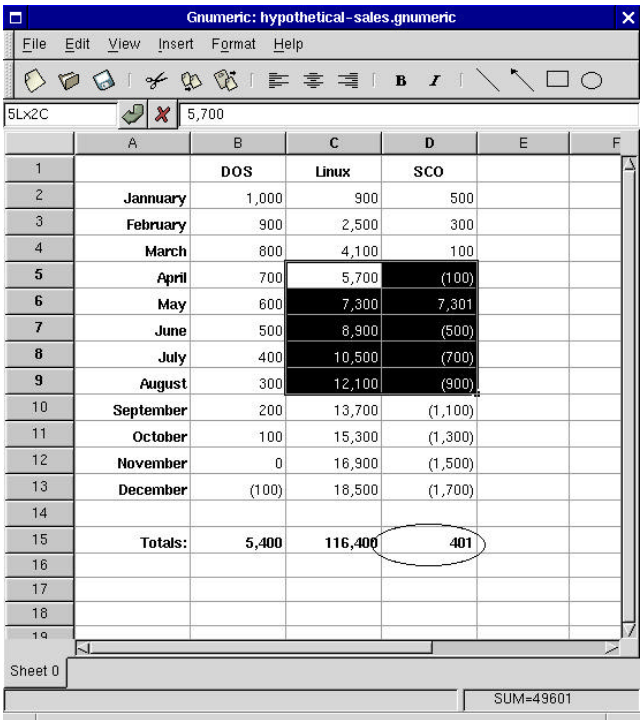
第13章 GnomeApp构件和GnomeUIInfo

13.1 主窗口GnomeApp

所有的Gnome应用程序，除极少数有特殊需要的以外，都可以用GnomeApp作为其主窗口。GnomeApp构件是GtkWindow的子类，它在基本的顶级窗口上增加了很方便的菜单和工具条处理能力，如图13-1所示。用户可以配置GnomeApp构件的下列特性：

- 菜单和工具条可以与窗口分离开，或者在窗口上重新安排位置。
- 用户可以选择禁止菜单和工具条与Gnome应用程序的窗口分离。
- 用户可以选择是否在应用程序的菜单上显示小图标。

今后，GnomeApp构件还会添加更多的特性。



	A	B	C	D	E	F
1		DOS	Linux	SCO		
2	Jannuary	1,000	900	500		
3	February	900	2,500	300		
4	March	800	4,100	100		
5	April	700	5,700	(100)		
6	May	600	7,300	7,301		
7	June	500	8,900	(500)		
8	July	400	10,500	(700)		
9	August	300	12,100	(900)		
10	September	200	13,700	(1,100)		
11	October	100	15,300	(1,300)		
12	November	0	16,900	(1,500)		
13	December	(100)	18,500	(1,700)		
14						
15	Totals:	5,400	116,400	401		
16						
17						
18						

图13-1 Gnumeric电子表格软件，用GnomeApp构件创建界面

GnomeApp有一个与其他构件类似的创建函数，见下面的函数列表。第一个参数 app_id是一个Gnome与应用程序打交道的内部名称。它应该与传递到 gnome_init()函数中的app_id完全一样，一般来说，可以使用应用程序可执行文件的名称。第二个参数是应用程序窗口的标题，如果设为NULL，则不为窗口设置标题。

```
#include <libgnomeui/gnome-app.h>
GtkWidget* gnome_app_new(gchar* app_id,
```

```
                gchar* title)
```

GnomeApp构件有一个唯一的“内容区”，你可以将应用程序的主要功能放在该区域。在中心区域的四边，可以放置工具条、菜单条、状态条等。函数列表列出了相关的函数。

这些函数很容易理解，一般从字面上就可以知道其作用。它们的主要作用是在 GnomeApp 构件的合适位置放置所需要的构件。为 GnomeApp创建菜单条、工具条、状态条也有很简单的方法。

添加构件到GnomeApp上，可输入以下语句：

```
#include <libgnomeui/gnome-app.h>
/* 将构件contents添加到GnomeApp构件app的窗口上*/
void gnome_app_set_contents(GnomeApp* app,
                      GtkWidget* contents)

/* 将菜单条menubar添加到GnomeApp构件app的窗口上*/
void gnome_app_set_menus(GnomeApp* app,
                   GtkMenuBar* menubar)

/* 将工具条toolbar添加到GnomeApp构件app上*/
void gnome_app_set_toolbar(GnomeApp* app,
                     GtkToolbar* toolbar)

/* 将状态条statusbar添加到GnomeApp构件上*/
void gnome_app_set_statusbar(GnomeApp* app,
                       GtkWidget* statusbar)
```

