Linux 下用 C++ 进行 OOP 窗口编程

李宋琛

摘　要　本文介绍了使用 C++ 在 Linux 下开发 X Window 窗口程序的方法。

关键字　Linux，C++，OOP，窗口编程

GUI 编程一直是人们非常感兴趣的一个课题。长期以来，图形化的人机界面一直是 Mach 和微软 Windows 的天下。特别是 MS-Windows，由于窗口程序编写十分容易，因而吸引了大批的程序员为其编写程序，其编程工具也层出不穷，可视化的开发工具如，Visual Basic，Visual C++，C++ Builder 等已经十分普遍。而正是这些工具吸引了大批的程序员投向了 Windows。传统 Unix 系列的操作系统虽然功能强大，技术成熟，但由于使用不便，很难吸引普通的 PC 用户涉足，而 Unix 下虽然也可以进行图形界面的窗口程序设计，但是却令程序员十分头痛。像基于 XLib 或者 Motif 进行编写都是面向过程的开发形式，开发周期长，难度大，维护困难，只有少数程序员才能掌握，因此没有被广泛使用。

由于近年来 Linux 的快速发展和普及，这种现象正在发生着变化。实际上，当前所有的 Linux 发布版几乎都带有一套甚至更多的桌面系统。Linux 正在悄悄的进军 PC 桌面系统，并且可能凭其开放性和优越的性能而影响微软在此领域的霸主地位。在 Linux 下进行图形界面开发的方法多种多样，一般都是以某种库代码为基础而进行。本文将介绍使用 C++ 进行面向对象的窗口编程方法和实例。

本文将着重讲述一种 Linux 下的窗口编程方法，读者需要一定的 Linux 使用经验和 C++ 编程的知识。

一、X Window 系统

X Window 是由麻省理工学院（MIT）设计的一套开放性的口系统，当前版本为第六版，即 X11R6。这个系统实际上是由一系列的协议组成，因此有时候也称为 X Window 协议。与 Windows 的一个重要区别是，所有 X Window 下的程序都是 Client/Server 模型，其中的 Server 是控制实际绘图的程序，在 Linux 中一般就是图形驱动程序完成，并在系统安装时就已配好。而所有的用户程序是客户端，通过一系列的函数调用请求服务器完成绘图功能。实际上，X Window 协议是基于 TCP/IP 之上的，使用默认的 6000 端口进行通信。因此，X Window 系统本身就是分布式的，在其中一台计算机上运行，而在另外一台计算机上显示和在同一台计算机上进行并没有什么实质上的差别，程序员也不需要了结其中的细节。

进行 X Window 程序设计最基本的方法就是调用 Xlib 库，它是对 X Window 协议的直接描述。因此，使用其它库能写出的程序都可以用 Xlib 进行重写。然而，由于 Xlib 库中的函数过于初级，大都是画点，画线--类的非常初等的函数，用 Xlib 直接进行一般的应用程序开发工作量太大，并不实际。

由于上述的原因，一些公司或者组织开发了许多构建在 Xlib 等之上的库，提供一系列的高级函数，如 OSF 的 Motif, Athena 等。这些库的共同特点是，面向过程编程，提供画窗口，对话框等基本函数。Motif 在一定程度上获得了成功，并一度成为 Unix 下的标准 GUI 开发包，但是和九十年代中后期迅猛发展的 Windows 相比，差距还是十分明显，在 Unix 中急需一个类似 Microsoft MFC 的类库包使程序员能进行 OOP 的程序设计，提高代码的可重用性和好的维护性。

