第9章 杂项构件

9.1 标签构件GtkLabel

GtkLabel(标签构件)是GTK中最常用的构件，实际上它很简单。因为没有相关联的 X窗口，标签构件不能引发信号。如果需要引发信号，可以将它放在一个事件盒构件中，或放在按钮构件里面。

用以下函数创建新标签构件：

```
GtkWidget *gtk_label_new(char *str );
```

唯一的参数是要由标签显示的字符串。

创建标签构件后，要改变标签内的文本，用以下函数：

```
void gtk_label_set_text( GtkLabel *lacbel,char *str );
```

第一参数是前面创建的标签构件 (用GTK_LABEL()宏转换)，并且第二个参数是新字符串。如果需要，新字符串需要的空间会做自动调整。在字符串中放置换行符，可以创建多行标签。

用以下函数取得标签的当前文本：

```
void gtk_label_get( GtkLabel *Label,char **str );
```

第一个参数是前面创建的标签构件，并且第二个参数是要返回的字符串。不要释放返回的字符串，因为GTK内部要使用它。

标签的文本可以用以下函数设置对齐方式：

```
void gtk_label_set_justify( GtkLabel *Label,
                            GtkJustification jtype );
```

jtype的值可以是:

GTK_JUSTIFY_LEFT	左对齐
GTK_JUSTIFY_RIGHT	右对齐
GTK_JUSTIFY_CENTER	居中对齐(默认)
GTK_JUSTIFY_FILL	充满

标签构件的文本会自动换行。用以下函数激活“ 自动换行 ”:

```
void gtk_label_set_line_wrap (GtkLabel *Label, gboolean wrap);
```

wrap参数可取TRUE或FALSE，对应于自动换行和不自动换行。

如果想要使标签构件加下划线，可以在标签构件中设置显示模式：

```
void gtk_label_set_pattern (GtkLabel *Label,const gchar *pattern);
```

pattern参数指定下划线的外观。它由一串下划线和空格组成。下划线指示标签的相应字符应该加一个下划线。例如，“ _ _ ”将在标签的第1、第2个字符和第8、第9个字符加下划线。

下面是一个说明这些函数的短例子。这个例子用框架构件能更好地示范标签的风格。

/* GtkLabel示例开始 label.c */

```
#include <gtk/gtk.h>
```

```
int main( int   argc,
          char *argv[] )
{
  static GtkWidget *window = NULL;
  GtkWidget *hbox;
  GtkWidget *vbox;
  GtkWidget *frame;
  GtkWidget *label;

  /* 初始化GTK */
  gtk_init(&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC(gtk_main_quit),
                      NULL);

  gtk_window_set_title (GTK_WINDOW (window), "Label");
  vbox = gtk_vbox_new (FALSE, 5);
  hbox = gtk_hbox_new (FALSE, 5);
  gtk_container_add (GTK_CONTAINER (window), hbox);
  gtk_box_pack_start (GTK_BOX (hbox), vbox, FALSE, FALSE, 0);
  gtk_container_set_border_width (GTK_CONTAINER (window), 5);

  frame = gtk_frame_new ("Normal Label");
  label = gtk_label_new ("This is a Normal label");
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  frame = gtk_frame_new ("Multi-line Label");
  label = gtk_label_new ("This is a Multi-line label.\nSecond line\n" \
                         "Third line");
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  frame = gtk_frame_new ("Left Justified Label");
  label = gtk_label_new ("This is a Left-Justified\n" \
                         "Multi-line label.\nThird      line");
  gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_LEFT);
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  frame = gtk_frame_new ("Right Justified Label");
  label = gtk_label_new ("This is a Right-Justified\nMulti-line label.\n" \
                         "Fourth line, (j/k)");
  gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_RIGHT);
  gtk_container_add (GTK_CONTAINER (frame), label);
  gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

  vbox = gtk_vbox_new (FALSE, 5);
  gtk_box_pack_start (GTK_BOX (hbox), vbox, FALSE, FALSE, 0);
```

```
    frame = gtk_frame_new ("Line wrapped label");
    label = gtk_label_new ("This is an example of a line-wrapped label.  It " \
                           "should not be taking up the entire              "
                           /* 一大段空格, 用来测试间距 */\
                           "width allocated to it, but automatically " \
                           "wraps the words to fit.  " \
                           "The time has come, for all good men, to come to " \
                           "the aid of their party.  " \
                           "The sixth sheik's six sheep's sick.\n" \
                           "     It supports multiple paragraphs correctly, " \
                           "and  correctly   adds "\
                           "many          extra  spaces. ");
    gtk_label_set_line_wrap (GTK_LABEL (label), TRUE);
    gtk_container_add (GTK_CONTAINER (frame), label);
    gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

    frame = gtk_frame_new ("Filled, wrapped label");
    label = gtk_label_new ("This is an example of a line-wrapped, filled label
  " \
                           "It should be taking "\
                           "up the entire            width allocated to it.  "
\
                           "Here is a sentence to prove "\
                           "my point.  Here is another sentence. "\
                           "Here comes the sun, do de do de do.\n"\
                           "    This is a new paragraph.\n"\
                           "    This is another newer, longer, better " \
                           "paragraph.  It is coming to an end, "\
                           "unfortunately.");
    gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_FILL);
    gtk_label_set_line_wrap (GTK_LABEL (label), TRUE);
    gtk_container_add (GTK_CONTAINER (frame), label);
    gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

    frame = gtk_frame_new ("Underlined label");
    label = gtk_label_new ("This label is underlined!\n"
                           "This one is underlined in quite a funky fashion");
    gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_LEFT);
    gtk_label_set_pattern (GTK_LABEL (label),
                           "_________________________ _ _________ _ \
                            ______     __ _______ ___");
    gtk_container_add (GTK_CONTAINER (frame), label);
    gtk_box_pack_start (GTK_BOX (vbox), frame, FALSE, FALSE, 0);

    gtk_widget_show_all (window);

    gtk_main ();

    return(0);
  }
  /* 示例结束 */
```

