# [[置顶] Qt 之 自定义窗口标题栏](http://blog.csdn.net/goforwardtostep/article/details/53494800)

标签： [qt](http://www.csdn.net/tag/qt)[标题栏](http://www.csdn.net/tag/%e6%a0%87%e9%a2%98%e6%a0%8f)[自定义标题栏](http://www.csdn.net/tag/%e8%87%aa%e5%ae%9a%e4%b9%89%e6%a0%87%e9%a2%98%e6%a0%8f)[title](http://www.csdn.net/tag/title)[titlebar](http://www.csdn.net/tag/titlebar)

2016-12-06 23:20 7100人阅读 [评论](http://blog.csdn.net/GoForwardToStep/article/details/53494800#comments)(11) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/GoForwardToStep/article/details/53494800#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Qt（56） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

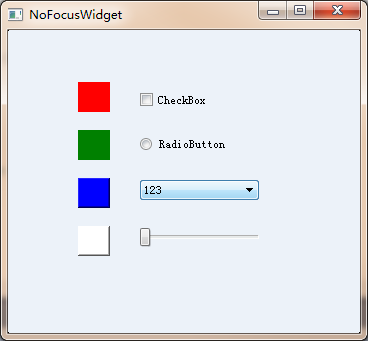
版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/GoForwardToStep/article/details/53494800)

# 一、简述

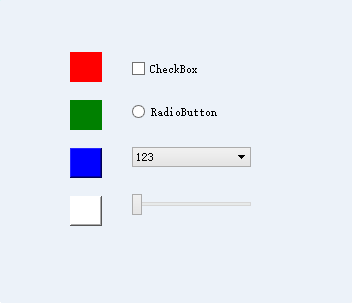
今天晚上就如何用**Qt自定义窗口标题栏**，写了一个小例子，比较基础，实用。在此分享一下。

首先Qt是跨平台的，所以在不同的平台上窗口的外观是不一样的。比如在windows平台上，用Qt显示的一个窗口就跟一般windows程序窗口的外观一致，如下图。

同时不同windows系统，窗口外观显示也不一致，上面**左边是我在win8系统上显示的窗口**，**右边是win7下显示的窗口**。可以看出两个窗口外观差别很大，同时在win8系统中标题内容居中了，win7标题则靠左。为了 解决窗口外观在不同系统下不一致的问题，我们干脆去掉窗口自带的边框，Qt中直接用一下代码即可**去除窗口边框效果**。

this->setWindowFlags(Qt::FramelessWindowHint);



上图就是去除窗口边框后的效果图。这样是不是看起来奇怪，是不是没有了标题栏。

一般情况下一个标题栏提供了窗口图标、窗口标题、最小化、最大化、关闭按钮等几个部分，这些对于一个窗口来说是十分重要的，窗口图标标志着这是一个什么程序，标题就表明这个窗口是干什么的，最小化、最大化、关闭等按钮则是用户对该窗口的操作。

下图是QQ的几个窗口标题栏。可以看出QQ的窗口都用了同一个窗口图标，一看到这个标志就知道是QQ程序的窗口，同时每个窗口标题栏不同，代表着这个窗口的作用以及用户可以在窗口中进行什么样的操作。



### ****QQ音乐的标题栏****

这里写图片描述

### ****有道词典的标题栏****



### ****PS软件的标题栏****



以上几款软件的标题栏都涵盖了以上几个重要的组成部分。

### ****自定义标题栏的好处****

* 首先自定义标题栏解决了我们程序**在不同平台乃至同一平台不同的版本中样式不同的问题，使用自定义标题栏，能够保证外观的一致性**，这也是为什么像QQ、有道词典、360等软件都没有使用系统标题栏，都是自定义标题栏了。
* 正是因为**自定义标题栏**，我们可以**自定义**，**不再受约束**，想怎么设计就怎么设计。
* 自定义标题栏可以使用自己设计的图标，当然比系统的图标要**美观**了，**颜值**当然更高了。

下面就讲一讲如何用Qt实现自定义窗口标题栏。

# 二、代码之路

这个是之前模仿QQ做的一个登录界面窗口。



这里我们先看一下本程序中简单一点的效果图：



// 标题栏类;

### ****mytitlebar.h****

#include <QWidget>

#include <QLabel>

#include <QPushButton>

#include <QTimer>

enum ButtonType

{

MIN\_BUTTON = 0, // 最小化和关闭按钮;

MIN\_MAX\_BUTTON , // 最小化、最大化和关闭按钮;

ONLY\_CLOSE\_BUTTON // 只有关闭按钮;

};

class MyTitleBar : public QWidget

{

Q\_OBJECT

public:

MyTitleBar(QWidget \*parent);

//这里parent没有给默认值NULL，保证在创建MyTitleBar对象时父指针必须得赋值;且赋值不为NULL;

~MyTitleBar();

// 设置标题栏背景色;

void setBackgroundColor(int r, int g, int b);

// 设置标题栏图标;

void setTitleIcon(QString filePath , QSize IconSize = QSize(25 , 25));

// 设置标题内容;

void setTitleContent(QString titleContent , int titleFontSize = 9);

// 设置标题栏长度;

void setTitleWidth(int width);

// 设置标题栏上按钮类型;

void setButtonType(ButtonType buttonType);

// 设置标题栏中的标题是否会滚动;具体可以看效果;

void setTitleRoll();

// 设置窗口边框宽度;

void setWindowBorderWidth(int borderWidth);

