

QT 绘图

一、基本知识

注意!!! 遇到不懂的函数, 在 QT Creator 可以按F1查看QT手册!!!

Qt 的绘图操作分为两种, 一种是使用 **QPainter** 绘图, 一种是使用 **Graphics View** 架构绘图。

Qt 的二维绘图基本功能是使用 **QPainter** 在绘图设备上绘图, 绘图设备包括 **QWidget**、**QPixmap**等, 通过绘制一些基本的点、线、圆等基本形状组成自己需要的图形, 得到的图形是不可交互操作的图形。

Qt 还提供了 **Graphics View** 架构, 使用 **QGraphicsView**、**QGraphicsScene** 和各种 **QGraphicsItem** 类绘图, 在一个场景中可以绘制大量图件, 且每个图件是可选择、可交互的, 如同矢量图编辑软件那样可以操作每个图件。

二、QPainter 基本绘图

1. QPainter绘图系统

QPainter 是用来进行绘图操作的类, 一般的绘图设备包括 **QWidget**、**QPixmap**、**QImage** 等, 这些绘图设备为 **QPainter** 提供一个“画布”。

QWidget 类及其子类是最常用的绘图设备, 从**QWidget** 类继承的类都有 **paintEvent()** 事件, 要在设备上绘图, 只需重定义此事件并编写响应代码。一般都是创建一个 **QPainter** 对象用来获取绘图设备的接口, 然后就可以在绘图设备的“画布”上绘图了。

采用 **QPainter** 绘图时需要在绘图设备的 **paintEvent()** 事件里编写绘图的程序, 实现整个绘图过程。这种方法如同使用 **Windows** 的画图软件在绘图, 绘制的图形是**位图**, 这种方法适合于绘制复杂性不高的固定图形, 不能实现图件的选择、编辑、拖放、修改等功能。

QPainter类的基本架构

绘图基本思想：绘图操作一般在**paintEvent()**事件中完成，在其他事件中，可以通过改变成员变量和**update()**去刷新**paintEvent()**事件。

Widget.h 头文件

```
#include <QMouseEvent>
#include <QWidget>
#include <QPainter>
#include <QDebug>
#include "windows.h"
#include <iostream>

class widget : public QWidget
{
    Q_OBJECT
public:
    explicit widget(QWidget *parent = 0);
    ~widget();

protected:
    void paintEvent(QPaintEvent *event) override;
    void mouseMoveEvent(QMouseEvent *event) override;
    void mouseReleaseEvent(QMouseEvent *event) override;

private:
};
```

Widget.h 源文件

```
widget::widget(QWidget *parent) : QWidget(parent)
{
    setPalette(QPalette(Qt::white)); //设置窗口为白色背景
    setAutoFillBackground(true);
}
widget::~~widget()
{
}
```

```

void Widget::paintEvent(QPaintEvent *event)
{
    QPainter painter (this); //创建 QPainter 对象
    painter.setRenderHint(QPainter::Antialiasing);
    painter.setRenderHint(QPainter::TextAntialiasing);
    int w =this->width(); //绘图区宽度
    int H =this->height (); //绘图区高度
    QRect rect(w/4, H/4,w/2, H/2); //中间区域矩形框
    //设置画笔
    QPen pen;
    pen.setWidth(3); //线宽
    pen.setColor(Qt::red); //划线颜色
    pen.setStyle(Qt::SolidLine); //线的样式，实线、虚线等
    pen.setCapStyle(Qt::FlatCap); //线端点样式
    pen.setJoinStyle(Qt::BevelJoin); //线的连接点样式
    painter.setPen(pen);
    //设置画刷
    QBrush brush;
    brush.setColor(Qt::yellow); //画刷颜色
    brush.setStyle(Qt::SolidPattern); //画刷填充样式
    painter.setBrush(brush);
    //绘图
    painter.drawRect(rect);
}

```

QPainter 绘图的主要属性



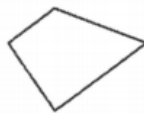
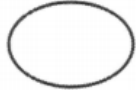


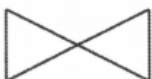
QPainter 在绘图设备上绘图，主要是绘制一些基本的图形元素，包括点、直线、圆形、矩形、曲线、文字等，控制这些绘图元素特性的主要是 QPainter 的3个属性，分别如下：

