代码化 UI 设计

原创 Qt 学习 Qt 学习 2019-11-11

UI 的可视化设计是对用户而言的，其实底层都是 C++ 的代码实现，只是 Qt 巧妙地进行了处理，让用户省去了很多繁琐的界面设计工作。

由于界面设计的底层其实都是由 C++ 语言实现的，底层实现的功能比可视化设计更加强大和灵活。某些界面效果是可视化设计无法完成的，或者某些人习惯了用纯代码的方式来设计界面，就可以采用纯代码的方式设计界面，如 Qt 自带的实例基本都是用纯代码方式实现用户界面的。

所以，本节介绍一个用纯代码方式设计 UI 的实例（该实例与上一篇基本相同），通过实例**了解用纯代码设计 UI 的基本原理。**与前面的可视化 UI 设计相对应，且称之为代码化 UI 设计。

本篇目录

运行环境：

win 10 + Qt 5.12.5 + Qt Creator 4.10

1. 实例功能

首先建立一个 Widget Application 项目 BasicLayouts，在创建项目向导中选择基类时，选择基类 QDialog，新类的名称命名为 Dialog，**关键是取消创建窗体，即不勾选 "Generate form" 复选框。**创建后的项目文件目录树下没有 *.ui 文件。本例完成后的运行效果如下图所示

2. 水平布局编组框

在上述新建项目 BasicLayouts 的 dialog.h 中添加如下加粗代码，用于实现界面中的第一个编组框。

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QGroupBox>
#include <QPushButton>

class Dialog : public QDialog
{
    Q_OBJECT

public:
    Dialog(QWidget *parent = nullptr);
    ~Dialog();

private:
```

```
    void createHorizontalGroupBox();

    QGroupBox *horizontalGroupBox;
    QPushButton *buttons[4];
};
#endif // DIALOG_H
```

其中，

- 加入实现 `GroupBox` 和 `PushButton` 控件的头文件
- 定义界面中 `GroupBox` 和 `PushButton` 控件对象（4 个）
- 函数 `void createHorizontalGroupBox();` 声明水平布局编组框函数，用于创建按钮及其布局

`Q_OBJECT` 宏的作用是启动 Qt 5 元对象系统的一些特性（如支持信号和槽等），它必须放置到类定义的私有区中。

在 dialog.cpp 文件中添加如下加粗代码

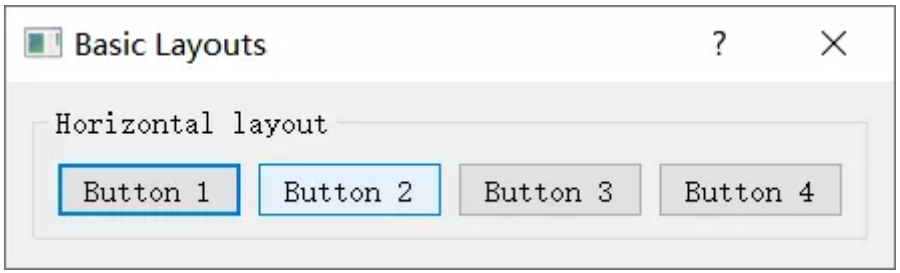
```
#include "dialog.h"
#include <QHBoxLayout>
#include <QVBoxLayout>

Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
{
    createHorizontalGroupBox();      // 定义水平布局编组框及编组框内的各按钮组件

    QVBoxLayout *mainLayout = new QVBoxLayout; // 创建主窗口布局管理器
    mainLayout->addWidget(horizontalGroupBox); // 将编组框套件加入主布局中

    setLayout(mainLayout);           // 设置对话框窗口布局
    setWindowTitle("Basic Layouts"); // 设置对话框窗口名
}

Dialog::~Dialog()
{
```

```
}

void Dialog::createHorizontalGroupBox()
{
    horizontalGroupBox = new QGroupBox(tr("Horizontal layout"));
    QHBoxLayout *layout = new QHBoxLayout;

    for (int i = 0; i < 4; ++i) { // 创建并添加布局按钮
        buttons[i] = new QPushButton(tr("Button %1").arg(i + 1));
        layout->addWidget(buttons[i]);
    }
    horizontalGroupBox->setLayout(layout); // 设置编组框的按钮布局
}
```

其中，

- 布局类 `QVBoxLayout` 和 `QVBoxLayout` 为实现布局管理器的头文件
- 布局对象 `mainLayout` 为 Basic Layouts 窗体的总体布局
- `QHBoxLayout *layout = new QHBoxLayout;` 声明 4 个按钮的水平布局管理器

运行程序，效果如下



3. 完善其余布局

界面其余部分的创建过程与 Horizontal layout 类似，下面给出主要过程。

3.1 创建 Grid Layout

首先，在头文件 dialog.h 中添加各控件对应的头文件

