

10.3 Graphics View 绘图架构

10.3.5 Graphics View 绘图程序

1. 示例功能

示例 samp10_4 只是演示了 Graphics View 的基本结构和三个坐标系的概念,为了演示 Graphics View 结构编程的更多功能,创建示例项目 samp10_5。这是一个基于 Graphics View 结构的简单绘图程序,通过这个示例可以发现 Graphics View 图形编程更多功能的使用方法。程序运行界面如图 10-3-6 所示。

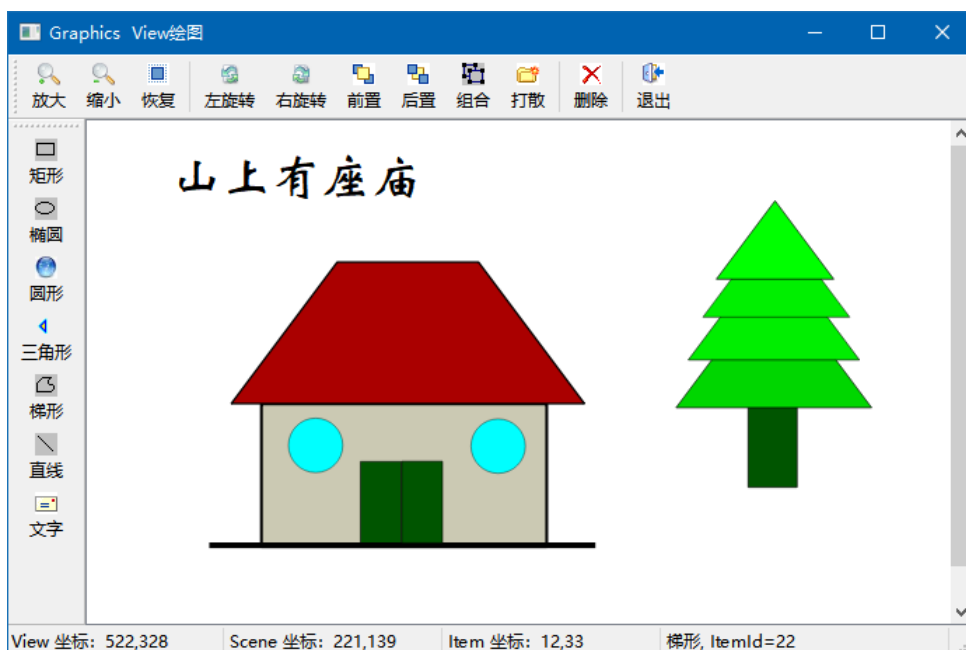


图 10-3-6 基于 Graphics View 结构的绘图程序

这个示例程序具有如下的功能:

- 可以创建矩形、椭圆、圆形、三角形、梯形、直线、文字等基本图形项。
- 每个图形项都可以被选择和拖动。
- 图形项或整个视图可以缩放和旋转。
- 图形项重叠时,可以调整前置或后置。
- 多个图形项可以组合,也可以解除组合。
- 可以删除选择的图形项。
- 鼠标在视图上移动时,会在状态栏显示视图坐标和场景坐标。
- 鼠标单击某个图形项时,会显示图形项的局部坐标,还会显示图形项的文字描述和编号。
- 双击某个图形项时,会根据图形项的类型调用颜色对话框或字体对话框,设置图形项的填充颜色、线条颜色或文字的字体。

- 选中某个图形项时，可以进行按键操作，Delete 键删除图形项，PgUp 放大，PgDn 缩小，空格键旋转 90 度，上下左右光标键移动图形项。

2. 主窗口可视化设计

创建项目时选择窗口基类为 QMainWindow，采用可视化方法设计主窗口界面。主窗口设计好的效果如图 10-3-7 所示。我们删除了主窗口的菜单栏，添加了一个工具栏，设置其 orientation 属性为 vertical，allowedAreas 属性只有 LeftToolBarArea，这个工具栏就停靠在窗口左侧了。工作区放置了一个 QGraphicsView 组件，设置其 objectName 为 view。

