

# UI美化实战课程(布局/样式表/高级控件/图标字体/图表曲线/仪表盘/精美换肤/未完待续)

## 第一章 无边框窗口（11讲）

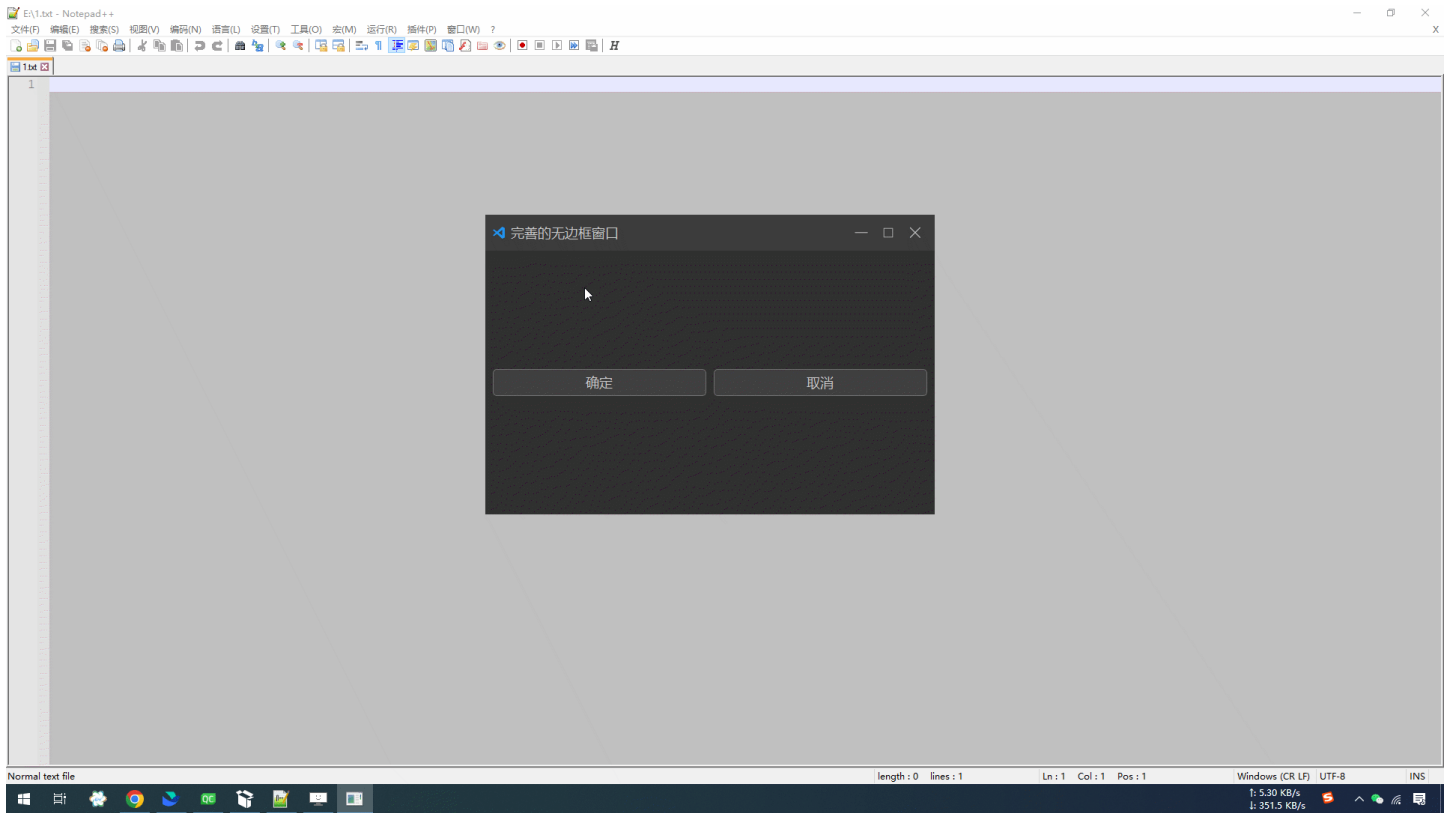
### 1. 效果演示、技术点

#### 1.1 效果演示

本章实现的无边框窗口，支持以下特性：

- ✓ 支持鼠标拖拽4个边框和4个角，来缩放窗体
- ✓ 同时支持鼠标拖动标题栏以及标题栏之外，来移动窗体
- ✓ 支持双击标题栏，来实现窗口最大化和恢复
- ✓ 最大化状态下，拖动标题栏恢复原始大小
- ✓ 支持窗口固定大小，不允许缩放

本节演示下最终实现的无边框窗口的效果，如下：



## 1.2 实现方式

之所以要自己定制无边框窗口，是因为 `QT` 自带的标题栏很丑，无法定制。

所以实际项目中的做法是：去除 `QT` 自带的标题栏，然后自己实现一个带有自定义标题栏的无边框窗口。

通常有两种做法：

### (1) 重载 `nativeEvent()` 函数


该函数拦截 `Windows` 系统消息（如 `WM_NCHITTEST`）来实现

这种方法需要了解 `Windows` 系统消息，并且是针对 `Windows` 系统的，对跨平台显然无法支持

### (2) 纯 `QT` 实现

这种方式通过处理 `QT` 中的一些事件，比如鼠标事件（`mouseMoveEvent`、`mousePressEvent`、`mouseReleaseEvent` 等），通过重载相应的事件处理函数，实现窗口的拉伸缩放、窗口移动等

这种方法是纯 `QT` 代码实现，支持跨平台

 **PS：**网上能查到的各种实现，各种小 `BUG`，包括但不限于：

1. 缩放窗口时，把窗口 “推走” ；
2. 最大化状态变为最小化后，再次恢复不是最大化状态
3. 点击标题栏后，右上角的按钮失去 “活性”
4. 右上角的按钮，鼠标悬浮上去后底纹背景无法变化

本章会从零开始、从零新建项目实现这个完善的无边框窗口，可以直接应用到商业项目中去！

## 2. 新建工程、去除自带标题栏

### 2.1 新建工程

从零新建一个基于 `Qt Widgets Application` 的项目：

#### Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name:	MyWidget	类名
Base class:	QWidget	父类
Header file:	mywidget.h	
Source file:	mywidget.cpp	
	<input type="checkbox"/> Generate form	不使用设计师界面，而是用纯代码实现
Form file:	mywidget.ui	

项目创建完毕，结构如下：

FramelessDemo

FramelessDemo.pro

头文件

mywidget.h

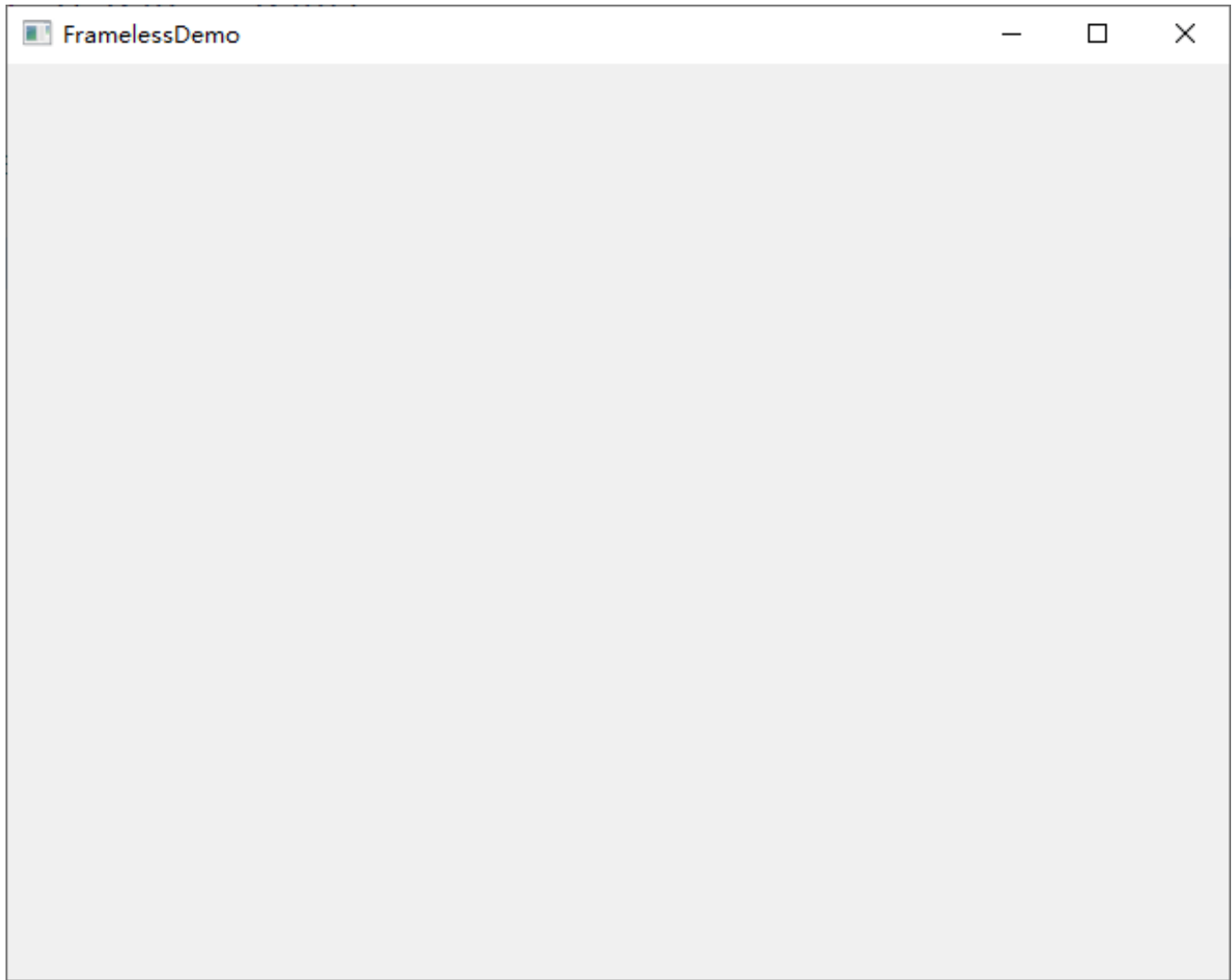
源文件

main.cpp

mywidget.cpp

```
1  #include "mywidget.h"
2
3  #include <QApplication>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      MyWidget w;
9      w.show();
10     return a.exec();
11 }
```

此时运行，是一个空白窗口，如下：



## 2.2 添加按钮，设置背景色

在 `MyWidget` 类中封装一个 `initForm()` 函数，用于初始化界面：

添加两个水平布局的按钮，并设置窗体以及按钮的样式。

首先，来到 `mywidget.h` 中，声明一个 `initForm()` 函数，如下：

```
1 class MyWidget : public QWidget
2 {
3 private:
4     void initForm();
5 };
```

然后，来到 `mywidget.cpp` 中，实现 `initForm()` 函数，如下：

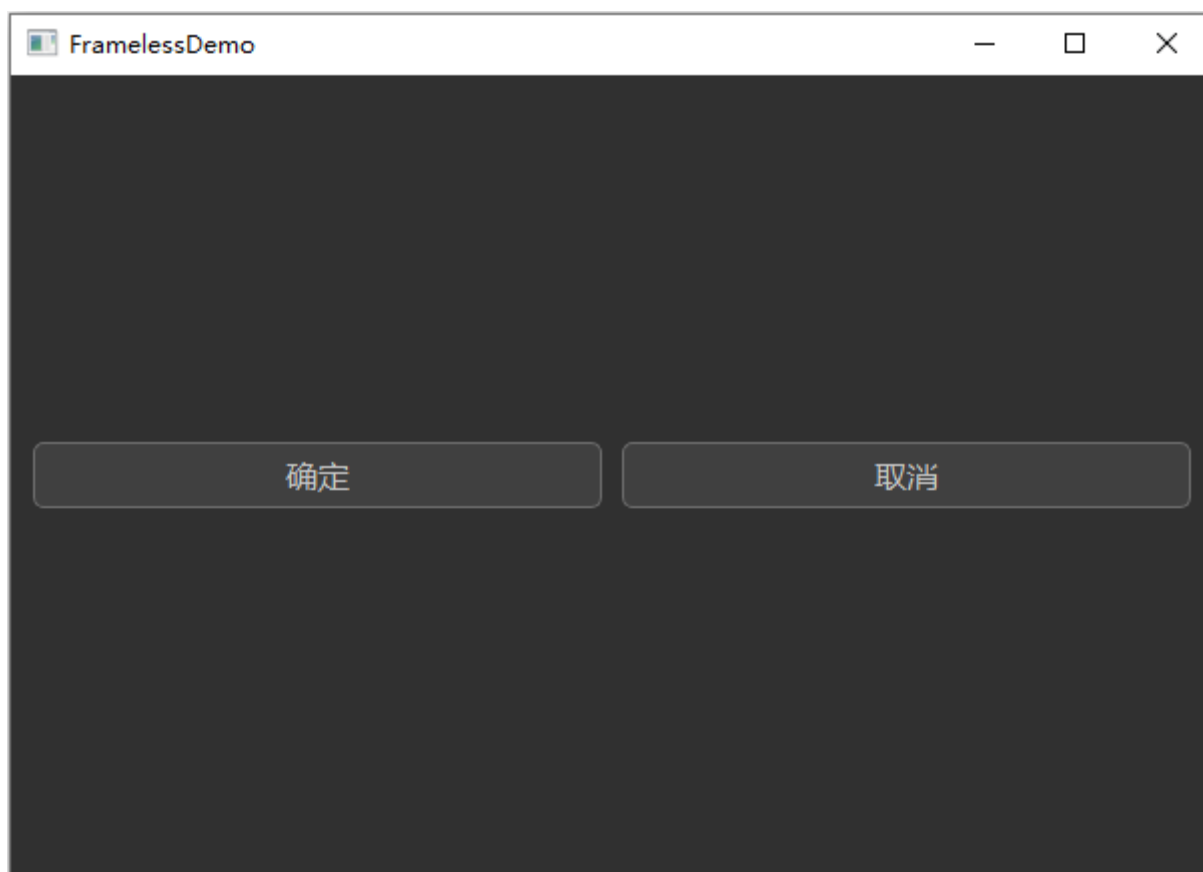
```
1 #include <QPushButton>
2 #include <QHBoxLayout>
3 #include <QDebug>
4
5 MyWidget::MyWidget(QWidget* parent) : QWidget(parent)
6 {
7     initForm();
8 }
9
10 void MyWidget::initForm()
11 {
12     // 1. 设置窗口
13     // 1.1 设置最小宽高
14     this->setMinimumWidth(600);
15     this->setMinimumHeight(400);
16
17     // 1.2 设置背景色
18     this->setStyleSheet("background:#303030");
19
20     // 1.3 当前窗口采用水平布局
21     QHBoxLayout* layout = new QHBoxLayout(this);
22     layout->setSpacing(10);
23
24     // 2. 添加按钮
25     // 2.1 创建两个按钮，并添加到布局中
26     QPushButton* btn1 = new QPushButton("确定", this);
27     QPushButton* btn2 = new QPushButton("取消", this);
28
29     layout->addWidget(btn1);
30     layout->addWidget(btn2);
31
32     // 2.2 为按钮设置样式表
33     QString style = R"(
34         QPushButton {
35             background-color: rgb(64, 64, 64);
36             font:16px "Microsoft YaHei";
37             color:rgb(200,200,200);
38             border: 1px solid #707070;
39             border-radius: 5px;
40             padding: 5px;
41         }
42         QPushButton:hover {
43             background-color: rgb(40, 40, 40);
```

```

44     }
45     QPushButton:pressed {
46         background-color: rgb(64, 64, 64);
47     }
48     }";
49
50     btn1->setStyleSheet(style);
51     btn2->setStyleSheet(style);
52
53     // 2.3 关联信号槽
54     connect(btn1, &QPushButton::clicked, this, [=] { qDebug() << btn1->text();
55     });
56     connect(btn2, &QPushButton::clicked, this, [=] { qDebug() << btn2->text();
57     });
58 }

```

此时运行，效果如下：

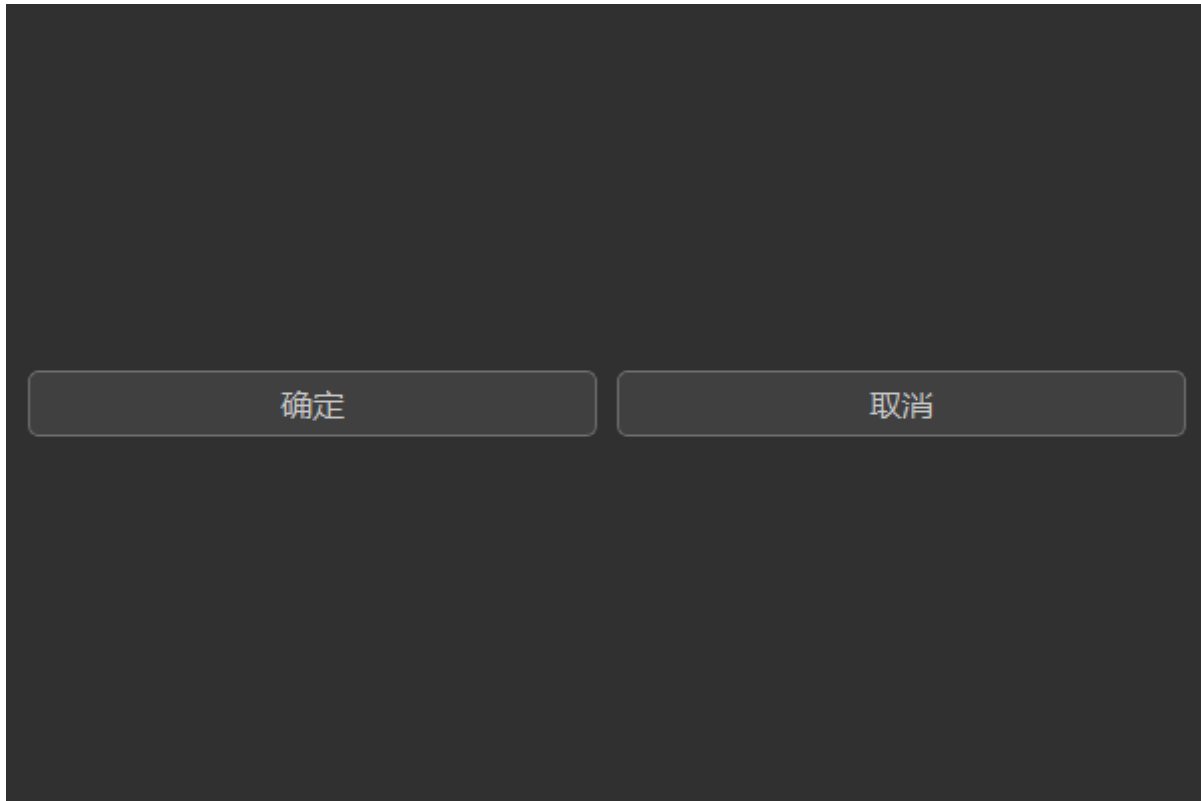


## 2.3 去除标题栏

自带的标题栏很丑，在 `initForm()` 中，添加一条语句即可去除标题栏，如下：

```
1 void MyWidget::initForm()
2 {
3     // 1.2 设置背景色、去除标题栏
4     this->setStyleSheet("background:#303030");
5     this->setWindowFlags(windowFlags() | Qt::FramelessWindowHint);
6 }
```

此时，运行效果如下：



### 3. 添加自定义标题栏






没有了标题栏，也就没有了右上角的 “关闭” 按钮，就无法退出程序了

接下来添加标题栏，并实现最大化、最小化、关闭功能

#### 3.1 添加资源文件

标题栏左侧的图标，以及最小化/最大化/关闭按钮的图标，先作为资源文件添加进项目中。

首先，在项目文件夹下新建 `res` 文件夹，并拷贝所需图标到该文件夹，如下：

FramelessDemo > res			
名称	修改日期	类型	大小
 close_20.svg	2023-03-22 12:28	Microsoft Edge HTML Document	1 KB
 maximize_20_max.svg	2023-09-06 17:50	Microsoft Edge HTML Document	1 KB
 maximize_20_normal.svg	2023-03-22 14:42	Microsoft Edge HTML Document	1 KB
 minimize_20.svg	2023-03-22 14:16	Microsoft Edge HTML Document	1 KB
 title_icon_20.svg	2023-03-22 11:44	Microsoft Edge HTML Document	1 KB

然后，右键项目名称，选择【添加新文件...】，选择【Qt】->【Qt Resource File】，如下：

Location

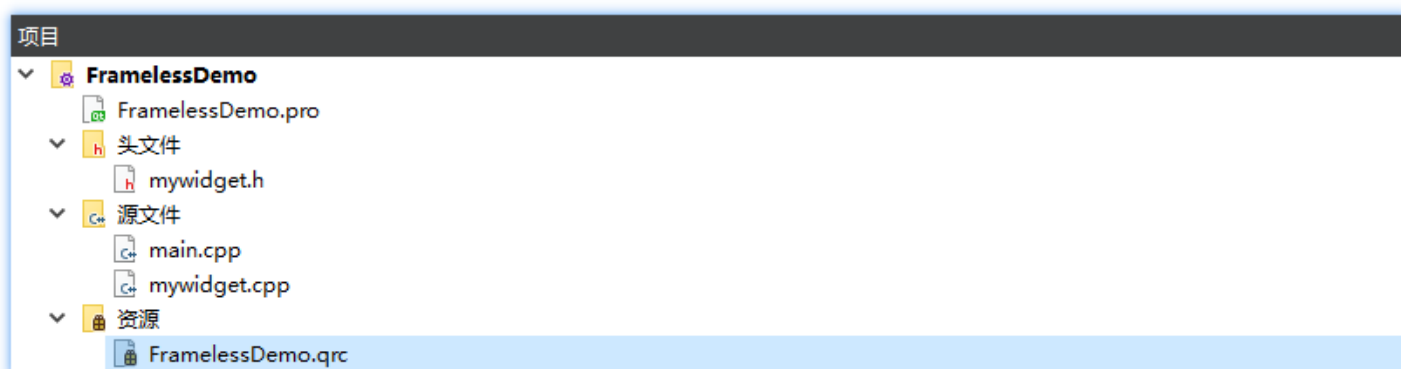
The default suffix if you do not explicitly specify a file extension is ".qrc".

文件名:  资源文件名，随便起，这里和项目名相同

路径:  资源文件保存在当前项目的根目录下

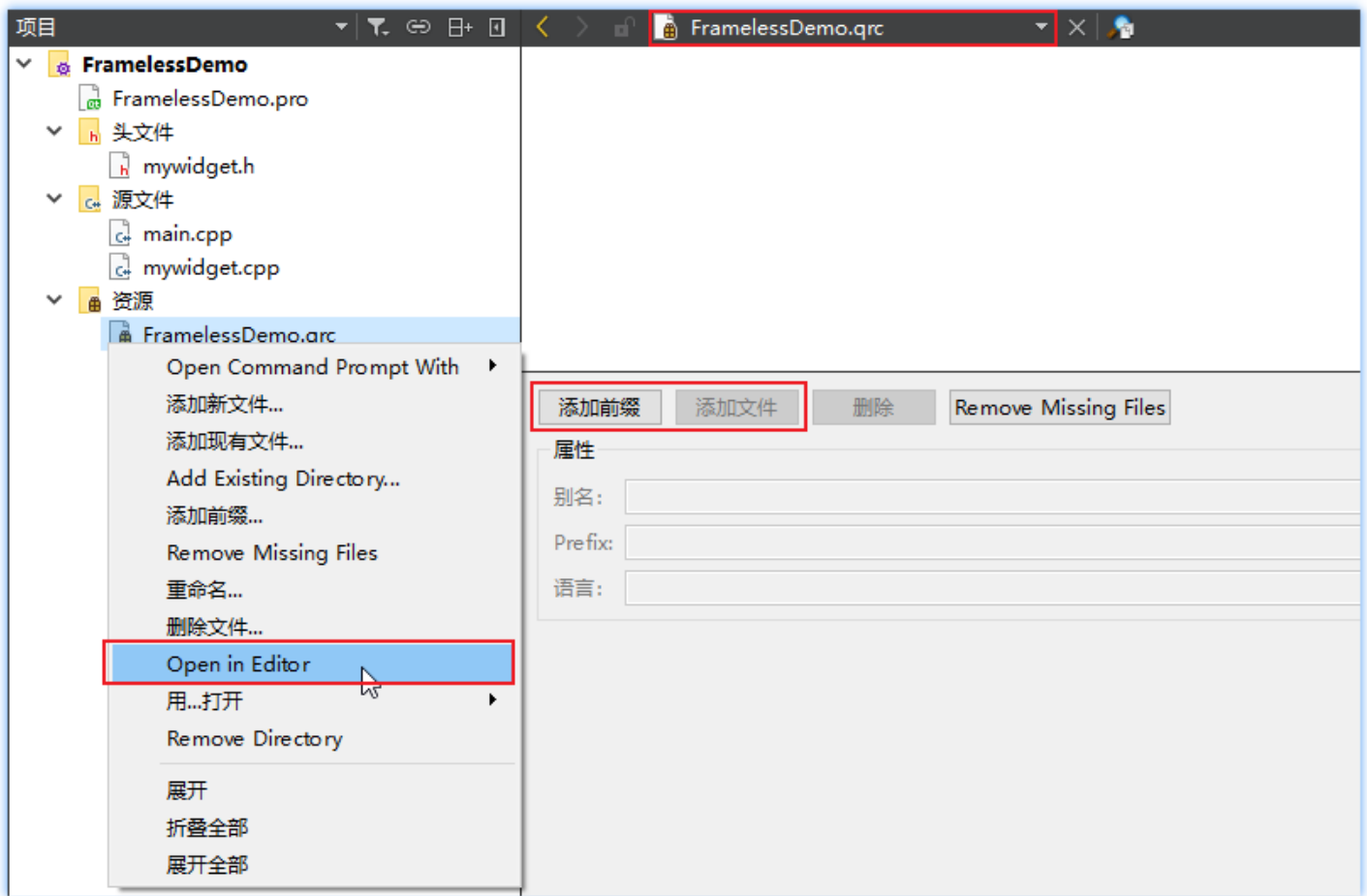
然后，点击完成，就会在项目的根目录下创建一个名为 `FramelessDemo.qrc` 的资源文件

此时的项目结构如下：

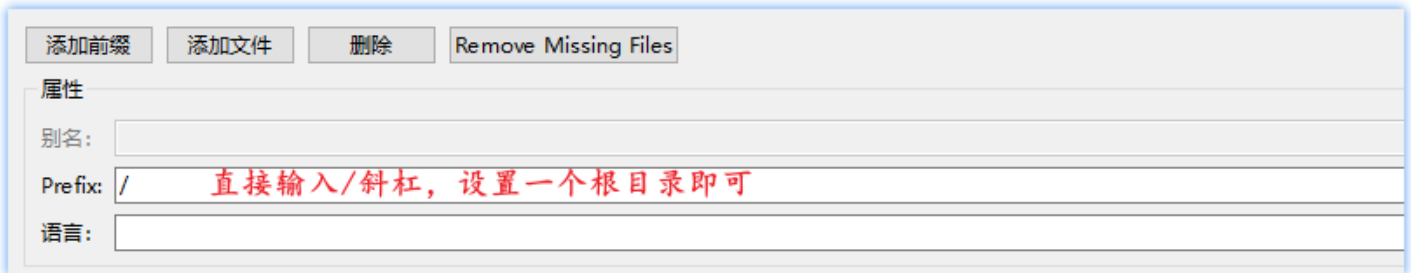


然后，右键 `FramelessDemo.qrc` 资源文件，选择【Open in Editor】，打开资源文件，如下：



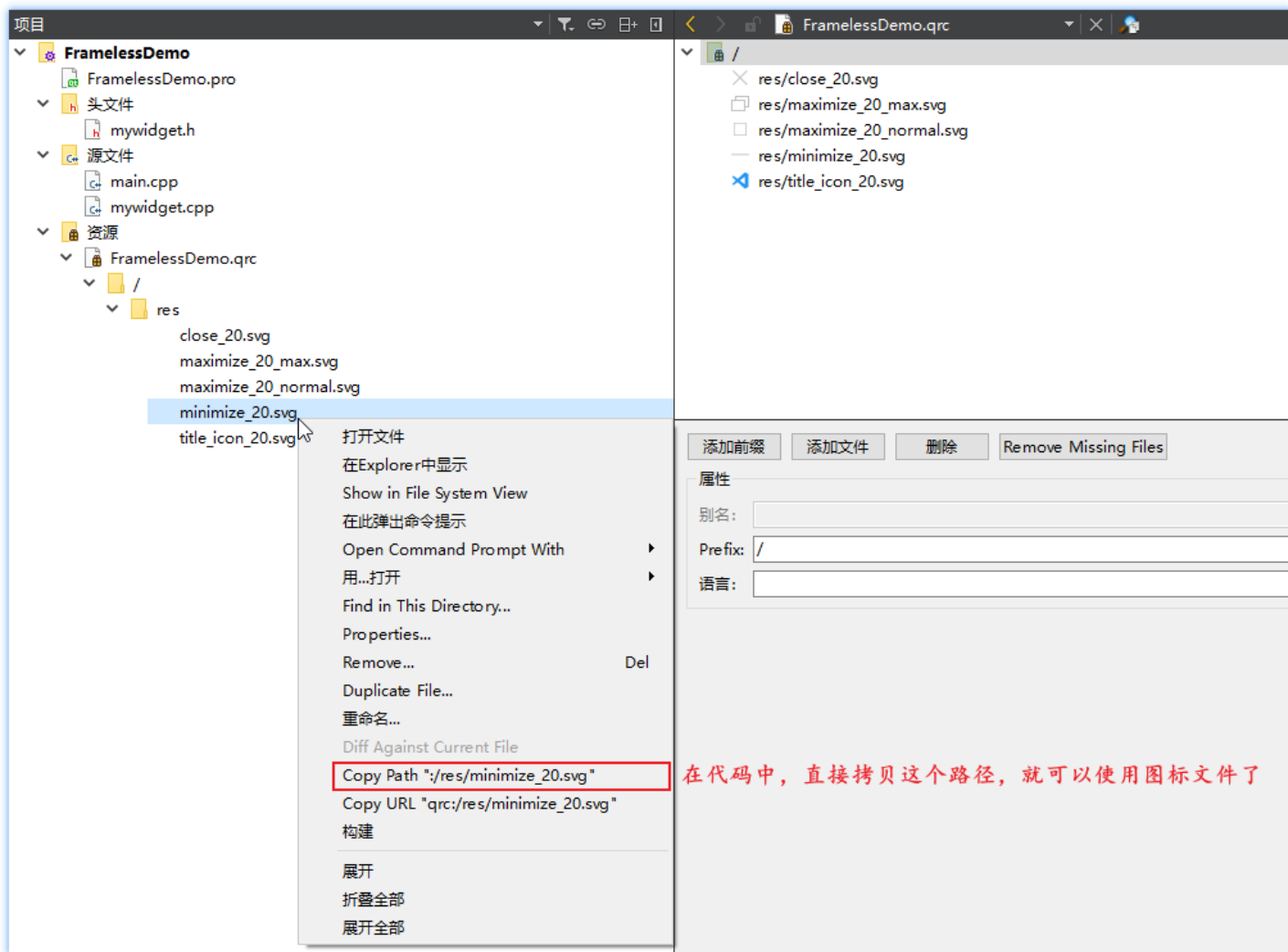


然后，点击【Add Prefix】，添加前缀：



然后，点击【Add Files】，选中项目下的 `res` 目录中的图标文件，添加进来。

之后，直接拷贝图标的资源文件路径，就可以在代码中使用了，如下：



## 3.2 窗口整体规划

窗口整体采用垂直布局，顶部是标题栏，底部是主体区域

- 顶部标题栏
  - 包含图标、文本、最小化最大化关闭按钮
  - 整体采用水平布局
- 底部的主体区域
  - 窗口的主体区域，目前暂放置两个水平布局的按钮
  - 整体采用水平布局

首先，来到 `mywidget.h` 文件，将按钮、标签等定义为成员变量，如下：

```
1 #include <QLabel>
2 #include <QPushButton>
3
```

```
4 class MyWidget : public QWidget
5 {
6 private:
7     // 标题栏
8     QWidget* titleWidget;
9     QLabel* lblIcon;
10    QLabel* lblTitle;
11    QPushButton* btnClose;
12    QPushButton* btnMax;
13    QPushButton* btnMin;
14
15    // 主体
16    QWidget* mainWidget;
17 };
```

然后，修改 `mywidget.cpp` 中的 `initForm()` 函数，如下：

```
1 void MyWidget::initForm()
2 {
3     // 1. 整体规划
4     // 1.1 设置最小宽高
5     this->setMinimumWidth(600);
6     this->setMinimumHeight(400);
7
8     // 1.2 去除标题栏
9     this->setWindowFlags(windowFlags() | Qt::FramelessWindowHint);
10
11    // 1.3 窗体整体采用垂直布局，
12    QVBoxLayout* vl = new QVBoxLayout(this);
13    vl->setSpacing(0);
14    vl->setContentsMargins(0, 0, 0, 0);
15
16    // 1.4 并嵌套一层QWidget
17    QWidget* frameWidget = new QWidget(this);
18    vl->addWidget(frameWidget);
19
20    // 1.5 嵌套的QWidget同样采用垂直布局
21    QVBoxLayout* layout = new QVBoxLayout(frameWidget);
22    layout->setSpacing(0);
23    layout->setContentsMargins(0, 0, 0, 0);
24 }
```

