第15章 对 话 框

在Gtk+中，要使用对话框是很麻烦的。每当你要通知用户一些事情，都得创建一个窗口、几个按钮、一个标签，并将按钮、标签等组装到窗口上，然后设置回调函数。同时，还要捕获delete_event事件，等等。Gnome提供了一个容易使用的、通用的对话框构件和几个子构件，用它们可以创建通用对话框。Gnome还有几个使用模态对话框的函数。

15.1 GnomeDialog构件

因为存在于普通Gtk+中的对话框是一种权宜之计，有多少个程序员，就有多少种创建对话框的方法。程序员必须决定对话框放在屏幕的什么地方，构件之间的填充空白大小是多少，是否在按钮之间放置分隔线，按钮放在什么容器中，以及应该设置什么快捷键，等等。GnomeDialog构件的前提就是程序员不应该关心这些事，用户可以用它们想要的方法设置它们。以程序员的观点来说，对话框“能使用就可以了”。

15.1.1 创建对话框

创建GnomeDialog很简单。下面是基本步骤摘要，后面有详细介绍：

- 如果有适合需要的对话框子类，则使用相应的子类，并且可以跳过下面几步。
- 用gnome_dialog_new()函数创建对话框构件。将对话框的标题、每个按钮的名字作为参数传递到函数中去。
- 用需要的内容组装GNOME_DIALOG(dialog)->vbox。
- 规划一下对话框要做什么用。可以为close或clicked信号连接适当的回调函数。关闭对话框时，可以隐藏或销毁对话框，也可以在点击对话框时自动关闭它，或由你自己处理。有多种交互用户与对话框的方法，所以很重要的一点是要确信所选择的设置组合，在不论用户做什么时都能起作用。

用gnome_dialog_new()函数创建新对话框。第一个参数是对话框的标题，后面的参数表是一个以NULL结束的列表，表示要插入到对话框中的按钮。例如，可以像下面这样设置代码：

```
GtkWidget* dialog;
dialog = gnome_dialog_new(_("My Dialog Title"),
                          _("OK"),
                          _("Cancel"),
                          NULL);
```

上面的代码创建了一个标题为“My Dialog Title”，并且带有一个OK按钮和一个Cancel按钮。字符串用_()宏标志它可以被翻译。OK按钮会放在对话框的最左边。

函数列表：创建GnomeDialog构件

```
#include <libgnomeui/gnome-dialog.h>
GtkWidget* gnome_dialog_new(const gchar* title,
                            ...)
```

GnomeDialog API将添加的按钮从0开始编号。因为并没有自动创建一个指向按钮的指针，所以后面可以用这些编号引用按钮。在上面的情况中， OK按钮是0号，Cancel按钮是1号。

在上面的例子中，把按钮命名为 OK和Cancel按钮。Gnome为常用的按钮提供了一套“ 内置按钮 ”。这些按钮保证每个人都使用 OK按钮而不是Ok或OK!按钮。它们还让翻译器只翻译常用字符串一次，并且，它们一般在按钮上插入图标，使按钮更富吸引力，也更容易识别。如果可能，应尽量使用内置按钮。

在gnome_dialog_new()函数中使用内置按钮，用内置按钮宏替代按钮名：

```
dialog = gnome_dialog_new(_("My Dialog Title"),
                          GNOME_STOCK_BUTTON_OK,
                          GNOME_STOCK_BUTTON_CANCEL,
                          NULL);
```

Gnome包括许多内置按钮、内置菜单项和内置的像素映射图片。在 libgnomeui/gnome-stock.h.中有这些东西的详细列表。

15.1.2 填充对话框

创建对话框后，就可以在其中放置一些东西。如果只想放一个标签，也许应该使用GnomeMessageBox构件或一些其他函数 (例如gnome_ok_dialog()函数)，而不是手动创建对话框。填充对话框很简单：

```
GtkWidget* button;
/* ...假定前面已经创建了对话框dialog... */
button = gtk_button_new_with_label(_("Push Me"));
gtk_box_pack_start(GTK_BOX(GNOME_DIALOG(dialog)->vbox)),
                   button,
                   TRUE,
                   TRUE,
                   0);
```