// 保存/获取 最大化前窗口的位置及大小;

void saveRestoreInfo(const QPoint point, const QSize size);

void getRestoreInfo(QPoint& point, QSize& size);

private:

void paintEvent(QPaintEvent \*event);

void mouseDoubleClickEvent(QMouseEvent \*event);

void mousePressEvent(QMouseEvent \*event);

void mouseMoveEvent(QMouseEvent \*event);

void mouseReleaseEvent(QMouseEvent \*event);

// 初始化控件;

void initControl();

// 信号槽的绑定;

void initConnections();

// 加载样式文件;

void loadStyleSheet(const QString &sheetName);

signals:

// 按钮触发的信号;

void signalButtonMinClicked();

void signalButtonRestoreClicked();

void signalButtonMaxClicked();

void signalButtonCloseClicked();

private slots:

// 按钮触发的槽;

void onButtonMinClicked();

void onButtonRestoreClicked();

void onButtonMaxClicked();

void onButtonCloseClicked();

void onRollTitle();

private:

QLabel\* m\_pIcon; // 标题栏图标;

QLabel\* m\_pTitleContent; // 标题栏内容;

QPushButton\* m\_pButtonMin; // 最小化按钮;

QPushButton\* m\_pButtonRestore; // 最大化还原按钮;

QPushButton\* m\_pButtonMax; // 最大化按钮;

QPushButton\* m\_pButtonClose; // 关闭按钮;

// 标题栏背景色;

int m\_colorR;

int m\_colorG;

int m\_colorB;

// 最大化，最小化变量;

QPoint m\_restorePos;

QSize m\_restoreSize;

// 移动窗口的变量;

bool m\_isPressed;

QPoint m\_startMovePos;

// 标题栏跑马灯效果时钟;

QTimer m\_titleRollTimer;

// 标题栏内容;

QString m\_titleContent;

// 按钮类型;

ButtonType m\_buttonType;

// 窗口边框宽度;

int m\_windowBorderWidth;

};

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98

### ****mytitlebar.cpp****

#include "mytitlebar.h"

#include <QHBoxLayout>

#include <QPainter>

#include <QFile>

#include <QMouseEvent>

#define BUTTON\_HEIGHT 30 // 按钮高度;

#define BUTTON\_WIDTH 30 // 按钮宽度;

#define TITLE\_HEIGHT 30 // 标题栏高度;

MyTitleBar::MyTitleBar(QWidget \*parent)

: QWidget(parent)

, m\_colorR(153)

, m\_colorG(153)

, m\_colorB(153)

, m\_isPressed(false)

, m\_buttonType(MIN\_MAX\_BUTTON)

, m\_windowBorderWidth(0)

{

// 初始化;

initControl();

initConnections();

// 加载本地样式 MyTitle.css文件;

loadStyleSheet("MyTitle");

}

MyTitleBar::~MyTitleBar()

{

}

// 初始化控件;

void MyTitleBar::initControl()

{

m\_pIcon = new QLabel;

m\_pTitleContent = new QLabel;

m\_pButtonMin = new QPushButton;

m\_pButtonRestore = new QPushButton;

m\_pButtonMax = new QPushButton;

m\_pButtonClose = new QPushButton;

m\_pButtonMin->setFixedSize(QSize(BUTTON\_WIDTH, BUTTON\_HEIGHT));

m\_pButtonRestore->setFixedSize(QSize(BUTTON\_WIDTH, BUTTON\_HEIGHT));

m\_pButtonMax->setFixedSize(QSize(BUTTON\_WIDTH, BUTTON\_HEIGHT));

m\_pButtonClose->setFixedSize(QSize(BUTTON\_WIDTH, BUTTON\_HEIGHT));

m\_pTitleContent->setObjectName("TitleContent");

m\_pButtonMin->setObjectName("ButtonMin");

m\_pButtonRestore->setObjectName("ButtonRestore");

m\_pButtonMax->setObjectName("ButtonMax");

m\_pButtonClose->setObjectName("ButtonClose");

m\_pButtonMin->setToolTip(QStringLiteral("最小化"));

m\_pButtonRestore->setToolTip(QStringLiteral("还原"));

m\_pButtonMax->setToolTip(QStringLiteral("最大化"));

m\_pButtonClose->setToolTip(QStringLiteral("关闭"));

QHBoxLayout\* mylayout = new QHBoxLayout(this);

mylayout->addWidget(m\_pIcon);

mylayout->addWidget(m\_pTitleContent);

mylayout->addWidget(m\_pButtonMin);

mylayout->addWidget(m\_pButtonRestore);

mylayout->addWidget(m\_pButtonMax);

mylayout->addWidget(m\_pButtonClose);

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

mylayout->setSpacing(0);

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

this->setFixedHeight(TITLE\_HEIGHT);

this->setWindowFlags(Qt::FramelessWindowHint);

}

// 信号槽的绑定;

void MyTitleBar::initConnections()

{

connect(m\_pButtonMin, SIGNAL(clicked()), this, SLOT(onButtonMinClicked()));

connect(m\_pButtonRestore, SIGNAL(clicked()), this, SLOT(onButtonRestoreClicked()));

connect(m\_pButtonMax, SIGNAL(clicked()), this, SLOT(onButtonMaxClicked()));

connect(m\_pButtonClose, SIGNAL(clicked()), this, SLOT(onButtonCloseClicked()));

}

// 设置标题栏背景色,在paintEvent事件中进行绘制标题栏背景色;

//在构造函数中给了默认值，可以外部设置颜色值改变标题栏背景色;

void MyTitleBar::setBackgroundColor(int r, int g, int b)