二、Qt 库

Qt 库是由挪威 Troll 公司设计开发。是一个直接用于 Xlib 的，比较成熟的 OOP 开发包。使用 Qt 进行窗口程序设计非常类似用 Visual C++ 的 MFC。因而自从其发布后，出现了一大批用 Qt 写成的自由软件，其中最有名的就是 KDE(K Desktop Environment)。虽然 Qt 作为商业软件并不遵从 GPL 许可证，但是，只要是用于非 Windows 平台的应用，都是免费的。而且正如上面提到的，基于 Qt 的程序可以方便地移植到 MS Windows 中。安装 Qt 非常简单，也许你的 Linux 系统中已经安装了。RedHat 6.0 以上都带有，你可以用 rpm 命令安装编译好的二进制包。如果你没有，可以到 Troll 公司的站点（www.trolltech.com）下载。具体安装方法不在此赘述，需要提醒的一点是要正确的设置环境变量 PATH，QTPATH，LD_LIBRARY_PATH 的值。在启始目录的 .bashrc 文件中加上如下语句：

```
export QTDIR = /usr/local/qt
export PATH = $PATH: $QTDIR/bin
export LD_LIBRARY_PATH = $QTDIR/lib
```

其中 QTDIR 应该设置成你 Qt 库的实际安装位置。

三、第一个“Hello, World”程序

本节将描述一个最简单的，可运行的窗口程序，以使读者能够有一个初步认识。用任何你喜欢的编辑器编辑以下的 hello.cpp 和 Makefile 文件(可以省略掉 / / 开始的注释)，注意，你必须修改 Makefile 中的 QTINC 和 QTLIB 宏，使它们和你的安装路径相匹配，否则编译 2 无法通过。然后在目录下输入“make”，如果没有什么错误的话，应该生成一个叫“hello”的程序，输入 ./hello 运行一下，看看效果。

hello.cpp

```
#include <qapplication.h>
#include <qpushbutton.h>
int
main(int argc, char * * argv)
{
    //QApplication 是应用程序类,由它创建程序实例,并完成命令行的参数分析
    //X Window 程序和其它的 Unix 程序一样,可以接受参数.并且,这些参数在所有
    //的 XWindow 程序中通用,比如 -geometry,可以把这些参数全部传给 QApplication
    //进行分析.所有的应用程序都应该有该类的实例.
    QApplication a(argc, argv);
    //构建一个 PushButton 实例
    QPushButton hello("Hello World!");
    //进行窗口的缩放,第一个参数是长,第二个参数是宽
    hello.resize(100, 30);
    //设置"主窗口";
    a.setMainWidget(& hello);
    //显示窗口并执行应用程序.
    hello.show();
    return a.exec();
}
Makefile
CC = g + +
QTINC = /usr/lib/qt - 1.45/include
QTLIB = /usr/lib/qt - 1.45/lib
hello: hello.o
    ${CC} -o hello hello.o -lqt -L${QTLIB}
hello.o: hello.cpp
    ${CC} -c hello.cpp -I${QTINC}
```

四、Signal - Slot 机制

要实现一个好的 OOP 库必须解决对象间通信的问题。实际对象间通信的问题一直是面向对象程序设计的一个重要内容，而实现解决这个问题的方法也多种多样，Qt 中的对象通信机制非常有特点。在 Qt 中的对象间通信方法被称作“Signal - Slot”，这也是 Qt 与其它一些方法的区别之一。

在 MS - Windows 中，程序通过消息机制和事件循环来实现对图形对象的行为进行触发。而在 Qt 中采用了另外的一种实现方法。一个类可以定义多个 Signal 和 Slot，Signal 就好像是“事件”，而 slot 则是响应事件的“方法”，并且和一般的成员函数没有太大的区别。而需要实现它们之间通信时，就将某个类的 Slot 与另外一个类的 Signal “连接”起来，从而实现“事件驱动”。举个例子，我们需要实现这个功能：当用户按了某个按钮，程序弹出一个对话框。我们的做法就是：当按钮按下时，发出一个 Signal，而这个 Signal 是事先和某个类的弹出对话框的方法，也就是 slot 相连接的。需要注意的是 Signal 和 Slot 函数的返回类型必须是 void，并且不支持缺省参数。

