

Linux 典藏大系

ChinaUnix

全面、系统、深入探讨Linux环境C程序设计的核心技术与思想  
高屋建筑，采用较高难度的复杂案例展示开源软件设计的思想



# Linux 环境

## C程序设计

徐诚 高莹婷 等编著



DVD-ROM

7小时多媒体语音视频讲解

另外赠送36.5小时Linux专题视频、Ubuntu安装文件

- 内容全面：涵盖Linux环境C编程的基础知识、高级技术与实践经验
- 融会贯通：将Linux系统开发技术、C/C++开发技术及软件工程思想结合起来讲解
- 内容深入：深入介绍GNOME桌面环境下的图形界面开发，代码达到可复用水平
- 重点突出：深入解析Linux调用函数功能，不用具备操作系统与Linux内核知识即可掌握
- 实践性强：全书贯穿近200个示例和近100个实例进行讲解，非常实用
- 案例经典：用较高难度的媒体播放器案例贯穿全书，并提供了大量的开发技巧
- 视频讲解：专门录制了7小时多媒体教学视频讲解书中的重点内容

清华大学出版社

## 第 26 章 Glade 设计程序界面

Glade 是 Linux 系统中设计 GTK+ 程序界面的可见即可得工具。开发者可将窗体构件作为画布，通过向画布添加界面构件设计程序界面。这种方式最大的优势在于设计的同时能直观地看到界面构件，并且可以随时调整界面的设计，设计界面如同画图一般。Glade 所设计的界面以 XML 格式保存，因此界面和程序逻辑是完全分离的，使程序界面设计更为轻松。本章将介绍 Glade 的使用方法，以及 C 语言接口函数库。

### 26.1 Glade 简介

Glade 界面设计软件是 GNOME 桌面环境的子项目，用于为 GNOME 桌面环境上运行的程序提供图形用户界面。Glade 使用 GPL 协议发布，虽然是开源软件，但它的设计思想和易用性都领先于大多数商业集成开发环境中的界面设计工具。

在 Glade 的界面中，大部分常用 GTK+ 界面构件被作为图标放在工具栏中。开发者如果需要向界面中添加某一个构件，只需从工具栏上选择即可，如图 26.1 所示。

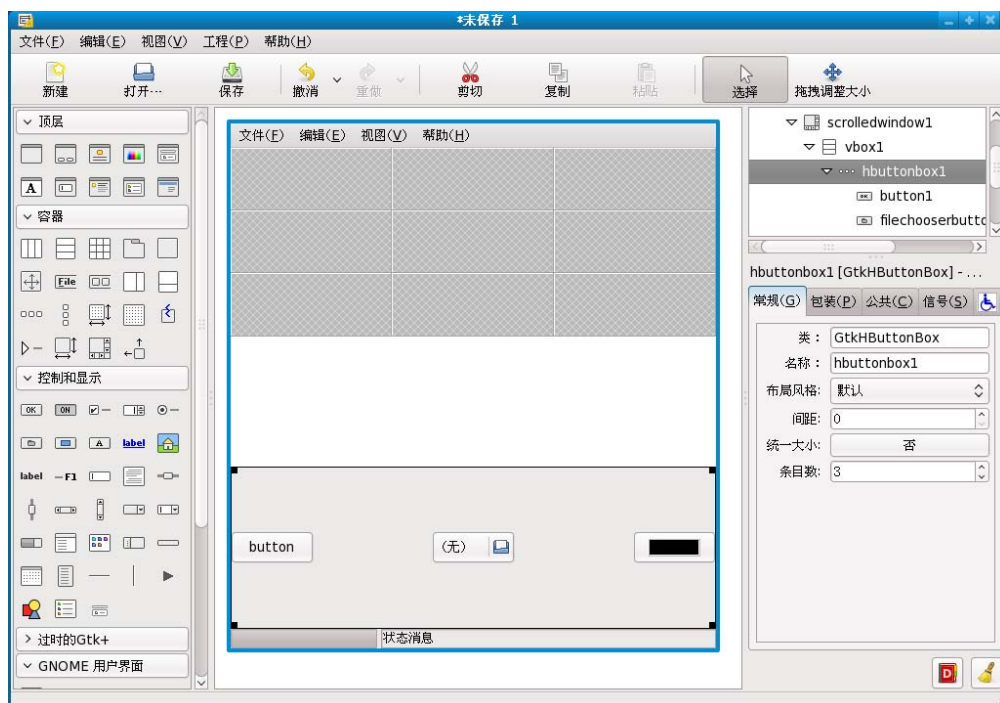


图 26.1 Glade 主界面

添加了界面构件后，可直接在 Glade 中为界面构件设置属性，以及连接回调函数。设计的结果可保存为一个 Glade 界面项目文件，实际该文件是 XML 文件。如下例所示：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE glade-interface SYSTEM "glade-2.0.dtd">
<!--Generated with glade3 3.4.5 on Thu Mar 26 21:13:51 2009 -->
<glade-interface>
  <widget class="GtkWindow" id="window">
    <child>
      <widget class="GtkButton" id="button">
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property name="receives_default">True</property>
        <property name="label" translatable="yes">button</property>
        <property name="response_id">0</property>
        <signal name="clicked" handler="gtk_main_quit"/>
      </widget>
    </child>
  </widget>
</glade-interface>
```

这段代码是用 Glade 生成的，它实现了一个窗体构件和窗体中放置的一个按钮构件。代码第一行定义了 XML 格式版本和字符编码，第二行是实际用途的说明。从第 5 行开始定义窗体构件，而按钮构件是作为窗体构件的子构件定义。其中，还为按钮构件的 `clicked` 信号连接了 `gtk_main_quit()` 函数，实现了按钮构件的功能。

XML 格式的引入是 Glade 最主要的特性，它使程序的界面部分完全独立。在大部分情况下，开发者不用去修改 XML 格式的内容，只需要通过 `libglade` 函数库将程序逻辑部分与界面项目文件连接起来。Glade 的另一特性是能够直接显示容器的层次，而阅读源程序很难理解复杂的容器结构。

安装 Glade 可在其官方网站下载源代码编译，地址为 <http://glade.gnome.org>。或者在终端输入下列命令：

```
yum install glade3 libglade2-devel glade3-libgladeui glade3-libgladeui-devel
```

安装成功后，可选择 GNOME 桌面的“应用程序”|“编程”|“Glade”命令启动 Glade 程序。`libglade` 函数库头文件的路径为“`/usr/include/libglade-2.0/glade`”。

## 26.2 构造图形界面

任何复杂的图形界面都可以使用 Glade 构造，它可以缩短图形界面设计的周期，并在最大程度上保证代码的正确性。在使用 Glade 前，开发者需要对 GTK+ 有初步的认识，本书前一部分的内容已介绍了这些知识。Glade 可成为首选的界面设计软件，替代 C 语言中繁复的编码过程。本节将介绍使用 Glade 构造图形界面的方法。

## 26.2.1 添加窗体

Glade 提供了 10 种窗体构件供用户选择，这些都是在 GTK+ 中所预定义的。开发者可在 Glade 主界面左侧“顶层”选项卡中选择所需要添加的窗体构件，如图 26.2 所示。

选项卡中每一个按钮对应着一种窗体构件，这些按钮的名称依次为：



图 26.2 顶层选项卡

### 1. 通用窗体构件

通用窗体构件即 `gtk_window_new()` 函数所创建的窗体，单击该构件可在 Glade 主界面的编辑区域创建一个新窗体，如图 26.3 所示。

Glade 中所显示的为窗体的主体部分，窗体的标题栏和边框不会显示。其蓝色边框所界定的范围为实际窗体的尺寸，可用鼠标拖动蓝色边框改变窗体的尺寸。窗体主体中间的网格区域表示是未添加界面构件的容器区域，该部分可放置界面构件。

一个 Glade 项目中可以建立多个窗体构件，每个窗体构件都作为一个顶层容器被显示在 Glade 主界面右上方的“容器”列表中，如图 26.4 所示。

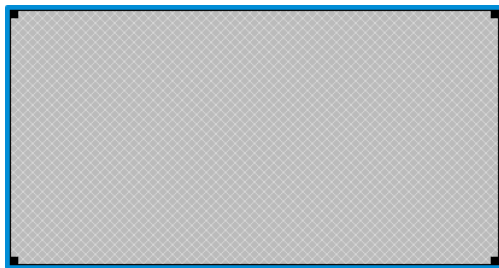


图 26.3 通用窗体构件

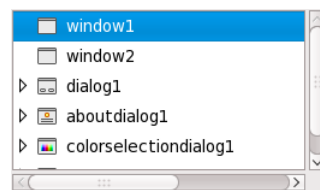


图 26.4 “容器”列表

可在“容器”列表中双击窗体构件的名称打开窗体进行编辑。或者右击窗体名称，在弹出菜单中选择“删除”命令，从项目中删除一个窗体构件。Glade 支持窗体的复制、剪切和粘贴操作，用于在同一个项目内创建窗体的副本，或者将窗体复制到不同项目中。

### 2. 通用对话框构件

通用对话框构件对应 `gtk_dialog_new_with_button()` 函数所创建的窗体，它的内部由一个纵向组装箱容器和一个按钮盒容器组成。通用对话框在程序运行时不显示最小化和最大化按钮，用户也不能通过拖拉操作改变其尺寸，如图 26.5 所示。

通用对话框的纵向组装箱内可放置其他容器或窗体构件。按钮盒预留了两个按钮的位置，该位置只能放置按钮构件或者按钮构件的子类。如果按钮的个数少于或多于按钮盒预留的位置，可在“常规”选项卡

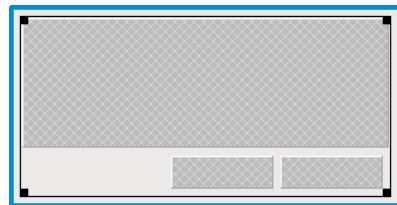


图 26.5 通用对话框构件

修改按钮的个数，如图 26.6 所示。

### 3. 关于对话框

关于对话框是通过 `gtk_about_dialog_new()` 函数建立的，用于显示当前应用程序的信息。关于对话框继承了通用对话框的特性，只是预先定义了一些界面构件在其内，如图 26.7 所示。



图 26.6 通用对话框构件



图 26.7 关于对话框构件

关于对话框中显示的内容可直接在“常规”选项卡中设置。这些内容对应所有应用程序的特性，并遵循通用版式，如下所示：

(1) 名称。对话框构件在程序中的名称，对应 `gtk_about_dialog_set_name()` 函数的功能。该函数的一般形式为：