{

m\_colorR = r;

m\_colorG = g;

m\_colorB = b;

// 重新绘制（调用paintEvent事件）;

update();

}

// 设置标题栏图标;

void MyTitleBar::setTitleIcon(QString filePath, QSize IconSize)

{

QPixmap titleIcon(filePath);

m\_pIcon->setPixmap(titleIcon.scaled(IconSize));

}

// 设置标题内容;

void MyTitleBar::setTitleContent(QString titleContent, int titleFontSize)

{

// 设置标题字体大小;

QFont font = m\_pTitleContent->font();

font.setPointSize(titleFontSize);

m\_pTitleContent->setFont(font);

// 设置标题内容;

m\_pTitleContent->setText(titleContent);

m\_titleContent = titleContent;

}

// 设置标题栏长度;

void MyTitleBar::setTitleWidth(int width)

{

this->setFixedWidth(width);

}

// 设置标题栏上按钮类型;

// 由于不同窗口标题栏上的按钮都不一样，所以可以自定义标题栏中的按钮;

// 这里提供了四个按钮，分别为最小化、还原、最大化、关闭按钮，如果需要其他按钮可自行添加设置;

void MyTitleBar::setButtonType(ButtonType buttonType)

{

m\_buttonType = buttonType;

switch (buttonType)

{

case MIN\_BUTTON:

{

m\_pButtonRestore->setVisible(false);

m\_pButtonMax->setVisible(false);

}

break;

case MIN\_MAX\_BUTTON:

{

m\_pButtonRestore->setVisible(false);

}

break;

case ONLY\_CLOSE\_BUTTON:

{

m\_pButtonMin->setVisible(false);

m\_pButtonRestore->setVisible(false);

m\_pButtonMax->setVisible(false);

}

break;

default:

break;

}

}

// 设置标题栏中的标题是否会自动滚动，跑马灯的效果;

// 一般情况下标题栏中的标题内容是不滚动的，但是既然自定义就看自己需要嘛，想怎么设计就怎么搞O(∩\_∩)O！

void MyTitleBar::setTitleRoll()

{

connect(&m\_titleRollTimer, SIGNAL(timeout()), this, SLOT(onRollTitle()));

m\_titleRollTimer.start(200);

}

// 设置窗口边框宽度;

void MyTitleBar::setWindowBorderWidth(int borderWidth)

{

m\_windowBorderWidth = borderWidth;

}

// 保存窗口最大化前窗口的位置以及大小;

void MyTitleBar::saveRestoreInfo(const QPoint point, const QSize size)

{

m\_restorePos = point;

m\_restoreSize = size;

}

// 获取窗口最大化前窗口的位置以及大小;

void MyTitleBar::getRestoreInfo(QPoint& point, QSize& size)

{

point = m\_restorePos;

size = m\_restoreSize;

}

// 绘制标题栏背景色;

void MyTitleBar::paintEvent(QPaintEvent \*event)

{

//设置背景色;

QPainter painter(this);

QPainterPath pathBack;

pathBack.setFillRule(Qt::WindingFill);

pathBack.addRoundedRect(QRect(0, 0, this->width(), this->height()), 3, 3);

painter.setRenderHint(QPainter::SmoothPixmapTransform, true);

painter.fillPath(pathBack, QBrush(QColor(m\_colorR, m\_colorG, m\_colorB)));

// 当窗口最大化或者还原后，窗口长度变了，标题栏的长度应当一起改变;

// 这里减去m\_windowBorderWidth ，是因为窗口可能设置了不同宽度的边框;

// 如果窗口有边框则需要设置m\_windowBorderWidth的值，否则m\_windowBorderWidth默认为0;

if (this->width() != (this->parentWidget()->width() - m\_windowBorderWidth))

{

this->setFixedWidth(this->parentWidget()->width() - m\_windowBorderWidth);

}

QWidget::paintEvent(event);

}

// 双击响应事件，主要是实现双击标题栏进行最大化和最小化操作;

void MyTitleBar::mouseDoubleClickEvent(QMouseEvent \*event)

{

// 只有存在最大化、还原按钮时双击才有效;

if (m\_buttonType == MIN\_MAX\_BUTTON)

{

// 通过最大化按钮的状态判断当前窗口是处于最大化还是原始大小状态;

// 或者通过单独设置变量来表示当前窗口状态;

if (m\_pButtonMax->isVisible())

{

onButtonMaxClicked();

}

else

{

onButtonRestoreClicked();

}

}

return QWidget::mouseDoubleClickEvent(event);

}

// 以下通过mousePressEvent、mouseMoveEvent、mouseReleaseEvent三个事件实现了鼠标拖动标题栏移动窗口的效果;

void MyTitleBar::mousePressEvent(QMouseEvent \*event)

{

if (m\_buttonType == MIN\_MAX\_BUTTON)

{

// 在窗口最大化时禁止拖动窗口;

if (m\_pButtonMax->isVisible())

{

m\_isPressed = true;

m\_startMovePos = event->globalPos();

}

}

else

{

m\_isPressed = true;

m\_startMovePos = event->globalPos();

}

return QWidget::mousePressEvent(event);

}

void MyTitleBar::mouseMoveEvent(QMouseEvent \*event)

{

if (m\_isPressed)

{

QPoint movePoint = event->globalPos() - m\_startMovePos;

QPoint widgetPos = this->parentWidget()->pos();

m\_startMovePos = event->globalPos();

this->parentWidget()->move(widgetPos.x() + movePoint.x(), widgetPos.y() + movePoint.y());