有了 Signal - Slot 机制，进行事件响应编程就变得十分的轻松。比如对于鼠标事件，在 Windows 中是通过发送鼠标左击事件给相应的窗口，而在 Qt 中，鼠标左击事件被设计成某个对象的 Signal。这样，如果需要响应鼠标左击事件，只需要简单的把需要的函数与该鼠标事件连接起来即可。

让我们来看一段简单的代码。以下的 3 个文件的编译方法同上。

```
test.h
#ifndef _TEST_H_
#define _TEST_H_
//包含必要的头文件
#include <iostream.h>
#include <qobject.h>
//在该类中有几个显著的特点:
//1. 有一个 Q_OBJECT 宏,所有派生于 QObject 并且要使用 Signal - Slot 机制的类都必须
//  定义这个宏
//2. public slots 下定义的是响应"事件"的"方法",这些成员函数可以通过 connect
//和 slog 相连接,但其本质上就是一般的公用成员函数.
//3. signals 下定义"事件",注意,这里定义的函数没有实现,只是原型
class classA: public QObject
{
  Q_OBJECT
  public:
      say(){
          emit sayHello();
      }
  public slots:
      void hello(){
          cout < < "hello, world\n";
      }
  signals:
      void sayHello();
};
#endif

test.cpp
#include "test.h"
//程序中通过 QObject::connect 函数将 sayHello 与 hello 连接
//QObject::connect 为 QObject 的静态成员(static public member)原型为:
//bool connect)(const QObject * sender, const char * signal, const QObject *
// receiver, const char * member)
//当执行 a.say()时,发生了 sayHello()事件,触发了 hello 响应
//产生了输出.
//注意一下输出的结果,只有一行"hello, world"而不是两行,原因在于
//执行 b.say()的时候虽然也发出了信号,但是 b 的 signal 并没有和 a 的
//slot 相连接,所以没有输出.
void main()
{
  classA a, b;
  QObject::connect(& a, SIGNAL(sayHello()), & b, SLOT
```

```
(hello()));
    a.say();
    b.say();
}
```

Makefile

```
#由于 test.h 中定义类的特殊性(含有 Q_OBJECT, public slots, signals 等关键字)
#在编译前必需进行特殊的宏处理,这通过一个程序 moc 来完成.
CC = g++
QTINC = /usr/lib/qt-1.44/include
QTLIB = /usr/lib/qt-1.44/lib
MOC = /usr/lib/qt-1.44/bin/moc
test: test.o test.moc.o
    ${CC} test.o test.moc.o -o test -lqt -L$(QTLIB)
test.o: test.cpp
    ${CC} -c test.cpp -I$(QTINC)
test.moc.o: test.moc.cpp
    ${CC} -c test.moc.cpp -I$(QTINC)
test.moc.cpp: test.h
    ${MOC} test.h -o test.moc.cpp
```

由于 Signal-Slot 机制的特殊性，当我们在使用这个机制时可能会碰到不少问题，但是，用 Qt 编程又不可能回避之，因此在使用 Signal-Slot 机制和 moc 时应该要注意以下几点：

1. 所有的头文件必须经过 moc 的处理后才能进行编译，使用方法(以 hello.h 为例)：

moc hello.h -o hello.moc.cpp

2. moc 生成的文件必须由 g++进行编译和连接

3. moc 不支持模板(template)

4. moc 不对#include 等宏进行扩展(简单跳过)

5. 使用多重继承时，moc 假设第一个类是 QObject 或其派生类，而其它的类不能是 QObject 或其派生类

6. signal，slot 可以接受参数，但不能是函数的指针

7. moc 不支持 friend 关键字

8. moc 不支持嵌套类

9. signal 和 slot 不能是构造函数

10. 不支持缺省参数

11. 所有的 signal 和 slot 不能有返回值(必须是 void)

12. slot 可以为虚函数

13. signal 的访问权限和 protected 相同，即只有本身及其派生类可以 emit signal。

五、使用控件(widget)