```
void gtk_about_dialog_set_name(GtkAboutDialog *about,
                               const gchar *name);
```

(2) 程序名称。当前项目所建立应用程序的名称，程序名称用大号显示在关于对话框中心区域。对应 `gtk_about_dialog_set_program_name()` 函数的功能，该函数的一般形式为：

```
void gtk_about_dialog_set_program_name(GtkAboutDialog *about,
                                       const gchar *name);
```

(3) 程序版本。当前项目的版本号，显示在程序名称之后，使用与程序名称相同的字号。对应 `gtk_about_dialog_set_version()` 函数的功能，它的一般形式为：

```
void gtk_about_dialog_set_version(GtkAboutDialog *about,
                                  const gchar *version);
```

(4) 版权字符串。当前项目的版权信息，显示在程序名称下方，使用较小的字号。对应 `gtk_about_dialog_set_copyright()` 函数的功能，它的一般形式为：

```
void gtk_about_dialog_set_copyright(GtkAboutDialog *about,
                                    const gchar *copyright);
```

(5) 评论字符串。评论字符串是当前应用程序主要功能的表述，显示在程序名称和版权字符串之间。对应 `gtk_about_dialog_set_comments()` 函数的功能，它的一般形式为：

```
void gtk_about_dialog_set_comments(GtkAboutDialog *about,
                                   const gchar *comments);
```

(6) 网站 URL。当前项目发行者的网站地址，显示在版权信息下方，字符串有下划线。单击该地址将在浏览器中打开所指向的网页。对应 `gtk_about_dialog_set_website()` 函数，它的一般形式为：

```
void gtk_about_dialog_set_website(GtkAboutDialog *about,
                                   const gchar *website);
```

(7) 网站标签。如果设置了网站标签，那么网站地址不会直接显示在关于对话框上，而是用网站标签内的字符串代替。对应 `gtk_about_dialog_set_website_label()` 函数，它的一般形式为：

```
void gtk_about_dialog_set_website_label(GtkAboutDialog *about,
                                         const gchar *website_label);
```

(8) 许可。设置许可信息后，关于对话框的左下角将出现一个许可按钮，按下该按钮会在一个新对话框中列出许可信息的内容。许可信息的内容通常为 GPL 协议相关信息，如图 26.8 所示。

许可信息可通过 `gtk_about_dialog_set_license()` 函数设置，它的一般形式为：

```
void gtk_about_dialog_set_license(GtkAboutDialog *about,
                                   const gchar *license);
```

(9) 作者。当前项目的程序开发者名称，可输入多个作者的信息。设置作者信息后，界面左下角将增加一个致谢按钮。按下该按钮会弹出“致谢”对话框，并在“作者”选项卡中列出作者信息，如图 26.9 所示。

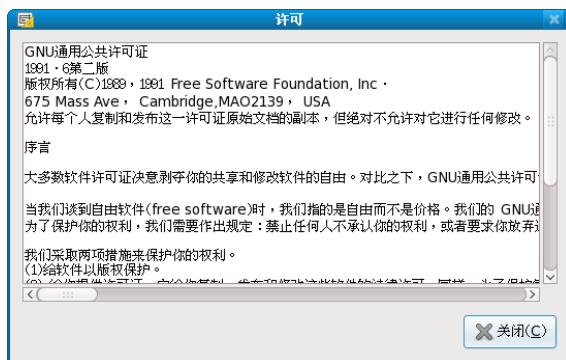


图 26.8 显示许可信息

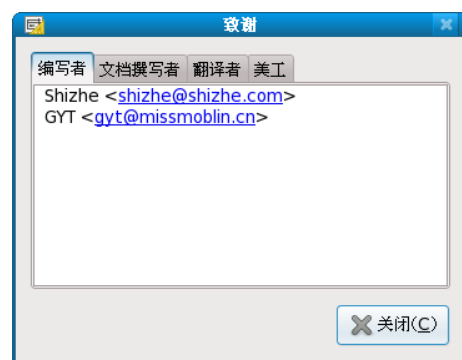


图 26.9 显示许可信息

如果要在作者名称后插入电子邮件地址或网络地址，并且使它们成为超级连接，可通过尖括号“<>”包围地址信息实现。作者信息可通过 `gtk_about_dialog_set_authors()` 函数设置，它的一般形式为：



```
void gtk_about_dialog_set_authors(GtkAboutDialog *about,
                                const gchar **authors);
```

(10) 文档撰写者。当前项目的说明书等文档撰写者的名称，该信息显示在“致谢”对话框中。对应 `gtk_about_dialog_set_documenters()` 函数的功能，它的一般形式为：

```
void gtk_about_dialog_set_documenters(GtkAboutDialog *about,
                                      const gchar **documenters);
```

(11) 翻译者。当前项目的翻译工作者名称，该信息显示在“致谢”对话框中。对应 `gtk_about_dialog_set_translator_credits()` 函数的功能，它的一般形式为：

```
void gtk_about_dialog_set_translator_credits(GtkAboutDialog *about,
                                             const gchar *translator_credits);
```

(12) 美工。当前项目的美工名称，该信息显示在“致谢”对话框中。对应 `gtk_about_dialog_set_artists()` 函数的功能，它的一般形式为：

```
void gtk_about_dialog_set_artists(GtkAboutDialog *about,
                                  const gchar **artists);
```

(13) 标志。用于设置当前项目的标志，可以是 GTK+ 支持的任何图形格式文件，显示在标题栏下方，如图 26.10 所示。设置标志文件可通过 `gtk_about_dialog_set_logo()` 函数实现，它的一般形式为：

