

Linux 下数据库应用的设计与实现

张绪兵, 陈今润

(重庆大学 自动化学院, 重庆 400044)

摘要:很多实际应用都需要使用数据库来保存数据,以供今后查询、修改等。在介绍 Linux 下 GTK+ 编程框架后,简略讲述 glade 产生的背景以及其配合 GTK+ 的使用方法,然后结合一事例来重点讨论使用 GTK+ 中 GtkTreeView 显示控件和 MySQL 数据库结合,以 C 为开发语言,开发数据显示和保存技术。实践表明,这种开发技术是可行的。

关键词:Linux; GTK+; MySQL; glade; 数据库

中图分类号:TP311.13

文献标识码:A

文章编号:1673-629X(2008)08-0226-04

Design and Implementation of Database under Linux Operation System

ZHANG Xu-bing, CHEN Jin-run

(College of Automation, Chongqing University, Chongqing 400044, China)

Abstract: In practical applications, people need database technology for saving data in order to query or modify the data in the future. Introduces the framework of the GTK application based on Linux. Then briefly describes the background on the glade and how to program with the combination of the GTK library and glade library. At last, canvases the way of developing database applications based on C language with MySQL and GtkTreeView, one of widget of the GTK+. The practice shows the effectiveness of this way.

Key words: Linux; GTK+; MySQL; glade; database

0 引言

GTK+ (GIMP Toolkit) 是一个在 Linux 平台上比较流行的 GUI 构建工具,著名的 Gnome 平台就是基于它的,且是当今百分之百零付费的工业级图形界面开发工具。MySQL 是一个精巧的 SQL 数据库管理系统,虽然它不是开放源代码产品,但可以自由使用。由于其功能强大、灵活性好、应用编程接口丰富以及系统结构精巧,从而受到广大软件用户的青睐^[1]。将两者结合起来使用在医疗检验仪器的数据处理和储存上,其实现方法对其他类似应用可以起借鉴作用。

1 应用 GTK+ 编程

GTK+ 程序的框架都是差不多的,一般模式如下:

```
gtk_init(&argc, &argv); /* 初始化 GTK+ 程序 */
```

然后创建基本控件,比如窗口、按钮等,并为控件

加回调函数以实现所要求的操作。例如下面是创建一个按钮并为其增加信号连接函数:

```
GtkWidget * btn = gtk_button_new_with_label("Hello World!");
```

```
g_signal_connect(G_OBJECT(btn), "clicked", G_CALLBACK(btn_click), (gpointer)"hello");
```

最后进入主循环 `gtk_main()`。GTK+ 是一种事件驱动工具包,这意味着它将 `gtk_main` 函数中一直等待,直到事件发生和控制权被传递给相应的函数^[2]。所以需要程序员创建一个信号连接函数来捕获该事件信号,并告诉 GTK 程序事件发生后调用哪个回调函数。信号连接函数一般形式为: `gint gtk_signal_connect (GtkWidget * obj, gchar * name, GtkSignalFunc func, gpointer data)`。第一个参数 * obj 是将要发出信号的构件,第二个参数 * name 是希望捕获的信号的名称,第三个参数 func 是捕获信号时要调用的函数,第四个参数是要传递给 func 的形参。通常 func 函数称为“回调函数”,它的形式通常是 `void callback_func (GtkWidget * widget, gpointer data)`。其中第一个形参是指向发出信号的构件的指针,第二个形参是指向传递给回调函数的用户数据的指针。注意上述对“信号”

收稿日期:2007-11-27

作者简介:张绪兵(1981-),男,江西九江人,硕士研究生,研究方向为图像处理及其在智能检测控制中应用;陈今润,硕士生导师,副教授,研究方向为检测技术与自动化装置。