图9-1是上面代码的运行结果。这个例子展示了GtkLabel构件的各种属性。

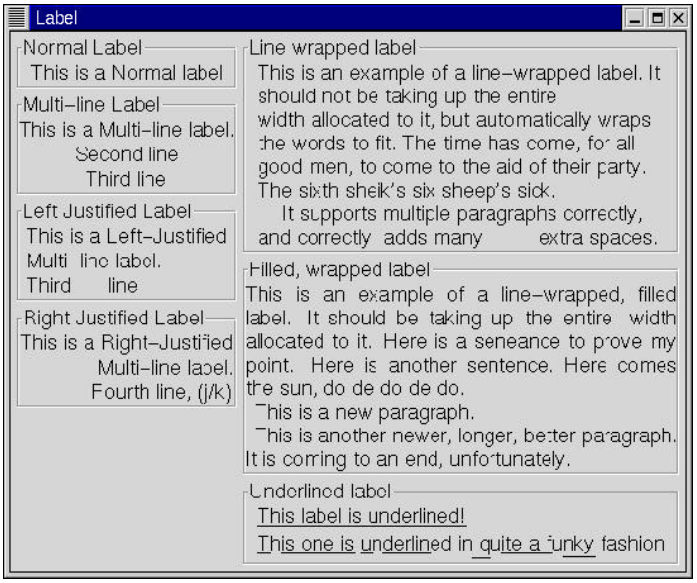

图9-1 标签构件

9.2 箭头构件GtkArrow

GtkArrow(箭头构件)画一个箭头，面向几种不同的方向，并有几种不同的风格。在许多应用程序中，常用于创建带箭头的按钮。和标签构件一样，它不能引发信号。

只有两个函数用来操纵箭头构件：

```
GtkWidget *gtk_arrow_new( GtkArrowType arrow_type,
                          GtkShadowType shadow_type );
void gtk_arrow_set( GtkArrow *arrow, GtkArrowType arrow_type,
                          GtkShadowType shadow_type );
```

第一个函数创建新的箭头构件，指明构件的类型和外观；第二个函数用来改变箭头构件类型和外观。

arrow_type参数指示箭头指向哪个方向，可以取下列值：

GTK_ARROW_UP	向上
GTK_ARROW_DOWN	向下
GTK_ARROW_LEFT	向左
GTK_ARROW_RIGHT	向右

shadow_type参数指明箭头的投影的类型，可以取下列值：

GTK_SHADOW_IN

GTK_SHADOW_OUT (缺省值)

GTK_SHADOW_ETCHED_IN

GTK_SHADOW_ETCHED_OUT

下面是说明这些类型和外观的例子。

```
/* GtkArrow示例arrow.c */
```

```
#include <gtk/gtk.h>

/* 用指定的参数创建一个箭头构件并将它组装到按钮中 */
GtkWidget *create_arrow_button( GtkArrowType  arrow_type,
                                GtkShadowType shadow_type )
{
  GtkWidget *button;
  GtkWidget *arrow;

  button = gtk_button_new();
  arrow = gtk_arrow_new (arrow_type, shadow_type);

  gtk_container_add (GTK_CONTAINER (button), arrow);

  gtk_widget_show(button);
  gtk_widget_show(arrow);

  return(button);
}

int main( int   argc,
          char *argv[] )
{
  /* 构件的存储类型是GtkWidget */
  GtkWidget *window;
  GtkWidget *button;
  GtkWidget *box;

  /* 初始化Gtk */
  gtk_init (&argc, &argv);

  /* 创建一个新窗口*/
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_window_set_title (GTK_WINDOW (window), "Arrow Buttons");

  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC (gtk_main_quit), NULL);

  /* 设置窗口的边框的宽度*/
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);

  /* 创建一个组装盒以容纳箭头/按钮*/
  box = gtk_hbox_new (FALSE, 0);
  gtk_container_set_border_width (GTK_CONTAINER (box), 2);
  gtk_container_add (GTK_CONTAINER (window), box);

  /* 组装、显示所有的构件*/
  gtk_widget_show(box);

  button = create_arrow_button(GTK_ARROW_UP, GTK_SHADOW_IN);
```

```
    gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 3);

    button = create_arrow_button(GTK_ARROW_DOWN, GTK_SHADOW_OUT);
    gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 3);

    button = create_arrow_button(GTK_ARROW_LEFT, GTK_SHADOW_ETCHED_IN);
    gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 3);

    button = create_arrow_button(GTK_ARROW_RIGHT, GTK_SHADOW_ETCHED_OUT);
    gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 3);

    gtk_widget_show (window);

    /* 进入主循环，等待用户的动作 */
    gtk_main ();

    return(0);
}
/* 示例结束 */
```

图9-2 GtkArrow构件

上面代码的运行效果见图9-2。在窗口上创建了四个带箭头的按钮。

9.3 工具提示对象GtkTooltips

工具提示对象（GtkTooltips）就是当鼠标指针移到按钮或其他构件上并停留几秒时，弹出的文本串。工具提示对象很容易使用，所以在此仅仅对它们进行解释，不再举例。本书的其他示例里面有很多都用到了工具提示对象。

不接收事件的构件(没有自己的X窗口的构件)不能和工具提示对象一起工作。

可以使用gtk_tooltips_new()函数创建工具提示对象。因为GtkTooltips对象可以重复使用，一般在应用程序中仅需要调用这个函数一次。

```
GtkTooltips *gtk_tooltips_new(void);
```

一旦已创建新的工具提示，并且希望在某个构件上应用它，可调用以下函数设置它：

```
void gtk_tooltips_set_tip( GtkTooltips *tooltip,
GtkWidget *widget,const gchar *tip_text,const
gchar *tip_private );
```

第一个参数是已经创建的工具提示对象，其后第二个参数是希望弹出工具提示的构件，第三个参数是要弹出的文本。最后一个参数是作为标识符的文本串，当用 GtkTipsQuery实现上下文敏感的帮助时要引用该标识符。目前,你可以把它设置为NULL.

下面有个短例子：

```
GtkTooltips *tooltips;
GtkWidget *button;
.
.
tooltips = gtk_tooltips_new ();
button = gtk_button_new_with_label ("button 1");
.
.
gtk_tooltips_set_tip (tooltips, button, "This is button 1", NULL);
```

还有其他与工具提示有关的函数，下面仅仅列出一些函数的简要描述。

```
void gtk_tooltips_enable( GtkTooltips *tooltip);
```

激活已经禁用的工具提示对象。

```
void gtk_tooltips_disable( GtkTooltips *tooltip);
```

禁用已经激活的工具提示对象。

```
void gtk_tooltips_set_delay( GtkTooltips *tooltip, gint delay);
```

设置鼠标在构件上停留多少毫秒后弹出工具提示，默认是 500毫秒(半秒)。

```
void gtk_tooltips_set_colors( GtkTooltips *tooltips,
                              GdkColor    *background,
                              GdkColor    *foreground );
```

设置工具提示的前景色和背景色。

上面是所有与工具提示有关的函数，实际上比你想要知道的还多。

9.4 进度条构件GtkProgressBar

进度条用于显示正在进行的操作的状态。它相当容易使用，在下面的代码中可以看到。下面的内容从创建一个新进度条开始。

有两种方法创建进度条，简单的方法不需要参数，另一种方法用一个调整对象作为参数。如果前者使用,进度条创建它拥有的调整对象。

```
GtkWidget *gtk_progress_bar_new(void);
GtkWidget *gtk_progress_bar_new_with_adjustment( GtkAdjustment *adjustment);
```