- **pen** 属性:是一个 **QPen** 对象，用于控制线条的颜色、宽度、线型等。
- **brush** 属性:是一个 **QBrush** 对象，用于设置一个区域的填充特性，可以设置填充颜色、填充方式、渐变特性等，还可以采用图片做材质填充。
- **font** 属性:是一个 **QFont** 对象，用于绘制文字时，设置文字的字体样式、大小等属性。

使用这 3 个属性基本就控制了绘图的基本特点，当然还有一些其他的功能结合使用，比如叠加模式、旋转和缩放等功能。

QPainter 绘制基本图形元件

QPainter 提供了很多绘制基本图形的功能，包括点、直线、椭圆、矩形、曲线等，由这些基本的图形可以构成复杂的图形。

函数名	功能和示例代码	示例图形
drawArc	画弧线，例如 <pre>QRect rect(W/4,H/4,W/2,H/2); int startAngle = 90 * 16; //起始 90° int spanAngle = 90 * 16; //旋转 90° painter.drawArc(rect, startAngle, spanAngle);</pre>	
drawChord	画一段弦，例如 <pre>QRect rect(W/4,H/4,W/2,H/2); int startAngle = 90 * 16; //起始 90° int spanAngle = 90 * 16; //旋转 90° painter.drawChord(rect, startAngle, spanAngle);</pre>	
drawConvexPolygon	根据给定的点画凸多边形 <pre>QPoint points[4]={ QPoint(5*W/12,H/4), QPoint(3*W/4,5*H/12), QPoint(5*W/12,3*H/4), QPoint(W/4,5*H/12), }; painter.drawConvexPolygon(points, 4);</pre>	
drawEllipse	画椭圆 <pre>QRect rect(W/4,H/4,W/2,H/2); painter.drawEllipse(rect);</pre>	
drawImage	在指定的矩形区域内绘制图片 <pre>QRect rect(W/4,H/4,W/2,H/2); QImage image(":/images/images/qt.jpg"); painter.drawImage(rect, image);</pre>	
drawLine	画直线 <pre>QLine line(W/4,H/4,W/2,H/2); painter.drawLine(line);</pre>	
drawLines	画一批直线 <pre>QRect rect(W/4,H/4,W/2,H/2); QVector<QLine> lines; lines.append(QLine(rect.topLeft(),rect.bottomRight())); lines.append(QLine(rect.topRight(),rect.bottomLeft())); lines.append(QLine(rect.topLeft(),rect.bottomLeft())); lines.append(QLine(rect.topRight(),rect.bottomRight())); painter.drawLines(lines);</pre>	