的回调函数的声明仅仅是一个通用的规则。其他规则参考文献[3]。

使用 GTK 编程的概念并不难。然而使用这些函数存在一些困难:创建程序界面的代码是非常繁琐的,特别是在使用不同的布局构件组装界面元素等时,不能在编写代码时直接看到显示效果。应该有一种工具,可以将人们从这些工作中解放出来,让人们能够专注于任务的核心。现在有个非常出色的界面生成工具:Glade。它可以用可视化的方法绘制应用程序界面,设置窗口、构件的外观、设置构件信号的回调函数,然后生成 C 语言代码。它类似于 Visual Basic。不同点在于 VB 是一个集成开发环境,不仅可以创建界面,还可以创建完整的应用程序,以及调试、编译等。而 Glade 仅仅是一个 GUI(图形用户接口)生成器,它只能用于生成创建界面的代码,实现应用程序的功能、编译、调试等工作需要使用其他工具。创建完界面后,再与 gtk 结合使用^[4]。其得到 GTK 的构件方法如下:

```
GladeXML * xml = glade_xml_new("udc.glade", "mainwin-
dow", NULL); /* 非 0 表示成功调用, mainwindow 为根构件,
udc.glade 为使用 glade 保存的文件名 */
```

```
GtkWidget * scrollwin_treeview = glade_xml_get_widget
(xml, "scrollwin_treeview"); /* 得到根构件下的一个子构件, 即
在 glade 中名字为 scrollwin_treeview 的构件 */
```

```
glade_xml_signal_autoconnect(xml); /* 自动连接所有与信
号句柄相同的函数 */
```

注意当 xml 不使用的时候,使用 `gtk_object_unref(GTK_OBJECT(xml))`;及时释放 xml。

2 用 MySQL 和 GTK+ 中 GtkTreeview 开发数据库

结合一事例来讲述整个流程,需要一个包含病人信息的结构体,为了一般性,定义如下:

```
struct patient_info { /* 病人资料信息 */
    guint32 databaseNo; /* 数据库里主键 */
    guint8 checkNo; /* 检验号 */
    gboolean emergency_treat_mark; /* 急诊标记 */
    gchar name[16]; /* 姓名 */
    gchar sample_date[16]; /* 标本日期 */
    gchar glucose[16]; /* 葡萄糖 */
};
```

在论述中,为了节约篇幅,所有的出错处理均省略。其流程如图 1 所示。

详细介绍如下:

```
... //流程所示的初始化;
```

```
GtkWidget * scrollwin_treeview = glade_xml_get_widget
(xml, "scrollwin_treeview");
```

```
GtkWidget * treeview_record = initialize_treeview( treeview_
```

```
record, scrollwin_treeview);
... mysql_database_init();
glade_xml_signal_autoconnect(xml); gtk_main();
```

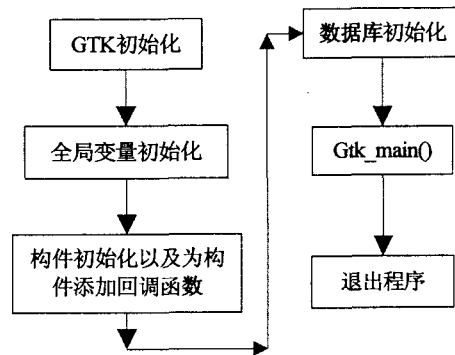


图 1 程序流程图

其中部分函数如下:

/* GtkTreeView 是将显示的数据和数据存储分开的抽象显示控件,显示有树型和列表两种,本程序用列表类型。

list_store_db 为 GtkListStore * 类型全局变量,目的方便函数间传递 */

```
GtkWidget * initialize_treeview ( GtkWidget * tree_view,
GtkWidget * scrollwin_treeview ) {
```

```
/* 创建一个 list,形参一表示列的总数,其他分别表示各列
的类型 */
```

```
list_store_db = gtk_list_store_new(NUM_COLUMNS, G_
TYPE_UINT, G_ TYPE_UINT, G_ TYPE_STRING, G_
TYPE_STRING, G_ TYPE_STRING, G_ TYPE_STRING);
```

```
tree_view = gtk_tree_view_new_with_model ( GTK_
TREE_MODEL(list_store_db));
```

```
gtk_tree_view_set_rules_hint ( GTK_ TREE_ VIEW( tree_
view), TRUE);
```

```
/* 将 tree_view 控件放到 scrollwin_treeview 控件里 */
```

```
gtk_container_add ( GTK_ CONTAINER ( scrollwin_ tree_
view), tree_view);
```

```
tree_view = treeview_set_col( tree_view, COL_ DATABASE_
NO, "id", 0, COL_ DEFAULT); /* 设置一列的属性 */ tree_
view = treeview_set_col ( tree_view, COL_ CHECK_ NO, "检验
号", 0, COL_ DEFAULT); tree_view = treeview_set_col ( tree_
view, COL_ EMERG_ TREAT, "急诊", 2, COL_ EMERG_ COL_
OR); tree_view = treeview_set_col( tree_view, COL_ NAME, "姓
名", 0, COL_ DEFAULT);
```

```
... /* 标本日期,葡萄糖写法同姓名类同 */ return tree_
view; }
```

```
/* 以下是向列表添加数据的代码,并显示出来 */
```

```
void show_patient_info ( struct patient_info * pt_info) /*
pt_info 指向一个完整病人信息的指针 */
```

```
{GtkTreeIter tree_iter;
```

```
... /* 先将数据进行转换,比如布尔变量急诊为真时转为
```

```

字符 yes */
/* 从 list 的最前面开始插入新的数据,还有 gtk_list_store_
append 等其他形式 */
gtk_list_store_prepend(GTK_LIST_STORE(list_store_
db), &tree_iter);
gtk_list_store_set(GTK_LIST_STORE(list_store_db),
&tree_iter,
COL_DATABASE_NO, pt_info->databaseNo, COL_
CHECK_NO, pt_info->checkNo,
COL_EMERG_TREAT, insert_emergency_treat_mark, /
* 表示急诊的字符串 */
/* 急诊的颜色,真值为 red,假值为 blue */
COL_EMERG_COLOR, insert_emergency_color_mark,
COL_NAME, pt_info->name, COL_SAMPLE_DATE,
pt_info->sample_dat,
COL_GLUCOSE, pt_info->glucose, -1); /* -1 表示
一次 column 结束 */
static GtkWidget *treeview_set_col(GtkWidget *tree_
view, enum COL_INFO col_x, gchar *column_title, int setCol_
or, enum COL_INFO column_extra_info){
    GtkTreeViewColumn *column; GtkCellRenderer *cell_ren_
derer;
    column = gtk_tree_view_column_new();
    if(COL_DATABASE_NO == col_x) gtk_tree_view_col_
umn_set_visible(column, FALSE); /* 是否显示该列 */ gtk_
tree_view_column_set_title(column, column_title); /* 设置列
标题 */
    gtk_tree_view_column_set_resizable(column, TRUE); /*
是否可以改变列宽 */
    gtk_tree_view_column_set_sizing(column, GTK_TREE_
VIEW_COLUMN_FIXED);
    gtk_tree_view_column_set_fixed_width(column, 85);
    gtk_tree_view_append_column(GTK_TREE_VIEW(tree_
view), column);
    cell_renderer = gtk_cell_renderer_text_new();
    gtk_tree_view_column_pack_start(column, cell_renderer,
TRUE);
    switch(setColor)
    { case 1: gtk_tree_view_column_set_attributes(column,
cell_renderer, "text", col_x, "background", column_extra_info,
NULL); break;
      case 2: gtk_tree_view_column_set_attributes(column, cell_
renderer, "text", col_x, "foreground", column_extra_info,
NULL); break;
      case 0: gtk_tree_view_column_set_attributes(column, cell_
renderer, "text", col_x, NULL); break; }
    gtk_tree_view_column_set_sort_column_id(column, col_
x);

