# Qt 之自定义界面（添加自定义标题栏）

# 简述

通过上节内容，我们实现了自定义窗体的移动，但是我们缺少一个标题栏来显示窗体的图标、标题，以及控制窗体最小化、最大化、关闭的按钮。

自定义标题栏后，所有的控件我们都可以定制，比如：在标题栏中添加换肤、设置按钮以及其他控件。

# 效果



# 自定义标题栏

## 实现

title\_bar.h

#ifndef TITLE\_BAR

#define TITLE\_BAR

#include <QWidget>

class QLabel;

class QPushButton;

class TitleBar : public QWidget

{

Q\_OBJECT

public:

explicit TitleBar(QWidget \*parent = 0);

~TitleBar();

protected:

// 双击标题栏进行界面的最大化/还原

virtual void mouseDoubleClickEvent(QMouseEvent \*event);

*// 进行鼠界面的拖动*

virtual void mousePressEvent(QMouseEvent \*event);

*// 设置界面标题与图标*

virtual bool eventFilter(QObject \*obj, QEvent \*event);

private slots:

// 进行最小化、最大化/还原、关闭操作

void onClicked();

private:

// 最大化/还原

void updateMaximize();

private:

QLabel \*m\_pIconLabel;

QLabel \*m\_pTitleLabel;

QPushButton \*m\_pMinimizeButton;

QPushButton \*m\_pMaximizeButton;

QPushButton \*m\_pCloseButton;

};

#endif // TITLE\_BAR

title\_bar.cpp

#include <QLabel>

#include <QPushButton>

#include <QHBoxLayout>

#include <QEvent>

#include <QMouseEvent>

#include <QApplication>

#include "title\_bar.h"

#ifdef Q\_OS\_WIN

#pragma comment(lib, "user32.lib")

#include <qt\_windows.h>

#endif

TitleBar::TitleBar(QWidget \*parent)

: QWidget(parent)

{

setFixedHeight(30);

m\_pIconLabel = new QLabel(this);

m\_pTitleLabel = new QLabel(this);

m\_pMinimizeButton = new QPushButton(this);

m\_pMaximizeButton = new QPushButton(this);

m\_pCloseButton = new QPushButton(this);

m\_pIconLabel->setFixedSize(20, 20);

m\_pIconLabel->setScaledContents(true);

m\_pTitleLabel->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);

m\_pMinimizeButton->setFixedSize(27, 22);

m\_pMaximizeButton->setFixedSize(27, 22);

m\_pCloseButton->setFixedSize(27, 22);

m\_pTitleLabel->setObjectName("whiteLabel");

m\_pMinimizeButton->setObjectName("minimizeButton");

m\_pMaximizeButton->setObjectName("maximizeButton");

m\_pCloseButton->setObjectName("closeButton");

m\_pMinimizeButton->setToolTip("Minimize");

m\_pMaximizeButton->setToolTip("Maximize");

m\_pCloseButton->setToolTip("Close");

QHBoxLayout \*pLayout = new QHBoxLayout(this);

pLayout->addWidget(m\_pIconLabel);

pLayout->addSpacing(5);

pLayout->addWidget(m\_pTitleLabel);

pLayout->addWidget(m\_pMinimizeButton);

pLayout->addWidget(m\_pMaximizeButton);

pLayout->addWidget(m\_pCloseButton);

pLayout->setSpacing(0);

pLayout->setContentsMargins(5, 0, 5, 0);

setLayout(pLayout);

connect(m\_pMinimizeButton, SIGNAL(clicked(bool)), this, SLOT(onClicked()));

connect(m\_pMaximizeButton, SIGNAL(clicked(bool)), this, SLOT(onClicked()));

connect(m\_pCloseButton, SIGNAL(clicked(bool)), this, SLOT(onClicked()));

}

TitleBar::~TitleBar()

{

}

void TitleBar::mouseDoubleClickEvent(QMouseEvent \*event)

{

Q\_UNUSED(event);

emit m\_pMaximizeButton->clicked();

}

void TitleBar::mousePressEvent(QMouseEvent \*event)