```
void gtk_about_dialog_set_logo(GtkAboutDialog *about,
                               GdkPixbuf *logo);
```

#### 4. 颜色选择对话框

颜色选择对话框对应 GTK+ 库中的 `gtk_color_selection_dialog_new()` 函数所建立的对话框，用于选择颜色。窗体中的大部分内容是固定的，不可被用户修改，用户只能在其中的纵向组装盒容器中添加界面构件，如图 26.11 所示。



图 26.10 项目标志



图 26.11 颜色选择对话框

## 5. 文件选择对话框

文件选择对话框可通过 `gtk_file_chooser_dialog_new()` 函数创建，它有一个纵向组装盒可用于放置界面构件，另外还提供了按钮盒放置按钮。如果没有指定按钮，那么 Glade3 会为其自动从按钮库添加 `GTK_STOCK_CANCEL` 和 `GTK_STOCK_OPEN`。

文件选择对话框有一个重要属性，即“动作”属性。可在“常规”选项卡中设置，它有 4 个选项，默认为“打开”，其他选项依次为“保存”、“选择目录”和“创建目录”。这 4 个选项用于设置对话框的功能特性，与此同时对话框的标题和外观也会跟随设置改变，如图 26.12 所示。

## 6. 字体选择对话框

字体选择对话框对应 `gtk_font_selection_dialog_new()` 函数的功能，它的大部分组件不能被修改，只提供了一个纵向组装盒用于添加界面构件，如图 26.13 所示。

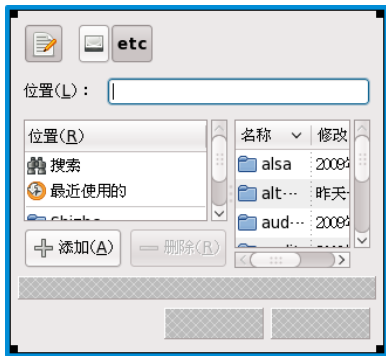


图 26.12 文件选择对话框（动作为打开）



图 26.13 字体选择对话框

## 7. 输入对话框

输入对话框对应 `gtk_input_dialog_new()` 函数的功能，它用于为鼠标、游戏操纵杆、画板等平面定位输入设备进行设置，在很多程序中是非常重要的。输入对话框的大部分功能都是 GTK+ 内部实现的，所以并不需要对其进行额外的设置，如图 26.14 所示。

## 8. 消息对话框

消息对话框对应 `gtk_message_dialog_new()` 函数的功能，所有内容均可在“常规”选项卡中设置。如下所示。

- ❑ 消息类型：用于定义消息对话框显示的风格，选项依次为“信息”、“警告”、“问题”、“错误”和“其他”。
- ❑ 消息按钮：用于定义消息对话框中所显示的按钮，选项依次为“无”、“确定”、“关闭”、“取消”、“是，否”和“确定，取消”。
- ❑ 文字：用大字体显示的消息文本。



□ 次要文本：用小字体显示的消息文本，如图 26.15 所示。



图 26.14 输入对话框



图 26.15 消息对话框

## 9. 最近选择对话框

最近选择对话框对应 `gtk_recent_chooser_dialog_new()` 函数的功能，用于显示最近用户编辑过的文件。“常规”选项卡的“限制”微调框可设置文件显示的最多个数。“排序类型”下拉列表框可设置文件列表的排序方法，依次为“无”、“最近使用最多的一个”、“最近使用最少的一个”和“定制”。对话框中有一个按钮盒构件，可装入要显示的按钮，如图 26.16 所示。

## 10. 辅助

辅助是一种分为多页显示内容的向导窗体，在 GTK+ 库中可使用 `gtk_assistant_new()` 函数创建。每一页中都默认放置着一个文本标签构件，用于显示文本信息。如果需要放置其他构件，可将文本标签删除。窗体的右下方有两个按钮，分别用于向前翻页和向后翻页，如图 26.17 所示。

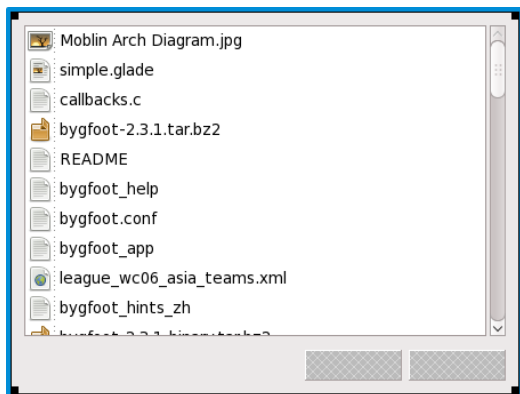


图 26.16 最近选择对话框



图 26.17 辅助窗口

如果当前页面是第一面，“后退”按钮将被隐藏。如果是最后一页，“前进”按钮会

被“应用”按钮替代。

## 26.2.2 添加容器

Glade 提供了 19 种容器构件供用户选择，这些都是在 GTK+ 中所预定义的。开发者可在 Glade 主界面左侧“容器”选项卡中选择所需要添加的容器构件，如图 26.18 所示。

选项卡中每一个按钮对应着一种容器构件。根据使用方法和作用的不同，可将这些容器依次分为下列类别。

### 1. 横向组装盒与纵向组装盒

单击横向组装盒与纵向组装盒按钮时，Glade 会提示输入条目数，该数值是容器中单元格的个数。在使用 `gtk_hbox_new()` 和 `gtk_vbox_new()` 时并不需要提供这些参数，设置单元格的个数是为了便于可视化编辑。设置后可在“常规”选项卡中修改单元格的个数，如图 26.19 所示。

在容器中可继续装入其他的容器，容器的层次并没有限制。Glade 对容器的管理非常灵活，其主界面右上方的“容器”列表内将根据容器名称显示出容器的层次，如图 26.20 所示。

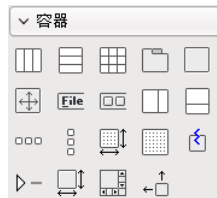


图 26.18 容器选项卡

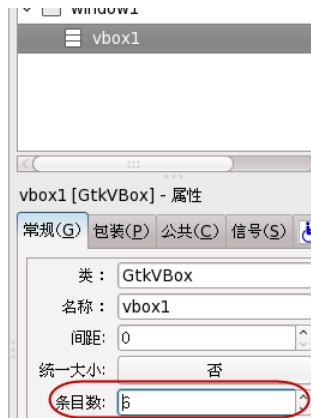


图 26.19 修改单元格的个数

如果需要在容器的上一级增加一个容器，可右击编辑区内的容器，或者右击“容器”列表中的容器名。在弹出菜单“添加上一级”子菜单中，选择要添加的容器，如图 26.21 所示。

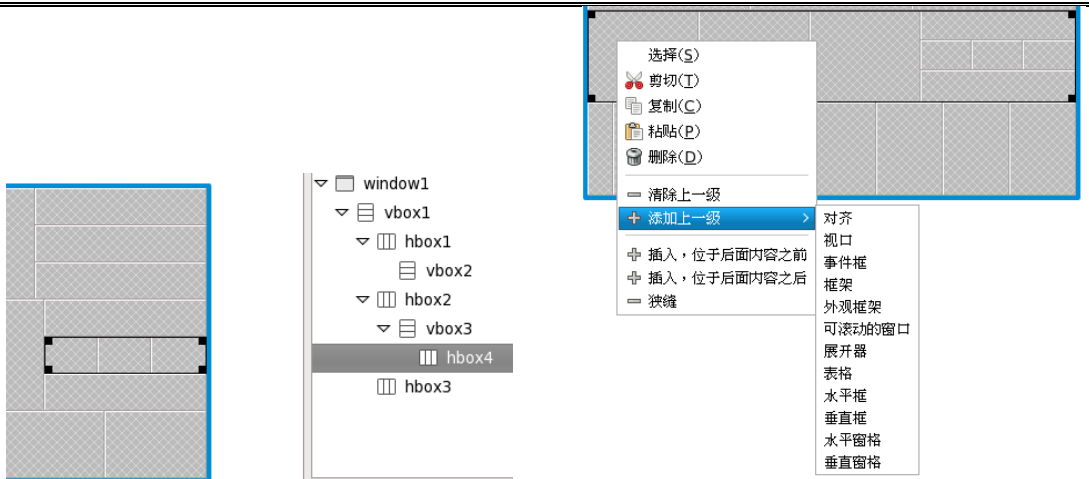


图 26.20 容器的层次

图 26.21 添加上一级容器

删除容器则有两种方式。第一种是右击编辑区中的容器或“容器”列表中的容器名，在弹出菜单中选择“删除”命令。这将删除容器本身，以及容器内的所有界面构件。另一种方法是在弹出菜单中选择“清除上一级”命令，只有容器的上一级容器被删除，容器本身的层次向前移了一位。

复制、剪切和粘贴也可用于容器，影响的将是容器内的所有界面构件。Glade 会为这些构件的副本重新命名。

## 2. 表格

表格按钮对应 `gtk_table_new()` 函数的功能，按下时将提示输入表格的行数和列数。或者在创建表格后，通过“常规”选项卡中的“行数”和“列数”输入框修改，如图 26.22 所示。



图 26.22 创建表格

### 3. 笔记本

笔记本按钮对应 `gtk_notebook_new()` 函数，按下时将提示输入笔记本的页数。该页数可在创建笔记本后通过“常规”选项卡中“页”微调框中修改。笔记本构件中选项卡的名称作为文本标签构件列在“容器”列表内，可单击该名称，在“常规”选项卡的“标签”文本框中修改，如图 26.23 所示。