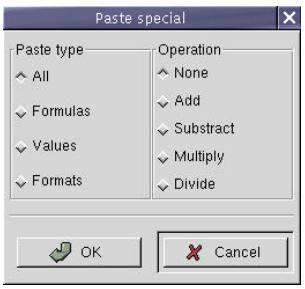

图15-1 Gnumeric 电子表格中的一个GnomeDialog对话框

图15-1显示了Gnumeric 电子表格软件中的对话框，它的各部分都加上了标签。

15.1.3 处理GnomeDialog的信号

创建对话框后，应该能够对用户的动作作出响应。下面是一个可能的动作的列表：

- 按Esc键关闭对话框。
- 点击窗口管理器的关闭按钮关闭对话框。
- 点击其中的某个按钮。
- 与对话框的内容进行交互。
- 如果对话框不是模态的，与应用程序的其他部分进行交互。

除了从它的父类继承的信号外， GnomeDialog还引发另外两个信号。如果用户点击其中的某个按钮，会引发一个 clicked点击信号 (这个信号不是GtkButton的clicked信号，它是一个不同的信号，由GnomeDialog引发)。GnomeDialog的clicked信号处理函数应该有三个参数：引发信号的对话框、被点击的按钮的编号，以及回调数据。

GnomeDialog还有一个close信号。当调用 gnome_dialog_close()函数时，引发这个信号。

所有的内建处理程序(例如，Esc快捷键)都调用这个函数来关闭对话框。GnomeDialog对close信号的缺省处理程序有两种可能的响应行为：对对话框调用 gtk_widget_hide()或者gtk_widget_destroy()函数。响应行为可以用gnome_dialog_close_hides()函数进行配置。

函数列表：关闭GnomeDialog对话框

```
#include <libgnomeui/gnome-dialog.h>
void gnome_dialog_close_hides(GnomeDialog* dialog,
                              gboolean setting)

void
gnome_dialog_set_close(GnomeDialog* dialog,
                       gboolean setting)
```

gnome_dialog_close_hides函数设置对话框接收到 close信号时的行为。如果第二个参数setting设为TRUE，则接收到close信号后，对话框不会关闭，而是隐藏起来。若设置为FALSE，对话框将立即关闭。

gnome_dialog_set_close函数的第二个参数 setting若设置为FALSE，对话框将不能关闭，设置为TRUE，则对话框可以关闭。

缺省情况下，close信号会销毁对话框。通常这也是我们所想要的，不过，如果创建对话框很耗时间，那么可以将它隐藏起来，使它需要时重新显示，而不是每次都销毁，需要时再创建。或许可以将对话框先隐藏起来，提取出每个构件的状态，然后用 gtk_widget_destroy()函数销毁它，具体做法依赖于代码的结构。然而，通常在点击对话框时销毁它很简单，不容易出错。要想知道构件在对话框中的状态，可以连接到 clicked信号。

如果给close信号连接一个处理函数，函数应该返回一个布尔值。如果返回 TRUE，将不会发生“隐藏”或“销毁”事件。你可以用这种方法阻止用户关闭对话框，例如，如果用户还没有填写一个表单中的所有域。

close信号用于收集用户的几种可能的动作，并连接到一个单一的处理函数上：当用户按Esc键或点击窗口管理器的关闭按钮时引发该信号。当用户点击对话框上的某个按钮时引发这个信号也会很方便。

注意，当对话框接受一个 delete_event事件时才引发close信号，这意味着只需要为对话框的所有关闭事件写一个处理函数，不需要对 delete_event事件单独对待。

