用 Qt 编制 Linux 中 X-windows 下的应用程序

周 利

一、Qt 概述

Qt™ 是一个多平台的工具包（for C++），它是 Troll Tech 公司的产品，大部分操作系统，从 Microsoft Windows 到 Unix/X Windows 及 Linux/X Windows 对其都有支持。在 Linux 下有一个 Qt 的免费版本及详尽的教程，其编程风格有些像 Java 和 Borland C++，习惯用 Java 或 Borland C++的程序员很快便可掌握。

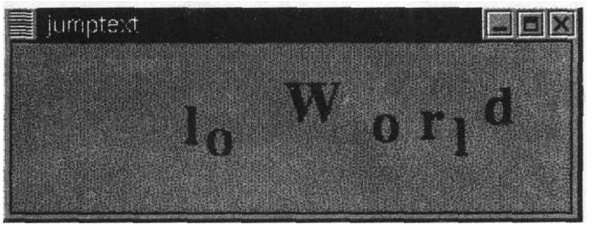
二、如何用 Qt 开发应用程序

关于 Qt 的使用详见 Qt 目录下的 Html 目录（大多数版本的 Linux 都将其安装在/usr/lib/qt 目录下），其内容集帮助与教学于一体，相信您看完后，对 X Windows 的编程技术定会了解不少。这里我仅通过一个应用程序，向您展示一下如何使用 Qt 编程。

一个基本的 Windows 应用程序由一个主窗口和一组窗口过程组成。X-Windows 下的应用程序也是如此。

```
/*++++++++++++++   jumptext.cpp   ++++++++++++++*/
#include "jumptext.h"
#include <qapplication.h>
#include <stdlib.h>
#include <qtimer.h>
#include <math.h>
#include <qpainter.h>
#include <qpixmap.h>
.....
.....//类 JumpText 的具体实现，后面详细讲述。
.....
int main(int argc,char** argv)
{
    //首先，构造一个 Application 对象。
    QApplication app(argc,argv);
    //构造主窗口，JumpText 是下面将要构造的一个窗口类。
    JumpText jtWnd("Hello World!");
    //设置应用程序的主窗口
    app.setMainWidget(&jtWnd);
    //设置显示字体，可再类内实现
    jtWnd.setFont(QFont("Times",32,QFont::Bold));
    //显示主窗口
    jtWnd.show();
//进入窗口过程，即进入消息循环。
    return app.exec();
}
```