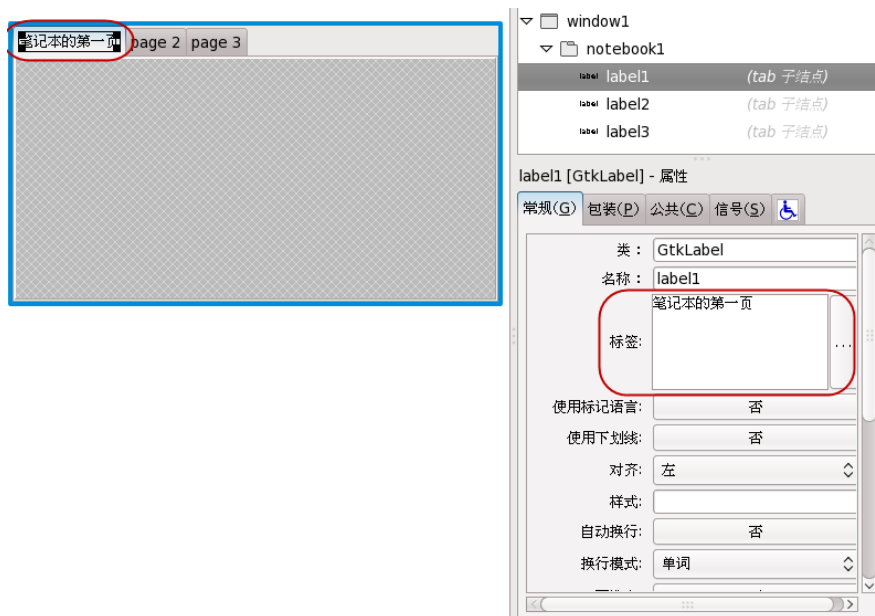


图 26.23 修改选项卡名称

### 4. 框架和外观框架

创建框架构件所对应的是 `gtk_frame_new()` 函数，使用 Glade 创建框架构件时会自动添加一个对齐构件和一个标签构件。对齐构件是框架内的下一层容器，标签构件显示在框架的右上方，如图 26.24 所示。

框架的边框风格可在“常规”选项卡内的“框架阴影”下拉列表框中设置，选项依次为“无”、“里面”、“突出”、“向内蚀刻”和“向外蚀刻”。

外观框架又称比例框架构件，所对应的是 `gtk_aspect_frame_new()` 函数。外观框架的比例属性可在“常规”选项卡内的“比率”微调框内设置，如图 26.25 所示。

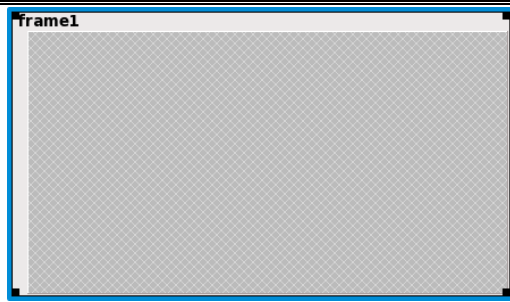


图 26.24 框架

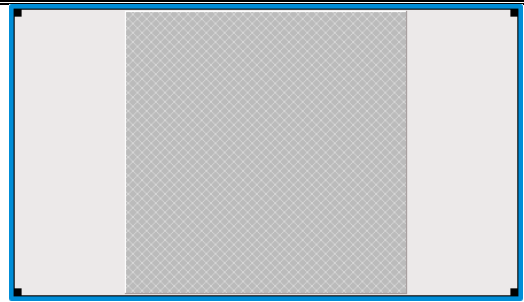


图 26.25 外观框架

## 5. 菜单条

Glade 添加菜单条的功能远比 `gtk_menu_bar_new()` 函数所实现的功能要丰富，它能同时添加菜单容器和菜单项。Glade 没有将菜单容器和菜单项作为独立的界面构件，而是提供了菜单编辑器专门用于设计菜单。右击编辑区中的菜单，在弹出菜单中选择“编辑”命令，将打开菜单编辑器，如图 26.26 所示。

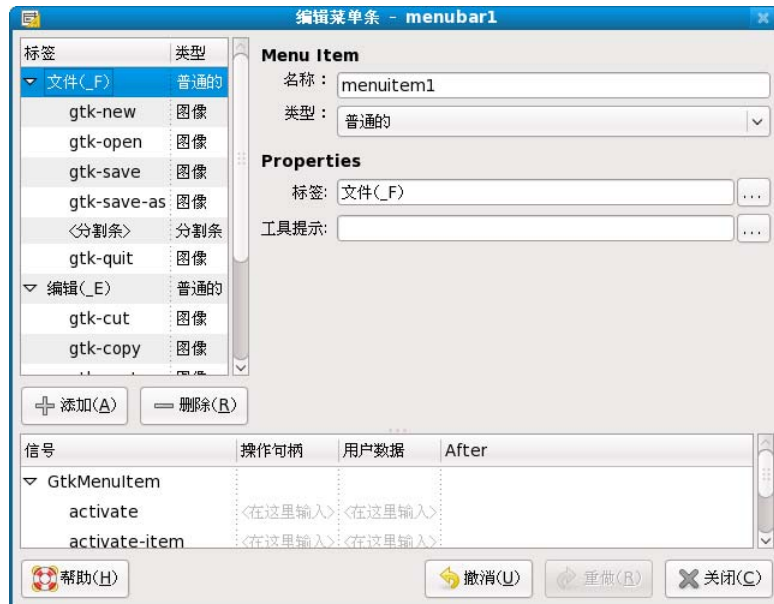


图 26.26 菜单编辑器

在菜单编辑器左侧的标签列表中选择菜单项名称后，可编辑该菜单项。菜单编辑器右侧有 4 个属性可以设置，依次如下。

- ☐ 名称：在代码中访问该菜单项的名称。
- ☐ 类型：根据 GTK+ 对菜单项的定义，可选取的值有“普通的”、“图像”、“复选”、“单选”和“分割条”。
- ☐ 标签：显示在菜单中的字符串。

- ❑ 工具提示：鼠标悬停时显示的文本，菜单编辑器会为菜单项自动添加工具提示对象。
- ❑ 库存条目：该选项在菜单项“类型”设置为“图像”时显示，可从图像库中选择菜单项的图形。

如果要添加一个菜单项，可单击“添加”按钮，新菜单项将在菜单项列表中所选菜单项后一位，且处于同一层。或者右击列表中的菜单项，选择“添加子项目”命令，创建所选菜单项的下一级菜单。

菜单编辑器的下方是信号与事件的列表，可直接在此为菜单项连接事件与回调函数。如果要为菜单项添加快捷方式，操作步骤为：

(1) 在“容器”列表内选择菜单项。

(2) 选择“容器”列表下的公共选项卡，单击“加速键”后的编辑按钮，如图 26.27 所示。

(3) 在“选择加速键”对话框中，选择对应的信号、按键和控制键，如图 26.28 所示。



图 26.27 加速键

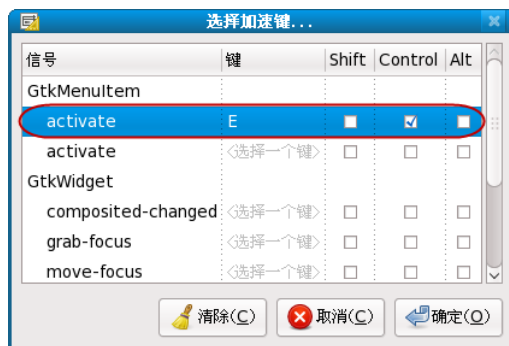


图 26.28 选择加速键

## 6. 工具条

工具条对应 `gtk_toolbar_new()` 函数的功能，创建后在编辑区右击工具条，在弹出的快捷菜单中选择“编辑”命令，可打开“工具条编辑器”对话框，如图 26.29 所示。

在“工具条编辑器”中，可单击“添加”按钮添加一个工具构件。“类型”下拉列表框用于定义工具构件的类型，默认为“按钮”。工具构件的信号与事件可在对话框下侧的信号列表中设置。



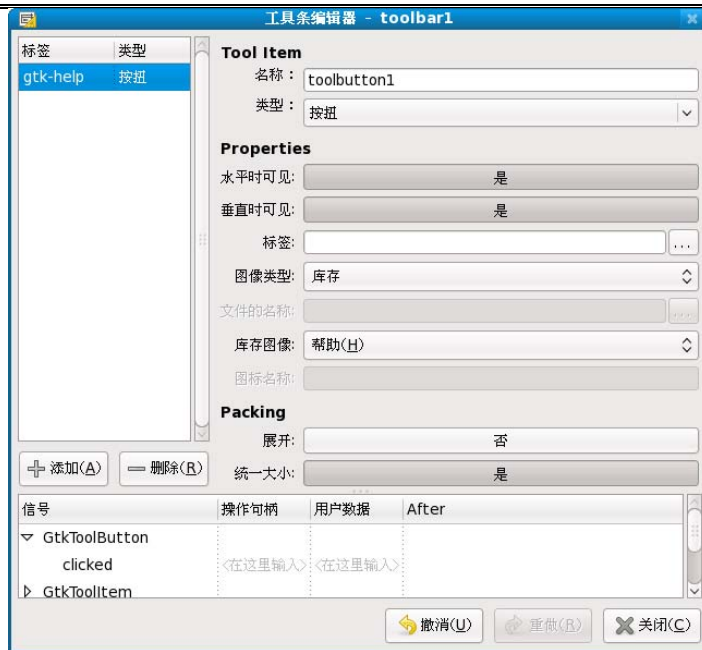


图 26.29 工具条编辑器

## 7. 水平窗格和垂直窗格

水平窗格和垂直窗格对应 `gtk_hpaned_new()`和 `gtk_vpaned_new()`函数的功能，初始位置可在“常规”选项卡内的“位置”微调框中设置，并且要将“位置设置”的值设为“是”才能在程序中生效，如图 26.30 所示。

## 8. 横向按钮盒与纵向按钮盒

横向按钮盒与纵向按钮盒对应 `gtk_hbutton_box_new()`和 `gtk_vbutton_box_new()`函数的功能。为了方便编辑，需要在“常规”选项卡内的“条目数”微调框中指定按钮盒内单元格的个数，默认值为 3，如图 26.31 所示。

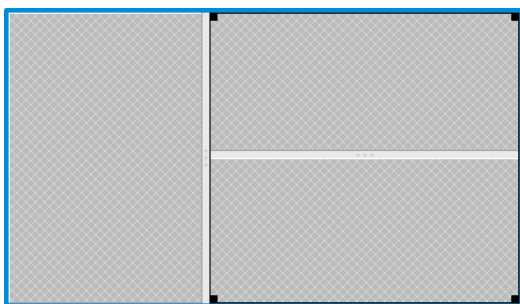


图 26.30 水平窗格和垂直窗格

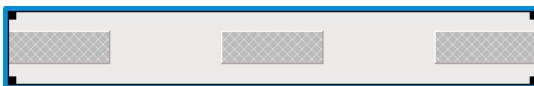


图 26.31 按钮盒

## 9. 陈列

陈列即是指布局容器，对应 `gtk_layout_new()` 函数的功能。布局容器最大尺寸可在“常规”选项卡内的“宽度”和“高度”微调框中设置。

## 10. 固定

固定容器对应 `gtk_fixed_new()` 函数的功能。

## 11. 事件框

事件框对应 `gtk_event_box_new()` 函数的功能。

## 12. 展开器

展开器对应 `gtk_expander_new()` 函数的功能，它由一个箭头构件、一个标签和一个容器所组成。单击箭头可改变箭头的方向。当箭头构件指向下时，展开器内的容器构件将显示。而在箭头指向右方时，展开器内的容器将被隐藏，如图 26.32 所示。

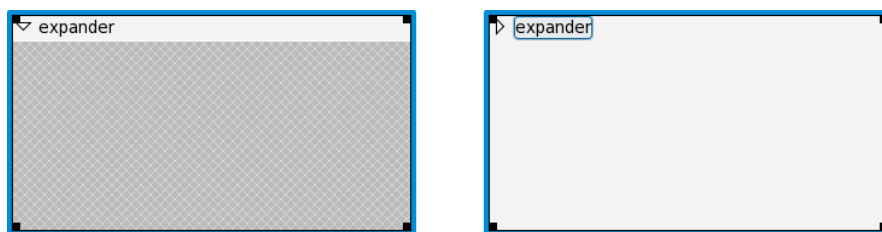


图 26.32 展开器的展开与收缩状态

## 13. 视口

视口即视见区，对应 `gtk_viewport_new()` 函数的功能。“常规”选项卡内的“阴影类型”下拉列表框可设置其边框的类型，选项依次为“无”、“里面”、“突出”、“向内蚀刻”和“向外蚀刻”。

## 14. 可滚动的窗口

可滚动的窗口即滚动条窗体构件，对应 `gtk_scrolled_window_new()` 函数的功能。它包括一组滚动条构件和一个视见区，但在 Glade 中不可直接访问其子构件的属性。如果要设置滚动条构件的显示状态，可通过“常规”选项卡内的“水平滚动条策略”和“垂直滚动条策略”下拉列表框设置，如图 26.33 所示。

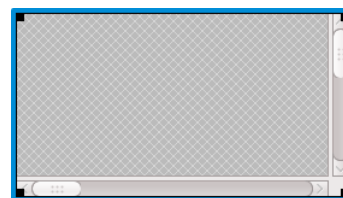


图 26.33 可滚动的窗口

## 15. 对齐

对齐容器对应 `gtk_alignment_new()` 函数。在“常规”

选项卡中可设置其属性，这些属性依次为：

- ☐ 水平排列：取值范围为 0.0 至 1.0，即最左到最右。
- ☐ 垂直排列：取值范围为 0.0 至 1.0，即最上到最下。
- ☐ 水平缩放比率：如果水平方向可用的空间比子构件所需要的多，设置子部件将使用多少。0.0 表示不用，1.0 表示全部。
- ☐ 垂直缩放比率：如果垂直方向可用的空间比子构件所需要的多，设置子部件将使用多少。0.0 表示不用，1.0 表示全部。
