第14章　状态条构件

14.1　状态条构件简介

Gtk+构件库中有一个状态条构件　GtkStatusbar，Gnome构件库中也有一个状态条GnomAppBar。这两者之间没有多大差别，所以选择哪一个构件并没有什么特别的关系。状态条一般用来显示一些提示性的信息。因为有的用户，特别是新用户可能根本就注意不到状态条上的信息，因此，不能将在状态条上显示信息（特别是重要信息）作为唯一的提示方式。

为GnomeApp构件添加状态条很简单。只需调用 gnome_app_set_statusbar 函数，并将第二个参数设置为已经创建好的 statusbar构件。当鼠标指向某个菜单时，可以用状态条显示菜单的帮助。Gnome有几个很方便的函数可以实现这种功能。

函数列表：设置状态条

```
#include <libgnomeui/gnome-app.h>
void
gnome_app_set_statusbar(GnomeApp* app,
                        GtkWidget* statusbar)
```

14.2　GnomeAppBar构件

并没有特别的理由选择是用 GnomeAppBar还是GtkStatusbar作为状态条，主要区别在于它们拥有不同的API函数。GnomeAppBar构件是后写的，目的在于以下几点：

- 简化GtkStatusbar 构件的API调用。
- 支持Netscape风格的状态条，在状态条上显示一个进度条。
- 最终目的是要支持像Emacs编辑器的“ minibuffer ”功能那样的交互功能。不过，这个功能在Gnome 1.0中还没有实现。

用gnome_appbar_new()函数能够创建 GnomeAppBar构件。用这个构建函数还可以配置GnomeAppBar构件的功能：有或者没有进度条，有或者没有状态文本区，可以或不可以与用户交互。注意，必须有一个进度条或状态文本区。其中，　GnomePreferencesType是一种扩展型的布尔值：

- GNOME_PREFERENCES_NEVER 表明GnomeAppBar 构件是不可交互的。
- GNOME_PREFERENCES_USER 表明如果用户已经在Gnome环境设置中激活这种特性，GnomeAppBar就是交互的。
- GNOME_PREFERENCES_ALWAYS 表明GnomeAppBar 总是可交互的。

Gnome 1.0还没有完全实现交互性，所以要避免使用 GNOME_PREFERENCES_ALWAYS.。还有一些实验性的Gnome函数，可以用于提取某些用户交互动作，并允许用户在对话框和Emacs风格的“ minibuffer ”之间作出选择。当 Gnome得到进一步发展后，　GNOME_PREFERENCES_USER 会起作用，即使并没有明确使用“ 交互性 ”。建议将 GnomePreferencesType设置为GNOME_PREFERENCES_USER。

函数列表： 创建GnomeAppBar构件

```
#include <libgnomeui/gnome-appbar.h>
GtkWidget*
gnome_appbar_new(gboolean has_progress,
                 gboolean has_status,
                 GnomePreferencesType interactivity)
```

GnomeAppBar的用法很简单。进度条元素代表一个 GtkProgress接口，要使用该接口，只需用 gnome_appbar_get_progress()函数将 GtkProgress 提取出来，然后就可以使用与GtkProgress构件的相关函数对它进行操作了。注意，不要假想 Progress Bar接口是GtkProgress的子类；不要将它转换为GtkProgressBar类型的指针。

函数列表： 提取GtkProgress

```
#include <libgnomeui/gnome-appbar.h>
GtkProgress* gnome_appbar_get_progress(GnomeAppBar* appbar)
```

状态文本存储在一个栈中。当 GnomeAppBar 刷新时，显示栈中最上面的元素。每次对栈进行操作时，GnomeAppBar 都会刷新。所以将状态文本压入栈时，该文本就会显示出来。

状态文本还有另外两种设置方法。你可以设置一些“缺省”文本，如果栈是空的，会显示缺省文本。缺省的“缺省”文本是空字符串。你还可以仅设置状态文本而不改变栈，则“暂时”文本立即显示在状态文本区，但不存储在栈中。在下次刷新时 (下次压入、弹出或设置缺省文本时)，该文本会永久消失，并被栈顶的值所取代。

下面的函数列表列出了操纵状态文本的函数。 gnome_appbar_set_status()函数用于设置“暂时”状态文本；gnome_appbar_refresh()强行刷新而不改变栈 —— 这样可以保证“暂时”文本已经清除。其他的函数意义都很明显。

注意 可以将GnomeAppBar用作简单的标签，它一次显示一条信息，且总是取代前一条信息，只要设置缺省文本或暂时文本就可以了，根本不需使用栈。

函数列表： 设置GnomeAppBar的文本

```
#include <libgnomeui/gnome-appbar.h>
void gnome_appbar_set_status(GnomeAppBar* appbar,
                             const gchar* status)
void gnome_appbar_set_default(GnomeAppBar* appbar,
                              const gchar* default_status)
void gnome_appbar_push(GnomeAppBar* appbar,
                       const gchar* status)
void gnome_appbar_pop(GnomeAppBar* appbar)
void gnome_appbar_clear_stack(GnomeAppBar* appbar)
void gnome_appbar_refresh(GnomeAppBar* appbar)
```

