**实验三**

**基于 Windows API 的前端 UI 开发(一)**

**【实验目的】**

1. 了解设备描述表和 GDI 对象的特性；

2. 掌握 MFC 中常用 GDI 对象的使用方法；

3. 基于 MFC 编写绘制个性化图形图像的应用程序。

**【实验任务】**

1. 进一步理解设备描述表的概念；

2. 了解掌握设备描述表的获取和使用方法；

3. 使用 MFC 设备描述表和 GDI 对象进行界面图形绘制。

**【实验设备】**

个人 PC，Win11操作系统，Qt5.15.2开发环境

**【计划学时】**

4-8 学时

**【实验内容】**

1. 使用设备描述表和 GDI 对象开发具有个性化图形显示的 MFC 应用程序，实现 多个不同颜色的矩形和圆形的叠加显示和动态控制，界面设计如图 1。

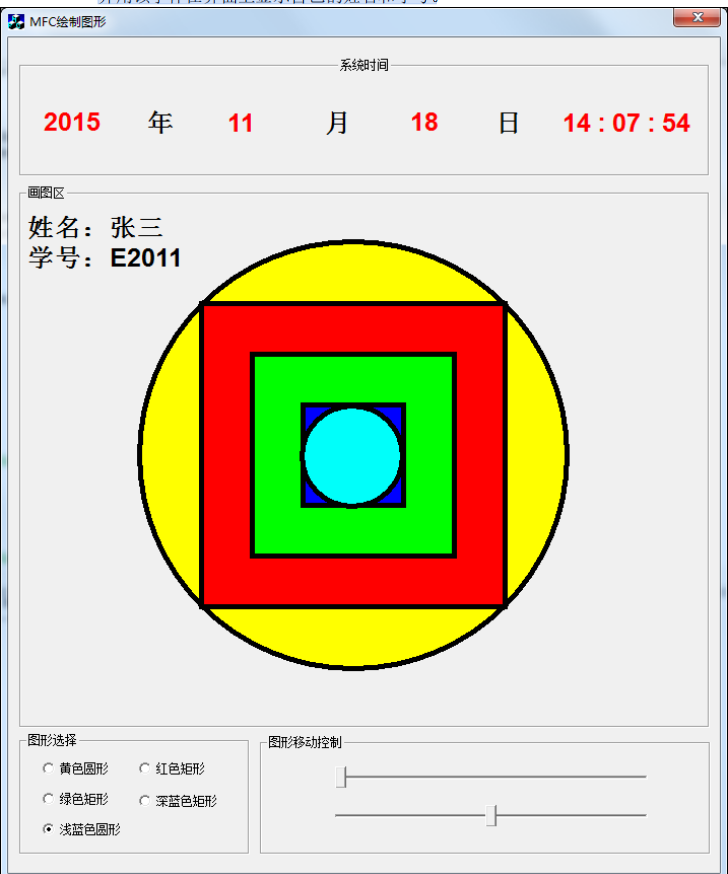


图 1 系统界面

功能要求：

1．重叠显示 5 个图形，按从大到小、从底部到顶部的顺序分别为：一个黄色圆形 (255,255,0)，一个红色矩形(255,0,0)，一个绿色矩形(0,255,0)，一个深蓝色矩形