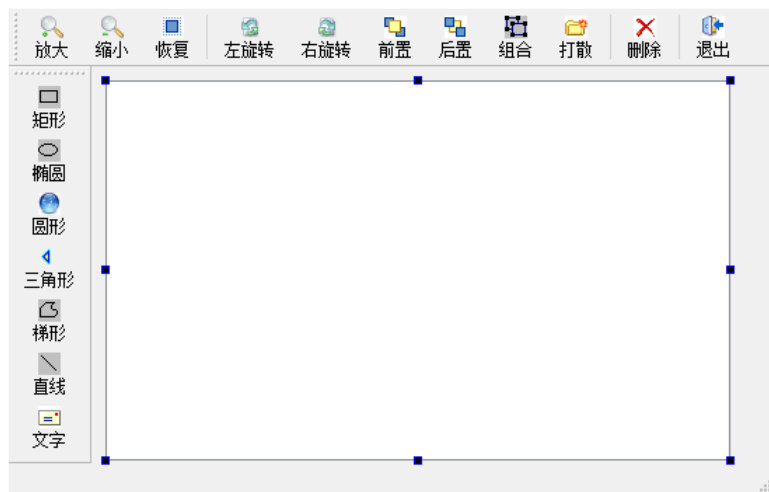


图 10-3-7 可视化设计时的主窗口

在主窗口中设计的 Actions 如图 10-3-8 所示，两个工具栏上的按钮由这些 Actions 创建。这些 Actions 分为两大类，一类是对场景、视图或图形项进行操作的 Actions，用于创建水平工具栏上的按钮；另一类是创建各种图形项的 Actions，用于创建左侧垂直工具栏上的按钮。这些 Actions 都没有设置快捷键，我们为视图组件添加对 keyPress 事件的处理，实现一些快捷键操作。

Name	Used	Text	ToolTip	Shortcut	Checkable
actItem_Rect	<input checked="" type="checkbox"/>	矩形	添加矩形		<input type="checkbox"/>
actItem_Ellipse	<input checked="" type="checkbox"/>	椭圆	添加椭圆型		<input type="checkbox"/>
actItem_Line	<input checked="" type="checkbox"/>	直线	添加直线		<input type="checkbox"/>
actEdit_Delete	<input checked="" type="checkbox"/>	删除	删除选中的图元		<input type="checkbox"/>
actQuit	<input checked="" type="checkbox"/>	退出	退出本系统		<input type="checkbox"/>
actItem_Text	<input checked="" type="checkbox"/>	文字	添加文字		<input type="checkbox"/>
actEdit_Front	<input checked="" type="checkbox"/>	前置	居于最前面		<input type="checkbox"/>
actEdit_Back	<input checked="" type="checkbox"/>	后置	居于最后面		<input type="checkbox"/>
actItem_Polygon	<input checked="" type="checkbox"/>	梯形	添加梯形		<input type="checkbox"/>
actZoomIn	<input checked="" type="checkbox"/>	放大	放大		<input type="checkbox"/>
actZoomOut	<input checked="" type="checkbox"/>	缩小	缩小		<input type="checkbox"/>
actRotateLeft	<input checked="" type="checkbox"/>	左旋转	左旋转		<input type="checkbox"/>
actRotateRight	<input checked="" type="checkbox"/>	右旋转	右旋转		<input type="checkbox"/>
actRestore	<input checked="" type="checkbox"/>	恢复	恢复大小		<input type="checkbox"/>
actGroup	<input checked="" type="checkbox"/>	组合	组合		<input type="checkbox"/>
actGroupBreak	<input checked="" type="checkbox"/>	打散	取消组合		<input type="checkbox"/>
actItem_Circle	<input checked="" type="checkbox"/>	圆形	圆形		<input type="checkbox"/>
actItem_Triangle	<input checked="" type="checkbox"/>	三角形	三角形		<input type="checkbox"/>

图 10-3-8 在主窗口中设计的 Actions

3. 自定义图形视图类 TGraphicsView

与示例 samp10_4 类似，我们需要从 QGraphicsView 类继承定义一个图形视图类 TGraphicsView，在自定义类中增加鼠标和按键事件的处理，将鼠标和按键事件转换为信号，以便在主程序中设计槽函数做相应的处理。

可以将项目 samp10_4 中的文件 tgraphicsview.h/.cpp 复制到本项目来，然后在其基础上修改。下面是 TGraphicsView 类的定义代码，它重定义了 4 个事件处理函数，定义了 4 个信号。

```
class TGraphicsView : public QGraphicsView
{
    Q_OBJECT
protected:
    void mouseMoveEvent(QMouseEvent *event);
    void mousePressEvent(QMouseEvent *event);
    void mouseDoubleClickEvent(QMouseEvent *event);
    void keyPressEvent(QKeyEvent *event);
public:
    TGraphicsView(QWidget *parent = nullptr);
signals:
    void mouseMovePoint(QPoint point);      // 鼠标移动
    void mouseClicked(QPoint point);        // 鼠标单击
    void mouseDoubleClick(QPoint point);    // 双击事件
    void keyPress(QKeyEvent *event);        // 按键事件
};
```