}

return QWidget::mouseMoveEvent(event);

}

void MyTitleBar::mouseReleaseEvent(QMouseEvent \*event)

{

m\_isPressed = false;

return QWidget::mouseReleaseEvent(event);

}

// 加载本地样式文件;

// 可以将样式直接写在文件中，程序运行时直接加载进来;

void MyTitleBar::loadStyleSheet(const QString &sheetName)

{

QFile file(":/Resources/" + sheetName + ".css");

file.open(QFile::ReadOnly);

if (file.isOpen())

{

QString styleSheet = this->styleSheet();

styleSheet += QLatin1String(file.readAll());

this->setStyleSheet(styleSheet);

}

}

// 以下为按钮操作响应的槽;

void MyTitleBar::onButtonMinClicked()

{

emit signalButtonMinClicked();

}

void MyTitleBar::onButtonRestoreClicked()

{

m\_pButtonRestore->setVisible(false);

m\_pButtonMax->setVisible(true);

emit signalButtonRestoreClicked();

}

void MyTitleBar::onButtonMaxClicked()

{

m\_pButtonMax->setVisible(false);

m\_pButtonRestore->setVisible(true);

emit signalButtonMaxClicked();

}

void MyTitleBar::onButtonCloseClicked()

{

emit signalButtonCloseClicked();

}

// 该方法主要是让标题栏中的标题显示为滚动的效果;

void MyTitleBar::onRollTitle()

{

static int nPos = 0;

QString titleContent = m\_titleContent;

// 当截取的位置比字符串长时，从头开始;

if (nPos > titleContent.length())

nPos = 0;

m\_pTitleContent->setText(titleContent.mid(nPos));

nPos++;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151
* 152
* 153
* 154
* 155
* 156
* 157
* 158
* 159
* 160
* 161
* 162
* 163
* 164
* 165
* 166
* 167
* 168
* 169
* 170
* 171
* 172
* 173
* 174
* 175
* 176
* 177
* 178
* 179
* 180
* 181
* 182
* 183
* 184
* 185
* 186
* 187
* 188
* 189
* 190
* 191
* 192
* 193
* 194
* 195
* 196
* 197
* 198
* 199
* 200
* 201
* 202
* 203
* 204
* 205
* 206
* 207
* 208
* 209
* 210
* 211
* 212
* 213
* 214
* 215
* 216
* 217
* 218
* 219
* 220
* 221
* 222
* 223
* 224
* 225
* 226
* 227
* 228
* 229
* 230
* 231
* 232
* 233
* 234
* 235
* 236
* 237
* 238
* 239
* 240
* 241
* 242
* 243
* 244
* 245
* 246
* 247
* 248
* 249
* 250
* 251
* 252
* 253
* 254
* 255
* 256
* 257
* 258
* 259
* 260
* 261
* 262
* 263
* 264
* 265
* 266
* 267
* 268
* 269
* 270
* 271
* 272
* 273
* 274
* 275
* 276
* 277
* 278
* 279
* 280
* 281
* 282
* 283
* 284
* 285
* 286
* 287
* 288
* 289
* 290
* 291
* 292
* 293
* 294
* 295
* 296
* 297
* 298
* 299
* 300
* 301
* 302
* 303
* 304
* 305
* 306
* 307
* 308
* 309
* 310
* 311
* 312
* 313

// 窗口基类;

### ****basewindow.h****

#include <QWidget>

#include "mytitlebar.h"

class BaseWindow : public QWidget

{

Q\_OBJECT

public:

BaseWindow(QWidget \*parent = 0);

~BaseWindow();

private:

void initTitleBar();

void paintEvent(QPaintEvent \*event);

void loadStyleSheet(const QString &sheetName);

private slots:

void onButtonMinClicked();

void onButtonRestoreClicked();

void onButtonMaxClicked();

void onButtonCloseClicked();

protected:

MyTitleBar\* m\_titleBar;

};

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26

### basewindow.cpp

#include "basewindow.h"

#include <QDesktopWidget>

#include <QApplication>

#include <QPainter>

#include <QFile>

BaseWindow::BaseWindow(QWidget \*parent)

: QWidget(parent)

{

// FramelessWindowHint属性设置窗口去除边框;

// WindowMinimizeButtonHint 属性设置在窗口最小化时，点击任务栏窗口可以显示出原窗口;

this->setWindowFlags(Qt::FramelessWindowHint | Qt::WindowMinimizeButtonHint);

// 设置窗口背景透明;

setAttribute(Qt::WA\_TranslucentBackground);

// 关闭窗口时释放资源;

setAttribute(Qt::WA\_DeleteOnClose);

// 初始化标题栏;

initTitleBar();

}

BaseWindow::~BaseWindow()

{

}

void BaseWindow::initTitleBar()

{

m\_titleBar = new MyTitleBar(this);

m\_titleBar->move(0, 0);

connect(m\_titleBar, SIGNAL(signalButtonMinClicked()), this, SLOT(onButtonMinClicked()));

connect(m\_titleBar, SIGNAL(signalButtonRestoreClicked()), this, SLOT(onButtonRestoreClicked()));

connect(m\_titleBar, SIGNAL(signalButtonMaxClicked()), this, SLOT(onButtonMaxClicked()));

connect(m\_titleBar, SIGNAL(signalButtonCloseClicked()), this, SLOT(onButtonCloseClicked()));

}

void BaseWindow::paintEvent(QPaintEvent\* event)