- ☐ 顶部留空：上方的边界值。
- ☐ 底部留空：下方的边界值。
- ☐ 左部留空：左面的边界值。
- ☐ 右部留空：右面的边界值。

### 26.2.3 添加构件

Glade 提供了两组界面构件，分别位于“控制和显示”选项卡与“过时的 Gtk+”选项卡中，如图 26.34 所示。

后者是 GTK+ 为了保持与旧版本兼容所以仍然在使用的界面构件。这些界面构件均已被其他构件所替代，并且不再被更新，甚至可能会被将来的版本抛弃，应谨慎选择这些构件。常用的界面构件可分为如下几类。

#### 1. 按钮

按钮构件共有 9 种。单击代表构件的按钮后，将鼠标指针移动到编辑区的容器上方，可见光标变为一个加号外加构件图标形状。再次按下鼠标左键，构件将被添加到容器以内。这些按钮依次为。

- ☐ 普通按钮对应 `gtk_button_new()` 函数的功能。
- ☐ 开关按钮对应 `gtk_toggle_button_new()` 函数的功能。
- ☐ 复选按钮对应 `gtk_check_button_new()` 函数的功能。
- ☐ 微调按钮对应 `gtk_spin_button_new()` 函数的功能。
- ☐ 单选按钮对应 `gtk_radio_button_new()` 函数的功能，Glade 可以自动为单选按钮添加 GSList 链表。如果要使多个单选按钮使用同一个链表，即划为同一组，可单击“常规”选项卡“组”后的编辑按钮，弹出“在工程中选择 单选按钮”对话框。然后选择该组中第一个单选按钮的名称，如图 26.35 所示。
- ☐ 文件选择按钮对应 `gtk_file_chooser_button_new()` 函数的功能。
- ☐ 颜色按钮对应 `gtk_color_button_new()` 函数的功能。
- ☐ 字体按钮对应 `gtk_font_button_new()` 函数的功能。
- ☐ 连接按钮对应 `gtk_link_button_new()` 函数的功能，连接的网络地址可在“常规”选

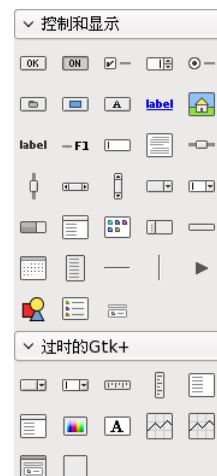


图 26.34 构件选项卡

项卡内的 URL 文本框中输入。

## 2. 图像

图像对应 `gtk_image_new_from_stock()` 函数的功能，可在“常规”选项卡内“库存图像”下拉列表框中设置图像，默认情况下使用的是图像库内 `GTK_MISSING_IMAGE`。图像的尺寸可在“图标大小”微调框内设置，取值对应 `GtkIconSize` 枚举类型，有效取值范围为 0 至 6。如果要在图像构件中使用文件，可将“编辑类型”设为文件名，然后在“文件的名称”中进行设置。

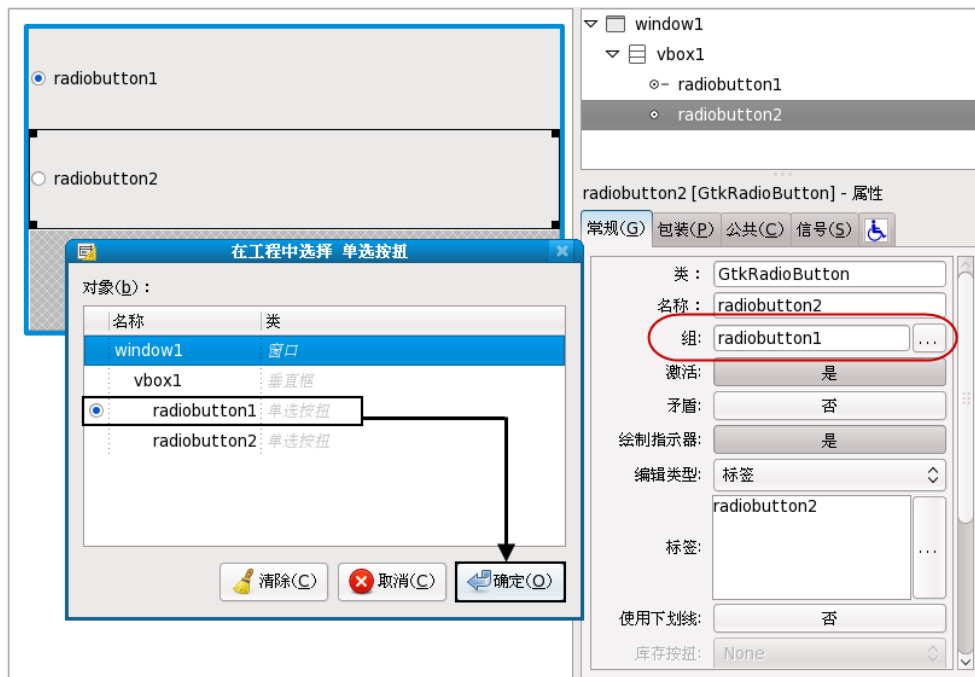


图 26.35 为单选按钮分组

## 3. 标签和加速键列表

标签对应 `gtk_label_new()` 函数的功能。“常规”选项卡内“标签”文本框用于编辑显示的文字，“对齐”下拉列表框用于定义对齐方式。

加速键列表即快捷标签，对应 `gtk_accel_label_new()` 函数的功能。快捷键在“公共”选项卡，“加速键”文本框中设置。

## 4. 文本条目和文本视图

文本条目即文本框，对应 `gtk_entry_new()` 函数的功能。文本视图对应 `gtk_text_view_new()` 函数的功能。“常规”选项卡中，“可编辑”用于决定是否锁定文本框，“可见状态”用于设置是否显示文本框中的文本，“文字”文本框中可设置初始文本。

## 5. 范围构件

范围构件共有 4 种分别是水平比例、垂直比例、水平滚动条和垂直滚动条。“常规”选项卡“调整部件”中可设置范围构件的属性。

## 6. 组合框与组合框条目

组合框对应 `gtk_combo_box_new()` 函数的功能，组合框条目对应 `gtk_combo_box_entry_new()` 函数的功能。后者比前者多出一个文本框子构件。单击“常规”选项卡内“条目”文本框后的编辑按钮，可弹出“编辑文本”对话框。其中可编辑需要显示的条目，多个条目用回车键分隔，如图 26.36 所示。

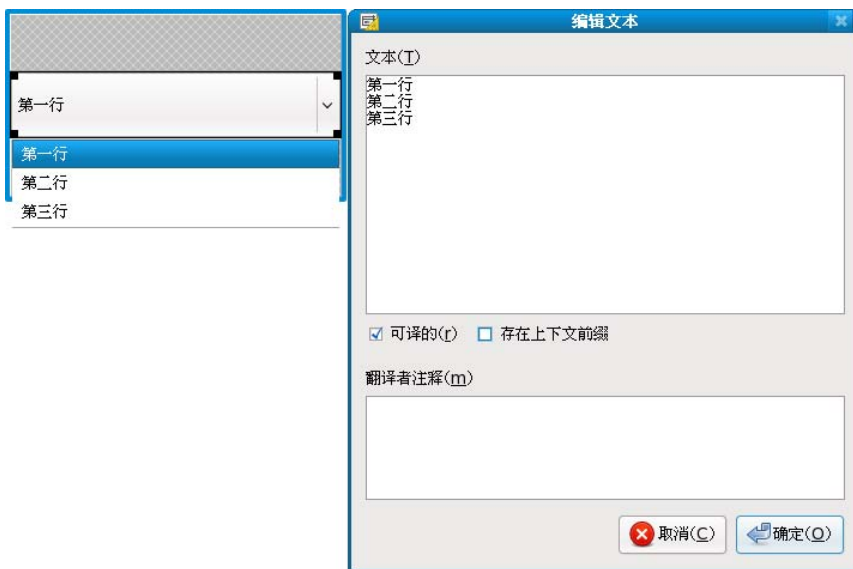


图 26.36 编辑文本对话框

## 7. 进度条

进度条对应 `gtk_progress_bar_new()` 函数的功能。进度条已完成的进度比例可在“常规”选项卡“完成比例”微调框中设置。

## 8. 树视图和图标视图

树视图对应 `gtk_tree_view_new()` 函数的功能，图标视图对应 `gtk_icon_view_new()` 函数的功能。

## 9. 可移动的框

可移动的框对应 `gtk_handle_box_new()` 函数的功能。

## 10. 状态栏

状态栏对应 `gtk_statusbar_new()` 函数的功能。

## 11. 日历

日历构件对应 `gtk_calendar_new()` 函数的功能，可在“常规”选项卡“年”、“月”、“日”微调框中设置默认选中的。其中“月份”的取值范围为 0 至 11，如果“日”的值设为 0 则不指定具体天数。

## 12. 弹出式菜单

弹出式菜单并不会直接在编辑区中显示，添加后会列出在“容器”列表中。可使用菜单编辑器进行编辑。

## 13. 水平分割条和垂直分割条

水平分割条对应 `gtk_hseparator_new()` 函数的功能，垂直分割条对应 `gtk_vseparator_new()` 函数的功能。

## 14. 箭头

箭头对应 `gtk_arrow_new()` 函数的功能。箭头的方向可在“常规”选项卡内“箭头方向”下拉列表框中设置。

## 15. 绘图区域

绘图区域对应 `gtk_drawing_area_new()` 函数的功能。

## 16. 最近选择器

最近选择器对应 `gtk_recent_chooser_widget_new()` 函数的功能。其设置方法与最近选择对话框类似。

## 17. 文件选择部件

文件选择部件对应 `gtk_file_chooser_widget_new()` 函数的功能。其设置方法与文件选择对话框类似。

### 26.2.4 设置构件属性

在 Glade 中，界面构件的属性被分为 3 类，分别位于“常规”、“包装”、“公共”选项卡中。“常规”选项卡内主要是构件基本信息和特有的属性。基本信息包括下列内容。

- ☐ 类：构件对应 GTK+ 库的类名，该值不可修改。
- ☐ 名称：在程序中访问构件的名称，添加构件时 Glade 会为其自动指定一个。



“包装”选项卡用于设置构件在容器中的位置，对于窗体和顶级容器不可用。其中属性的设置如下。