第二种方法的优势是我们能用调整对象明确指定进度条的范围参数。

进度条的调整对象能用下面的函数动态改变：

```
void gtk_progress_set_adjustment( GtkProgress *progress,
                                  GtkAdjustment *adjustment);
```

既然进度条已经创建，那就可以使用它了。

```
void gtk_progress_bar_update( GtkProgressBar *pbar, gfloat percentage);
```

更新进度条时，第一个参数是希望操作的进度条，第二个参数是“已完成”的百分比，意思是进度条从0～100%已经填充的数量。它以0～1范围的实数传递给函数。

GTK 1.2版已经给进度条添加了一个新的功能，那就是允许它以不同的方法显示其值，并通知用户它的当前值和范围。

进度条可以用以下函数设置它的移动方向：

```
void gtk_progress_bar_set_orientation( GtkProgressBar *pbar,
                       GtkProgressBarOrientation orientation);
```

orientation参数可以取下列值之一，以指示进度条的移动方向：

GTK_PROGRESS_LEFT_TO_RIGHT	从左向右
GTK_PROGRESS_RIGHT_TO_LEFT	从右向左
GTK_PROGRESS_BOTTOM_TO_TOP	从下向上
GTK_PROGRESS_TOP_TO_BOTTOM	从上向下

进度条可以以连续和间断的方式显示进度处理的数值。在连续的方式下，进度条每个值都会更新；在间断方式下，进度条以不连续的方式更新。更新的次数是可配置的。

进度条的式样可以用以下函数进行更新：

```
void gtk_progress_bar_set_bar_style( GtkProgressBar *pbar,
                                     GtkProgressBarStyle style);
```

style参数取以下两种值：

GTK_PROGRESS_CONTINUOUS　　连续更新

GTK_PROGRESS_DISCRETE　　间断更新

间断更新的次数可以用以下函数设置：

```
gtk_progress_bar_set_discrete_blocks( GtkProgressBar *pbar,
                                      guint blocks);
```

除了指示进度已经发生的数量以外，进度条还可以设置为仅仅指示有活动在继续。这些设置在进度无法按数值度量的情况下很有用。进度条的活动模式不会受进度条的 style参数的影响，并且会覆盖它的设置。其方式只能是TRUE或者FALSE，可以用下列函数确定：

```
void gtk_progress_set_activity_mode( GtkProgress *progress,
                                     guint activity_mode );
```

活动指示的步数和活动的块数由以下函数设置：

```
void gtk_progress_bar_set_activity_step( GtkProgressBar *pbar, guint step);
void gtk_progress_bar_set_activity_blocks( GtkProgressBar *pbar, guint
blocks);
```

在连续的方式下，进度条可以用下列函数在滑槽内显示一个可配置的文本串：

```
void gtk_progress_set_format_string( GtkProgress *progress, gchar *format);
```

其中，format参数与C语言中的printf函数的格式参数相似。

下列格式可以用于格式化串：

%p　百分比

%v　值

%l　低范围值

%u　高限范围值

是否显示文本串可以用以下函数开/关：

```
void gtk_progress_set_show_text( GtkProgress *progress, gint show_text );
```

show_text参数是布尔型的，可取值为TRUE/FALSE。

文本的外观能做进一步的修改，如下所示：

```
void gtk_progress_set_text_alignment( GtkProgress *progress,
gfloat x_align, gfloat y_align );
```

x_align和 y_align参数取0.0和 1.0之间的值。这些值指定文本串在滑槽内的位置。两个值都取为0.0将把文本串放在左上角；取值0.5(默认)让文本居于滑槽中心；设为1.0则将文本串置于右下角。

进度条对象的当前文本设置能由下列两个函数从当前文本或从指定的调整对象中取得。这些函数返回的字符串应该由应用程序释放 (用 g_free()函数)。这些函数返回要显示在滑槽内的格式化字符串。

```
gchar *gtk_progress_get_current_text( GtkProgress *progress );
gchar *gtk_progress_get_text_from_value( GtkProgress *progress, gfloat value );
```

还有另一种改变进度条对象的范围和取值的方法。使用下列函数：

```
void gtk_progress_configure( GtkProgress *progress, gfloat value, gfloat min,
gfloat max);
```

这个函数提供了相当简单的使用进度条对象的范围和取值的接口。

其余的函数可以以各种类型和格式获得或设置当前进度条对象的值：

```
void gtk_progress_set_percentage( GtkProgress *progress, gfloat percentage);
void gtk_progress_set_value( GtkProgress *progress, gfloat value);
gfloat gtk_progress_get_value( GtkProgress *progress);
gfloat gtk_progress_get_current_percentage( GtkProgress *progress);
gfloat gtk_progress_get_percentage_from_value( GtkProgress *progress,
gfloat value);
```

这些函数从字面上就可以了解它的含义。最后一个函数使用指定进度条的调整对象计算给定范围值的百分比。

进度条通常和timeout或其他函数同时使用，使应用程序就像是多任务一样。一般都用同样的方式使用gtk_progress_bar_update函数。

下面是一个进度条的例子，用 timeout函数更新进度条的值。代码也演示了怎样复位进度条。

```
/* GtkProgressBar示例progressbar.c */

#include <gtk/gtk.h>

typedef struct _ProgressData {
    GtkWidget *window;
    GtkWidget *pbar;
    int timer;
} ProgressData;

/* 更新进度条，这样就能够看到进度条的移动 */
gint progress_timeout( gpointer data )
{
    gfloat new_val;
    GtkAdjustment *adj;

    /* 使用在调整对象中设置的取值范围计算进度条的值 */

    new_val = gtk_progress_get_value( GTK_PROGRESS(data) ) + 1;

    adj = GTK_PROGRESS (data)->adjustment;
    if (new_val > adj->upper)
      new_val = adj->lower;

    /* 设置进度条的新值 */
    gtk_progress_set_value (GTK_PROGRESS (data), new_val);

    /* 这是一个timeout函数，返回TRUE，这样它就能够继续调用 */
    return(TRUE);
}

/* 回调函数，切换在进度条内的滑槽上的文本显示 */
void toggle_show_text( GtkWidget    *widget,
                       ProgressData *pdata )
```

```
{
    gtk_progress_set_show_text (GTK_PROGRESS (pdata->pbar),
                                GTK_TOGGLE_BUTTON (widget)->active);
}

/* 回调函数，切换进度条的活动模式 */
void toggle_activity_mode( GtkWidget    *widget,
                           ProgressData *pdata )
{
    gtk_progress_set_activity_mode (GTK_PROGRESS (pdata->pbar),
                                    GTK_TOGGLE_BUTTON (widget)->active);
}

/* 回调函数，切换进度条的连续模式 */
void set_continuous_mode( GtkWidget    *widget,
                          ProgressData *pdata )
{
    gtk_progress_bar_set_bar_style (GTK_PROGRESS_BAR (pdata->pbar),
                                    GTK_PROGRESS_CONTINUOUS);
}

/* 回调函数，切换进度条的间断模式 */
void set_discrete_mode( GtkWidget    *widget,
                        ProgressData *pdata )
{
    gtk_progress_bar_set_bar_style (GTK_PROGRESS_BAR (pdata->pbar),
                                    GTK_PROGRESS_DISCRETE);
}

/* 清除分配的内存，删除timeout函数*/
void destroy_progress( GtkWidget     *widget,
                       ProgressData *pdata)
{
    gtk_timeout_remove (pdata->timer);
    pdata->timer = 0;
    pdata->window = NULL;
    g_free(pdata);
    gtk_main_quit();
}

int main( int   argc,
          char *argv[])
{
    ProgressData *pdata;
    GtkWidget *align;
    GtkWidget *separator;
    GtkWidget *table;
    GtkAdjustment *adj;
    GtkWidget *button;
    GtkWidget *check;
    GtkWidget *vbox;
```