接着要做的，就是制作一个如下的窗口：



下面给出 JumpText 的实现过程：

```
/*++++++++++++   jumptext.h   ++++++++++++++*/
#ifndef JUMPTEXT_H
#define JUMPTEXT_H
//qwidget.h 为 JumpText 基类的头文件
#include <qwidget.h>

class JumpText:public QWidget
{
    //Q_OBJECT 必不可少，因为要用到一个自定的定时消息。
    Q_OBJECT
public:
    JumpText(const char* text=0,QWidget* parent=0,const char*
      name=0);
protected:
    //类 Qwidget 的派生函数，用于接受窗口的重绘消息
    void paintEvent(QPaintEvent* );
protected slots:
    //用于接收定时消息，由于这是一个自定义的消息，所以定义为 slots
    // 由 于 语 句  connect(timer,SIGNAL(timeout()),SLOT(draw
Text()));
    //完成将定时消息与 drawText () 函数相关联
    //关于消息的定义及接收相见下文
    void drawText();
private:
    //所要绘制的字符串，即图上所示的 Hello World!
    QString caption;
};

#endif

/*++++++++++++++   jumptext.cpp   ++++++++++++++*/
//将这一段代码加在上面有省略号（.....）的地方。
//JumpText 的构造函数，text 指明要显示的文本，parent 指明父窗口，name
指明窗口名。
JumpText::JumpText(const char* text,QWidget* parent,const char*
name)
    :QWidget(parent,name),caption(text)
{
    //首先，构造一定时器。
    QTimer* timer=new QTimer(this);
```

```
    //将定时消息与 drawtext () 函数相关联。
    connect(timer,SIGNAL(timeout()),SLOT(drawText()));
    //启动定时器，每 200 毫秒一次。
    timer->start(200);
    //重新调整窗口大小
    resize(320,100);
}

void JumpText::drawText()
{
    //以不擦除背景的方式重绘。
    repaint(FALSE);
}

void JumpText::paintEvent(QPaintEvent* )
{
    //重绘过程，制造出文字蹦蹦跳跳的效果。

    //如果没有要显示的文字，则不用画了
    if(caption.isEmpty())return;
    static int ci=0;
    ci=(ci+1)%caption.length();
    //字体信息
    QFontMetrics fm=fontMetrics();

    int w=fm.width(caption)+10*caption.length();
    int h=fm.height()*3;
    int dx=fm.width("a")/2;
    int dy=h/2;
    int pmx=width()/2-w/2;
    int pmy=height()/2-h/2;

    //QPainter 与 MFC 的 CDC 类似
    QPainter p;
    //Qpixmap 与 MFC 的 Cbitmap 类似
    QPixmap pm(w,h);
    //以窗口中（pmx，pmy）点（即正中央）的背景颜色填充位图。
    pm.fill(this,pmx,pmy);
    //开始向位图绘制文字
    p.begin(&pm);
    //与 MFC 的 SelectObject 函数类似
    p.setFont(font());
   int x=10,y=h/2+fm.descent();
   int len=caption.length();
   for(int i=0;caption[i];i++){
        int xpos=x+(rand()%dx-dx/2);
        int ypos=y+(rand()%(dy/2)-dy/4);
        x+=fm.width(caption[i])+10;
        int cc=(ci+i)&len;
        //设置画笔颜色
        p.setPen(QColor((len-cc)*(255/len),255,255,
            Qcolor::Hsv));
        //绘制文字
        p.drawText(xpos,ypos,&caption[i],1);
    }
    //结束绘制过程
    p.end();
    //像屏幕输出图形
    bitBlt(this,pmx,pmy,&pm);
}
```

几点注意：

1. 关于消息处理的说明：

在 Qt 中消息的处理通过 SIGNAL/SLOT 方式完成，对于常见的消息，Qt 在类中已定义好了消息的处理函数，只需重载它即可完成。但对于像菜单消息，或是我们想要定义新消息，添加新的消息处理函数，并将其特定的消息相关联，就必须使用 SIGNAL/SLOT 模型。

一个典型的 Qt 类如下：

```
class Foo : public QObject
{
    Q_OBJECT
public:
    Foo();
    int  value() const { return val; }
public slots:
    void setValue( int );
signals:
    void valueChanged( int );
private:
    int  val;
};
```

在此类中，有一私有变量 val，使用了 SIGNAL/SLOT 模型，此类能够通过发出一个“信号”（signal）valueChanged () 来告诉其对象，其私有变量 val 发生了改变，并且，此类还具有一个“插槽”（slot），可以使其对象发出消息改变 val 的值。

下面是 setValue 的一种可能实现：

```
void Foo::setValue( int v )
{
    if ( v != val ) {
        val = v;
        emit valueChanged(v);
    }
}
```

这里举一个例子，来表明如何将两个对象关联起来：

```
Foo a, b;
//将 a 的值的改变信号（valueChanged）与 b 的设置值的函数
 (setValue) 相关联
connect(&a, SIGNAL(valueChanged(int)), &b, SLOT(setValue(int)));
b.setValue( 11 );//b 发出值的改变信号
setValue( 79 );//a 发出值的改变信号，由于 b 的 setValue 函数
                与此信号相关联，
//故此时，b 的值为 79
b.value();//得到 b 的值，为 79
```

这个例子十分清楚的表现出 signals 和 slots 的作用。signals 和 slots 是定制消息处理的重要手段，比起 M$ Windows 的消息的处理，要在上万个消息中查找到要处理的消息，然后，再按格式格式化 WPARAM 和 LPARAM 参数，再进行处理，不知简单了多少。千万别忘了在类中加入 Q_OBJECT 声明。

2. 关于如何编译的说明：

(下转第 52 页)