14.3 状态条构件GtkStatusbar

GtkStatusbar是一个简单的构件，一般用来显示文本消息。它将文本消息压入到一个栈里面，当弹出当前消息时，将重新显示前一条文本消息。

为了让应用程序的不同部分使用同一个状态条显示消息，状态条构件使用上下文标识符来识别不同“用户”。在栈顶部的消息就是要显示的消息，不管它的上下文是什么。消息在栈里面是以先进后出的方式保存的，而不是按上下文标识符顺序。

状态条构件用下面的函数创建：

```
GtkWidget *gtk_statusbar_new( void );
```

用一个上下文的简短文本描述调用下面的函数，可以获得新的上下文标识符：

```
guint gtk_statusbar_get_context_id( GtkStatusbar *statusbar,
                                    const gchar  *context_description );
```

有三个函数用来操作状态条：

```
guint gtk_statusbar_push( GtkStatusbar *statusbar,
                          guint         context_id,
                          gchar        *text );
void gtk_statusbar_pop( GtkStatusbar *statusbar)
                        guint         context_id );
void gtk_statusbar_remove( GtkStatusbar *statusbar,
                           guint         context_id,
                           guint         message_id );
```

第一个函数gtk_statusbar_push用于将新消息加到状态栏中，它返回消息的上下文标识符。这个标识符可以用在 gtk_statusbar_remove函数中将该消息从状态条的栈中删除。函数gtk_statusbar_pop删除在栈中给定上下文标识符的最上面的一条消息。

下面的例子创建了一个状态条和两个按钮，一个将消息压入到状态条栈中，另一个将最上面一条消息弹出。

```
/* 状态条示例开始 statusbar.c */
#include <gtk/gtk.h>
#include <glib.h>

GtkWidget *status_bar;
void push_item (GtkWidget *widget, gpointer data)
{
  static int count = 1;
  char buff[20];
  g_snprintf(buff, 20, "Item %d", count++);
  gtk_statusbar_push( GTK_STATUSBAR(status_bar),
                      GPOINTER_TO_INT(data), buff);
  return;
}

void pop_item (GtkWidget *widget, gpointer data)
{
  gtk_statusbar_pop( GTK_STATUSBAR(status_bar),
                     GPOINTER_TO_INT(data) );
  return;
}

int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *vbox;
    GtkWidget *button;
    gint context_id;
```

```
    gtk_init (&argc, &argv);

    /* 创建新窗口 */
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_widget_set_usize( GTK_WIDGET (window), 200, 100);
    gtk_window_set_title(GTK_WINDOW (window), "GTK Statusbar Example");
    gtk_signal_connect(GTK_OBJECT (window), "delete_event",
                       (GtkSignalFunc) gtk_exit, NULL);
    vbox = gtk_vbox_new(FALSE, 1);
    gtk_container_add(GTK_CONTAINER(window), vbox);
    gtk_widget_show(vbox);
    status_bar = gtk_statusbar_new();
    gtk_box_pack_start (GTK_BOX (vbox), status_bar, TRUE, TRUE, 0);
    gtk_widget_show (status_bar);

    context_id = gtk_statusbar_get_context_id(
                    GTK_STATUSBAR(status_bar),
                    "Statusbar example");

    button = gtk_button_new_with_label("push item");
    gtk_signal_connect(GTK_OBJECT(button), "clicked",
        GTK_SIGNAL_FUNC (push_item), GINT_TO_POINTER(context_id) );
    gtk_box_pack_start(GTK_BOX(vbox), button, TRUE, TRUE, 2);
    gtk_widget_show(button);

    button = gtk_button_new_with_label("pop last item");
    gtk_signal_connect(GTK_OBJECT(button), "clicked",
        GTK_SIGNAL_FUNC (pop_item), GINT_TO_POINTER(context_id) );
    gtk_box_pack_start(GTK_BOX(vbox), button, TRUE, TRUE, 2);
    gtk_widget_show(button);
    /* 将窗口最后显示，以防止屏幕闪烁 */
    gtk_widget_show(window);
    gtk_main ();
    return 0;
}
     /*示例结束 */
```

将上面的代码保存为statusbar.c，然后编写一个如下所示的Makefile文件。

```
CC = gcc
statusbar: statusbar.c
   $(CC) `gtk-config --cflags`  statusbar.c \
            -o statusbar `gtk-config --libs`
clean:
   rm -f *.o statusbar
```

编译后，运行结果如图 14-1所示。按下push item按钮，向状态条栈中压入一条消息，按pop last item按钮，弹出最后一条消息，在状态条上显示下一条消息。

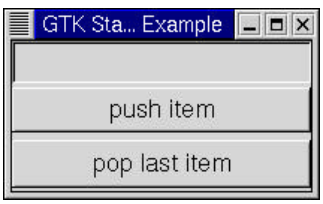

图14-1 状态条示例