```
#include <QLabel>
#include <QLineEdit>
#include <QTextEdit>
```

以及在 `private` 中声明函数 `void createGridGroupBox();` 和 Group Box、Label、Line Edit 和 Text Edit 控件对象。

```
private:
    void createGridGroupBox();

    QGroupBox *gridGroupBox;
    QLabel *labels[3];
    QLineEdit *lineEdits[3];
    QTextEdit *smallEditor;
```

在源文件中定义 `createGridGroupBox()` 函数

```
void Dialog::createGridGroupBox()
{
    gridGroupBox = new QGroupBox(tr("Grid layout"));
    QGridLayout *layout = new QGridLayout;

    for (int i = 0; i < 3; ++i) {
        labels[i] = new QLabel(tr("Line %1:").arg(i + 1));
        lineEdits[i] = new QLineEdit;
        layout->addWidget(labels[i], i, 0);
        layout->addWidget(lineEdits[i], i, 1);
```

```
    }

    smallEditor = new QTextEdit;
    smallEditor->setPlainText(tr("This widget takes up about two thirds of the grid layout."));
    layout->addWidget(smallEditor, 0, 2, 3, 1);

    layout->setColumnStretch(1, 1); // 窗体大小改变时，lineEdits 与
    layout->setColumnStretch(2, 2); // smallEditot 会按 1:2 的比例发生改变
    gridGroupBox->setLayout(layout);
}
```

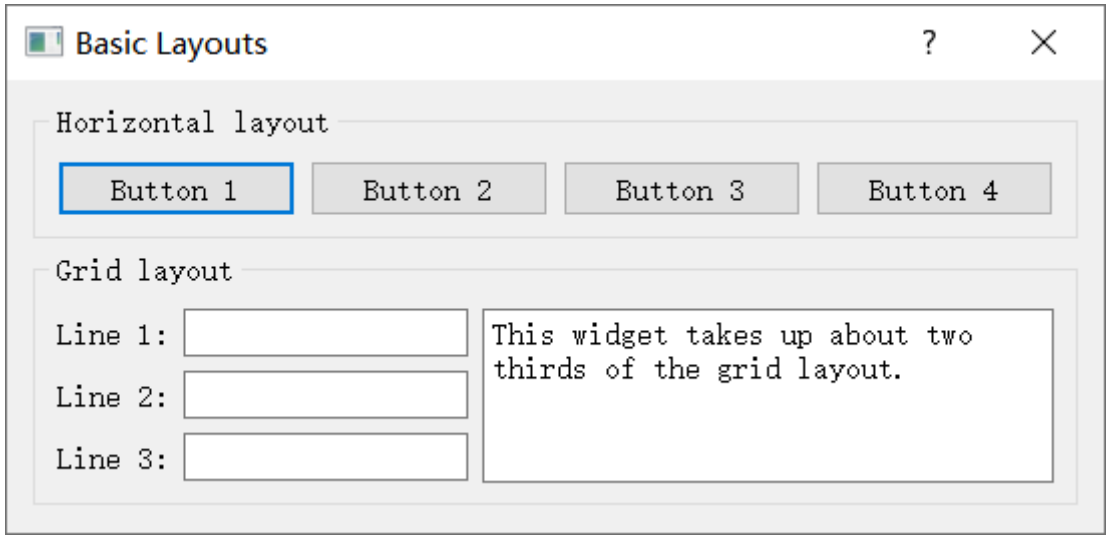
在构造函数中添加两行代码

```
createGridGroupBox();
```

和

```
mainLayout->addWidget(gridGroupBox);
```

以及在源文件中添加头文件 `#include <QGridLayout>` 。运行程序的效果如下



3.2 创建 Form Layout 和 Text Edit

Form Layout 的创建与 Grid Layout 的创建存在些许差别。在头文件的 `private` 中添加代码

