QT第三天

一、QMessageBox对话框类

1.1 对象版实现

```
    需要使用QMessageBox类实例化对象
    设置该对象对应的属性
    调用该对象的extc来展示该对话框
    构造函数：QMessageBox(QMessageBox::Icon icon,       //图标
                          const QString &title,        //对话框的标题
                          const QString &text,         //对话框的文本信息
                          QMessageBox::StandardButtons buttons = NoButton,   //对话框对应的按钮
                          QWidget *parent = nullptr)         //父组件

        参数值：图标
QMessageBox::NoIcon 0 没有任何图标.
QMessageBox::Question 4 询问信息的图标.
QMessageBox::Information 1 表示该消息没有异常.
QMessageBox::Warning 2 一个能够被处理的警告信息
QMessageBox::Critical 3 一个错误信息

        参数值：按钮
QMessageBox::Ok  Ok按钮.
QMessageBox::Open 打开按钮
QMessageBox::Save  保存按钮
QMessageBox::Cancel  取消按钮
QMessageBox::Close  关闭按钮

    举个例子：
      QMessageBox msgBox;
      msgBox.setText("The document has been modified.");
      msgBox.setInformativeText("Do you want to save your changes?");
      msgBox.setStandardButtons(QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel);
      msgBox.setDefaultButton(QMessageBox::Save);
      int ret = msgBox.exec();
```

```
31
32
```

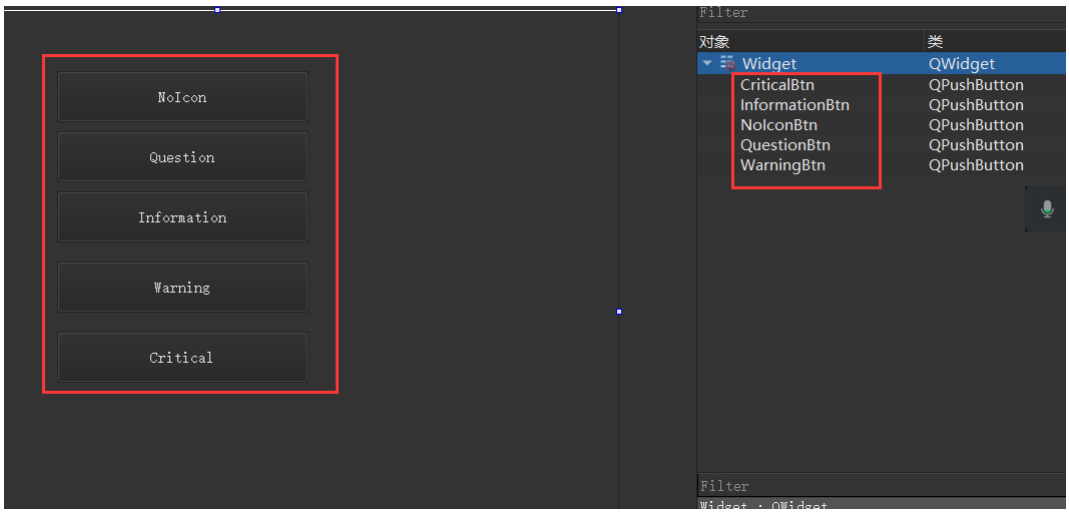
1.2 静态成员函数版

```
1  无需实例化对象，直接通过类名调用静态成员函数即可
2  一共有四个静态成员函数
3  静态成员函数版，没有设置图标选项，因为函数名自带图标
4  静态成员函数中，没有无图标的静态成员函数
5
6  //错误对话框
7  1>  QMessageBox::StandardButton
8  critical(QWidget *parent, const QString &title, const QString
   &text, QMessageBox::StandardButtons buttons = Ok,
   QMessageBox::StandardButton defaultButton = NoButton)
9  //信息对话框
10 2>  QMessageBox::StandardButton
11 information(QWidget *parent, const QString &title, const
   QString &text, QMessageBox::StandardButtons buttons = Ok,
   QMessageBox::StandardButton defaultButton = NoButton)
12 //询问对话框
13 3>  QMessageBox::StandardButton
14 question(QWidget *parent, const QString &title, const QString
   &text, QMessageBox::StandardButtons buttons =
   StandardButtons(Yes | No), QMessageBox::StandardButton
   defaultButton = NoButton)
15 //警告对话框
16 4>  QMessageBox::StandardButton
17 warning(QWidget *parent, const QString &title, const QString
   &text, QMessageBox::StandardButtons buttons = Ok,
   QMessageBox::StandardButton defaultButton = NoButton)
18
19 以信息对话框为例：
20 QMessageBox::StandardButton              //函数返回值是按钮，用户点
   击后的按钮
21 information(                              //函数名
22     QWidget *parent,                     //父组件
23     const QString &title,                //对话框标题
24     const QString &text,                 //对话框文本内容
25     QMessageBox::StandardButtons buttons = Ok,      //对话框上面
   的按钮
26     QMessageBox::StandardButton defaultButton = NoButton)
    //默认按钮
27
```

```
举个例子：
      int ret = QMessageBox::warning(this, tr("My Application"),
                                     tr("The document has been modified.\n"
                                        "Do you want to save your changes?"),
                                     QMessageBox::Save | QMessageBox::Discard
                                     | QMessageBox::Cancel,
                                     QMessageBox::Save);