{

#ifdef Q\_OS\_WIN

if (ReleaseCapture())

{

QWidget \*pWindow = this->window();

if (pWindow->isTopLevel())

{

SendMessage(HWND(pWindow->winId()), WM\_SYSCOMMAND, SC\_MOVE + HTCAPTION, 0);

}

}

event->ignore();

#else

#endif

}

bool TitleBar::eventFilter(QObject \*obj, QEvent \*event)

{

switch (event->type())

{

case QEvent::WindowTitleChange:

{

QWidget \*pWidget = qobject\_cast<QWidget \*>(obj);

if (pWidget)

{

m\_pTitleLabel->setText(pWidget->windowTitle());

return true;

}

}

case QEvent::WindowIconChange:

{

QWidget \*pWidget = qobject\_cast<QWidget \*>(obj);

if (pWidget)

{

QIcon icon = pWidget->windowIcon();

m\_pIconLabel->setPixmap(icon.pixmap(m\_pIconLabel->size()));

return true;

}

}

case QEvent::WindowStateChange:

case QEvent::Resize:

updateMaximize();

return true;

}

return QWidget::eventFilter(obj, event);

}

void TitleBar::onClicked()

{

QPushButton \*pButton = qobject\_cast<QPushButton \*>(sender());

QWidget \*pWindow = this->window();

if (pWindow->isTopLevel())

{

if (pButton == m\_pMinimizeButton)

{

pWindow->showMinimized();

}

else if (pButton == m\_pMaximizeButton)

{

pWindow->isMaximized() ? pWindow->showNormal() : pWindow->showMaximized();

}

else if (pButton == m\_pCloseButton)

{

pWindow->close();

}

}

}

void TitleBar::updateMaximize()

{

QWidget \*pWindow = this->window();

if (pWindow->isTopLevel())

{

bool bMaximize = pWindow->isMaximized();

if (bMaximize)

{

m\_pMaximizeButton->setToolTip(tr("Restore"));

m\_pMaximizeButton->setProperty("maximizeProperty", "restore");

}

else

{

m\_pMaximizeButton->setProperty("maximizeProperty", "maximize");

m\_pMaximizeButton->setToolTip(tr("Maximize"));

}

m\_pMaximizeButton->setStyle(QApplication::style());

}

}

## 接口说明

* mousePressEvent

之前，我们将界面移动的事件写在主界面里面，这会有一个问题，一般情况下，是界面随着标题栏的移动而移动，而并非界面中的所有位置都可以进行拖动，所以我们将事件写在标题栏中比较合理。

* mouseDoubleClickEvent

双击标题栏会进行窗体的最大化/还原，所以我们需要重写此事件进行控制。

* eventFilter
  1. 事件过滤器，这里被监听的窗体为标题栏所在的窗体，所以当窗体标题、图标等信息发生改变时，标题栏也应该随之改变。
  2. 最好不要通过直接调用接口的形式来操作对应的行为，比如：TitleBar中定义一个public函数来专门修改标题与图标，这样会造成不必要的麻烦，因为Qt本身就是基于事件的，所以此处采用过滤器的方式。
* updateMaximize

因为窗体大小发生变化的时候，最大化的图标、提示应该对应的发生变化，所以在eventFilter中事件触发时调用。

## 使用方式

Widget::Widget(QWidget \*parent)

: QWidget(parent)

{

setWindowFlags(Qt::FramelessWindowHint | windowFlags());

TitleBar \*pTitleBar = new TitleBar(this);

installEventFilter(pTitleBar);

resize(400, 300);

setWindowTitle("Custom Window");

setWindowIcon(QIcon(":/Images/logo"));

QPalette pal(palette());

pal.setColor(QPalette::Background, QColor(50, 50, 50));

setAutoFillBackground(true);

setPalette(pal);

QVBoxLayout \*pLayout = new QVBoxLayout();

pLayout->addWidget(pTitleBar);

pLayout->addStretch();

pLayout->setSpacing(0);

pLayout->setContentsMargins(0, 0, 0, 0);

setLayout(pLayout);

}