前面介绍一个简单的例子和 Signal-Slot 机制。但是要进行完整的图形界面的程序设计还需要大量的高级控件，这些控件在 Qt 中被称为“widget”。与 Windows 下的程序开发过程类似，Qt 中也有若干类似的控件可以使用，这对我们进行快速应用开发提供了条件。这些控件包括常用的各种按钮，进度条，菜单，工具条，状态条等，甚至还包括了 OpenGL 控件。为了解释一般的开发过程，下面分步描述如何实现一个带菜单，工具条，状态条和模态对话框的程序框架。

同其它的 Unix 程序一样，代码从 main 开始编写。在 main 中主要做两件事情，其一是设置应用程序实例，这通常由构造 QApplication 来实现，另外一个是确定主控件，或者叫“主客户窗口”。看下面这段代码：

```
int
main(int argc, char ** argv)
{
  QApplication a(argc, argv);
  MainWidget w;
  w.setGeometry(150, 150, 300, 200);
  a.setMainWidget(&w);
  w.show();
  return a.exec();
}
```

以上的这段代码十分典型。首先构造一个 QApplication 的实例，并把命令行参数进行分析。然后构造一个“主控件”MainWidget。执行 w.show()和 a.exec()是必须的，否则程序不能正常的显示和运行。

有 Windows 编程经验的读者可能会问了：为什么 setMainWidget()设置的是“主客户窗口”而不是“主窗口”呢？其实这是由 X Window 协议本身的特点决定的。在 X Window 中的窗口程序，它们的“边框”的样式和风格是应用程序自身无法控制的，而是由另外一个叫“窗口管理器”的程序来进行控制。应用程序只能控制“边框”里面的部分，而不能实现，像在 Windows 中那样可以设置边框风格的功能。如果程序一定要实现这个功能，就必须和这个“窗口管理器”进行“协商”。当然，由于这并非强制性的，所以窗口管理器可以不理会你的程序提出的要求。

以上的程序还不完整，因为 MainWidget 这个类并不是由 Qt 库提供的，而是我们定义在 mainwidget.h 这个头文件里面的。所以我们必须实现之。这个 MainWidget 可以是任何 Qwidget 的派生类，由于我们常常要使用的是普通的应用程序框架，所以我们在这里从 QmainWindow 派生。

```
class MainWidget: public QMainWindow{
  Q_OBJECT
  public:
MainWidget(QWidget * parent=0, const char * name=0);
  public slots:
      void setColor();
      void exitMain();
  private:
      QMenuBar * menu;
};
```

上面是 MainWidget 的定义。首先是 Q_OBJECT 宏，上面曾经提到过，所有要使用 Signal-Slot 机制的类都必须使用这个宏定义。另外有两个 Slot：setColor()和 exitMain()。先看看这个类的实现部分。

```
#include "mainwidget.h"
MainWidget::MainWidget(QWidget * parent, const char *
```

```
name)
: QMainWindow(parent, name)
{
setCaption("Example");
//设置窗口的标题
QPopupMenu * option = new QPopupMenu;
//构造一个弹出式的菜单
option ->insertItem("& Set color", this, SLOT(setColor()));
    option ->insertSeparator();
    option ->insertItem("E& xit", this, SLOT(exitMain()));
    menu = new QMenuBar(this);
    menu ->insertItem("& Option", option);
}
void
MainWidget: : setColor()
{
    MainDialog m;
    m. exec();
}
void
MainWidget: : exitMain()
{
    QApplication: : exit();
}
```

在构造函数中主要是新建一个菜单。在 Qt 中的菜单实际上有两个部分组成，一个是菜单条 QMenuBar，另外一个是弹出式的菜单 QPopupMenu，这两个部分必须单独构建。需要注意的是 insertItem()方法的第 2，3 个参数，这实际上是调用了 connect()方法把菜单被选择这个事件与某个类的 Slot 方法连接起来，这是很典型的用法。