```

```

return tree_view; }
void mysql_database_init(void){
    iscreate = FALSE; /* gboolean 类型全局变量,用来判断是
否已经创建了数据库 */
    isclosed = FALSE; /* gboolean 类型全局变量,用来判断是
否关闭了数据库 */
    if(TRUE == my_connect_base()) /* 数据库服务器打开成
功 */
    { mysql_database_create();
      mysql_database_open();
      mysql_database_table_create();
      mysql_database_table_select(&pat_info, NULL, NULL);
      static gboolean my_connect_base(void){
          myconnect1 = mysql_init(myconnect1); /* myconnect1 为
MYSQL * 类型全局变量 */
          if(mysql_real_connect(&myconnect1, "localhost", NULL,
NULL, NULL, MYSQL_PORT, NULL, 0)) /* 用来连接数据
库服务器 */ { return TRUE; }
          else { myconnect1 = NULL; return FALSE; }
          static void mysql_database_close(void){
              mysql_close(myconnect1); myconnect1 = NULL;
          }
          static void mysql_database_create(void){
              gchar query_buf[256]; if(iscreate == FALSE) /* 如果没有
创建数据库,就创建数据库 */
              { sprintf(query_buf, "CREATE DATABASE %s", "patient_
database1");
                if(mysql_query(myconnect1, query_buf) == 0) { iscreate =
TRUE; } /* 创建成功 */
                else iscreate = FALSE; }
              static void mysql_database_open(void){
                  gchar query_buf[256]; sprintf(query_buf, "USE %s", "pa_
tient_database1");
                  if(mysql_query(myconnect1, query_buf) == 0) /* 数据库
打开成功 */
                  else; /* 数据库打开失败 */
                  static void mysql_database_table_create(void){
                      const gchar *sql_query = "CREATE TABLE patient1 (id int
(8) AUTO_INCREMENT PRIMARY KEY, checkNo int(8), e_
mergency_treat_mark char(8), name char(32), sample_date
date, glucose char(32));" /* 这里第一个是主键 */
                      if(mysql_query(myconnect1, sql_query) == 0) /* 表格创建
成功 */
                      else; /* 表格创建失败 */
                      void mysql_database_table_insert(struct patient_info *pt_
info){
                          /* 向表格里插入一个病人的数据 */
                          gchar query_buf[256];
                          ... /* 在 pt_info 中布尔型,转换为字符串插入;例如 布尔
真转换为 "yes" */

```

```

    sprintf(query_buf, "INSERT INTO patient1 value ( \ '%s'
    \', \ '%d\' , \ '%s\' , \ '%s\' , \ '%s\' , \ '%s\' )
    ",
    "", pt_info->checkNo, insert_emergency_treat_mark,
    pt_info->name, pt_info->sample_date, pt_info->glucose)[5];

    if (mysql_query(myconnect1, query_buf) == 0); /* 数据 insert 表格 patient1 成功 */
    else ...; /* 数据 insert 失败 */
}

void mysql_database_table_select(struct patient_info * pt_info, gchar * select_conf)
{ /* 从数据库表格里选出符合要求的数据 */
    gchar query_buf[256]; guint8 i, j; guint32 total_line;
    /* MySQL 中有多种形式的 SELECT 语句, 此处选最简单的一种 */
    strcpy(query_buf, "SELECT * FROM patient1", 256);
    if (0 == mysql_query(myconnect1, query_buf))
    { res = mysql_store_result(myconnect1); /* res 为 MYSQL_RES * 类型的全局变量; */
        j = mysql_num_fields(res); /* 得到数据库表格中的总列数 */
        total_line = (guint32)mysql_num_rows(res); /* 得到符合查询条件总数 */
        for(i=0; i<j; i++)
        { fd = mysql_fetch_field(res); /* fd 为 MYSQL_FIELD * 类型的全局变量; */
            /* 可以使用输出函数查看到 fd 内容, 例如: fd->name 显示列名; fd->table 显示从属于哪个表名等等, 具体可以参考 MySQL 帮助手册 */
            while((row = mysql_fetch_row(res))) /* 取得结果集中的各数据行, 直到无行可以取为止; */
            { for(i=0; i<j; i++)
                /* row[i] 得到给定行对应的 j 列的数据, 然后将数据赋给 struct patient_info * pt_info, 例如如下: */
                if(3 == i) strcpy(pt_info->name, row[i], 32);
            }
            /* 已得到一个病人的信息, 将该病人信息传给 treeview 列表显示 */
            show_patient_info(pt_info);
            mysql_free_result(res); /* 释放目前 MySQL 数据库 query 返回所占用的内存 */
        }
        enum COL_INFO
        {
            COL_DATABASE_NO,
            COL_CHECK_NO,
            COL_EMERG_TREAT,
            COL_EMERG_COLOR,
            COL_NAME, COL_SAMPLE_DATE,
            COL_GLUCOSE,
            NUM_COLUMNS,
            COL_DEFAULT;
        };
    }