(0,0,255)，一个蓝色圆形(0,255,255)，各个图形大小面积不一，重叠显示；

2．以上图形绘制在界面上的一个静态文本控件确定的区域内，图像超出该区域不显 示；

3．通过一组单选按钮可以选定某一个图形作为控制的对象，通过两个滑动条可以控制

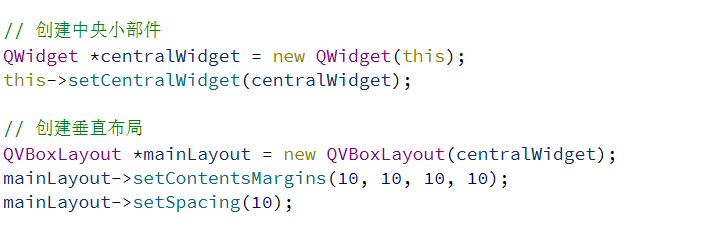
选定的图形进行两个方向上的动态移动；

4． 在主界面上方，设置一个区域用于动态显示时间。时间使用新创建的特定字体显示， 并用该字体在界面上显示自己的姓名和学号。

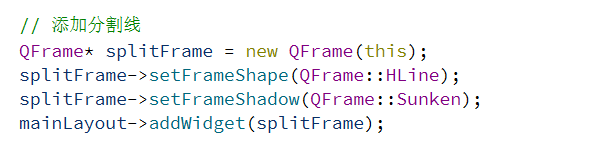
实现思路：

1. **整体布局**

对主窗口，采用垂直布局，自上而下分为三部分，分别为时间显示区域；图型绘制区域；图型操控区域



每个区域之间通过分隔线分割，分隔线通过QFrame设置

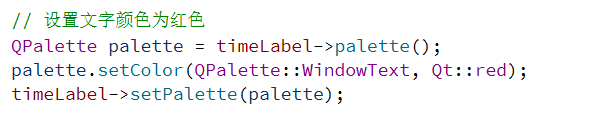


1. **时间显示**

通过QLabel显示时间，通过QFont设置字体样式

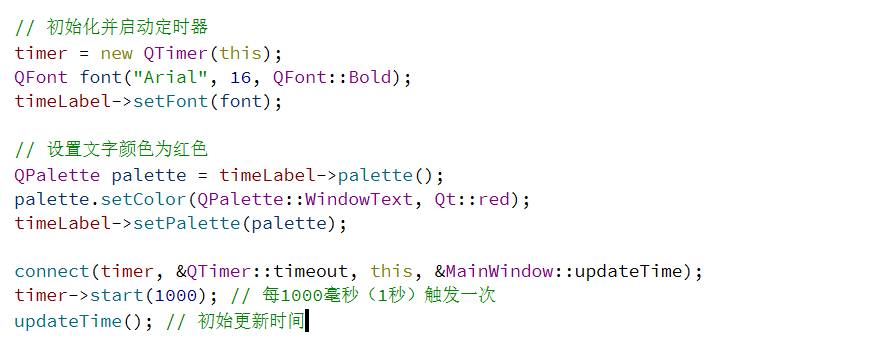


通过QPalette设置字体颜色



获取时间并显示：

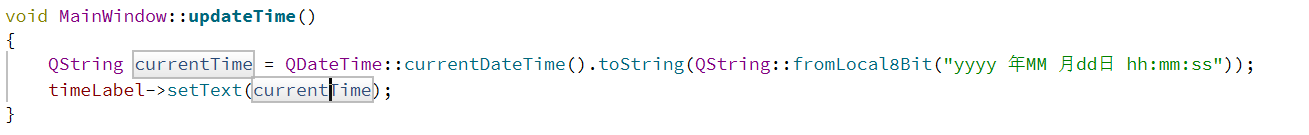
设置一个QTimer计时器，触发间隔设置为1000ms，通过connect将每次计时器结束的信号与自定义的槽函数updateTime绑定。调用一次updateTime显示初始时间。



updateTime槽函数的实现：

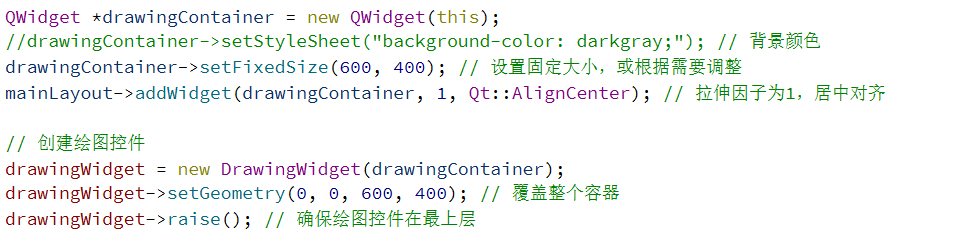
通过QDateTime::currentDateTime()获取当前时间，再通过toString方式设置时间格式为年-月-日小时:分钟:秒的形式

