为一个占位符组件使用，并设置用户界面。
- 单一继承方法：使用 widget 的基类（ QWidget 或 QDialog，例如），包括表单用户界面对象的私有实现。
- 多重继承的方法：您既继承窗体的基类，又包含表单的用户界面对象。

为了演示，我们创建了一个简单的计算器应用程序。它是基于原始计算器表格的例子。

该应用程序包含一个源文件、main.cpp 中和一个 UI 文件。

calculatorform.ui 文件使用 Qt 设计师来设计：



我们将使用的 qmake 生成可执行文件，所以我们需要写一个.pro 文件：

```
HEADERS= calculatorform.h
```

此文件的功能是告诉 qmake 定义了哪一个文件。在这种情况下，calculatorform.ui 文件被用来创建可被任何文件列表使用的 ui_calculatorform.h 文件。

注意：您可以使用 Qt Creator 创建计算器表单项目。它会自动生成 main.cpp、用户界面和的.pro 文件，然后你就可以修改。

- 直接方法

要使用直接的方式，我们直接在 main.cpp 里包含 ui_calculatorform.h 文件：

```
#include "ui_calculatorform.h"
```

主要功能是创建计算机控件，该控件通过 QWidgetcalculatorform.ui 文件创建一个标准用户界面。

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *widget = new QWidget;
    Ui::CalculatorForm ui;
    ui.setupUi(widget);

    widget->show();
    return app.exec();
}
```

在这种情况下，UI :: CalculatorForm 是设置所有对话框控件以及信号和槽之间的连接的 ui_calculatorform.h 文件的接口描述对象。

直接的方法快速，使用方法简单，应用程序组件自包含。然而，Qt Designer 创建 componens 往往需要与应用程序代码的其余部分紧密结合。例如，上面提供的 CalculatorForm 代码编译和运行，但 QSpinBox 对象将不与的 QLabel 互动，因为我们需要一个自定义槽增加操作以及显示 QLabel 中的结果。为了实现这一目标，我们需要使用单继承的方法。

- 单一继承方法

要使用单继承的方法，我们继承一个标准的 Qt 物件，包括表单用户界面对象的私有实例。这可以采取以下形式：

- 成员变量
- 指针成员变量

『使用一个成员变量』

在这种方法中，我们继承了一个 Qt 窗口小控件，并设置了用户界面。所生成的 UI ::

CalculatorForm 结构是类的一个成员。

这种方法是用在 Calculator Form 例子中。

为了确保我们可以使用用户界面，我们需要包括之前 UIC 生成的头文件：

```
#include "ui_calculatorform.h"
```

这意味着，.pro 文件必须包含 calculatorform.h：

```
HEADERS     = calculatorform.h
```

子类是通过以下方式定义：

```
class CalculatorForm : public QWidget
{
   Q_OBJECT

public:
   CalculatorForm(QWidget *parent = 0);

private slots:
   void on_inputSpinBox1_valueChanged(int value);
   void on_inputSpinBox2_valueChanged(int value);

private:
   Ui::CalculatorForm ui;
};
```

类的重要特征是私有的 ui 对象，它提供了用于设置的代码和管理的用户接口。

子类结构、控件和布局的配置以及对话框的布局都是调用 setupUi()函数来构造。一旦这已经完成，就可以根据需要来修改用户界面。

```
CalculatorForm::CalculatorForm(QWidget *parent)
   : QWidget(parent)
{
   ui.setupUi(this);
}
```

前缀 - 我们可以在用户界面控件连接信号和槽，槽函数通过在信号前面添加 on_<object name>前缀区分。欲了解更多信息，请参阅 widgets-and-dialogs-with-auto-connect 文档。

这种方法的优点是它使用简单，它的继承是提供一种基于 QWidget 的。我们可以用这种方法在同一控件内定义的若干用户界面，其中的每一个包含在其自己的名称空间，并覆盖（或组合）它们。例如，这种方法可以被用来创建从现有表单各个选项卡。

『使用指针成员变量』

可替代地，UI:: CalculatorForm 结构可制成类的一个指针成员。看起来如下：

```
namespace Ui {
   class CalculatorForm;
}

class CalculatorForm : public QWidget
...
virtual ~CalculatorForm();
...
private:
   Ui::CalculatorForm *ui;
....
```

相应的源文件如下所示：