```
void createFormGroupBox();
QGroupBox *formGroupBox;
QTextEdit *bigEditor;
```

定义 `createFormGroupBox()` 函数

```
void Dialog::createFormGroupBox()
{
    formGroupBox = new QGroupBox(tr("Form layout"));
    QFormLayout *layout = new QFormLayout;
    layout->addRow(new QLabel(tr("Line 1:")), new QLineEdit);
    layout->addRow(new QLabel(tr("Line 2, long text:")), new QComboBox);
    layout->addRow(new QLabel(tr("Line 3:")), new QSpinBox);
```

```
    formGroupBox->setLayout(layout);
}
```

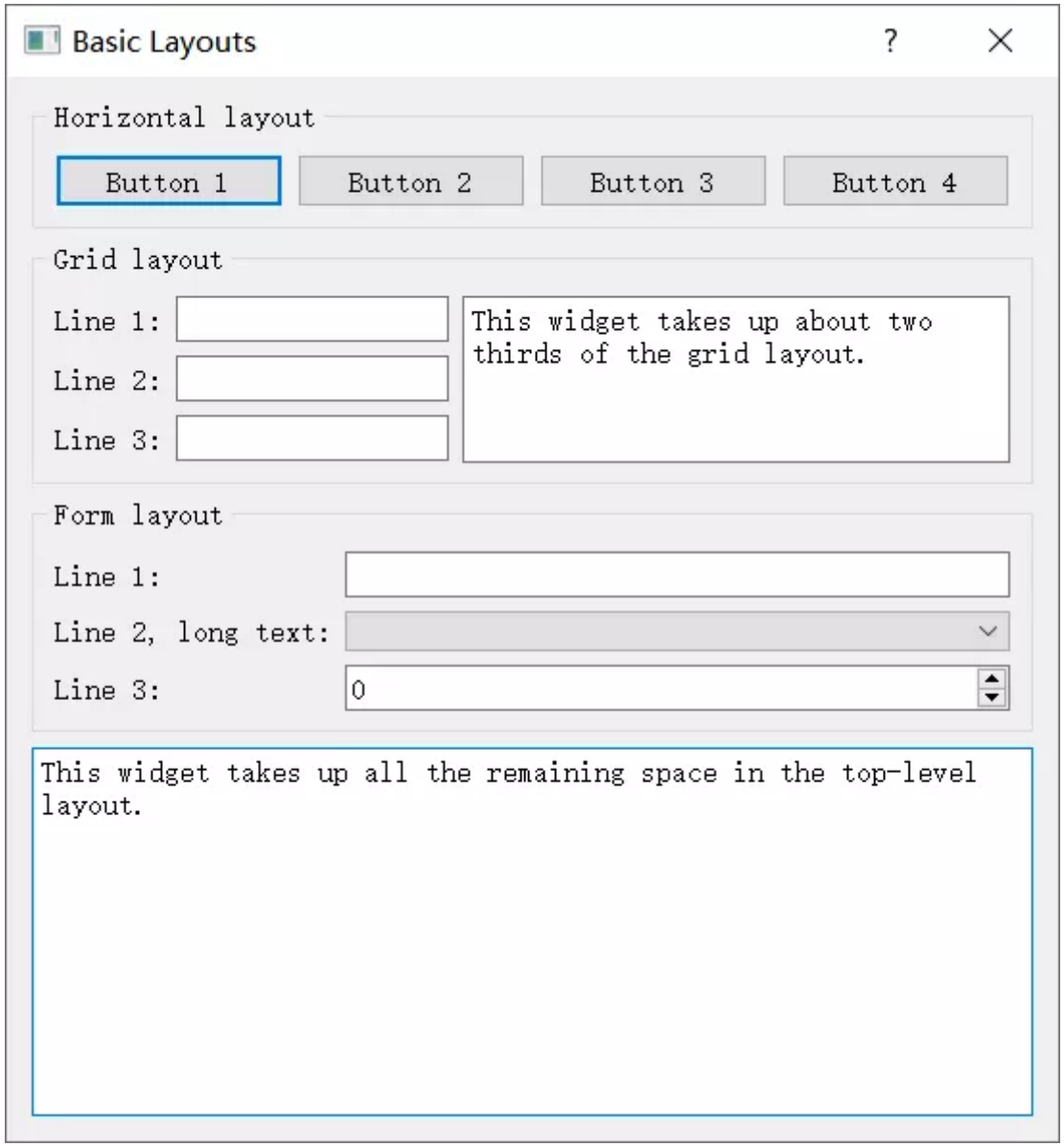
在源文件中添加头文件