### 3.3 添加标题栏

标题栏整体采用水平布局，从左到右依次为：图标、文本、最小化按钮、最大化按钮、关闭按钮

修改 `mywidget.cpp` 中的 `initForm()` 函数，如下：

```
1 void MyWidget::initForm()
2 {
3     // 1. 整体规划
4     // ...
5
6     // 2. 标题栏
7     // 2.1 设置固定高度、背景色
8     titleWidget = new QWidget(this);
9     titleWidget->setFixedHeight(48);
10    titleWidget->setStyleSheet("background-color:rgb(60,60,60)");
11
12    // 2.2 标题栏采用水平布局
13    QHBoxLayout* titleLayout = new QHBoxLayout(titleWidget); // 标题栏采用水平布
    局
14    titleLayout->setContentsMargins(8, 8, 8, 8); // 这里必须设置一个
    间距
15    titleLayout->setSpacing(0);
16
17    // 2.3 把标题栏添加到整体窗体布局中。之后就可以向titleLayout中添加控件了
18    layout->addWidget(titleWidget);
19
20    // 2.4 添加图标
21    lblIcon = new QLabel(this);
22    lblIcon->setFixedSize(20, 20);
23    lblIcon->setStyleSheet("background-
    image:url(:/res/title_icon_20.svg);border:none");
24    titleLayout->addWidget(lblIcon);
25
26    // 2.5 添加弹簧：在图标和标题之间增加间距
27    QSpacerItem* space = new QSpacerItem(6, 20, QSizePolicy::Fixed,
    QSizePolicy::Fixed);
28    titleLayout->addSpacerItem(space);
29
30    // 2.6 添加标题
31    lblTitle = new QLabel(this);
32    //    lblTitle->setText("UI美化实战课程_演示01_无边框窗口");
33    lblTitle->setText("完善的无边框窗口");
34    #if 0
35    // 默认地标签在水平方向和垂直方向的策略都是 Preferred。
```

```
36 // 因此会自动将右侧的最小化/最大化/关闭按钮顶到右侧，不需要增加弹簧
37 // QSizePolicy(horizontalPolicy = QSizePolicy::Preferred, verticalPolicy =
    QSizePolicy::Preferred)
38 qDebug() << lblTitle->sizePolicy();
39 lblTitle->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Preferred);
40 // QSizePolicy(horizontalPolicy = QSizePolicy::Fixed, verticalPolicy =
    QSizePolicy::Preferred)
41 qDebug() << lblTitle->sizePolicy();
42 #endif
43 lblTitle->setAlignment(Qt::AlignLeft | Qt::AlignVCenter);
44 lblTitle->setStyleSheet(R"(
45     background-color: rgba(255, 255, 255, 0);
46     font:18px "Microsoft YaHei";
47     color:rgb(200,200,200);
48     border:none
49 )");
50 titleLayout->addWidget(lblTitle);
51
52 // 2.7 添加最小化按钮
53 btnMin = new QPushButton(this);
54 btnMin->setCursor(QCursor(Qt::PointingHandCursor));
55 btnMin->setStyleSheet(R"(
56     QPushButton {
57         background-image:url(/res/minimize_20.svg);
58         border:none;
59         background-repeat:none;
60         background-position:center;
61     }
62     QPushButton:hover {
63         background-color:rgb(86, 86, 86);
64     }
65 )");
66 btnMin->setFixedSize(36, 32);
67 titleLayout->addWidget(btnMin);
68
69 // 2.8 添加最大化按钮
70 btnMax = new QPushButton(this);
71 btnMax->setCursor(QCursor(Qt::PointingHandCursor));
72 btnMax->setStyleSheet(R"(
73     QPushButton {
74         background-image:url(/res/maximize_20_normal.svg);
75         border:none;
76         background-repeat:none;
77         background-position:center;
78     }
79     QPushButton:hover {
80         background-color:rgb(86, 86, 86);
```

```

81         }
82     )");
83     btnMax->setFixedSize(36, 32);
84     titleLayout->addWidget(btnMax);
85
86     // 2.9 添加关闭按钮
87     btnClose = new QPushButton(this);
88     btnClose->setCursor(QCursor(Qt::PointingHandCursor));
89     btnClose->setStyleSheet(R"(
90         QPushButton {
91             background-image:url(:/res/close_20.svg);
92             border:none;
93             background-repeat:none;
94             background-position:center;
95         }
96         QPushButton:hover {
97             background-color:rgb(222, 6, 26);
98         }
99     )");
100     btnClose->setFixedSize(36, 32);
101     titleLayout->addWidget(btnClose);
102 }

```

## 3.4 添加主体

主体整体采用水平布局，暂且添加两个按钮

修改 `mywidget.cpp` 中的 `initForm()` 函数，如下：

```

1 void MyWidget::initForm()
2 {
3     // 1. 整体规划
4     // ...
5     // 2. 标题栏
6     // ...
7
8     // 3. 主体
9     // 3.1 设置背景色
10    mainWidget = new QWidget(this);
11    mainWidget->setStyleSheet("background:#303030"); // 设置背景颜色
12
13    // 3.2 主体采用水平布局
14    QHBoxLayout* mainLayout = new QHBoxLayout(mainWidget);
15    mainLayout->setSpacing(10);

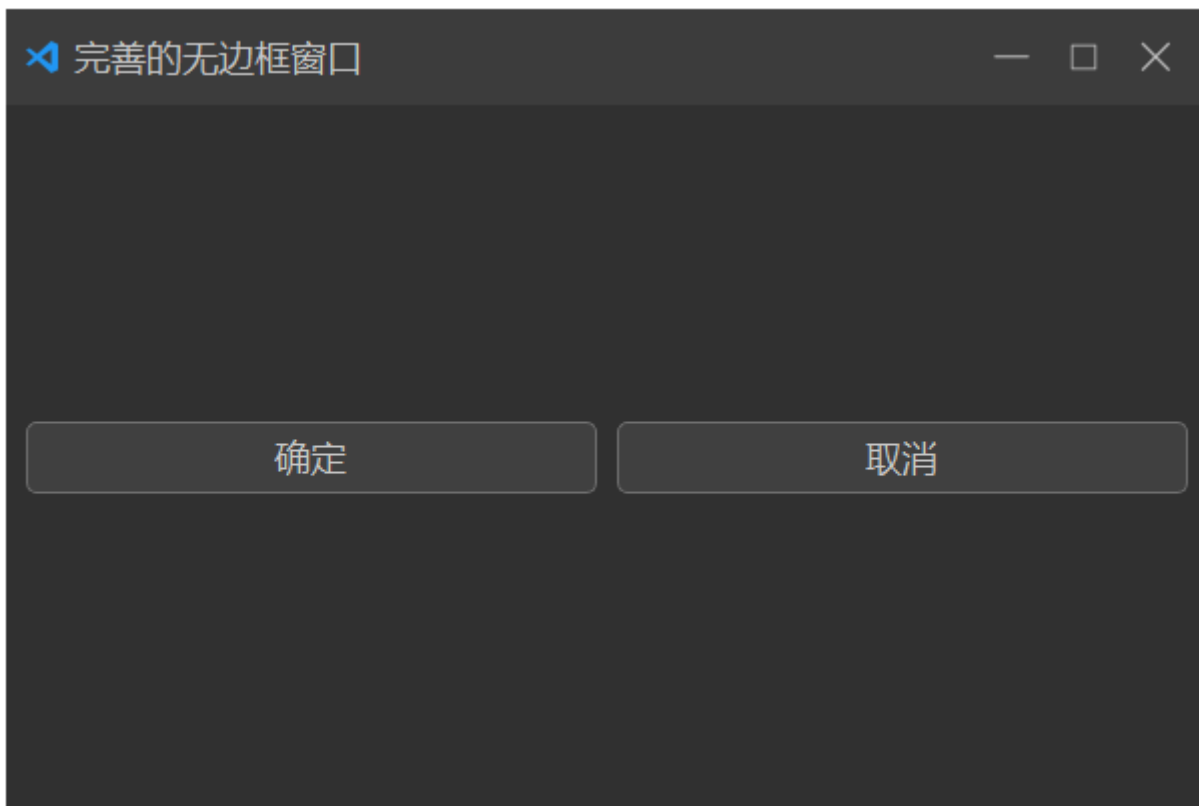
```

```

16     mainLayout->setContentsMargins(10, 10, 10, 10);
17
18     // 3.3 把主体添加到整体窗体布局中。之后就可以向mainLayout中添加控件了
19     layout->addWidget(mainWidget); // 将主体添加到布局中
20
21     // 3.4 添加确定和取消按钮
22     QPushButton* btn1 = new QPushButton("确定");
23     QPushButton* btn2 = new QPushButton("取消");
24
25     mainLayout->addWidget(btn1);
26     mainLayout->addWidget(btn2);
27
28     QString style = R"(
29         QPushButton {
30             background-color: rgb(64, 64, 64);
31             font:18px "Microsoft YaHei";
32             color:rgb(200,200,200);
33             border: 1px solid #707070;
34             border-radius: 5px;
35             padding: 5px;
36         }
37         QPushButton:hover {
38             background-color: rgb(40, 40, 40);
39         }
40         QPushButton:pressed {
41             background-color: rgb(64, 64, 64);
42         }
43     )";
44
45     btn1->setStyleSheet(style);
46     btn2->setStyleSheet(style);
47
48     connect(btn1, &QPushButton::clicked, this, [=] { qDebug() << btn1->text();
49     });
50     connect(btn2, &QPushButton::clicked, this, [=] { qDebug() << btn2->text();
51     });
52 }

```

此时运行，就可以看到比较美观的界面了，如下：



## 4. 实现最大化、最小化、关闭功能

上一节添加了自定义标题栏，但是右上角的最小化、最大化、关闭按钮的功能还没有实现，本节就来实现。

### 4.1 关联信号槽

来到 `mywidget.cpp` 的构造函数中，添加如下代码：

```
1 MyWidget::MyWidget(QWidget* parent) : QWidget(parent)
2 {
3     initForm();
4
5     // 关联信号槽
6     connect(btnClose, &QPushButton::clicked, [this]() { this->close(); });
7
8     connect(btnMax, &QPushButton::clicked, [this]() {
9         // 切换最大化/正常显示
10         this->setWindowState(this->windowState().testFlag(Qt::WindowNoState) ?
11             Qt::WindowMaximized : Qt::WindowNoState);
12     });
13
14     connect(btnMin, &QPushButton::clicked, [this]() {
15         // 注意不要这样写，否则最大化状态下点击最小化按钮，再次恢复时不是最大化状态而是
16         normal状态
```

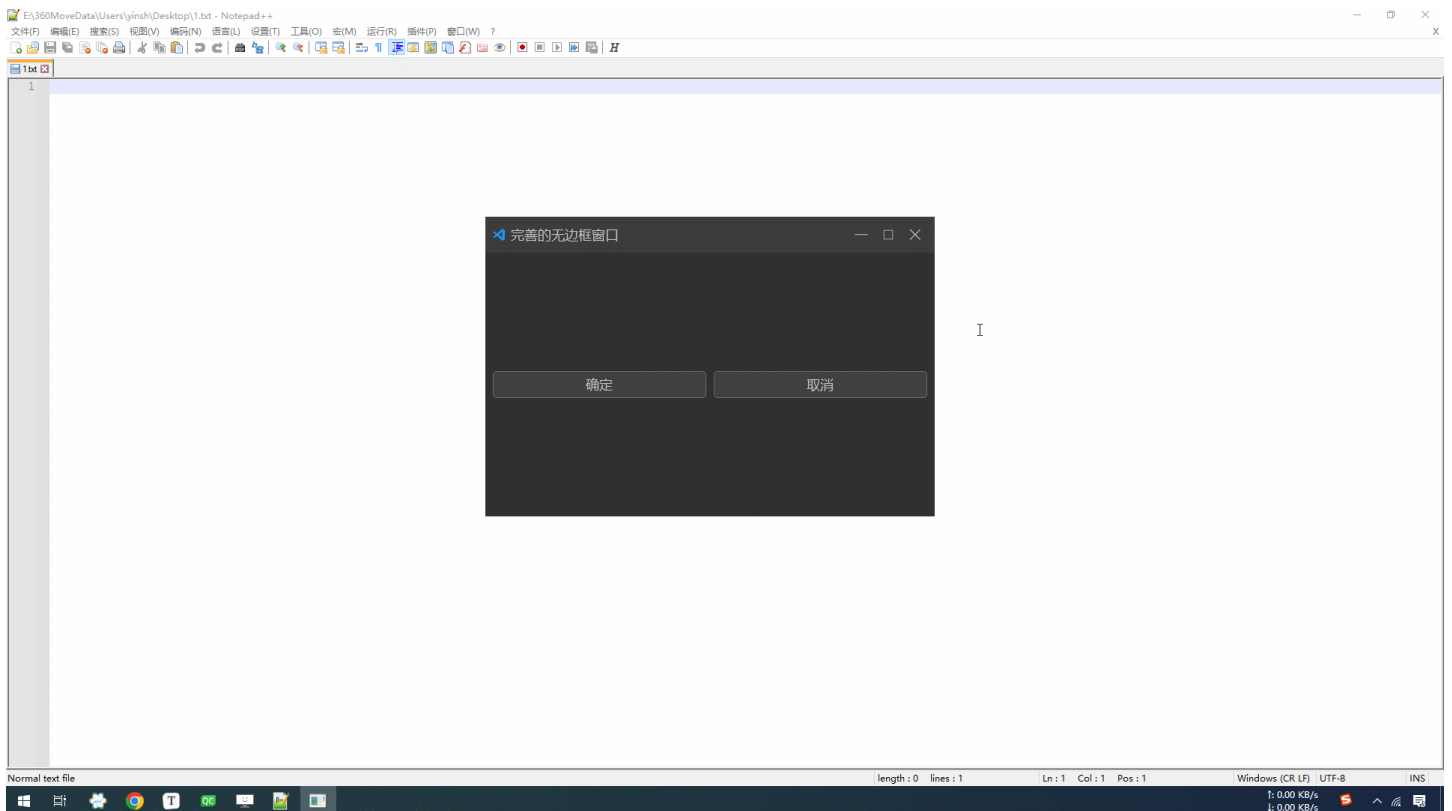


```

15         // this->setWindowState(Qt::WindowMinimized);
16
17         // 使用下面的与或非写法，不要改变原来的状态。
18         this->setWindowState(windowState() & ~Qt::WindowActive |
        Qt::WindowMinimized);
19     });
20 }

```

此时就实现了最小化、最大化、关闭功能，如下：



## 4.2 改变最大化图标

此时还有一个问题：最大化状态下，最大化按钮的图标应该变为“恢复”正常大小的样式。

实现方式有两种：

**方式一：**

点击最大化按钮后，

如果是正常显示，则最大化窗口，并**手动**设置图标；

如果是最大化显示，则恢复窗口，并**手动**设置图标

**方式二：**

点击最大化按钮后，

如果是正常显示，则最大化窗口；

如果是最大化显示，则恢复窗口；

重写 `changeEvent()` 事件，当窗口状态变化时，自动调用该函数，在该函数中**自动**设置图标  
我们这里就采用这种方式。

首先，来到 `mywidget.h` 中声明 `changeEvent()` 函数，如下：

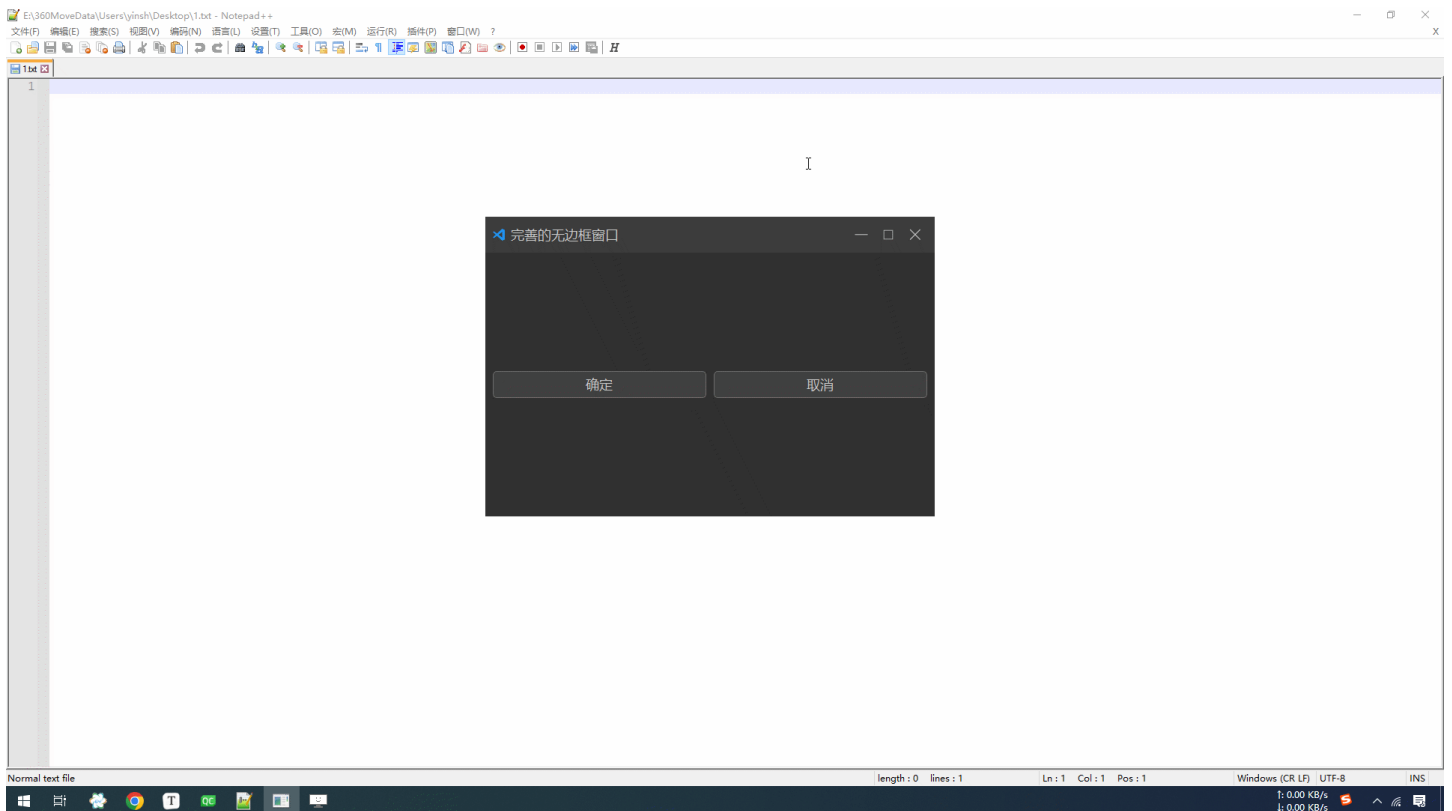
```
1 class MyWidget : public QWidget
2 {
3 protected:
4     void changeEvent(QEvent* event);
5 };
```

然后，来到 `mywidget.cpp` 中实现 `changeEvent()` 函数，如下：

```
1 #include <QEvent>
2
3 void MyWidget::changeEvent(QEvent* event)
4 {
5     if ( event->type() == QEvent::WindowStateChange ) {
6         if ( this->windowState().testFlag(Qt::WindowMaximized) ) {
7             btnMax->setStyleSheet(R"(
8                 QPushButton {
9                     background-image:url(/res/maximize_20_max.svg);
10                    border:none;
11                    background-repeat:none;
12                    background-position:center;
13                }
14                QPushButton:hover {
15                    background-color:rgb(86, 86, 86);
16                }
17            )");
18        } else if ( this->windowState().testFlag(Qt::WindowNoState) ) {
19            btnMax->setStyleSheet(R"(
20                QPushButton {
21                    background-image:url(/res/maximize_20_normal.svg);
22                    border:none;
23                    background-repeat:none;
24                    background-position:center;
25                }
26            )");
27        }
28    }
29 }
```

```
26         QPushButton:hover {
27             background-color:rgb(86, 86, 86);
28         }
29     }");
30 }
31 }
32 }
```

此时运行，在窗口最大化和恢复时，图标就可以变化了，如下：



## 5. 拖动窗口

上一节实现了最小化、最大化、关闭功能，本节来实现窗口移动功能。

由于拖动窗口、缩放窗口是无边框窗口通用的功能，因此这里我们定义一个基类 `FramelessWindow`，把这些功能在基类中实现。

### 5.1 定义基类 `FramelessWindow`

在左侧项目上右键，选择【添加新文件】，选择【C++ Class】，类信息如下：

Define Class

Class name:  类名

Base class:  基类

☐ Include QObject

☒ Include QWidget

☐ Include QMainWindow

☐ Include QDeclarativeItem - Qt Quick 1

☐ Include QQuickItem - Qt Quick 2

☐ Include QSharedData

☒ Add Q\_OBJECT

☐ Add QML\_ELEMENT

Header file:  头文件和源文件

Source file:

路径:  浏览...

此时项目目录如下：

FramelessDemo

- FramelessDemo.pro
- 头文件
  - FramelessWindow.h
  - mywidget.h
- 源文件
  - FramelessWindow.cpp
  - main.cpp
  - mywidget.cpp
- 资源
  - FramelessDemo.qrc

```
1  #ifndef FRAMELESSWINDOW_H
2  #define FRAMELESSWINDOW_H
3
4  #include <QWidget>
5
6  class FramelessWindow : public QWidget
7  {
8      Q_OBJECT
9  public:
10     explicit FramelessWindow(QWidget* parent = nullptr);
11
12     signals:
13 };
14
15 #endif // FRAMELESSWINDOW_H
```

## 5.2 当前窗口继承 FramelessWindow

自己创建的窗口类 `MyWidget`，默认继承自 `QWidget`，接下来让它继承自刚创建的 `FramelessWindow`

首先，来到 `mywidget.h`，修改构造函数：

```
1  #include "FramelessWindow.h"
2
3  class MyWidget : public FramelessWindow
4  {
5      // ...
```

```
6 };
```

然后，来到 `mywidget.cpp`，修改构造函数：

```
1 MyWidget::MyWidget(QWidget* parent) : FramelessWindow(parent)
2 {
3     // ...
4 }
```

此时，直接运行效果和之前完全一致，这里就不贴图了！

最后，由于设置无边框是每个窗口都需要的功能，因此：

把设置无边框的代码从 `MyWidget` 类注释掉：

```
1 void MyWidget::initForm()
2 {
3     // 1.2 去除标题栏
4     // this->setWindowFlags(windowFlags() | Qt::FramelessWindowHint);
5 }
```

把它移动到 `FramelessWindow` 类中：

```
1 FramelessWindow::FramelessWindow(QWidget* parent) : QWidget{parent}
2 {
3     // 去除标题栏
4     this->setWindowFlags(windowFlags() | Qt::FramelessWindowHint);
5 }
```

此时，运行结果和之前也是一致的！

## 5.3 重写鼠标按下、移动、释放事件

除了重写这3个鼠标事件，还需要定义几个成员变量。

首先，来到 `FramelessWindow.h`，定义成员变量和成员函数：

```

1 class FramelessWindow : public QWidget
2 {
3 protected:
4     void setTitlebarWidgets(QVector<QWidget*> widgets);
5
6     void mousePressEvent(QMouseEvent* event);
7     void mouseMoveEvent(QMouseEvent* event);
8     void mouseReleaseEvent(QMouseEvent* event);
9
10 private:
11     bool leftPressed;           // 左键是否按下
12     bool leftPressedInTitle;    // 左键是否在标题栏按下
13     QVector<QWidget*> titlebarWidgets; // 记录标题栏中的控件
14
15     QPoint pressPos; // 鼠标点击的位置
16     QPoint wndPos;   // 当前窗体的位置，也就是窗口左上角的坐标
17 };

```

接下来实现以上4个函数

## (1) setTitleWidgets() 函数

将标题栏中的控件设置进来，就可以在 `mousePressEvent()` 中判断鼠标是否在标题栏按下了

```

1 #include <QMouseEvent>
2 #include <QApplication>
3
4 void FramelessWindow::setTitlebarWidgets(QVector<QWidget*> widgets)
5 {
6     titlebarWidgets = widgets;
7 }

```

## (2) mousePressEvent() 函数

当鼠标按下时，主要干了两件事：

- 记录鼠标按下的位置、当前窗口的位置
- 判断鼠标按下的位置是否在标题栏中

```

1 void FramelessWindow::mousePressEvent(QMouseEvent* event)

```

```

2 {
3     // 如果不是左键，直接返回
4     if ( event->button() != Qt::LeftButton ) {
5         return;
6     }
7
8     leftPressed = true;
9
10    wndPos = this->pos();           // 记录当前窗体的位置，也就是窗体左上角的坐标
11    pressPos = event->globalPos(); // 记录鼠标按下的位置
12
13    // 判断左键按下的位置是否在标题栏中
14    QWidget* pressedWidget = QApplication::widgetAt(event->globalPos());
15    // qDebug() << "pressedWidget: " << pressedWidget << ", " << event-
    >globalPos();
16    if ( pressedWidget ) {
17        foreach (QWidget* widget, titlebarWidgets) {
18            if ( pressedWidget == widget ) {
19                leftPressedInTitle = true;
20                break;
21            }
22        }
23    }
24 }

```

### (3) `mouseMoveEvent()` 函数

左键在标题栏中按下并移动时，修改窗口位置，实现窗口跟随鼠标移动的效果。

```

1 void FramelessWindow::mouseMoveEvent(QMouseEvent* event)
2 {
3     QPoint globalPos = event->globalPos();
4
5     // 1. 左键未按下
6     if ( !leftPressed ) {
7         return;
8     }
9
10    // 2. 左键按下，并且是在标题栏中按下
11    if ( leftPressedInTitle ) {
12        this->move(wndPos + (event->globalPos() - pressPos));
13    }
14 }

```

#### (4) `mouseReleaseEvent()` 函数

在鼠标释放时，恢复标志位

```
1 void FramelessWindow::mouseReleaseEvent(QMouseEvent* event)
2 {
3     leftPressed = false;
4     leftPressedInTitle = false;
5 }
```

当然，记得在构造中，给这两个标志位初始化，如下：

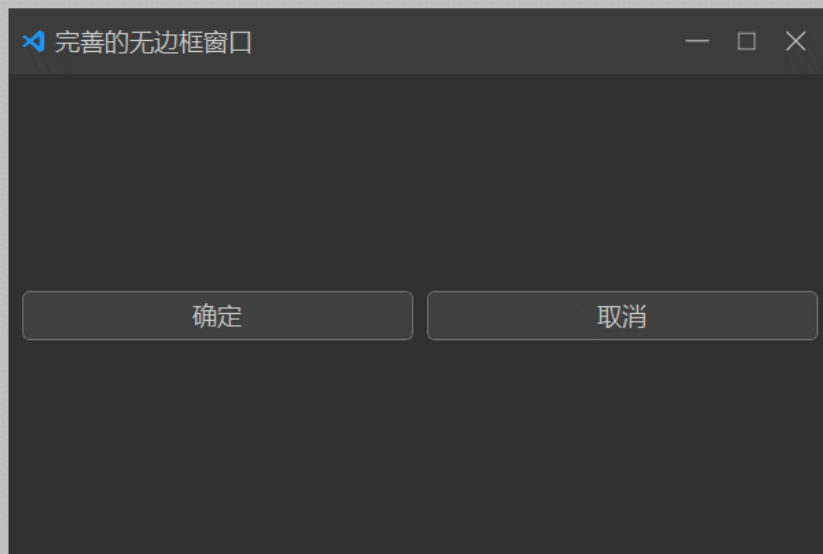
```
1 FramelessWindow::FramelessWindow(QWidget* parent) : QWidget{parent}
2 {
3     leftPressed = false;
4     leftPressedInTitle = false;
5 }
```

实现以上4个函数后，需要在 `mywidget.cpp` 构造中，调用 `setTitleWidgets()` 函数，如下：

```
1 MyWidget::MyWidget(QWidget* parent) : FramelessWindow(parent)
2 {
3     initForm();
4     setTitlebarWidgets({titleLabel, lblIcon, lblTitle});
5 }
```

此时，运行效果如下：





## 6. 边界位置改变光标形状

完成了拖动标题栏来移动窗口的功能后，本节开始实现点击窗口边界来缩放窗口的功能，这其实包含两个步骤：

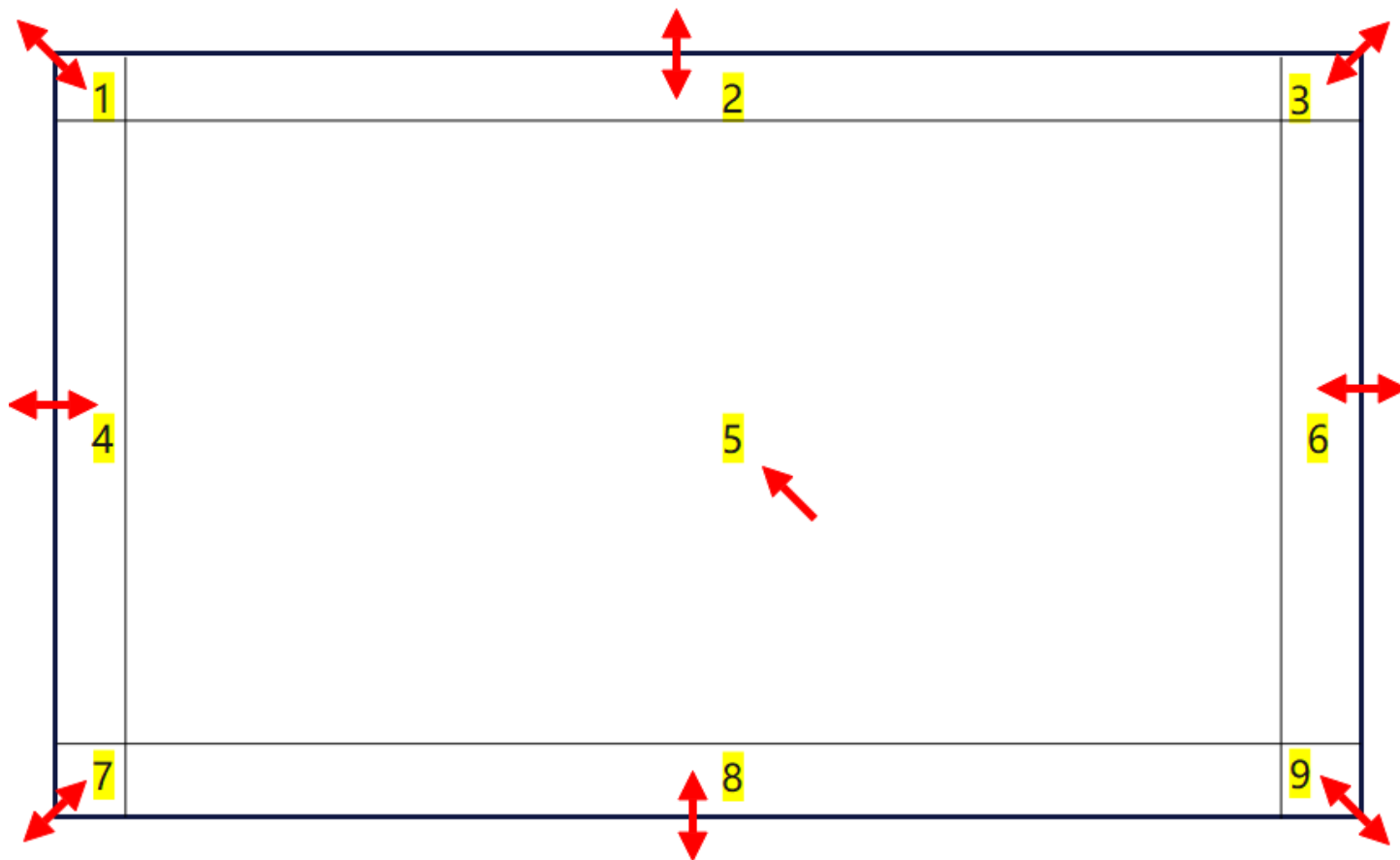
- (1) 鼠标移动到窗口边界，光标变成缩放样式
- (2) 鼠标左键按下，拖动来缩放窗口

本节先来实现第一步：鼠标移动到窗口边界，光标变成缩放样式

### 6.1 定义区域枚举

这里将整个窗口分成 9 个区域：

- 当鼠标位于 12346789 区域时，鼠标形状变成缩放样式
- 当鼠标位于 5 区域时，鼠标恢复正常形状



来到 `FramelessWindow.h`，定义一个枚举类型 `Location` 来表示这 9 个区域：

```
1 // 针对9个区域，定义了一个枚举变量；
2 enum Location {
3     TOP,
4     BOTTOM,
5     LEFT,
6     RIGHT,
7     TOP_LEFT,
8     TOP_RIGHT,
9     BOTTOM_LEFT,
10    BOTTOM_RIGHT,
11    CENTER
12 };
```

## 6.2 设置鼠标追踪

窗口边界改变光标形状，前提是要设置鼠标追踪

首先，来到 `FramelessWindow.h`，声明以下两个函数：

```
1 class FramelessWindow : public QWidget
```

```

2 {
3 protected:
4     // 用于设置鼠标追踪
5     void setAllWidgetMouseTracking(QWidget* widget);
6     bool eventFilter(QObject* target, QEvent* event);
7 };

```

然后，来到 `FramelessWindow.cpp`，实现这两个函数：

```

1 void FramelessWindow::setAllWidgetMouseTracking(QWidget* widget)
2 {
3     widget->setMouseTracking(true);
4
5     QObjectList list = widget->children(); // typedef QList<QObject*>
        QObjectList;
6     foreach (QObject* obj, list) {
7         // qDebug() << obj->metaObject()->className();
8         if ( obj->metaObject()->className() == QStringLiteral("QWidget") ) {
9             QWidget* w = (QWidget*)obj;
10            w->setMouseTracking(true);
11            setAllWidgetMouseTracking(w);
12        }
13    }
14 }
15
16 /* 当前窗口绘制时，给窗口中的所有控件设置鼠标追踪。
17  * 这样鼠标在移动到边界时，可以追踪得到，从而设置光标为缩放形状
18  */
19 bool FramelessWindow::eventFilter(QObject* target, QEvent* event)
20 {
21     if ( event->type() == QEvent::Paint ) {
22         static bool init = false;
23         if ( !init ) {
24             init = true;
25             setAllWidgetMouseTracking(this);
26         }
27     }
28
29     // 查看源码可知，父类的实现就是 return false，表示让事件接着传递，也就是传递给对应的
        控件
30     return QWidget::eventFilter(target, event);
31 }

```

最后，需要在 `FramelessWindow.cpp` 的构造中安装事件过滤器，这样才能自动调用 `eventFilter()` 函数：

```
1 FramelessWindow::FramelessWindow(QWidget* parent) : QWidget{parent}
2 {
3     // ...
4     this->installEventFilter(this);
5 }
```