{

//设置背景色;

QPainter painter(this);

QPainterPath pathBack;

pathBack.setFillRule(Qt::WindingFill);

pathBack.addRoundedRect(QRect(0, 0, this->width(), this->height()), 3, 3);

painter.setRenderHint(QPainter::SmoothPixmapTransform, true);

painter.fillPath(pathBack, QBrush(QColor(238, 223, 204)));

return QWidget::paintEvent(event);

}

void BaseWindow::loadStyleSheet(const QString &sheetName)

{

QFile file(":/Resources/" + sheetName + ".css");

file.open(QFile::ReadOnly);

if (file.isOpen())

{

QString styleSheet = this->styleSheet();

styleSheet += QLatin1String(file.readAll());

this->setStyleSheet(styleSheet);

}

}

void BaseWindow::onButtonMinClicked()

{

showMinimized();

}

void BaseWindow::onButtonRestoreClicked()

{

QPoint windowPos;

QSize windowSize;

m\_titleBar->getRestoreInfo(windowPos, windowSize);

this->setGeometry(QRect(windowPos, windowSize));

}

void BaseWindow::onButtonMaxClicked()

{

m\_titleBar->saveRestoreInfo(this->pos(), QSize(this->width(), this->height()));

QRect desktopRect = QApplication::desktop()->availableGeometry();

QRect FactRect = QRect(desktopRect.x() - 3, desktopRect.y() - 3, desktopRect.width() + 6, desktopRect.height() + 6);

setGeometry(FactRect);

}

void BaseWindow::onButtonCloseClicked()

{

close();

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87

以上代码实现了自定义标题栏,新建的窗口类**直接继承BaseWindow类，并对标题栏进行设置即可**，下面就测试一下吧！

### ****测试代码****

### ****NoFocusWidget.cpp****

#include "nofocuswidget.h"

NoFocusWidget::NoFocusWidget(QWidget \*parent)

: BaseWindow(parent)

{

ui.setupUi(this);

// 初始化标题栏;

initTitleBar();

}

NoFocusWidget::~NoFocusWidget()

{

}

void NoFocusWidget::initTitleBar()

{

// 设置标题栏跑马灯效果，可以不设置;

m\_titleBar->setTitleRoll();

m\_titleBar->setTitleIcon(":/Resources/titleicon.png");

m\_titleBar->setTitleContent(QStringLiteral("前行中的小猪-前行之路还需前行"));

m\_titleBar->setButtonType(MIN\_MAX\_BUTTON);

m\_titleBar->setTitleWidth(this->width());

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26

### ****main.cpp****

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

NoFocusWidget w;

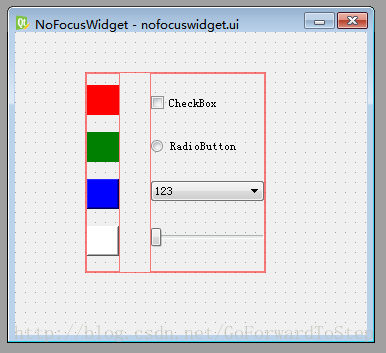
w.show();

return a.exec();

