搜索

在C++应用程序中使用设计器 UI 文件

Qt Designer UI文件表示XML格式的表单的小部件树。表格可以处理：

- 在编译时，这意味着表单将转换为可编译C++代码。
- 在运行时，这意味着表单由QUiLoader类处理，该类在分析 XML 文件时动态构造小部件树。

编译时表单处理

您可以使用*Qt Designer*创建用户界面组件，并使用Qt的集成构建工具qmake和uic在构建应用程序时为它们生成代码。生成的代码包含窗体的用户界面对象。它是一个C++结构，包含：

- 指向窗体的小组件、布局、布局项、按钮组和操作的指针。
- 调用的成员函数，用于在父小部件上构建小部件树。`setupUi()`
- 调用的成员函数处理窗体的字符串属性的转换。有关更多信息，请参见对语言更改做出反应。`retranslateUi()`

生成的代码可以包含在应用程序中，并直接从应用程序中使用。或者，您可以使用它来扩展标准小部件的子类。

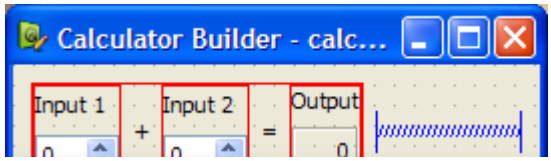
编译时处理的表单可以通过以下方法之一在应用程序中使用：

- 直接方法：构建一个小部件以用作组件的占位符，并在其中设置用户界面。
- 单一继承方法：对表单的基类（例如 QWidget或QDialog）进行子类化，并包含表单用户界面对象的私有实例。
- 多重继承方法：对窗体的基类和窗体的用户界面对象进行子类化。这允许在子类的范围内直接使用表单中定义的小部件。

为了演示，我们创建了一个简单的计算器表单应用程序。它基于原始计算器表单示例。

该应用程序由一个源文件和一个 UI 文件组成。`main.cpp`

使用*Qt Designer*设计的文件如下所示：`calculatorform.ui`



我们将用于构建可执行文件，因此我们需要编写 afile：qmake.pro

```
HEADERS     = calculatorform.h
```

此文件的特殊功能是声明，它告诉要处理哪些文件。在这种情况下，该文件用于创建可由声明中列出的任何文件使用的文件。FORMSqmakeuiccalculatorform.uiui_calculatorform.hSOURCES

> **注意：**您可以使用Qt Creator创建计算器表单项目。它会自动生成主文件.cpp、UI 和 .pro 文件，然后您可以修改这些文件。

直接方法

要使用直接方法，我们将文件直接包含在：ui_calculatorform.hmain.cpp

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

该函数通过构造一个标准QWidget来创建计算器小部件，我们用它来托管文件描述的用户界面。maincalculatorform.ui

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

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

在这种情况下，这是来自该文件的界面描述对象，用于设置对话框的所有小部件及其信号和插槽之间的连接。Ui::CalculatorFormui_calculatorform.h

直接方法提供了一种在应用程序中使用简单、独立的组件的快速简便方法。但是，使用*Qt Designer*创建的组件通常需要与其余的应用程序代码紧密集成。例如，上面提供的代码将编译并运行，但QSpinBox对象不会与 QLabel 交互，因为我们需要一个自定义插槽来执行添加操作并在QLabel中显示结果。为此，我们需要使用单一继承方法。CalculatorForm

单一继承方法

为了使用单一继承方法，我们对标准Qt小部件进行子类化，并包含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

使用成员变量

在这种方法中，我们子类化了一个Qt小部件，并从构造函数中设置用户界面。以这种方式使用的组件将表单中使用的小部件和布局公开给Qt小部件子类，并提供一个标准系统，用于在用户界面和应用程序中的其他对象之间建立信号和插槽连接。生成的结构是类的成员。`Ui::CalculatorForm`

此方法在计算器窗体示例中使用。

为了确保我们可以使用用户界面，我们需要在引用之前包含生成的头文件：`uicUi::CalculatorForm`

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

这意味着必须更新该文件以包括：`.procalculatorform.h`

```
HEADERS     = calculatorform.h
```

子类按以下方式定义：

```
class CalculatorForm : public QWidget
{
    Q_OBJECT

public:
    explicit CalculatorForm(QWidget *parent = nullptr);

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

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

该类的重要功能是私有对象，它提供用于设置和管理用户界面的代码。`ui`

子类的构造函数仅通过调用对象的函数来构造和配置对话框的所有小部件和布局。完成此操作后，可以根据需要修改用户界面。`uisetupUi()`

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

我们可以通过添加on_<对象名称>_<前缀>以通常的方式连接用户界面小部件中的信号和插槽。有关详细信息

量。我们可以使用此方法在同一小部件中定义许多用户界面，每个用户界面都包含在自己的命名空间中，并覆盖（或组合）它们。例如，此方法可用于从现有表单创建单个选项卡。ui

使用指针成员变量

或者，可以将结构设置为类的指针成员。然后，标头如下所示：Ui::CalculatorForm