```

1.3 案例

1> ui界面



2> 头文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include<QMessageBox>       //消息对话框对应的头文件
#include<QDebug>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

```

```
public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

private slots:
    void on_NoIconBtn_clicked();

    void on_QuestionBtn_clicked();

    void on_InformationBtn_clicked();

private:
    Ui::Widget *ui;
};
#endif // WIDGET_H

```

3> 源文件

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
}

Widget::~Widget()
{
    delete ui;
}

//无图标的按钮对应的槽函数
void Widget::on_NoIconBtn_clicked()
{
    //对象版
    QMessageBox msgBox;         //调用无参构造

    msgBox.setWindowTitle("无图标的对话框");     //设置对话框的标题
    msgBox.setIcon(QMessageBox::NoIcon);        //设置为无图标
    msgBox.setText("今天是个好天气，可以晒晒暖了");       //设置对话框文本信息
```

```
25      msgBox.setStandardButtons(QMessageBox::Ok |
    QMessageBox::Yes);      //设置标准按钮
26
27      int btn = msgBox.exec();                //展示对话框
28                                  //该函数返回值是整形，结果是用户点击消息
    对话框的按钮的值
29
30      //判断点击了消息对话框的哪个按钮
31      if(btn == QMessageBox::Ok)
32      {
33          qDebug()<<"您点击了Ok按钮";
34      }else if(btn == QMessageBox::Yes)
35      {
36          qDebug()<<"您点击了Yes按钮";
37      }
38
39  }
40
41  //询问按钮对应的槽函数
42  void Widget::on_QuestionBtn_clicked()
43  {
44      //实例化对象，调用有参构造
45      QMessageBox msgBox(QMessageBox::Question,       //图标
46                         "询问对话框",                  //对话框标题
47                         "你今天吃早餐了吗？",           //对话框文本
    内容
48                         QMessageBox::Yes | QMessageBox::No);
      //选择的按钮
49
50      //执行
51      int btn = msgBox.exec();
52
53      //对选择的结果进行判断
54      if(btn==QMessageBox::Yes)
55      {
56          qDebug()<<"怎么吃那么早呀，我还想着请你吃饭呢！";
57      }else if(btn == QMessageBox::No)
58      {
59          qDebug()<<"那你快去吃吧，我刚吃过";
60      }
61
62  }
63
64
65  //信息对话框对应的槽函数
```

```
void Widget::on_InformationBtn_clicked()
{
    QMessageBox::StandardButton btn = QMessageBox::information(this,                //父组件
                                "信息对话框",          //标题
                                "中午放学等着哈！",      //文本内容
                                QMessageBox::Yes|QMessageBox::No,   //按钮
                                QMessageBox::Yes);  //默认按钮
    //功能：调出信息对话框
    //返回值：用户点击的对话框上的按钮

    //对返回结果进行判断
    if(btn==QMessageBox::Yes)
    {
        qDebug()<<"摇好人，老地方";
    }else if(btn==QMessageBox::No)
    {
        qDebug()<<"干嘛呀，都是自己人";
    }

}
```