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

/* 为传递到回调函数中的数据分配内存 */
pdata = g_malloc( sizeof(ProgressData) );

pdata->window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_policy (GTK_WINDOW (pdata->window), FALSE, FALSE, TRUE);

gtk_signal_connect (GTK_OBJECT (pdata->window), "destroy",
                    GTK_SIGNAL_FUNC (destroy_progress),
                    pdata);
gtk_window_set_title (GTK_WINDOW (pdata->window), "GtkProgressBar");
gtk_container_set_border_width (GTK_CONTAINER (pdata->window), 0);

vbox = gtk_vbox_new (FALSE, 5);
gtk_container_set_border_width (GTK_CONTAINER (vbox), 10);
gtk_container_add (GTK_CONTAINER (pdata->window), vbox);
gtk_widget_show(vbox);

/* 创建一个居中对齐的对象 */
align = gtk_alignment_new (0.5, 0.5, 0, 0);
gtk_box_pack_start (GTK_BOX (vbox), align, FALSE, FALSE, 5);
gtk_widget_show(align);

/* 创建一个调整对象，以保持进度条的范围值 */
adj = (GtkAdjustment *) gtk_adjustment_new (0, 1, 150, 0, 0, 0);

/* 使用前面创建的调整对象创建一个进度条 */
pdata->pbar = gtk_progress_bar_new_with_adjustment (adj);

/* 设置能显示在进度条的滑槽内的字符串的格式
 * %p - 百分比
 * %v - 值
 * %l - 底限范围值
 * %u - 上限范围值 */
gtk_progress_set_format_string (GTK_PROGRESS (pdata->pbar),
                                "%v from [%l-%u] (=%p%%)");
gtk_container_add (GTK_CONTAINER (align), pdata->pbar);
gtk_widget_show(pdata->pbar);

/* 添加一个计时的回调函数，以更新进度条的值 */
pdata->timer = gtk_timeout_add (100, progress_timeout, pdata->pbar);

separator = gtk_hseparator_new ();
gtk_box_pack_start (GTK_BOX (vbox), separator, FALSE, FALSE, 0);
gtk_widget_show(separator);

/* 行数、列数、同质性 */
table = gtk_table_new (2, 3, FALSE);
gtk_box_pack_start (GTK_BOX (vbox), table, FALSE, TRUE, 0);
gtk_widget_show(table);
```

```
/* 添加一个检查按钮，以选择显示在滑槽内的文本 */
check = gtk_check_button_new_with_label ("Show text");
gtk_table_attach (GTK_TABLE (table), check, 0, 1, 0, 1,
                  GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                  5, 5);
gtk_signal_connect (GTK_OBJECT (check), "clicked",
                    GTK_SIGNAL_FUNC (toggle_show_text),
                    pdata);
gtk_widget_show(check);

/* 添加一个检查按钮，切换活动状态 */
check = gtk_check_button_new_with_label ("Activity mode");
gtk_table_attach (GTK_TABLE (table), check, 0, 1, 1, 2,
                  GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                  5, 5);
gtk_signal_connect (GTK_OBJECT (check), "clicked",
                    GTK_SIGNAL_FUNC (toggle_activity_mode),
                    pdata);
gtk_widget_show(check);

separator = gtk_vseparator_new ();
gtk_table_attach (GTK_TABLE (table), separator, 1, 2, 0, 2,
                  GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                  5, 5);
gtk_widget_show(separator);

/* 添加一个无线按钮，以选择连续选择模式 */
button = gtk_radio_button_new_with_label (NULL, "Continuous");
gtk_table_attach (GTK_TABLE (table), button, 2, 3, 0, 1,
                  GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                  5, 5);
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (set_continuous_mode),
                    pdata);
gtk_widget_show (button);

/* 添加一个无线按钮，以选择间断模式 */
button = gtk_radio_button_new_with_label(
          gtk_radio_button_group (GTK_RADIO_BUTTON (button)),
          "Discrete");
gtk_table_attach (GTK_TABLE (table), button, 2, 3, 1, 2,
                  GTK_EXPAND | GTK_FILL, GTK_EXPAND | GTK_FILL,
                  5, 5);
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (set_discrete_mode),
                    pdata);
gtk_widget_show (button);

separator = gtk_hseparator_new ();
gtk_box_pack_start (GTK_BOX (vbox), separator, FALSE, FALSE, 0);
gtk_widget_show(separator);
```

```
    /* 添加一个按钮，用来退出应用程序 */
    button = gtk_button_new_with_label ("close");
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               (GtkSignalFunc) gtk_widget_destroy,
                               GTK_OBJECT (pdata->window));
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);

    /* 将按钮设置为缺省的*/
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);

    /* 将按钮冻结为缺省按钮，只要按回车键
     * 就相当于点击了这个按钮*/
    gtk_widget_grab_default (button);
    gtk_widget_show(button);

    gtk_widget_show (pdata->window);

    gtk_main ();
    return(0);
}   /* 示例结束 */
```

图9-3 进度条构件

将上面的代码保存为progressbar.c，编译后的运行效果见图9-3。其中演示了进度条的不同更新方式，你还可以设置是否显示文本，以及是否处于获得模式。

9.5 对话框构件

对话构件非常简单，事实上它仅仅是一个预先组装了几个构件到里面的窗口。

对话框是如下定义的：

```
struct GtkDialog { GtkWindow window;
GtkWidget *vbox; GtkWidget *action_area; };
```

从上面可以看到，对话框只是简单地创建一个窗口，并在顶部组装一个 GtkVBox，然后在GtkVBox中组装一个分隔线，再加一个称为“活动区”的GtkHBox.

对话框构件可以用于弹出消息，或者其他类似的任务。它只有一个函数：