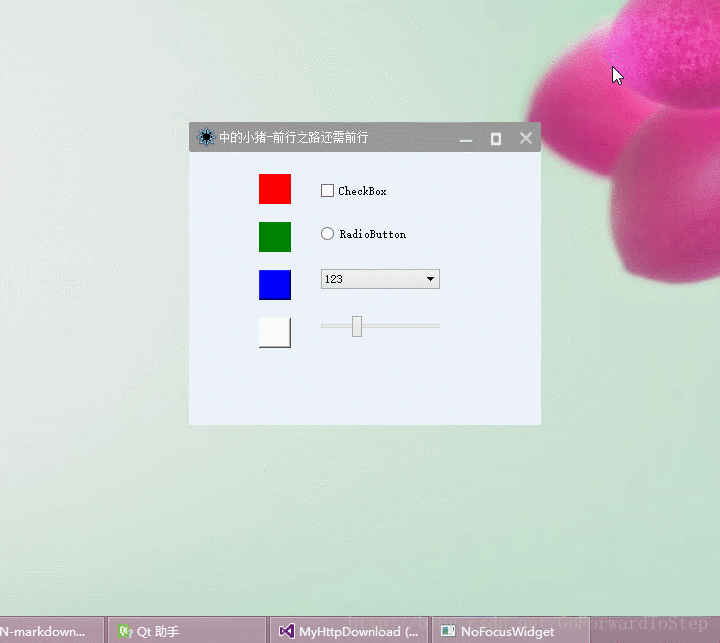
}

* 1
* 2
* 3
* 4
* 5
* 6
* 7

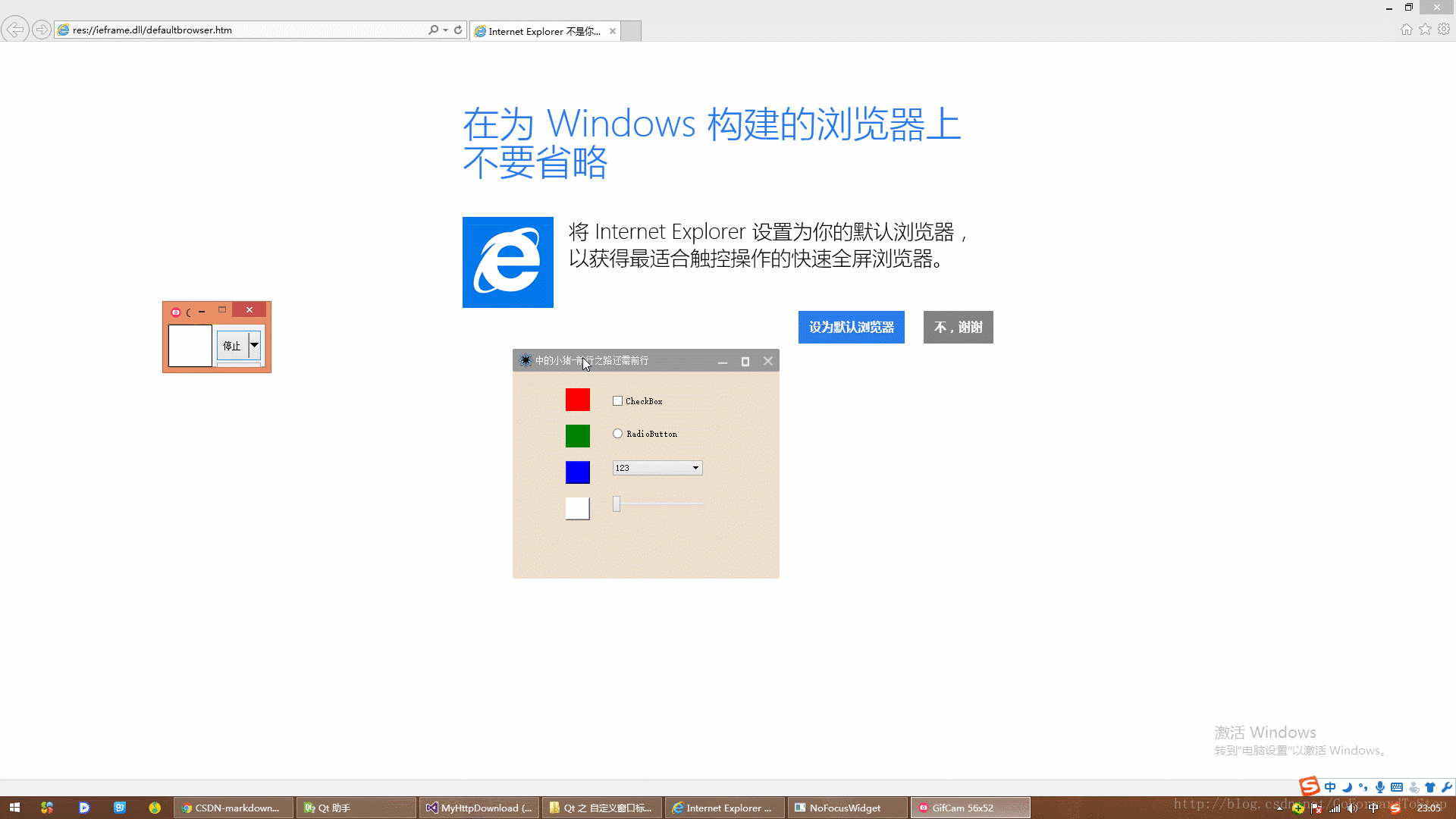
NoFocusWidget为新建的窗口类，通过继承了**BaseWindow类**，并在**initTitleBar()方法中对标题栏进行设置**即可。下图为**NoFocusWidget窗口的设计图**。



### ****动态效果图****



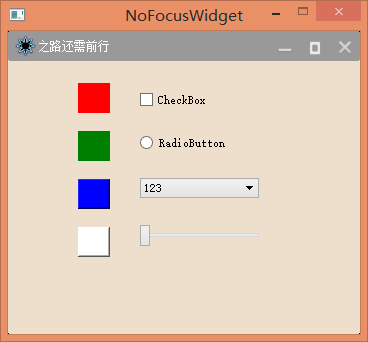
### ****最大化、还原效果图****



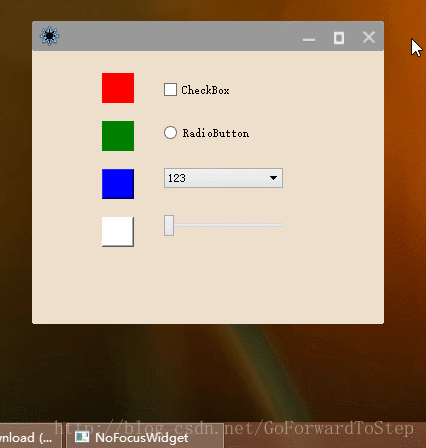
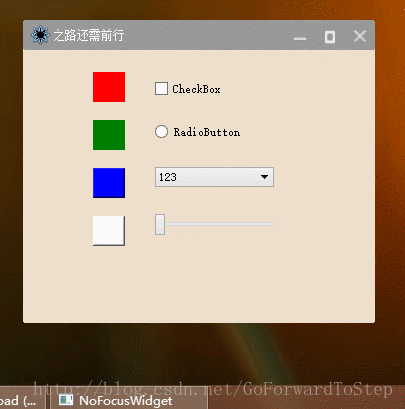
### ****注意****

如果在设置窗口属性时不加上Qt::FramelessWindowHint 就是左图效果。

this->setWindowFlags(Qt::FramelessWindowHint | Qt::WindowMinimizeButtonHint);



如果不加上 Qt::WindowMinimizeButtonHint 就是左图效果。加上之后可以看右图效果。

# 尾

以上代码用Qt实现了自定义窗口标题栏，代码很简单，但是很实用，可以应用到每一个窗口中去，保持每个窗口外观的一致性，同时自定义的标题栏也比系统自带的漂亮很多，所以还是非常实用的。