4> ui文件

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Widget</class>
 <widget class="QWidget" name="Widget">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>600</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Widget</string>
  </property>
  <widget class="QPushButton" name="NoIconBtn">
   <property name="geometry">
    <rect>
     <x>240</x>
```

```
      <y>60</y>
      <width>251</width>
      <height>51</height>
     </rect>
    </property>
    <property name="text">
     <string>NoIcon</string>
    </property>
   </widget>
   <widget class="QPushButton" name="QuestionBtn">
    <property name="geometry">
     <rect>
      <x>240</x>
      <y>120</y>
      <width>251</width>
      <height>51</height>
     </rect>
    </property>
    <property name="text">
     <string>Question</string>
    </property>
   </widget>
   <widget class="QPushButton" name="InformationBtn">
    <property name="geometry">
     <rect>
      <x>240</x>
      <y>180</y>
      <width>251</width>
      <height>51</height>
     </rect>
    </property>
    <property name="text">
     <string>Information</string>
    </property>
   </widget>
   <widget class="QPushButton" name="WarningBtn">
    <property name="geometry">
     <rect>
      <x>240</x>
      <y>250</y>
      <width>251</width>
      <height>51</height>
     </rect>
    </property>
    <property name="text">
```

```
    <string>Warning</string>
   </property>
  </widget>
  <widget class="QPushButton" name="CriticalBtn">
   <property name="geometry">
    <rect>
     <x>240</x>
     <y>320</y>
     <width>251</width>
     <height>51</height>
    </rect>
   </property>
   <property name="text">
    <string>Critical</string>
   </property>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>

```

练习：将登陆界面的两个按钮设置消息对话框

对于登陆按钮，如果登陆成功，给出信息框，说明登录成功，点击上面的ok键后，整个页面关闭

如果登陆失败，给出错误对话框，文本信息：“账号和密码不匹配，是否重新登录”，在消息对话框上给出两个按钮，yes|no，点击yes后，将账号和密码的编辑内容清空，完成下一次登录，点击no后，退出整个页面

对于，取消按钮，点击后，给出警告对话框，文本信息：“是否确定退出登录”，给出两个按钮，yes|no，

如果点击no，则无操作，如果选择yes则退出页面.

要求，对于登录按钮弹出的对话框用类对象实现，对于取消按钮弹出的对话框用静态成员函数版实现

```
//登录按钮对应的槽函数
void Widget::on_loginBtn_clicked()
{
    //实例化一个消息对话框类的对象
    QMessageBox msgBox;

    if(ui->userNameEdit->text()=="admin" && ui->pwsEdit->text()=="123456")
    {


```

```
        msgBox.setIcon(QMessageBox::Information);        //设置为信息对话框
        msgBox.setWindowTitle("信息");                   //设置窗口标题
        msgBox.setText("登录成功");                      //设置消息内容
        msgBox.exec();                                   //执行

        //发射对应的信号，在其槽函数中，执行相应的逻辑

        this->close();                                   //关闭登录页面

    }else
    {
        msgBox.setIcon(QMessageBox::Critical);           //设置为信息对话框
        msgBox.setWindowTitle("错误");                   //设置窗口标题
        msgBox.setText("账号和密码不匹配，是否重新登录？");   //设置消息内容

 msgBox.setStandardButtons(QMessageBox::Yes|QMessageBox::No);   //设置按钮

        int btn =  msgBox.exec();                        //执行

        //对用户选择的对话框按钮进行判断
        if(btn==QMessageBox::Yes)
        {
            //将输入的信息清空
            ui->userNameEdit->clear();
            ui->pwsEdit->clear();
        }else if(btn==QMessageBox::No)
        {
            this->close();
        }
    }
}
//取消按钮对应的槽函数
void Widget::on_cancelBtn_clicked()
{
    QMessageBox::StandardButton btn = QMessageBox::warning(this, "警告","您确定要退出登录吗？",