```
GtkWidget *gtk_dialog_new(void);
```

要创建对话框，可以使用如下所示的函数：

```
GtkWidget *window;
window = gtk_dialog_new ();
```

你还可以在活动区中组装一个按钮：

```
button = ...
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (window)->action_area),
                    button,TRUE,TRUE, 0);
gtk_widget_show (button);
```

还可以在vbox里面组装一个标签构件，像下面那样：

```
label = gtk_label_new ("XXXXX");
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (window)->vbox),label,
                    TRUE,TRUE, 0);
gtk_widget_show (label);
```

作为一个例子，可以在对话框里面组装两个按钮：一个“确定”按钮和一个“取消”按钮，以便向用户提出疑问，或显示一个错误信息。然后可以把不同信号连接到每个按钮，对用户的选择进行响应。

如果由对话框提供的垂直和水平组装盒的简单功能不能满足你的需要，可以简单地在组装盒中添加其他外观构件。例如，可以在其中添加一个表格构件。

因为GtkDialog太简单，不能满足快速开发应用程序的要求，所以建议你使用GnomeDialog及其子构件。GnomeDialog将在第15章中介绍。

9.6 pixmap

pixmaps是包含图像的数据结构。这些图像可以用于不同的地方，但是最常见的是X桌面的图标，或用作鼠标指针。

仅有2种颜色的pixmap称为位图，另外有一些例程用于处理这种特殊情况。

了解pixmap有助于了解X窗口系统如何工作。在X系统下，与用户交互的应用程序不一定要在同一台计算机上运行。不同的应用程序，称为“客户”同时与一个显示图像和处理键盘和鼠标的程序通讯。这个直接与用户交互的程序被称为“显示服务器”或“X服务器”。

既然通信可能发生在网络上，那么，与X服务器保持一些信息是非常重要的。例如pixmap，就是储存在X服务器的内存中的。这意味着一旦设置了pixmap的值，它们就不再需要通过网络传输，而是传送一个“在此处显示编号为XYZ的pixmap”。即使你目前没有在X系统下使用GTK，使用像pixmap这样的结构也会使应用程序在X下工作得更好。

pixmap能从内存中的数据创建，或者用从文件中读出的数据创建。这里我们将介绍每种创建pixmap的方法。

```
GdkPixmap *gdk_bitmap_create_from_data( GdkWindow *window,
gchar *data, gint width, gint height );
```

这个函数使用从内存读取的数据创建单个位平面的pixmap (2种颜色)。数据的每位分别代表像素是打开或关闭。宽和高是以像素度量的。GdkWindow指针指向当前窗口，因为pixmap的资源指的是它要显示的屏幕的场合。

```
GdkPixmap *gdk_pixmap_create_from_data( GdkWindow *window,
                                        gchar     *data,
                                        gint       width,
                                        gint       height,
                                        gint       depth,
                                        GdkColor  *fg,
                                        GdkColor  *bg );
```

这个函数用于从指定的位图数据创建给定深度（颜色数）的pixmap图片。fg和bg是要用到的前景和背景色。

```
GdkPixmap *gdk_pixmap_create_from_xpm_d( GdkWindow  *window,
                                         GdkBitmap **mask,
                                         GdkColor   *transparent_color,
                                         gchar     **data );
```

XPM格式是X窗口系统中一种可读的pixmap图形表示，它得到了广泛的应用。有许多实用程序可以用来创建这种格式的图片。由filename参数指定的文件必须包含XPM格式的图形，这种图形会被加载到pixmap结构中。Mask参数指定在一个pixmap中哪一位是不透明的。所有

的其他位都使用由transparent_color所指定的颜色着色。下面是一个例子：

```
GdkPixmap *gdk_pixmap_create_from_xpm_d( GdkWindow  *window,
                                         GdkBitmap **mask,
                                         GdkColor   *transparent_color,
                                         gchar     **data );
```

小图像能以XPM格式合并到程序里面，一个 pixmap格式的图片能够用这些数据创建，而不是从文件中读进来。下面是这种数据的例子。

```
/* XPM */
static const char * xpm_data[] = {
"16 16 3 1",
"       c None",
".      c #000000000000",
"X      c #FFFFFFFFFFFF",
"                ",
"   ......       ",
"   .XXX.X.      ",
"   .XXX.XX.     ",
"   .XXX.XXX.    ",
"   .XXX.....    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .........    ",
"                ",
"                "};
```

当已经用完pixmap，并且不太可能再次使用它时，最好用 gdk_pixmap_unref()函数将这些资源释放。pixmap应该被看作是宝贵的资源，因为它们占据了终端用户的 X服务器进程的内存。即使开发应用程序时所用的计算机是一台功能强劲的“ 服务器 ”，最终用户使用的却可能是速度较慢的PC，因而释放不再使用的资源是一件很重要的工作。

一旦我们创建了pixmap，就能将它作为Gtk构件显示。我们必须创建 一个GtkPixmap构件以包含GDK pixmap图片。用下面的函数实现：

```
GtkWidget *gtk_pixmap_new( GdkPixmap *pixmap, GdkBitmap *mask );
```

其他的相关函数是：

```
guint gtk_pixmap_get_type(void);
void gtk_pixmap_set( GtkPixmap *pixmap, GdkPixmap *val, GdkBitmap *mask);
void gtk_pixmap_get( GtkPixmap *pixmap, GdkPixmap **val, GdkBitmap **mask);
```

gtk_pixmap_set函数一般用于正在管理的构件的 pixmap图片。val是用GDK创建的 pixmap图片。

下面的例子是在一个按钮上使用 pixmap图片。

```
/* pixmap示例开始pixmap.c */
#include <gtk/gtk.h>
```

```
/* XPM数据，用于"打开文件"图标 */
static const char * xpm_data[] = {
"16 16 3 1",
"       c None",
".      c #000000000000",
"X      c #FFFFFFFFFFFF",
"                ",
"   ......       ",
"   .XXX.X.      ",
"   .XXX.XX.     ",
"   .XXX.XXX.    ",
"   .XXX.....    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .XXXXXXX.    ",
"   .........    ",
"                ",
"                "};

/* 调用这个函数时(通过delete_event信号)，终止应用程序*/
void close_application( GtkWidget *widget, GdkEvent *event, gpointer data ) {
    gtk_main_quit();
}

/*点击按钮时调用这个函数，并打印一条信息*/
void button_clicked( GtkWidget *widget, gpointer data ) {
    printf( "button clicked\n" );
}

int main( int argc, char *argv[] )
{
    /* 构件的存储格式是GtkWidget */
    GtkWidget *window, *pixmapwid, *button;
    GdkPixmap *pixmap;
    GdkBitmap *mask;
    GtkStyle *style;

    /* 创建主窗口，为delete_event信号设置回调函数以终止应用程序*/
    gtk_init( &argc, &argv );
    window = gtk_window_new( GTK_WINDOW_TOPLEVEL );
    gtk_signal_connect( GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (close_application), NULL );
    gtk_container_set_border_width( GTK_CONTAINER (window), 10 );
    gtk_widget_show( window );