## 6.3 改变光标形状

完成以上两步，就可以真正实现改变光标形状的功能了。

首先，来到 `FramelessWindow.h`，声明2个成员变量、1个成员函数：

```
1 class FramelessWindow : public QWidget
2 {
3 protected:
4     // 用于设置鼠标追踪
5     void setCursorShape(const QPoint& point);
6
7 private:
8     // 拖动缩放
9     int padding;
10    Location hoverPos;
11 };
```

然后，来到 `FramelessWindow.cpp`，修改 `mouseMoveEvent()` 函数，如下：

```
1 void FramelessWindow::mouseMoveEvent(QMouseEvent* event)
2 {
3     QPoint globalPos = event->globalPos();
4
5     // 1. 左键未按下
6     if ( !leftPressed ) {
7         // 窗口不是最大化状态，则光标移动到边界时，要变成缩放的形状（窗口处于最大化状态
           时，就无须改变光标形状了）
8     }
```

```

8         if ( this->windowState().testFlag(Qt::WindowNoState) ) {
9             setCursorShape(globalPos);
10        }
11        return;
12    }
13 }

```

这样，当鼠标在窗口中移动时，会调用 `setCursorShape()` 函数来设置光标的形状。

然后，真正实现 `setCursorShape()` 函数，如下：

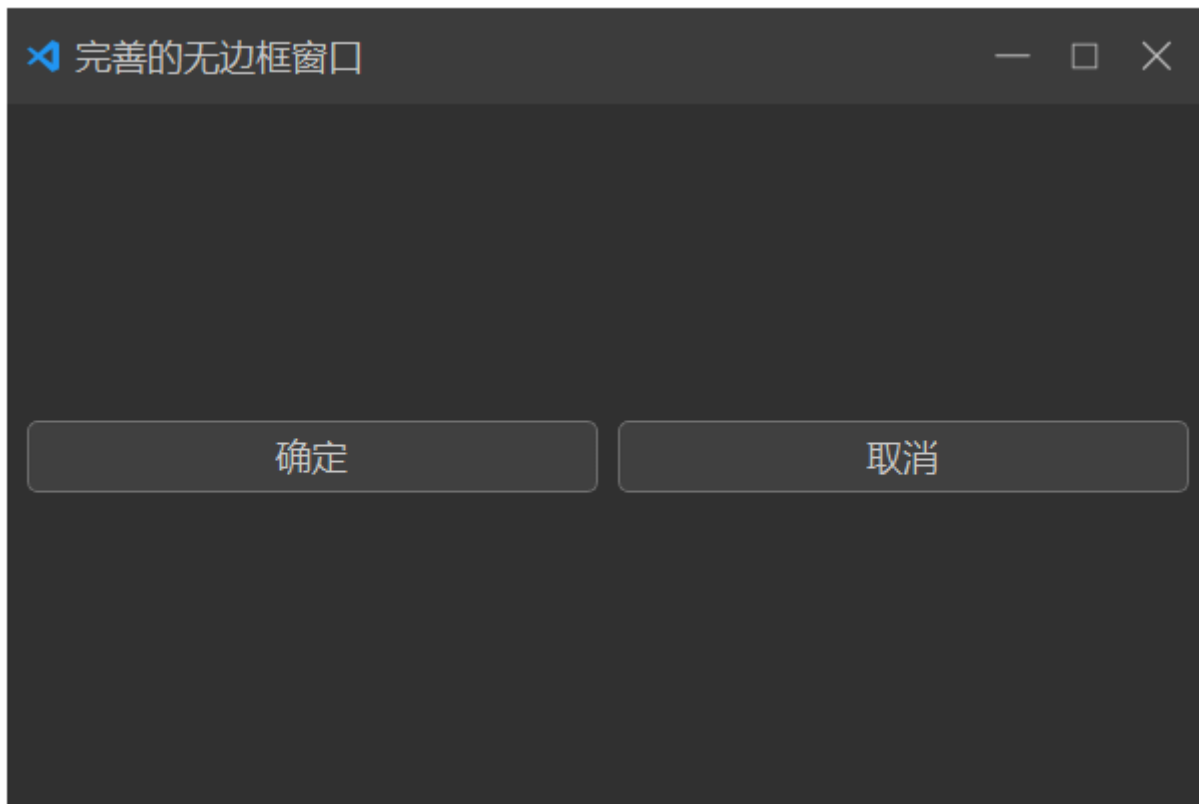
```

1 void FramelessWindow::setCursorShape(const QPoint& point)
2 {
3     QRect rect = this->rect();
4     #if 0
5         // topLeft= QPoint(0,0) , bottomRight= QPoint(599,399)
6         qDebug() << " topLeft=" << rect.topLeft() << ", bottomRight=" <<
            rect.bottomRight();
7     #endif
8     QPoint topLeft = mapToGlobal(rect.topLeft());
9     QPoint bottomRight = mapToGlobal(rect.bottomRight());
10
11     int x = point.x();
12     int y = point.y();
13
14     if ( x >= topLeft.x() && x <= topLeft.x() + padding && y >= topLeft.y() &&
        y <= topLeft.y() + padding ) {
15         // 左上角
16         hoverPos = TOP_LEFT;
17         this->setCursor(QCursor(Qt::SizeFDiagCursor));
18     } else if ( x <= bottomRight.x() && x >= bottomRight.x() - padding && y <=
        bottomRight.y() && y >= bottomRight.y() - padding ) {
19         // 右下角
20         hoverPos = BOTTOM_RIGHT;
21         this->setCursor(QCursor(Qt::SizeFDiagCursor));
22     } else if ( x >= topLeft.x() && x <= topLeft.x() + padding && y <=
        bottomRight.y() && y >= bottomRight.y() - padding ) {
23         // 左下角
24         hoverPos = BOTTOM_LEFT;
25         this->setCursor(QCursor(Qt::SizeBDiagCursor));
26     } else if ( x <= bottomRight.x() && x >= bottomRight.x() - padding && y >=
        topLeft.y() && y <= topLeft.y() + padding ) {
27         // 右上角
28         hoverPos = TOP_RIGHT;

```

```
29     this->setCursor(QCursor(Qt::SizeBDiagCursor));
30 } else if ( x >= topLeft.x() && x <= topLeft.x() + padding ) {
31     // 左边
32     hoverPos = LEFT;
33     this->setCursor(QCursor(Qt::SizeHorCursor));
34 } else if ( x <= bottomRight.x() && x >= bottomRight.x() - padding ) {
35     // 右边
36     hoverPos = RIGHT;
37     this->setCursor(QCursor(Qt::SizeHorCursor));
38 } else if ( y >= topLeft.y() && y <= topLeft.y() + padding ) {
39     // 上边
40     hoverPos = TOP;
41     this->setCursor(QCursor(Qt::SizeVerCursor));
42 } else if ( y <= bottomRight.y() && y >= bottomRight.y() - padding ) {
43     // 下边
44     hoverPos = BOTTOM;
45     this->setCursor(QCursor(Qt::SizeVerCursor));
46 } else {
47     // 中间
48     hoverPos = CENTER;
49     this->setCursor(QCursor(Qt::ArrowCursor));
50 }
51 }
```

经过以上步骤，就实现了鼠标移动到边界时，改变光标形状的功能，如下：



## 7. 拖动鼠标缩放窗口

点击窗口边界来缩放窗口的功能，其实包含两个步骤：

- (1) 鼠标移动到窗口边界，光标变成缩放样式
- (2) 鼠标左键按下，拖动来缩放窗口

上一节已经实现了第1步，本节来实现第2步：鼠标移动到窗口边界，左键按下缩放窗口

只需修改 `FramelessWindow.cpp` 中的 `mouseMoveEvent()` 函数，如下：

```
1 void FramelessWindow::mouseMoveEvent(QMouseEvent* event)
2 {
3     QPoint globalPos = event->globalPos();
4
5     // 1. 左键未按下
6     // ...
7
8     // 2. 左键按下时
9     if ( hoverPos != CENTER ) {
10         // 2.1 在边界处按下
11         QRect rect = this->rect();
12         QPoint topLeft = mapToGlobal(rect.topLeft());
```

```

13     QPoint bottomRight = mapToGlobal(rect.bottomRight());
14
15     QRect rMove(topLeft, bottomRight);
16
17     switch ( hoverPos ) {
18         case TOP:
19             // 如果不加if判断, 则窗口高度达到最小高度后, 会被鼠标 "向下推走"
20             if ( bottomRight.y() - globalPos.y() > this->minimumHeight() )
11 {
21                 rMove.setY(globalPos.y());
22             }
23             break;
24         case BOTTOM:
25             rMove.setHeight(globalPos.y() - topLeft.y());
26             break;
27         case LEFT:
28             // 如果不加if判断, 则窗口高度达到最小宽度后, 会被鼠标 "向右推走"
29             if ( bottomRight.x() - globalPos.x() > this->minimumWidth() ) {
30                 rMove.setX(globalPos.x());
31             }
32             break;
33         case RIGHT:
34             rMove.setWidth(globalPos.x() - topLeft.x());
35             break;
36         case TOP_LEFT:
37             if ( bottomRight.y() - globalPos.y() > this->minimumHeight() )
12 {
38                 rMove.setY(globalPos.y());
39             }
40             if ( bottomRight.x() - globalPos.x() > this->minimumWidth() ) {
41                 rMove.setX(globalPos.x());
42             }
43             break;
44         case TOP_RIGHT:
45             if ( bottomRight.y() - globalPos.y() > this->minimumHeight() )
13 {
46                 rMove.setY(globalPos.y());
47             }
48             rMove.setWidth(globalPos.x() - topLeft.x());
49             break;
50         case BOTTOM_LEFT:
51             rMove.setHeight(globalPos.y() - topLeft.y());
52             if ( bottomRight.x() - globalPos.x() > this->minimumWidth() ) {
53                 rMove.setX(globalPos.x());
54             }
55             break;
56         case BOTTOM_RIGHT:

```



```

57         rMove.setHeight(globalPos.y() - topLeft.y());
58         rMove.setWidth(globalPos.x() - topLeft.x());
59         break;
60     default:
61         break;
62     }
63     this->setGeometry(rMove);
64 } else {
65     // 2.2 在标题栏内按下
66     if ( leftPressedInTitle ) {
67         this->move(wndPos + (event->globalPos() - pressPos));
68     }
69 }
70 }

```

代码说明：

- (1) 鼠标在边界按下则缩放窗口，在标题栏按下则移动窗口
- (2) 缩放窗口的原理就是：通过调用 `setGeometry()` 来设置窗口的几何参数
- (3) 向下和向右移动光标时，要加 `if` 判断，否则窗口会被“向下推走”和“向右推走”

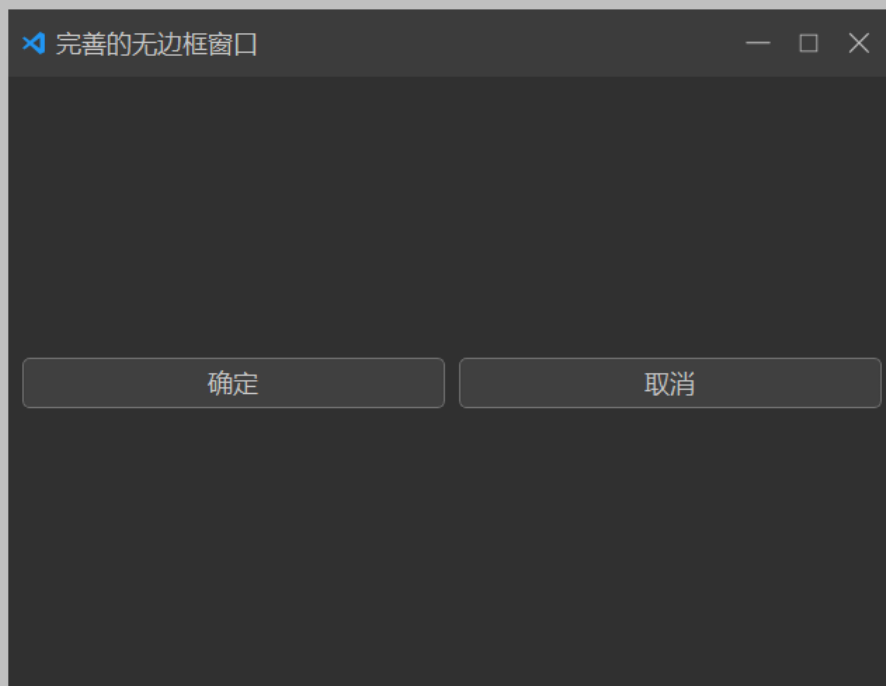
**?** 问：向上和向左移动光标，为啥不加 `if` 判断呢？

答：

因为向下和向右是调用 `setX()` 和 `setY()`，会导致位置变化；

而向上和向左是调用 `setHeight()` 和 `setWidth()`，而窗口本身是有最小宽高的，当小于等于最小宽高时，就会保持最小宽高，而不会导致窗口位置的变化

以上，就可以实现在边界处按下鼠标，来缩放窗口了，效果如下：



## 7.1 双击标题栏最大化窗口

常见的软件，在双击标题栏时：

- 如果当前窗口正常显示，则可以最大化窗口
- 如果当前窗口最大化显示，则可以恢复原大小

本节课来实现这一功能。

首先，来到 `FramelessWindow.h`，声明鼠标双击事件 `mouseDoubleClickEvent()`，如下：

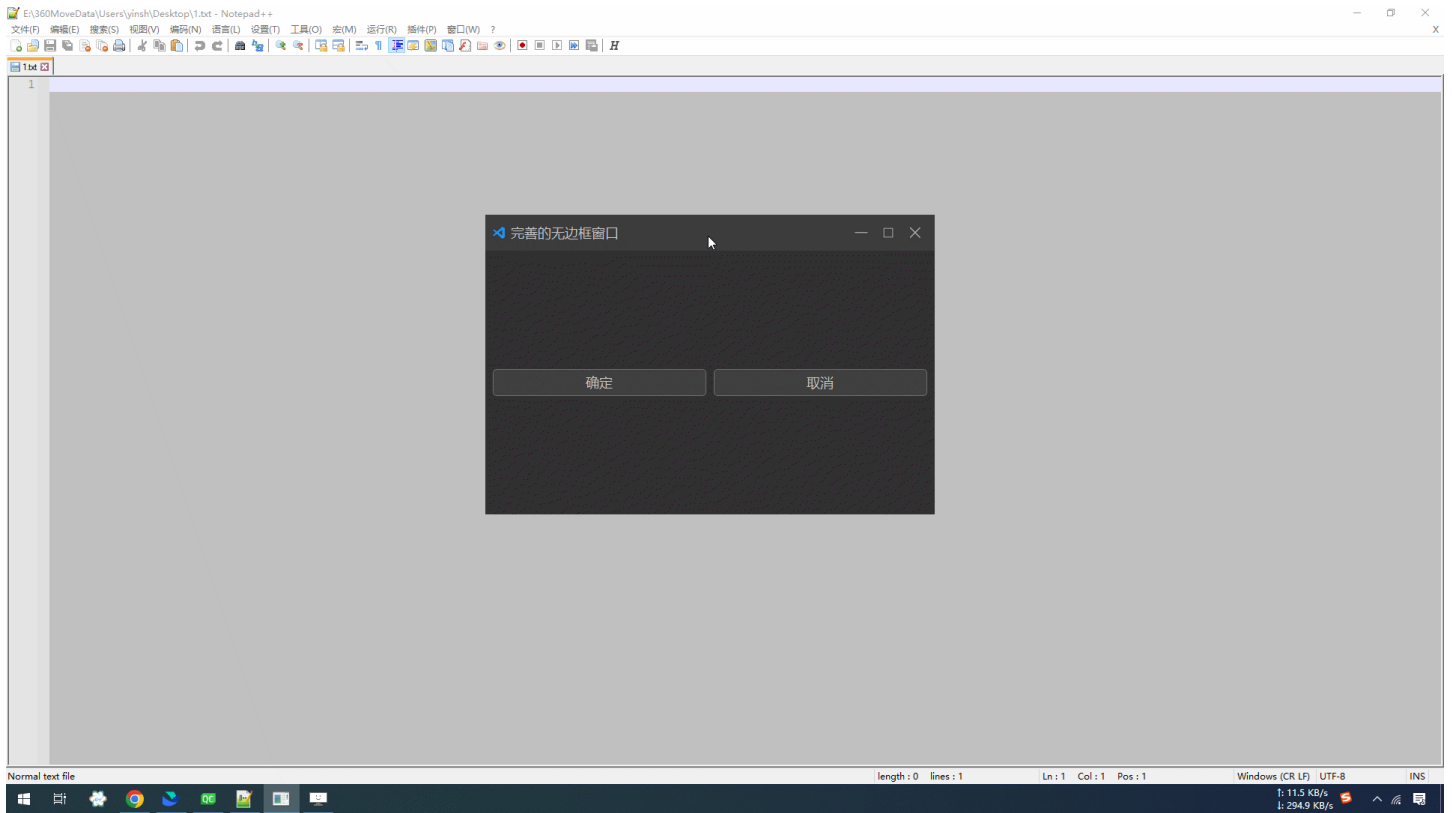
```
1 class FramelessWindow : public QWidget
2 {
3 protected:
4     // 双击标题栏放大
5     void mouseDoubleClickEvent(QMouseEvent* event);
6 };
```

然后，来到 `FramelessWindow.cpp`，实现 `mouseDoubleClickEvent()`，如下：

```
1 void FramelessWindow::mouseDoubleClickEvent(QMouseEvent* event)
2 {
3     QWidget* clickedWidget = QApplication::widgetAt(event->globalPos());
4     //      qDebug() << "clickedWidget: " << clickedWidget << ", " << event-
        >globalPos();
5     if ( clickedWidget ) {
6         bool inTitle = false;
7         foreach (QWidget* widget, titlebarWidgets) {
8             if ( clickedWidget == widget ) {
9                 inTitle = true;
10                break;
11            }
12        }
13
14        if ( inTitle ) {
15            this->setWindowState(isMaximized() ? Qt::WindowNoState :
                Qt::WindowMaximized);
16        }
17    }
18 }
```

说明：调用 `QApplication` 类的 `widgetAt()` 方法，可以获取点击位置处的控件。

以上，就可以实现双击标题栏来最大化和恢复窗口了，效果如下：



## 7.2 窗口最大化时，拖动恢复大小

上一节，在窗口最大化时，拖动窗口仍然是可以移动的

然而，通常窗口最大化显示时，拖动标题栏，不是移动窗口，而是恢复窗口大小

本节课来实现这一功能。

### 7.2.1 修改代码

首先，来到 `FramelessWindow.cpp`，修改 `mouseMoveEvent()`，如下：

```
1 void FramelessWindow::mouseMoveEvent(QMouseEvent* event)
2 {
3     // ...
4
5     // 2. 左键按下时
6     if ( hoverPos != CENTER ) {
7         // 2.1 在边界处按下
8         // ...
9     } else {
10        // 2.2 在标题栏内按下
11        if ( leftPressedInTitle ) {
12            if ( this->isMaximized() ) {
13                // 窗口最大化时鼠标拖动标题栏需要完成两个操作：
14                // a. 窗口恢复
```

```

15 // b. 鼠标相对窗口的位置不变
16 // 相对位置不变指的是：鼠标点击拖动窗口1/4处进行拖动，复原时鼠标依然
   位于窗口1/4处
17 // 达到此效果，仅需更改窗口的位置即可
18
19 // 计算全屏时，鼠标在 x 轴上，相对于屏幕宽度的百分比
20 float width_ratio = float(event->globalPos().x()) / float(this-
   >width());
21 qDebug() << "ratio=" << width_ratio;
22 qDebug() << "before width = " << this->width(); // 1920
23
24 // a. 窗口恢复
25 this->setWindowState(Qt::WindowNoState);
26 qDebug() << "after width = " << this->width(); // 600
27
28 // b. 鼠标相对窗口的相对位置不变
29 // 和双击时一样，默认会回到最大化之前的位置，所以要修改窗口的位置
30 int offset = this->width() * width_ratio; // 当前窗口相对于鼠标位
   置的偏移
31
32 wndPos.setX(event->globalPos().x() - offset);
33 wndPos.setY(0);
34 } else {
35     this->move(wndPos + (event->globalPos() - pressPos));
36 }
37 }
38 }
39 }

```

然后，来到 `FramelessWidow.cpp` 中，修改构造，添加一行代码：

```

1 FramelessWindow::FramelessWindow(QWidget* parent) : QWidget{parent}
2 {
3     // 去除标题栏
4     this->setWindowFlags(windowFlags() | Qt::FramelessWindowHint);
5     setAttribute(Qt::WA_TranslucentBackground); // 添加这一行代码
6 }

```

只有添加了这一行，才可以正确显示窗口恢复前后的宽度，如下：

```

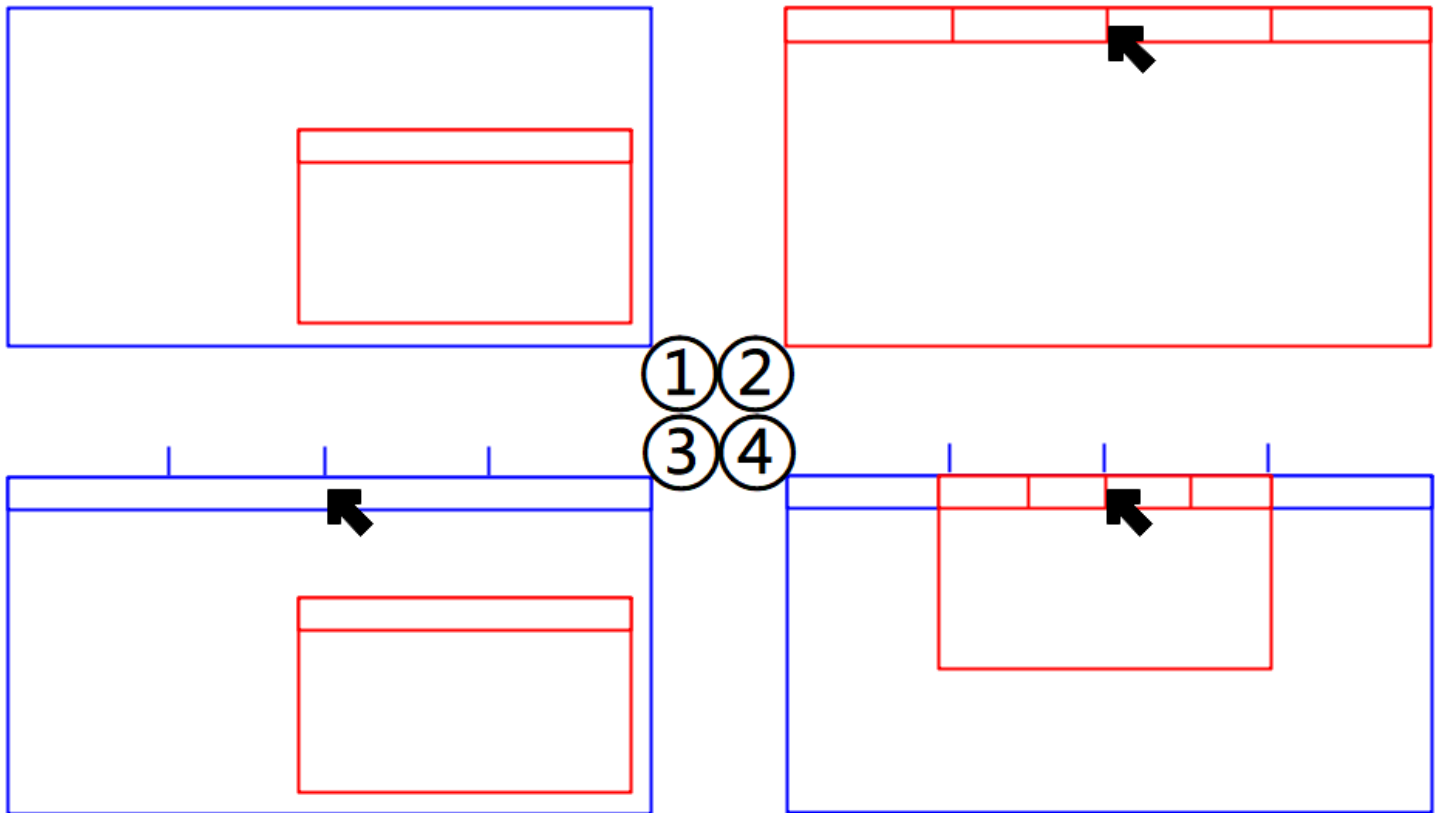
1 qDebug() << "before width = " << this->width(); // 1920
2 this->setWindowState(Qt::WindowNoState);

```

```
3 qDebug() << "after width = " << this->width(); // 600, 否则这里还是 1920
```

## 7.2.2 画图说明

以上代码注释很详细，以下画图说明：



### 📌 窗口状态：

- ① 初始状态，窗口正常显示（非最大化显示）
- ② 窗口最大化状态，此时点击标题栏的1/2处
- ③ 窗口恢复到最大化之前的位置和大小
- ④ 设置窗口位置，让鼠标相对窗口的位置不变

## 8. 支持标题栏之外拖动窗口

有些软件，不仅可以通过拖动标题栏来移动窗口，还可以通过拖动窗口的主体来移动窗口

这里定义一个函数 `setOnlyMoveByTitlebar(bool b)`，它用来决定是否只允许拖动标题栏来移动窗口

本节就来实现这一功能

## 8.1 功能实现

首先，来到 `FramelessWindow.h`，声明 `setOnlyMoveByTitlebar()` 函数以及成员变量，如下：

```
1 class FramelessWindow : public QWidget
2 {
3     protected:
4         void setOnlyMoveByTitlebar(bool b);
5
6     private:
7         bool onlyMoveByTitlebar;
8 };
```

然后，来到 `FramelessWindow.cpp`，实现 `setOnlyMoveByTitlebar()` 函数，如下：

```
1 void FramelessWindow::setOnlyMoveByTitlebar(bool b)
2 {
3     onlyMoveByTitlebar = b;
4 }
```

并在构造中将 `onlyMoveByTitlebar` 初始化为 `true`，默认只允许拖动标题栏，来移动窗口，如下：

```
1 FramelessWindow::FramelessWindow(QWidget* parent) : QWidget{parent}
2 {
3     // 默认只允许拖动标题栏，来移动窗口
4     onlyMoveByTitlebar = true;
5 }
```

然后，来到 `FramelessWindow.cpp`，修改 `mouseMoveEvent()` 函数，如下：

```
1 void FramelessWindow::mouseMoveEvent(QMouseEvent* event)
2 {
3     QPoint globalPos = event->globalPos();
4
5     // 1. 左键未按下
```

```

6      // ...
7
8      // 2. 左键按下时
9      if ( hoverPos != CENTER ) {
10         // 2.1 在边界处按下
11         // ...
12     } else {
13         // 2.2 在非边界处按下
14         if ( leftPressedInTitle ) {
15             // 2.2.1 在标题栏内按下
16             // ...
17         } else {
18             // 2.2.2 在主体内按下
19             if ( !onlyMoveByTitlebar && !this->isMaximized() ) {
20                 this->move(wndPos + (event->globalPos() - pressPos));
21             }
22         }
23     }
24 }

```

最后，来到 `mywidget.cpp`，允许拖动标题栏之外来移动窗口，如下：

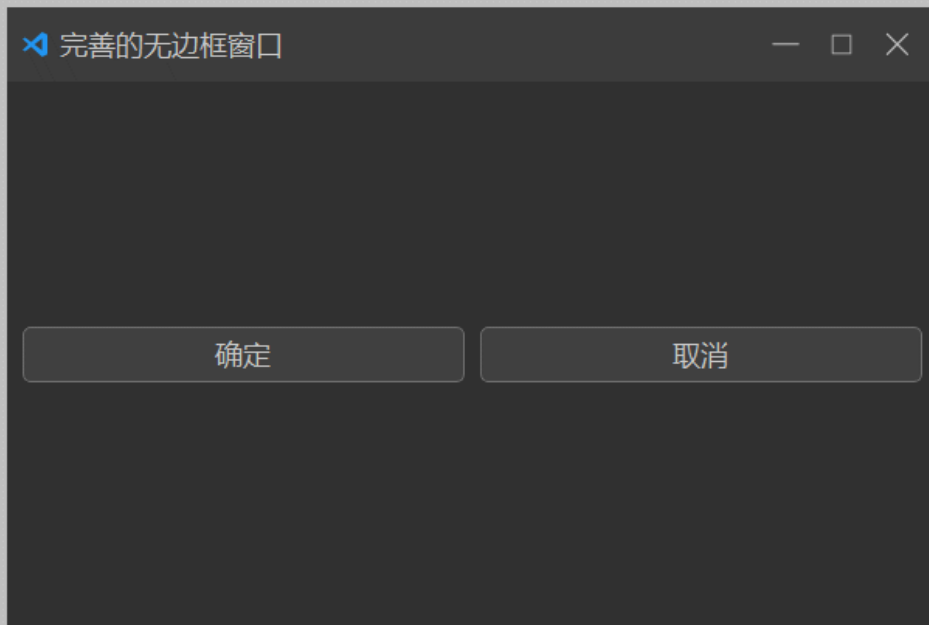
```

1 MyWidget::MyWidget(QWidget* parent) : FramelessWindow(parent)
2 {
3     // ...
4     setOnlyMoveByTitlebar(false); // 默认只允许拖动标题栏实现移动窗口
5 }

```

此时，运行程序，既可以拖动标题栏，也可以拖动主体来实现窗口的移动，如下：





## 8.2 如何使用

如果你的需求是只能通过标题栏移动窗口，那可以什么都不做，因为默认就是只允许标题栏移动窗口。当然也可以调用 `setOnlyMoveByTitlebar()` 函数，如下：

```
1 setOnlyMoveByTitlebar(true);
```

如果允许标题栏之外移动窗口，则必须显式地调用 `setOnlyMoveByTitlebar()`，如下：

```
1 setOnlyMoveByTitlebar(false);
```

## 9. 支持窗口禁止缩放

有些窗口是不允许缩放的，这里定义一个函数 `setResizable(bool b)`，它用来决定是否允许窗口缩放。

本节就来实现这一功能。

## 9.1 功能实现

首先，来到 `FramelessWindow.h`，声明 `setResizable()` 函数以及成员变量，如下：

```
1 class FramelessWindow : public QWidget
2 {
3 protected:
4     void setResizable(bool b);
5
6 private:
7     // 窗口可缩放
8     bool canResize;
9 };
```

然后，来到 `FramelessWindow.cpp`，实现 `setResizable()` 函数，如下：

```
1 void FramelessWindow::setResizable(bool b)
2 {
3     canResize = b;
4 }
```

并在构造中将 `canResize` 初始化为 `true`，默认允许窗口缩放，如下：

```
1 FramelessWindow::FramelessWindow(QWidget* parent) : QWidget{parent}
2 {
3     // 默认允许窗口缩放
4     canResize = true;
5 }
```

然后，来到 `FramelessWindow.cpp`，修改 `mouseDoubleClickEvent()` 函数，将之前的逻辑包裹在 `if` 语句中，如下：

```
1 void FramelessWindow::mouseDoubleClickEvent(QMouseEvent* event)
2 {
3     if ( canResize ) {
4         // ...
5     }
```

```
6 }
```

来到 `FramelessWindow.cpp`，修改 `mouseMoveEvent()` 函数，如下：

```
1 void FramelessWindow::mouseMoveEvent(QMouseEvent* event)
2 {
3     // 1. 左键未按下
4     if ( !leftPressed ) {
5         // 窗口不是最大化状态 && 窗口可缩放，则光标移动到边界时，要变成缩放的形状（窗口处于
        // 最大化状态时，就无须改变光标形状了）
6         if ( this->WindowState().testFlag(Qt::WindowNoState) && canResize ) {
7             setCursorShape(globalPos);
8         }
9         return;
10    }
11 }
```

最后，来到 `mywidget.cpp`，禁止窗口缩放，如下：

```
1 MyWidget::MyWidget(QWidget* parent) : FramelessWindow(parent)
2 {
3     setResizable(false);
4 }
```

当然，此时还要将窗口上的最大化按钮删除掉，来到 `mywidget.cpp` 中，删除最大化按钮相关的代码，即可。

或者直接将最大化按钮隐藏 `btnMax->hide();`

此时，运行程序，不论是双击标题栏，还是拖放边界，都无法改变窗口的大小，如下：



## 9.2 如何使用

如果允许窗口缩放，那可以什么都不做，因为默认就是**允许窗口缩放**

当然也可以调用 `setResizable()` 函数，如下：

```
1 setResizable(true);
```

如果不允许窗口缩放，则传参 `false` 即可，如下：

```
1 setResizable(false);
```

当然了，如果不允许窗口缩放，也就不需要最大化按钮了！

## 第二章 图标字体

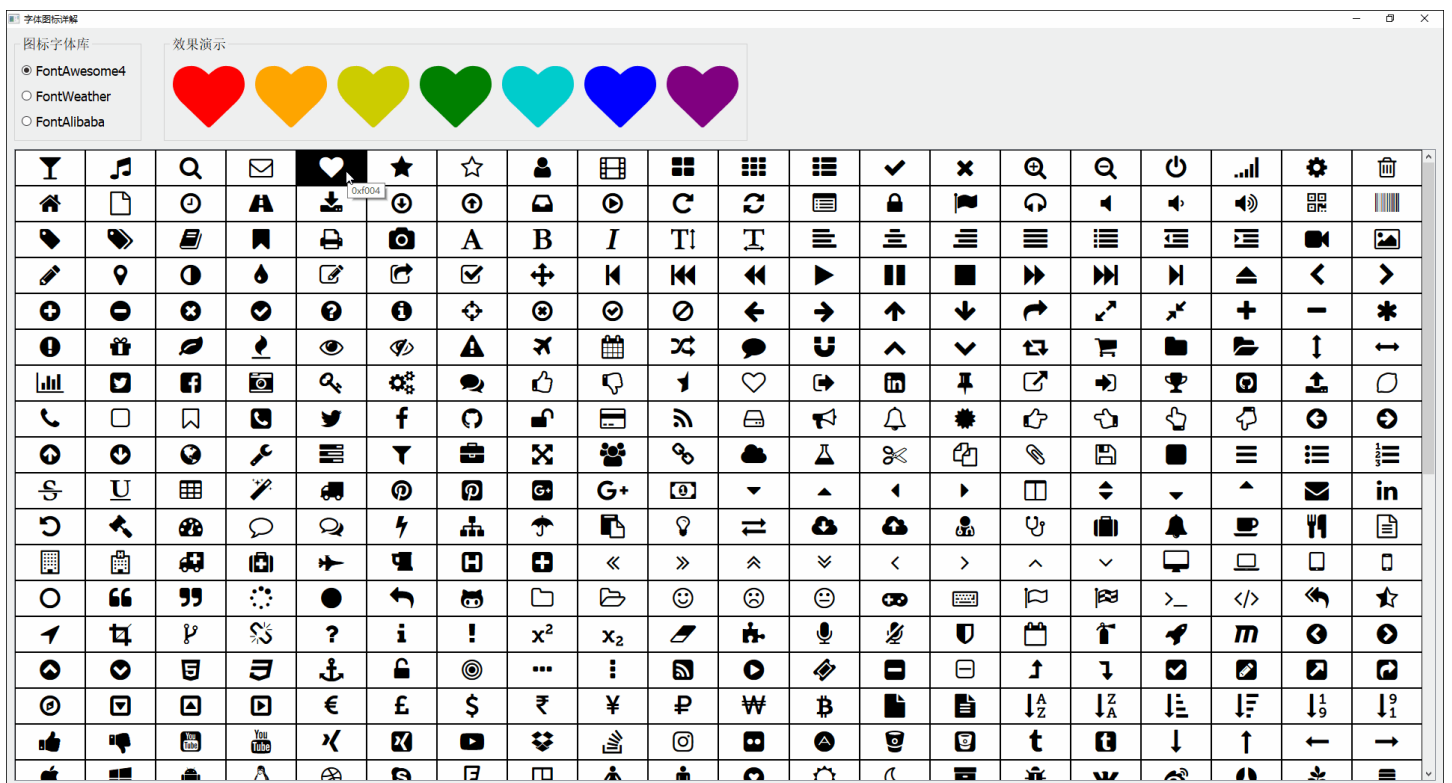
通过本章的学习，你将掌握：

- (1) 为什么要用到图标字体
- (2) 常用字体库一览: `FontAwesome`、`FontWeather`、`IconFont`
- (3) 创建自己的字体库: 把任何你想要的图标包含到自己的图标字体库中
- (4) 封装 `IconHelper` 类: 方便使用图标字体
- (5) 如何在你的项目中使用 `IconHelper` 类

接下来, 通过 10 个小节, 详细地、一步步地实现。

## 1. 效果演示、技术点

### 1.1 效果演示



### 1.2 为什么需要图标字体

在没有图标字体之前, 通常使用 `PNG` 或者 `SVG` 图片来设置控件的背景图片。此时就需要准备图片文件添加为资源文件

然而, 由于:

- 正常状态、按下状态、悬浮状态, 通常需要修改图片的颜色
- 由于换肤, 也需要修改图片的颜色

这就导致一个样式的图片, 需要多准备几张, 显然比较繁琐

有了图标字体库, 可以直接将图标对应的编码, 使用 `setText()` 作为文本设置给控件即可

可以随意设置文本的大小和颜色，达到多种图片的效果

## 2. 常用的图标字体库

图标字体库文件，以 `.ttf` 为后缀名

**TTF**（**TrueTypeFont**）是 **Apple** 公司和 **Microsoft** 公司共同推出的字体文件格式，它是最常用的一种字体文件格式

下面下载几个常用的字体库，然后使用专用的软件工具查看其中的图标。

### 2.1 FontAwesome

官网：<https://fontawesome.com/>，截止目前，最新版本是 **Font Awesome 6**

#### 2.1.1 Font Awesome 6

根据官网介绍（<https://fontawesome.com/icons>）：

### Font Awesome 6

Our latest, current, and most awesome version yet packed with tons more icons, new categories, a new Thin icon style, and the whole new Sharp family of icon styles.