```
#include "ui_calculatorform.h"

CalculatorForm::CalculatorForm(QWidget *parent) :
   QWidget(parent), ui(new Ui::CalculatorForm)
{
   ui->setupUi(this);
}

CalculatorForm::~CalculatorForm()
{
   delete ui;
}
```

这种方法的优点是，用户接口对象可以被前置声明，这意味着我们没有必要在头中包含 ui_calculatorform.h 文件。当文件发生变化时，这种形式不需要重新编译代码，这一点尤其重要。

我们一般建议使用这种方法开发库和大型应用程序。欲了解更多信息，请参阅 Creating Shared Libraries。

- 多重继承方法

使用 Qt Designer 创建的控件与标准 QWidget 基类中的子类彼此间成为子类。这种方法使得所有用户界面组件在子类的范围内直接访问，并允许信号和槽的连接，以在与 connect（）函数以通常的方式进行。

这种方法是用在多重继承的例子。

我们需要包括 ui_calculatorform.h 头文件，内容如下：

```
#include "ui_calculatorform.h"
```

以类似单继承的方式定义类，不同之处在于这次我们从 QWidget 和 UI:: CalculatorForm 两个类中继承，如下：

```
class CalculatorForm : public QWidget, private Ui::CalculatorForm
{
   Q_OBJECT

public:
   CalculatorForm(QWidget *parent = 0);

private slots:
   void on_inputSpinBox1_valueChanged(int value);
   void on_inputSpinBox2_valueChanged(int value);
};
```

我们私有继承 UI:: CalculatorForm 以确保用户界面对象在我们的子类中是私有的。我们也可以用同样的方式公共或保护继承。

该子类如单继承示例中一样执行许多相同的任务：

```
CalculatorForm::CalculatorForm(QWidget *parent)
   : QWidget(parent)
{
   setupUi(this);
}
```

在这种情况下，用户界面中使用的控件能以用手代码创建控件相同的方式进行访问。我们不再需要 UI 前缀来访问它们。

如果用户界面变化，Qt 通过发送一个 QEvent:: LanguageChange 类型的事件通知应用程序。为了调用用户界面对象的成员函数 retranslateUi（），我们重新实现了 QWidget:: changeEvent 中（）类，如下所示：

```
void CalculatorForm::changeEvent(QEvent *e)
{
   QWidget::changeEvent(e);
   switch (e->type()) {
   case QEvent::LanguageChange:
      ui->retranslateUi(this);
      break;
   default:
      break;
  }
}
```

运行时表单处理

可替换地，表单可以在运行时被处理，产生 dynamically-生成的用户界面。这可以通过使用 QtUiTools 模块完成，该模块提供了 QUiLoader 类处理 Qt Designer 的内容。

『UiTools 方法』

处理方式在运行时需要包含 UI 文件中的资源文件。此外，应用程序需要被配置为使用 QtUiTools 模块。这是通过在一个 Qmake 工程文件中包含以下声明完成的，以确保应用程序编译和链接正确。

```
QT += uitools
```

QUiLoader 类提供了一个形式加载对象构造用户界面。该用户界面可以从任何 QIODevice 进行检索，例如，QFile 对象，以获得在工程的资源文件中的存储形式。在文件里 QUiLoader::load()函数构造形式使用用户界面小控件描述包含。

使用以下指令包含 QtUiTools 模块类：

```
#include <QtUiTools>
```

正如下面显示的，QUiLoader:: load（）函数在文字搜索例子里被调用：

```
QWidget* TextFinder::loadUiFile()
{
    QUiLoader loader;

    QFile file(":/forms/textfinder.ui");
    file.open(QFile::ReadOnly);

    QWidget *formWidget = loader.load(&file, this);
    file.close();

    return formWidget;
}
```

在使用 QtUiTools 在运行时建立用户接口的类中，我们可以找到使用 qFindChild（）格式的对象。例如，在 followig 代码中，我们找到了一些基于它们的对象名和控件类型的组件：