将标签值修改为时间字符串



1. **绘制图型**

首先在总布局中添加一个区域，在该区域中进行绘制。这里采用添加一个Widget窗口



创建一个绘图类DrawingWidget，继承自QWidget。在该类中，重写paintEvent虚函数。



枚举变量Shape给出了图形种类与数量

setShapeOffset函数用于设置图形相对于区域的偏移量

进行绘制：调用paintEvent函数。

PaintEvent函数会在每次界面创建或更新后自动调用，因此，如果想要更新图形，应手动调用update方法更新界面。

Paint Event重写：

void DrawingWidget::paintEvent(QPaintEvent \*event)

{

Q\_UNUSED(event);

QPainter painter(this);

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

// 更新绘图区域的中心点

initialCenter = QPoint(width() / 2, height() / 2);

// 绘制每个图形

// 1. 黄色圆形

QPoint pos = initialCenter + shapeOffsets[YellowCircle];

QRect yellowCircleRect(pos.x() - shapeSizes[YellowCircle].width()/2,

pos.y() - shapeSizes[YellowCircle].height()/2,

shapeSizes[YellowCircle].width(),

shapeSizes[YellowCircle].height());

painter.setBrush(QColor(255, 255, 0)); // 黄色

painter.setPen(Qt::NoPen);

painter.drawEllipse(yellowCircleRect);

// 2. 红色矩形

pos = initialCenter + shapeOffsets[RedRectangle];

QRect redRect(pos.x() - shapeSizes[RedRectangle].width()/2,

pos.y() - shapeSizes[RedRectangle].height()/2,

shapeSizes[RedRectangle].width(),

shapeSizes[RedRectangle].height());

painter.setBrush(QColor(255, 0, 0)); // 红色

painter.drawRect(redRect);

// 3. 绿色矩形

pos = initialCenter + shapeOffsets[GreenRectangle];

QRect greenRect(pos.x() - shapeSizes[GreenRectangle].width()/2,

pos.y() - shapeSizes[GreenRectangle].height()/2,

shapeSizes[GreenRectangle].width(),

shapeSizes[GreenRectangle].height());

painter.setBrush(QColor(0, 255, 0)); // 绿色

painter.drawRect(greenRect);

// 4. 深蓝色矩形

pos = initialCenter + shapeOffsets[DeepBlueRectangle];

QRect deepBlueRect(pos.x() - shapeSizes[DeepBlueRectangle].width()/2,

pos.y() - shapeSizes[DeepBlueRectangle].height()/2,

shapeSizes[DeepBlueRectangle].width(),

shapeSizes[DeepBlueRectangle].height());

painter.setBrush(QColor(0, 0, 255)); // 深蓝色

painter.drawRect(deepBlueRect);

// 5. 浅蓝色圆形

pos = initialCenter + shapeOffsets[LightBlueCircle];

QRect lightBlueCircleRect(pos.x() - shapeSizes[LightBlueCircle].width()/2,

pos.y() - shapeSizes[LightBlueCircle].height()/2,

shapeSizes[LightBlueCircle].width(),

shapeSizes[LightBlueCircle].height());

painter.setBrush(QColor(0, 255, 255)); // 浅蓝色

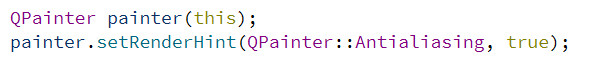
painter.setPen(Qt::NoPen);

painter.drawEllipse(lightBlueCircleRect);

}

解析：

首先，实例化QPainter对象，通过该对象进行绘制操作



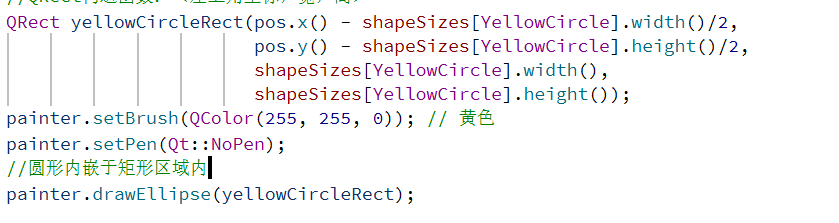
绘制逻辑：以 pos 为中心，减去半宽和半高得到左上角坐标，宽高保持不变。

创建绘制图形：

通过QRect创建一个矩形对象。构造函数的参数分别为（左上角x坐标，左 上角y坐标，宽，高）

通过setBrush方法设置画笔颜色

调用drawEllipse绘制圆形，传入的参数为绘制的区域，这里传入的是一个矩形，此时绘制的圆形内嵌于该矩形（若为正方形，则绘制出的图形是一个圆；若是普通矩形（长!=宽），绘制出的则会是一个矩形）