And we expanded our styling toolkit, improved performance, and added even more plugins and packages.

Latest Version: **6.4.2**

26,233 Pro Icons

2,025 Free Icons

8 Icon Styles

68 Categories

Released: **Aug 8th, 2023**

Download

Documentation

Changelog

可见最新版本 **Font Awesome 6** 中，总共有 **26233** 个图标，**8** 种样式，**68** 个分类，**2025** 个免费的图标

68 个分类包括：

Accessib ility	Alert	Alphabet	Animals	Arrows	Astronomy	Automot ive	Buildi ngs	Busine ss
Campin g	Charity	Charts + Diagrams	Childhoo d	Clothing + Fashion	Coding	Commu nication	Conn ectivi ty	Constr uction
Design	Devices + Hardware	Disaster + Crisis	Editing	Educatio n	Emoji	Energy	Files	Film + Video
Food + Beverag e	Fruits + Vegetables	Gaming	Genders	Hallowee n	Hands	Holidays	Hous ehold	Huma nitaria n

Logistic s	Maps	Maritime	Marketin g	Mathema tics	Media Playback	Medical + Health	Mone y	Movin g
Music + Audio	Nature	Numbers	Photos + Images	Political	Punctuatio n + Symbols	Religion	Scien ce	Scienc e Fiction
Security	Shapes	Shopping	Social	Spinners	Sports + Fitness	Text Formatti ng	Time	Toggle
Transpo rtation	Travel + Hotel	Users + People	Weather	Writing				

8 种样式包括：






SOLID 、 REGULAR 、 LIGHT 、 THIN 、 DUOTONE 、 SHARP SOLID 、 SHARP  
REGULAR 、 SHARP LIGHT ,外加一个 BRANDS

如下来自官网说明：




## Families + Styles

There are three families of Font Awesome icons - each with a unique look, class name, and `@font-face` font-family. In both Font Awesome Classic and Sharp, there are five [styles of Font Awesome icons](#). Here are some examples:


### Classic Family

Style	Availability	Style class	font-weight	Looks like
<a href="#">Solid</a>	Free Plan	<code>fa-solid</code>	900	
<a href="#">Regular</a>	<a href="#">Pro only</a>	<code>fa-regular</code>	400	
<a href="#">Light</a>	<a href="#">Pro only</a>	<code>fa-light</code>	300	
<a href="#">Thin</a>	<a href="#">Pro only</a>	<code>fa-thin</code>	100	
<a href="#">Duotone</a>	<a href="#">Pro only</a>	<code>fa-duotone</code>	900	

### Sharp Family









Style	Availability	Style class	font-weight	Looks like
<a href="#">Solid</a>	<a href="#">Pro only</a>	<code>fa-sharp fa-solid</code>	900	
<a href="#">Regular</a>	<a href="#">Pro only</a>	<code>fa-sharp fa-regular</code>	400	
<a href="#">Light</a>	<a href="#">Pro only</a>	<code>fa-sharp fa-light</code>	300	
Thin	Coming Soon!			
Duotone	Coming Soon!			

### Brands Family

Style	Availability	Style class	font-weight	Looks like
<a href="#">Brands</a>	Free Plan	<code>fa-brands</code>	400	

从官网下载的 `Font Awesome 6` 的免费版本中，免费出来的图标基本都是 `SOLID` 样式 `fa-solid-900.ttf`



fontawesome-free-6.4.2-web > webfonts			
名称	修改日期	类型	大小
 fa-brands-400.ttf	2023-08-01 17:27	TTF 文件	186 KB
 fa-regular-400.ttf	2023-08-01 17:27	TTF 文件	62 KB
 fa-solid-900.ttf	2023-08-01 17:27	TTF 文件	386 KB
 fa-v4compatibility.ttf	2023-08-01 17:27	TTF 文件	10 KB
 fa-brands-400.woff2	2023-08-01 17:27	WOFF2 文件	108 KB
 fa-regular-400.woff2	2023-08-01 17:27	WOFF2 文件	24 KB
 fa-solid-900.woff2	2023-08-01 17:27	WOFF2 文件	147 KB
 fa-v4compatibility.woff2	2023-08-01 17:27	WOFF2 文件	5 KB

实际工作中，如果免费的 **SOLID** 样式不如 **REGULAR** 样式的图标更符合 **UI** 的需求，此时就需要购买收费的 **REGULAR** 样式图标了！

如果不想购买，建议使用阿里巴巴的图标，下面介绍！

## 2.1.2 Font Awesome 4

**Font Awesome 4** 是一个旧的版本。下载地址：<https://fontawesome.com/versions>

**Below Are Old + Unmaintained Versions!**

While the versions below still work and can be used, we are no longer actively maintaining them nor supporting them in our current services. But if you need or miss them for some reason, they'll always be here for you.


### Font Awesome 4

Latest Version: **4.7.0**      Released: **Oct 10th, 2016**

 [675 Free Icons](#)

 [16 Categories](#)


 [Download](#)


 [Documentation](#)

 [Changelog](#)

### Font Awesome 3


Latest Version: **3.2.1**      Released: **Jun 6th, 2013**

 [361 Free Icons](#)

 [7 Categories](#)

 [Download](#)

 [Documentation](#)

 [Changelog](#)

**Font Awesome 4** 版本是免费的，它提供了总共 **16** 个类别，共 **675** 个图标

下载解压后，将 **fontawesome-webfont.ttf** 文件添加到资源文件即可，如下：

font-awesome-4.7.0 > fonts			
名称	修改日期	类型	大小
FontAwesome.otf	2019-08-23 1:35	OpenType 字体...	132 KB
fontawesome-webfont.eot	2019-08-23 1:35	EOT 文件	162 KB
fontawesome-webfont.svg	2019-08-23 1:35	Microsoft Edge ...	434 KB
fontawesome-webfont.ttf	2019-08-23 1:35	TTF 文件	162 KB
fontawesome-webfont.woff	2019-08-23 1:35	WOFF 文件	96 KB
fontawesome-webfont.woff2	2019-08-23 1:35	WOFF2 文件	76 KB

## 2.2 FontWeather

官网: <https://www.pixeden.com/>

官网上同样有多个类别的图标: Food、Weather、Transportation、Books and Text、Location、Media 等

不过只有 Weather 图标是免费的, 总共有 208 个

下载解压后, 将 pe-icon-set-weather.ttf 文件添加到资源文件即可, 如下:

Weather-icon-set-font-PIXEDEN-v-1-0-0-PIXEDEN > pe-icon-set-weather > fonts			
名称	修改日期	类型	大小
pe-icon-set-weather.eot	2015-11-14 4:55	EOT 文件	80 KB
pe-icon-set-weather.svg	2015-11-14 4:55	Microsoft Edge ...	219 KB
pe-icon-set-weather.ttf	2015-11-14 4:55	TTF 文件	80 KB
pe-icon-set-weather.woff	2015-11-14 4:55	WOFF 文件	80 KB

## 2.3 FontAlibaba

官网: <https://www.iconfont.cn/>

IconFont 平台是阿里打造的矢量图标管理平台。

通常, 设计师将图标上传到该平台, 用户可下载多种格式的图标, 包括 png、svg、ttf。

官网显示, 目前已有 2600W+ 个图标

如何使用阿里图标呢？

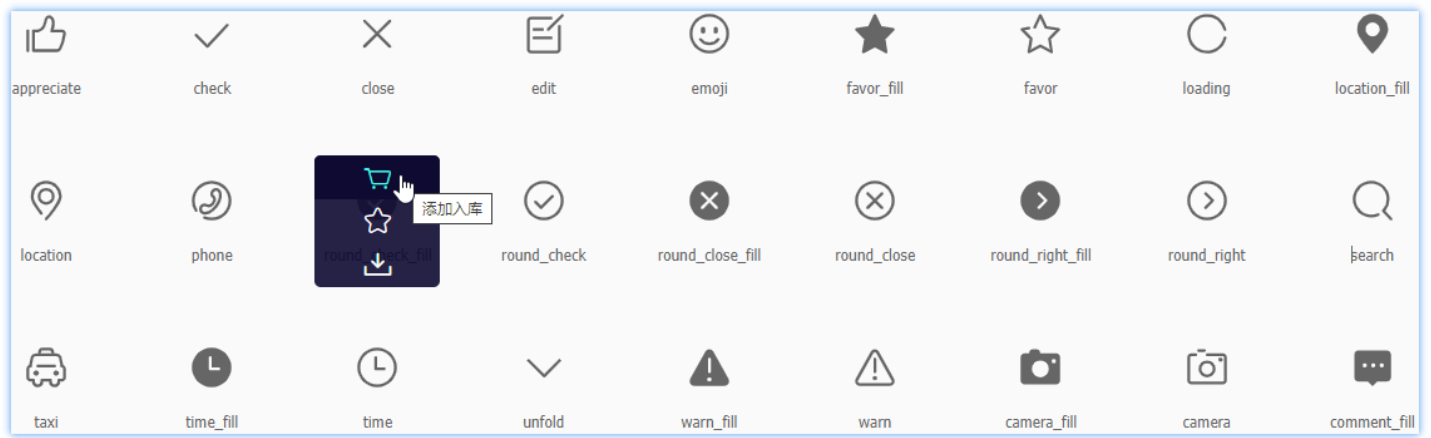
可以下载单个图标使用：找到需要的图标，设置图标大小、颜色，然后直接下载对应的 `png` 或 `svg` 格式，如下：



不过，通常的做法是：将多个图标打包成一个 `.ttf` 文件，下载使用，下面讲解使用步骤

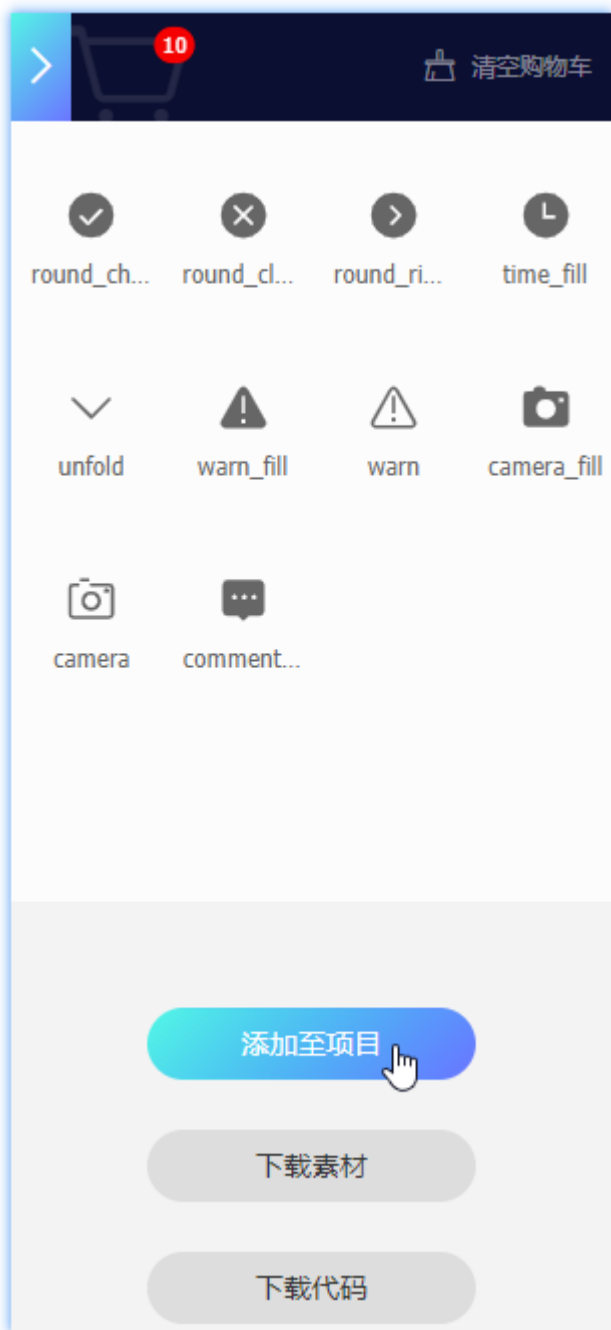
### 2.3.1 找图标，添加入库

打开阿里图标官网，找到喜欢的图标，鼠标悬浮其上，点击 "添加入库"，如下：



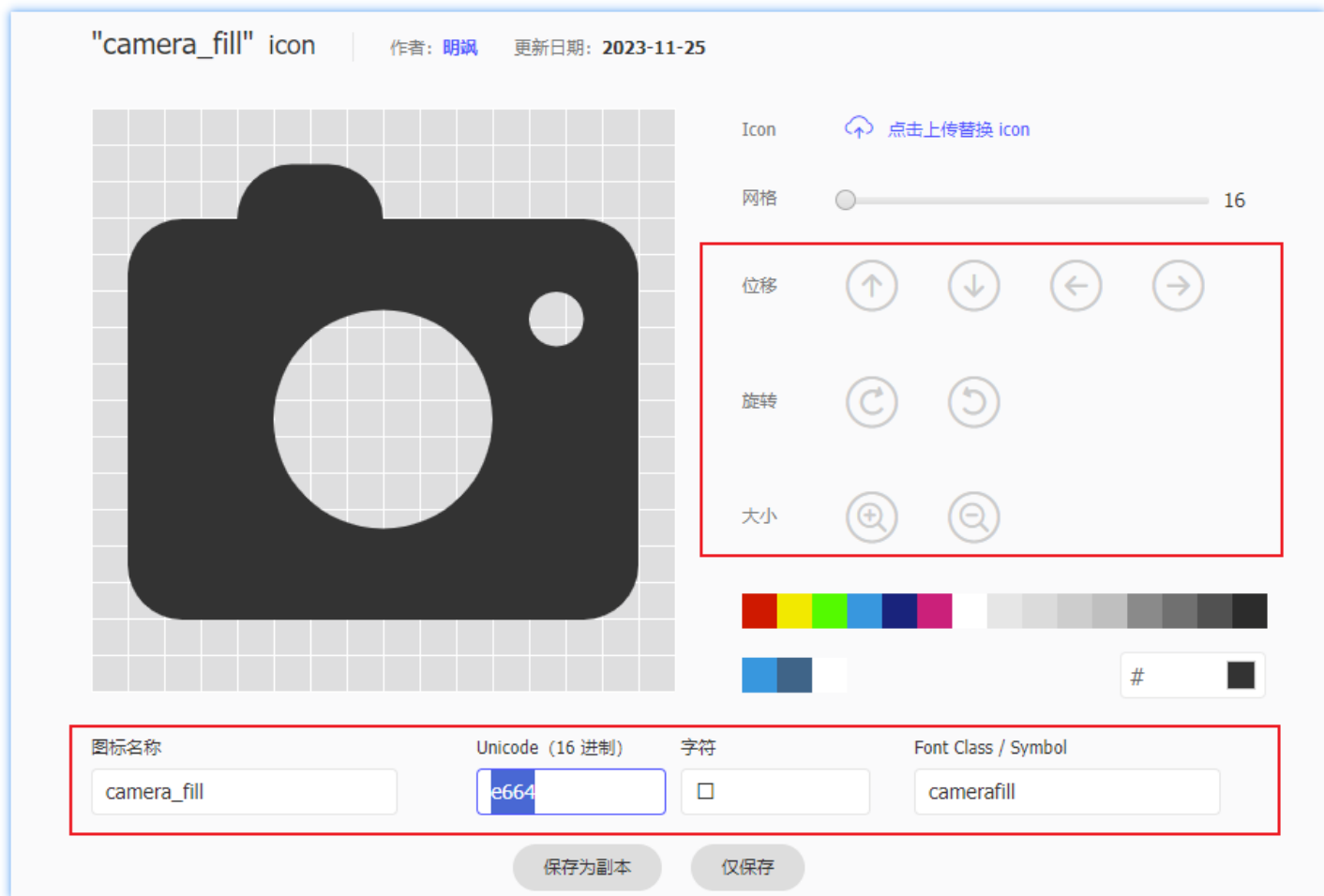
### 2.3.2 添加至项目

点击右上角的 "购物车" 图标，然后点击 "添加至项目"（没有项目会提示创建），将选中的10个图标添加到项目 **alibaba** 中



### 2.3.3 修改图标（可选）

打开刚创建的 `alibaba` 项目，鼠标悬浮在图标上边，点击 "编辑" 按钮，  
可以修改图标的位移、旋转、大小，还可以修改图标名称、Unicode编码、Font Class 等信息,如下：



### 2.3.4 修改项目（可选）

创建的项目，默认对应的 `font-family` 是 "`iconfont`"，它会在 Qt 程序中用到。可以点击 "项目设置"，进行修改，如下：

项目设置

项目名称

alibaba

项目描述

请输入项目描述 (可选)

GUID ?

生成 GUID

FontClass/  
Symbol 前缀

icon-

Font Family

iconfont

字体格式

☐ 彩色 ?

☒ WOFF2

☒ WOFF

☒ TTF

☐ EOT

☐ SVG

☐ Base64 ?

所有者 (超级  
管理员)

北京小小军(如需修改, 请在更多操作里转让项目)

协作者





请在「项目成员」中统一管理

保存

取消

## 2.3.5 打包下载

点击 "下载至本地" 按钮, 就可以打包下载到本地了。下载至本地的文件如下:

名称	修改日期	类型	大小
 demo.css	2023-11-25 10:42	层叠样式表文档	9 KB
 demo_index.html	2023-11-25 10:42	Chrome HTML D...	16 KB
 iconfont.css	2023-11-25 10:42	层叠样式表文档	1 KB
 iconfont.js	2023-11-25 10:42	JavaScript 文件	8 KB
 iconfont.json	2023-11-25 10:42	JSON File	2 KB
 iconfont.ttf	2023-11-25 10:42	TTF 文件	4 KB
 iconfont.woff	2023-11-25 10:42	WOFF 文件	2 KB
 iconfont.woff2	2023-11-25 10:42	WOFF2 文件	2 KB

将 `iconfont.ttf` 作为资源文件添加到 Qt 项目中即可。

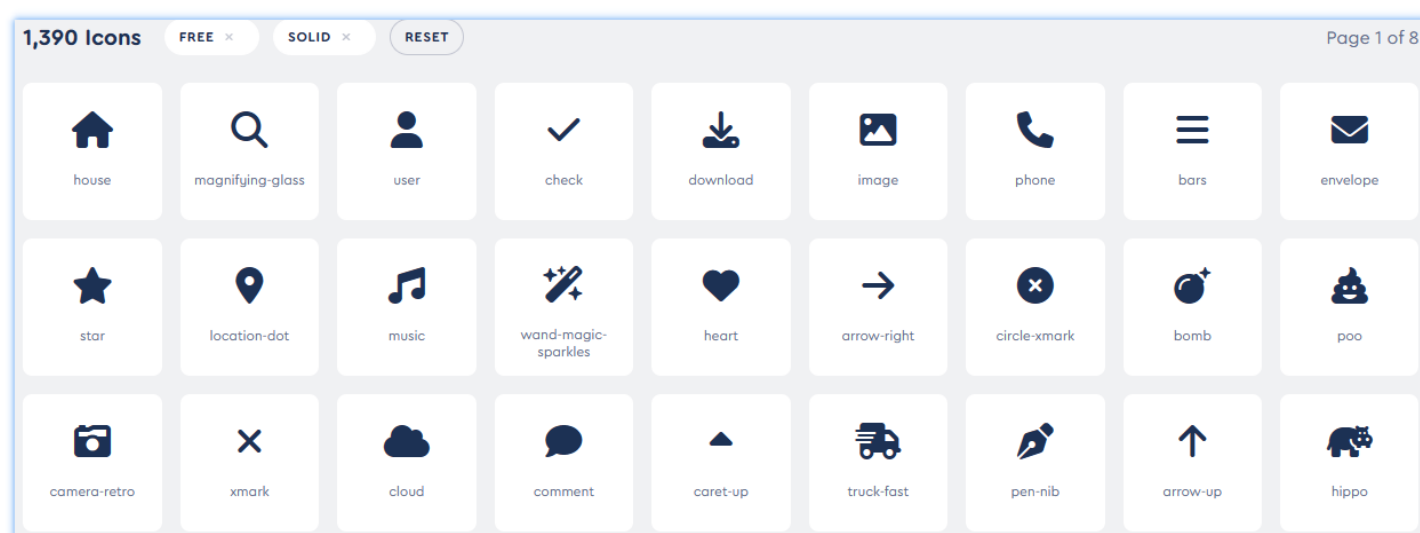
其中，还有一个 `demo_index.html` 文件，可以打开，其中有每个图标的效果图、编码，以及使用方法

## 2.4 查看字体库详细信息

下载到本地的 `.ttf` 文件，其中包含了很多图标，如何查看这些图标以及图标对应的 `Unicode` 编码呢？

### 2.4.1 官网查看图标信息

前面讲到 `Font Awesome 6` 的免费版本中，免费出来的图标基本都是 `SOLID` 样式，该样式的免费图标有 1390 个，如下：



在此，可以看到各种图标样式，并且点击对应的按钮，可以看到详细信息，包括 `Unicode` 编码以及使用方法

### 2.4.2 使用 FontForge

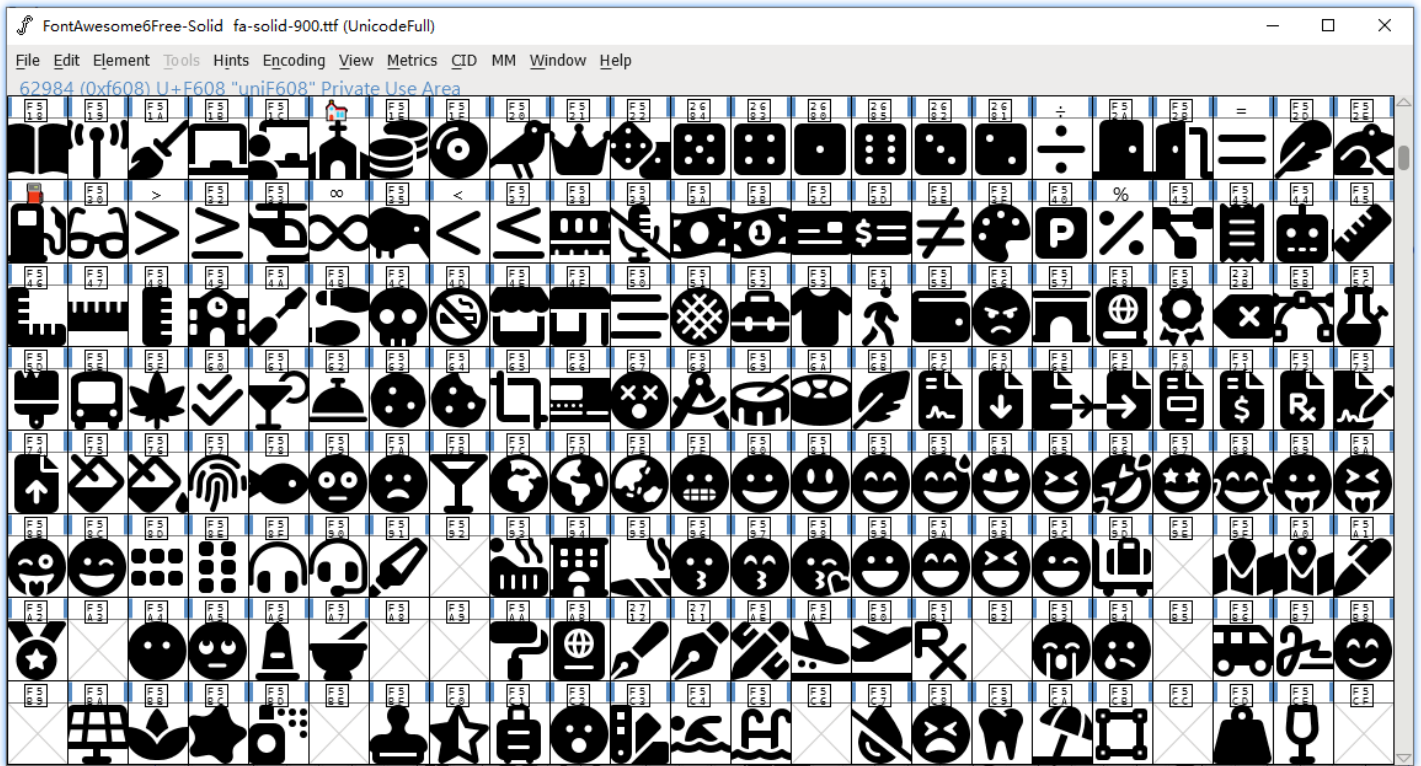
使用 `Font Forge` 软件，除了可以查看 `.ttf` 文件中的图标，还可以编辑或者自己创建图标。

官网为：<https://fontforge.org/>

以上 `Font Awesome 6` 免费版本中的 `SOLID` 样式的图标有 1390 个，其对应的文件是 `fa-solid-900.ttf`

使用 `Font Forge` 打开 `fa-solid-900.ttf` 文件的效果如下：

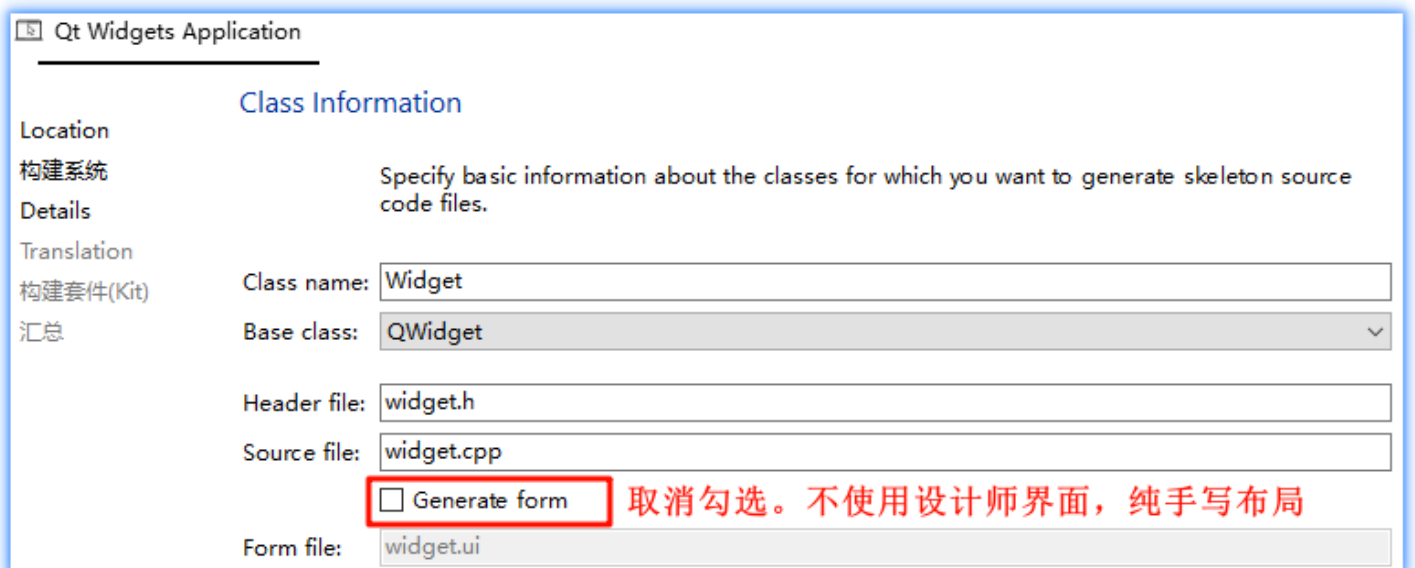




### 3. 界面布局(1)

#### 3.1 新建项目

首先创建一个基于 `QWidget` 的项目，不要勾选 “`Generate form`”，如下：



创建完成的后的项目目录如下：



## 3.2 完成布局

本节先完成整体界面布局，之后再显示图标字体。整体采用**垂直布局**：

- **顶部区域**

- 采用水平布局
- 字体库选择使用一个 `QGroupBox`，它采用垂直布局
- 效果演示使用一个 `QGroupBox`，它采用水平布局

- **图标字体区域**

- 整体使用 `QScrollArea`，因为图标过多，使用它可以滚动显示
- 内部采用 `QGridLayout`，方便以行列的形式显示图标

接下来，从零实现该布局

首先，来到 `widget.h` 中，声明以下成员变量，如下：

```
1 #include <QLabel>
2 #include <QRadioButton>
3 #include <QButtonGroup>
4 #include <QGridLayout>
5
6 class Widget : public QWidget
7 {
8 private:
9     QButtonGroup* radioGroup;
10    QGridLayout* gridLayout;
11
12    QLabel* lblRed;
13    QLabel* lblOrange;
14    QLabel* lblYellow;
15    QLabel* lblGreen;
16    QLabel* lblCyan;
17    QLabel* lblBlue;
18    QLabel* lblPurple;
```

```
19     QList<QLabel*> lblList;
20 };
```