    /* 现在用gdk创建Pixmap */
    style = gtk_widget_get_style( window );
```

```
    pixmap = gdk_pixmap_create_from_xpm_d( window->window,  &mask,
                                        &style->bg[GTK_STATE_NORMAL],
                                        (gchar **)xpm_data );

    /* 用一个Pixmap构件包含这个pixmap图片 */
    pixmapwid = gtk_pixmap_new( pixmap, mask );
    gtk_widget_show( pixmapwid );

    /* 将pixmap构件放在一个按钮中*/
    button = gtk_button_new();
    gtk_container_add( GTK_CONTAINER(button), pixmapwid );
    gtk_container_add( GTK_CONTAINER(window), button );
    gtk_widget_show( button );

    gtk_signal_connect( GTK_OBJECT(button), "clicked",
                        GTK_SIGNAL_FUNC(button_clicked), NULL );

    /* 显示主窗口*/
    gtk_main ();

    return 0;
}
/*示例结束 */
```

如果要从当前目录的icon0.xpm中加载图片，我们可以用下面的方法创建 pixmap图片：

```
    /* 从文件加载pixmap图片*/
    pixmap = gdk_pixmap_create_from_xpm( window->window, &mask,
                                      &style->bg[GTK_STATE_NORMAL],
                                      "./icon0.xpm" );
    pixmapwid = gtk_pixmap_new( pixmap, mask );
    gtk_widget_show( pixmapwid );
    gtk_container_add( GTK_CONTAINER(window), pixmapwid );
```

使用pixmap图片的不利之处是它显示的对象总是矩形的，而不管图片内容如何。我们可以用具有自然形状的图片来创建应用程序和桌面程序。例如 ,对游戏软件的界面，我们希望有一个圆角按钮，那么可以用“ 有形窗口 ”来做到这一点。

一个有形窗口实际上就是 pixmap图像，它的背景颜色是透明的。这样，当背景图片有多种颜色时，我们就不会用不匹配的图标矩形边界来覆盖它。下面的例子在桌面上显示一个完整的独轮手推车。

```
/*示例开始—独轮手推车 wheelbarrow.c */
#include <gtk/gtk.h>
/* XPM */
static char * WheelbarrowFull_xpm[] = {
"48 48 64 1",
"       c None",
".      c #DF7DCF3CC71B",
"X      c #965875D669A6",
"o      c #71C671C671C6",
"O      c #A699A289A699",
"+      c #965892489658",
"@      c #8E38410330C2",
```

```
"#      c #D75C7DF769A6",
"$      c #F7DECF3CC71B",
"%      c #96588A288E38",
"&      c #A69992489E79",
"*      c #8E3886178E38",
"=      c #104008200820",
"-      c #596510401040",
";      c #C71B30C230C2",
":      c #C71B9A699658",
">      c #618561856185",
",      c #20811C712081",
"<      c #104000000000",
"1      c #861720812081",
"2      c #DF7D4D344103",
"3      c #79E769A671C6",
"4      c #861782078617",
"5      c #41033CF34103",
"6      c #000000000000",
"7      c #49241C711040",
"8      c #492445144924",
"9      c #082008200820",
"0      c #69A618611861",
"q      c #B6DA71C65144",
"w      c #410330C238E3",
"e      c #CF3CBAEAB6DA",
"r      c #71C6451430C2",
"t      c #EFBEDB6CD75C",
"y      c #28A208200820",
"u      c #186110401040",
"i      c #596528A21861",
"p      c #71C661855965",
"a      c #A69996589658",
"s      c #30C228A230C2",
"d      c #BEFBA289AEBA",
"f      c #596545145144",
"g      c #30C230C230C2",
"h      c #8E3882078617",
"j      c #208118612081",
"k      c #38E30C300820",
"l      c #30C2208128A2",
"z      c #38E328A238E3",
"x      c #514438E34924",
"c      c #618555555965",
"v      c #30C2208130C2",
"b      c #38E328A230C2",
"n      c #28A228A228A2",
"m      c #41032CB228A2",
"M      c #104010401040",
"N      c #492438E34103",
"B      c #28A2208128A2",
"V      c #A699596538E3",
"C      c #30C21C711040",
```

```
"Z      c #30C218611040",
"A      c #965865955965",
"S      c #618534D32081",
"D      c #38E31C711040",
"F      c #082000000820",
"                                                ",
"          .XoO                                  ",
"         +@#$%o&                                ",
"         *=-;#::o+                              ",
"           >,<12#:34                            ",
"             45671#:X3                          ",
"               +89<02qwo                        ",
"e*                 >,67;ro                      ",
"ty>                  459@>+&&                   ",
"$2u+                   ><ipas8*                 ",
"%$;=*                  *3:.Xa.dfg>              ",
"Oh$;ya               *3d.a8j,Xe.d3g8+           ",
" Oh$;ka            *3d$a8lz,,xxc:.e3g54         ",
"  Oh$;kO         *pd$%svbzz,sxxxxfX..&wn>       ",
"   Oh$@mO     *3dthwlsslszjzxxxxxxx3:td8M4      ",
"    Oh$@g& *3d$XNlvvvlllm,mNwxxxxxxxfa.:,B*     ",
"     Oh$@,Od.czlllllzlmmqV@V#V@fxxxxxxxf:%j5&   ",
"      Oh$1hd5lllslllCCZrV#r#:#2AxxxxxxxxxcdwM*  ",
"       OXq6c.%8vvvllZZiqqApA:mq:Xxcpcxxxxxfdc9* ",
"        2r<6gde3bllZZrVi7S@SV77A::qApxxxxxxfdcM ",
"        :,q-6MN.dfmZZrrSS:#riirDSAX@Af5xxxxxfevo",
"         +A26jguXtAZZZC7iDiCCrVVii7Cmmmxxxxxx%3g",
"          *#16jszN..3DZZZZrCVSA2rZrV7Dmmwxxxx&en",
"           p2yFvzssXe:fCZZCiiD7iiZDiDSSZwwxx8e*>",
"           OA1<jzxwwc:$d%NDZZZZCCCZCCZZCmxxfd.B ",
"            3206Bwxxszx%et.eaAp77m77mmmf3&eeeg* ",
"             @26MvzxNzvlbwfpdettttttttttt.c,n&  ",
"             *;16=lsNwwNwgsvslbwwvccc3pcfu<o    ",
"              p;<69Bvwwsszslllbllllllllu<5+     ",
"              OS0y6FBlvvvzvzss,u=Blllj=54       ",
"               c1-699Blvllllllu7k96MMMg4         ",
"               *10y8n6FjvllllB<166668           ",
"                S-kg+>666<M<996-y6n<8*           ",
"                p71=4 m69996kD8Z-66698&&         ",
"                &i0ycm6n4 ogk17,0<6666g          ",
"                 N-k-<>     >=01-kuu666>         ",
"                 ,6ky&      &46-10ul,66,         ",
"                 Ou0<>       o66y<ulw<66&        ",
"                  *kk5        >66By7=xu664       ",
"                   <<M4       466lj<Mxu66o       ",
"                   *>>        +66uv,zN666*       ",
"                               566,xxj669        ",
"                               4666FF666>        ",
"                                >966666M         ",
"                                 oM6668+         ",
"                                   *4            ",
```