```
#include <QFormLayout>
#include <QComboBox>
#include <QSpinBox>
```

以及在构造函数中添加代码

```
createFormGroupBox();

bigEditor = new QTextEdit;
bigEditor->setPlainText(tr("This widget takes up all the remaining space in the top-level layout."));

mainLayout->addWidget(formGroupBox);
mainLayout->addWidget(bigEditor);
```

程序运行效果


Basic Layouts
Horizontal layout
Button 1
Button 2
Button 3
Button 4
Grid layout
Line 1:
Line 2:
Line 3:
This widget takes up about two thirds of the grid layout.
Form layout
Line 1:
Line 2, long text:
Line 3:
0
This widget takes up all the remaining space in the top-level layout.

4. 窗口事件

这一节包含两个内容：添加确定、取消按钮和菜单栏。

添加菜单栏的目的为了说明可视化 UI 设计存在的一些缺陷，如无法添加菜单栏，也无法在工具栏上添加 ComboBox 组件等等。采用纯代码方式进行 UI 设计虽然无所不能，但是设计效率太低，过程非常繁琐，而可视化 UI 设计简单高效，也更直观。

在头文件中添加 `#include <QDialogButtonBox>`，然后声明确定、取消按钮盒子。

```
QDialogButtonBox *buttonBox;
```

在对话框窗口中添加按钮（构造函数中添加如下代码）

```
...
buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel);
connect(buttonBox, SIGNAL(accepted()), this, SLOT(accept())); // 信号与槽关联函数
connect(buttonBox, SIGNAL(rejected()), this, SLOT(reject())); // 信号与槽关联函数
...
mainLayout->addWidget(buttonBox);
...
```

添加菜单栏的过程与添加 Group Box 组件的过程相似。在头文件中添加如下代码

```
...
#include <QMenuBar>
#include <QMenu>
#include <QAction>
...
private:
    void createMenu();