15.1.4 最后的修饰

好对话框和伟大的对话框的区别在于细节上。GnomeDialog对话框带有一些特性，用它们可以很容易地修饰对话框。下面的函数列表集中介绍了这些函数。

函数列表：修饰GnomeDialog

```
#include <libgnomeui/gnome-dialog.h>
void gnome_dialog_set_parent(GnomeDialog* dialog,
                             GtkWindow* parent)

void gnome_dialog_set_default(GnomeDialog* dialog,
                              gint button)

void gnome_dialog_editable_enters(GnomeDialog* dialog,
```

```
                                GtkEditable* editable)

Void gnome_dialog_set_sensitive(GnomeDialog* dialog,
                                gint button,
                                gboolean setting)
```

对话框有一个逻辑上的双亲，通常是应用程序的主窗口。用 gnome_dialog_set_parent函数设置父子关系，这样 Gnome会尊重用户的设置，并向窗口管理器指明这种父子关系。大多数窗口管理器会在父窗口最小化时将子窗口也最小化，并让子窗口在父窗口上面。

只能将gnome_dialog_set_parent()函数和“暂时”对话框一起使用，这一点很重要。“暂时”对话框是指显示和消失相对速度较快的对话框。 GnomeDialog就是一个“暂时”对话框。一些对话框只是一个小窗口，比如说 Gimp上的工具选项板。这些稳固的 (“浮动”)对话框应该可以最小化而无需父窗口最小化，并且它们不应该强制停留在父窗口上。

对话框应该有一个敏感的缺省按钮 —— 当用户按回车键时激活该按钮。用 gnome_dialog_set_default() 函数可以指定缺省按钮。应该弄清楚将哪个按钮设置为缺省按钮。通常，最好的选择是最没有破坏性的动作 (例如，Cancel而不是OK按钮)，但是，如果几个按钮都没有什么破坏性，用户的习惯就是最好的选择。

典型情况下，退出应用程序或删除数据等操作用 Cancel或No按钮作为缺省按钮，要求用户输入文本或其他信息的对话框用 OK作为缺省按钮。在许多窗口管理器中，当窗口弹出时，该窗口会获得焦点，所以用户想对当前应用程序击键，可实际上却是对对话框的击键。如果对话框将“删除所有文件”选项作为缺省按钮，会有很多人咒骂你。

当在GtkEditable构件 —— 编辑构件（及其子构件）上按下回车键时，会引发一个 activate 信号。典型情况下，用户期望按回车键激活对话框的缺省按钮，但是如果对话框上有一个编辑构件，例如 GtkEntry，它会捕获回车键，对话框的缺省按钮不会对回车响应。当 GtkEditable 是激活的时，gnome_dialog_editable_enters() 函数激活对话框的缺省按钮，解决了上面的问题。

gnome_dialog_set_sensitive()函数实际上是对其中的按钮调用 gtk_widget_set_sensitive()函数。如果点击某个按钮时什么反应也没有，这个按钮肯定是不敏感的。

最后，应该保证没有创建一个对话框的多个实例。许多应用程序允许弹出多个 Preferences 或About对话框。用户并不经常引发这种错误，但是，最好能避免这样的问题。下面的代码用一种简单的方法处理这种问题 (注意，创建和显示对话框的代码省略掉了)。

```
void
do_dialog()
{
  static GtkWidget* dialog = NULL;

  if (dialog != NULL)
    {

      gdk_window_show(dialog->window);
      gdk_window_raise(dialog->window);
    }
  else
    {
```

```
        dialog = gnome_dialog_new();

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

        /* 此处写显示对话框，连接回调函数等代码 */
    }
}
```

gtk_widget_destroyed()是在gtk/gtkwidget.h中定义的，把它的第二个参数设置为 NULL就可以了。每次用户关闭对话框时，代码会重置对话框变量。如果用户在另一个对话框处活动状态时试图使用它，会将对话框提升为当前窗口，或将对话框从图标中恢复。注意，窗口管理器在提升或者恢复方面有一定的发言权，所以不能保证上面的情况一定会发生。

