第五部分　附　　录

附录A　GnomeHello源代码

这里是本书中多次用到的GnomeHello的源代码。它是学习Gnome编程非常好的材料。源代码共分为五个文件，分别是hello.c、app.c、app.h、menu.c以及menu.h。

A.1　第一部分：hello.c

```
#include <config.h>
#include <gnome.h>

#include "app.h"

static void session_die(GnomeClient* client, gpointer client_data);

static gint save_session(GnomeClient *client, gint phase,
                         GnomeSaveStyle save_style,
                        gint is_shutdown, GnomeInteractStyle interact_style,
                         gint is_fast, gpointer client_data);

static int greet_mode = FALSE;
static char* message  = NULL;
static char* geometry = NULL;

struct poptOption options[] = {
  {
    "greet",
    'g',
    POPT_ARG_NONE,
    &greet_mode,
    0,
    N_("Say hello to specific people listed on the command line"),
    NULL
  },
  {
    "message",
    'm',
    POPT_ARG_STRING,
    &message,
    0,
    N_("Specify a message other than \"Hello, World!\""),
    N_("MESSAGE")
  },
```

```
  {
    "geometry",
    '\0',
    POPT_ARG_STRING,
    &geometry,
    0,
    N_("Specify the geometry of the main window"),
    N_("GEOMETRY")
  },
  {
    NULL,
    '\0',
    0,
    NULL,
    0,
    NULL,
    NULL
  }
};

int
main(int argc, char* argv[])
{
  GtkWidget* app;

  poptContext pctx;

  char** args;
  int i;

  GSList* greet = NULL;

  GnomeClient* client;

  bindtextdomain(PACKAGE, GNOMELOCALEDIR);
  textdomain(PACKAGE);

  gnome_init_with_popt_table(PACKAGE, VERSION, argc, argv,
                             options, 0, &pctx);

  /* 参数分析*/

  args = poptGetArgs(pctx);

  if (greet_mode && args)
    {
      i = 0;
      while (args[i] != NULL)
        {
          greet = g_slist_prepend(greet, args[i]);
          ++i;
```

```
        }
      /* Put them in order */
      greet = g_slist_reverse(greet);
    }
  else if (greet_mode && args == NULL)
    {
      g_error(_("You must specify someone to greet."));
    }
  else if (args != NULL)
    {
      g_error(_("Command line arguments are only allowed with --greet."));
    }
  else
    {
      g_assert(!greet_mode && args == NULL);
    }

  poptFreeContext(pctx);

  /* 会话管理*/

  client = gnome_master_client ();
  gtk_signal_connect (GTK_OBJECT (client), "save_yourself",
                      GTK_SIGNAL_FUNC (save_session), argv[0]);
  gtk_signal_connect (GTK_OBJECT (client), "die",
                      GTK_SIGNAL_FUNC (session_die), NULL);

  app = hello_app_new(message, geometry, greet);

  g_slist_free(greet);

  gtk_widget_show_all(app);

  gtk_main();

  return 0;
}

static gint
save_session (GnomeClient *client, gint phase, GnomeSaveStyle save_style,
              gint is_shutdown, GnomeInteractStyle interact_style,
              gint is_fast, gpointer client_data)
{
  gchar** argv;
  guint argc;

  argv = g_malloc0(sizeof(gchar*)*4);
  argc = 1;

  argv[0] = client_data;
```

```
  if (message)
    {
      argv[1] = "--message";
      argv[2] = message;
      argc = 3;
    }

  gnome_client_set_clone_command (client, argc, argv);
  gnome_client_set_restart_command (client, argc, argv);

  return TRUE;
}

static void
session_die(GnomeClient* client, gpointer client_data)
{
  gtk_main_quit ();
}
```

A.2 第二部分：app.h

```
#ifndef GNOMEHELLO_APP_H
#define GNOMEHELLO_APP_H

#include <gnome.h>

GtkWidget* hello_app_new(const gchar* message,
                         const gchar* geometry,
                         GSList* greet);

void       hello_app_close(GtkWidget* app);

#endif
```