目前我们在**MyTitleBar**类以及**BaseWindow**类中只添加了一些基础功能，小伙伴们可以根据需要添加更多自定义的功能，而其他窗口只需继承**BaseWindow**类即可。

欢迎大家一起交流 O(∩\_∩)O！

20170927

<http://blog.csdn.net/scien2011/article/details/52830794>

关于安装OpenCV到qt中

20170913

1. 今天在使用addStretch，布局的时候，发现addStretch竟然是可以平均分配的，有意思。比如：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | QVBoxLayout \*buttonLayout = new QVBoxLayout;  buttonLayout->addStretch(1);  buttonLayout->addWidget(Button1);  buttonLayout->addStretch(1);  buttonLayout->addWidget(Button2);  buttonLayout->addStretch(1);  buttonLayout->addWidget(Button3);  buttonLayout->addStretch(6); |

您会发现，buttonLayout的布局将空白没有widget的地方分成了9份，然后按照您所规定的地方分配弹簧，于是布局起来就方便多了，以前没有发现这个，一直还为布局头疼呢，现在好了点。

# 2. Qt之QLabel

[一去二三里](https://yq.aliyun.com/users/1312259910216341?spm=5176.100239.blogcont24416.2.nhsdH7) 2016-03-12 18:01:00 浏览629 评论0

**摘要：** 简述 QLabel提供了一个文本或图像的显示，没有提供用户交互功能。 一个QLabel可以包含以下任意内容类型： 内容 设置 纯文本 使用setText()设置一个QString 富文本 使用setText()设置一个富文本的QString 图像 使用setPixmap()设置一个图像 动画 使用setMovie()设置一个动画 数字 使用setNum()设置int或double，并转换为纯文本。

简述

QLabel提供了一个文本或图像的显示，没有提供用户交互功能。

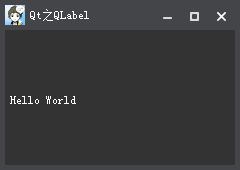
一个QLabel可以包含以下任意内容类型：

| **内容** | **设置** |
| --- | --- |
| 纯文本 | 使用setText()设置一个QString |
| 富文本 | 使用setText()设置一个富文本的QString |
| 图像 | 使用setPixmap()设置一个图像 |
| 动画 | 使用setMovie()设置一个动画 |
| 数字 | 使用setNum()设置int或double，并转换为纯文本。 |
| Nothing | 空的纯文本，默认的，使用clear()设置 |

* [简述](https://yq.aliyun.com/articles/24416#%E7%AE%80%E8%BF%B0)
* [纯文本](https://yq.aliyun.com/articles/24416#%E7%BA%AF%E6%96%87%E6%9C%AC)
  + [显示](https://yq.aliyun.com/articles/24416#%E6%98%BE%E7%A4%BA)
  + [对齐方式](https://yq.aliyun.com/articles/24416#%E5%AF%B9%E9%BD%90%E6%96%B9%E5%BC%8F)
  + [自动换行](https://yq.aliyun.com/articles/24416#%E8%87%AA%E5%8A%A8%E6%8D%A2%E8%A1%8C)
  + [设置行高](https://yq.aliyun.com/articles/24416#%E8%AE%BE%E7%BD%AE%E8%A1%8C%E9%AB%98)
  + [省略](https://yq.aliyun.com/articles/24416#%E7%9C%81%E7%95%A5)
  + [垂直显示](https://yq.aliyun.com/articles/24416#%E5%9E%82%E7%9B%B4%E6%98%BE%E7%A4%BA)
* [富文本](https://yq.aliyun.com/articles/24416#%E5%AF%8C%E6%96%87%E6%9C%AC)
* [图像](https://yq.aliyun.com/articles/24416#%E5%9B%BE%E5%83%8F)
* [动画](https://yq.aliyun.com/articles/24416#%E5%8A%A8%E7%94%BB)
* [数字](https://yq.aliyun.com/articles/24416#%E6%95%B0%E5%AD%97)
* [超链接](https://yq.aliyun.com/articles/24416#%E8%B6%85%E9%93%BE%E6%8E%A5)
  + [方法一](https://yq.aliyun.com/articles/24416#%E6%96%B9%E6%B3%95%E4%B8%80)
  + [方法二](https://yq.aliyun.com/articles/24416#%E6%96%B9%E6%B3%95%E4%BA%8C)
* [总结](https://yq.aliyun.com/articles/24416#%E6%80%BB%E7%BB%93)

纯文本

显示



首先我们构造一个QLabel对象，其中this为其所在的父窗体。通过调用setText可以为标签设置文本（Hello World），这时标签就可以正常显示出来了。为了显示更佳的效果，我们可以通过调用setStyleSheet来设置样式。color: white-顾名思义，就是为标签设置一个文本色（白色）。

QLabel \*pLabel = new QLabel(this);

pLabel->setText("Hello World");

pLabel->setStyleSheet("color: white");

对齐方式

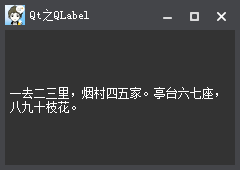
默认的标签文本对齐方式为：左对齐、垂直居中，我们可以通过setAlignment来设置，包括：左、上、右、下、居中对齐，一般情况，我们会进行两两组合（水平方向、垂直方向）。比如：居中对齐

setAlignment(Qt::AlignCenter);

