第8篇 (2D绘图) 绘制简单图形

代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/Painter01

目录

一、绘制一条直线

1．新建Qt Gui应用，项目名称为`painter_1`，类信息界面不用修改，即类名为`MainWindow`，基类为`QMainWindow`。

2．在`mainwindow.h`文件中添加重绘事件处理函数的声明：

```
protected:
void paintEvent(QPaintEvent *);
```

所有的绘制操作都要在这个函数里面完成。

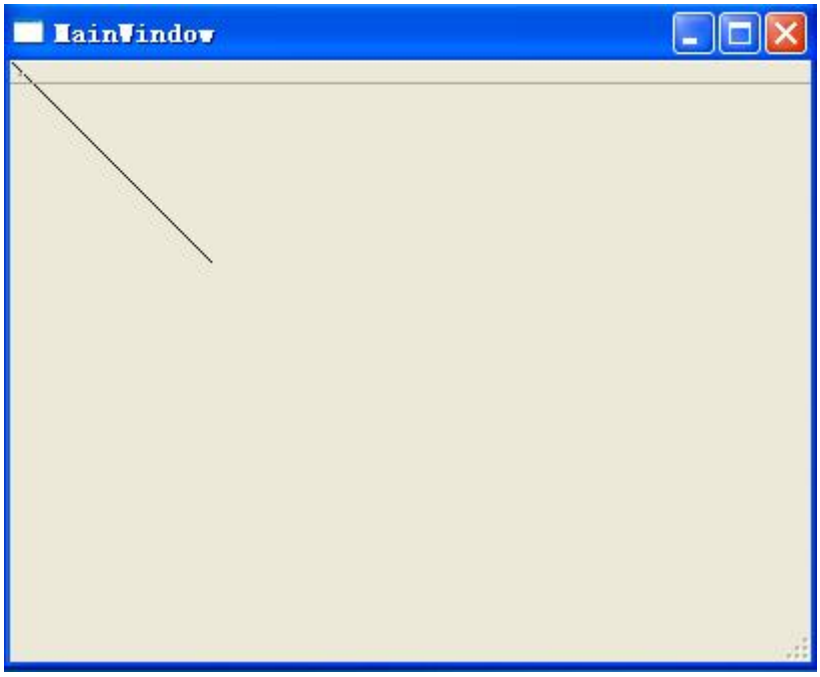
3．下面到`mainwindow.cpp`文件中先需要添加头文件包含：

```
#include <QPainter>
```

然后添加该函数的定义：

```
void MainWindow::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    painter.drawLine(QPointF(0, 0), QPointF(100, 100));
}
```

这里首先为该部件创建了一个`QPainter`对象，用于后面的绘制。然后使用`drawLine()`函数绘制了一条线段，线段的起点为`(0, 0)`，终点为`(100, 100)`，这里的单位是像素。效果如下图所示。



可以看到，在Qt窗口里面，`(0, 0)`点就是窗口的左上角，但这里是不包含外边框的。而在`MainWindow`主窗口里面绘制时，左上角并不是指中心区域的左上角，而是包含了工具栏。

4．我们将光标定位到`QPainter`类名上，然后按下键盘上的`F1`按键，这时会自动跳转到该类的帮助页面。当然，也可以到帮助模式，直接索引查找该类名。在帮助里面我们可以看到很多相关的绘制函数，如下图所示。

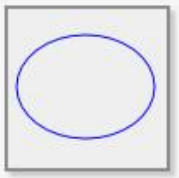
```
void drawArc ( const QRectF & rectangle, int startAngle, int spanAngle )
void drawArc ( const QRect & rectangle, int startAngle, int spanAngle )
void drawArc ( int x, int y, int width, int height, int startAngle, int spanAngle )
void drawChord ( const QRectF & rectangle, int startAngle, int spanAngle )
void drawChord ( const QRect & rectangle, int startAngle, int spanAngle )
void drawChord ( int x, int y, int width, int height, int startAngle, int spanAngle )
void drawConvexPolygon ( const QPointF * points, int pointCount )
void drawConvexPolygon ( const QPoint * points, int pointCount )
void drawConvexPolygon ( const QPolygonF & polygon )
void drawConvexPolygon ( const QPolygon & polygon )
void drawEllipse ( const QRectF & rectangle )
```

5．我们任意点击一个函数名，就会跳转到该函数的介绍段落。例如我们点击`drawEllipse()`函数，就跳转到了该函数的介绍处，上面还提供了一个例子。如下图所示。我们可以直接将例子里面的代码复制到`paintEvent()`函数里面，测试效果。

```
void QPainter::drawEllipse ( const QRectF & rectangle )
```

Draws the ellipse defined by the given *rectangle*.

