QPainter ::setPen() 可以对画笔进行设置，包括颜色、宽度、线型、拐点风格等。

画刷用来填充几何形状的图案， QPainter ::setBrush() 可以对画刷进行设置，包括颜色和纹理风格。

绘制文本时需要对字体进行设置， QPainter ::setFont() 对字体进行设置，包括字体族和磅值。

setRenderHint反走样：

void PaintedWidget::paintEvent(QPaintEvent \*event)   
http://devbean.blog.51cto.com/images/editer/InBlock.gif{   
http://devbean.blog.51cto.com/images/editer/InBlock.gif        QPainter painter(this);   
http://devbean.blog.51cto.com/images/editer/InBlock.gif        painter.setPen(QPen(Qt::black, 5, Qt::DashDotLine, Qt::RoundCap));   
http://devbean.blog.51cto.com/images/editer/InBlock.gif        painter.setBrush(Qt::yellow);   
http://devbean.blog.51cto.com/images/editer/InBlock.gif        painter.drawEllipse(50, 150, 200, 150);   
http://devbean.blog.51cto.com/images/editer/InBlock.gif   
http://devbean.blog.51cto.com/images/editer/InBlock.gif        painter.setRenderHint(QPainter::Antialiasing, true);   
http://devbean.blog.51cto.com/images/editer/InBlock.gif        painter.setPen(QPen(Qt::black, 5, Qt::DashDotLine, Qt::RoundCap));   
http://devbean.blog.51cto.com/images/editer/InBlock.gif        painter.setBrush(Qt::yellow);   
http://devbean.blog.51cto.com/images/editer/InBlock.gif        painter.drawEllipse(300, 150, 200, 150);   
http://devbean.blog.51cto.com/images/editer/InBlock.gif}

Antialiasing (抗锯齿处理) 简单地说主要是应用调色技术将图形边缘的"锯齿"缓和，边缘更平滑。抗锯齿是相对来来说较复杂的技术，一直是高档加速卡的一个主要特征。目前的低档3D加速卡大多不支持反锯齿。 跟据全国科学技术名词审定委员会:http://www.term.gov.cn/index.jsp的权威标准 anti-aliasing应被翻译为"图形保真"

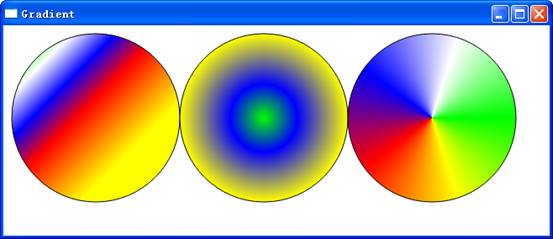
DashDotLine点画线，DashDotLine，正如同其名字所示，是一个短线和一个点相间的类型；

**RoundCap**，也就是圆形笔帽。然后我们使用一个黄色的画刷填充，画了一个椭圆。

渐变：QLinearGradient:条形渐变 QRadialGradient:圆形渐变

1. **void** MainWindow::paintEvent(QPaintEvent \*)
2. {
3. QPainter painter(**this**);
4. QLinearGradient linearGradient(100,150,300,150); //从点（100，150）开始到点（300，150）结束，确定一条直线
5. linearGradient.setColorAt(0,Qt::red);
6. linearGradient.setColorAt(0.2,Qt::black);
7. linearGradient.setColorAt(0.4,Qt::yellow);
8. linearGradient.setColorAt(0.6,Qt::white);
9. linearGradient.setColorAt(0.8,Qt::green);
10. linearGradient.setColorAt(1,Qt::blue); //将直线开始点设为0，终点设为1，然后分段设置颜色
11. painter.setBrush(linearGradient);
12. painter.drawRect(100,100,200,100);//绘制矩形，线性渐变线正好在矩形的水平中心线上
13. }
14. 
15. **void** MainWindow::paintEvent(QPaintEvent \*)
16. {
17. QRadialGradient radialGradient(200,100,100,200,100);
18. //其中参数分别为圆形渐变的圆心（200，100），半径100，和焦点（200，100）
19. //这里让焦点和圆心重合，从而形成从圆心向外渐变的效果
20. radialGradient.setColorAt(0,Qt::black);
21. radialGradient.setColorAt(1,Qt::yellow);
22. //渐变从焦点向整个圆进行，焦点为起始点0，圆的边界为1
23. QPainter painter(**this**);
24. painter.setBrush(radialGradient);
25. painter.drawEllipse(100,0,200,200); //绘制圆，让它正好和上面的圆形渐变的圆重合
26. }





**[cpp]** [view plain](http://blog.csdn.net/wangwei890702/article/details/8552482) [copy](http://blog.csdn.net/wangwei890702/article/details/8552482)