15.2 模态对话框

有时，当用户与对话框打交道时，要阻止用户与应用程序的其他部分交互。“冻结”了应用程序的其他部分的对话框称为“模态”对话框。

什么时候使用模态对话框还有许多争论。一些用户非常讨厌它，但有许多时候它还是很必要的。为模态对话框写代码很容易，因为你能够在函数的中间停下来，等着用户做出响应，然后继续。对非模态对话框，必须将控制转交给主应用程序，并安排回调函数，以便用户最后处理对话框时能够回到对话框交出控制权的地方。在一个复杂的对话框中，这样做的结果就是丑陋的面条式的代码。这诱使许多程序员不管什么时候都使用模态对话框，或者至少是经常使用。

如果当用户使用对话框时还想回过头在应用程序主窗口中查阅一些信息，或者用户想在主窗口和对话框之间剪切/粘贴，要避免使用模态对话框。“属性”对话框通常是非模态的，因为用户也许想试验一下他们所做的改变的效果，而不想关闭对话框。没有理由将无足轻重的信息对话框设为模态的，因为在它们上面点击对应用程序的其余部分没有影响。

然而，如果有必要，也不要害怕使用模态对话框。 Gnome的文件管理器的“文件属性”对话框是模态的。如果不是这样，用户可能在正在编辑某个文件的属性时将文件删除。这样用户会很困惑。没有什么法则来决定应该使用何种对话框，这完全靠程序员的感觉。

总而言之，创建模态对话框非常容易。在 Gtk+中，任何窗口都可以用 gtk_window_set_modal()设置为模态的。

函数列表：模态窗口

```
#include <gtk/gtkwindow.h>
gtk_window_set_modal(GtkWindow* window,
                     gboolean modality)
```

因为GnomeDialog是一个GtkWindow子类，所以你可以用这个函数设置模态窗口。它阻止与模态对话框以外的窗口进行交互。

典型情况下，你可能想更进一步，如等待用户点击对话框的某个按钮，而不用设置多个回调函数。在Gtk+中这是通过执行gtk_main()的另一个实例，它是通过进入另一个嵌套的事件循环来实现的。第二个循环退出时，控制权返回到刚才调用的 gtk_main()后面。然而，由于有

大量的关闭对话框的方法，也就有很多的复杂因素和先决条件，最后的代码可能把程序员自己弄糊涂，并且也容易发生错误。下面的两个函数能解决这个问题。

函数列表：“运行”一个对话框

```
#include <libgnomeui/gnome-dialog.h>
gint gnome_dialog_run(GnomeDialog* dialog)
gint gnome_dialog_run_and_close(GnomeDialog* dialog)
```

执行这两个函数时，程序将停止运行，直到用户点击了一个对话框按钮、点击了窗口管理器的关闭按钮，或者使用快捷键做了同样的事为止。如果点击了某个按钮，它们返回按钮的编号。请注意，GnomeDialog的按钮是从左到右由0开始编号的。如果没有点击按钮(对话框是由窗口管理器关闭的)，它们将返回-1。

在上面的函数调用过程中，对话框自动设置为模态的，否则会引起混乱。例如，从主应用程序代码中调用gtk_main_quit()函数会退出嵌套的gtk_main()，而不是应用程序的gtk_main()。然而，如果计划在gnome_dialog_run()返回之后让对话框依然呈打开状态，并且想让它是模态的，应该手动将它设置为模态，gnome_dialog_run()只是临时改变对话框的模态属性。

作为程序员，有责任在调用 gnome_dialog_run()之前解决如何关闭或者销毁对话框问题。你可以设置使用户的任何动作都不能销毁它，然后在 gnome_dialog_run()返回之后再销毁它。或者，设置对话框使所有的用户对话框都可以销毁它，然后在 gnome_dialog_run()返回之后将它完全忘记。还可以写一个循环，重复调用 gnome_dialog_run()函数直到用户输入了合法值，并且只有在循环结束之后才能关闭对话框。如果写一个循环，要注意手工将对话框设置为模态，否则它可能会在某个一个较短间隔内是非模态的，这样会引起混乱。

gnome_dialog_run_and_close()监视对话框的close和destroy信号，当且仅当对话框没有响应用户点击或击键关闭它时才关闭对话框。使用这个函数保证了 gnome_dialog _close()函数在返回之前只会调用一次。实际上，gnome_dialog_run_and_close()没什么大用处，它不过是一种避免考虑如何关闭对话框的方法而已。