三、 CData 类的使用

至此，CData 对话框类已全部建立起来，接下来就可以在视图（单文档或多文档）中进行数据的维护和管理。为此，在 ResourceView 中双击 Menu 项，然后双击 IDR_MAINFRAME 或 IDR_MULTYPE(对于多文档)，在菜单编辑器中增加数据综合维护菜单项，并在视图类中添加响应的菜单响应函数 OnManageMaintance()，然后在视类的实现文件中包含 CData 类的头文件，即在 gridView.cpp 函数的开头处添加#include “Data.h”，如下所示：

```
#include "gridView.h"
#include "Data.h"
#ifdef _DEBUG
...
```

包含 CData 类的头文件之后，即可在视类的 OnManageMaintance()成员函数中进行调用，具体过程如下：

```
void CGridView::OnManageMaintance()
{
    CData dlg;//定义 CData 对话框类
    dlg.DoModal();//调用 CData 类对话框
}
```

调用方法十分简单，将 CData 类灵活地嵌入到程序中使用，可以起到事半功倍的效果。最后运行的效果如图 3 所示（数据文件为 test.dat）。

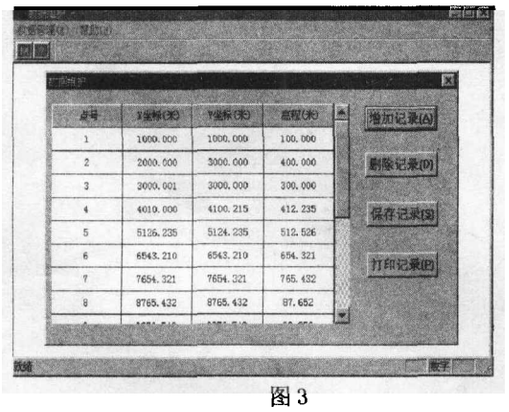


图 3

四、小结

本文是结合 MFCGridCtrl 控件，使用了其中的一部分函数，另外还有很多函数功能也十分强大，在此不能一一介绍，笔者利用该控件实现对数据的全面维护，包括显示、修改、删除、增加、保存、打印，已基本满足数据操作人员的各种需要，对于显示还可以进行很多提高，如增加图形，图标显示，改变文本背景和文字颜色，字体等等，由于篇幅所限，不可能在此全部介绍，读者可自己去进一步的研究。

（收稿日期：1999 年 11 月 4 日）

（上接第 19 页）

尽管 Linux 为我们提供了这么好的工具包，我还没有找到关于 Qt 的真正的可视化的编程环境。所以，没办法，我们只能找个编辑器编程，再用命令行的方式或自己定制一 Makefile 进行编译。建议大家多看一些有关编译程序的使用方法。

这里简单地说一下 Makefile 的使用。对于初学者，你可能不知道怎样制作 Makefile。让我们耍点小聪明，从 Qt 目录下的例程中拷贝一个 Makefile 到你的程序所在目录，然后按下列步骤对其进行修改，最后在该目录下，键入 make，便可生成一可执行文件，键入”./文件名”即可执行它，当然，要保证在图形状态下。

修改步骤：

1. 找到有“QTDIR=……”的那行，删除等号后的字符，在等号后输入 Qt 头文件所在的目录（Mandrake Linux 的头文件目录通常在/usr/lib/qt/include 内，Turbo Linux 4.0 的头文件目录通常在/usr/include/qt 内）。

2. 找到有“TARGET=…”的那行，删除等号后的字符，在等号后输入要生成的可执行文件名。

3. 在“#########Files ”的下行输入所要进行编译的程序(所有以“#”开头的句子都是注释)。本例中如下：

```
HEADERS =   jumptext.h      #头文件
SOURCES =   jumptext.cpp    #类的实现文件
OBJECTS =   jumptext.o      #编译生成的OBJ文件
SRCMOC = moc_main.cpp   #加入Q_OBJECT声明后，要为定义的signals和slots生成一个文件供编译使用。该文件由编译程序自动生成。
    OBJMOC =moc_main.o #OBJ 文件名。
```

4. 找到“###########Compile”的那行，仿照其编译规则替换为你自己的编译规则。本例如下：

```
###### Compile
jumptext.o: jumptext.cpp \
            jumptext.h
moc_main.o: moc_main.cpp \
            jumptext.h
moc_main.cpp: jumptext.h
        $(MOC) jumptext.h -o moc_main.cpp
```

5. 进入终端机，到程序所在目录下，用make 命令进行编译，即可。

（收稿日期：1999 年 10 月 14 日）