QMessageBox::Yes|QMessageBox::No);

    if(btn==QMessageBox::Yes)
    {
```

```
        this->close();
    }
}
```

二、颜色对话框和字体对话框

2.1 设置字体

```
1>  一般使用静态成员函数，getFont函数，调取字体对话框，获取某个字体
2>  函数原型如下
QFont                   //返回值，是一个字体
getFont(              //函数名
        bool *ok,    //是否选中字体
        const QFont &initial,  //调用字体对话框时的初始字体
        QWidget *parent = nullptr,      //父对象
        )
3>  举个例子
     bool ok;
  QFont font = QFontDialog::getFont(
                  &ok, QFont("Helvetica [Cronyx]", 10), this);
  if (ok) {
      // the user clicked OK and font is set to the font the
user selected
  } else {
      // the user canceled the dialog; font is set to the
initial
      // value, in this case Helvetica [Cronyx], 10
  }

4>  所需要的类
    字体对话框类：QFontDialog
    字体类：QFont

```

2.2 设置颜色

```
1>  一般使用静态成员函数，getColor函数，调取颜色对话框，获取某个颜色
2>  函数原型如下
QColor                                  //函数返回值
    QColorDialog::getColor(     //函数名
            const QColor &initial = Qt::white,  //初始颜色
            QWidget *parent = nullptr,          //父对象
            const QString &title = QString())    //窗口标题
3>  所需类
        颜色对话框类：QColorDialog
        颜色类：QColor
```

三、软件发布

```
第一步：进入我的电脑，在空白处右键选择属性---------->高级系统设置
择环境变量：
4.选择环境变量Path
5.输入Qt目录下MinGw的编译套件的目录
6.在你的工程发布的xxxxxx.exe文件夹下面 按住shift + 右键  后的选项中点击》》》》》》在此处打开Powershell窗口。
运行windeployqt  + xxxxxx.exe   加载所有相关的动态库就可以生成一个绿色的程序工作，就可运行到任何一台电脑之上。
```

四、事件处理机制

```
1. 什么是事件？  (重点)
    事件是由窗口系统或者自身产生的，用以响应所发生的
    各类事情，比如用户按下并释放了键盘或者鼠标、窗口因
    暴露而需要重绘、定时器到时而应有所动作，等等

    从某种意义上讲，事件比信号更原始，甚至可以认为大多
    数信号其实都是由事件产生的。比如一个下压式按钮首先
    感受到的是鼠标事件，在进行必要的处理以产生按钮下沉
    继而弹起的视觉效果之后，才会发射 clicked()信号

2. 如何处理事件？  (重点)
   myWnd(自定义类) -继承-> QWidget -继承-> QObject
   1> 当事件发生时，首先被调用的是QObject类中的虚函数event()，
   其 QEvent型参数标识了具体的事件类型
    bool QObject:: event (QEvent* e)
    {
        if (e == mouseEvent)
        {
            void QWidget::mousePressEvent (QMouseEvent* e)
```

```
            void QWidget:: mouseReleaseEvent (QMouseEvent* e)
        }
        if(e == keyEvent){
            void QWidget::keyPressEvent (QMouseEvent* e)
            void QWidget:: keyReleaseEvent (QMouseEvent* e)
        }
    }
   2> 作为QObject类的子类， QWidget类覆盖了其基类中的
   event()虚函数，并根据具体事件调用具体事件处理函数
    void QWidget::mousePressEvent (QMouseEvent* e)
    void QWidget::mouseReleaseEvent (QMouseEvent* e)
    void QWidget::keyPressEvent (QMouseEvent* e)
    void QWidget:: keyReleaseEvent (QMouseEvent* e)
    void QWidget::paintEvent (QPaintEvent* e):
   3> 而这些事件处理函数同样也是虚函数，也可以被 QWidget类
   的子类覆盖，以提供针对不同窗口部件类型的事件处理