```
"                                                    ",
"                                                    "};
/* 调用这个函数时(通过delete_event信号)，终止应用程序*/
void close_application( GtkWidget *widget,
GdkEvent *event, gpointer data )
{
    gtk_main_quit();
}
int main (int argc, char *argv[])
{
    /* 构件的存储类型是GtkWidget */
    GtkWidget *window, *pixmap, *fixed;
    GdkPixmap *gdk_pixmap;
    GdkBitmap *mask;
    GtkStyle *style;
    GdkGC *gc;
    /* 创建主窗口，为delete_event信号设置回调函数以终止应用程序。
     * 注意，主窗口没有标题条，因为我们让它是一个弹出窗口 */
    gtk_init (&argc, &argv);
    window = gtk_window_new( GTK_WINDOW_POPUP );
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (close_application), NULL);
    gtk_widget_show (window);
    /* 下面创建pixmap图片和pixmap构件*/
    style = gtk_widget_get_default_style();
    gc = style->black_gc;
    gdk_pixmap = gdk_pixmap_create_from_xpm_d( window->window, &mask,
                                        &style->bg[GTK_STATE_NORMAL],
                                        WheelbarrowFull_xpm );
    pixmap = gtk_pixmap_new( gdk_pixmap, mask );
    gtk_widget_show( pixmap );
    /* 要显示pixmap图片，我们将pixmap构件放在GtkFixed构件上*/
    fixed = gtk_fixed_new();
    gtk_widget_set_usize( fixed, 200, 200 );
    gtk_fixed_put( GTK_FIXED(fixed), pixmap, 0, 0 );
    gtk_container_add( GTK_CONTAINER(window), fixed );
    gtk_widget_show( fixed );
    /* 下面的函数屏蔽了除图片本身以外的所有东西 */
    gtk_widget_shape_combine_mask( window, mask, 0, 0 );
    /* 显示窗口 */
    gtk_widget_set_uposition( window, 20, 400 );
    gtk_widget_show( window );
    gtk_main ();
    return(0);
  }
/* 示例结束 */
```

要想让“手推车”图片敏感（对点击响应），可以给窗口连接一个button_press信号，当我们点击窗口时，就会调用一个回调函数。下面几行代码让图片敏感，当鼠标按下时终止应用程序。

```
    gtk_widget_set_events( window,
                           gtk_widget_get_events( window ) |
```

```
                    GDK_BUTTON_PRESS_MASK );
    gtk_signal_connect( GTK_OBJECT(window), "button_press_event",
                        GTK_SIGNAL_FUNC(close_application), NULL );
```

9.7 标尺构件GtkRuler

GtkRuler(标尺构件)一般用于在给定窗口中指示鼠标指针的位置。一个窗口可以有一个横跨整个窗口宽度的水平标尺和一个占据整个窗口高度的垂直标尺。标尺上有一个小三角形的指示器标出鼠标指针相对于标尺的精确位置。

有两种标尺构件：GtkHRuler（水平）和GtkVRuler（垂直）。

首先，必须创建标尺构件。水平和垂直标尺用下面的函数创建：

```
GtkWidget *gtk_hruler_new(void);水平标尺 */
GtkWidget *gtk_vruler_new(void);垂直标尺 */
```

一旦创建了标尺，我们就能指定它的度量单位。标尺的度量单位可以是 beGTK_PIXELS, GTK_INCHES或 GTK_CENTIMETERS。可以用下面的函数设置：

```
void gtk_ruler_set_metric( GtkRuler      *ruler,
                           GtkMetricType  metric );
```

默认的度量单位是GTK_PIXELS。

```
gtk_ruler_set_metric( GTK_RULER(ruler), GTK_PIXELS );
```

标尺构件的另一个重要属性是怎样标志刻度单位以及位置指示器一开始应该放在哪里。可以用下面的函数设置：

```
void gtk_ruler_set_range( GtkRuler *ruler,
                          gfloat    lower,
                          gfloat    upper,
                          gfloat    position,
                          gfloat    max_size );
```

其中lower和upper参数定义标尺的范围，max_size是要显示的最大可能数值。第四个参数position定义了标尺的指针指示器的初始位置。

垂直标尺能跨越800像素宽的窗口，因此：

```
    gtk_ruler_set_range( GTK_RULER(vruler), 0, 800, 0, 800);
```

标尺上显示标志会从0到800，每100个像素一个数字。如果我们想让标尺的范围从 7到16，可以使用下面的代码：

```
gtk_ruler_set_range( GTK_RULER(vruler), 7, 16, 0, 20);
```

标尺上的指示器是一个小三角形的标记，指示鼠标指针相对于标尺的位置。如果标尺是用于跟踪鼠标器指针的，应该将 motion_notify_event信号连接到标尺的motion_notify_event方法。要跟踪鼠标在整个窗口区域内的移动，应该这样做：

```
#define EVENT_METHOD(i, x) GTK_WIDGET_CLASS(GTK_OBJECT(i)->klass)->x
    gtk_signal_connect_object( GTK_OBJECT(area), "motion_notify_event",
          (GtkSignalFunc)EVENT_METHOD(ruler, motion_notify_event),
          GTK_OBJECT(ruler) );
```

下列例子创建一个绘图区，上面加一个水平标尺，左边加一个垂直标尺。绘图区的大小是 600像素宽×400像素高。水平标尺范围是从7到13，每100像素加一个刻度；垂直标尺范围从0到400，每100像素加一个刻度。

绘图区和标尺的定位是用表格构件（GtkTable）实现的。

```
/* 标尺示例开始 rulers.c */

#include <gtk/gtk.h>

#define EVENT_METHOD(i, x) GTK_WIDGET_CLASS(GTK_OBJECT(i)->klass)->x

#define XSIZE  600
#define YSIZE  400

/* 当点击"close"按钮时，退出应用程序*/
void close_application( GtkWidget *widget,
                        GdkEvent *event, gpointer data )
{
    gtk_main_quit();
}

/* 主函数 */
int main( int argc, char *argv[] ) {
    GtkWidget *window, *table, *area, *hrule, *vrule;