- ☐ 位置：如果上一级容器内有多个单元格，那么第一个单元格的位置为 0，依次类推。
- ☐ 留空：用于设置构件与上一级容器的上下间距。
- ☐ 展开：用于设置是否展开界面构件。
- ☐ 填充：用于设置是否让界面构件沾满整个容器。
- ☐ 包裹类型：可设置为“开始”或“结束”，用于定义界面装入容器时的顺序。

“公共”选项卡用于设置构件的公共属性，这些属性均为 `GtkWidget` 类中定义的，因此可用于所有界面构件。公共属性的设置如下。

- ☐ 宽度请求：设置构件最小需求尺寸中宽度的数值。
- ☐ 高度请求：设置构件最小需求尺寸中高度的数值。
- ☐ 可见：设置构件是否在界面中显示出来。
- ☐ 敏感：设置构件是否接受用户的输入。
- ☐ 工具提示：鼠标光标在构件上方悬停时所显示的文本，`Glade` 会自动创建工具提示对象。
- ☐ 不全部显示：用于屏蔽 `gtk_widget_show_all()` 函数对该构件的影响。
- ☐ 可绘图：设置应用程序是否可以直接在此构件上绘图。
- ☐ 接受焦点：设置构件是否可以接受输入焦点。对于按钮类构件，默认为“是”；对于容器类构件，默认为“否”。
- ☐ 有焦点：设置构件是否已经拥有输入焦点，对于“接受焦点”设置为“是”的构件有效。如果多个构件设置为“是”，只有第一个有效。
- ☐ 为焦点：设置构件是否是顶级容器内的聚焦部件。如果设置为“是”，当构件上一级容器获得焦点时，那么焦点会落在该构件上。对于“接受焦点”设置为“是”的构件有效。如果多个构件设置为“是”，只有第一个有效。
- ☐ 可成为默认：设置构件是否可以成为默认的构件，用于接受 `Enter` 键的响应。
- ☐ 接受默认动作：设置构件在成为焦点时是否可以接受默认动作，即对于空格键的响应。对于“接受焦点”设置为“是”的构件有效。如果多个构件设置为“是”，只有第一个有效。
- ☐ 事件：用于决定界面构件可接受哪些 `GtkEvent` 事件类型的响应。单击其右侧编辑按钮，将弹出“选择区域”对话框，可在其中“选择独立区域”列表中选择需要响应的事件，如图 26.37 所示。
- ☐ 扩展事件：用于决定构件可接受哪些扩展事件。
- ☐ 有工具提示：用于决定是否显示工具提示对象中的文本。
- ☐ 工具提示标记：工具提示对象显示的文本，在“有工具提示”设置为“是”时显示。

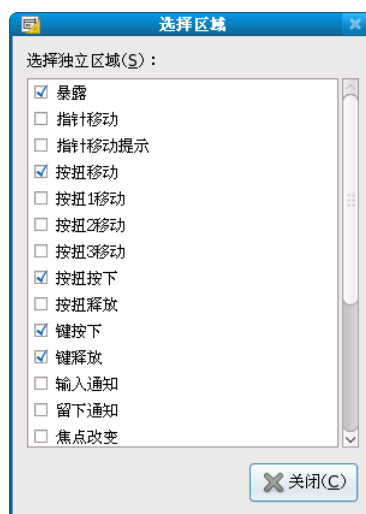


图 26.37 选择区域对话框

- ❑ 工具提示文本：如果设置了“工具提示文本”，那么“工具提示标记”将无效。
- ❑ 加速键：用于设置构件的快捷方式，单击右侧编辑按钮将弹出“选择加速键”对话框，可在其中编辑多组快捷方式。

### 26.2.5 添加事件和回调

Glade 主界面的“信号”选项卡中可为界面构件连接事件、信号和回调函数。所选构件可用事件将以该构件对应的类的继承关系显示信号，如图 26.38 所示。

上图是文本输入框所对应的信号。最底层为 GObject 类定义的信号，最顶层则是文本输入框所属的 GtkEntry 类定义的信号。单击类名称左侧的展开器，将显示出该类定义的所有信号，如图 26.39 所示。

⚠注意：GtkWidget 类中定义与 GDK 底层事件相关的信号必须选择“公共”选项卡中的“事件”列表框才能生效。



图 26.38 信号的分类

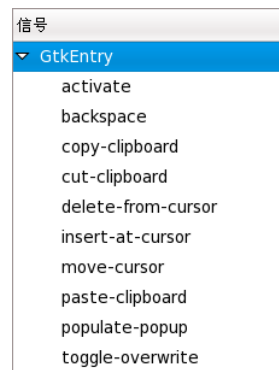


图 26.39 展开分类中的信号

选择信号名称后，可为该信号连接回调函数和数据，对应 `g_signal_connect()` 函数的功能。回调函数可单击对应单元格中的下拉列表选择，如图 26.40 所示。

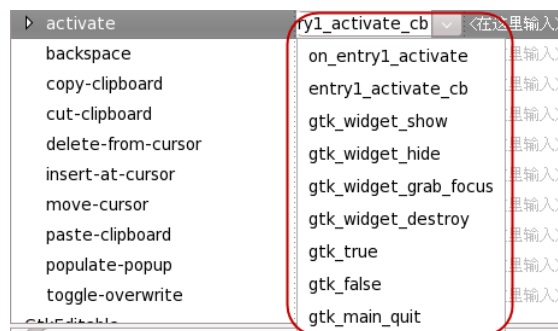


图 26.40 选择回调函数

回调函数列表中的前 2 条函数是 Glade 根据构件名称命名的，其余为可用的 GTK+ 函

数。如果需要自定义回调函数名称，可在单元格内直接输入。

回调函数后可设置传递给回调函数的用户数据，该数据通常是回调函数中最后一个实际参数的名称，可以为变量名或常量，如图 26.41 所示。

信号	操作句柄	用户数据	After
▼ GtkButton			
activate	<在这里输入>	<在这里输入>	
▶ clicked	gtk_widget_show	window2	<input checked="" type="checkbox"/>

图 26.41 设置回调函数数据

图 26.41 中，为一个按钮构件的 clicked 信号连接了 gtk\_widget\_show() 函数，用户数据设置为 window2。在实际程序中，按下该按钮即能显示项目中名为 window2 的构件。

如果回调函数并非 GTK+ 中提供的函数，那么回调函数的实现必须在具体 C 语言代码中进行，两者使用的名称必须一致。

信号列表中有一项 After 单选框，选择后将使用 g\_signal\_connect\_after() 函数连接信号与回调函数。当为信号设置回调函数后，信号名的左侧会多出一个展开器。如果需要为同一个信号连接更多的回调函数，可单击该展开器添加更多回调函数，如图 26.42 所示。

信号	操作句柄	用户数据	After
▼ GtkButton			
activate	<在这里输入>	<在这里输入>	
▼ clicked	gtk_widget_show	window2	<input checked="" type="checkbox"/>
	gtk_widget_destroy	window1	<input type="checkbox"/>
	on_button1_clicked	"已关闭"	<input checked="" type="checkbox"/>
	<在这里输入>	<在这里输入>	

图 26.42 添加更多回调函数

## 26.3 C 语言代码联编

Glade 的项目文件是一个单独的“.glade”文件，可通过 libglade 库将该项目文件添加到 C 语言源代码中。这样，就能在 C 语言编程写的程序中直接使用 Glade 设计的用户界面。本节将介绍 libglade 库的使用方法。

### 26.3.1 libglade 函数库编程基础

通过 libglade 函数库连接 Glade 项目文件至少包含两个步骤，这些工作必须在 GTK+ 库初始化后，且没有进入 GTK+ 主循环时完成。步骤如下：

(1) 创建 GladeXML 对象。GladeXML 对象是用于动态加载 XML 格式用户界面的类型，可使用 glade\_xml\_new() 函数创建。它的一般形式为：

GladeXML 对象名；