下面是 4 个事件处理函数的实现代码，在每个事件里发射相应的信号。

```
void TGraphicsView::mouseMoveEvent(QMouseEvent *event)
{ // 鼠标移动事件
    QPoint point=event->pos();      //QGraphicsView 的坐标
    emit mouseMovePoint(point);    //发射信号
    QGraphicsView::mouseMoveEvent(event);
}

void TGraphicsView::mousePressEvent(QMouseEvent *event)
{ // 鼠标左键按下事件
    if (event->button()==Qt::LeftButton)
    {
        QPoint point=event->pos(); //QGraphicsView 的坐标
        emit mouseClicked(point);  //发射信号
    }
    QGraphicsView::mousePressEvent(event);
}

void TGraphicsView::mouseDoubleClickEvent(QMouseEvent *event)
{ // 鼠标双击事件
    if (event->button()==Qt::LeftButton)
    {
        QPoint point=event->pos();      //QGraphicsView 的坐标
```

```

        emit mouseDoubleClick(point);    //发射信号
    }
    QGraphicsView::mouseDoubleClickEvent(event);
}

void TGraphicsView::keyPressEvent(QKeyEvent *event)
{ //按键事件
    emit keyPress(event);                //发射信号
    QGraphicsView::keyPressEvent(event);
}

```

在设计好 TGraphicsView 后,需要将主窗口界面上的 QGraphicsView 组件提升为 TGraphicsView 类,操作方法见图 10-3-5 和相关解释。

4. 主窗口类定义和初始化

在主窗口类 MainWindow 中增加一些变量和函数定义, MainWindow 的定义代码如下,省略了 Actions 的槽函数的定义。

```

class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    const quint32 boundValue=100;    //随机数上限值
    const int ItemId = 1;            //图形项自定义数据的 key
    const int ItemDescription = 2;    //图形项自定义数据的 key
    int seqNum=0;    //用于图形项的编号,每个图形项有一个编号
    int backZ=0;    //用于 bring to front
    int frontZ=0;    //用于 bring to back
    QGraphicsScene *scene;    //场景
    QLabel *labViewCord;    //用于状态栏
    QLabel *labSceneCord;
    QLabel *labItemCord;
    QLabel *labItemInfo;

    void setItemProperty(QGraphicsItem* item, QString description);    //设置图形项的属性
public:
    MainWindow(QWidget *parent = nullptr);
private slots:
    //自定义槽函数
    void do_mouseMovePoint(QPoint point);    //鼠标移动
    void do_mouseClicked(QPoint point);    //鼠标单击
    void do_mouseDoubleClick(QPoint point);    //鼠标双击
    void do_keyPress(QKeyEvent *event);    //按键
private:
    Ui::MainWindow *ui;
};

```

MainWindow 中定义了几个常数或变量,各常数或变量的作用描述如下:

- 常数 boundValue 用于设置随机数发生器产生随机数的数值上限值。

- 常数 `ItemId` 和 `ItemDescription` 是用于设置图形项的自定义数据时用到的键。
- 变量 `seqNum` 用于给每个图形项编号，每个图形项有一个唯一编号。
- 变量 `frontZ` 用于设置图形项的叠放顺序，数值越大，越在前面显示。
- 变量 `backZ` 用于设置图形项的叠放顺序，数值越小，越在后面显示。

私有函数 `setItemProperty()` 用于在场景图形项后，设置图形项的属性。`MainWindow` 中定义了 4 个自定义槽函数，用于与 `TGraphicsView` 组件的 4 个信号关联。

`MainWindow` 类的构造函数代码如下：

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    labViewCord=new QLabel("View 坐标: "); //创建状态栏上的标签
    labViewCord->setMinimumWidth(150);
    ui->statusBar->addWidget(labViewCord);
    labSceneCord=new QLabel("Scene 坐标: ");
    labSceneCord->setMinimumWidth(150);
    ui->statusBar->addWidget(labSceneCord);
    labItemCord=new QLabel("Item 坐标: ");
    labItemCord->setMinimumWidth(150);
    ui->statusBar->addWidget(labItemCord);
    labItemInfo=new QLabel("ItemInfo: ");
    labItemInfo->setMinimumWidth(200);
    ui->statusBar->addWidget(labItemInfo);

    scene=new QGraphicsScene(-300,-200,600,400); //创建 QGraphicsScene
    ui->view->setScene(scene); //scene 与 view 关联
    ui->view->setCursor(Qt::CrossCursor); //设置鼠标光标
    ui->view->setMouseTracking(true); //设置鼠标跟踪
    ui->view->setDragMode(QGraphicsView::RubberBandDrag); //设置拖动模式
    this->setCentralWidget(ui->view);