drawPath	绘制由 QPainterPath 对象定义的路线	
	<pre>QRect rect(W/4,H/4,W/2,H/2); QPainterPath path; path.addEllipse(rect); path.addRect(rect); painter.drawPath(path);</pre>	
drawPie	绘制扇形	
	<pre>QRect rect(W/4,H/4,W/2,H/2); int startAngle = 40 * 16; //起始 40° int spanAngle = 120 * 16; //旋转 120° painter.drawPie(rect, startAngle, spanAngle);</pre>	
drawPixmap	绘制 QPixmap 图片	
	<pre>QRect rect(W/4,H/4,W/2,H/2); QPixmap pixmap(":images/images/qt.jpg"); painter.drawPixmap(rect, pixmap);</pre>	
drawPoint	画一个点	
	<pre>painter.drawPoint(QPoint(W/2,H/2));</pre>	
drawPoints	画一批点	
	<pre>QPoint points[]={ QPoint(5*W/12,H/4), QPoint(3*W/4,5*H/12), QPoint(2*W/4,5*H/12) }; painter.drawPoints(points, 3);</pre>	
drawPolygon	画多边形, 最后一个点会和第一个点闭合	
	<pre>QPoint points[]={ QPoint(5*W/12,H/4), QPoint(3*W/4,5*H/12), QPoint(5*W/12,3*H/4), QPoint(2*W/4,5*H/12) }; painter.drawPolygon(points, 4);</pre>	
drawPolyline	画多点连接的线, 最后一个点不会和第一个点连接	
	<pre>QPoint points[]={ QPoint(5*W/12,H/4), QPoint(3*W/4,5*H/12), QPoint(5*W/12,3*H/4), QPoint(2*W/4,5*H/12) }; painter.drawPolyline(points, 4);</pre>	
drawRect	画矩形	
	<pre>QRect rect(W/4,H/4,W/2,H/2); painter.drawRect(rect);</pre>	
drawRoundedRect	画圆角矩形	
	<pre>QRect rect(W/4,H/4,W/2,H/2); painter.drawRoundedRect(rect,20,20);</pre>	
drawText	绘制文本, 只能绘制单行文字, 字体的大小等属性由 QPainter::font() 决定。	
	<pre>QRect rect(W/4,H/4,W/2,H/2); QFont font; font.setPointSize(30); font.setBold(true); painter.setFont(font); painter.drawText(rect,"Hello,Qt");</pre>	
eraseRect	擦除某个矩形区域, 等效于用背景色填充该区域	
	<pre>QRect rect(W/4,H/4,W/2,H/2); painter.eraseRect(rect);</pre>	
fillPath	填充某个 QPainterPath 定义的绘图路径, 但是轮廓线不显示	
	<pre>QRect rect(W/4,H/4,W/2,H/2); QPainterPath path; path.addEllipse(rect); path.addRect(rect); painter.fillPath(path,Qt::red);</pre>	
fillRect	填充一个矩形, 无边框线	
	<pre>QRect rect(W/4,H/4,W/2,H/2); painter.fillRect(rect,Qt::green);</pre>	

```
void Widget::paintEvent(QPaintEvent *event)
{
    QPainter painter (this); //创建 QPainter 对象
    painter.setRenderHint(QPainter::Antialiasing);
    painter.setRenderHint(QPainter::TextAntialiasing);
    int w =this->width(); //绘图区宽度
    int H =this->height (); //绘图区高度
    QRect rect(w/4, H/4,w/2, H/2); //中间区域矩形框
    //设置画笔
    QPen pen;
    pen.setWidth(3); //线宽
    pen.setColor(Qt::red); //划线颜色
    pen.setStyle(Qt::SolidLine); //线的样式，实线、虚线等
    pen.setCapStyle(Qt::FlatCap); //线端点样式
    pen.setJoinStyle(Qt::BevelJoin); //线的连接点样式
    painter.setPen(pen);
    //设置画刷
    QBrush brush;
    brush.setColor(Qt::yellow); //画刷颜色
    brush.setStyle(Qt::SolidPattern); //画刷填充样式
    painter.setBrush(brush);
    //绘图
    painter.drawRect(rect);
}
```

可以看出，一般的绘图步骤是：

- ① 创建 QPainter 对象；
- ② 获取 窗口 的高度和宽度；
- ③ 设定 矩形/线/点/圆形 等等的范围；
- ④ 设定 pen/brush/font 的属性；

QPainterPath 的使用

QPainterPath 是一系列绘图操作的顺序集合，便于重复使用。一个 PainterPath 由许多基本的绘图操作组成，如绘图点移动、划线、画圆、画矩形等，一个闭合的 PainterPath 是终点和起点连接起来的绘图路径。使用 QPainterPath 的优点是绘制某些复杂形状时只需创建一个 PainterPath，然后调用 QPainter::drawPath() 就可以重复使用。

例如绘制一个复杂的星星图案需要多次调用 lineTo() 函数，定义一个 QPainterPath 类型的变量 path 记录这些绘制过程，再调用 drawPath(path) 就可以完成星型图案的绘制。

QPainterPath 演示实例

```
//生成五角星的 5 个顶点的坐标，假设原点在五角星中心
qreal R=100; //半径
const qreal Pi=3.14159;
qreal deg=Pi*72/180;
QPoint points[5]={
    QPoint(R, 0),QPoint(R*std::cos(deg),-R*std::sin(deg)),
    QPoint(R*std::cos(2*deg),-R*std::sin(2*deg)),
    QPoint(R*std::cos(3*deg),-R*std::sin(3*deg)),
    QPoint(R*std::cos(4*deg),-R*std::sin(4*deg)),
};

//设置画笔
QPenpenLine;
penLine.setWidth(2); //线宽
penLine.setColor(Qt::blue); //划线颜色
penLine.setStyle(Qt::SolidLine); //线的类型
penLine.setCapStyle(Qt::FlatCap); //线端点样式
penLine.setJoinStyle(Qt::BevelJoin); //线的连接点样式
painter.setPen(penLine);

//设置画刷
QBrushbrush;
brush.setColor(Qt::yellow); //画刷颜色
brush.setStyle(Qt::SolidPattern); //画刷填充样式
painter.setBrush(brush);

//设计绘制五角星的 PainterPath,以便重复使用
QPainterPath starPath;
starPath.moveTo(points[0]);
starPath.lineTo(points[2]);
starPath.lineTo(points[4]);
```

```

starPath.lineTo(points[1]);
starPath.lineTo(points[3]);
starPath.closeSubpath(); //闭合路径，最后一个点与第一个点相连
starPath.addText(points[0], font, "0"); //显示端点编号
starPath.addText(points[1], font, "1");
starPath.addText(points[2], font, "2");
starPath.addText(points[3], font, "3");
starPath.addText(points[4], font, "4");
//绘图
painter.save(); //保存坐标状态
painter.translate(100, 120); //平移
painter.drawPath(starPath); //画星星

