1 对话框的基本概念

对话框是 GUI 程序中不可或缺的组成部分。很多不能或者不适合放入主窗口的功能组件都必须放在对话框中设置。对话框通常会是一个顶层窗口，出现在程序最上层，用于实现短期任务或者简洁的用户交互。

Qt 中使用QDialog类实现对话框。就像主窗口一样，我们通常会设计一个类继承QDialog。QDialog（及其子类，以及所有Qt::Dialog类型的类）的对于其 parent 指针都有额外的解释：如果 parent 为 NULL，则该对话框会作为一个顶层窗口，否则则作为其父组件的子对话框（此时，其默认出现的位置是 parent 的中心）。顶层窗口与非顶层窗口的区别在于，顶层窗口在任务栏会有自己的位置，而非顶层窗口则会共享其父组件的位置。

对话框分为模态对话框和非模态对话框。

- 模态对话框，就是会阻塞同一应用程序中其它窗口的输入。

模态对话框很常见，比如“打开文件”功能。你可以尝试一下记事本的打开文件，当打开文件对话框出现时，我们是不能对除此对话框之外的窗口部分进行操作的。

- 与此相反的是非模态对话框，例如查找对话框，我们可以在显示着查找对话框的同时，继续对记事本的内容进行编辑。

2 标准对话框

所谓标准对话框，是 Qt 内置的一系列对话框，用于简化开发。事实上，有很多对话框都是通用的，比如打开文件、设置颜色、打印设置等。这些对话框在所有

程序中几乎相同，因此没有必要在每一个程序中都自己实现这么一个对话框。

Qt 的内置对话框大致分为以下几类：

QColorDialog： 选择颜色；

QFileDialog： 选择文件或者目录；

QFontDialog： 选择字体；

QInputDialog： 允许用户输入一个值，并将其值返回；

QMessageBox： 模态对话框，用于显示信息、询问问题等；

QPageSetupDialog： 为打印机提供纸张相关的选项；

QPrintDialog： 打印机配置；

QPrintPreviewDialog: 打印预览；

QProgressDialog： 显示操作过程。

```
#include "widget.h"
#include <QFileDialog>
#include <QFileDialog>
#include <QString>
#include <QDebug>
Widget::Widget(QWidget *parent)
 : QWidget(parent)
{
 this->resize(600,400);
  button = new QPushButton("选择文件",this);
  button->resize(100,100);
  button->move(250,100);
  connect(button,&QPushButton::clicked,this,[](){
  // QString str = QFileDialog::getOpenFileName();
  // qDebug()<<str;
  QStringList str = QFileDialog::getOpenFileNames();
  qDebug()<<str;
  });
 }

Widget::~Widget()
{

}

```

3 模态对话框

模态对话框

Qt 有两种级别的模态对话框：

应用程序级别的模态

当该种模态的对话框出现时，用户必须首先对对话框进行交互，直到关闭对话框，然后才能访问程序中其他的窗口。

窗口级别的模态

该模态仅仅阻塞与对话框关联的窗口，但是依然允许用户与程序中其它窗口交互。窗口级别的模态尤其适用于多窗口模式。

一般默认是应用程序级别的模态。

在下面的示例中，我们调用了exec()将对话框显示出来，因此这就是一个模态对话框。当对话框出现时，我们不能与主窗口进行任何交互，直到我们关闭了该对话框。

```
QDialog dialog;
dialog.setWindowTitle(tr("Hello, dialog!"));
dialog.exec();
```

```
#include "widget.h"
#include <QFileDialog>
#include <QFileDialog>
#include <QString>
#include <QDebug>
#include <QDialog>
Widget::Widget(QWidget *parent)
  : QWidget(parent)
{
  this->resize(600,400);
  button = new QPushButton("选择文件",this);
  button->resize(100,100);
  button->move(250,100);
  connect(button,&QPushButton::clicked,this,[](){
  // QString str = QFileDialog::getOpenFileName();
  // qDebug()<<str;
  QStringList str = QFileDialog::getOpenFileNames();
  qDebug()<<str;
  });
```

```
20  button1 = new QPushButton("模态对话框按钮",this);
21  button1->resize(100,100);
22  button1->move(250,200);
23  connect(button1,&QPushButton::clicked,this,[](){
24  //定义一个对话框
25  QDialog dialog;
26  dialog.setWindowTitle(tr("Hello, dialog!"));//设置对话框标题
27  dialog.exec();//阻塞对话框
28  });
29
30
31 }
32
33 Widget::~Widget()
34 {
35
36 }
37
```