然后，来到 `widget.cpp` 中，在构造中添加代码，如下：

```
1  #include <QGroupBox>
2  #include <QScrollArea>
3
4  Widget::Widget(QWidget* parent) : QWidget(parent)
5  {
6      this->setWindowTitle("图标字体详解");
7
8      // 整体采用垂直布局
9      QVBoxLayout* vLayout = new QVBoxLayout(this);
10     vLayout->setSpacing(10);
11     vLayout->setContentsMargins(10, 10, 10, 10);
12
13     // 1. 顶部栏
14     QWidget* topWidget = new QWidget(this);
15     QHBoxLayout* topLayout = new QHBoxLayout(topWidget);
16     topLayout->setSpacing(30);
17     topLayout->setContentsMargins(0, 0, 0, 0);
18
19     // 1.1 字体库选择
20     QGroupBox* fontGroup = new QGroupBox(this);
21     fontGroup->setStyleSheet("font: 18px");
22     fontGroup->setTitle("图标字体库");
23     QVBoxLayout* groupLayout = new QVBoxLayout(fontGroup);
24
25     QString radioText[3] = {"FontAwesome4", "FontWeather", "FontAlibaba"};
26     radioGroup = new QButtonGroup(this);
27     for ( int i = 0; i < 3; i++ ) {
28         QRadioButton* radio = new QRadioButton(this);
29         radio->setText(radioText[i]);
30         radio->setMinimumWidth(150);
31         radioGroup->addButton(radio, i);
32         groupLayout->addWidget(radio);
33     }
34
35     topLayout->addWidget(fontGroup);
36
37     // 1.2 效果演示
38     QGroupBox* showGroup = new QGroupBox(this);
39     showGroup->setStyleSheet("font: 18px");
```

```
40     showGroup->setTitle("效果演示");
41     QHBoxLayout* showLayout = new QHBoxLayout(showGroup);
42     showLayout->setSpacing(10);
43
44     for ( int i = 0; i < 7; i++ ) {
45         QLabel* lbl = new QLabel(this);
46         lbl->setMinimumSize(100, 100);
47         lbl->setAlignment(Qt::AlignCenter);
48
49         showLayout->addWidget(lbl);
50
51         lblList.append(lbl);
52     }
53
54     topLayout->addWidget(showGroup);
55
56     // 1.3 添加弹簧
57     QSpacerItem* sp = new QSpacerItem(8, 20, QSizePolicy::Expanding,
    QSizePolicy::Fixed);
58     topLayout->addItem(sp);
59
60     vLayout->addWidget(topWidget);
61
62     // 2. 主体
63     QScrollArea* scrollArea = new QScrollArea(this);
64     scrollArea->setWidgetResizable(true);
65     scrollArea->setMinimumHeight(400);    // 为防止主体部分高度过小，设置一个最下高度
66
67     QWidget* scrollAreaContents = new QWidget(this);
68
69     QVBoxLayout* verticalLayout = new QVBoxLayout(scrollAreaContents);
70     verticalLayout->setSpacing(0);
71     verticalLayout->setContentsMargins(0, 0, 0, 0);
72
73     QFrame* frame = new QFrame(scrollAreaContents);
74
75     gridLayout = new QGridLayout(frame);
76     gridLayout->setSpacing(0);
77     gridLayout->setContentsMargins(0, 0, 0, 0);
78
79     verticalLayout->addWidget(frame);
80
81     QSpacerItem* sp2 = new QSpacerItem(8, 20, QSizePolicy::Fixed,
    QSizePolicy::Expanding);
82     verticalLayout->addItem(sp2);
83
84     scrollArea->setWidget(scrollAreaContents);
```

```
85
86     vLayout->addWidget(scrollArea);
87 }
```

## 代码说明

### (1) 样式表

通过 `this->setStyleSheet("font: 18px");` 设置当前窗口中所有控件的字体大小

### (2) 单选按钮放到 `QButtonGroup` 中

除了实现单选按钮之间的互斥，还可以通过 `QButtonGroup` 来方便地获取选中了哪一个单选按钮

### (3) 合理使用弹簧

可以让控件顶格，防止缩放窗口时，控件宽度改变

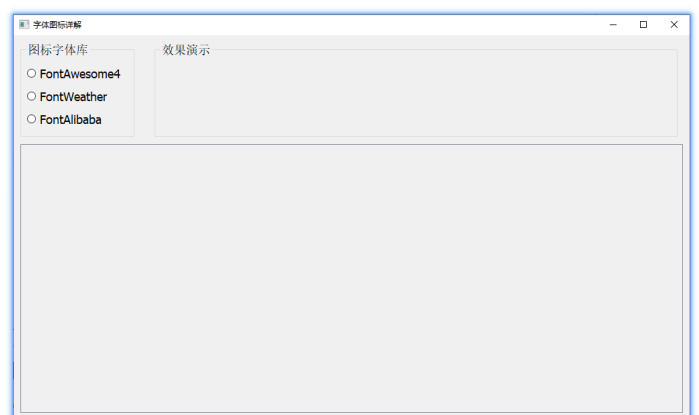
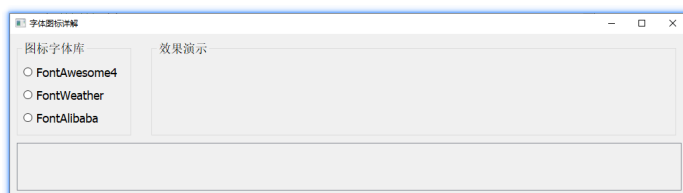
### (4) `QScrollArea`

当窗口无法容纳内容时，可以滚动显示完整内容

为防止打开窗口时，主体部分高度过小，加入以下语句：

```
1 scrollArea->setMinimumHeight(400);
```

增加前后，效果对比如下：



## 4. 界面布局(2)

上一节基本完成了界面布局，本节初始化标签数据，并把图标字体库添加到项目中

## 4.1 初始化标签

假设4个图标库中，图标个数为：

- FontAwesome4: 500个
- FontWeather: 200个
- FontAlibaba: 100个

当点击对应的单选按钮时，切换显示对应的图标（在 QLabel 中，显示 0,1,2,3,4 ...），可以看到 `QScrollArea` 的效果

首先，来到 `widget.h` 中，为单选按钮添加槽函数：

```
1 class Widget : public QWidget
2 {
3 private slots:
4     void radioClicked();
5 }
```

然后，来到 `widget.cpp` 中，实现 `radioClicked()`：

```
1 void Widget::radioClicked()
2 {
3     // 1. 先清空原来的标签
4     int count = gridLayout->count();
5     while ( count-- ) {
6         QWidget* widget = gridLayout->takeAt(count)->widget();
7         gridLayout->removeWidget(widget);
8         widget->setVisible(false); // 否则底部原来空白的地方被覆盖
9         widget->deleteLater();     // 否则看任务管理器，内存一直增长
10    }
11
12    // 2. 假设图标字体的个数
13    int checkedId = radioGroup->checkedId();
14    if ( checkedId == 0 ) {
15        count = 500;
16    } else if ( checkedId == 1 ) {
17        count = 200;
18    } else if ( checkedId == 2 ) {
```

```

19         count = 100;
20     }
21
22     // 3. 显示图标 (模拟-显示对应的索引)
23     int row, column;
24     for ( int i = 0; i < count; i++ ) {
25         QLabel* lbl = new QLabel();
26
27         lbl->setText(QString::number(i + 1));
28         lbl->setMinimumSize(48, 48);
29         lbl->setAlignment(Qt::AlignCenter);
30
31         // 设置正常颜色、光标悬浮的颜色
32         lbl->setStyleSheet(R"(
33             QLabel {color:#000000;border:1px solid #000000;background-
color:rgb(255 , 255 , 255);}
34             QLabel:hover {color:#ffffff;border:1px solid #000000;background-
color:rgb(0 , 0 , 0);}
35         )");
36
37         // 每行显示 20 个
38         row = i / 20;
39         column = i % 20;
40         gridLayout->addWidget(lbl, row, column);
41     }
42 }

```

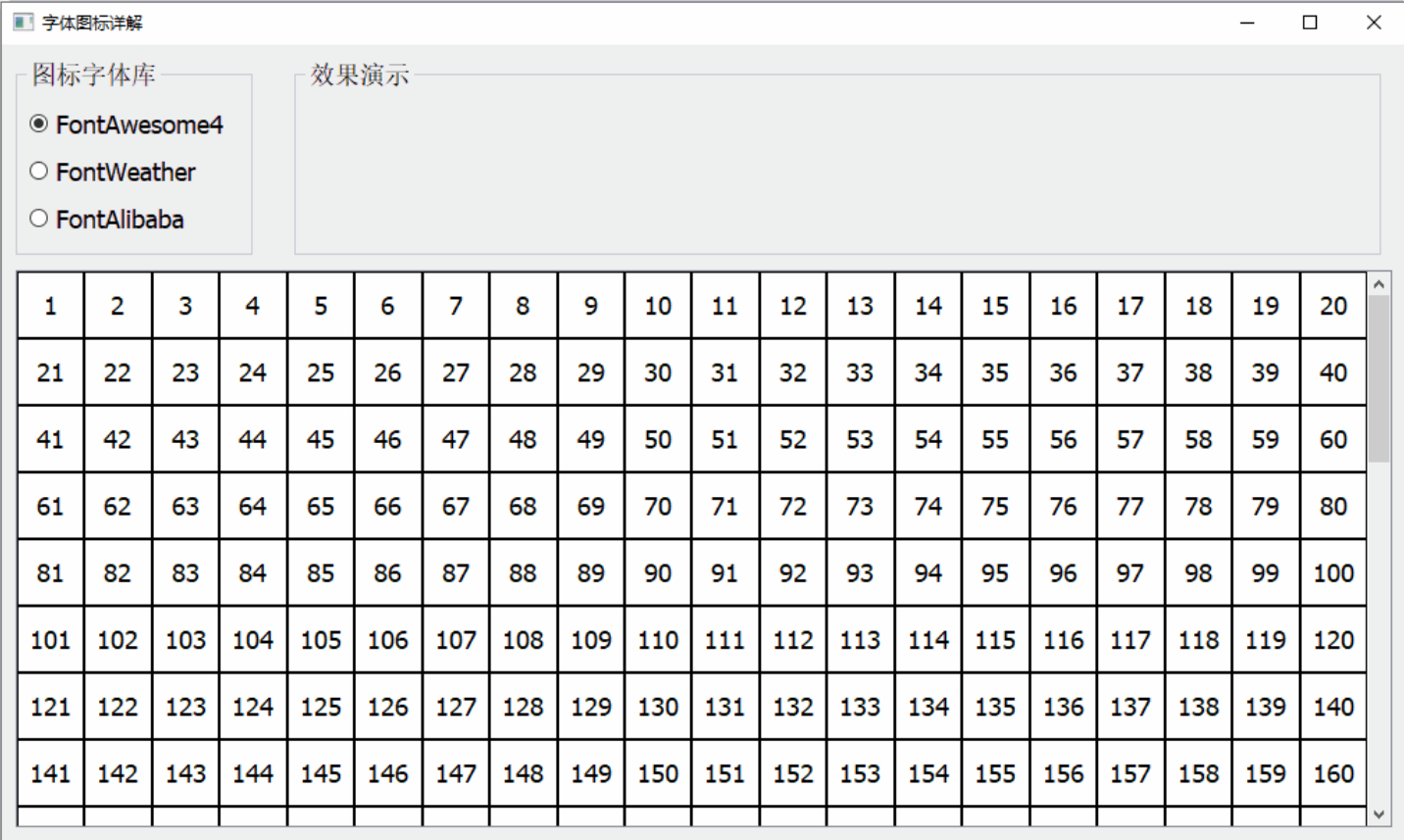
最后，来到 `widget.cpp` 的构造中，关联信号和槽：

```

1 Widget::Widget(QWidget* parent) : QWidget(parent)
2 {
3     // ...
4
5     for ( int i = 0; i < 3; i++ ) {
6         // ...
7         connect(radio, &QRadioButton::clicked, this, &Widget::radioClicked);
8     }
9
10    // 默认显示 FontAwesome
11    radioGroup->button(0)->click();
12 }

```

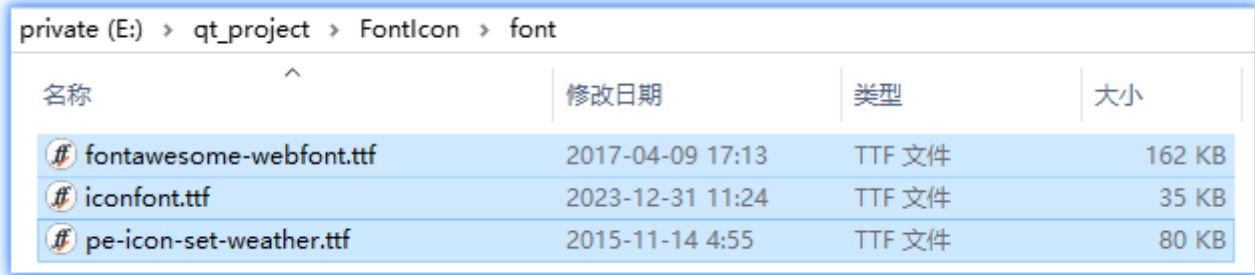
此时，运行效果如下：



## 4.2 将字体库添加为资源文件

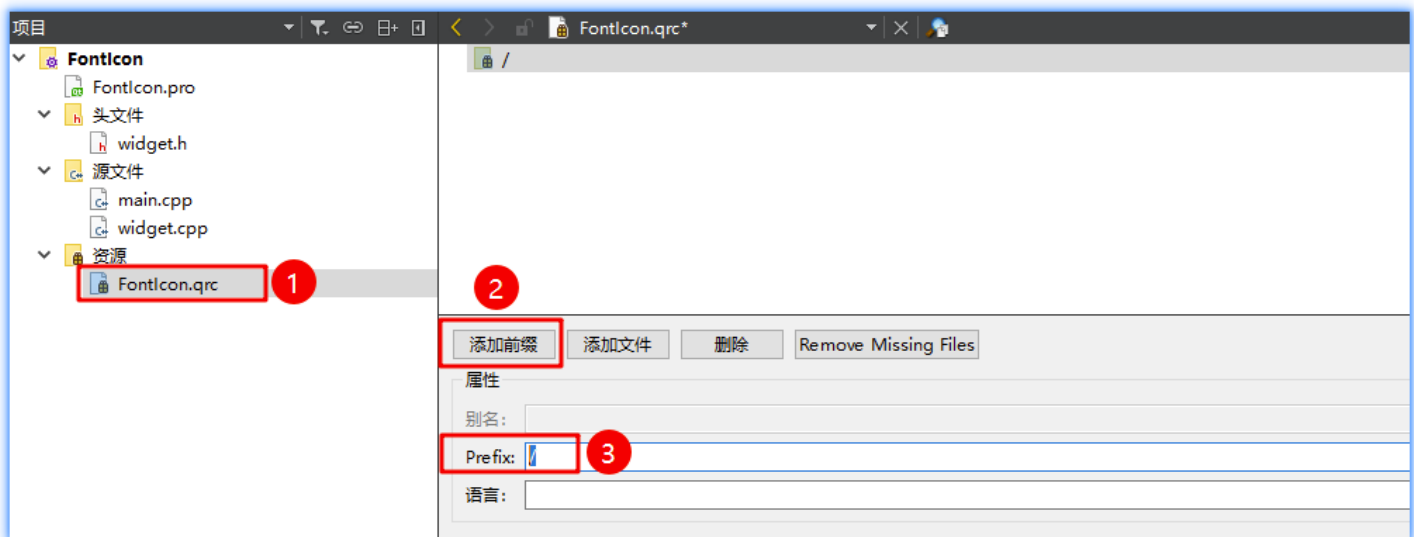
接下来，将3个图标字体库对应的 `.ttf` 文件，作为资源文件添加到项目中

首先，在项目根目录下创建一个 `font` 目录，并将 3个 `.ttf` 文件拷贝进来，如下：

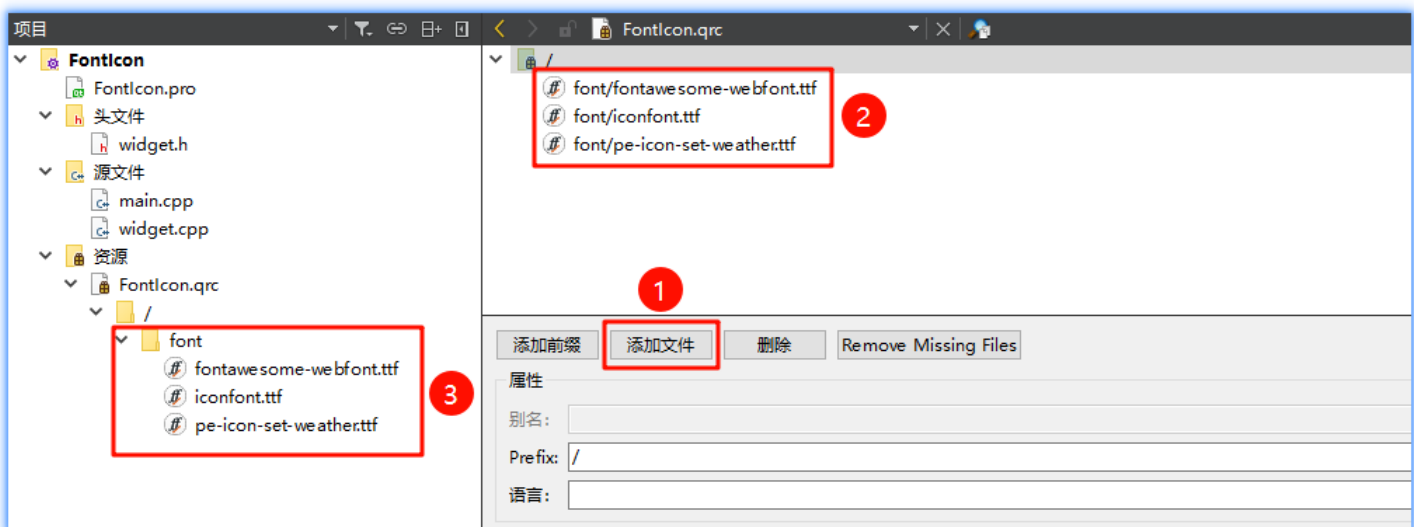


接着，在项目名称右键，选择“添加新文件”，选择“`Qt Resource File`”，将资源文件命名为 `FontIcon`，并添加前缀，如下：





接着，点击“添加文件”，将3个图标字体库文件，添加进来，如下：



之后，就可以在代码中引用这3个图标字体库了！

## 5. 封装IconHelper(1) 加载图标字体库

为了方便使用图标字体，首先来封装一个 `IconHelper` 类。

### 5.1 查看当前系统字体

Qt 中提供了一个 `QFontDatabase` 类，通过它可以获取到系统中可用字体族、字体样式、字体大小以常用的 `Consolas` 字体为例：

```
1 "Consolas"
```

2	"Regular"	"6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72 "
3	"Bold"	"6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72 "
4	"Bold Italic"	"6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72 "
5	"Italic"	"6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72 "

其中，`Consolas` 是字体族，`Regular`、`Bold`、`Bold Italic`、`Italic` 是样式，`6 7 8 9 ... 48 72` 是字体大小

可以使用以下代码段，来查看系统中所有支持的字体：

```
1 QFontDatabase database;
2 foreach (QString strFamily, database.families()) {
3     qDebug() << strFamily;
4     foreach (QString strStyle, database.styles(strFamily)) {
5         QString strSizes;
6         foreach (int points, database.smoothSizes(strFamily, strStyle)) {
7             strSizes += QString::number(points) + " ";
8         }
9         qDebug() << "\t" << strStyle << "\t" << strSizes;
10    }
11 }
```

把这段代码拷贝到 `widget.cpp` 的构造中，执行结果如下：

```
1 "Agency FB"
2     "Regular"        "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72
   "
3     "Bold"           "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72 "
4 "Algerian"
5     "Regular"        "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72
   "
6 "Arial"
7     "Regular"        "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72
   "
8     "Bold"           "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72 "
9     "Bold Italic"    "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36
   48 72 "
10    "Italic"          "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72 "
11
12 // 省略几百行
13
14 "宋体"
```

```

15      "常规"          "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72 "
16  "幼圆"
17      "Regular"      "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72
18  "
19  "微软雅黑"
20      "Regular"      "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72
21  "
22  "微软雅黑 Light"
23      "Regular"      "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72
24  "
25  "隶书"
26      "Regular"      "6 7 8 9 10 11 12 14 16 18 20 22 24 26 28 36 48 72
27  "

```

当向系统中添加了 3 个图标字体后，以上打印也会列出新添加的 3 个图标字体

## 5.2 加载图标字体库

### 5.2.1 新建 IconHelper 类

在项目名右键，添加 `IconHelper` 类，让它继承 `QObject`，如下：

Define Class

Class name:

Base class:

☒ Include QObject  
☐ Include QWidget  
☐ Include QMainWindow  
☐ Include QDeclarativeItem - Qt Quick 1  
☐ Include QQuickItem - Qt Quick 2  
☐ Include QSharedData  
☒ Add Q\_OBJECT  
☐ Add QML\_ELEMENT

Header file:

Source file:

路径:

## 5.2.2 创建单例

首先，来到 `icon_helper.h` 中，定义宏，单例，以及 `QFont` 成员变量，如下：

```
1 #include <QLabel>
2 #include <QAbstractButton>
3
4 #define FONT_PATH_AWESOME4    ":/font/fontawesome-webfont.ttf"
5 #define FONT_FAMILY_AWESOME  "FontAwesome"
6
7 #define FONT_PATH_WEATHER    ":/font/pe-icon-set-weather.ttf"
8 #define FONT_FAMILY_WEATHER  "pe-icon-set-weather"
9
10 #define FONT_PATH_ALIBABA    ":/font/iconfont.ttf"
11 #define FONT_FAMILY_ALIBABA  "FontAlibaba"
12
13 class IconHelper : public QObject
14 {
15 private:
16     static IconHelper* self;
17     explicit IconHelper(QObject* parent = nullptr);
18
19 public:
20     static IconHelper* instance();
21
22 private:
23     QFont iconFontAwesome4;
24     QFont iconFontWeather;
25     QFont iconFontAlibaba;
26 };
```

接着，在 `icon_helper.cpp` 中实现单例，如下：

```
1 #include <QMutex>
2
3 IconHelper* IconHelper::self = nullptr;
4 IconHelper* IconHelper::instance()
5 {
6     if ( !self ) {
7         /* 这里定义局部变量，简化锁定和解锁操作
8          * 1. 在该locker被创建时候上锁，
```

```

9      * 2. 当所函数执行完毕后, locker被销毁, 自动解锁。
10     */
11     QMutex mutex;
12     QMutexLocker locker(&mutex);
13     //上面等价于:
14     // mutex.lock();
15     // ...
16     // mutex.unlock();
17     if ( !self ) {
18         self = new IconHelper();
19     }
20 }
21
22 return self;
23 }

```

### 5.2.3 加载图标字体库

在 `icon_helper.cpp` 的构造中, 把3个图标字体添加到系统中, 如下:

```

1  #include <QFontDatabase>
2
3  IconHelper::IconHelper(QObject* parent) : QObject{parent}
4  {
5      QFontDatabase fontDb;
6      int fontId;
7
8      // 添加 FontAwesome
9      if ( !fontDb.families().contains(FONT_FAMILY_AWESOME4) ) {
10         fontId = fontDb.addApplicationFont(FONT_PATH_AWESOME4);
11         QStringList fontFamilies = fontDb.applicationFontFamilies(fontId);
12         if ( fontFamilies.count() > 0 ) {
13             iconFontAwesome4 = QFont(FONT_FAMILY_AWESOME4);
14             iconFontAwesome4.setHintingPreference(QFont::PreferNoHinting);
15         }
16     }
17
18     // 添加 FontWeather
19     if ( !fontDb.families().contains(FONT_FAMILY_WEATHER) ) {
20         fontId = fontDb.addApplicationFont(FONT_PATH_WEATHER);
21         QStringList fontFamilies = fontDb.applicationFontFamilies(fontId);
22         if ( fontFamilies.count() > 0 ) {
23             iconFontWeather = QFont(FONT_FAMILY_WEATHER);

```

```

24         iconFontWeather.setHintingPreference(QFont::PreferNoHinting);
25     }
26 }
27
28 // 添加 FontAlibaba
29 if ( !fontDb.families().contains(FONT_FAMILY_ALIBABA) ) {
30     fontId = fontDb.addApplicationFont(FONT_PATH_ALIBABA);
31     QStringList fontFamilies = fontDb.applicationFontFamilies(fontId);
32     if ( fontFamilies.count() > 0 ) {
33         iconFontAlibaba = QFont(FONT_FAMILY_ALIBABA);
34         iconFontAlibaba.setHintingPreference(QFont::PreferNoHinting);
35     }
36 }
37 }

```

此时就把这 3 个字体族添加到了系统中，在 `widget.cpp` 的构造函数的最后，再次打印系统中所有字体，就可以看到加载到系统中的 3 个图标字体了，如下：

```

1 Widget::Widget(QWidget* parent) : QWidget(parent)
2 {
3     // ...
4
5     #if 1
6     IconHelper::instance();
7
8     QFontDatabase database;
9     foreach (QString strFamily, database.families()) {
10         qDebug() << strFamily;
11         foreach (QString strStyle, database.styles(strFamily)) {
12             QString strSizes;
13             foreach (int points, database.smoothSizes(strFamily, strStyle)) {
14                 strSizes += QString::number(points) + " ";
15             }
16             qDebug() << "\t" << strStyle << "\t" << strSizes;
17         }
18     }
19 #endif
20 }

```

## 6. 封装IconHelper(2) 获取图标字体编码

每一个图标字体库中，有很多的图标，每个图标对应一个编码。本节获取每个图标字体库中所有的图标编码。

### 6.1 声明3个成员函数

在 `icon_helper.h` 中，声明4个成员函数，如下：

```
1 class IconHelper : public QObject
2 {
3 public:
4     QList<int> getAwesome4Icons();
5     QList<int> getWeatherIcons();
6     QList<int> getAlibabaIcons();
7 };
```

### 6.2 获取 FontAwesome4 所有图标

在 `icon_helper.cpp` 中，实现 `getAwesome4Icons()` 函数，如下：

```
1 QList<int> IconHelper::getAwesome4Icons()
2 {
3     int start = 0xf000;
4     int end = 0xf2e0;
5
6     // 0xf000~0xf2e0 范围内，有些是空的
7     QList<int> emptyList = {0xf00f, 0xf01f, 0xf020, 0xf03f, 0xf04f, 0xf05f,
8                             0xf06f, 0xf07f, 0xf08f, 0xf09f, 0xf0af, 0xf0b3, 0xf0b4, 0xf0b5, 0xf0b6, 0xf0b7,
9                             0xf0b8, 0xf0b9, 0xf0ba, 0xf0bb, 0xf0bc, 0xf0bd,
10                            0xf0be, 0xf0bf, 0xf0bf, 0xf0cf, 0xf0df, 0xf0ef, 0xf0ff, 0xf10f, 0xf116, 0xf117,
11                            0xf11f, 0xf12f, 0xf13f, 0xf14f, 0xf15f, 0xf16f,
12                            0xf17f, 0xf18f, 0xf19f, 0xf1af, 0xf1bf, 0xf1cf, 0xf1df, 0xf1ef, 0xf1ff, 0xf20f,
13                            0xf21f, 0xf220, 0xf22e, 0xf22f, 0xf23f, 0xf24f,
14                            0xf25f, 0xf26f, 0xf27f, 0xf28f, 0xf29f, 0xf2af, 0xf2bf, 0xf2cf, 0xf2df};
15
16     QList<int> list;
17     for ( int icon = start; icon <= end; icon++ ) {
18         // 0xf000~0xf2e0 范围内，有些是空的
19         if ( emptyList.contains(icon) ) {
```

```

16         continue;
17     }
18     list.append(icon);
19 }
20
21 return list;
22 }

```

## 6.3 获取 FontWeather 所有图标

在 `icon_helper.cpp` 中，实现 `getWeatherIcons()` 函数，如下：

```

1 QList<int> IconHelper::getWeatherIcons()
2 {
3     QList<int> list;
4
5     for ( int i = 0xe900; i <= 0xe9cf; i++ ) {
6         list.append(i);
7     }
8
9     return list;
10 }

```

## 6.4 获取 FontAlibaba 所有图标

在 `icon_helper.cpp` 中，实现 `getAlibabaIcons()` 函数，如下：

```

1 QList<int> IconHelper::getAlibabaIcons()
2 {
3     QList<int> list;
4
5     // 图表
6     for ( int i = 0xa000; i <= 0xa027; i++ ) {
7         list.append(i);
8     }
9
10    // 仪表盘
11    for ( int i = 0xa100; i <= 0xa106; i++ ) {
12        list.append(i);

```



```
13     }
14
15     // 表格
16     for ( int i = 0xa200; i <= 0xa20e; i++ ) {
17         list.append(i);
18     }
19
20     // 日历
21     for ( int i = 0xa300; i <= 0xa30b; i++ ) {
22         list.append(i);
23     }
24
25     // 树状图
26     for ( int i = 0xa400; i <= 0xa402; i++ ) {
27         list.append(i);
28     }
29
30     // 标签页
31     for ( int i = 0xa500; i <= 0xa505; i++ ) {
32         list.append(i);
33     }
34
35     // 最大化最小化
36     for ( int i = 0xa600; i <= 0xa603; i++ ) {
37         list.append(i);
38     }
39
40     // 文本编辑器
41     for ( int i = 0xa700; i <= 0xa70a; i++ ) {
42         list.append(i);
43     }
44
45     // 加号
46     for ( int i = 0xa900; i <= 0xa901; i++ ) {
47         list.append(i);
48     }
49
50     // 天气
51     for ( int i = 0xaa00; i <= 0xaa0e; i++ ) {
52         list.append(i);
53     }
54
55     // 换肤
56     for ( int i = 0xab00; i <= 0xab03; i++ ) {
57         list.append(i);
58     }
59
```

```
60     return list;
61 }
```

## 7. 封装IconHelper(3) 为控件设置图标字体

接下来，实现两个函数来为标签、按钮来设置图标

### 7.1.1 为标签设置图标字体

首先，在 `icon_helper.h` 中，声明 `setIcon()` 函数，如下：

```
1 class IconHelper : public QObject
2 {
3     // 给标签设置图标
4     void setIcon(QLabel* lbl, QString fontFamily, int icon, quint8 size = 10);
5 }
```

然后，在 `icon_helper.cpp` 中，实现 `setIcon()` 函数，如下：

```
1 void IconHelper::setIcon(QLabel* lbl, QString fontFamily, int icon, quint8
    size)
2 {
3     if ( fontFamily == FONT_FAMILY_AWESOME4 ) {
4         iconFontAwesome4.setPixelSize(size);
5         lbl->setFont(iconFontAwesome4);
6         lbl->setText((QChar)icon);
7     } else if ( fontFamily == FONT_FAMILY_WEATHER ) {
8         iconFontWeather.setPixelSize(size);
9         lbl->setFont(iconFontWeather);
10        lbl->setText((QChar)icon);
11    } else if ( fontFamily == FONT_FAMILY_ALIBABA ) {
12        iconFontAlibaba.setPixelSize(size);
13        lbl->setFont(iconFontAlibaba);
14        lbl->setText((QChar)icon);
15    }
16 }
```

## 7.1.2 为按钮设置图标字体

首先，在 `icon_helper.h` 中，声明 `setIcon()` 函数，如下：

```
1 class IconHelper : public QObject
2 {
3     // 给按钮设置图标
4     void setIcon(QAbstractButton* btn, QString fontFamily, int icon, quint8
        size = 10);
5 }
```

然后，在 `icon_helper.cpp` 中，实现这两个函数，如下：

```
1 void IconHelper::setIcon(QAbstractButton* btn, QString fontFamily, int icon,
    quint32 size)
2 {
3     if ( fontFamily == FONT_FAMILY_AWESOME4 ) {
4         iconFontAwesome4.setPixelSize(size);
5         btn->setFont(iconFontAwesome4);
6         btn->setText((QChar)icon);
7     } else if ( fontFamily == FONT_FAMILY_WEATHER ) {
8         iconFontWeather.setPixelSize(size);
9         btn->setFont(iconFontWeather);
10        btn->setText((QChar)icon);
11    } else if ( fontFamily == FONT_FAMILY_ALIBABA ) {
12        iconFontAlibaba.setPixelSize(size);
13        btn->setFont(iconFontAlibaba);
14        btn->setText((QChar)icon);
15    }
16 }
```

## 8. 封装IconHelper(4) 用图标字体生成图片

有时候有这么一种需求：将图标字体转换为图片。这里来实现这个功能

首先，在 `icon_helper.h` 中，声明 `getPixmap()` 函数，如下：

```

1 class IconHelper : public QObject
2 {
3     // 根据给定的图标字体, 生成一个QPixmap
4     QPixmap getPixmap(const QColor& color, QString fontFamily, int icon,
5         quint8 size = 10, quint8 width = 12, quint8 height = 12);
6 }

```

然后, 在 `icon_helper.cpp` 中, 实现该函数, 如下:

```

1 QPixmap IconHelper::getPixmap(const QColor& color, QString fontFamily, int
    icon, quint8 size, quint8 width, quint8 height)
2 {
3     QFont font;
4
5     if ( fontFamily == FONT_FAMILY_AWESOME4 ) {
6         font = iconFontAwesome4;
7     } else if ( fontFamily == FONT_FAMILY_WEATHER ) {
8         font = iconFontWeather;
9     } else if ( fontFamily == FONT_FAMILY_ALIBABA ) {
10        font = iconFontAlibaba;
11    }
12
13    // 把图形字体绘制到图片上
14    QPixmap pix(width, height);
15    pix.fill(Qt::transparent);
16
17    QPainter painter;
18    painter.begin(&pix);
19    painter.setRenderHints(QPainter::Antialiasing |
20        QPainter::TextAntialiasing);
21    painter.setPen(color);
22
23    font.setPixelSize(size);
24    painter.setFont(font);
25
26    painter.drawText(pix.rect(), Qt::AlignCenter, (QChar)icon);
27    painter.end();
28
29    return pix;
30 }