    connect(ui->view,&TGraphicsView::mouseMovePoint,this,&MainWindow::do_mouseMovePoint);
    connect(ui->view,&TGraphicsView::mouseClicked,this,&MainWindow::do_mouseClicked);
    connect(ui->view,&TGraphicsView::keyPress,this,&MainWindow::do_keyPress);
    connect(ui->view,&TGraphicsView::mouseDoubleClick,
            this,&MainWindow::do_mouseDoubleClick);
}
```

在此构造函数里，程序创建了场景对象 `scene`，并且与界面上的视图组件 `view` 关联。程序还将 4 个自定义槽函数与视图组件 `view` 的 4 个信号关联。

5. 图形项的创建

主窗口左侧工具栏上的按钮用于创建各种标准的图形项。下面是创建椭圆的 `Action` 的槽函数，以及自定义函数 `setItemProperty()` 的代码：

```
void MainWindow::on_actItem_Ellipse_triggered()
{ //添加一个椭圆
```

```

    QGraphicsEllipseItem *item=new QGraphicsEllipseItem(-50,-30,100,60);
    item->setBrush(QBrush(Qt::blue)); //填充颜色
    setItemProperty(item, "椭圆");
}

void MainWindow::setItemProperty(QGraphicsItem *item,QString description)
{
    item->setFlags(QGraphicsItem::ItemIsMovable //可移动
                  | QGraphicsItem::ItemIsSelectable //可选中
                  | QGraphicsItem::ItemIsFocusable); //可以获得焦点
    item->setZValue(++frontZ); //叠放顺序号
    quint32 v1=QRandomGenerator::global()->bounded(boundValue); //范围[0, boundValue)
    quint32 v2=QRandomGenerator::global()->bounded(boundValue);
    item->setPos(v1,v2); //在场景中的位置

    item->setData(ItemId,++seqNum); //图形项编号
    item->setData(ItemDescription,description); //图形项描述
    scene->addItem(item); //添加到场景
    scene->clearSelection();
    item->setSelected(true);
}

```

Action 的槽函数里首先创建了一个 QGraphicsEllipseItem 类型的图形项 item，这是椭圆图形项。这里使用的 QGraphicsEllipseItem 类的构造函数原型定义如下：

```

QGraphicsEllipseItem(qreal x, qreal y, qreal width, qreal height, QGraphicsItem *parent = nullptr)

```

这表示在左上方顶点坐标为(x, y)，宽度为 width，高度为 height 的一个矩形框内创建一个椭圆，坐标 (x, y) 是图形项的局部坐标。代码里 (x, y) = (-50, -30)，宽度为 100，高度为 60，所以创建的椭圆的中心点在图形项局部坐标的原点 (0, 0)。一般的图形项的中心点都设置在其局部坐标的原点。

自定义函数 setItemProperty()用于设置图形项的属性，它通过 QGraphicsItem 类的接口函数做了如下的一些设置：

- setFlags()函数设置图形项的特性，程序中将图形项设置为可移动、可选择、可以获得焦点。
- setZValue()函数设置图形项的 Z 值，Z 值控制叠放顺序，当有多个图形项叠放在一起时，Z 值最大的显示在最前面。
- setPos(x, y)函数设置图形项的位置，如果图形项有父容器项，坐标 (x, y) 是父容器的坐标，否则就是图形场景的坐标。程序中使用了全局的随机数发生器 QRandomGenerator::global()，使用函数 bounded() 产生限制范围的随机数。
- setData()函数用于设置图形项的自定义数据，这个函数的原型定义如下：

```

void QGraphicsItem::setData(int key, const QVariant &value)

```

参数 key 是数据名称，value 是具体的数据内容，可以是任何类型。key 和 value 是一个键值对，使用 setData()一次可以设置一个键值对，可以为一个图形项设置多个自定义键值对。程序里设置了两个自定义数据：

```

item->setData(ItemId,++seqNum); //图形项编号

```

```
item->setData(ItemDescription,description); //图形项描述
```

ItemId 是图形项的编号，其取值 seqNum 是一个递增变量，ItemDescription 是图形项的描述，其取值 description 是函数 setItemProperty() 的输入参数。这样，每个图形项有一个唯一的编号，有一个文字描述。在窗口上单击某个图形项时，会提取这两个自定义数据在状态栏上显示。

创建其他几种类型图形项的代码如下，它们用到了不同的图形项类。