13.2 GnomeUIInfo

13.2.1 创建GnomeUIInfo

Gtk提供了两种方法来为应用程序创建菜单。但是，用这两种方法创建一个很大的菜单是很冗长乏味的，特别是如果菜单还带有图标和快捷键时更是如此。 Gnome提供了一个简单的解决方案。为每一个菜单项创建一个 GnomeUIInfo 模板，并列出它的一些特性：名称、图标、快捷键等。 Gnome库函数会自动地用 GnomeUIInfo数组模板创建菜单。你也可以用同样的方法创建工具条。

下面是GnomeUIInfo. 结构的声明：

```
typedef struct {
  GnomeUIInfoType type;
  gchar* label;
  gchar* hint;
  gpointer moreinfo;
  gpointer user_data;
  gpointer unused_data;
  GnomeUIPixmapType pixmap_type;
  gpointer pixmap_info;
  guint accelerator_key;
  GdkModifierType ac_mods;
  GtkWidget* widget;
} GnomeUIInfo;
```

填充上面的结构最方便的方法就是用一段静态的初始化程序 (当然，如果愿意，也可以动态地创建)。Gnome函数可以接受一个GnomeUIInfo数组，同时，还有一些宏可以简化、标准化最常用的静态初始化程序。下面是一个典型的例子——一个“文件”菜单：

```
static GnomeUIInfo file_menu[] = {
GNOMEUIINFO_MENU_NEW_ITEM(N_("_New Window"),
                          N_("Create a new text viewer window"),
                          new_app_cb, NULL),

/*"打开"（Open）菜单项*/
GNOMEUIINFO_MENU_OPEN_ITEM(open_cb,NULL),

/*"另存为"（Save As）菜单项*/
GNOMEUIINFO_MENU_SAVE_AS_ITEM(save_as_cb,NULL),

/*分隔线*/
GNOMEUIINFO_SEPARATOR,

/*"关闭"（Close）菜单项*/
GNOMEUIINFO_MENU_CLOSE_ITEM(close_cb,NULL),

/*"退出"（Exit）菜单项*/
GNOMEUIINFO_MENU_EXIT_ITEM(exit_cb,NULL),

/*菜单结束*/
GNOMEUIINFO_END
};
```

然而，大多数情况下，是不能用上面的宏来创建菜单的，所以有时候还得自己动手指定结构的每一个成员：

```
{
  GNOME_APP_UI_ITEM, N_("_Select All"),
  N_("Select all cells in the spreadsheet"),
  select_all_cb, NULL,
  NULL, 0, 0,'a', GDK_CONTROL_MASK
}
```

下面我们简要介绍GnomeUIInfo结构中各个成员的含义：

- type 是一个GnomeUIInfoType枚举类型的值，也是类型标记，参看表 13-1。
- label是菜单或工具条按钮上的文本。一般情况下，它应该用 N_()宏作国际化标注。
- hint 是菜单项或按钮功能的描述，对工具条来说，它以工具提示文本的形式显示。对菜单条来说，它可以显示在状态条上。
- moreinfo 依赖于数据项的类型，参见表 13-1。
- user_data 如果这个菜单项有回调函数， user_data会被传递到回调函数。
- unused_data 应该设置为NULL，现在还没有用到。Gnome的后续版本也许会用到它。
- pixmap_type 是一个GnomeUIPixmapType枚举类型的值，它的作用是指定它的下一个成员pixmap_info的类型。
- pixmap_info 可以是原始的pixmap数据、一个文件名，或者是一个 Gnome 内置pixmap图

片。

- accelerator_key 是这个菜单项的快捷键。可以用一个字符，比如“ a ”，或者是gdk/gdkkeysyms.h中定义的一个值来表示。
- ac_mods 是一个用于快捷键的组合键屏蔽值。
- widget 应该设置为NULL。当Gnome创建菜单项或工具条按钮时，会将构件填充在其中。如果想要用某种方式操纵该构件，可以检索到它。

菜单项的名字的下划线用于标志菜单项的快捷键。翻译程序会根据需要将下划线移开，让菜单文本在其他国家的语言中是可读的。 Gnome会分析菜单项名称，并取得加速键，然后将下划线删除。

表中概括了GnomeUIInfo结构中type域的可能取值。其中有几个最可能的取值，但是其他几个值由Gnome内部使用。下表对应用程序来说应该足够了。

表13-1 GnomeUIInfoType值

GnomeUIInfoType	描 述	moreinfo域
GNOME_APP_UI_ENDOFINFO	终止一个GnomeUIInfo表	None
GNOME_APP_UI_ITEM	普通菜单或工具条项(或在无线按钮组中的单选项)	回调函数
GNOME_APP_UI_TOGGLEITEM	切换/检查项	回调函数
GNOME_APP_UI_RADIOITEMS	无线按钮项组	组中的无线按钮项数组
GNOME_APP_UI_SUBTREE	子菜单	菜单子树中的GnomeUIInfo数组
GNOME_APP_UI_SEPARATOR	分隔线	None
GNOME_APP_UI_HELP	帮助项	要加载的帮助节点