```

## 9. 显示所有图标字体

来到 `widget.cpp`，修改 `radioClicked()` 槽函数，如下：

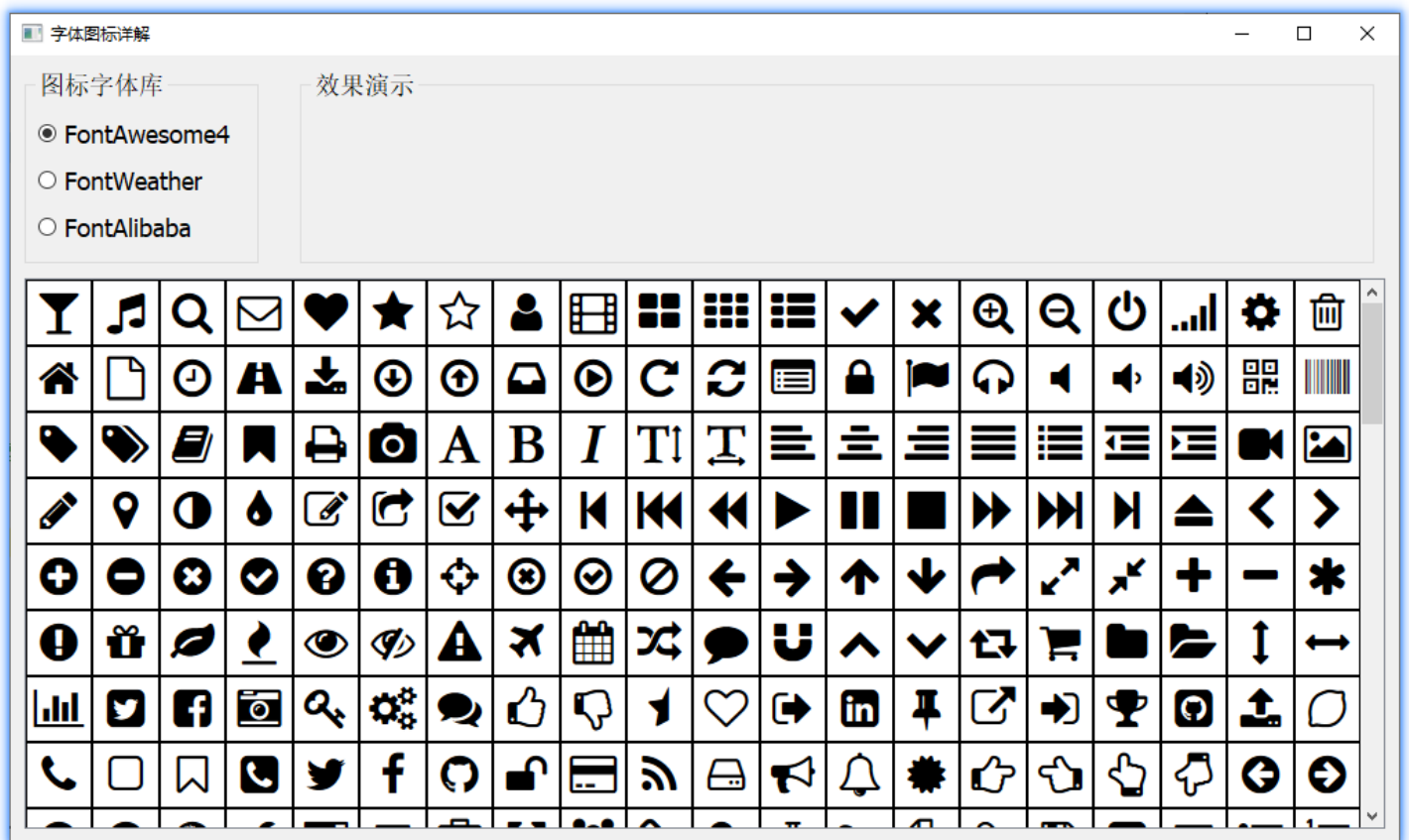
```
1 void Widget::radioClicked()
2 {
3     // 1. 先清空原来的标签
4     int count = gridLayout->count();
5     while ( count-- ) {
6         QWidget* widget = gridLayout->takeAt(count)->widget();
7         gridLayout->removeWidget(widget);
8         widget->setVisible(false); // 否则底部原来空白的地方被覆盖
9         widget->deleteLater();     // 否则看任务管理器，内存一直增长
10    }
11
12    // 2. 获取所有的图标
13    QString iconFontName;
14    QList<int> icons;
15
16    int checkedId = radioGroup->checkedId();
17    if ( checkedId == 0 ) {
18        iconFontName = FONT_FAMILY_AWESOME4;
19        icons = IconHelper::instance()->getAwesome4Icons();
20    } else if ( checkedId == 1 ) {
21        iconFontName = FONT_FAMILY_WEATHER;
22        icons = IconHelper::instance()->getWeatherIcons();
23    } else if ( checkedId == 2 ) {
24        iconFontName = FONT_FAMILY_ALIBABA;
25        icons = IconHelper::instance()->getAlibabaIcons();
26    }
27
28    // 3. 显示图标
29    int row, column;
30    for ( int i = 0; i < icons.size(); i++ ) {
31        QLabel* lbl = new QLabel();
32
33        lbl->setText(QChar(icons[i]));
34        lbl->setMinimumSize(48, 48);
35        lbl->setAlignment(Qt::AlignCenter);
36
37        lbl->setStyleSheet(R"(
38            QLabel {color:#000000;border:1px solid #000000;background-
39            color:rgb(255 , 255 , 255);}
40            QLabel:hover {color:#ffffff;border:1px solid #000000;background-
41            color:rgb(0 , 0 , 0);}
42        )");
```

```

41
42     IconHelper::instance()->setIcon(lbl, iconFontName, icons[i], 24);
43
44     row = i / 20;
45     column = i % 20;
46     gridLayout->addWidget(lbl, row, column);
47 }
48 }

```

此时，运行程序就可以显示各个图标库中的图标了，如下：



## 10. 七色显示选中图标

接下来实现，当鼠标悬浮在对应标签上时，将对应的图标显示在 "效果展示" 区，并以 "赤橙黄绿青蓝紫" 七色来显示。

### 10.1.1 为标签添加 ToolTip

来到 `widget.cpp`，修改 `radioClicked()` 函数，如下：

```

1 void IconFontWidget::radioClicked()

```

```

2 {
3     // ...
4     for ( int i = 0; i < icons.size(); i++ ) {
5         // ...
6         lbl->setToolTip("0x" + QString::number(icons[i], 16));
7     }
8 }

```

此时，当鼠标悬浮在标签上时，会显示出图标对应的编码。

### 10.1.2 显示区显示图标

首先，来到 `widget.h`，声明事件过滤函数，如下：

```

1 class IconFontWidget : public QWidget
2 {
3     // ...
4 protected:
5     bool eventFilter(QObject* watched, QEvent* event);
6 };

```

然后，来到 `widget.cpp`，实现事件过滤函数，如下：

```

1 bool Widget::eventFilter(QObject* watched, QEvent* event)
2 {
3     if ( event->type() == QEvent::Enter ) {
4         QLabel* lbl = (QLabel*)watched;
5         if ( lbl != nullptr ) {
6             int icon = lbl->toolTip().toInt(nullptr, 16);
7             QString iconFontName;
8
9             int id = radioGroup->checkedId();
10            if ( id == 0 ) {
11                iconFontName = FONT_FAMILY_AWESOME4;
12            } else if ( id == 1 ) {
13                iconFontName = FONT_FAMILY_WEATHER;
14            } else if ( id == 2 ) {
15                iconFontName = FONT_FAMILY_ALIBABA;
16            }
17        }
18    }
19    return false;
20 }

```

```

17
18         for ( int i = 0; i < 7; i++ ) {
19             IconHelper::instance()->setIcon(lblList[i], iconFontName,
            icon, 92);
20         }
21     }
22 }
23
24     return QWidget::eventFilter(watched, event);
25 }

```

然后，修改 `widget.cpp` 中的 `radioClicked()` 函数，在创建标签的同时，为每个标签安装事件过滤器，如下：

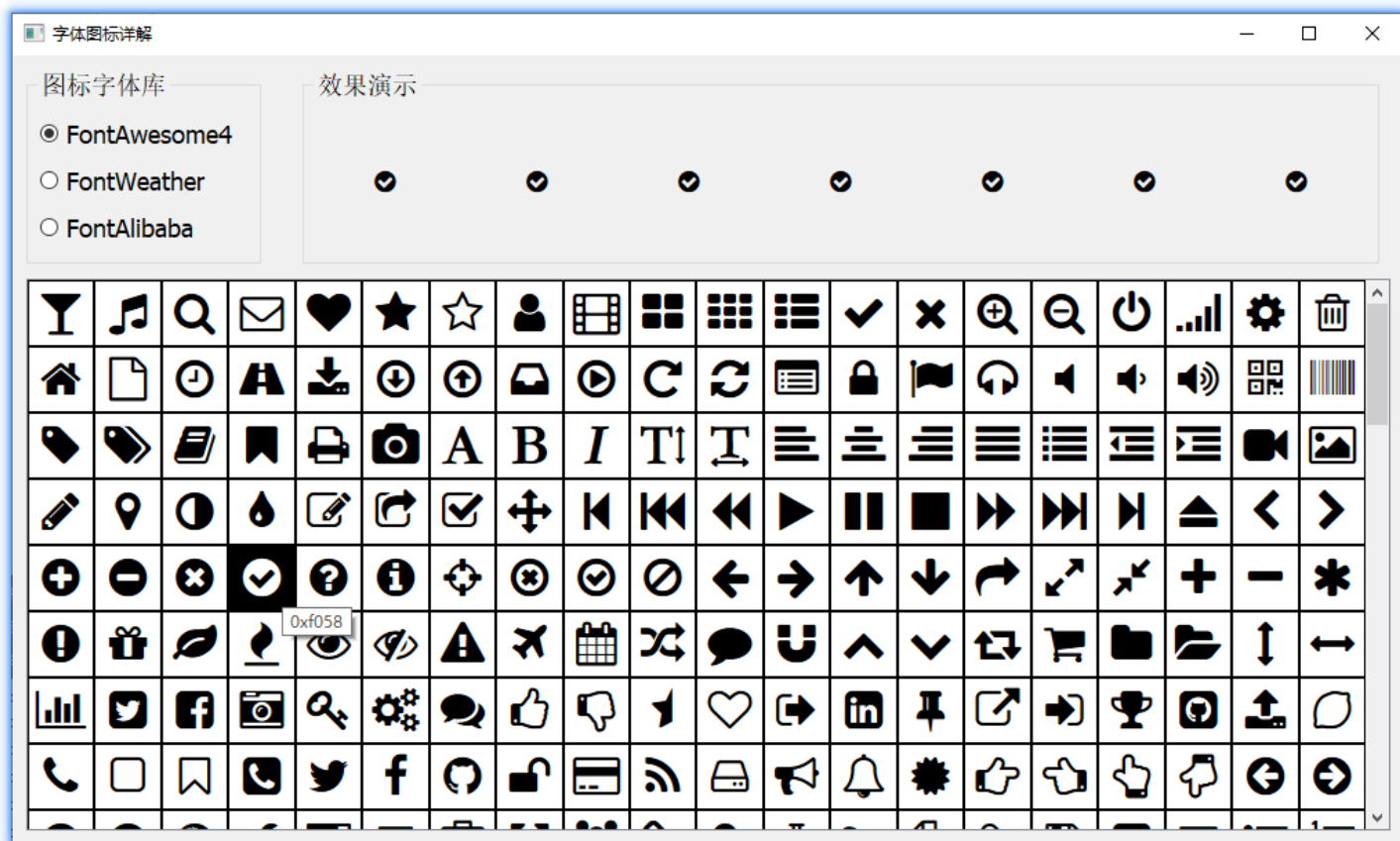
```

1 void Widget::radioClicked()
2 {
3     // ...
4     for ( int i = 0; i < icons.size(); i++ ) {
5         // ...
6         lbl->installEventFilter(this);
7     }
8 }

```

此时，当鼠标在标签上悬浮时，就可以在顶部的 "效果演示" 区域显示图标了，如下：





问题：在 `eventFilter()` 中，已经将显示区标签图标设置为 92 了，为什么显示还那么小呢？

```
1 IconHelper::instance()->setIcon(lblList[i], iconFontName, icon, 92);
```

原因：因为在 `widget.cpp` 的构造中，将 `showGroup` 的字体大小设置为了 `18px`

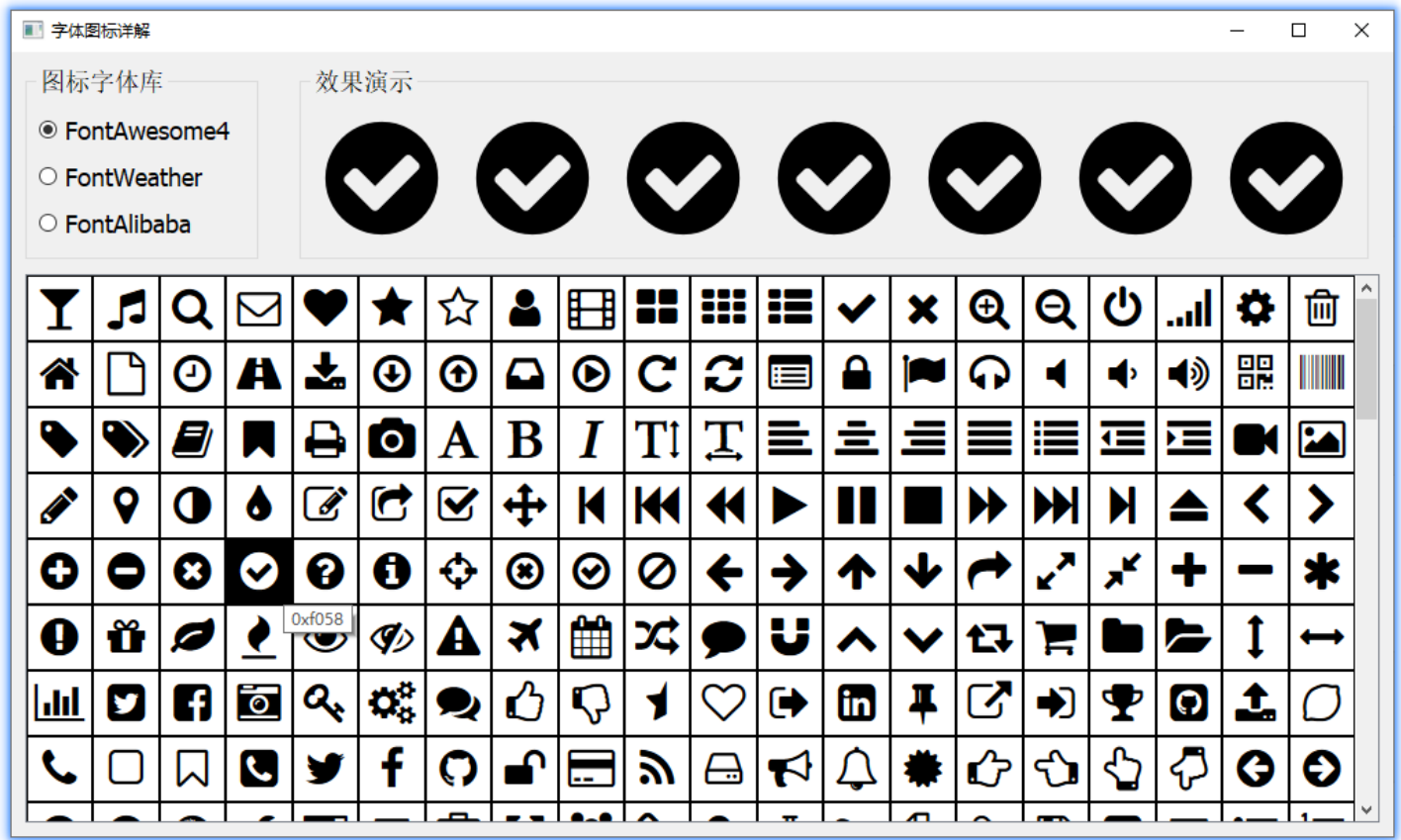
```
1 showGroup->setStyleSheet("font: 18px");
```

而显示区的 7 个标签，都在 `showGroup` 中，也就继承了这个样式

解决方式：使用 `id` 选择器。只修改 `showGroup` 的样式，而不修改其中子控件的样式，如下：

```
1 showGroup->setObjectName("showGroup");
2 showGroup->setStyleSheet("QGroupBox#showGroup {font: 18px}");
```

此时，重新运行，效果正常：



### 10.1.3 七色显示

为了"七色显示", 来到 `widget.cpp` 构造中, 为每个标签设置样式表, 如下:

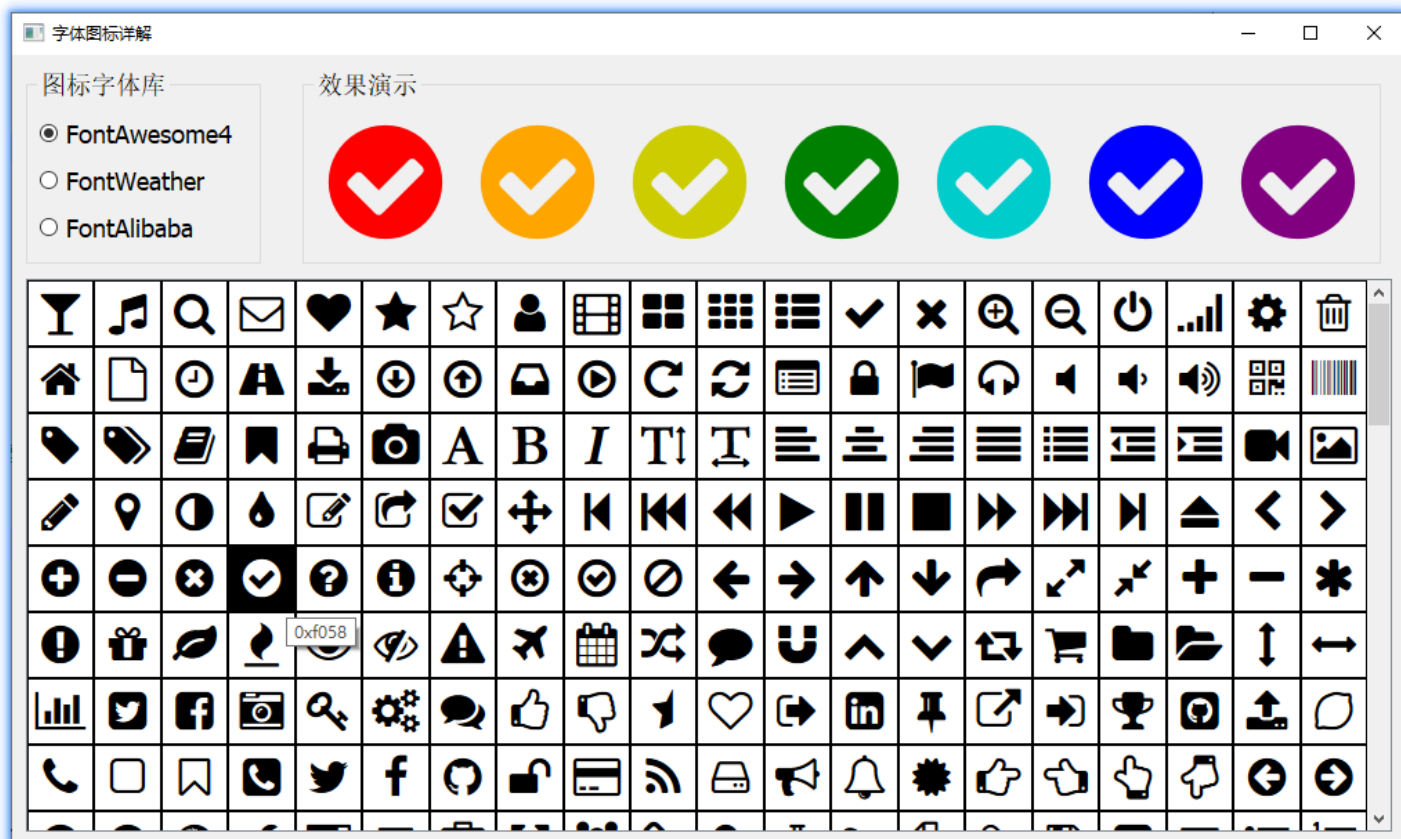
```
1 Widget::Widget(QWidget* parent) : QWidget(parent)
2 {
3     for ( int i = 0; i < 7; i++ ) {
4         // ...
5         if ( i == 0 ) {
6             lbl->setStyleSheet("color: red");
7         } else if ( i == 1 ) {
8             lbl->setStyleSheet("color: orange");
9         } else if ( i == 2 ) {
10            //             lbl->setStyleSheet("color: yellow");
11            lbl->setStyleSheet("color: #cccc00");
12        } else if ( i == 3 ) {
13            lbl->setStyleSheet("color: green");
14        } else if ( i == 4 ) {
15            //             lbl->setStyleSheet("color: cyan");
16            lbl->setStyleSheet("color: #00cccc");
17        } else if ( i == 5 ) {
18            lbl->setStyleSheet("color: blue");
```

```

19         } else if ( i == 6 ) {
20             lbl->setStyleSheet("color: purple");
21         }
22     }
23 }

```

此时，运行效果如下：

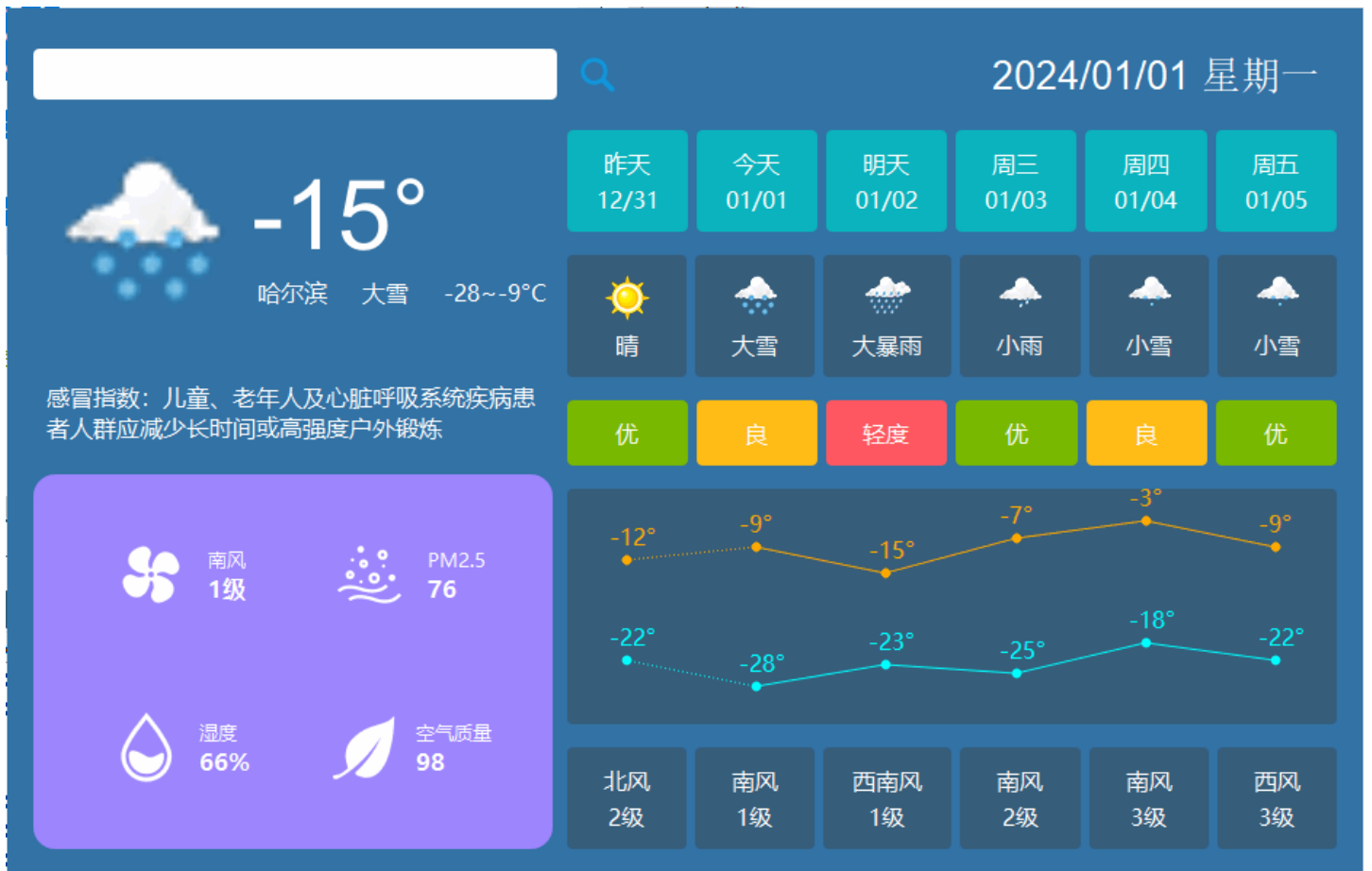


## 第三章 天气预报UI（纯代码写布局）

本章会手写一个天气预报的界面。

### 1. 整体效果、技术点

本章实现的天气预报项目，整体效果如下：



## 完整项目的效果是：

在左上角输入要查询的城市，然后点击查询按钮，就会发送 `HTTP` 请求给服务器，请求回来的天气数据 `JSON` 格式

通过解析 `JSON` 可以获取以下信息：

- 今天的信息

温度、湿度、风向、风力、天气类型（晴、多云、小雨等）、`PM2.5`、温馨提示、感冒指数、日出日落

- 未来15天的信息

日期、星期、天气类型（晴、多云、小雨等）、`PM25`、最高温、最低温

需要观看完整项目的小伙伴，请移步我的 B 站（[明王讲Qt](#)）观看完整视频！

[【QT开发专题-天气预报】1. 效果演示、技术分析\\_哔哩哔哩\\_bilibili](#)

而这里，我们专注于手写整个界面，不涉及 `JSON` 解析以及 `HTTP` 通信。本章涉及的技术点如下：

## 1.1 纯代码手写布局

为什么要手写布局呢？

- 实际工作中，基本就是手写布局，很少用设计师界面
- 使用设计师界面，拖拽控件完成布局后，最终也要转化成 `C++` 代码
- 往往，布局是动态的，不是一成不变的，就无法提前拖放控件来完成布局
- 手写布局，能够更深入理解布局，便于更精细地调整

## 1.2 样式表的设置

合理地使用样式表，可以使界面更加美观，这里设置的样式表如下：

- **背景图片**  
为整个窗体设置一张背景图片
- **背景色**  
设置控件背景透明，或者设置一个透明度
- **圆角**  
为控件设置圆角
- **字体颜色和大小**  
为控件设置合适的字体颜色和字体大小

## 1.3 事件

为了界面的美观，我们将窗口设置为无标题栏，这样就无法通过右上角的【关闭】按钮，退出程序。

因此增加了右键菜单退出的功能

还重写了鼠标移动事件，让窗口可以跟随鼠标移动

## 1.4 绘图

绘制高低温曲线，根据每天高低温数据，可以绘制一个曲线，更直观地展示温度变化趋势

- QPainter
- QChart
- QCustomPlot

## 1.5 资源文件

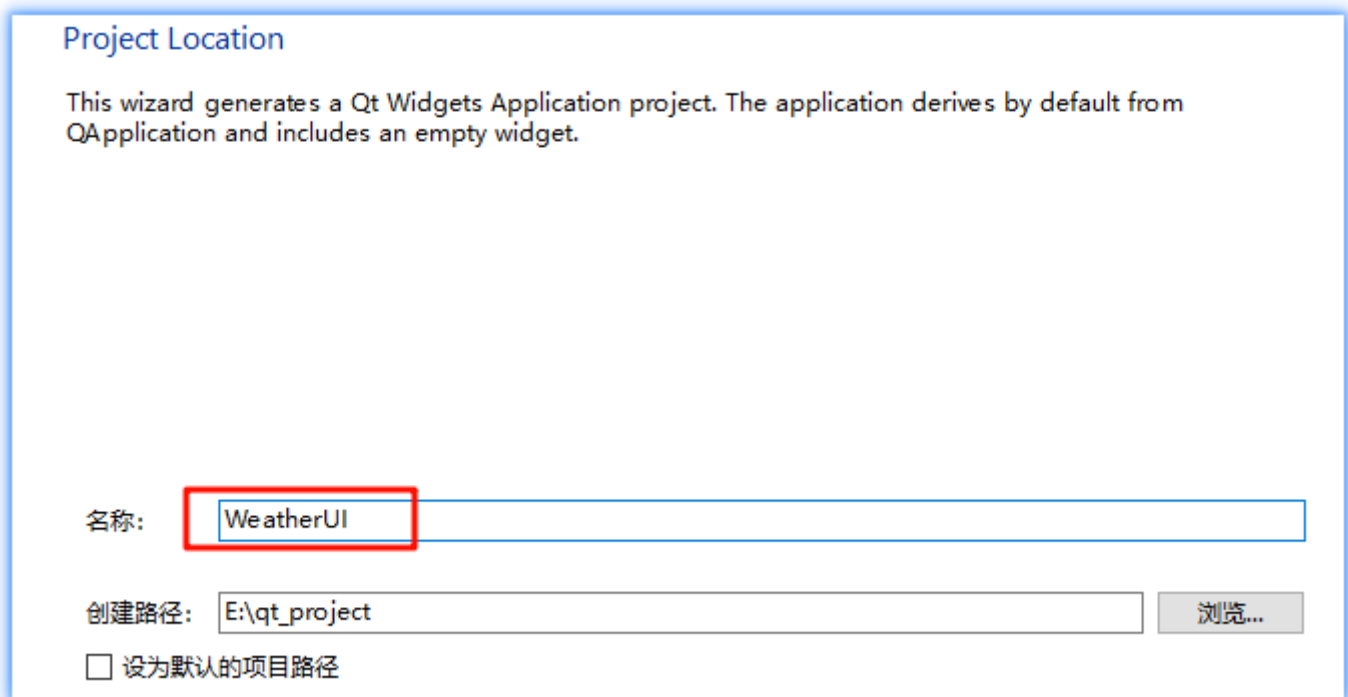
根据不同的天气类型，还可以用不同的图标进行展示，更加直观

而这些图标通常会放到资源文件中，这样它们可以一同被打包进 Qt 的可执行程序中

## 2. 新建项目、添加资源图标

### 2.1 新建项目

首先创建一个名为 WeatherUI 的项目，如下：



The image shows the 'Project Location' dialog box in Qt. It contains the following elements:

- Title:** Project Location
- Description:** This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.
- Name:** A text field containing 'WeatherUI', which is highlighted with a red rectangle.
- Create path:** A text field containing 'E:\qt\_project'.
- Buttons:** A '浏览...' (Browse...) button next to the 'Create path' field.
- Checkbox:** An unchecked checkbox labeled '设为默认的项目路径' (Set as default project path).