15.3 一个对话框示例

下面是从一个Gnome绘图和制表组件Guppi中摘取的一段代码。这是一个用于“打开”文件的对话框。今后的Gnome版本会有一个GnomeFileSelection构件，对这样的任务它比自己定制一个对话框更合适，但是这里的例子还是有一定的指导作用的。

```
GtkWidget * dialog;
dialog = gnome_dialog_new(_("Guppi: Open"),
                          GNOME_STOCK_BUTTON_OK,
                          GNOME_STOCK_BUTTON_CANCEL,
                          NULL);
gnome_dialog_set_close(GNOME_DIALOG(dialog), TRUE);
gnome_dialog_close_hides(GNOME_DIALOG(dialog), TRUE);

guppi_setup_dialog(dialog);

GtkWidget* fileentry =
  gnome_file_entry_new("guppi:guppi_loadsave_history",
                       _("Guppi: Browse Files For Open"));
```

```
gnome_dialog_editable_enters(GNOME_DIALOG(dialog),
GTK_EDITABLE(gnome_file_entry_gtk_entry(GNOME_FILE_ENTRY(fileentry))));
gnome_dialog_set_default(GNOME_DIALOG(dialog), GNOME_OK);

gtk_box_pack_start(GTK_BOX(GNOME_DIALOG(dialog)->vbox),
                   fileentry,
                   TRUE, TRUE, GNOME_PAD);


gtk_widget_show_all(dialog);

int reply = gnome_dialog_run(GNOME_DIALOG(dialog));

if (reply == GNOME_OK)
  {
    gchar* s =
      gnome_file_entry_get_full_path(GNOME_FILE_ENTRY(fileentry),
                                     TRUE);
    /* 此处是应用程序中用于加载文件的详细代码
     * 因为与本节无关，此处略去
     */
  }

gtk_widget_destroy(dialog);
```

此示例中调用了gnome_dialog_set_close()，所以如果点击对话框的任何按钮，对话框都会关闭。然而，这里关闭对话框只是调用了gtk_widget_hide()函数，而不是销毁它。这种行为是由gnome_dialog_close_hides()函数配置的。guppi_setup_dialog()是gnome_dialog_set_parent()函数的一个封装，它将应用程序的主窗口设置为对话框的父窗口。

因为对话框的作用是取得一个文件名，按回车键时按下OK按钮是很方便的，因而，OK按钮应该是缺省按钮。但是，通常文本输入框会截取回车键，而gnome_dialog_editable_enters()正好可以用来修正这个问题。gnome_dialog_run()等待用户采取行动，如果点击“OK”按钮，我们提取文本输入框中的内容，并加载相应的文件。注意，因为调用了gnome_dialog_close_hides()函数，所以gnome_dialog_run()返回之后，对话框并不会销毁。然而，gnome_dialog_run()返回之后，对话框会关闭，因为代码保证了所有的用户行为都会关闭它(使用gnome_dialog_set_close()函数，依靠窗口管理器的关闭按钮的缺省行为)。

最后，gtk_widget_destroy()函数是必要的，因为当关闭对话框时，并没有销毁它。

15.4 特殊对话框

本节介绍一些特殊类型的对话框，它们都是GnomeDialog的子类，使用这些对话框很方便，并且能够保持用户界面的一致性。当然，前面介绍的关于GnomeDialog的一些特性也适用于它的子类。

15.4.1 GnomeAbout

Gnome应用程序应该有一个“关于...”菜单项，用来显示一个关于对话框。Gnome提供的

GnomeAbout可以用来显示一些关于应用程序的信息，比如作者、版权信息等。 GnomeAbout使用非常容易，只涉及到一个函数：gnome_about_new()。它的参数分别是：应用程序的标题、应用程序的版本、一行的版权信息、一个以 NULL结尾的供应商作者的名字，一个小段落(用来显示一些信息)，以及一个用来显示应用程序徽标的图像文件名。只有作者参数是必要的，其他的都可以是NULL，但是如果其他的内容都是NULL，设置“ 关于对话框 ”没有什么意义。图15-2所示的是Glade的“ 关于本程序 ”对话框。