A filled ellipse has a size of *rectangle*.size(). A stroked ellipse has a size of *rectangle*.size() plus the pen width.



```
QRectF rectangle(10.0, 20.0, 80.0, 60.0);

QPainter painter(this);
painter.drawEllipse(rectangle);
```

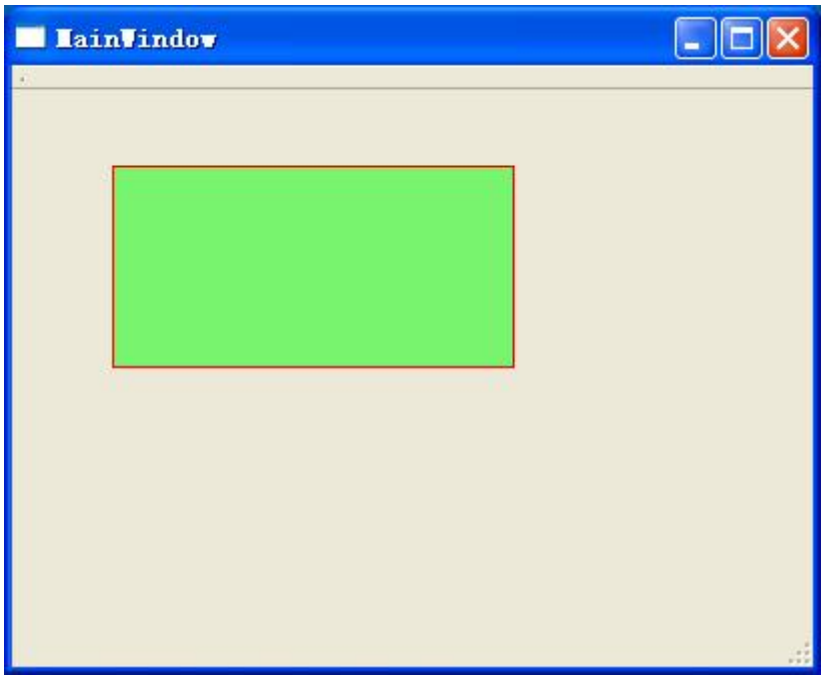
二、画笔和画刷

1．我们先将`paintEvent()`函数的内容更改如下：

```
void MainWindow::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    QPen pen; //画笔
    pen.setColor(QColor(255, 0, 0));
    QBrush brush(QColor(0, 255, 0, 125)); //画刷
    painter.setPen(pen); //添加画笔
    painter.setBrush(brush); //添加画刷
    painter.drawRect(50, 50, 200, 100); //绘制矩形
}
```

这里分别新建了一个画笔`QPen`，和画刷`QBrush`。其中画笔使用了`setColor()`函数为其设置了颜色，而画刷是在构建的时候直接为其设置的颜色。这里的颜色都是使用的`QColor`类提供的，里面如果是三个参数，那么分别是红、绿、蓝分量的值，也就是经常说的rgb，取值范围都是0-255，比如这里的`(255, 0, 0)`就表明红色分量为255，其他分量为0，那么出来就是红色。如果是四个参数，最后一个参数`alpha`是设置透明度的，取值范围也是0-255,0表示完全透明，而255表示完全不透明。