若绘制矩形，则调用drawRect方法

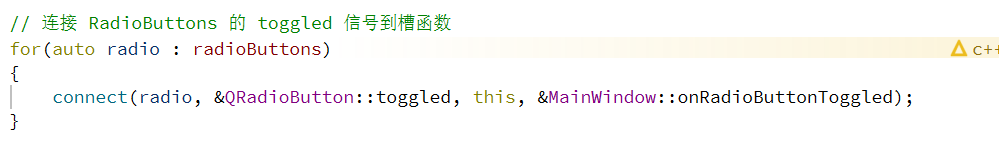
1. **设置滑动条控制图形移动：**

在总布局中新增一个水平布局，用于显示一组单选按钮以及滑动条

布局左侧添加一组RadioButton，表示选择的图形



绑定点击单选按钮对应的事件，选定被点击的图形



槽函数实现：

当选中某个图形时，不光要确定当前选中的图形（枚举变量中的），还要根据选中图形的偏移量，去更新当前两个滑动条的值。

!!!特别注意，在这个槽函数内更新slider的值时，一定要阻塞slider的的信号！！！否则会触发滑动条的槽函数，造成崩溃！！！



右侧添加两个slider滑动条，分别表示水平偏移量与竖直偏移量

滑动条通过setRange方法设置数值范围

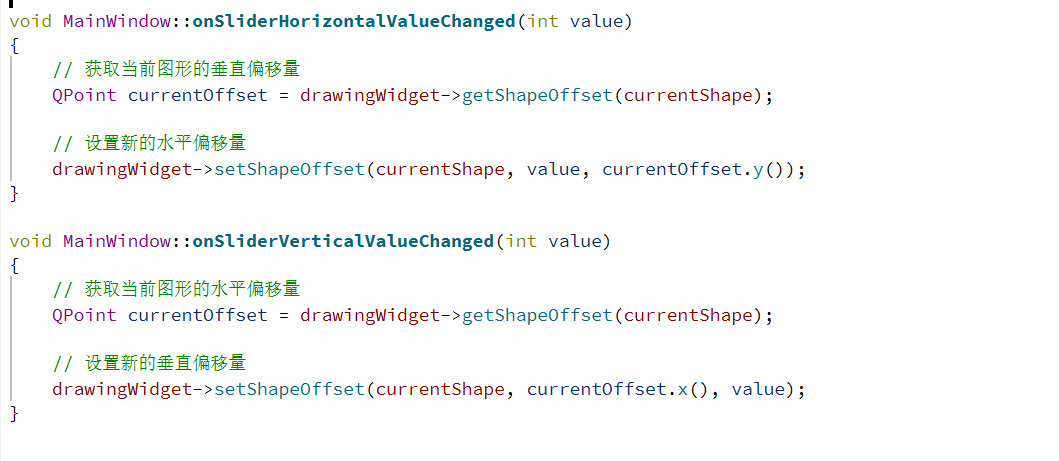


接着，对滑动条的改变数值事件设置相应的响应函数，使得选定的图形可以根据滑动条的偏移量进移动。



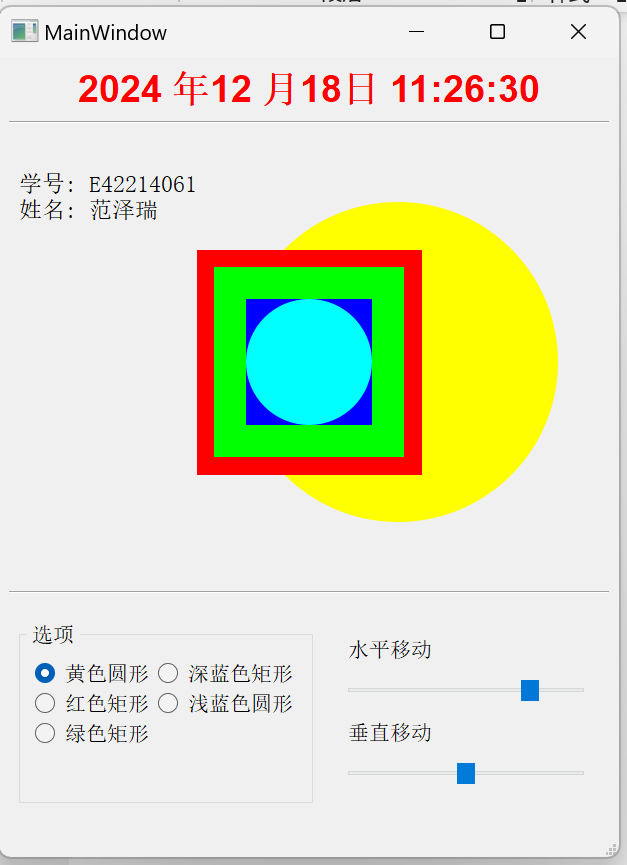