4 非模态对话框

下面我们试着将exec()修改为show()，看看非模态对话框：

```
QDialog dialog(this);
dialog.setWindowTitle(tr("Hello, dialog!"));
 dialog.show();
```

是不是事与愿违？对话框竟然一闪而过！这是因为，show()函数不会阻塞当前线程，对话框会显示出来，然后函数立即返回，代码继续执行。注意，dialog是建立在栈上的，show()函数返回，MainWindow::open()函数结束，dialog 超出作用域被析构，因此对话框消失了。知道了原因就好改了，我们将 dialog 改成堆上建立，当然就没有这个问题了：

```
QDialog *dialog = new QDialog;
dialog->setWindowTitle(tr("Hello, dialog!"));
dialog->show();
```

如果你足够细心，应该发现上面的代码是有问题的：dialog 存在内存泄露！dialog 使用 new 在堆上分配空间，却一直没有 delete。解决方案也很简单：将 MainWindow 的指针赋给 dialog 即可。还记得我们前面说过的 Qt 的对象系统吗？

不过，这样做有一个问题：如果我们的对话框不是在一个界面类中出现呢？由于QWidget的 parent 必须是QWidget指针，那就限制了我们不能将一个普通的 C++ 类指针传给 Qt 对话框。另外，如果对内存占用有严格限制的话，当我们将主窗口作为 parent 时，主窗口不关闭，对话框就不会被销毁，所以会一直占用内存。在这种情景下，我们可以设置 dialog 的WindowAttribute：

```
QDialog *dialog = new QDialog;
dialog->setAttribute(Qt::WA_DeleteOnClose);
dialog->setWindowTitle(tr("Hello, dialog!"));
dialog->show();
```

setAttribute()函数设置对话框关闭时，自动销毁对话框。

```
#include "widget.h"
#include <QFileDialog>
#include <QFileDialog>
#include <QString>
#include <QDebug>
#include <QDialog>
Widget::Widget(QWidget *parent)
  : QWidget(parent)
{
  this->resize(600,400);
  button = new QPushButton("选择文件",this);
  button->resize(100,100);
  button->move(250,100);
  connect(button,&QPushButton::clicked,this,[](){
  // QString str = QFileDialog::getOpenFileName();
  // qDebug()<<str;
  QStringList str = QFileDialog::getOpenFileNames();
  qDebug()<<str;
  });
  button1 = new QPushButton("模态对话框按钮",this);
  button1->resize(100,100);
  button1->move(250,200);
  connect(button1,&QPushButton::clicked,this,[](){
  //定义一个对话框
  QDialog dialog;
  dialog.setWindowTitle(tr("Hello, dialog!"));//设置对话框标题
  dialog.exec();//阻塞对话框
```

```
28   });
29
30   button2 = new QPushButton("非模态对话框按钮",this);
31   button2->resize(100,100);
32   button2->move(450,200);
33   connect(button2,&QPushButton::clicked,this,[](){
34
35   QDialog *dialog = new QDialog;
36   dialog->setAttribute(Qt::WA_DeleteOnClose);//关闭窗口自动释放 dialog->set
WindowTitle(tr("非模态, dialog!"));
37   dialog->setWindowTitle(tr("Hello, dialog!"));
38   dialog->show();
39   });
40
41  }
42
43  Widget::~Widget()
44  {
45
46  }
47
```

5 消息对话框

QMessageBox用于显示消息提示。我们一般会使用其提供的几个 static 函数：

| 显示关于对话框。

void about(QWidget * parent, const QString & title, const QString & text)

这是一个最简单的对话框，其标题是 title，内容是 text，父窗口是 parent。对话框只有一个 OK 按钮。

| 显示关于 Qt 对话框。该对话框用于显示有关 Qt 的信息。

void aboutQt(QWidget * parent, const QString & title = QString());

| 显示严重错误对话框。

StandardButton critical(QWidget * parent,

const QString & title,

const QString & text,

StandardButtons buttons = Ok,

StandardButton defaultButton = NoButton);

这个对话框将显示一个红色的错误符号。我们可以通过 buttons 参数指明其显示的按钮。默认情况下只有一个 Ok 按钮，我们可以使用StandardButtons类型指定多种按钮。