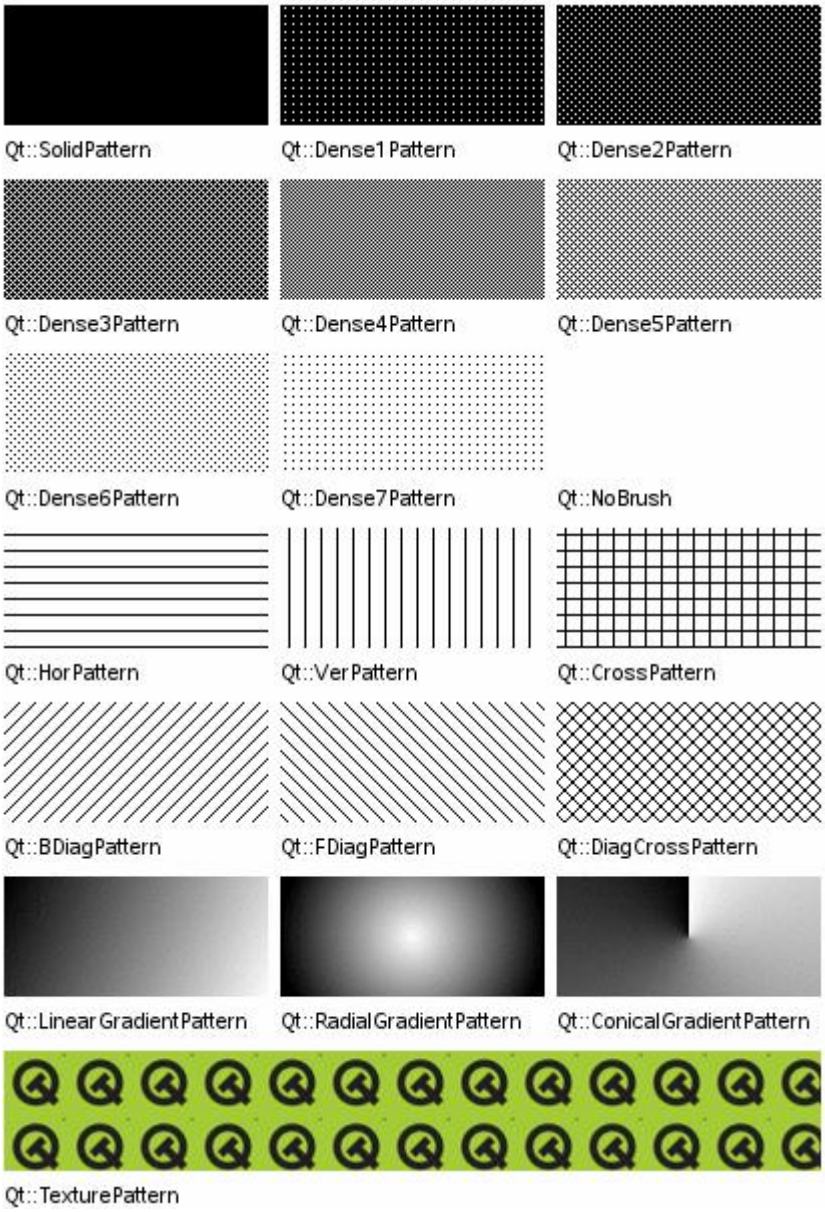
然后我们将画笔和画刷设置到了`painter`上，并使用`drawRect()`绘制了一个矩形，其左上角顶点在`(50, 50)`，宽为200，高为100。运行程序，效果如下图所示。



2．画笔还有许多其他的设置，可以查看该类的帮助文档。例如，可以使用`pen.setStyle()`来设置画笔样式，可用的画笔样式如下图所示。

Qt provides several built-in styles represented by the Qt::PenStyle enum:

3．画刷也有很多其他设置，这个也可以查看其帮助文档。在Qt中为画刷提供了一些可用的样式，可以使用`setStyle()`函数来设置。如下图所示。



这里面包含了渐变填充效果，这个会在下一节讲到。

4．下面我们写一个简单的例子演示一下。将`paintEvent()`函数更改如下：

```
void MainWindow::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    QPen pen(Qt::DotLine);
    QBrush brush(Qt::blue);
    brush.setStyle(Qt::HorPattern);
    painter.setPen(pen);
    painter.setBrush(brush);
    painter.drawRect(50,50,200,200);
}
```

这里的颜色使用了Qt预定义的颜色，可以在帮助中索引`Qt::GlobalColor`关键字查看。如下图所示。

enum Qt::GlobalColor

Qt's predefined QColor objects:

Constant	Value	Description
Qt::white	3	White (#ffffff)
Qt::black	2	Black (#000000)
Qt::red	7	Red (#ff0000)
Qt::darkRed	13	Dark red (#800000)
Qt::green	8	Green (#00ff00)
Qt::darkGreen	14	Dark green (#008000)
Qt::blue	9	Blue (#0000ff)
Qt::darkBlue	15	Dark blue (#000080)
Qt::cyan	10	Cyan (#00ffff)
Qt::darkCyan	16	Dark cyan (#008080)
Qt::magenta	11	Magenta (#ff00ff)
Qt::darkMagenta	17	Dark magenta (#800080)
Qt::yellow	12	Yellow (#ffff00)
Qt::darkYellow	18	Dark yellow (#808000)
Qt::gray	5	Gray (#a0a0a4)
Qt::darkGray	4	Dark gray (#808080)
Qt::lightGray	6	Light gray (#c0c0c0)
Qt::transparent	19	a transparent black value (i.e., QColor(0, 0, 0, 0))
Qt::color0	0	0 pixel value (for bitmaps)
Qt::color1	1	1 pixel value (for bitmaps)