1. #include<QtGui/QApplication>
2. #include"gradient\_test.h"
3. intmain(intargc,**char**\*argv[])
4. {
5. QApplicationa(argc,argv);
6. gradient\_testw;
7. w.show();
8. returna.exec();
9. }
10. //gradient\_test.h:
11. #ifndefGRADIENT\_TEST\_H
12. #defineGRADIENT\_TEST\_H
13. #include<QtGui/QWidget>
14. #include<QPainter>
15. #include<QStyleOption>
16. #include<QResizeEvent>
17. classgradient\_test:publicQWidget
18. {
19. Q\_OBJECT
20. **public**:
21. explicitgradient\_test(QWidget\*parent=0);
22. virtualQSizesizeHint()**const**;
23. **protected**:
24. voidpaintEvent(QPaintEvent\*event);
25. voidresizeEvent(QResizeEvent\*event);
26. **private**:
27. QSizeinitSize;
28. QImagewheel;
29. voiddrawLine(constQSize&newSize);
30. };
31. #endif//GRADIENT\_TEST\_H

34. //gradient\_test.cpp:
35. #include"gradient\_test.h"
36. gradient\_test::gradient\_test(QWidget\*parent)
37. :QWidget(parent),initSize(300,300)
38. {
39. resize(650,250);    //窗口大小
40. }
41. voidgradient\_test::drawLine(constQSize&newSize)
42. {
43. QPainterpainter(&wheel);   //wheel作为画图对象？
44. painter.setRenderHint(QPainter::Antialiasing);  //消除锯齿
45. wheel.fill(Qt::white);
46. //线性渐变
47. QLinearGradientlinearGradient(20,20,150,150);
48. //创建了一个QLinearGradient对象实例，参数为起点和终点坐标，可作为颜色渐变的方向
49. //painter.setPen(Qt::NoPen);
50. linearGradient.setColorAt(0.0,Qt::green);
51. linearGradient.setColorAt(0.2,Qt::white);
52. linearGradient.setColorAt(0.4,Qt::blue);
53. linearGradient.setColorAt(0.6,Qt::red);
54. linearGradient.setColorAt(1.0,Qt::yellow);
55. painter.setBrush(QBrush(linearGradient));
56. painter.drawEllipse(10,10,200,200);
57. //前面为左边，后面两个参数为横轴和纵轴,上面的四行分别设置渐变的颜色和路径比例
59. //辐射渐变
60. QRadialGradientradialGradient(310,110,100,310,110);
61. //创建了一个QRadialGradient对象实例，参数分别为中心坐标，半径长度和焦点坐标,如果需要对称那么中心坐标和焦点坐标要一致
62. radialGradient.setColorAt(0,Qt::green);
63. //radialGradient.setColorAt(0.2,Qt::white);
64. radialGradient.setColorAt(0.4,Qt::blue);
65. //radialGradient.setColorAt(0.6,Qt::red);
66. radialGradient.setColorAt(1.0,Qt::yellow);
67. painter.setBrush(QBrush(radialGradient));
68. painter.drawEllipse(210,10,200,200);//在相应的坐标画出来
69. //弧度渐变
70. QConicalGradientconicalGradient(510,110,0);
71. //创建了一个QConicalGradient对象实例，参数分别为中心坐标和初始角度
72. conicalGradient.setColorAt(0,Qt::green);
73. conicalGradient.setColorAt(0.2,Qt::white);
74. conicalGradient.setColorAt(0.4,Qt::blue);
75. conicalGradient.setColorAt(0.6,Qt::red);
76. conicalGradient.setColorAt(0.8,Qt::yellow);
77. conicalGradient.setColorAt(1.0,Qt::green);
78. //设置渐变的颜色和路径比例
79. painter.setBrush(QBrush(conicalGradient));
80. painter.drawEllipse(410,10,200,200);//在相应的坐标画出来
81. }
82. QSizegradient\_test::sizeHint()**const**
83. {
84. returnQSize(height(),height());
85. }
86. voidgradient\_test::resizeEvent(QResizeEvent\*event)
87. {
88. wheel=QImage(event->size(),QImage::Format\_ARGB32\_Premultiplied);
89. wheel.fill(palette().background().color());
90. drawLine(event->size());
91. update();
92. }
93. voidgradient\_test::paintEvent(QPaintEvent\*event)
94. {
95. QPainterpainter(**this**);
96. QStyleOptionopt;
97. opt.init(**this**);
98. painter.drawImage(0,0,wheel);
99. style()->drawPrimitive(QStyle::PE\_Widget,&opt,&painter,**this**);
100. }

  QPainterPath path;

    path. moveTo( 10 , 320 );//起点

path. cubicTo( 200 , 80 , 400 , 80 , 480 , 320 );//前两个坐标控制弧度，后一个点是终点