    QMenuBar *menuBar;
```

```
    QMenu *fileMenu;
    QAction *exitAction;
...
```

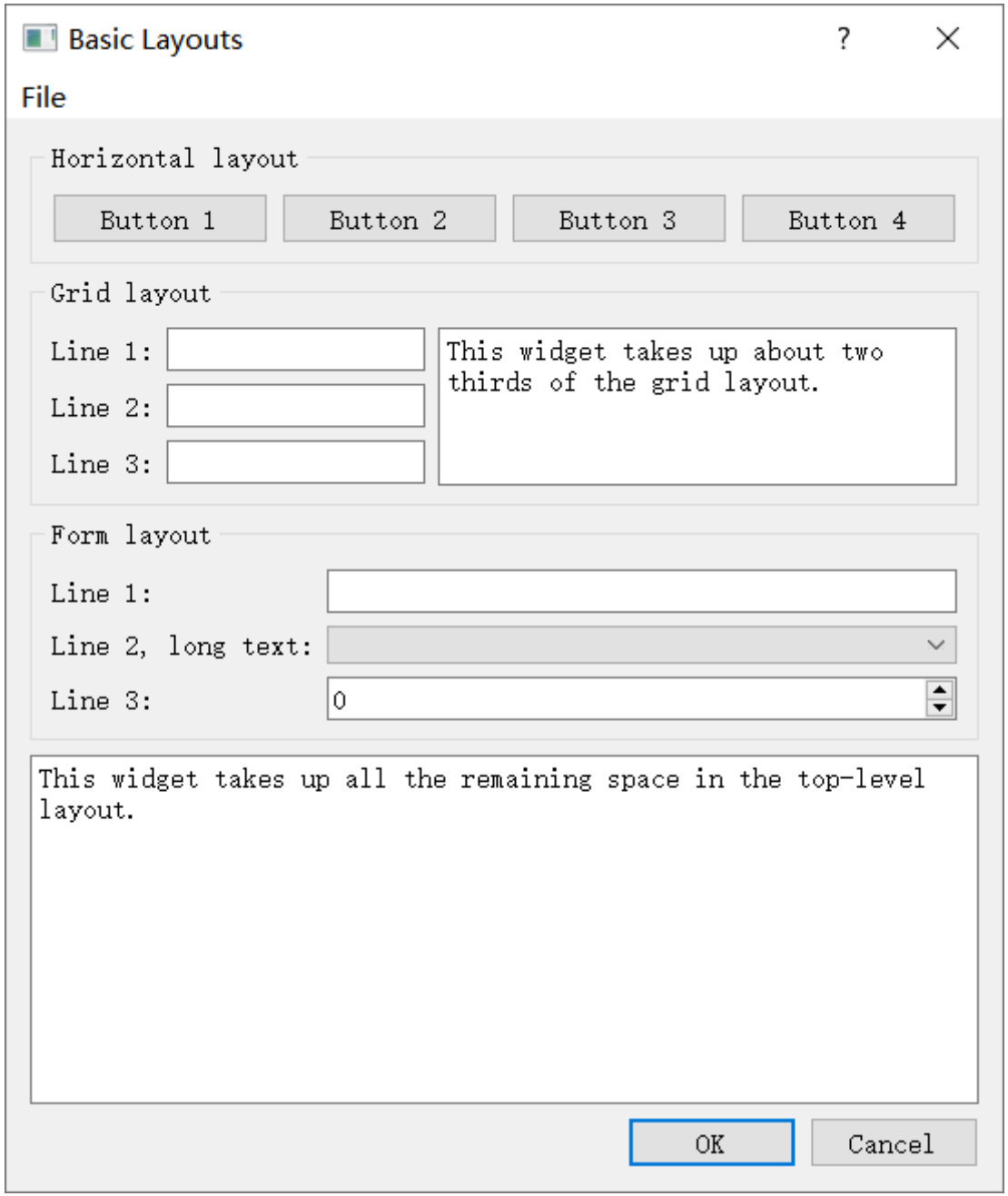
同时，定义创建菜单的成员函数

```
void Dialog::createMenu()
{
    menuBar = new QMenuBar;

    fileMenu = new QMenu(tr("&File"), this);
    exitAction = fileMenu->addAction(tr("E&xit"));
    menuBar->addMenu(fileMenu);

    connect(exitAction, SIGNAL(triggered()), this, SLOT(accept())); // 信号与槽关联函数
}
```

在构造函数中添加如下代码

```
createMenu();
mainLayout->setMenuBar(menuBar);
```

最后运行结果


Basic Layouts
File
Horizontal layout
Button 1
Button 2
Button 3
Button 4
Grid layout
Line 1:
Line 2:
Line 3:
This widget takes up about two thirds of the grid layout.
Form layout
Line 1:
Line 2, long text:
Line 3:
0
This widget takes up all the remaining space in the top-level layout.
OK
Cancel

额外说明：

在上述代码中，出现了三处信号与槽关联函数

```
connect(buttonBox, SIGNAL(accepted()), this, SLOT(accept()));
connect(buttonBox, SIGNAL(rejected()), this, SLOT(reject()));
connect(exitAction, SIGNAL(triggered()), this, SLOT(accept()));
```

上述三个关联函数 `connect()` 将信号 `SIGNAL` 与槽 `SLOT` 关联在一起，使创建的程序对用户的触发信号时有了响应（槽）。下一篇将对此进行详细介绍。

小结

本篇介绍了一个用纯代码方式设计 UI 的实例，涉及菜单栏、核心窗口布局设计、信号与槽函数的链接。可视化与代码化 UI 设计对比如下表：

	可视化UI设计	**代码化UI设计**
优点	简单、高效、直观	界面组件功能不全
缺点	功能强大且灵活	效率低、代码量大

源代码：

链接: https://pan.baidu.com/s/1WvClFiKajc_01S-1XfVW7A
提取码: ut2n

相关阅读：

《可视化 UI 设计（设计器 Qt Designer 实现）》

《UI 文件设计与运行机制》

《编写一个 Hello World 程序》

20年的嵌入式经验：如何从零开始开发一款嵌入式产品(值得收藏的高质量文章!)

芯片之家