现在运行程序，效果如下图所示。

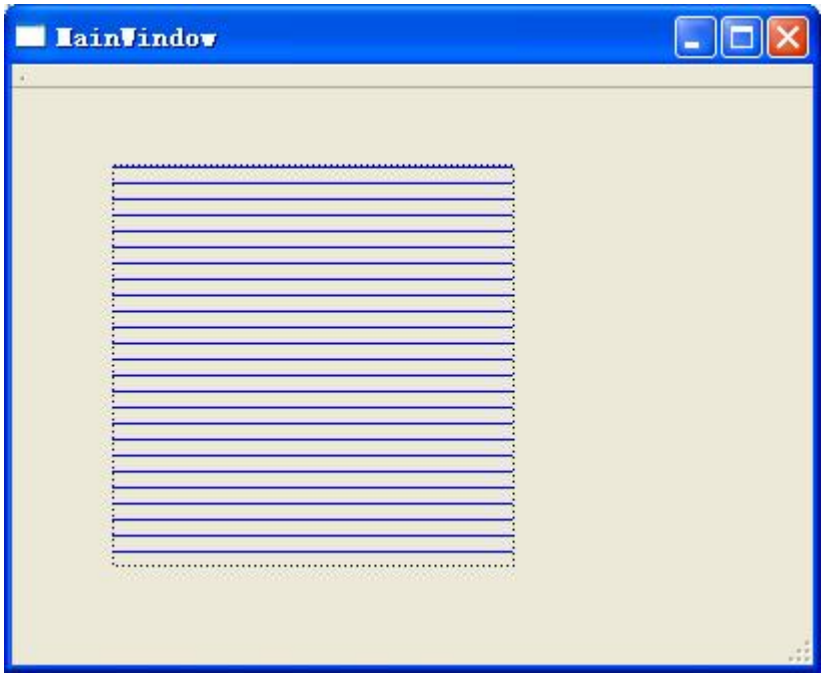
enum Qt::GlobalColor

Qt's predefined QColor objects:

Constant	Value	Description
Qt::white	3	White (#ffffff)
Qt::black	2	Black (#000000)
Qt::red	7	Red (#ff0000)
Qt::darkRed	13	Dark red (#800000)
Qt::green	8	Green (#00ff00)
Qt::darkGreen	14	Dark green (#008000)
Qt::blue	9	Blue (#0000ff)
Qt::darkBlue	15	Dark blue (#000080)
Qt::cyan	10	Cyan (#00ffff)
Qt::darkCyan	16	Dark cyan (#008080)
Qt::magenta	11	Magenta (#ff00ff)
Qt::darkMagenta	17	Dark magenta (#800080)
Qt::yellow	12	Yellow (#ffff00)
Qt::darkYellow	18	Dark yellow (#808000)
Qt::gray	5	Gray (#a0a0a4)
Qt::darkGray	4	Dark gray (#808080)
Qt::lightGray	6	Light gray (#c0c0c0)
Qt::transparent	19	a transparent black value (i.e., QColor(0, 0, 0, 0))
Qt::color0	0	0 pixel value (for bitmaps)
Qt::color1	1	1 pixel value (for bitmaps)

三、绘制弧线

为了帮助大家学习，这里再举一个绘制弧线的例子，其实在帮助文档中已经给出了这个例子。如下图所示。



我们将`paintEvent()`函数更改如下：

```
void MainWindow::paintEvent(QPaintEvent *)
{
    QRectF rectangle(10.0, 20.0, 80.0, 60.0); //矩形
    int startAngle = 30 * 16;     //起始角度
    int spanAngle = 120 * 16;   //跨越度数
    QPainter painter(this);
    painter.drawArc(rectangle, startAngle, spanAngle);
}
```

这里要说明的是，画弧线时，角度被分成了十六分之一，就是说，要想为30度，就得是`30*16`。它有起始角度和跨度，还有位置矩形，要想画出自己想要的弧线，就要有一定的几何知识了。这里就不再祥述。