在 exitMain()方法中简单地调用了 QApplication 类的静态方法 exit()，这也是退出应用程序的一般途径。而在 setColor()方法中，我们又定义了一个新的类：MainDialog，并使用和 QApplication 类似的 exec()方法使之运行。以下是 MainDialog 的定义和实现：

```
class MainDialog: public QDialog{
    Q_OBJECT
    public:
MainDialog(QWidget * parentl = 0, const char * namel = 0);
};
MainDialog: : MainDialog(QWidget * parent, const char *
name)
    : QDialog(parent, name, TRUE)
{
    setCaption("Choose color");
    QPushButton * r, * b;
    r = new QPushButton("Red", this);
    b = new QPushButton("Blue", this);
this ->setFixedSize(140, 80);   //设置对话框的大小
    r ->setGeometry(20, 20, 40, 30);
    b ->setGeometry(80, 20, 40, 30);
    connect(r, SIGNAL(clicked()), SLOT(accept()));
    connect(b, SIGNAL(clicked()), SLOT(reject()));
}
```

这个类的实现很简单，安排两个按钮。最后两个 connect 调用需要注意。每个按钮的 Signal - clicked()分别和一个 Slot 相连接。而 accept()和 reject()都是 Qdialog 的成员函数，作用是关闭对话框并返回“真”或者“假”。由此就可以判断对话框中选择的内容。

如果实际应用中是个很复杂的对话框，含有各种按钮，列表等，这些数据不能简单的用“真”或者“假”来表示怎么办呢？方法也很简单，就是在 MainDialog 的私有变量中保留这些选择的数据，并在对话框完成后通过公共接口进行访问即可。

五、响应鼠标事件和在窗口中画图

在一般的 GUI 程序中，鼠标是最主要的输入工具。如果不能够很好的处理鼠标事件，应用程序的友好性将大打折扣。幸运的是，Qt 在这方面提供了完整的解决方法，使我们在进行程序设计时难度大大降低了，实际上和在 MS - Windows 下进行的程序设计已经十分类似。通常，鼠标事件分成三类：左键单击，右键单击和左键双击。而这三个事件则对应了 Qwidget 中的一个虚方法，不同的鼠标事件有不同的入口参数，这是和 Windows 不同的地方。捕获鼠标事件首先要重载这些虚方法，使鼠标事件发生时能够调用适当的过程。

先看段代码。

```
class MainWidget: public QMainWindow{
  Q_OBJECT
  public:
MainWidget(QWidget * parent = 0, const char * name = 0);
  public slots:
      void setColor();
      void exitMain();
  protected:
      void paintEvent(QPaintEvent * );
      void mousePressEvent(QMouseEvent * );
  private:
      QMenuBar * menu;
      QColor color;
      QPoint point;
};
```

下面是对上一个例子的扩充，其中增加了一个保护的 mousePressEvent()方法. 我们看看它的实现部分.

```
void
MainWidget: : mousePressEvent(QMouseEvent * e)
{
if(e ->button() = = LeftButton){      //看是否按的是左键
point = e ->pos(); //把鼠标的当前位置存入私有成员变量中
repaint();                            //强制进行重绘
  }
}
```

从上面的代码可以看出，在 Qt 中，所有的鼠标事件都调用 mousePressEvent()方法，而要区分它们，就必须比较入口参数 QmouseEvent * 所提供的信息。

与处理鼠标事件类似，要在窗口中画图，需要重载 Qwidget 的虚方法 paintEvent()。以下是 paintEvent 的实现。

```
void
MainWidget::paintEvent(QPaintEvent *)
{
QPainter p;                          //构造一个图形引擎
QPen pen(color);                     //构造"一支笔"
p.begin(this);                       //开始画图
p.setPen(pen);                       //选择一只笔
p.drawText(point, "Hello, World!");  //在 point 处写上文字
p.end();                             //结束画图
}
```