图15-2 关于对话框GnomeAbout

创建GnomeAbout ：

```
#include <libgnomeui/gnome-about.h>
GtkWidget* gnome_about_new(const gchar* title,
              const gchar* version,
              const gchar* copyright,
              const gchar** authors,
              const gchar* comments,
              const gchar* logo)
```

点击“ 确定 ”按钮时， GnomeAbout会自动关闭，所以根本不需为此操心。你只需创建、显示它就可以了。记住，要确保任何时候都只有一个实例在运行。

下面是一个显示关于对话框的菜单项，它来自于 Gnome中的Calendar(日历)程序：

```
static void
about_calendar_cmd (GtkWidget *widget, void *data)
{
   GtkWidget *about;
   const gchar *authors[] = {
     "Miguel de Icaza <miguel@kernel.org>",
     "Federico Mena <federico@gimp.org>",
     "Arturo Espinosa <arturo@nuclecu.unam.mx>",
     NULL
   };

   about = gnome_about_new (_("Gnome Calendar"), VERSION,
                        "(C) 1998 the Free Software Foundation",
                        authors,
                        _("The GNOME personal
                        calendar and schedule manager."),
```

```
                                NULL);
    gtk_window_set_modal (GTK_WINDOW (about), TRUE);

    gtk_widget_show (about);
}
```

上面的VERSION宏来自于config.h头文件，它是由configure脚本定义的。Gnome Calendar的作者选择将对话框设置为模态来防止多个对话框的实例运行 —— 对话框打开时用户不能再选择菜单项。

15.4.2 GnomePropertyBox —— 属性框

GnomePropertyBox用于应用程序的参数设置，或者用于编辑用户可见对象的属性。它是一个对话框，内部有一个GtkNootbook构件和四个按钮：OK、Apply、Close、Help。OK按钮等价于点击Apply然后点击Close。放在GnomePropertyBox中的构件用于设置属性或参数值，点击Apply按钮可以让用户请求的改变立即生效。 Help按钮用来显示帮助。 OK和Close是由属性框自动处理的，所有可以忽略它们。

不需要直接处理属性框的按钮，因为GnomePropertyBox会引发“ apply ”和“ help ”信号。这两个信号的处理函数应该是这个样子：

```
void handler(GtkWidget* propertybox, gint page_num, gpointer data);
```

其中page_num是对话框中的GtkNotebook构件中的当前活动页(GtkNotebook是从前往后，从0开始编号的。前页指的是加到GtkNotebook 中的第一页)。点击“ 帮助 ”按钮时，可以根据页号提供一个上下文敏感的帮助。当用户点击 Apply或OK按钮时，对每一页引发一次apply信号，然后最后引发一个信号，并且以-1为page_num值。

要创建一个属性框，首先创建一个对话框，然后创建每一页，添加到对话框中。用gnome_property_box_new()创建GnomePropertyBox，这个函数不带参数。

函数列表：创建GnomePropertyBox

```
#include <libgnomeui/gnome-propertybox.h>
GtkWidget* gnome_property_box_new()
Gint gnome_property_box_append_page(GnomePropertyBox* pb,
                                    GtkWidget* page,
                                    GtkWidget* tab)
```

然后为每一页创建一个构件 (可能是一个里面包含许多构件的容器)，然后将它用gnome_property_box_append_page()函数添加到属性框中。它的page参数就是放在新笔记本页中的构件， tab参数是用在笔记本标签页上的构件。函数返回新增页的页号，所以不用自己计数。

作为程序员，有责任跟踪用户与每一页内容的任何交互动作。当用户改变了一个设置时，必须通知属性框。当且仅当有变化没有生效时，属性框才会用这些信息设置 Apply和OK按钮的敏感性。