   4> 组件的使用者所关心的往往是定义什么样的槽处理什么样的信号，
   而组件的实现者更关心覆盖哪些事件处理函数

   5> 当有下列情况之一发生时，窗口部件会收到绘制事件，
    即 QWidget类的 paintEvent()虚函数会被调用
    最终调用的是子类中重写的PaintEvent()事件函数
    窗口被创建以后第一次显示出来
    窗口由隐藏状态转变为可见状态
    窗口由最小化状态转变为正常或最大化状态
    窗口超出屏幕边界的区域进入屏幕范围之内
    窗口被遮挡的区域因某种原因重新暴露出来
    窗口因尺寸大小的变化需要呈现更多的内容
    QWidget类的 update()成员函数被调用
   6> 作为 QWidget类的子类，可以在对该虚函数的覆盖版本中
    实现诸如显示文本、绘制图形、濾染图像等操作
    void ShowPicsDlg::paintEvent(QPaintEvent *e)
   7> QPainter类是Qt的二维图形引擎，该类具有如下功能
    绘制矢量文字
    绘制几何图形
    绘制响度映射和图像
    反走样，像素混合，渐变和矢量路径
    平移、选择、错切、缩放等线性变换
   8> QPainter类通过构造函数接收绘制设备，即在什么上画
    QPainter::QPainter(QPaintDevice* device);
   9> QPainter类用于渲染图像的众多成员函数之一
    void QPainter::drawImage(const QRect& rect,const QImage&
image);
```

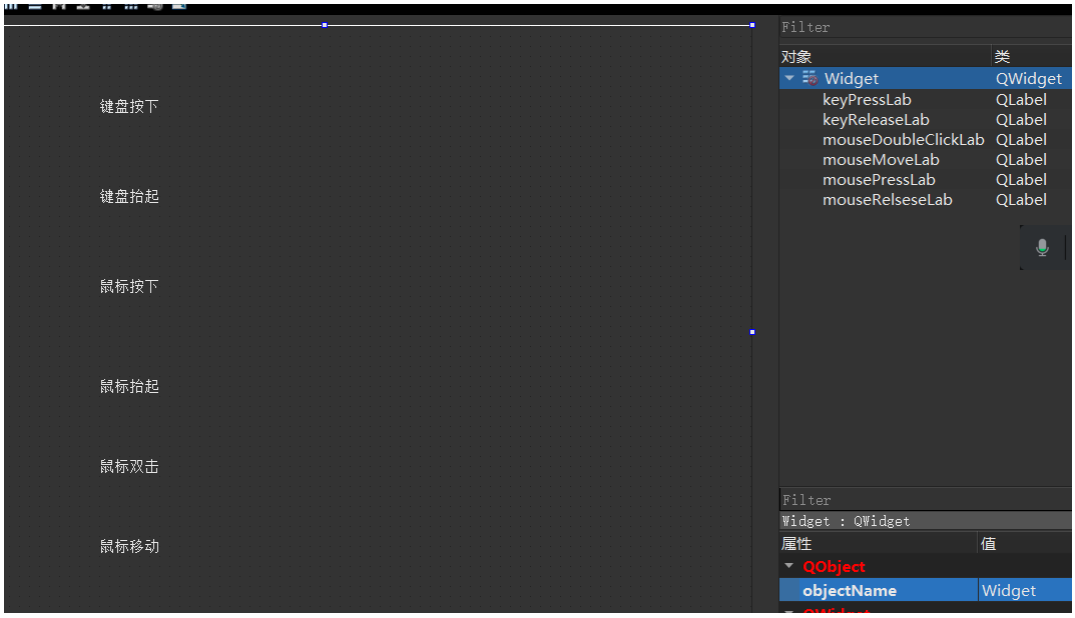
```
QObject类 提供了那些可以重写的虚函数
    [virtual] bool QObject::event(QEvent *e)
            // 参数：事件的类型

QWidgets类， 提供了那些可以重写的虚函数
    [override virtual protected] bool QWidget::event(QEvent *event)