- 与QMessageBox::critical()类似，不同之处在于这个对话框提供一个普通信息图标。

```
StandardButton information(QWidget * parent,
    const QString & title,
    const QString & text,
    StandardButtons buttons = Ok,
    StandardButton defaultButton = NoButton)
```

- 与QMessageBox::critical ()类似，不同之处在于这个对话框提供一个问号图标，并且其显示的按钮是“是”和“否”。

```
StandardButton question(QWidget * parent,
    const QString & title,
    const QString & text,
    StandardButtons buttons = StandardButtons( Yes | No ),
    StandardButton defaultButton = NoButton)
```

- 与QMessageBox::critical()类似，不同之处在于这个对话框提供一个黄色叹号图标。

```
StandardButton warning(QWidget * parent,
    const QString & title,
    const QString & text,
    StandardButtons buttons = Ok,
    StandardButton defaultButton = NoButton)
```

我们可以通过下面的代码来演示下如何使用QMessageBox。

```
if (QMessageBox::Yes == QMessageBox::question(this,
        tr("Question"), tr("Are you OK?"),
        QMessageBox::Yes | QMessageBox::No,
        QMessageBox::Yes))
{
  QMessageBox::information(this, tr("Hmmm..."),
    tr("I'm glad to hear that!"));
}
```

```
else
{
   QMessageBox::information(this, tr("Hmmm..."),
      tr("I'm sorry!"));
}
```

我们使用QMessageBox::question()来询问一个问题。

- 这个对话框的父窗口是 this。

QMessageBox是QDialog的子类，这意味着它的初始显示位置将会是在 parent 窗口的中央。

- 第二个参数是对话框的标题。
- 第三个参数是我们想要显示的内容。
- 第四个参数是关联的按键类型，我们可以使用或运算符（|）指定对话框应该出现的按钮。比如我们希望是一个 Yes 和一个 No。
- 最后一个参数指定默认选择的按钮。

这个函数有一个返回值，用于确定用户点击的是哪一个按钮。按照我们的写法，应该很容易的看出，这是一个模态对话框，因此我们可以直接获取其返回值。

QMessageBox类的 static 函数优点是方便使用，缺点也很明显：非常不灵活。我们只能使用简单的几种形式。为了能够定制QMessageBox细节，我们必须使用QMessageBox的属性设置 API。如果我们希望制作一个询问是否保存的对话框，我们可以使用如下的代码：

```
QMessageBox msgBox;
msgBox.setText(tr("The document has been modified."));
msgBox.setInformativeText(tr("Do you want to save your changes?"));
msgBox.setDetailedText(tr("Differences here..."));
msgBox.setStandardButtons(QMessageBox::Save
                        | QMessageBox::Discard
                        | QMessageBox::Cancel);
msgBox.setDefaultButton(QMessageBox::Save);
int ret = msgBox.exec();
switch (ret)
{
case QMessageBox::Save:
```

```
    qDebug() << "Save document!";
    break;
case QMessageBox::Discard:
    qDebug() << "Discard changes!";
    break;
case QMessageBox::Cancel:
    qDebug() << "Close document!";
    break;
}
```

msgBox 是一个建立在栈上的QMessageBox实例。我们设置其主要文本信息为“The document has been modified.”，informativeText 则是会在对话框中显示的简单说明文字。下面我们使用了一个detailedText，也就是详细信息，当我们点击了详细信息按钮时，对话框可以自动显示更多信息。我们自己定义的对话框的按钮有三个：保存、丢弃和取消。然后我们使用了exec()是其成为一个模态对话框，根据其返回值进行相应的操作。

```
#include "widget.h"
#include <QMessageBox>
#include <QDebug>

Widget::Widget(QWidget *parent)
  : QWidget(parent)
{
 this->resize(600,400);
 button = new QPushButton("关于",this);
  button->resize(100,100);
  button->move(100,100);
  connect(button,&QPushButton::clicked,this,[=](){
  QMessageBox::about(this,"标题","内容");
  });

  button1 = new QPushButton("询问",this);
  button1->resize(100,100);
  button1->move(200,100);
  connect(button1,&QPushButton::clicked,this,[=](){
  int ret = QMessageBox::question(this,"标题","你需要吗",QMessageBox::Open|QMessageBox::Save);
  if(ret == QMessageBox::Open)
```

```
22  {
23
24  qDebug()<<"open";
25  }
26  else
27  {
28
29  qDebug()<<"save";
30  }
31  });
32
33
34
35  }
36
37  Widget::~Widget()
38  {
39
40  }
41
```