```
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界面对象可以向前声明，这意味着我们不必在标头中包含生成的文件。然后，可以更改表单，而无需重新编译依赖源文件。如果类受到二进制兼容性限制，这一点尤其重要。
ui_calculatorform.h

对于库和大型应用程序，我们通常建议使用此方法。有关更多信息，请参阅创建共享库。

多重继承方法

使用*Qt Designer*创建的表单可以与基于QWidget的标准类一起子类化。这种方法使表单中定义的所有用户界面组件都可以在子类的范围内直接访问，并允许使用connect（） 函数以通常的方式进行信号和插槽连接。

此方法在多重继承示例中使用。

我们需要包含从该文件生成的头文件，如下所示：uiccalculatorform.ui

该类的定义方式与单一继承方法中使用的类似，只是这次*我们*从QWidget 和 继承，如下所示：
Ui::CalculatorForm

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

public:
    explicit CalculatorForm(QWidget *parent = nullptr);

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

我们私有地继承，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theor关键字继承它，就像我们在前一种情况下可以公开或保护的方式一样。Ui::CalculatorFormpublicprotectedui

子类的构造函数执行许多与单个继承示例中使用的构造函数相同的任务：

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

在这种情况下，可以访问用户界面中使用的小部件，就像手动在代码中创建的小部件一样。我们不再需要前缀来访问它们。ui

对语言变化做出反应

如果用户界面语言发生更改，Qt 会通过发送类型为 QEvent：：LanguageChange 的事件来通知应用程序。为了调用用户界面对象的成员函数，我们在窗体类中重新实现，如下所示：
retranslateUi()QWidget::changeEvent()

```
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

或者，可以在运行时处理表单，生成动态生成的用户界面。这可以使用QtUiTools模块来完成，该模块提供QUiLoader类来处理使用*Qt Designer*创建的表单。

UiTools 方法

在运行时处理表单需要包含 UI 文件的资源文件。此外，还需要将应用程序配置为使用QtUiTools模块。这是通过在项目文件中包含以下声明来完成的，确保正确编译和链接应用程序。qmake

```
QT += uitools
```

类提供了一个表单加载器对象来构造用户界面。此用户界面可以从任何QIODevice（例如QFile对象）中检索，以获取存储在项目资源文件中的表单。QUiLoader：：load（） 函数使用文件中包含的用户界面描述构造表单小部件。

QtUiTools模块类可以使用以下指令包含在内：

```
#include <QtUiTools>
```

调用QUiLoader：：load（） 函数，如文本查找器示例中的以下代码所示：

```
static QWidget *loadUiFile(QWidget *parent)
{
    QFile file(":/forms/textfinder.ui");
    file.open(QIODevice::ReadOnly);

    QUiLoader loader;
    return loader.load(&file, parent);
}
```

在使用QtUiTools在运行时构建其用户界面的类中，我们可以使用QObject：：findChild（）在表单中定位对象。例如，在下面的代码中，我们根据一些组件的对象名称和小部件类型来定位它们：

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

在运行时处理表单使开发人员可以自由地更改程序的用户界面，只需更改 UI 文件即可。这在自定义程序以满足各种用户需求时非常有用，例如超大图标或辅助功能支持的不同配色方案。

自动连接

动连接的对话框进行比较。

没有自动连接的对话框

我们以与以前相同的方式定义对话框，但现在除了构造函数之外还包括一个插槽：

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

public:
    ImageDialog(QWidget *parent = 0);

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

插槽将用于验证用户提供的值。checkValues()

在对话框的构造函数中，我们像以前一样设置小部件，并将**取消**按钮的clicked () 信号连接到对话框的 reject () 插槽。我们还禁用了两个按钮中的autoDefault属性，以确保对话框不会干扰行编辑处理返回键事件的方式：

```
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 () 信号连接到对话框的 checkValues () 插槽，我们实现如下：

```
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();
}
```

具有自动连接功能的小部件和对话框

虽然在对话框中实现自定义插槽并在构造函数中连接它很容易，但我们可以使用QMetaObject 的自动连接功能将OK按钮的 clicked（） 信号连接到子类中的插槽.auto 在对话框的函数中生成代码来执行此操作，因此我们只需要声明和实现一个名称遵循标准约定的插槽：uicsetupUi()

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

使用此约定，我们可以定义并实现一个响应鼠标单击“**确定**”按钮的插槽：

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

public:
    ImageDialog(QWidget *parent = 0);

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

自动信号和插槽连接的另一个例子是带有插槽的文本查找器。on_findButton_clicked()

我们使用QMetaObject的系统来启用信号和插槽连接：

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

这使我们能够实现插槽，如下所示：

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

    bool found = false;

    // undo previous change (if any)
    document->undo();

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

```
        ...
            cursor.endEditBlock();

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

信号和插槽的自动连接为小部件设计人员提供了标准的命名约定和显式接口。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是否实际工作，而无需自己编写代码。

‹ 将样式表与Qt Designer一起使用

在 Qt for Python 应用程序中使用设计器 UI 文件 ›

论坛
维基
下载
市场

反馈 登录