```

COL_NAME, COL_SAMPLE_DATE,
COL_GLUCOSE,
NUM_COLUMNS,
COL_DEFAULT};

从上可知: 分别初始化 MySQL 数据库和控件 GtkTreeView 之后, 调用 mysql_database_table_select 函数将数据库中的数据传给 show_patient_info 函数, 在此函数中完成了数据的显示。图 2 是用 glade 绘制界面, 用上面的流程做的部分界面。

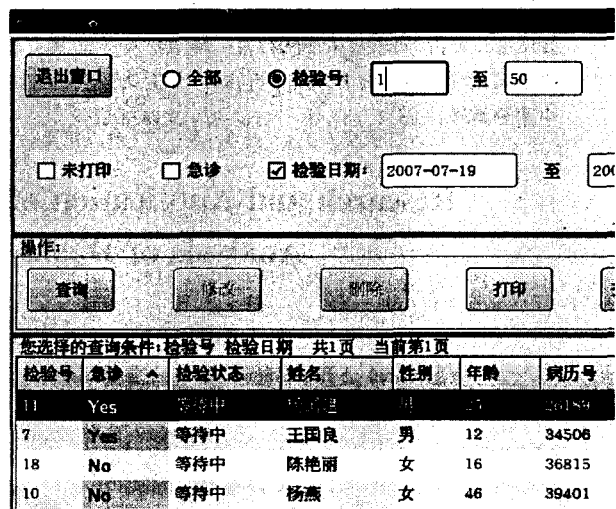


图 2 程序部分截图

3 结束语

GTK 中的 treeview 功能非常强大, 这里只介绍最常用的也是最稳定的一部分。读者可以参考 GTK 的 API 文档使用其他功能, 将 treeview 和 MySQL 结合起来使用。使得在现实应用中, 遇到相关的问题, 除了考虑使用 Windows 编程之外, 还多了一种其他的选择。这里只介绍了最基本的操作, 进一步比如 GtkTreeView 中右键操作等可以在此基础上进一步操作。

参考文献:

- [1] Wall K. GNU/Linux 编程指南[M]. 第2版. 北京: 清华大学出版社, 2002.
- [2] 黄 瑛, 刘少君. 基于 GTK+ 库数控机床界面的设计与实现[J]. 制造业自动化, 2005, 27: 53-57.
- [3] 宋国伟. GTK+ 2.0 编程范例[M]. 北京: 清华大学出版社, 2002: 106-308.
- [4] 吴向峰. GNOME/GTK+ 编程宝典[M]. 北京: 电子工业出版社, 2000.
- [5] 孔令富, 曹立强. 用 GTK+ 和 MySQL 开发数据库应用[J]. 计算机工程与科学, 2001, 23: 45-54.