A.3 第三部分：app.c

```
#include <config.h>
#include "app.h"
#include "menus.h"

/* 保留一个所有打开的窗口的链表*/
static GSList* app_list = NULL;

static gint delete_event_cb(GtkWidget* w, GdkEventAny* e, gpointer data);
static void button_click_cb(GtkWidget* w, gpointer data);

GtkWidget*
hello_app_new(const gchar* message,
              const gchar* geometry,
              GSList* greet)
{
```

```
GtkWidget* app;
GtkWidget* button;
GtkWidget* label;
GtkWidget* status;
GtkWidget* frame;

app = gnome_app_new(PACKAGE, _("Gnome Hello"));

frame = gtk_frame_new(NULL);

button = gtk_button_new();

label  = gtk_label_new(message ? message : _("Hello, World!"));

gtk_window_set_policy(GTK_WINDOW(app), FALSE, TRUE, FALSE);
gtk_window_set_default_size(GTK_WINDOW(app), 250, 350);
gtk_window_set_wmclass(GTK_WINDOW(app), "hello", "GnomeHello");

gtk_frame_set_shadow_type(GTK_FRAME(frame), GTK_SHADOW_IN);

gtk_container_set_border_width(GTK_CONTAINER(button), 10);

gtk_container_add(GTK_CONTAINER(button), label);

gtk_container_add(GTK_CONTAINER(frame), button);

gnome_app_set_contents(GNOME_APP(app), frame);

status = gnome_appbar_new(FALSE, TRUE, GNOME_PREFERENCES_NEVER);

gnome_app_set_statusbar(GNOME_APP(app), status);

hello_install_menus_and_toolbar(app);

gtk_signal_connect(GTK_OBJECT(app),
                   "delete_event",
                   GTK_SIGNAL_FUNC(delete_event_cb),
                   NULL);

gtk_signal_connect(GTK_OBJECT(button),
                   "clicked",
                   GTK_SIGNAL_FUNC(button_click_cb),
                   label);

if (geometry != NULL)
  {
    gint x, y, w, h;
    if ( gnome_parse_geometry( geometry,
                               &x, &y, &w, &h ) )
      {
        if (x != -1)
```

```
            {
              gtk_widget_set_uposition(app, x, y);
            }

          if (w != -1)
            {
              gtk_window_set_default_size(GTK_WINDOW(app), w, h);
            }
        }
      else
        {
          g_error(_("Could not parse geometry string `%s'"), geometry);
        }
    }


  if (greet != NULL)
    {
      GtkWidget* dialog;
      gchar* greetings = g_strdup(_("Special Greetings to:\n"));
      GSList* tmp = greet;

      while (tmp != NULL)
        {
          gchar* old = greetings;

          greetings = g_strconcat(old,
                                  (gchar*) tmp->data,
                                  "\n",
                                  NULL);
          g_free(old);

          tmp = g_slist_next(tmp);
        }

      dialog = gnome_ok_dialog(greetings);

      g_free(greetings);

      gnome_dialog_set_parent(GNOME_DIALOG(dialog), GTK_WINDOW(app));
    }

  app_list = g_slist_prepend(app_list, app);

  return app;
}

void
hello_app_close(GtkWidget* app)
{
```

```
  g_return_if_fail(GNOME_IS_APP(app));

  app_list = g_slist_remove(app_list, app);

  gtk_widget_destroy(app);

  if (app_list == NULL)
    {
      /* No windows remaining */
      gtk_main_quit();
    }
}

static gint
delete_event_cb(GtkWidget* window, GdkEventAny* e, gpointer data)
{
  hello_app_close(window);

  /* 阻止销毁窗口，因为我们在hello_app_close中销毁它
   */
  return TRUE;
}

static void
button_click_cb(GtkWidget* w, gpointer data)
{
  GtkWidget* label;
  gchar* text;
  gchar* tmp;

  label = GTK_WIDGET(data);

  gtk_label_get(GTK_LABEL(label), &text);

  tmp = g_strdup(text);

  g_strreverse(tmp);

  gtk_label_set_text(GTK_LABEL(label), tmp);

  g_free(tmp);
}
```

A.4 第四部分：menus.h

```
#ifndef GNOMEHELLO_MENUS_H
#define GNOMEHELLO_MENUS_H

#include <gnome.h>

void hello_install_menus_and_toolbar(GtkWidget* app);
```

```
#endif
```