这段代码的功能是在鼠标单击的位置用对话框中设置的颜色写出“Hello, World!”语句。在进行绘图时，首先必须构造一个 Qpainter 对象，这个对象是 Qt 的图形引擎，所有的画图语句都是 Qpainter 的一个成员函数。画图从 begin()语句开始，注意 begein()的参数，这实际上是把当前的这个控件（MainWidget）引入到图形场景中。

Qt 的图形引擎也有“笔”，“刷子”的概念，在这一点上是和 Windows 相同的。通常的画图过程是调用 begin,选择适当的笔和刷子，使用 Qpainter 的成员函数进行化点，线，矩形，坐标变换等过程，最后调用 end 结束绘图过程。

由于加入了图形代码，我们的 setColor 方法也有所变化。

```
void
MainWidget::setColor()
{
    MainDialog m;
    color = m.exec()?red:blue;
}
```

显然，颜色将设置成所选的值。

五、一个完整的应用程序框架

上面的所讨论的代码讲述了一个基本的框架，但这对普通的程序还不够。我们希望在程序中能做到和 Windows 同样的界面和功能。要实现这些，工具条和状态条必不可少。在 Qt 中添加工具条和状态条十分简单。构造一个 QtoolBar 需要若干的 QtoolButton,这些小按钮需要从位图生成，读者可以用 Gimp 在 Linux 下绘制 16x16 的图形，并存为以 .xpm 为后缀的文件，这种图形文件类似 Windows 中的 BMP 文件，是 X Window 的“标准”图形文件，Qt 中也支持。而如果你的主控件是从 QmainWindow 派生的，那么生成状态条只需要一句代码。StatusBar()成员函数会在没有状态条的情况下生成，如果已有，则返回状态条对象的指针。但如果你的 MainWidget 不是从 QmainWindow 派生，使用状态条就要麻烦一些。在 MainWidget 的构造函数中，我们添加了以下代码：

```
MainWidget::MainWidget(QWidget * parent, const char * name)
    : QMainWindow(parent, name), color(red), point(0, 0)
{
    //前面的代码在此省略...
    QToolBar * tools = new QToolBar("example", this);  //构造一个 ToolBar
QPixmap colorIcon(setcolor_xpm), exitIcon(exit_xpm); //从文件生成一副位图
QToolButton * setcolor = new QToolButton(colorIcon, "Set Color", 0,
this, SLOT(setColor()),
tools, "set color");                 //在 ToolBar 上放按钮
QToolButton * exitmain = new QToolButton(exitIcon, "Exit", 0,
  this, SLOT(exitMain()),
  tools, "exit");
  statusBar()->message("Ready", 2000);
}
```

到此为止，一个基本的应用程序框架已经讲述完成。下面贴上窗口图像和完整的程序代码供读者参考。全部程序包括 4 个文件：Makefile, main. cpp, mainwidget. h, mainwidget. cpp。

另外，读者必须自己准备两幅 16x16 的 XPM 格式图形，分别存为“setcolor. xpm”和“exit. xpm”。然后在目录下输入“make”会产生一个“example”的可执行文件。

Makefile:

```
CC = g++
QTINC = /usr/lib/qt-1.44/include
QTLIB = /usr/lib/qt-1.44/lib
MOC = /usr/lib/qt-1.44/bin/moc
example: main.o mainwidget.o mainwidget.moc.o
  ${CC} main.o mainwidget.o mainwidget.moc.o -o example -lqt -L${QTLIB}
main.o: main.cpp
  ${CC} -c main.cpp -I${QTINC}
mainwidget.o: mainwidget.cpp
  ${CC} -c mainwidget.cpp -I${QTINC}
mainwidget.moc.o: mainwidget.moc.cpp
  ${CC} -c mainwidget.moc.cpp -I${QTINC}
mainwidget.moc.cpp: mainwidget.h
  ${MOC} mainwidget.h -o mainwidget.moc.cpp
```

