1.QWidget类中的事件

在Qt中几乎所有的事件都是基于QWidget类的，所以我们在使用事件时，都是通过继承QWidget类，然后重写事件函数去实现的（注意：所有的事件函数都要写在protected下）；

virtual void actionEvent(QActionEvent *event)	
virtual void changeEvent(QEvent *event)	窗体变化
virtual void closeEvent(QCloseEvent *event)	窗口关闭
virtual void contextMenuEvent(QContextMenuEvent *event)	鼠标右键菜单
void create(WId window = 0, bool initializeWindow = true, bool destroyOldWindow = true)	
void destroy(bool destroyWindow = true, bool destroySubWindows = true)	
virtual void dragEnterEvent(QDragEnterEvent *event)	将文件拖拽进入的事件
virtual void dragLeaveEvent(QDragLeaveEvent *event)	将文件拖拽离开的事件
virtual void dragMoveEvent(QDragMoveEvent *event)	将文件拖拽移动的事件
virtual void dropEvent(QDropEvent *event)	将文件拖拽放下的事件
virtual void enterEvent(QEvent *event)	鼠标进入
virtual void focusInEvent(QFocusEvent *event)	获得焦点
bool focusNextChild()	
virtual bool focusNextPrevChild(bool next)	
virtual void focusOutEvent(QFocusEvent *event)	释放焦点
bool focusPreviousChild()	
virtual void hideEvent(QHideEvent *event)	窗体从屏幕上消失时的事件（包括最小化，不包括被遮挡）
virtual void inputMethodEvent(QInputMethodEvent *event)	输入法事件
virtual void keyPressEvent(QKeyEvent *event)	键盘按键按下事件
virtual void keyReleaseEvent(QKeyEvent *event)	键盘按键释放事件
virtual void leaveEvent(QEvent *event)	鼠标离开

virtual void mouseDoubleClickEvent(QMouseEvent *event)	鼠标双击事件
virtual void mouseMoveEvent(QMouseEvent *event)	鼠标按下后的移动事件
virtual void mousePressEvent(QMouseEvent *event)	鼠标点击事件
virtual void mouseReleaseEvent(QMouseEvent *event)	鼠标释放事件
virtual void moveEvent(QMoveEvent *event)	窗口移动
virtual bool nativeEvent(const QByteArray &eventType, void *message, long *result)	
virtual void paintEvent(QPaintEvent *event)	窗口重绘
virtual void resizeEvent(QResizeEvent *event)	窗口改变大小
virtual void showEvent(QShowEvent *event)	窗体显示在屏幕上时的事件
virtual void tabletEvent(QTabletEvent *event)	
virtual void wheelEvent(QWheelEvent *event)	鼠标滚轮事件

2.Qt中处理事件的5种方式

（1）重写事件处理器函数

这是大部分情况最常用的一种，如重写 paintEvent()、mousePressEvent()、keyPressEvent() 等事件处理器虚函数。

（2）重写 QObject::event() 函数

通过重写 event() 函数可以在事件到达特定的事件处理器之前处理它们。这种方式常用于拦截 Tab 键的处理(Tab 键默认的意义是焦点切换，但是对于某些控件这个可能要用来完成文本缩进之类的工作)，以及没有事件处理器的不常见事件如 QEvent::HoverEnter 。当重写 event() 时，对于没有处理的事件必须调用基类的 event() 函数。

（3）安装事件过滤器

需要被监视的目标对象调用 installEventFilter() 注册监视对象后，被监视目标的所有事件都会先被监视对象的过滤器 eventFilter() 拦截，可以在这个函数里面对监视目标的某个事件做特殊处理。一个对象可以安装多个事件过滤器，事件过滤器的执行顺序和安装顺序相反。会从最近安装的那个 eventFilter() 开始执行。

（4）在 QApplication 对象中安装事件过滤器

一旦在 qApp(全局唯一的运用程序对象) 中注册了事件过滤器，则运用程序中所有对象的每一个事件在发送到其他事件过滤器之前，都会被注册的那个监视器拦截。这对于调试非常有用。(就是定义一个 QObject 类的子类，重写 eventFilter() 函数，实例化一个过滤器对象 filter，然后 qApp->installEventFilter(filter))

（5）继承 QApplication 重写 notify() 函数

Qt 是通过调用 QApplication::notify() 函数来发送事件的，重写这个函数即可在事件过滤器之前拦截所有事件。

3.Qt中安装事件过滤器

（1）在目标对象上调用installEventFilter(),将监测对象注册到目标对象上。

（2）在监测对象的eventFilter()方法里处理目标对象的事件。

```
myMenu::myMenu(QWidget *parent) : QWidget(parent)
{
 //构造函数中为控件设置过滤器
    ui->zoomCheckBox->installEventFilter(this);
    ui->muteCheckBox->installEventFilter(this);
}

bool myMenu::eventFilter(QObject *obj, QEvent *event)
{
 //如果对象是zoomCheckBox，当鼠标进入（悬停）其上，则zoomTips显示
    if (obj == (QObject*)ui->zoomCheckBox)
    {
        if (event->type() == QEvent::Enter)
        {
            ui->zoomTips->setVisible(true);
        }
        else if(event->type() == QEvent::Leave)
        {
            ui->zoomTips->setVisible(false);
        }
        return false;
    }
    //监测对象：muteCheckBox；关注的事件：Enter和Leave
    else if (obj == (QObject*)ui->muteCheckBox)
    {
        if (event->type() == QEvent::Enter)
        {
            ui->muteTips->setVisible(true);
        }
        else if(event->type() == QEvent::Leave)
```

```
        {
            ui->muteTips->setVisible(false);
        }
        return false;
    }
}
```

4.键盘事件中的常用方法

```
this->grabKeyboard();//监听键盘事件，需要在构造函数中加入这句代码

event->modifiers() == Qt::ControlModifier;//判断释放按下Ctrl键
event->key() == Qt::Key_Control;//判断释放按下Ctrl键
event->modifiers() == Qt::ShiftModifier;//判断释放按下Shift键
event->key() == Qt::Key_Shift;//判断释放按下Shift键
event->isAutoRepeat();//判断是否是长按
```

5.鼠标事件中的常用方法

```
this->grabMouse();//在监听鼠标按键是否按下时，需要在构造函数中加入这句代码

event->button() == Qt::LeftButton;//判断鼠标左键是否按下
event->button() == Qt::RightButton;//判断鼠标右键是否按下
event->button() == Qt::MidButton;//判断鼠标中键是否按下
event->button() == Qt::MiddleButton;//判断鼠标中键是否按下

//这两个方法只有当鼠标放置在对应组件上时才会生效
event->delta() > 0;//鼠标向使用者的反方向滚动
event->delta() < 0;//鼠标向使用者的方向滚动

//这个方法主要应用于鼠标按下后的移动事件中
qDebug()<<event->x();//获取鼠标的x坐标
qDebug()<<event->y();//获取鼠标的y坐标
qDebug()<<event->pos();//获取鼠标的坐标(相对于窗体的坐标),如QPoint(265,156)
qDebug()<<event->globalPos();//获取鼠标的坐标(相对于当前显示器的坐标)
```