```
void MainWindow::on_actItem_Rect_triggered()
{ //添加一个矩形
    QGraphicsRectItem *item=new QGraphicsRectItem(-50,-25,100,50);
    item->setBrush(QBrush(Qt::yellow));
    setItemProperty(item, "矩形");
}

void MainWindow::on_actItem_Circle_triggered()
{ //添加圆形
    QGraphicsEllipseItem *item=new QGraphicsEllipseItem(-50,-50,100,100);
    item->setBrush(QBrush(Qt::cyan));
    setItemProperty(item,"圆形");
}

void MainWindow::on_actItem_Triangle_triggered()
{ //添加三角形
    QGraphicsPolygonItem *item=new QGraphicsPolygonItem;
    QPolygonF points;
    points.append(QPointF(0,-40)); //添加顶点坐标
    points.append(QPointF(60,40));
    points.append(QPointF(-60,40));
    item->setPolygon(points); //三角形就是一种多边形
    item->setBrush(QBrush(Qt::magenta));
    setItemProperty(item,"三角形");
}

void MainWindow::on_actItem_Polygon_triggered()
{ //添加梯形
    QGraphicsPolygonItem *item=new QGraphicsPolygonItem;
    QPolygonF points;
    points.append(QPointF(-40,-40)); //添加顶点坐标
    points.append(QPointF(40,-40));
    points.append(QPointF(100,40));
    points.append(QPointF(-100,40));
    item->setPolygon(points); //创建多边形
    item->setBrush(QBrush(Qt::green));
    setItemProperty(item,"梯形");
}

void MainWindow::on_actItem_Line_triggered()
{ //添加直线
    QGraphicsLineItem *item=new QGraphicsLineItem(-100,0,100,0);
    QPen pen(Qt::red);
```

```

    pen.setWidth(3);
    item->setPen(pen);
    setItemProperty(item, "直线");
}

void MainWindow::on_actItem_Text_triggered()
{ //添加文字
    QString str=QInputDialog::getText(this, "输入文字", "请输入文字");
    if (str.isEmpty())
        return;

    QGraphicsTextItem *item=new QGraphicsTextItem(str);
    QFont font=this->font();
    font.setPointSize(20);
    font.setBold(true);
    item->setFont(font); //设置字体
    setItemProperty(item, "文字");
}

```

6. 图形项操作

主窗口上水平工具栏上的一些按钮实现图形项的缩放、旋转、组合等操作。

(1) 缩放

图形项的缩放使用 QGraphicsItem 的 setScale()函数，参数大于 1 是放大，小于 1 是缩小。下面是“放大”按钮关联的槽函数代码：

```

void MainWindow::on_actZoomIn_triggered()
{ //放大
    int cnt=scene->selectedItems().count(); //选中图形项的个数
    if (cnt==1) //缩放单个图形项
    {
        QGraphicsItem *item;
        item=scene->selectedItems().at(0);
        item->setScale(0.1+item->scale());
    }
    else //缩放视图
        ui->view->scale(1.1, 1.1);
}

```

QGraphicsScene 的 selectedItems()函数返回场景中选中的图形项的列表。如果只有一个图形项被选中，就用 QGraphicsItem 的 setScale()函数对图形项进行缩放；如果选中的图形项个数大于 1 个，或没有图形项被选中，就用 QGraphicsView 的 scale()函数对绘图视图进行缩放。

(2) 旋转

图形项的旋转使用 QGraphicsItem 的 setRotation()函数，参数为角度值，正值表示顺时针旋转，负值表示逆时针旋转。下面是“左旋转”按钮关联的槽函数代码：

```

void MainWindow::on_actRotateLeft_triggered()

```



```

{ // 逆时针旋转, 左旋转
    int cnt=scene->selectedItems().count();
    if (cnt==1) // 单个图形项旋转
    {
        QGraphicsItem* item=scene->selectedItems().at(0);
        item->setRotation(-30+item->rotation());
    }
    else // 视图旋转
        ui->view->rotate(-30);
}

```

(3) 恢复坐标变换

缩放和旋转都是坐标变换, 要取消所有变换恢复初始状态, 调用 QGraphicsItem 或 QGraphicsView 的 resetTransform() 函数。下面是“恢复”按钮关联的槽函数代码:

```

void MainWindow::on_actRestore_triggered()
{ // 取消所有变换
    int cnt=scene->selectedItems().count(); // 选中图形项的个数
    if (cnt==1) // 针对单个图形项
    {
        QGraphicsItem* item=scene->selectedItems().at(0);
        item->setRotation(0); // 复位角度
        item->setScale(1.0); // 复位大小
        // item->resetTransform(); // 不起作用
    }
    else // 针对视图
        ui->view->resetTransform();
}

```

实际测试中发现 QGraphicsItem 的 resetTransform() 函数不起作用, 所以直接用 setRotation(0) 恢复原始角度, 用 setScale(1.0) 恢复原始大小。

(4) 叠放顺序

QGraphicsItem 的 zValue() 函数返回值表示图形项在 Z 轴的值, 若有多个图形项叠加在一起, zValue() 值最大的显示在最前面, zValue() 值最小的显示在最后面。用 setZValue() 函数可以设置这个属性值。下面是工具栏上的“前置”和“后置”按钮关联槽函数的代码。

```

void MainWindow::on_actEdit_Front_triggered()
{ // bring to front, 前置
    int cnt=scene->selectedItems().count();
    if (cnt>0)
    { // 只处理选中的第 1 个图形项
        QGraphicsItem* item=scene->selectedItems().at(0);
        item->setZValue(++frontZ);
    }
}

void MainWindow::on_actEdit_Back_triggered()

```

```

{ //bring to back, 后置
    int cnt=scene->selectedItems().count();
    if (cnt>0)
    { //只处理选中的第 1 个图形项
        QGraphicsItem* item=scene->selectedItems().at(0);
        item->setZValue(--backZ);
    }
}

```

frontZ 和 backZ 是在 MainWindow 类中定义的私有变量，专门用于存储叠放次序的编号。frontZ 只增加，所以每增加一次都是最大值，设置该值的图形项就可以显示在最前面；backZ 只减少，所以每减小一次都是最小值，设置该值的图形项就可以显示在最后面。

(5) 图形项的组合

可以将多个图形项组合为一个图形项，当做一个整体进行操作，如同 PowerPoint 软件里图形组合功能一样。使用 QGraphicsItemGroup 类实现多个图形项的组合，QGraphicsItemGroup 是 QGraphicsItem 的子类，所以，实质上也是一个图形项。下面是工具栏上的“组合”按钮的关联槽函数代码：

```

void MainWindow::on_actGroup_triggered()
{ //组合
    int cnt=scene->selectedItems().count();
    if (cnt>1)
    {
        QGraphicsItemGroup* group =new QGraphicsItemGroup; //创建组合
        scene->addItem(group); //添加到场景中
        for (int i=0;i<cnt;i++) //将选择的图形项添加到组合中
        {
            QGraphicsItem* item=scene->selectedItems().at(0);
            item->setSelected(false); //取消选择
            item->clearFocus(); //清除焦点状态
            group->addToGroup(item); //添加到组合
        }
        setItemProperty(group, "组合"); //设置特性
    }
}

```

当有多个图形项被选择时，程序创建一个 QGraphicsItemGroup 类型的对象 group，并添加到场景中，然后将选中的图形项逐一添加到 group 中。这样创建的 group 就是场景中的一个图形项，可以对其进行缩放、旋转等操作。

一个组合对象也可以被打散，使用 QGraphicsScene 的 destroyItemGroup() 函数可以打散一个组合对象。这个函数打散组合，删除组合对象，但是不删除原来组合里的图形项。下面是工具栏上的“打散”按钮关联的槽函数代码：

```

void MainWindow::on_actGroupBreak_triggered()
{ //break group, 打散组合
    int cnt=scene->selectedItems().count();
    if (cnt==1)
    {

```

```

    QGraphicsItemGroup *group;
    group=(QGraphicsItemGroup*)scene->selectedItems().at(0);
    scene->destroyItemGroup(group); //打散组合
}
}

```

这里假设在单击“打散”按钮时，选中的是一个组合对象，并没有做类型判断。

(6) 图形项的删除

使用 QGraphicsScene 的 removeItem()函数从场景中移除某个图形项，下面是工具栏上的“删除”按钮关联的槽函数代码：

```

void MainWindow::on_actEdit_Delete_triggered()
{ //删除所有选中的图形项
    int cnt=scene->selectedItems().count();
    for (int i=0;i<cnt;i++)
    {
        QGraphicsItem* item=scene->selectedItems().at(0);
        scene->removeItem(item); //移除图形项
        delete item; //删除对象，释放内存
    }
}

```

QGraphicsScene 的 removeItem()函数只是将图形项从场景中移除，并不会从内存中删除这个图形项，所以，还需要用 delete 删除这个图形项，以释放内存。添加到场景中的图形项最后无需手工删除，在场景被删除时，其中的图形项也会自动被删除。

7. 鼠标与键盘操作

在 MainWindow 类的构造函数里，我们将界面视图组件 view 的 4 个信号与 MainWindow 的 4 个自定义槽函数关联，用于实现鼠标和键盘操作。

(1) 鼠标移动

鼠标在视图上移动时，在状态栏显示光标处的视图坐标和场景坐标，on_mouseMovePoint()槽函数的代码如下：

视图组件 view 的 mouseMovePoint()信号与自定义槽函数 do_mouseMovePoint()关联，该槽函数的代码如下：