属性框状态相关的函数：

```
#include <libgnomeui/gnome-propertybox.h>
void gnome_property_box_changed(GnomePropertyBox* pb)
void gnome_property_box_set_state(GnomePropertyBox* pb,
                                  gboolean setting)
```

gnome_property_box_changed()函数告诉属性框发生的变化。当apply信号引发时，属性框会自动取消内部的“变化未决”标志。如果需要改变这个内部标志 (不太可能会)，可以使用gnome_property_box_set_state()函数。

15.4.3 GnomeMessageBox——消息框

GnomeMessageBox是GnomeDialog的一个子类，它传递一个短消息，或者向用户询问一个简单的问题。Gnome提供了几种类型的消息框，它们的图标和相应的标题在显示的文本前都有所不同。图标可以使用户快速地将显示的消息分类。

GnomeMessageBox的API非常简单，它除了构建函数以外，没有针对 GnomeMessageBox的其他函数。第一个参数是要显示的消息；第二个参数是指定消息框类型的字符串。然后，列出任何按钮，就像gnome_dialog_new()中的一样。不像未加修饰的GnomeDialog， 缺省情况下，在GnomeMessageBox上点击任何按钮都会关闭消息框。当然，可以用 gnome_dialog_set_close()函数改变这种响应方式。

函数列表： 消息框

```
#include <libgnomeui/gnome-messagebox.h>
GtkWidget* gnome_message_box_new(const gchar* message,
                  const gchar* messagebox_type,
                  ...)
```

用下面的宏指定消息框的类型：

- GNOME_MESSAGE_BOX_INFO：应该用来显示“供参考”的消息。
- GNOME_MESSAGE_BOX_WARNING：应该用来显示非致命错误。
- GNOME_MESSAGE_BOX_ERROR：如果操作完全失败，使用这种类型的消息框。
- GNOME_MESSAGE_BOX_QUESTION：如果要问一个问题，使用这种消息框。
- GNOME_MESSAGE_BOX_GENERIC：如果上面的都不适用，使用这种消息框。

下面是GnomeMessageBox的用法：

```
GtkWidget * mbox;
mbox = gnome_message_box_new (message,
                              GNOME_MESSAGE_BOX_INFO,
                              GNOME_STOCK_BUTTON_OK,
                              NULL);
gtk_widget_show (mbox);
```

注意 GnomeMessageBox，像GnomeDialog的大多数子类一样，当点击时它会自动关闭。所以不用手工销毁它。

实用对话框

消息框一般用于向用户显示某些信息，比如提示、警告和错误等。因为消息框差不多总是有同样的按钮(单个OK)，Gnome提供了几个实用函数处理这种情况。每一个函数都有一个_parented()后缀的变体，它调用gnome_dialog_set_parent()函数设置父窗口。下面的函数列表中的三个函数对分别显示一个信息框、警告框和一个错误框。它们创建并显示对话框，所以，如果愿意可以忽略返回值。

这些函数的唯一目的就是节省打字输入的时间。

函数列表：实用对话框

```
#include <libgnomeui/gnome-dialog-util.h>
```

显示一个对话框，其中有一个“ OK ”按钮，并显示一段消息：

```
GtkWidget* gnome_ok_dialog(const gchar* message)
```

与上面的对话框类似，但是可以为它指定一个父窗口：

```
GtkWidget* gnome_ok_dialog_parented(const gchar* message,
                                    GtkWindow* parent)
```

对话框，显示一条消息，并带一个警告图标：

```
GtkWidget* gnome_warning_dialog(const gchar* warning)
```

与上面的对话框类似，但是可以为它指定一个父窗口：

```
GtkWidget* gnome_warning_dialog_parented(const gchar* warning,
                                         GtkWindow* parent)
```

对话框，显示一条消息，并带一个“错误”图标：

```
GtkWidget* gnome_error_dialog(const gchar* error)
```

与上面的对话框类似，但是可以为它指定一个父窗口：

```
GtkWidget* gnome_error_dialog_parented(const gchar* error,
                                       GtkWindow* parent)
```