    [virtual protected] void QWidget::keyPressEvent(QKeyEvent *event)
    [virtual protected] void QWidget::keyReleaseEvent(QKeyEvent *event)
    [virtual protected] void QWidget::mouseMoveEvent(QMouseEvent *event)
    [virtual protected] void QWidget::mousePressEvent(QMouseEvent *event)
    [virtual protected] void QWidget::mouseReleaseEvent(QMouseEvent *event)
    [virtual protected] void QWidget::mouseDoubleClickEvent(QMouseEvent *event)
    [virtual protected] void QObject::timerEvent(QTimerEvent *event)

QPainter类 ---> 画家类
     void SimpleExampleWidget::paintEvent(QPaintEvent *)
     {
         QPainter painter(this);
         painter.setPen(Qt::blue);
         painter.setFont(QFont("Arial", 30));
         painter.drawText(rect(), Qt::AlignCenter, "Qt");
     }

```

五、鼠标和键盘事件

```
1>  需要引入的头文件
        #include <QKeyEvent>          //键盘事件处理的头文件
        #include <QMouseEvent>        //鼠标的头文件
2>  需要重写的事件处理函数
    [virtual protected] void QWidget::keyPressEvent(QKeyEvent *event)
    [virtual protected] void QWidget::keyReleaseEvent(QKeyEvent *event)
    [virtual protected] void QWidget::mouseMoveEvent(QMouseEvent *event)
    [virtual protected] void QWidget::mousePressEvent(QMouseEvent *event)
    [virtual protected] void QWidget::mouseReleaseEvent(QMouseEvent *event)
    [virtual protected] void QWidget::mouseDoubleClickEvent(QMouseEvent *event)
```

案例:

1> ui界面



2> 头文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include<QMouseEvent>          //鼠标处理事件
#include<QKeyEvent>            //键盘处理事件

QT_BEGIN_NAMESPACE
```

```
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

    void keyPressEvent(QKeyEvent *event);           //重写键盘按下处理事件函数
    void keyReleaseEvent(QKeyEvent *event);          //重写键盘抬起处理事件函数
    void mousePressEvent(QMouseEvent *event);          //重写鼠标按下事件函数
    void mouseReleaseEvent(QMouseEvent *event);        //重写鼠标释放事件函数


private:
    Ui::Widget *ui;
};
#endif // WIDGET_H

```

3> 源文件

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
}

Widget::~Widget()
{
    delete ui;
}

//键盘按下处理事件函数的定义
```

```cpp
void Widget::keyPressEvent(QKeyEvent *event)
{
    QString msg;
    msg = event->text() + "被按下   ascii---->"+
QString::number(event->key());
                //a被按下  ascii----> 97

    ui->keyPressLab->setText(msg);          //将信息展示到标签中
}

//键盘抬起处理事件函数的定义
void Widget::keyReleaseEvent(QKeyEvent *event)
{
    QString msg;
    msg = event->text() + "被抬起   ascii---->"+
QString::number(event->key());
                //a被按下  ascii----> 97

    ui->keyReleaseLab->setText(msg);          //将信息展示到标签中
}

//鼠标按下事件函数
void Widget::mousePressEvent(QMouseEvent *event)
{
    if(event->buttons() == Qt::LeftButton)
    {
        ui->mousePressLab->setText("鼠标左键被按下");
    }else if(event->buttons() == Qt::RightButton)
    {
        ui->mousePressLab->setText("鼠标右键被按下");
    }else if(event->buttons() == Qt::MidButton)
    {
        ui->mousePressLab->setText("鼠标中间被按下");
    }

}
//鼠标抬起事件函数
void Widget::mouseReleaseEvent(QMouseEvent *event)
{
    if(event->button() == Qt::LeftButton)
    {
        ui->mouseRelseseLab->setText("鼠标左键被抬起");
    }else if(event->button() == Qt::RightButton)
    {
        ui->mouseRelseseLab->setText("鼠标右键被抬起");
```

```
60      }else if(event->button() == Qt::MidButton)
61      {
62          ui->mouseRelseseLab->setText("鼠标中间被抬起");
63      }
64  }
65
```

作业：

自己完成鼠标双击事件和鼠标移动事件的处理函数内容