main.cpp

```
#include "mainwidget.h"
int
main(int argc, char * * argv)
{
  QApplication a(argc, argv);
  MainWidget w;
  w.setGeometry(150, 150, 300, 200);
  a.setMainWidget(&w);
  w.show();
  return a.exec();
}
```

mainwidget.h

```
#ifndef _MAINWIDGET_H_
#define _MAINWIDTET_H_
#include <qmenubar.h>
#include <qapplication.h>
#include <qdialog.h>
#include <qpushbutton.h>
#include <qpainter.h>
#include <qpen.h>
```

```
#include <qtoolbar.h>
#include <qpixmap.h>
#include <qtoolbutton.h>
#include <qmainwindow.h>
#include <qstatusbar.h>
class MainWidget: public QMainWindow{
    Q_OBJECT
    public:
MainWidget(QWidget * parent =0, const char * name =0);
    public slots:
    void setColor();
    void exitMain();
    protected:
    void paintEvent(QPaintEvent * );
    void mousePressEvent(QMouseEvent * );
    private:
    QMenuBar * menu;
    QColor color;
    QPoint point;
};
class MainDialog: public QDialog{
Q_OBJECT
public:
MainDialog(QWidget * parentl =0, const char * namel =0);
};
#endif
mainwidget.cpp
#include "mainwidget.h"
#include "setcolor.xpm"
#include "exit.xpm"
MainWidget:: MainWidget(QWidget * parent, const char *
name)
    : QMainWindow(parent, name), color(red), point(0, 0)
{
    setCaption("Example");
    setBackgroundColor(white);
    QPopupMenu * option =new QPopupMenu;
    option->insertItem("& Set color", this, SLOT(setColor
()));
    option->insertSeparator();
    option->insertItem("E& xit", this, SLOT(exitMain()));
    menu =new QMenuBar(this);
    menu->insertItem("& Option", option);
    //add toolbox
    QToolBar * tools =new QToolBar("example", this);
    QPixmap colorIcon(setcolor_xpm), exitIcon(exit_xpm);
    QToolButton * setcolor =new QToolButton(colorIcon, "
Set Color", 0,
    this, SLOT(setColor()),
    tools, "set color");
QToolButton * exitmain =new QToolButton(exitIcon, "Exit", 0,
    this, SLOT(exitMain()),
    tools, "exit");
    statusBar()->message("Ready", 2000);
}
void
MainWidget:: paintEvent(QPaintEvent * )
{
  QPainter p;
  QPen pen(color);
  p.begin(this);
  p.setPen(pen);
  p.drawText(point, "Hello, World!");
  p.end();
}
void
MainWidget:: mousePressEvent(QMouseEvent * e)
{
  if(e->button() == LeftButton){
statusBar()->message("You pressed mouse key", 2000);
      point =e->pos();
      repaint();
  }
}
void
MainWidget:: setColor()
{
  statusBar()->message("You opened a dialog", 2000);
  MainDialog m;
  color =m.exec()?red: blue;
}
void
MainWidget:: exitMain()
{
  QApplication:: exit();
}
MainDialog:: MainDialog(QWidget * parent, const char *
name)
  : QDialog(parent, name, TRUE)
{
  setCaption("Choose color");
  QPushButton *r, *b;
  r =new QPushButton("Red", this);
  b =new QPushButton("Blue", this);
  this->setFixedSize(140, 80);
  r->setGeometry(20, 20, 40, 30);
  b->setGeometry(80, 20, 40, 30);
  connect(r, SIGNAL(clicked()), SLOT(accept()));
  connect(b, SIGNAL(clicked()), SLOT(reject()));
}
```

七、进一步讨论

Qt 的功能十分的强大，远远不只是上面的介绍的功能，感兴趣的读者可以参考 Qt 附带的电子文档，Qt 的主页上也有很多关于此方面的讨论。另外，网上已经有不少开放源代码的程序，对我们进一步学习也很有帮助。

（收稿日期：2000 年 11 月 7 日）