```

void MainWindow::do_mouseMovePoint(QPoint point)
{
    labViewCord->setText(QString::asprintf(
        "View 坐标: %d,%d", point.x(),point.y()));
    QPointF pointScene=ui->view->mapToScene(point); //转换到 Scene 坐标
    labSceneCord->setText(QString::asprintf(
        "Scene 坐标: %.0f,%.0f", pointScene.x(),pointScene.y()));
}

```

参数 point 是鼠标光标在视图上的坐标，用 QGraphicsView::mapToScene()函数可以将此坐标转换为

场景中的坐标。

（2）鼠标单击

视图组件 view 的 mouseClicked()信号与自定义槽函数 do_mouseClicked()关联。在视图上单击鼠标选中一个图形项时，程序会在状态栏上显示图形项的局部坐标，并提取其自定义信息并显示。该槽函数的代码如下：

```
void MainWindow::do_mouseClicked(QPoint point)
{
    QPointF pointScene=ui->view->mapToScene(point);    //转换到 Scene 坐标
    QGraphicsItem *item=NULL;
    item=scene->itemAt(pointScene,ui->view->transform());    //获取光标下的图形项
    if (item != NULL)
    {
        QPointF pointItem=item->mapFromScene(pointScene);    //转换为图形项的局部坐标
        labItemCord->setText(QString::asprintf(
            "Item 坐标: %.0f,%.0f",pointItem.x(),pointItem.y()));
        labItemInfo->setText(item->data(ItemDescription).toString()+" , ItemId="+
            item->data(ItemId).toString());
    }
}
```

程序首先将视图坐标 point 转换为场景中的坐标 pointScene，再利用 QGraphicsScene 的 itemAt()函数获得光标处的图形项。利用 QGraphicsItem 的 mapFromScene()函数将 pointScene 转换为图形项的局部坐标 pointItem。

在创建图形项时，我们使用 QGraphicsItem 的 setData()函数设置了 2 个自定义数据，这里用 data()函数提取图形项的这 2 个自定义数据，并进行显示。

（3）鼠标双击

当鼠标双击某个图形项时，我们希望根据图形项的类型，调用不同的对话框进行图形项的设置。例如，当图形项是矩形、圆形、梯形等有填充色的对象时打开一个颜色选择对话框，设置其填充颜色；当图形项是直线时，设置其线条颜色；当图形项是文字时，打开一个字体对话框，设置其字体。自定义槽函数 do_mouseDoubleClick()的代码如下：

```
void MainWindow::do_mouseDoubleClick(QPoint point)
{
    QPointF pointScene=ui->view->mapToScene(point);    //转换到 Scene 坐标
    QGraphicsItem *item=NULL;
    item=scene->itemAt(pointScene,ui->view->transform());    //获取光标下的图形项
    if (item == NULL)
        return;

    switch (item->type())    //图形项的类型
    {
    case    QGraphicsRectItem::Type:    //矩形框，QGraphicsRectItem 的枚举值 Type
    {
        QGraphicsRectItem *theItem =qgraphicsitem_cast<QGraphicsRectItem*>(item);
```

```

        setBrushColor(theItem);
        break;
    }
    case QGraphicsEllipseItem::Type: //椭圆或圆
    {
        QGraphicsEllipseItem *theItem = qgraphicsitem_cast<QGraphicsEllipseItem*>(item);
        setBrushColor(theItem);
        break;
    }
    case QGraphicsPolygonItem::Type: //梯形或三角形
    {
        QGraphicsPolygonItem *theItem = qgraphicsitem_cast<QGraphicsPolygonItem*>(item);
        setBrushColor(theItem);
        break;
    }
    case QGraphicsLineItem::Type: //直线，设置线条颜色
    {
        QGraphicsLineItem *theItem = qgraphicsitem_cast<QGraphicsLineItem*>(item);
        QPen pen = theItem->pen();
        QColor color = theItem->pen().color();
        color = QColorDialog::getColor(color, this, "选择线条颜色");
        if (color.isValid())
        {
            pen.setColor(color);
            theItem->setPen(pen);
        }
        break;
    }
    case QGraphicsTextItem::Type: //文字，设置字体
    {
        QGraphicsTextItem *theItem = qgraphicsitem_cast<QGraphicsTextItem*>(item);
        QFont font = theItem->font();
        bool ok = false;
        font = QFontDialog::getFont(&ok, font, this, "设置字体");
        if (ok)
            theItem->setFont(font);
        break;
    }
}

```

双击鼠标时，程序获取光标下的图形项 `item`，由于不知道具体是什么类型的图形项，`item` 定义为 `QGraphicsItem` 类型，即所有图形项的父类。

`QGraphicsItem` 的函数 `type()` 返回一个整数，表示图形项的具体类型，每一种图形项的 `type()` 函数值时不同的，但是都等于该类中的一个枚举值 `Type`。例如，`QGraphicsRectItem` 是矩形框图形项，它的函数 `type()` 返回值等于枚举值 `QGraphicsRectItem::Type`。

在自定义图形项类时，必须重定义函数 `type()`，并定义一个枚举值 `Type`。自定义图形项类的枚举值 `Type` 的值必须大于枚举值 `QGraphicsItem::UserType`（等于 65536），示例代码如下：