A.5 第五部分：menus.c

```
#include <config.h>
#include "menus.h"
#include "app.h"

static void nothing_cb(GtkWidget* widget, gpointer data);
static void new_app_cb(GtkWidget* widget, gpointer data);
static void close_cb  (GtkWidget* widget, gpointer data);
static void exit_cb   (GtkWidget* widget, gpointer data);
static void about_cb  (GtkWidget* widget, gpointer data);

static GnomeUIInfo file_menu [] = {
  GNOMEUIINFO_MENU_NEW_ITEM(N_("_New Hello"),
                            N_("Create a new hello"),
                            new_app_cb, NULL),

  GNOMEUIINFO_MENU_OPEN_ITEM(nothing_cb, NULL),

  GNOMEUIINFO_MENU_SAVE_ITEM(nothing_cb, NULL),

  GNOMEUIINFO_MENU_SAVE_AS_ITEM(nothing_cb, NULL),

  GNOMEUIINFO_SEPARATOR,

  GNOMEUIINFO_MENU_CLOSE_ITEM(close_cb, NULL),

  GNOMEUIINFO_MENU_EXIT_ITEM(exit_cb, NULL),

  GNOMEUIINFO_END
};

static GnomeUIInfo edit_menu [] = {
  GNOMEUIINFO_MENU_CUT_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_COPY_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_PASTE_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_SELECT_ALL_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_CLEAR_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_UNDO_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_REDO_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_FIND_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_FIND_AGAIN_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_REPLACE_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_MENU_PROPERTIES_ITEM(nothing_cb, NULL),
  GNOMEUIINFO_END
};

static GnomeUIInfo help_menu [] = {
```

```
  GNOMEUIINFO_HELP ("gnome-hello"),

  GNOMEUIINFO_MENU_ABOUT_ITEM(about_cb, NULL),

  GNOMEUIINFO_END
};

static GnomeUIInfo menu [] = {
  GNOMEUIINFO_MENU_FILE_TREE(file_menu),
  GNOMEUIINFO_MENU_EDIT_TREE(edit_menu),
  GNOMEUIINFO_MENU_HELP_TREE(help_menu),
  GNOMEUIINFO_END
};

static GnomeUIInfo toolbar [] = {
  GNOMEUIINFO_ITEM_STOCK (N_("New"),
                  N_("Create a new hello"),
                  nothing_cb, GNOME_STOCK_PIXMAP_NEW),

  GNOMEUIINFO_SEPARATOR,

  GNOMEUIINFO_ITEM_STOCK (N_("Prev"),
                   N_("Previous hello"),
                   nothing_cb, GNOME_STOCK_PIXMAP_BACK),
  GNOMEUIINFO_ITEM_STOCK (N_("Next"),
                   N_("Next hello"),
                   nothing_cb, GNOME_STOCK_PIXMAP_FORWARD),

  GNOMEUIINFO_END
};

void
hello_install_menus_and_toolbar(GtkWidget* app)
{
  gnome_app_create_toolbar_with_data(GNOME_APP(app), toolbar, app);
  gnome_app_create_menus_with_data(GNOME_APP(app), menu, app);
  gnome_app_install_menu_hints(GNOME_APP(app), menu);
}

static void
nothing_cb(GtkWidget* widget, gpointer data)
{
  GtkWidget* dialog;
  GtkWidget* app;

  app = (GtkWidget*) data;

  dialog = gnome_ok_dialog_parented(
         _("This does nothing; it is only a demonstration."),
         GTK_WINDOW(app));
```

```
}

static void
new_app_cb(GtkWidget* widget, gpointer data)
{
  GtkWidget* app;

  app = hello_app_new(_("Hello, World!"), NULL, NULL);

  gtk_widget_show_all(app);
}

static void
close_cb(GtkWidget* widget, gpointer data)
{
  GtkWidget* app;

  app = (GtkWidget*) data;

  hello_app_close(app);
}

static void
exit_cb(GtkWidget* widget, gpointer data)
{
  gtk_main_quit();
}

static void
about_cb(GtkWidget* widget, gpointer data)
{
  static GtkWidget* dialog = NULL;
  GtkWidget* app;

  app = (GtkWidget*) data;

  if (dialog != NULL)
    {
      g_assert(GTK_WIDGET_REALIZED(dialog));
      gdk_window_show(dialog->window);
      gdk_window_raise(dialog->window);
    }
  else
    {
      const gchar *authors[] = {
        "Havoc Pennington <hp@pobox.com>",
        NULL
      };

      gchar* logo = gnome_pixmap_file("gnome-hello-logo.png");
```

```
        dialog = gnome_about_new (_("GnomeHello"), VERSION,
                                  "(C) 1999 Havoc Pennington",
                                  authors,
                                  _("A sample GNOME application."),
                                  logo);

        g_free(logo);

        gtk_signal_connect(GTK_OBJECT(dialog),
                           "destroy",
                           GTK_SIGNAL_FUNC(gtk_widget_destroyed),
                           &dialog);

        gnome_dialog_set_parent(GNOME_DIALOG(dialog), GTK_WINDOW(app));

        gtk_widget_show(dialog);
    }
}
```