```

2. QPainter 坐标系统和坐标变换

逻辑坐标系统是 QPainter 绘制图像的地方，也是我们常说的“窗口”(Window)，而物理坐标系统默认是绘图设备区域，也是我们常说的“视口”(Viewport)。我们的绘图是在逻辑坐标上进行的，通过setWindow决定展示所绘图片的一部分或者全部。通过setViewport决定放在绘图设备的那个区域进行展示。

分组	函数原型	功能
坐标变换	void translate(qreal dx, qreal dy)	坐标系统平移一定的偏移量，坐标原点平移到新的点
	void rotate(qreal angle)	坐标系统顺时针旋转一个角度
	void scale(qreal sx, qreal sy)	坐标系统缩放
	void shear(qreal sh, qreal sv)	坐标系统做扭转变换
状态保存与恢复	void save()	保存 painter 当前的状态，就是将当前状态压入堆栈
	void restore()	恢复上一次状态，就是从堆栈中弹出上次状态
	void resetTransform()	复位所有的坐标变换

QPainter 坐标变换演示实例

```

void widget_x::paintEvent(QPaintEvent *event)
{

    Q_UNUSED(event);

    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing, true);
    int width = this->width();
    int height = this->height();
    int side = qMin(width, height);

```



```

    painter.setViewport((width - side) / 2, (height - side) / 2,
side, side);
    painter.setWindow(-50, -51, 100, 102);

    // 画一个圆
    painter.drawEllipse(-50, -50, 100, 100);

    // 画 360 个刻度
    for (int i = 0; i < 360; i+=4)
    {
        painter.save();
        painter.rotate(i);
        painter.drawLine(0, -48, 0, -50);
        painter.restore();
    }
}

```

三、QGraphicsView 基本绘图

Qt 为绘制复杂的可交互图形提供了 Graphics View 绘图架构，是一种基于图形项 (GraphicsItem) 的模型/视图模式。使用 Graphics View 架构可以绘制复杂的有几万个基本图形元件的图形，并且每个图形元件是可选择、可拖放和修改的，类似于 **矢量** 绘图软件的绘图功能。Graphics View 架构主要由 3 个部分组成，即场景、视图和图形项，其构成的 Graphics View 绘图系统结构如图所示。

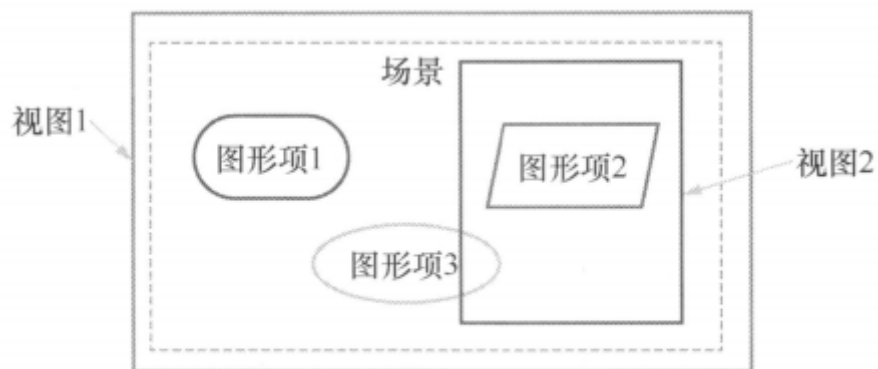


图 8-16 视图、场景、图形项的关系