```
对象名 = glade_xml_new(const char *fname,
                      const char *root,
                      const char *domain);
```

其中，`fname` 参数为 Glade 项目文件的路径和名称。`root` 参数为项目文件中顶层构件节点，`NULL` 表示使用 Glade 项目文件中定义的所有构件。如果希望 GladeXML 对象只代表一个窗体构件和窗体内的子构件，那么 `root` 参数可设置为该窗体构件在 Glade 项目中定义的名称。`domain` 参数为翻译文件的名称，`NULL` 为默认。

(2) 获得界面构件。可从有效的 GladeXML 对象中获得界面构件，然后对其进行操作。至少要获得顶层窗体构件，然后使用 `GTK+` 函数将其显示出来。`glade_xml_get_widget()` 函数用于获得界面构件，它的一般形式为：

```
GtkWidget *glade_xml_get_widget(GladeXML *self,
                                const char *name);
```

`self` 参数为 GladeXML 对象的名称，`name` 参数为 Glade 项目中界面构件的名称。函数返回值是 `GtkWidget` 对象。

(3) 连接信号。在 Glade 中定义了信号后，可使用 `glade_xml_signal_autoconnect()` 函数将这些信号全部连接到 C 语言代码中。它的一般形式为：

```
void glade_xml_signal_autoconnect(GladeXML *self);
```

下面用一个例子说明 `libglade` 函数库的基本操作方法，首先在 Glade 里创建一个名为 `ui.glade` 的项目文件。然后在项目文件中添加一个窗体构件，并在其中装入一个纵向组装箱、一个标签构件和一个按钮构件。将窗体命名为 `MainWindow`，标签命名为 `label`。在“公共”选项卡内将按钮的“编辑类型”属性设为“库存”，将“库存按钮”属性设为“退出(Q)”，如图 26.43 所示。

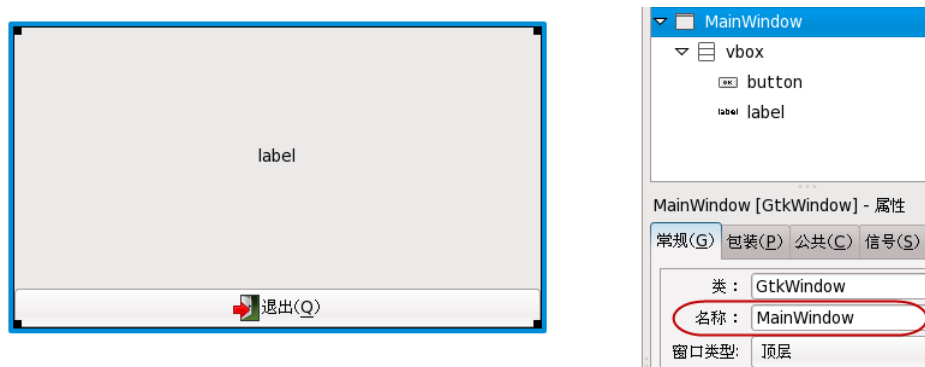


图 26.43 一个简单的 Glade 项目

然后为窗体构件连接信号。选择窗体，单击“信号”选项卡。找到信号列表中 `GtkWidget` 项，单击左侧展开器，展开 `GtkWidget` 类中定义的信号。选择 `delete-event` 信号，为其添加回调函数 `gtk_main_quit`，如图 26.44 所示。

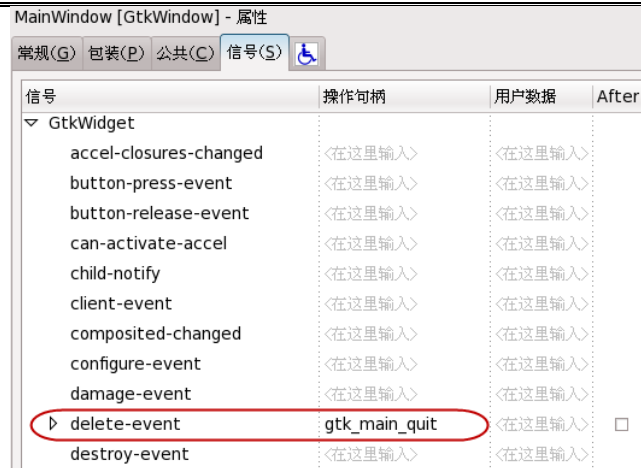


图 26.44 为窗体连接信号

选择按钮构件，选择“信号”选项卡。找到信号列表中 `GtkButton` 项，单击左侧展开器，展开 `GtkWidget` 类中定义的信号。选择 `clicked` 信号，为其添加回调函数 `gtk_main_quit`。保存 Glade 项目文件。

在上述 Glade 项目文件同一个目录中建立一个 C 语言源代码文件，可以使用任意文件名。编辑该文件，写入如下代码：