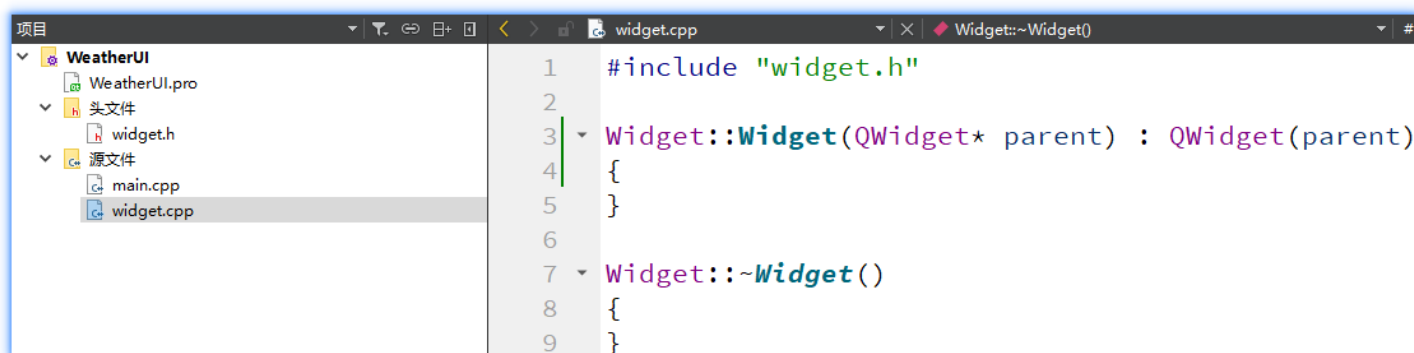
然后，基类选择 `QWidget`，不要勾选 “`Generate form`”，因为我们不使用设计师界面，而用纯代码来实现整个界面，如下：

## Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name:	<input type="text" value="Widget"/>
Base class:	<input type="text" value="QWidget"/>
Header file:	<input type="text" value="widget.h"/>
Source file:	<input type="text" value="widget.cpp"/>
<input type="checkbox"/> Generate form	取消勾选
Form file:	<input type="text" value="widget.ui"/>

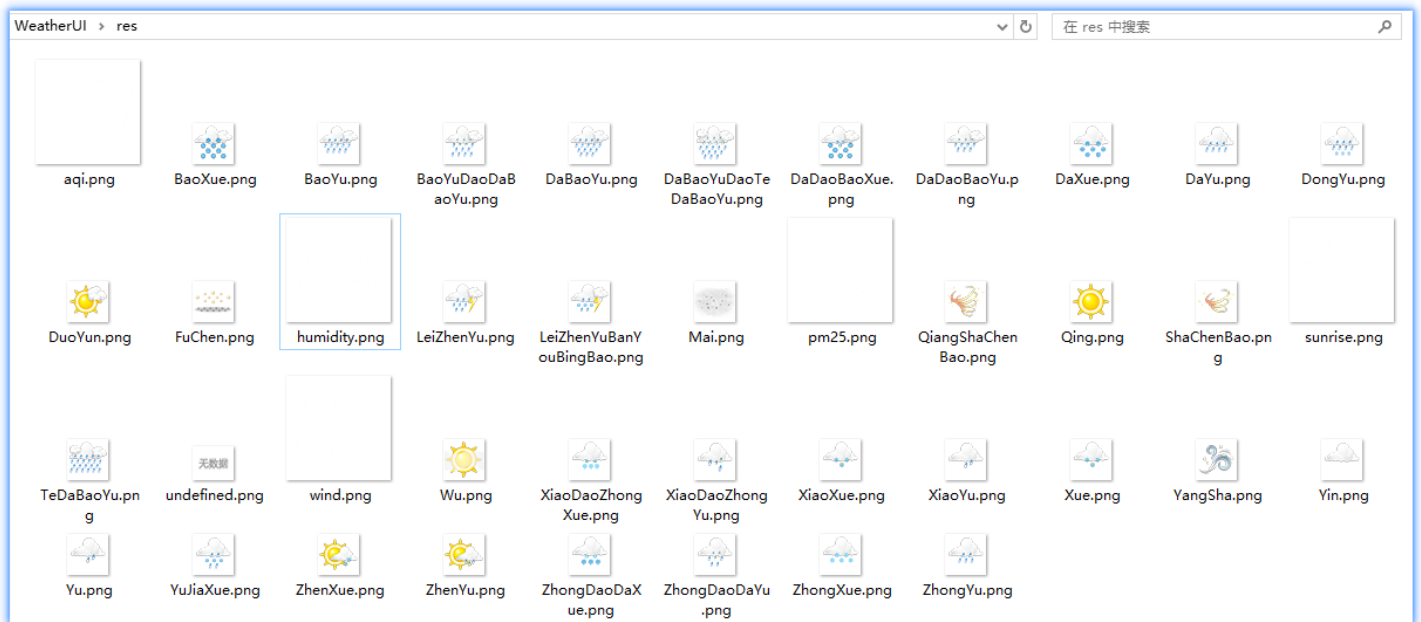
最后，创建完成的后的项目目录如下：



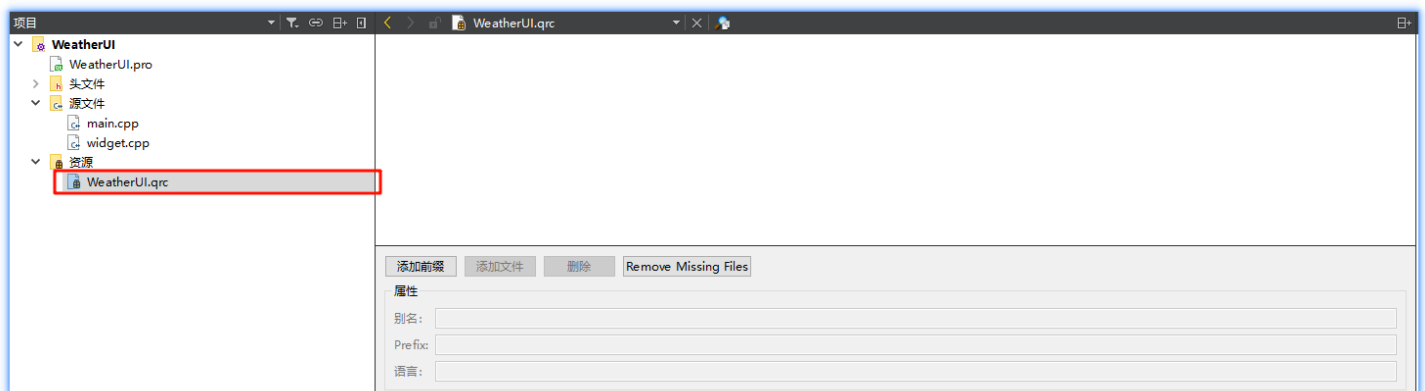
## 2.2 添加资源文件

界面上会用到“风雨雷电”等图标。因此，这里首先创建一个资源文件，把这些图标添加到工程中来

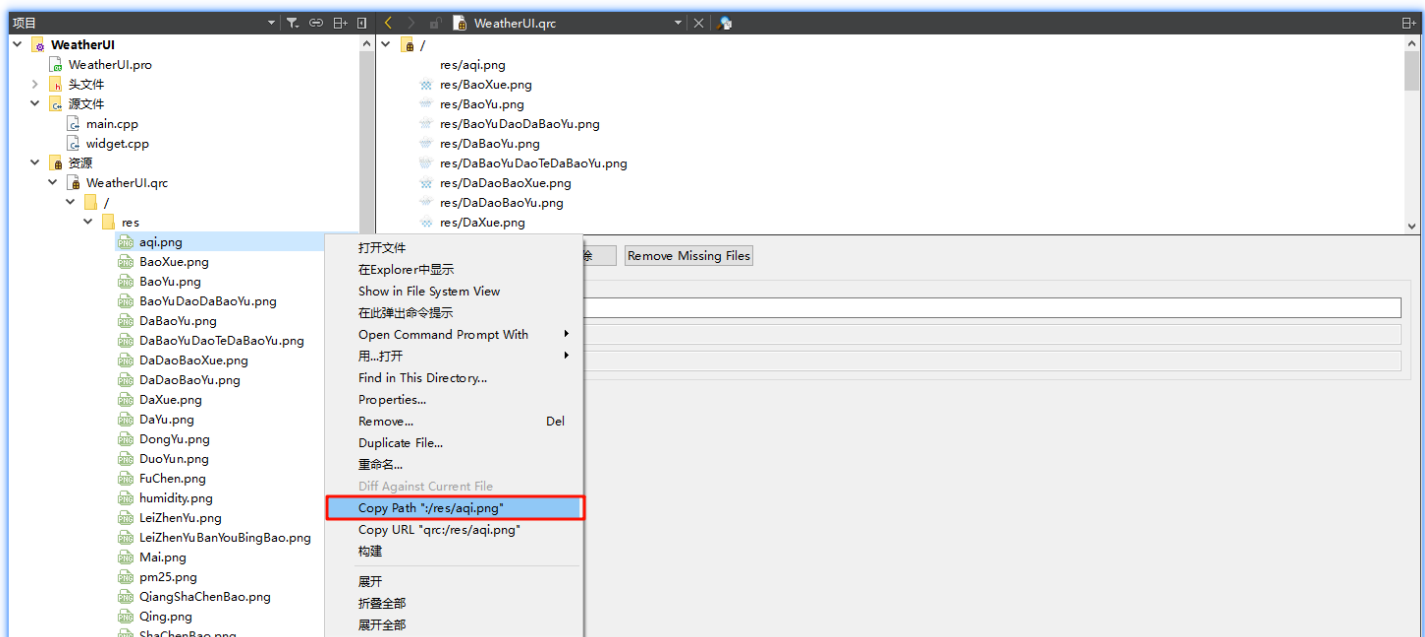
首先，在项目根目录下创建一个 `res` 目录，并把用到的天气图标拷贝进来，如下：



然后，在项目中添加一个名为 `WeatherUI.qrc` 的资源文件，如下：



最后，点击”添加前缀“和”添加文件“，把图标添加进来，最终效果如下：





之后，在图标文件上右键拷贝路径，就可以直接在代码中使用图标了，比如 `(":/res/aqi.png"`

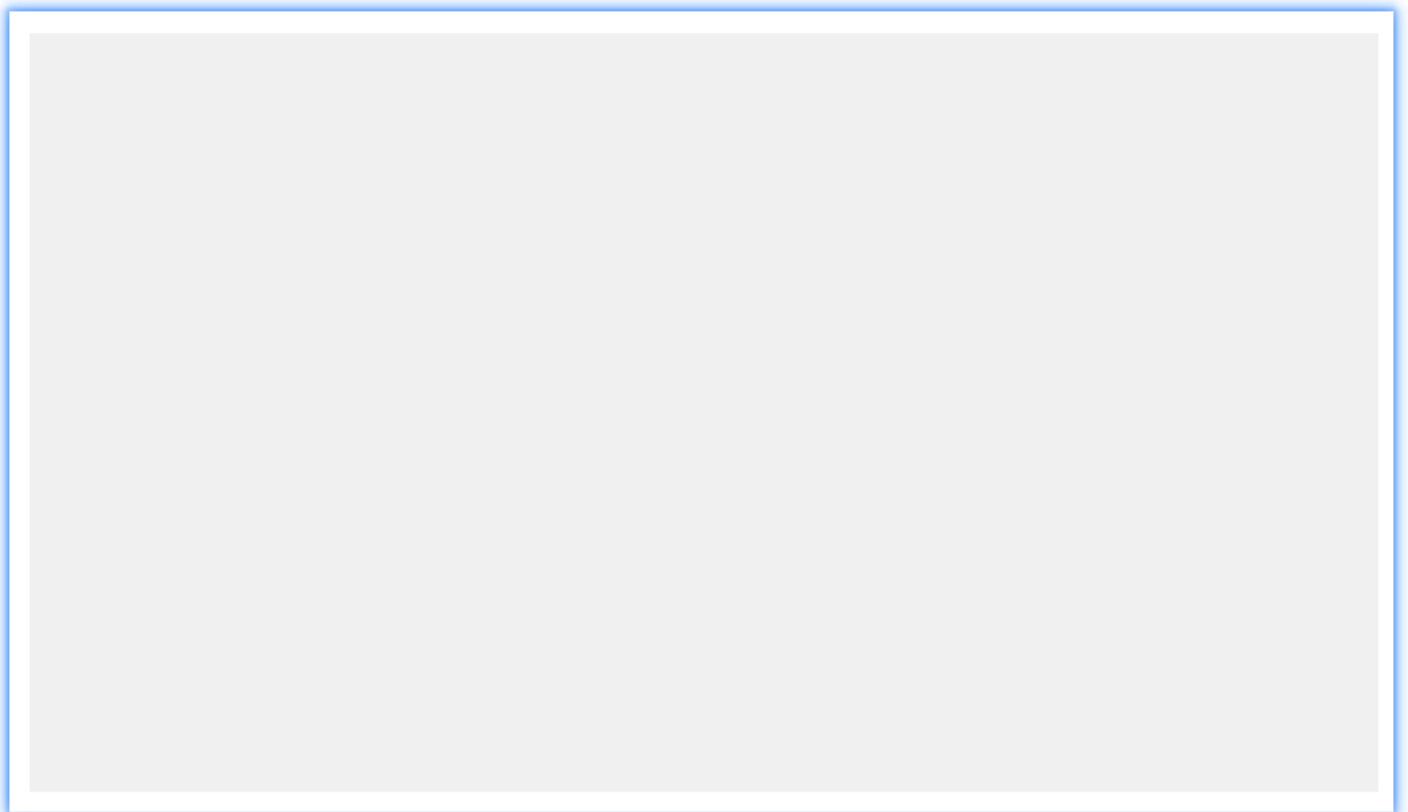
## 3. 设置无边框、右键退出

### 3.1 设置窗口固定大小，并设置无边框

来到 `widget.cpp`，在构造中添加如下代码：

```
1 Widget::Widget(QWidget* parent) : QWidget(parent)
2 {
3     // 无边框
4     setWindowFlag(Qt::FramelessWindowHint);
5 }
```

此时运行效果如下：



### 3.2 右键退出

由于没有了边框，就无法点击右上角的  退出程序，因此添加右键菜单的退出功能  
要实现右键菜单，只需重写父类的 `contextMenuEvent` 虚函数即可：

```
1 void QWidget::contextMenuEvent(QContextMenuEvent *event)
```

具体步骤为：

首先，在 `widget.h` 头文件中，声明该虚函数，并添加菜单

```
1 #include <QMenu>
2
3 class Widget : public QWidget
4 {
5 protected:
6     void contextMenuEvent(QContextMenuEvent* event);
7
8 private:
9     QMenu* mExitMenu;    // 右键退出的菜单
10    QAction* mExitAct;    // 退出的行为
11 };
```

然后，来到 `widget.cpp` 源文件，做如下实现：

```
1 #include <QApplication>
2 #include <QContextMenuEvent>
3
4 Widget::Widget(QWidget* parent) : QWidget(parent)
5 {
6     // 右键菜单：退出程序
7     mExitMenu = new QMenu(this);
8     mExitAct = new QAction();
9     mExitAct->setText(tr("退出"));
10    mExitAct->setIcon(QIcon(":/res/close.ico"));
11    mExitMenu->addAction(mExitAct);
12
13    connect(mExitAct, &QAction::triggered, this, [=]() { qApp->exit(0); });
14 }
15
16 // 重写父类的虚函数
```

```

17 // 父类中默认的实现是忽略右键菜单事件
18 // 重写之后，处理右键菜单事件
19 void Widget::contextMenuEvent(QContextMenuEvent* event)
20 {
21     mExitMenu->exec(QCursor::pos());
22
23     // 调用 accept 表示，这个事件我已经处理，不需要向上传递了
24     event->accept();
25 }

```

此时，在窗口上右键，就可以弹出右键菜单，并且点击其中的退出菜单项可以退出程序了。

## 4. 窗口随鼠标移动

由于窗口无边框，无法通过标题栏来移动窗口。但是可以重写父类的鼠标按下、鼠标移动事件，来实现窗口随鼠标移动

首先，在 `widget.h` 声明两个鼠标事件

```

1 class Widget : public QWidget
2 {
3 protected:
4     void mousePressEvent(QMouseEvent* event);
5     void mouseMoveEvent(QMouseEvent* event);
6
7 private:
8     QPoint mOffset; // 窗口移动时，鼠标与窗口左上角的偏移
9 };

```

然后，在 `widget.cpp` 中，实现如下：

```

1 void Widget::mousePressEvent(QMouseEvent* event)
2 {
3     qDebug() << "窗口左上角：" << this->pos() << "，鼠标坐标点：" << event-
        >globalPos();
4     mOffset = event->globalPos() - this->pos();
5 }
6

```

```
7 void Widget::mouseMoveEvent(QMouseEvent* event)
8 {
9     this->move(event->globalPos() - mOffset);
10 }
```

### 程序说明：

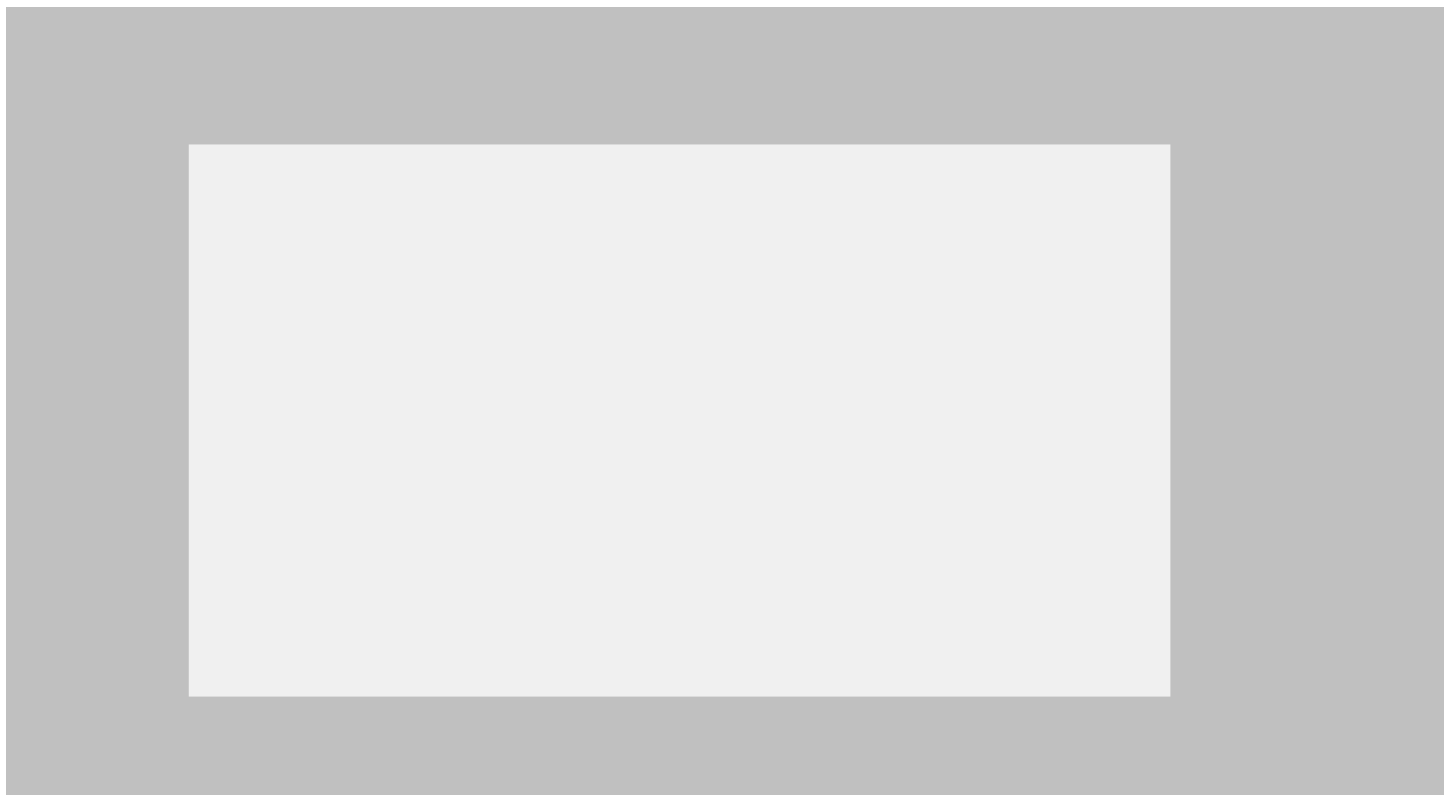


`move()` 函数移动窗口时，其实是设置窗口左上角的位置

因此需要定义了一个全局变量 `mOffset`，它记录了鼠标位置与窗口左上角的偏移

在移动过程中，设置的窗体位置时，需要用鼠标当前位置减去偏移量 `mOffset`

经过以上，按下鼠标左键就可以移动窗口了，如下：



## 5. 顶部界面

### 5.1 设置窗体背景

来到 `widget.cpp` 的构造中，设置窗体背景，以及标签的样式，如下：

```
1 Widget::Widget(QWidget* parent) : QWidget(parent)
2 {
3     // ...
4
5     this->setObjectName("weather_widget");
6     this->setStyleSheet(R"(
7         QWidget#weather_widget {
8             background-color: rgba(50,115,165,255);
9         }
10        QLabel {
11            font: 12pt "Microsoft YaHei";
12            border-radius: 4px;
13            color: rgb(255, 255, 255);
14            padding: 12px;
15        }
16    )");
17 }
```

此时，运行效果如下：



## 5.2 整体布局规划

整体采用垂直布局

- 顶部布局

- 底部布局（采用水平布局）
  - 左侧布局
  - 右侧布局



首先，来到 `widget.h` 中，声明几个成员变量，如下：

```
1 #include <QVBoxLayout>
2 #include <QHBoxLayout>
3 #include <QLabel>
4
5 class Widget : public QWidget
6 {
7 private:
8     void initTop();
9
10 private:
11     //////////////// 主体
12     QVBoxLayout* mainLayout;
13
14     //////////////// 顶部
15     QHBoxLayout* topLayout;
```

```

16     QLabel* lblDate; // 日期
17
18     //////////// 左侧
19     QVBoxLayout* leftLayout;
20
21     //////////// 右侧
22     QVBoxLayout* rightLayout;
23 };

```

接下来，来到 widget.cpp 的构造中，完成布局之间的嵌套关系，如下：

```

1 Widget::Widget(QWidget* parent) : QWidget(parent)
2 {
3     // ...
4
5     // 1. 创建主布局
6     mainLayout = new QVBoxLayout(this);
7     mainLayout->setSpacing(10);
8     mainLayout->setContentsMargins(18, 18, 18, 18);
9
10    // 2. 顶部布局
11    topLayout = new QHBoxLayout();
12
13    // 3. 底部布局
14    QHBoxLayout* bottomLayout = new QHBoxLayout();
15    leftLayout = new QVBoxLayout();
16    leftLayout->setContentsMargins(0, 0, 0, 0);
17    rightLayout = new QVBoxLayout();
18    rightLayout->setSpacing(16); // 设置一个间距
19
20    bottomLayout->addLayout(leftLayout);
21    bottomLayout->addLayout(rightLayout);
22    bottomLayout->setStretch(0, 2);
23    bottomLayout->setStretch(1, 4);
24
25    // 4. 将顶部和底部布局，添加到主布局
26    mainLayout->addLayout(topLayout);
27    mainLayout->addLayout(bottomLayout);
28
29    // ...
30 }

```

## 5.3 完成顶部布局

首先，来到 widget.h 中，声明 `initTop()` 成员函数：

```
1 class Widget : public QWidget
2 {
3 private:
4     void initTop();
5 }
```

然后，来到 widget.cpp 中，实现 `initTop()` 函数：

```
1 void Widget::initTop()
2 {
3     // 1. 城市输入框
4     QLineEdit* leCity = new QLineEdit(this);
5     leCity->setFixedWidth(360);
6     leCity->setStyleSheet(R"(
7         font: 14pt Microsoft YaHei;
8         background-color: rgb(255, 255, 255);
9         border-radius: 4px;
10        padding: 4px 8px
11    )");
12
13    // 2. 搜索按钮
14    QPushButton* btnSearch = new QPushButton(this);
15    btnSearch->setStyleSheet("background-color: rgba(255, 255, 255,0);");
16    btnSearch->setIcon(QIcon(":/res/search.png"));
17    btnSearch->setIconSize(QSize(24, 24));
18
19    // 3. 弹簧
20    QSpacerItem* space = new QSpacerItem(32, 32, QSizePolicy::Expanding,
21        QSizePolicy::Minimum);
22
23    // 4. 日期
24    lblDate = new QLabel(this);
25    lblDate->setStyleSheet(R"(
26        font: 20pt Arial;
27        background-color: rgba(255, 255, 255,0);
28    )");
29    lblDate->setAlignment(Qt::AlignCenter);
```



```
29     lblDate->setText("2024/01/12 星期五");
30
31     topLayout->addWidget(leCity);
32     topLayout->addWidget(btnSearch);
33     topLayout->addItem(space);
34     topLayout->addWidget(lblDate);
35 }
```

最后，来到 widget.cpp 的构造中，调用 `initTop()` 函数：

```
1 Widget::Widget(QWidget* parent) : QWidget(parent)
2 {
3     // ...
4
5     initTop();
6
7     // ...
8 }
```

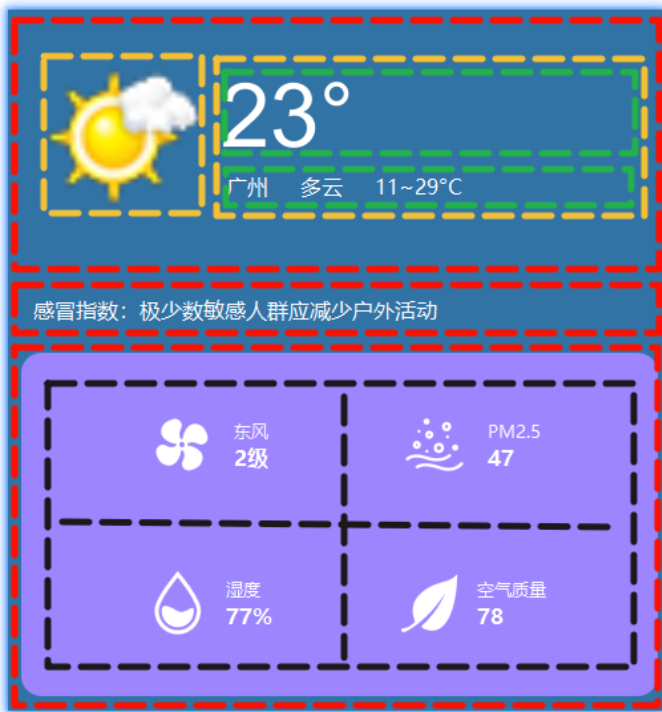
此时，运行程序，效果如下：



## 6. 左侧界面

左侧界面整体采用垂直布局：

- 天气/温度/城市（采用水平布局）
  - 天气类型
  - 温度/城市（垂直布局）
    - 当天温度（水平布局）



- 城市、天气类型、高低温（水平布局）
- 感冒指数
- 风力/PM2.5/湿度/空气质量（栅格布局）
  - 风力（水平布局）
    - 图标
    - 东风、2级别（垂直布局）
      - 东风
      - 二级
  - PM2.5（同上）
  - 湿度（同上）
  - 空气质量（同上）

首先，来到 `widget.h` 中，声明左侧界面相关的成员变量和成员函数 `initLeft()`，如下：

```
1
2 class WeatherWidget : public QWidget
3 {
4 private:
5     void initLeft();
6
7 private:
8     //////////// 左侧
9     QVBoxLayout* leftLayout;
10    QLabel* lblTypeIcon; // 天气图标
11    QLabel* lblTemp;     // 温度
12    QLabel* lblCity;     // 城市
13    QLabel* lblType;     // 天气
14    QLabel* lblLowHigh;  // 高低温
15
16    QLabel* lblGanMao;   // 感冒指数
17
18    QLabel* lblFlIcon;   // 风力图标
19    QLabel* lblFx;       // 风向
20    QLabel* lblFl;       // 风力
21
22    QLabel* lblPM25Icon; // PM25图标
23    QLabel* lblPM25Title; // PM25标题
24    QLabel* lblPM25;     // PM25值
25
```

```

26     QLabel* lblHumidityIcon;    // 湿度图标
27     QLabel* lblHumidityTitle;   // 湿度标题
28     QLabel* lblHumidity;        // 湿度值
29
30     QLabel* lblQualityIcon;     // 空气质量图标
31     QLabel* lblQualityTitle;    // 空气质量标题
32     QLabel* lblQuality;         // 空气质量值
33 };

```

然后，来到 `widget.cpp` 中，实现 `initLeft()` 函数：

```

1 void Widget::initLeft()
2 {
3     // 1. 天气/温度/城市
4     QHBoxLayout* layout = new QHBoxLayout();
5     layout->setSpacing(0);
6
7     // 1.1 天气类型
8     lblTypeIcon = new QLabel(this);
9     lblTypeIcon->setFixedSize(150, 150);
10    lblTypeIcon->setStyleSheet("background-color: rgba(255, 255, 255, 0);");
11    lblTypeIcon->setPixmap(QPixmap(":/res/DuoYun.png"));
12    lblTypeIcon->setScaledContents(true);
13    lblTypeIcon->setAlignment(Qt::AlignCenter);
14    layout->addWidget(lblTypeIcon);
15
16    QVBoxLayout* vLayout = new QVBoxLayout();
17    QHBoxLayout* hLayout1 = new QHBoxLayout();
18    QHBoxLayout* hLayout2 = new QHBoxLayout();
19
20    // 1.2
21    // 温度
22    lblTemp = new QLabel(this);
23    lblTemp->setText("28°");
24    lblTemp->setStyleSheet(R"(
25        font: 50pt "Arial";
26        background-color: rgba(255, 255, 255, 0);
27        padding: 0px;
28    )");
29    lblTemp->setAlignment(Qt::AlignBottom);
30    lblTypeIcon->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
31    hLayout1->addWidget(lblTemp);
32
33    // 城市

```

```
34     lblCity = new QLabel(this);
35     lblCity->setText("北京");
36     lblCity->setStyleSheet(R"(
37         background-color: rgba(255, 255, 255,0);
38         font: 12pt "Microsoft YaHei";
39         padding: 0px 0px 24px 0px;
40     )");
41     lblCity->setAlignment(Qt::AlignTop);
42     lblCity->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
43     hLayout2->addWidget(lblCity);
44
45     // 弹簧
46     QSpacerItem* item2 = new QSpacerItem(16, 1, QSizePolicy::Fixed,
47     QSizePolicy::Minimum);
48
49     // 类型
50     lblType = new QLabel(this);
51     lblType->setText("晴转多云");
52     lblType->setStyleSheet(R"(
53         background-color: rgba(255, 255, 255,0);
54         font: 12pt "Microsoft YaHei";
55         padding: 0px 0px 24px 0px;
56     )");
57     lblType->setAlignment(Qt::AlignTop);
58     lblType->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
59
60     hLayout2->addWidget(lblType);
61
62     // 弹簧
63     QSpacerItem* item3 = new QSpacerItem(16, 1, QSizePolicy::Fixed,
64     QSizePolicy::Minimum);
65
66     // 高低温
67     lblLowHigh = new QLabel(this);
68     lblLowHigh->setText("19~31°");
69     lblLowHigh->setStyleSheet(R"(
70         background-color: rgba(255, 255, 255,0);
71         font: 12pt "Microsoft YaHei";
72         padding: 0px 0px 24px 0px;
73     )");
74     lblLowHigh->setAlignment(Qt::AlignLeft | Qt::AlignVCenter);
75     lblLowHigh->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
76     hLayout2->addWidget(lblLowHigh);
77
78     vLayout->addLayout(hLayout1);
```

```
79     vLayout->addLayout(hLayout2);
80     layout->addLayout(vLayout);
81     leftLayout->addLayout(layout);
82
83     // 2. 弹簧
84     QSpacerItem* spaceItem2 = new QSpacerItem(1, 1, QSizePolicy::Fixed,
QSizePolicy::Expanding);
85     leftLayout->addItem(spaceItem2);
86
87     // 3. 感冒指数
88     lblGanMao = new QLabel(this);
89     lblGanMao->setText("感冒指数：儿童、老年人及心脏、呼吸系统疾病患者人群应减少长时间
或高强度户外锻炼");
90     lblGanMao->setStyleSheet(R"(
91         background-color: rgba(60, 60, 60, 0);
92         padding-left: 5px;
93         padding-right: 5px;
94         font: 12pt "Microsoft YaHei"
95     )");
96     lblGanMao->setWordWrap(true);
97     leftLayout->addWidget(lblGanMao);
98
99     // 4. 风力/PM2.5/湿度/空气质量
100    QWidget* widget = new QWidget(this);
101    widget->setStyleSheet(R"(
102        background-color: rgb(157, 133, 255);
103        border-radius: 15px
104    )");
105    QGridLayout* gridLayout = new QGridLayout(widget);
106    gridLayout->setHorizontalSpacing(40);
107    gridLayout->setVerticalSpacing(30);
108    gridLayout->setContentsMargins(40, 25, 40, 25);
109
110    // 4.1 风力
111    QHBoxLayout* hItem0 = new QHBoxLayout();
112    hItem0->setSpacing(0);
113
114    lblFlIcon = new QLabel(this);
115    lblFlIcon->setFixedSize(72, 72);
116    lblFlIcon->setStyleSheet("background-color: rgba(255, 255, 255, 0);");
117    lblFlIcon->setPixmap(QPixmap(":/res/wind.png"));
118    lblFlIcon->setScaledContents(true);
119    lblFlIcon->setAlignment(Qt::AlignCenter);
120
121    hItem0->addWidget(lblFlIcon);
122
123    QVBoxLayout* vItem0 = new QVBoxLayout();
```

```
124     lblFx = new QLabel(this);
125     lblFx->setText("西北风");
126     lblFx->setStyleSheet(R"(
127         background-color: rgba(255, 255, 255,0);
128         font: 10pt "Microsoft YaHei";
129         padding: 24px 0px 0px 0px;
130     )");
131     lblFx->setAlignment(Qt::AlignCenter);
132     lblFx->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
133     vItem0->addWidget(lblFx);
134
135     lblFl = new QLabel(this);
136     lblFl->setText("3级");
137     lblFl->setStyleSheet(R"(
138         background-color: rgba(255, 255, 255,0);
139         font: bold 12pt "Microsoft YaHei";
140         padding: 0px 0px 24px 0px;
141     )");
142     lblFl->setAlignment(Qt::AlignCenter);
143     lblFl->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
144     vItem0->addWidget(lblFl);
145
146     hItem0->addLayout(vItem0);
147     gridLayout->addLayout(hItem0, 0, 0, 1, 1);
148
149     // 4.2 PM2.5
150     QHBoxLayout* hItem1 = new QHBoxLayout();
151     hItem1->setSpacing(0);
152
153     lblPM25Icon = new QLabel(this);
154     lblPM25Icon->setFixedSize(72, 72);
155     lblPM25Icon->setStyleSheet("background-color: rgba(255, 255, 255, 0);");
156     lblPM25Icon->setPixmap(QPixmap(":/res/pm25.png"));
157     lblPM25Icon->setScaledContents(true);
158     lblPM25Icon->setAlignment(Qt::AlignCenter);
159
160     hItem1->addWidget(lblPM25Icon);
161
162     QVBoxLayout* vItem1 = new QVBoxLayout();
163     lblPM25Title = new QLabel(this);
164     lblPM25Title->setText("PM2.5");
165     lblPM25Title->setStyleSheet(R"(
166         background-color: rgba(255, 255, 255,0);
167         font: 10pt "Microsoft YaHei";
168         padding: 24px 0px 0px 0px;
169     )");
170     lblPM25Title->setAlignment(Qt::AlignCenter);
```

```
171     lblPM25Title->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
172     vItem1->addWidget(lblPM25Title);
173
174     lblPM25 = new QLabel(this);
175     lblPM25->setText("10");
176     lblPM25->setStyleSheet(R"(
177         background-color: rgba(255, 255, 255,0);
178         font: bold 12pt "Microsoft YaHei";
179         padding: 0px 0px 24px 0px;
180     )");
181     lblPM25->setAlignment(Qt::AlignCenter);
182     lblPM25->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
183     vItem1->addWidget(lblPM25);
184
185     hItem1->addLayout(vItem1);
186     gridLayout->addLayout(hItem1, 0, 1, 1, 1);
187
188     // 4.3 湿度
189     QHBoxLayout* hItem2 = new QHBoxLayout();
190     hItem2->setSpacing(0);
191
192     lblHumidityIcon = new QLabel(this);
193     lblHumidityIcon->setFixedSize(72, 72);
194     lblHumidityIcon->setStyleSheet("background-color: rgba(255, 255, 255,
195     0);");
196     lblHumidityIcon->setPixmap(QPixmap(":/res/humidity.png"));
197     lblHumidityIcon->setScaledContents(true);
198     lblHumidityIcon->setAlignment(Qt::AlignCenter);
199
200     hItem2->addWidget(lblHumidityIcon);
201
202     QVBoxLayout* vItem2 = new QVBoxLayout();
203     lblHumidityTitle = new QLabel(this);
204     lblHumidityTitle->setText("湿度");
205     lblHumidityTitle->setStyleSheet(R"(
206         background-color: rgba(255, 255, 255,0);
207         font: 10pt "Microsoft YaHei";
208         padding: 24px 0px 0px 0px;
209     )");
210     lblHumidityTitle->setAlignment(Qt::AlignCenter);
211     lblHumidityTitle->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
212     vItem2->addWidget(lblHumidityTitle);
213
214     lblHumidity = new QLabel(this);
215     lblHumidity->setText("60%");
216     lblHumidity->setStyleSheet(R"(
217         background-color: rgba(255, 255, 255,0);
```

```
217         font: bold 12pt "Microsoft YaHei";
218         padding: 0px 0px 24px 0px;
219     )");
220     lblHumidity->setAlignment(Qt::AlignCenter);
221     lblHumidity->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
222     vItem2->addWidget(lblHumidity);
223
224     hItem2->addLayout(vItem2);
225     gridLayout->addLayout(hItem2, 1, 0, 1, 1);
226
227     // 4.4 空气质量
228     QHBoxLayout* hItem3 = new QHBoxLayout();
229     hItem3->setSpacing(0);
230
231     lblQualityIcon = new QLabel(this);
232     lblQualityIcon->setFixedSize(72, 72);
233     lblQualityIcon->setStyleSheet("background-color: rgba(255, 255, 255, 0);");
234     lblQualityIcon->setPixmap(QPixmap(":/res/aqi.png"));
235     lblQualityIcon->setScaledContents(true);
236     lblQualityIcon->setAlignment(Qt::AlignCenter);
237
238     hItem3->addWidget(lblQualityIcon);
239
240     QVBoxLayout* vItem3 = new QVBoxLayout();
241     lblQualityTitle = new QLabel(this);
242     lblQualityTitle->setText("空气质量");
243     lblQualityTitle->setStyleSheet(R"(
244         background-color: rgba(255, 255, 255,0);
245         font: 10pt "Microsoft YaHei";
246         padding: 24px 0px 0px 0px;
247     )");
248     lblQualityTitle->setAlignment(Qt::AlignCenter);
249     lblQualityTitle->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
250     vItem3->addWidget(lblQualityTitle);
251
252     lblQuality = new QLabel(this);
253     lblQuality->setText("60%");
254     lblQuality->setStyleSheet(R"(
255         background-color: rgba(255, 255, 255,0);
256         font: bold 12pt "Microsoft YaHei";
257         padding: 0px 0px 24px 0px;
258     )");
259     lblQuality->setAlignment(Qt::AlignCenter);
260     lblQuality->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
261     vItem3->addWidget(lblQuality);
262
263     hItem3->addLayout(vItem3);
```



```
264     gridLayout->addLayout(hItem3, 1, 1, 1, 1);
265
266     leftLayout->addWidget(widget);
267 }
```