自动换行

如果文本过长，我们可以采用自动换行的方式来显示。

setWordWrap(true);



注意：当使用英文的时候，如果写为类似形式”abcdefghijklmnopqrstuvwxyz”则是不能换行的，why？因为中间没有空格，所以需要写为”abcde fghij klmno pqrst uvwxyz”。

设置行高

一般情况下，自动换行之后文本上下行会距离比较近，我们可以通过下面方式来设置行高。

pLabel->setWordWrap(true);

QString strText = QStringLiteral("一去二三里，烟村四五家。亭台六七座，八九十枝花。");

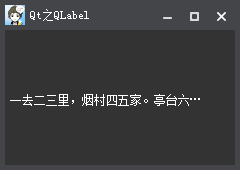
QString strHeightText = "<p style=\"line-height:%1%\">%2<p>";

strText = strHeightText.arg(150).arg(strText);

pLabel->setText(strText);

省略

如果过长，我们又不想换行，只想把其中一部分省略为…，那么我们可以通过QFontMetrics来实现，这里先不介绍QFontMetrics，感兴趣的童鞋可以先自行研究。



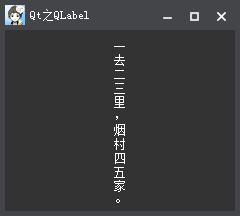
QString strText = QStringLiteral("一去二三里，烟村四五家。亭台六七座，八九十枝花。");

QString strElidedText = pLabel->fontMetrics().elidedText(strText, Qt::ElideRight, 200, Qt::TextShowMnemonic);

pLabel->setText(strElidedText);

垂直显示

默认情况下，文本显示方式为水平方向，如果我们需要在垂直方向上显示，需要用一些小技巧来处理。



QString strText = QStringLiteral("一去二三里，烟村四五家。");

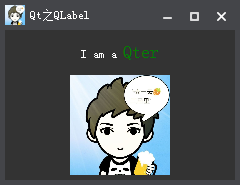
pLabel->setText(strText.split("", QString::SkipEmptyParts).join("\n"));

pLabel->setAlignment(Qt::AlignCenter);

富文本

我们可以在助手中查找关于Using HTML Markup in Text Widgets的资料，查看Qt支持哪些HTML标记。

下面我们来写一段HTML代码，O(∩\_∩)O哈哈~。。。显示不同颜色的文本以及图片



QString strHTML = QString("<html> \

<head> \

<style> \

font{color:white;} #f{font-size:18px; color: green;} \

</style> \

</head> \

<body>\

<font>%1</font><font id=\"f\">%2</font> \

<br/><br/> \

<img src=\":/Images/logo\" width=\"100\" height=\"100\"> \

</body> \

</html>").arg("I am a ").arg("Qter");

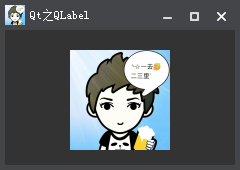
pLabel->setText(strHTML);

pLabel->setAlignment(Qt::AlignCenter);

是不是很神奇？经常我们要用好几个控件来组合才能实现的功能，就仅仅几行HTML代码就搞定了。。。何乐而不为！

图像

首先我们构建一个QPixmap来作为显示的图片，然后设置标签的大小，可以通过setScaledContents按比例缩放图片达到理想的效果。



QPixmap pixmap(":/Images/logo");

pLabel->setPixmap(pixmap);

pLabel->setFixedSize(100, 100);

pLabel->setScaledContents(true);

动画

这里我们需要使用另外一个类QMovie来控制动画，start()可以进行播放与stop()则可以停止，也可以通过调用setSpeed()来设置动画的播放速度。



QMovie \*pMovie = new QMovie(":/Images/movie");

pLabel->setMovie(pMovie);

pLabel->setFixedSize(135, 200);

pLabel->setScaledContents(true);

pMovie->start();

数字

如果我们需要显示一个数字，则可以调用setNum()，他可以将内容转换为纯文本。

pLabel->setNum(66.6);

很简单就一段代码，我们可以打开源码瞅瞅，究竟setNum是如何实现的。

void QLabel::setNum(int num)

{

QString str;

str.setNum(num);

setText(str);

}

就三行代码，呵呵哒。。。超简单吧，我们要有一颗随时看源码的心！

超链接

我们需要简单使用标签<a></a>写一段简单的HTML超链接代码

方法一

比较简单，直接调用setOpenExternalLinks(true)即可。

pLabel->setText(QString("<a href = \"%1\">%2</a>").arg("http://blog.csdn.net/liang19890820").arg(QStringLiteral("一去丶二三里")));

pLabel->setOpenExternalLinks(true);

方法二

声明一个槽openUrl，将其与linkActivated信号关联。

pLabel->setText(QString("<a href = \"%1\">%2</a>").arg("http://blog.csdn.net/liang19890820").arg(QStringLiteral("一去丶二三里")));

connect(pLabel, SIGNAL(linkActivated(QString)), this, SLOT(openUrl(QString)));

void MainWindow::openUrl(const QString &link)

{

QDesktopServices::openUrl(QUrl(link));

}

总结

通过上面的学习，我们基本将标签的大部分用法都分享了，可以看出HTML、CSS的重要性，所以无论你现在所从事的工作是什么，或者以后做什么，我都建议好好研究下Web，技多不压身。。。上面的内容虽多，但都很简单，也很重要，所以都需要掌握，我们不只是Qter，我们更是工匠-精益求精、严谨、耐心，专注，坚持。