```
#include <gtk-2.0/gtk/gtk.h>
#include <glib-2.0/glib.h>
#include <libglade-2.0/glade/glade.h> // 包含 libglade 函数库
int main(int argc, char *argv[])
{
    gtk_init(&argc, &argv); // 初始化 GTK+ 库
    GladeXML *ui; // 声明 GladeXML 类型变量
    ui = glade_xml_new("ui.glade", NULL, NULL); // 创建 GladeXML 对象
    GtkWidget *window; // 声明 GtkWidget 类型变量
    window = glade_xml_get_widget(ui, "MainWindow"); // 从 GladeXML 对象获得 GtkWidget 界面构件

    GtkWidget *label;
    label = glade_xml_get_widget(ui, "label");
    gtk_label_set_label(GTK_LABEL(label), "Hello World!"); // 修改界面构件的属性

    gtk_widget_show_all(window); // 显示 window 内的所有构件
    glade_xml_signal_autoconnect(ui); // 连接 GladeXML 对象所有已定义信号
    gtk_main(); // 开始 GTK+ 主循环
    return 0;
}
```

在编译参数中加入编译参数“``pkg-config --cflags --libs glib-2.0 gtk+-2.0 libglade-2.0``”，编译该程序。编译成功后运行程序，如图 26.45 所示。


程序中使用 `glade_xml_new()` 函数读取 Glade 项目文件 `ui.glade`，创建了一个 `GladeXML` 对象。



图 26.45 libglade 演示

然后通过 `glade_xml_get_widget()` 函数获得了 Glade 项目中的 `window` 和 `label` 构件。程序运行时，修改了标签 `label` 的字符串，并使用 `gtk_widget_show_all()` 函数将窗体构件 `window` 内的所有构件显示出来。

Glade 项目中为 `window` 和 `button` 构件所连接的信号在执行 `glade_xml_signal_autoconnect()` 函数后即可被程序所使用。所以单击窗体的关闭按钮，或者按下“退出”按钮时，将调用 `gtk_main_quit()` 函数结束程序。

 **注意：**通过 C 语言源代码文件编译的可执行文件与 Glade 项目文件是分离的，如果删除了 Glade 项目文件或改变其路径，那么可执行文件也无法启动图形界面，并且造成执行错误。如果在 Glade 中修改了项目文件，只要不影响可执行文件的调用，那么无需重新修改和编译源代码。例如在 Glade 中只该变了窗体构件的标题属性，再次运行可执行文件时，将显示新修改的窗体标题。

## 26.3.2 使用 libglade 多语言支持

Linux 系统本身具备完善的多语言支持体系，可使同一个可执行文件拥有不同地方语言的图形界面。这其中涉及到两个重要概念：国际化与本地化。

国际化是指将开发者原先使用的母语翻译成多种其他的语言。由于实现翻译的途径、翻译的工作效率、翻译的可重用性等因素各不相同，使翻译工作面临很大困境，也阻碍了软件的推广和应用。为了方便地将软件翻译成不同语言的版本，就需要一套翻译规范和通用工具，于是就诞生了 i18n 工具集。i18n 即“internationalization”的缩写形式，主要使用 `gettext` 软件包实现国际化支持。

本地化是指可执行文件能够根据当前的语言环境选择图形界面上使用的语言。除语言以外，字符编码、语法、度量单位、日期时间格式、阅读习惯、使用习惯等也是需要考虑的问题，因此设计了 l10n 工具集。l10n 是 localization 的缩写形式，主要使用 `locale` 软件包实现本地化支持。

在概念中，本地化包含国际化，两者相辅相成。本节假设开发者的母语为英语，本地应用为简体中文，演示 `libglade` 对多语言的支持。下面是具体操作步骤：

### 1. 为 Glade 项目创建 po 和 mo 文件

`po` 文件意为可移植对象；`mo` 文件意为机器对象。`po` 文件是面向翻译人员、提取于 Glade 项目的一种资源文件。当软件升级的时候，通过使用 `gettext` 软件包处理 `po` 文件，可以在一定程度上使翻译成果得以继承，减轻翻译人员的负担。`mo` 文件是面向计算机的、由 `po` 文件通过 `gettext` 软件包编译而成的二进制文件。程序通过读取 `mo` 文件使自身的界面转换成用户使用的语言。

假设已建立了一个 Glade 项目文件 `ui.glade`，其中包含一个主窗体 `window`，窗体标题为 `User Information`。窗体中放置着一个表格容器，然后在容器中装入一组标签构件，构件的标签分别是 `Name`、`Gender` 和 `Age`。再将标签为 `Male` 和 `Female` 的单选按钮装入表格。



最后装入一个文本框、一个微调按钮和一组按钮库按钮，如图 26.46 所示。

在包含 Glade 项目文件的目录中创建名为 `po` 的目录，进入 `po` 目录，创建一个名为 `POTFILES.in` 的新文件。用文本编辑器打开该文件，输入 Glade 项目文件的文件名后保存，如下所示：

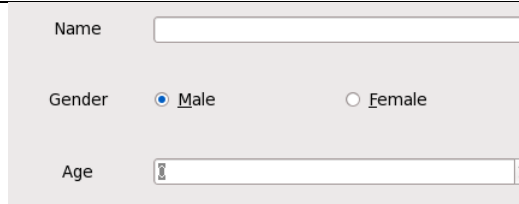


图 26.46 Glade 设计的英文界面

```
ui.glade
```

安装 `intltool` 工具集，其中包含了 `gettext` 软件包。在终端中输入下列命令：

```
yum install intltool
```

安装后，将工作目录设为“`POTFILES.in`”文件的目录。输入下列命令创建 `po` 文件：

```
intltool-update --pot gettext-package=ui
```

`gettext-package` 参数的名称为 Glade 项目文件的前缀名。命令执行成功时会创建 `untitled.pot` 文件，该文件是 `po` 文件的模板。复制该文件，创建名为 `zh_CN.po` 的副本。用文本编辑器打开该文件进行，将 `charset=CHARSET` 改为 `utf-8`，将 `msgid` 后的英文字符串翻译到下一行 `msgstr` 后。如下所示：

```
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: 8bit\n"
#: ../ui.glade.h:1
msgid "Age"
msgstr "年龄"
#: ../ui.glade.h:2
msgid "Gender"
msgstr "性别"
#: ../ui.glade.h:3
msgid "Name"
msgstr "姓名"
#: ../ui.glade.h:4
msgid "User Information"
msgstr "用户信息"
#: ../ui.glade.h:5
msgid "_Female"
msgstr "女"
#: ../ui.glade.h:6
msgid "_Male"
msgstr "男"
#: ../ui.glade.h:7
msgid "gtk-cancel"
msgstr "gtk-cancel"
#: ../ui.glade.h:8
msgid "gtk-ok"
msgstr "gtk-ok"
```

如果该文件中包含按钮库按钮的名称，如 `gtk-cancel`，则在译文中也保留原来的名称。然后执行下列命令，将 `po` 文件编译为 `mo` 文件：

```
msgfmt zh_CN.po
```

在 po 中创建 zh\_CN/LC\_MESSAGES 目录，复制 zh\_CN.mo 到新建立的目录，并改名为 ui.mo。

## 2. 编辑C语言源代码文件

在 Glade 项目文件所在的目录创建一个 C 语言源代码文件，文件的内容如下：

```
#include <gtk-2.0/gtk/gtk.h>
#include <glib-2.0/glib.h>
#include <libglade-2.0/glade/glade.h>
#include <libintl.h>                                // 提供 gettext 支持
#define _(String) gettext(String)                  // 翻译字符串
#define N_(String) String
#define PACKAGE "ui"                                // 定义 mo 文件前缀名
#define LOCALEDIR "./po"                            // 定义 mo 文件搜索路径
int main(int argc, char *argv[])
{
    gtk_set_locale();                                // 设置本地语言和字符集
    bindtextdomain(PACKAGE, LOCALEDIR);               // 设置 mo 文件的路径
    textdomain(PACKAGE);                             // 设置 mo 文件前缀名
    gtk_init(&argc, &argv);
    GladeXML *ui;
    ui = glade_xml_new("ui.glade", NULL, NULL);
    GtkWidget *window;
    window = glade_xml_get_widget(ui, "window");
    gtk_widget_show_all(window);                     // 显示窗体
    glade_xml_signal_autoconnect(ui);
    gtk_main();
    return 0;
}
```

程序中定义了宏 `#define _(String) gettext(String)` 和 `#define N_(String) String`。前一条宏命令是将 “\_( )” 内的字符串作为 `gettext()` 函数的参数，Glade 项目文件中可翻译字符串在 C 语言源代码中是用 “\_( )” 包围起来的。不可翻译的字符串则是用 “N\_( )” 包围，后一条宏将其直接转化为普通的字符串。

主函数中使用 `gtk_set_locale()` 函数设置了本地语言和字符集。可在终端中使用 `locale` 命令查看。该设置会影响到翻译的结果，如下所示：

```
LANG=zh_CN.UTF-8
LC_CTYPE="zh_CN.UTF-8"
LC_NUMERIC="zh_CN.UTF-8"
LC_TIME="zh_CN.UTF-8"
LC_COLLATE="zh_CN.UTF-8"
LC_MONETARY="zh_CN.UTF-8"
LC_MESSAGES="zh_CN.UTF-8"
LC_PAPER="zh_CN.UTF-8"
LC_NAME="zh_CN.UTF-8"
LC_ADDRESS="zh_CN.UTF-8"
LC_TELEPHONE="zh_CN.UTF-8"
LC_MEASUREMENT="zh_CN.UTF-8"
LC_IDENTIFICATION="zh_CN.UTF-8"
LC_ALL=
```

可在环境变量中修改 locale 的值，例如修改用户主目录下的 .bash\_profile 文件，在末尾加入如下语句：

```
export LANG=zh_CN.UTF-8
export LANGUAGE=zh_CN.UTF-8
export LC_ALL=zh_CN.UTF-8
```

上述操作对当前登录的用户有效，如果要对所有用户生效，可修改 “/etc/sysconfig” 目录下的 .i18n 文件，将 LANG 选项设置为：

```
lang zh_CN.UTF-8
```

bindtextdomain() 函数定义了 mo 文件的路径，textdomain() 函数定义了 mo 文件的前缀名。编译程序后，假设当前 Linux 发行版的语言设置为简体中文 (zh\_CN)，并使用 UTF-8 字符集，那么显示的程序界面将为中文，如图 26.47 所示。



图 26.47 显示为中文的 Glade 界面

## 26.4 小 结

本章介绍了使用 Glade 设计程序界面的方法，以及使用 libglade 函数库在 C 语言代码中进行代码联编的方法。Glade 是非常方便的界面开发工具，在项目中使用 Glade 可缩短界面代码的开发周期。但是，Glade 也有其不足之处，对于过于复杂的界面或有个性化要求的界面不能起到简化编码的作用。因此，在项目中使用 Glade 设计程序界面前应先进行评估，对于大多数管理类、数据库类程序可优先考虑使用 Glade。