槽函数的实现：

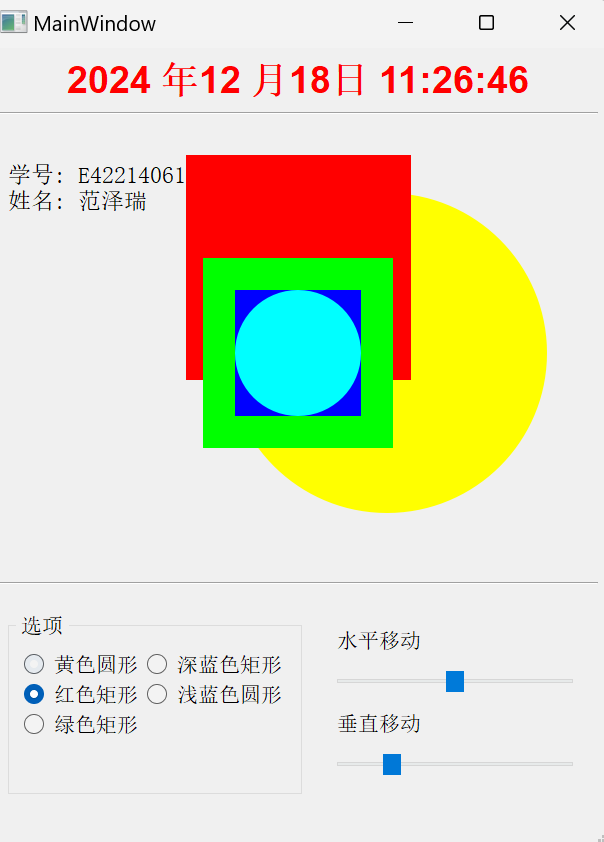
根据滑块值改变偏移量



**运行结果：**







**【实验总结】**

本次实验通过Qt实现了实时时间的获取以及图形的绘制。在Qt中，绘制图形需要自定义一个绘制类，并重写paintEvent虚函数。paintEvent函数会在每次界面创建或更新时调用。可以通过定义图形的偏移量，放大因子等参数，自由的调整绘制图形的区域，大小。以下是一些开发中的注意事项：

!!!特别注意，在这个槽函数内更新slider的值时，一定要阻塞slider的的信号！！！否则会触发滑动条的槽函数，造成崩溃！！！

上文中提到了，修改已经绑定槽函数的控件的属性时，一定要注意修改是否会触发控件的槽函数，如果错误触发，很可能会导致程序崩溃。因此，为确保程序稳定运行，在修改已绑定槽函数的控件属性时，最好先通过blockSignals(true)阻塞信号的触发，完成修改后再通过blockSignals（false）重新接收信号。