```
class CustomItem : public QGraphicsItem
```

```

{
public:
    enum { Type = UserType + 1 };          //定义枚举值 Type
    int type() const                       //重定义函数 type(), 返回值就是 Type
    {
        // 使得 QGraphicsItem_cast 可以使用这个图形项
        return Type;
    }
    ...
};

```

槽函数 `do_mouseDoubleClick()` 中根据 `item->type()` 的值判断图形项的类型，并根据类型进行处理。

若被双击的是一个 `QGraphicsLineItem` 图形项，就设置其线条颜色。设置线条颜色可以调用 `QGraphicsLineItem::setPen()` 函数，但是 `QGraphicsItem` 没有 `setPen()` 函数。所以，需要使用图形项的强制类型转换函数 `QGraphicsItem_cast()` 将 `item` 转换为 `QGraphicsLineItem` 类型的 `theItem`，即：

```
QGraphicsLineItem *theItem=QGraphicsItem_cast<QGraphicsLineItem*>(item);
```

然后就调用 `QColorDialog::getColor()` 函数选择颜色，设置为 `theItem` 的线条颜色。

`QGraphicsItem` 类也没有 `setFont()` 和 `setBrush()` 函数，当选择的图形项是 `QGraphicsTextItem` 类型时，需要将其强制转换为 `QGraphicsTextItem` 类型，调用字体选择对话框后，用 `QGraphicsTextItem::setFont()` 函数设置字体。

对于 `QGraphicsRectItem`、`QGraphicsEllipseItem` 或 `QGraphicsPolygonItem` 类型，可以调用这三个类的 `setBrush()` 函数设置填充颜色。这里用一个函数 `setBrushColor()` 为三种不同类型的对象进行填充颜色的设置。函数 `setBrushColor()` 并不是使用不同类型参数的 `overload` 函数，而是在文件 `mainwindow.cpp` 中定义的一个模板函数，该函数代码如下：

```

template<class T> void setBrushColor(T *item)
{
    QColor color=item->brush().color();
    color=QColorDialog::getColor(color,NULL,"选择填充颜色");
    if (color.isValid())
        item->setBrush(QBrush(color));
}

```

编译器会自动根据调用 `setBrushColor()` 的参数类型生成三个不同参数类型的函数，减少了代码的冗余性。

(4) 按键操作

在选中一个图形项之后，我们可以通过键盘按键实现一些快捷操作，例如缩放、旋转、移动等。视图组件 `view` 的 `keyPress()` 信号与自定义槽函数 `do_keyPress()` 关联，该槽函数代码如下：

```

void MainWindow::do_keyPress(QKeyEvent *event)
{
    if (scene->selectedItems().count() != 1)
        return;          //没有选中的图形项，或选中的多于 1 个
    QGraphicsItem *item=scene->selectedItems().at(0);
}

```

```
if (event->key()==Qt::Key_Delete)           //删除
    scene->removeItem(item);
else if (event->key()==Qt::Key_Space)        //顺时针旋转 90 度
    item->setRotation(90+item->rotation());
else if (event->key()==Qt::Key_PageUp)       //放大
    item->setScale(0.1+item->scale());
else if (event->key()==Qt::Key_PageDown)     //缩小
    item->setScale(-0.1+item->scale());
else if (event->key()==Qt::Key_Left)         //左移
    item->setX(-1+item->x());
else if (event->key()==Qt::Key_Right)        //右移
    item->setX(1+item->x());
else if (event->key()==Qt::Key_Up)           //上移
    item->setY(-1+item->y());
else if (event->key()==Qt::Key_Down)         //下移
    item->setY(1+item->y());
}
```

这段代码限定只有一个图形项被选中时才可以执行键盘操作。