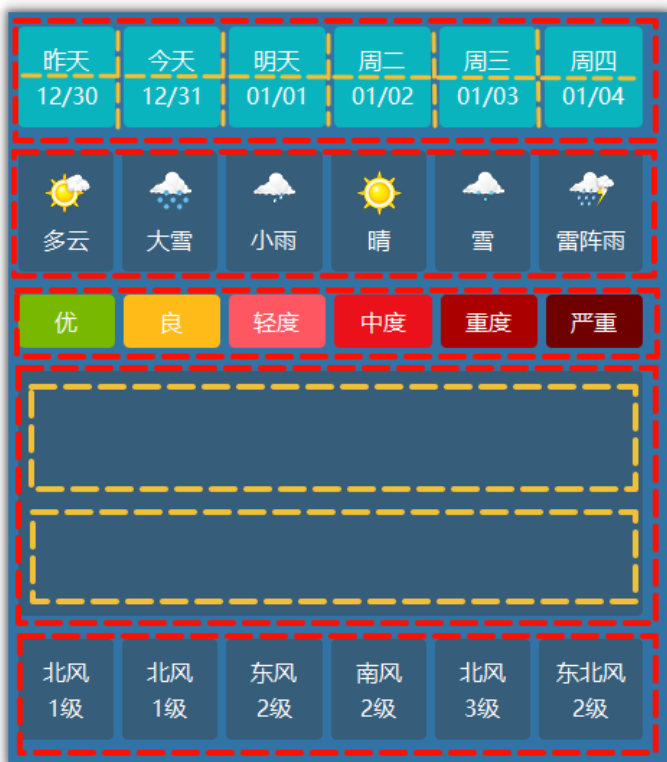
此时，运行效果如下：



## 7. 右侧界面

右侧侧界面整体采用垂直布局：

- 星期、日期（栅格布局）
  - 星期



- 日期
- 天气类型（栅格布局）
  - 图标
  - 描述
- 空气质量（水平布局）
- 温度（垂直布局）
  - 高温曲线
  - 低温曲线
- 风力（栅格布局）
  - 标题
  - 风力级别

首先，来到 `widget.h` 中，声明左侧界面相关的成员变量和成员函数 `initRight()`，如下：

```

1
2 class WeatherWidget : public QWidget
3 {
4 private:
5     void initRight();
6
7 private:
8     //////////// 右侧
9     QVBoxLayout* rightLayout;
10    QList<QLabel*> mWeekList; // 星期
11    QList<QLabel*> mDateList; // 日期
12
13    QList<QLabel*> mTypeList; // 天气
14    QList<QLabel*> mTypeIconList; // 天气图标
15    QMap<QString, QString> mTypeMap; // 天气类型的map
16
17    QList<QLabel*> mAqiList; // 天气指数
18
19    QLabel* lblHigh; // 高温
20    QLabel* lblLow; // 低温
21
22    QList<QLabel*> mFxList; // 风向
23    QList<QLabel*> mFlList; // 风力
24 };

```

然后，来到 `widget.cpp` 中，实现 `initRight()` 函数：

```
1 void Widget::initRight()
2 {
3     // 1. 星期/日期
4     QGridLayout* gridLayout1 = new QGridLayout();
5     gridLayout1->setHorizontalSpacing(6);
6     gridLayout1->setVerticalSpacing(0);
7
8     QStringList weekList = {"昨天", "今天", "明天", "周二", "周三", "周四"};
9     QStringList dateList = {"12/30", "12/31", "01/01", "01/02", "01/03",
10 "01/04"};
11     for ( int i = 0; i < weekList.size(); i++ ) {
12         QLabel* lblWeek = new QLabel(this);
13         QLabel* lblDate = new QLabel(this);
14         lblWeek->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
15         lblDate->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
16         lblWeek->setText(weekList[i]);
17         lblDate->setText(dateList[i]);
18         lblWeek->setStyleSheet(R"(
19             border-bottom-left-radius: 0px;
20             border-bottom-right-radius: 0px;
21             padding-bottom: 2px;
22             padding-left:20px;
23             padding-right:20px;
24             background-color: rgb(10, 180, 190);
25         )");
26         lblDate->setStyleSheet(R"(
27             border-top-left-radius: 0px;
28             border-top-right-radius: 0px;
29             padding-top: 2px;
30             padding-left:20px;
31             padding-right:20px;
32             background-color: rgb(10, 180, 190);
33         )");
34         lblWeek->setAlignment(Qt::AlignCenter);
35         lblDate->setAlignment(Qt::AlignCenter);
36
37         mWeekList << lblWeek;
38         mDateList << lblDate;
39
40         gridLayout1->addWidget(lblWeek, 0, i);
41         gridLayout1->addWidget(lblDate, 1, i);
```

```

41     }
42
43     rightLayout->addLayout(gridLayout1);
44
45     // 2. 天气类型
46     QGridLayout* gridLayout2 = new QGridLayout();
47     gridLayout2->setHorizontalSpacing(6);
48     gridLayout2->setVerticalSpacing(0);
49
50     QMap<QString, QString> typeMap;
51     typeMap.insert("大雪", ":/res/DaXue.png");
52     typeMap.insert("晴", ":/res/Qing.png");
53     typeMap.insert("小雨", ":/res/XiaoYu.png");
54     typeMap.insert("雷阵雨", ":/res/LeiZhenYu.png");
55     typeMap.insert("雪", ":/res/Xue.png");
56     typeMap.insert("多云", ":/res/DuoYun.png");
57
58     QList<QString> keys = typeMap.keys();
59     for ( int i = 0; i < keys.size(); i++ ) {
60         QLabel* lblTypeIcon = new QLabel(this);
61         QLabel* lblType = new QLabel(this);
62         lblTypeIcon->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
63         lblType->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
64         lblTypeIcon->setPixmap(typeMap[keys[i]]);
65         lblType->setText(keys[i]);
66         lblTypeIcon->setStyleSheet(R"(
67             border-bottom-left-radius: 0px;
68             border-bottom-right-radius: 0px;
69             padding-bottom: 2px;
70             padding-left:20px;
71             padding-right:20px;
72             background-color: rgb(54, 93, 122);
73         )");
74         lblType->setStyleSheet(R"(
75             border-top-left-radius: 0px;
76             border-top-right-radius: 0px;
77             padding-top: 2px;
78             padding-left:20px;
79             padding-right:20px;
80             background-color: rgb(54, 93, 122);
81         )");
82         lblTypeIcon->setAlignment(Qt::AlignCenter);
83         lblType->setAlignment(Qt::AlignCenter);
84
85         mTypeList << lblType;
86         mTypeIconList << lblTypeIcon;
87

```

```

88     gridLayout2->addWidget(lblTypeIcon, 0, i);
89     gridLayout2->addWidget(lblType, 1, i);
90 }
91 rightLayout->addLayout(gridLayout2);
92
93 // 3. 空气质量
94 QWidget* qualityWidget = new QWidget(this);
95 qualityWidget->setStyleSheet("background-color:rgba(51,115,163,255);");
96 qualityWidget->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
97 // 需要加这一行, 否则缩放时, qualityWidget也会被缩放
98 QHBoxLayout* hLayout = new QHBoxLayout(qualityWidget);
99 hLayout->setSpacing(6);
100 hLayout->setContentsMargins(0, 0, 0, 0);
101 QStringList qualityList = {"优", "良", "轻度", "中度", "重度", "严重"};
102 QStringList qualityBgList;
103 qualityBgList << "rgb(121, 184, 0)"
104                 << "rgb(255, 187, 23)"
105                 << "rgb(255, 87, 97)"
106                 << "rgb(235, 17, 27)"
107                 << "rgb(170, 0, 0)"
108                 << "rgb(110, 0, 0)";
109 for ( int i = 0; i < qualityList.size(); i++ ) {
110     QLabel* lblQuality = new QLabel(this);
111     lblQuality->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
112     lblQuality->setText(qualityList[i]);
113     lblQuality->setStyleSheet(QString("padding:8px;background-
114 color:%1;").arg(qualityBgList[i]));
115     lblQuality->setAlignment(Qt::AlignCenter);
116
117     mAqiList << lblQuality;
118
119     hLayout->addWidget(lblQuality);
120 }
121
122 rightLayout->addWidget(qualityWidget);
123
124 // 4. 温度曲线
125 QVBoxLayout* vLayout = new QVBoxLayout();
126 vLayout->setSpacing(0);
127
128 lblHigh = new QLabel(this);
129 lblHigh->setMinimumHeight(80);
130 lblHigh->setStyleSheet(R"(
131     border-bottom-left-radius: 0px;
132     border-bottom-right-radius: 0px;
133     padding-bottom: 2px;
134     padding-left:20px;

```

```

133         padding-right:20px;
134         background-color: rgba(54, 93, 122,255);
135     )");
136     lblLow = new QLabel(this);
137     lblLow->setMinimumHeight(80);
138     lblLow->setStyleSheet(R"(
139         border-top-left-radius: 0px;
140         border-top-right-radius: 0px;
141         padding-top: 2px;
142         padding-left:20px;
143         padding-right:20px;
144         background-color: rgba(54, 93, 122,255);
145     )");
146     vLayout->addWidget(lblHigh);
147     vLayout->addWidget(lblLow);
148     rightLayout->addLayout(vLayout);
149
150     // 5. 风力
151     QGridLayout* gridLayout3 = new QGridLayout();
152     gridLayout3->setHorizontalSpacing(6);
153     gridLayout3->setVerticalSpacing(0);
154
155     QStringList fxList = {"北风", "北风", "东风", "南风", "北风", "东北风"};
156     QStringList flList = {"1级", "1级", "2级", "2级", "3级", "2级"};
157     for ( int i = 0; i < fxList.size(); i++ ) {
158         QLabel* lblFx = new QLabel(this);
159         QLabel* lblFl = new QLabel(this);
160         lblFx->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
161         lblFl->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
162         lblFx->setText(fxList[i]);
163         lblFl->setText(flList[i]);
164         lblFx->setStyleSheet(R"(
165             border-bottom-left-radius: 0px;
166             border-bottom-right-radius: 0px;
167             padding-bottom: 2px;
168             padding-left:20px;
169             padding-right:20px;
170             background-color: rgb(54, 93, 122);
171         )");
172         lblFl->setStyleSheet(R"(
173             border-top-left-radius: 0px;
174             border-top-right-radius: 0px;
175             padding-top: 2px;
176             padding-left:20px;
177             padding-right:20px;
178             background-color: rgb(54, 93, 122);
179         )");

```

```
180         lblFx->setAlignment(Qt::AlignCenter);
181         lblFl->setAlignment(Qt::AlignCenter);
182
183         mFxList << lblFx;
184         mFlList << lblFl;
185
186         gridLayout3->addWidget(lblFx, 0, i);
187         gridLayout3->addWidget(lblFl, 1, i);
188     }
189
190     rightLayout->addLayout(gridLayout3);
191 }
```

此时，运行效果如下：



## 8. 更新界面

更新界面要实现的效果是：

每隔 3 秒，自动切换界面显示，哈尔滨天气 -> 北京天气 -> 杭州天气 -> 广州天气



## 8.1 定义成员变量、成员函数

首先，来到 `widget.h` 中，定义一个结构体，用于描述所有的天气信息：

```
1 struct WeatherInfo {
2     QString city;
3     QString dateWeek;
4     qint8 temp;
5
6     QString ganMao;
7     qint8 pm25;
8     qint8 humidity;
9     QString quality;
10
11     QList<QString> weekList;
12     QList<QString> dateList;
13     QList<QString> typeList;
14     QList<qint16> qualityList;
15     QList<qint8> highTemp;
16     QList<qint8> lowTemp;
17     QList<QString> fx;
```



```
18     QList<qint8> fl;  
19 };
```

然后，来到 `widget.h` 中，定义相关的成员变量和成员函数：

```
1 class Widget : public QWidget  
2 {  
3 private:  
4     // ...  
5     void initData(); // 初始化数据  
6     void updateUI(); // 更新UI界面  
7  
8 private:  
9     // ...  
10    // 定义一个天气列表，用于模拟多个城市  
11    QList<WeatherInfo> WeatherInfoList;  
12    qint8 cityIndex;  
13 };
```

最后，使用 Alt + Enter 快捷键，快速生成几个成员函数的空实现。

## 8.2 初始化数据

初始化 4 个城市的天气数据，进而模拟 4 个城市切换。

首先，来到 `widget.cpp` 中，实现 `initData()` 函数：

```
1 void Widget::initData()  
2 {  
3     // 1. 初始化天气类型，以及对应的图标  
4     mTypeMap.insert("暴雪", ":/res/weather/BaoXue.png");  
5     mTypeMap.insert("暴雨", ":/res/weather/BaoYu.png");  
6     mTypeMap.insert("暴雨到大暴雨", ":/res/weather/BaoYuDaoDaBaoYu.png");  
7     mTypeMap.insert("大暴雨", ":/res/weather/DaBaoYu.png");  
8     mTypeMap.insert("大暴雨到特大暴雨", ":/res/weather/DaBaoYuDaoTeDaBaoYu.png");  
9     mTypeMap.insert("大到暴雪", ":/res/weather/DaDaoBaoXue.png");  
10    mTypeMap.insert("大雪", ":/res/weather/DaXue.png");  
11    mTypeMap.insert("大雨", ":/res/weather/DaYu.png");
```

```

12     mTypeMap.insert("冻雨", ":/res/weather/DongYu.png");
13     mTypeMap.insert("多云", ":/res/weather/DuoYun.png");
14     mTypeMap.insert("浮沉", ":/res/weather/FuChen.png");
15     mTypeMap.insert("雷阵雨", ":/res/weather/LeiZhenYu.png");
16     mTypeMap.insert("雷阵雨伴有冰雹",
    ":/res/weather/LeiZhenYuBanYouBingBao.png");
17     mTypeMap.insert("霾", ":/res/weather/Mai.png");
18     mTypeMap.insert("强沙尘暴", ":/res/weather/QiangShaChenBao.png");
19     mTypeMap.insert("晴", ":/res/weather/Qing.png");
20     mTypeMap.insert("沙尘暴", ":/res/weather/ShaChenBao.png");
21     mTypeMap.insert("特大暴雨", ":/res/weather/TeDaBaoYu.png");
22     mTypeMap.insert("undefined", ":/res/weather/undefined.png");
23     mTypeMap.insert("雾", ":/res/weather/Wu.png");
24     mTypeMap.insert("小到中雪", ":/res/weather/XiaoDaoZhongXue.png");
25     mTypeMap.insert("小到中雨", ":/res/weather/XiaoDaoZhongYu.png");
26     mTypeMap.insert("小雪", ":/res/weather/XiaoXue.png");
27     mTypeMap.insert("小雨", ":/res/weather/XiaoYu.png");
28     mTypeMap.insert("雪", ":/res/weather/Xue.png");
29     mTypeMap.insert("扬沙", ":/res/weather/YangSha.png");
30     mTypeMap.insert("阴", ":/res/weather/Yin.png");
31     mTypeMap.insert("雨", ":/res/weather/Yu.png");
32     mTypeMap.insert("雨夹雪", ":/res/weather/YuJiaXue.png");
33     mTypeMap.insert("阵雪", ":/res/weather/ZhenXue.png");
34     mTypeMap.insert("阵雨", ":/res/weather/ZhenYu.png");
35     mTypeMap.insert("中到大雪", ":/res/weather/ZhongDaoDaXue.png");
36     mTypeMap.insert("中到大雨", ":/res/weather/ZhongDaoDaYu.png");
37     mTypeMap.insert("中雪", ":/res/weather/ZhongXue.png");
38     mTypeMap.insert("中雨", ":/res/weather/ZhongYu.png");
39
40     // 2. 初始化4个城市天气信息
41     // 哈尔滨
42     WeatherInfo harbin;
43     harbin.city = "哈尔滨";
44     harbin.dateWeek = "2024/01/01 星期一";
45     harbin.temp = -15;
46     harbin.ganMao = "儿童、老年人及心脏呼吸系统疾病患者人群应减少长时间或高强度户外锻
    炼";
47     harbin.pm25 = 76;
48     harbin.humidity = 66;
49     harbin.quality = "轻度";
50     harbin.weekList = {"周日", "周一", "周二", "周三", "周四", "周五"};
51     harbin.dateList = {"12/31", "01/01", "01/02", "01/03", "01/04", "01/05"};
52     harbin.typeList = {"晴", "大雪", "大暴雨", "小雨", "小雪", "小雪"};
53     harbin.qualityList = {23, 98, 150, 45, 98, 33};
54     harbin.highTemp = {-12, -9, -15, -7, -3, -9};
55     harbin.lowTemp = {-22, -28, -23, -25, -18, -22};
56     harbin.fx = {"北风", "南风", "西南风", "南风", "南风", "西风"};

```

```
57     harbin.fl = {2, 1, 1, 2, 3, 3};
58
59     // 北京
60     WeatherInfo beijing;
61     beijing.city = "北京";
62     beijing.dateWeek = "2024/01/01 星期一";
63     beijing.temp = -2;
64     beijing.ganMao = "儿童、老年人及心脏呼吸系统疾病患者人群应减少长时间或高强度户外锻
炼";
65     beijing.pm25 = 92;
66     beijing.humidity = 55;
67     beijing.quality = "轻度";
68     beijing.weekList = {"周日", "周一", "周二", "周三", "周四", "周五"};
69     beijing.dateList = {"12/31", "01/01", "01/02", "01/03", "01/04", "01/05"};
70     beijing.typeList = {"晴", "晴", "多云", "晴", "多云", "小雨"};
71     beijing.qualityList = {12, 45, 156, 88, 23, 9};
72     beijing.highTemp = {2, 6, 12, 8, 5, 10};
73     beijing.lowTemp = {-5, -9, -6, -12, -5, -1};
74     beijing.fx = {"北风", "北风", "西北风", "西北风", "西北风", "西北风"};
75     beijing.fl = {2, 1, 2, 2, 2, 2};
76
77     // 上海
78     WeatherInfo shanghai;
79     shanghai.city = "上海";
80     shanghai.dateWeek = "2024/01/01 星期一";
81     shanghai.temp = 6;
82     shanghai.ganMao = "极少数敏感人群应减少户外活动";
83     shanghai.pm25 = 74;
84     shanghai.humidity = 57;
85     shanghai.quality = "良";
86     shanghai.weekList = {"周日", "周一", "周二", "周三", "周四", "周五"};
87     shanghai.dateList = {"12/31", "01/01", "01/02", "01/03", "01/04", "01/05"};
88     shanghai.typeList = {"霾", "大暴雨", "小雪", "晴", "晴", "晴"};
89     shanghai.qualityList = {12, 65, 12, 23, 34, 155};
90     shanghai.highTemp = {10, 15, 12, 8, 11, 14};
91     shanghai.lowTemp = {5, 8, 4, 12, 6, 11};
92     shanghai.fx = {"北风", "北风", "东北风", "西北风", "东南风", "北风"};
93     shanghai.fl = {3, 2, 2, 3, 2, 2};
94
95     // 广州
96     WeatherInfo guangzhou;
97     guangzhou.city = "广州";
98     guangzhou.dateWeek = "2024/01/01 星期一";
99     guangzhou.temp = 23;
100    guangzhou.ganMao = "极少数敏感人群应减少户外活动";
101    guangzhou.pm25 = 47;
102    guangzhou.humidity = 77;
```

```

103     guangzhou.quality = "良";
104     guangzhou.weekList = {"周日", "周一", "周二", "周三", "周四", "周五"};
105     guangzhou.dateList = {"12/31", "01/01", "01/02", "01/03", "01/04",
        "01/05"};
106     guangzhou.typeList = {"霾", "多云", "小雨", "晴", "晴", "晴"};
107     guangzhou.qualityList = {23, 78, 33, 45, 66, 155};
108     guangzhou.highTemp = {25, 27, 21, 18, 24, 20};
109     guangzhou.lowTemp = {14, 11, 18, 9, 12, 16};
110     guangzhou.fx = {"北风", "东风", "南风", "北风", "东风", "东南风"};
111     guangzhou.fl = {1, 2, 2, 3, 2, 1};
112
113     WeatherInfoList.append(harbin);
114     WeatherInfoList.append(beijing);
115     WeatherInfoList.append(shanghai);
116     WeatherInfoList.append(guangzhou);
117 }

```

然后，来到 `widget.cpp` 的构造中，调用一下 `initData()` 函数：

```

1 Widget::Widget(QWidget* parent) : QWidget(parent)
2 {
3     // ...
4
5     initData();
6 }

```

## 8.3 定时切换城市

接下来实现，每隔 3 秒切换城市，更新界面数据的功能

首先，来到 `widget.cpp` 构造中，创建并启动定时器：

```

1 #include <QTimer>
2
3 Widget::Widget(QWidget* parent) : QWidget(parent)
4 {
5     // ...
6
7     QTimer* timer = new QTimer();

```

```
8     connect(timer, &QTimer::timeout, this, &Widget::updateUI);
9     cityIndex = 0;
10    timer->start(2000);
11 }
```

然后，来到 `widget.cpp` 中，完成 `updateUI()` 函数：

```
1 void Widget::updateUI()
2 {
3     cityIndex++;
4     if ( cityIndex > 3 ) {
5         cityIndex = 0;
6     }
7
8     WeatherInfo info = WeatherInfoList[cityIndex];
9
10    // 1. 更新日期
11    lblDate->setText(info.dateWeek);
12
13    // 2. 更新天气类型、城市、高低温
14    lblTypeIcon->setPixmap(mTypeMap[info.typeList[1]]);
15    lblTemp->setText(QString::number(info.temp) + "°");
16    lblCity->setText(info.city);
17    lblType->setText(info.typeList[1]);
18    lblLowHigh->setText(QString::number(info.lowTemp[1]) + "~" +
19    QString::number(info.highTemp[1]) + "°C");
20
21    lblGanMao->setText("感冒指数: " + info.ganMao);
22
23    lblFx->setText(info.fx[1]);
24    lblFl->setText(QString::number(info.fl[1]) + "级");
25
26    lblPM25->setText(QString::number(info.pm25));
27
28    lblHumidity->setText(QString::number(info.humidity) + "%");
29
30    lblQuality->setText(QString::number(info.qualityList[1]));
31
32    // 3. 更新六天
33    for ( int i = 0; i < 6; i++ ) {
34        // 3.1 更新星期和日期
35        mWeekList[i]->setText(info.weekList[i]);
36
37        //设置 昨天/今天/明天 的星期显示 - 不显示星期几，而是显示
```

```

37 //“昨天”、“今天”、“明天”
38 mWeekList[0]->setText("昨天");
39 mWeekList[1]->setText("今天");
40 mWeekList[2]->setText("明天");
41
42 mDateList[i]->setText(info.dateList[i]);
43
44 // 3.2 更新天气类型
45 mTypeIconList[i]->setPixmap(mTypeMap[info.typeList[i]]);
46 mTypeList[i]->setText(info.typeList[i]);
47
48 // 3.3 更新空气质量
49 if ( info.qualityList[i] >= 0 && info.qualityList[i] <= 50 ) {
50     mAqiList[i]->setText("优");
51     mAqiList[i]->setStyleSheet("background-color: rgb(121, 184, 0);");
52 } else if ( info.qualityList[i] >= 50 && info.qualityList[i] <= 100 ) {
53     mAqiList[i]->setText("良");
54     mAqiList[i]->setStyleSheet("background-color: rgb(255, 187, 23);");
55 } else if ( info.qualityList[i] >= 100 && info.qualityList[i] <= 150 )
56 {
57     mAqiList[i]->setText("轻度");
58     mAqiList[i]->setStyleSheet("background-color: rgb(255, 87, 97);");
59 } else if ( info.qualityList[i] >= 150 && info.qualityList[i] <= 200 )
60 {
61     mAqiList[i]->setText("中度");
62     mAqiList[i]->setStyleSheet("background-color: rgb(235, 17, 27);");
63 } else if ( info.qualityList[i] >= 200 && info.qualityList[i] <= 300 )
64 {
65     mAqiList[i]->setText("重度");
66     mAqiList[i]->setStyleSheet("background-color: rgb(170, 0, 0);");
67 } else {
68     mAqiList[i]->setText("严重");
69     mAqiList[i]->setStyleSheet("background-color: rgb(110, 0, 0);");
70 }
71
72 // 3.4 更新风力/风向
73 mFxList[i]->setText(info.fx[i]);
74 mFlList[i]->setText(QString::number(info.fl[i]) + "级");
75
76 // 4. 绘制高低温曲线
77 lblHigh->update();
78 lblLow->update();
79 }

```

此时，就可以完成城市切换，如下：



## 9. 绘制高低温曲线

### 9.1 事件过滤器

以上还没有绘制高低温曲线，接下来实现

首先，来到 `widget.cpp` 的构造中，为高低温的标签 `lblHighCurve` 和 `lblLowCurve` 安装事件过滤器

```
1 void Widget::initRight()
2 {
3     // ...
4     lblHigh->installEventFilter(this);
5     lblLow->installEventFilter(this);
6 }
```

然后，来到 `widget.h` 中声明 `eventFilter()`，并声明两个成员函数：

- `paintHighCurve()` - 绘制高温曲线
- `paintLowCurve()` - 绘制低温曲线

```
1 class Widget : public QWidget
2 {
3 protected:
4     // 事件过滤器：过滤标签的 update 事件
5     bool eventFilter(QObject* watched, QEvent* event);
6
7 private:
8     // ...
9     // 绘制高低温曲线
10    void paintHighCurve();
11    void paintLowCurve();
12 };
```

最后，在 `widget.cpp` 中实现：

```
1 bool WeatherWidget::eventFilter(QObject* watched, QEvent* event)
2 {
3     if ( watched == lblHigh && event->type() == QEvent::Paint ) {
4         paintHighCurve();
5     }
6     if ( watched == lblLow && event->type() == QEvent::Paint ) {
7         paintLowCurve();
8     }
9     return QWidget::eventFilter(watched, event);
10 }
```

这样，在 `updateUI()` 函数的最后，调用 `QLabel` 的 `update()` 函数，就可以实现曲线的绘制

```
1 void Widget::updateUI()
2 {
3     // ...
4
5     // 4. 绘制高低温曲线
6     lblHigh->update();
```



```
7     lblLow->update();
8 }
```

具体流程：

- 调用标签的 `update()` 函数
- 框架发送 `QEvent::Paint` 事件 给标签
- 事件被当前窗口拦截，进而调用其 `eventFilter()` 函数
- 在 `eventFilter()` 中，调用 `paintHighCurve()` 和 `paintLowCurve()` 来真正绘制曲线

## 9.2 绘制温度曲线

以下是绘制高温曲线的代码：

```
1  // 温度曲线相关的宏
2  #define INCREMENT      3    // 温度每升高/降低1度，y轴坐标的增量
3  #define POINT_RADIUS   3    // 曲线描点的大小
4  #define TEXT_OFFSET_X  12   // 温度文本相对于点的偏移
5  #define TEXT_OFFSET_Y  10   // 温度文本相对于点的偏移
6
7  void Widget::paintHighCurve()
8  {
9      WeatherInfo info = WeatherInfoList[cityIndex];
10
11      QPainter painter(lblHigh);
12
13      // 抗锯齿
14      painter.setRenderHint(QPainter::Antialiasing, true);
15
16      // 1. 获取 x 轴坐标
17      int pointX[6] = {0};
18      for ( int i = 0; i < 6; i++ ) {
19          pointX[i] = mAqiList[i]->pos().x() + mAqiList[i]->width() / 2;
20      }
21
22      // 2. 获取 y 轴坐标
23
24      // int temp[6] = {0};
25      int tempSum = 0;
26      int tempAverage = 0;
27
28      // 2.1 计算平均值
```

```

29     for ( int i = 0; i < 6; i++ ) {
30         tempSum += info.highTemp[i];
31     }
32
33     tempAverage = tempSum / 6; // 最高温平均值
34
35     //    qDebug() << "paintHighCurve" << tempAverage;
36
37     // 2.2 计算 y 轴坐标
38     int pointY[6] = {0};
39     int yCenter = lblHigh->height() / 2;
40     for ( int i = 0; i < 6; i++ ) {
41         pointY[i] = yCenter - ((info.highTemp[i] - tempAverage) * INCREMENT);
42     }
43
44     // 3. 开始绘制
45     // 3.1 初始化画笔
46     QPen pen = painter.pen();
47     pen.setWidth(1); //设置画笔宽度为1
48     pen.setColor(QColor(255, 170, 0)); //设置颜色
49     painter.save();
50
51     painter.setPen(pen);
52     painter.setBrush(QColor(255, 170, 0)); //设置画刷颜色
53
54     // 3.2 画点、写文本
55     for ( int i = 0; i < 6; i++ ) {
56         painter.drawEllipse(QPoint(pointX[i], pointY[i]), POINT_RADIUS,
POINT_RADIUS);
57         painter.drawText(QPoint(pointX[i] - TEXT_OFFSET_X, pointY[i] -
TEXT_OFFSET_Y), QString::number(info.highTemp[i]) + "°");
58     }
59
60     // 3.3 绘制曲线
61     for ( int i = 0; i < 5; i++ ) {
62         if ( i == 0 ) {
63             pen.setStyle(Qt::DotLine); //虚线
64             painter.setPen(pen);
65         } else {
66             pen.setStyle(Qt::SolidLine); // 实线
67             painter.setPen(pen);
68         }
69         painter.drawLine(pointX[i], pointY[i], pointX[i + 1], pointY[i + 1]);
70     }
71
72     painter.restore();
73 }

```

以下是绘制低温曲线的代码：

```
1 void Widget::paintLowCurve()
2 {
3     WeatherInfo info = WeatherInfoList[cityIndex];
4
5     QPainter painter(lblLow);
6
7     // 抗锯齿
8     painter.setRenderHint(QPainter::Antialiasing, true);
9
10    // 1. 获取 x 轴坐标
11    int pointX[6] = {0};
12    for ( int i = 0; i < 6; i++ ) {
13        pointX[i] = mAqiList[i]->pos().x() + mAqiList[i]->width() / 2;
14    }
15
16    // 2. 获取 y 轴坐标
17
18    // int temp[6] = {0};
19    int tempSum = 0;
20    int tempAverage = 0;
21
22    // 2.1 计算平均值
23    for ( int i = 0; i < 6; i++ ) {
24        tempSum += info.lowTemp[i];
25    }
26
27    tempAverage = tempSum / 6; // 最高温平均值
28
29    // qDebug() << "paintLowCurve" << tempAverage;
30
31    // 2.2 计算 y 轴坐标
32    int pointY[6] = {0};
33    int yCenter = lblLow->height() / 2;
34    for ( int i = 0; i < 6; i++ ) {
35        pointY[i] = yCenter - ((info.lowTemp[i] - tempAverage) * INCREMENT);
36    }
37
38    // 3. 开始绘制
39    // 3.1 初始化画笔
40    QPen pen = painter.pen();
41    pen.setWidth(1); //设置画笔宽度为1
```

```

42     pen.setColor(QColor(0, 255, 255)); //设置颜色
43     painter.save();
44
45     painter.setPen(pen);
46     painter.setBrush(QColor(0, 255, 255)); //设置画刷颜色
47
48     // 3.2 画点、写文本
49     for ( int i = 0; i < 6; i++ ) {
50         painter.drawEllipse(QPoint(pointX[i], pointY[i]), POINT_RADIUS,
POINT_RADIUS);
51         painter.drawText(QPoint(pointX[i] - TEXT_OFFSET_X, pointY[i] -
TEXT_OFFSET_Y), QString::number(info.lowTemp[i]) + "°");
52     }
53
54     // 3.3 绘制曲线
55     for ( int i = 0; i < 5; i++ ) {
56         if ( i == 0 ) {
57             pen.setStyle(Qt::DotLine); //虚线
58             painter.setPen(pen);
59         } else {
60             pen.setStyle(Qt::SolidLine); // 实线
61             painter.setPen(pen);
62         }
63         painter.drawLine(pointX[i], pointY[i], pointX[i + 1], pointY[i + 1]);
64     }
65
66     painter.restore();
67 }

```

## 9.3 程序说明

### (1) 宏定义

定义了以下几个宏，而不是在代码中直接写死：

```

1 // 温度曲线相关的宏
2 #define INCREMENT      3 // 温度每升高/降低1度，y轴坐标的增量
3 #define POINT_RADIUS  3 // 曲线描点的大小
4 #define TEXT_OFFSET_X 12 // 温度文本相对于点的偏移
5 #define TEXT_OFFSET_Y 10 // 温度文本相对于点的偏移

```

## (2) 绘图相关

绘制曲线时，有很多的小细节，就不详细展开了，比如：

- 画笔、画刷的颜色
- 划线、画圆
- 实线、虚线

相信你学过 Qt 中的绘图一节的话，这是再简单不过了！

程序最终完成的效果如下：