```
ui_findButton = findChild<QPushButton*>("findButton");
    ui_textEdit = findChild<QTextEdit*>("textEdit");
    ui_lineEdit = findChild<QLineEdit*>("lineEdit");
```

自动连接

既可以手动也可以自动设置编译或运行时的定义信号和槽的连接，使用 QMetaObject 的能力，建立信号和名称适当的槽之间的连接。

通常，在 QDialog 的，如果我们想接受之前处理由用户输入的信息，我们需要连接 clicked()信号到 OK 按钮以在我们的对话框定制槽。我们首先展示手动连接对话框的例子，作为比较，然后展示自动连接。

『一个对话框，不自动连接』

我们像以前那样定义对话框，但现在包含一个槽：

```
class ImageDialog : public QDialog, private Ui::ImageDialog
{
   Q_OBJECT

public:
   ImageDialog(QWidget *parent = 0);

private slots:
   void checkValues();
};
```

所述 checkValues（）槽将被用于验证用户提供的值。

在对话框的构造函数中，我们像以前那样设置控件，连接取消按钮的 clicked（）信号到的对话框的 reject（）槽。我们还禁止 autoDefault 属性两个按钮，以确保对话不受文本处理事件干扰：

```
ImageDialog::ImageDialog(QWidget *parent)
   : QDialog(parent)
{
   setupUi(this);
   okButton->setAutoDefault(false);
   cancelButton->setAutoDefault(false);
   ...
   connect(okButton, SIGNAL(clicked()), this, SLOT(checkValues()));
}
```

我们将 OK 按钮的 clicked（）信号连接到对话框中的 checkValuea（）槽，如下所示：

```
void ImageDialog::checkValues()
{
   if (nameLineEdit->text().isEmpty())
      (void) QMessageBox::information(this, tr("No Image Name"),
         tr("Please supply a name for the image."), QMessageBox::Cancel);
```

```
    else
        accept();
}
```

『窗口小控件和对话框使用自动连接』

实现对话框中的自定义插槽和连接它在构造函数中是很容易，我们可以转而使用 QMetaObject 的自动连接将 OK 按钮的 clicked（）信号连接到我们的子类中的一个插槽。UIC 在对话框的 setupUi（）函数中自动生成代码，所以我们只需要定义和实施与遵循标准约定的槽名称：

```
void on_<object name>_<signal name>(<signal parameters>);
```

使用这个惯例，我们可以定义并实现了响应鼠标点击 OK 按钮：

```
class ImageDialog : public QDialog, private Ui::ImageDialog
{
    Q_OBJECT

public:
    ImageDialog(QWidget *parent = 0);

private slots:
    void on_okButton_clicked();
};
```

自动信号和槽连接的另一个例子是 Text Finder 例子中的 on_findButton_clicked（）槽。

我们使用 QMetaObject 的体系，使信号和槽的连接：

```
QMetaObject:: connectSlotsByName（this）;
```

这使我们能够实现槽，如下所示：

```
void TextFinder::on_findButton_clicked()
{
    QString searchString = ui_lineEdit->text();
    QTextDocument *document = ui_textEdit->document();

    bool found = false;

    if (isFirstTime == false)
        document->undo();

    if (searchString.isEmpty()) {
```

```
        QMessageBox::information(this, tr("Empty Search Field"),
                "The search field is empty. Please enter a word and click Find.");
    } else {

        QTextCursor highlightCursor(document);
        QTextCursor cursor(document);

        cursor.beginEditBlock();
    ...
        cursor.endEditBlock();
        isFirstTime = false;

        if (found == false) {
            QMessageBox::information(this, tr("Word Not Found"),
                "Sorry, the word cannot be found.");
        }
    }
}
```

信号和槽的自动连接提供了一个标准的命名规则和明确的接口控件设计工作。通过提供实现给定接口的源代码，用户界面设计人员可以检查他们的设计实际工作，而无需编写代码本身。

总结：

在 Qt 文件使用使用 UI 文件的步骤（以 calculatorform.ui 为例）：

1. 在 Qt 文件中直接加入下列语句:

```
#include "ui_calculatorform.h"
```

2. 在 Qt 文件中加入下列语句：

```
Ui::CalculatorForm ui
```

其中 CalculatorForm 是 UI 文件中的对话框对象。

3. 使用 setupUi 函数将 UI 文件转化的 C++文件。

```
ui.setupUi(widget);
```

4. 上述步骤完成后，Ui 文件会加载到 widget 对话框中。