要创建一个完整的菜单树，可以用 GNOMEUIINFO_SUBTREE()宏生成一个指向上级菜单表的指针（注意，这里的file_menu就是前面创建初始化的GnomeUIInfo结构）：

```
static GnomeUIInfo main_menu[] = {
  GNOMEUIINFO_SUBTREE(N_("_File"), file_menu),
  GNOMEUIINFO_END
};
```

不过，在这个特殊情况中，还有一个更好的宏：

```
static GnomeUIInfo main_menu[] = {
  GNOMEUIINFO_MENU_FILE_TREE(file_menu),
  GNOMEUIINFO_END
};
```

这个宏的主要优点是标准化。它保证所有的 Gnome应用程序的文件菜单有同样的名称和快捷键。还有一些类似的宏，请参见 libgnomeui/gnome-app-helper.h头文件

13.2.2 将GnomeUIInfo转换为构件

一旦有了菜单表， Gnome就会对它进行处理，并将它转换成构件。可以使用下表中的函数来完成转换。

函数列表：由GnomeUIInfo创建构件

```
#include <libgnomeui/gnome-app-helper.h>
/*将uuiinfo转换为菜单并连接到GnomeApp窗口app上*/
```

```
void gnome_app_create_menus(GnomeApp* app,
                      GnomeUIInfo* uiinfo)

/*与上一个函数差不多，但是user_data会覆盖uiinfo中的user_data*/
void gnome_app_create_menus_with_data(GnomeApp* app,
                                GnomeUIInfo* uiinfo,
                                gpointer user_data)

/*将uiinfo转换为工具条，然后添加到GnomeApp窗口app上*/
void gnome_app_create_toolbar(GnomeApp* app,
                        GnomeUIInfo* uiinfo)

/*与上一个函数差不多，但是user_data会覆盖uiinfo中的user_data*/
void gnome_app_create_toolbar_with_data(GnomeApp* app,
                                  GnomeUIInfo* uiinfo,
                                  gpointer user_data)

/*将uiinfo转换为工具条构件。可以将这生成的工具条添加到普通的 GtkWindow上*/
void gnome_app_fill_toolbar(GtkToolbar* toolbar,
                      GnomeUIInfo* uiinfo,
                      GtkAccelGroup* accel_group)

/*与上一个函数类似，但是data会覆盖uiinfo中的userdata*/
void gnome_app_fill_toolbar_with_data(GtkToolbar* toolbar,
                                GnomeUIInfo* uiinfo,
                                GtkAccelGroup* accel_group,
                                gpointer data)

/*用uuinfo创建一个GtkMenuShell，可以将这个菜单项添加到GtkMenu上*/
void gnome_app_fill_menu(GtkMenuShell* menushell,
                   GnomeUIInfo* uiinfo,
                   GtkAccelGroup* accel_group,
                   gboolean uline_accels,
                   gint pos)

/*与上一个函数类似，但是user_data会覆盖uiinfo中的userdata*/
void gnome_app_fill_menu_with_data(GtkMenuShell* menushell,
                             GnomeUIInfo* uiinfo,
                             GtkAccelGroup* accel_group,
                             gboolean uline_accels,
                             gint pos,
                             gpointer user_data)
```

如果用GnomaApp作为主窗口，gnome_app_create_menus() 和gnome_app_create_toolbar()函数用所提供的GnomeUIInfo表创建菜单条和工具条，然后将它们连接到 GnomeApp窗口上。多数情况下，用这些函数就可以了。所有工作，如设置菜单和工具条等，都是由 Gnome自动完成的。上面的每个函数都有一个以 _with_data()结尾的变体，变体函数中的 user_data参数都会覆盖GnomeUIInfo结构中的user_data参数。

如果有更特殊的需要，可以手工填充一个菜单条或者工具条，然后把它添加到容器中。

上面最后四个函数就是用于完成这个工作的。填充函数需要指定一个快捷键组，对 GnomeApp 来说，在构件结构中已经有了一个快捷键组 (accel_group成员)。菜单的填充函数带两个参数 accel_group和uline_accels可以切换是否分析GnomeUIInfo结构标签中的下划线以提取快捷键，uline_accels为TRUE时提取快捷键放到 accel_group中，否则，不提取快捷键。对填充菜单，pos参数指定应该在GtkMenuShell 中的什么位置开始插入菜单项。

当用GnomeUIInfo 表创建菜单条和工具条时，指向单个菜单项或工具条按钮构件的指针存放在每个GnomeUIInfo 结构的widget成员中。可以用这些指针访问单个构件，例如，如果创建了一个检查菜单项，也许想设置检查项的状态。如果想手工创建部分菜单，这个指针也是很有用的，例如，可以创建一个空的菜单子树项，然后手工创建子树的内容。