    /* 初始化GTK，创建主窗口*/
    gtk_init( &argc, &argv );

    window = gtk_window_new( GTK_WINDOW_TOPLEVEL );
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
           GTK_SIGNAL_FUNC( close_application ), NULL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* 创建一个GtkTable构件，标尺和绘图区放在里面*/
    table = gtk_table_new( 3, 2, FALSE );
    gtk_container_add( GTK_CONTAINER(window), table );

    area = gtk_drawing_area_new();
    gtk_drawing_area_size( (GtkDrawingArea *)area, XSIZE, YSIZE );
    gtk_table_attach( GTK_TABLE(table), area, 1, 2, 1, 2,
                    GTK_EXPAND|GTK_FILL, GTK_FILL, 0, 0 );
    gtk_widget_set_events( area, GDK_POINTER_MOTION_MASK |
                           GDK_POINTER_MOTION_HINT_MASK );

    /* 水平标尺放在顶部。鼠标移动穿过绘图区时，一个
     * motion_notify_event被传递给标尺的相应的事件处理函数*/
    hrule = gtk_hruler_new();
    gtk_ruler_set_metric( GTK_RULER(hrule), GTK_PIXELS );
    gtk_ruler_set_range( GTK_RULER(hrule), 7, 13, 0, 20 );
    gtk_signal_connect_object( GTK_OBJECT(area), "motion_notify_event",
                               (GtkSignalFunc)EVENT_METHOD(hrule,
                               motion_notify_event),
                               GTK_OBJECT(hrule) );
    /*  GTK_WIDGET_CLASS(GTK_OBJECT(hrule)->klass)->motion_notify_event, */
```

```
    gtk_table_attach( GTK_TABLE(table), hrule, 1, 2, 0, 1,
                      GTK_EXPAND|GTK_SHRINK|GTK_FILL, GTK_FILL, 0, 0 );
    /* 垂直标尺显示在左边。当鼠标移动穿过绘图区时,
     * motion_notify_event被传递到标尺相应的事件处理函数中 */
    vrule = gtk_vruler_new();
    gtk_ruler_set_metric( GTK_RULER(vrule), GTK_PIXELS );
    gtk_ruler_set_range( GTK_RULER(vrule), 0, YSIZE, 10, YSIZE );
    gtk_signal_connect_object( GTK_OBJECT(area), "motion_notify_event",
                        (GtkSignalFunc)
                         GTK_WIDGET_CLASS(GTK_OBJECT(vrule)->klass)->
                         motion_notify_event,
                         GTK_OBJECT(vrule) );
    gtk_table_attach( GTK_TABLE(table), vrule, 0, 1, 1, 2,
                      GTK_FILL, GTK_EXPAND|GTK_SHRINK|GTK_FILL, 0, 0 );

    /* 现在显示所有的构件*/
    gtk_widget_show( area );
    gtk_widget_show( hrule );
    gtk_widget_show( vrule );
    gtk_widget_show( table );
    gtk_widget_show( window );
    gtk_main();

    return(0);
} /*示例结束 */
```

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

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

编译，运行结果如图9-4所示。当鼠标在绘图区移动时，水平和垂直标尺上都能清楚地显示鼠标的位置。

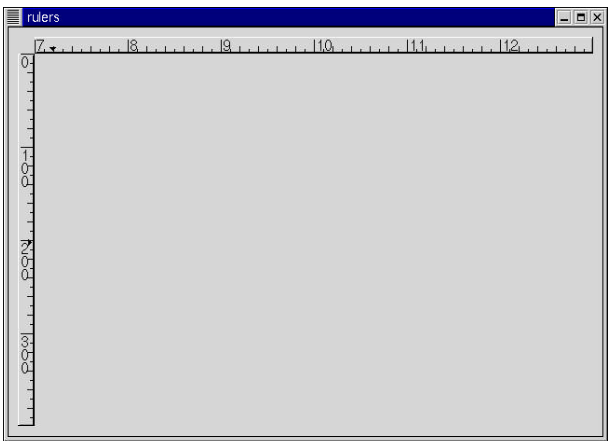

图9-4 标尺示例

9.8 文本输入构件GtkEntry

GtkEntry(文本输入构件)允许在一个单行文本框里输入和显示一行文本。文本可以用函数进行操作，如将新的文本替换、前插、追加到文本输入构件的当前内容中。

有两个用于创建文本输入构件的函数：

```
GtkWidget *gtk_entry_new(void);
GtkWidget *gtk_entry_new_with_max_length( guint16 max);
```

第一个函数创建新的文本输入构件，第二个函数创建新的最大文本长度为 max的文本输入构件。

有几个函数是用来改变构件内的文本内容的：

```
void gtk_entry_set_text( GtkEntry    *entry,
                         const gchar *text );
void gtk_entry_append_text( GtkEntry    *entry,
                            const gchar *text );
void gtk_entry_prepend_text( GtkEntry    *entry,
                             const gchar *text
```

函数gtk_entry_set_text设置文本输入构件内的文本内容目录。

函数gtk_entry_append_text和gtk_entry_prepend_text将新文本前插和追加到构件当前文本中。

下面的函数用于设置当前插入点。

```
void gtk_entry_set_position( GtkEntry *entry,
                             gint      position );
```

文本输入构件内的文本可以用下面的函数获取。这在下面介绍的回调函数中是很有用的。

```
gchar *gtk_entry_get_text( GtkEntry *entry );
```

函数返回的值在函数内部使用，不必用 free()或者g_free()释放。如果想让文本输入构件是只读的，可以改变它的可编辑状态。

```
void gtk_entry_set_editable( GtkEntry *entry,
                             gboolean  editable );
```

其中editable可设为TRUE（可编辑）或FALSE（不可编辑）。

如果想让文本输入构件输入的文本不回显（比如用于接收口令），可以使用下列函数，其中visible可以取为TRUE（显示）或FALSE（不显示）。

```
void gtk_entry_set_visibility( GtkEntry *entry,
                               gboolean  visible );
```

文本内的某一部分可以用下面的函数设置为选中。一般用于用文本输入构件为用户提供一个缺省值，这样当构件获得焦点时就可以全部选中其中的文本，用户的输入会直接替换全部文本。

```
void gtk_entry_select_region( GtkEntry *entry,
                              gint      start,
                              gint      end );
```

如果我们想在用户输入文本时进行响应，可以为 activate或者changed信号设置回调函数。当用户在文本输入构件内部按回车键时引发 activate信号；当文本构件内部的文本发生变化时引发changed信号。

下面的代码是一个使用文本输入构件的示例。

```
/* 文本输入构件示例开始 entry.c */
#include <gtk/gtk.h>
void enter_callback(GtkWidget *widget, GtkWidget *entry)
{
  gchar *entry_text;
  entry_text = gtk_entry_get_text(GTK_ENTRY(entry));
  printf("Entry contents: %s\n", entry_text);
}

void entry_toggle_editable (GtkWidget *checkbutton,
                            GtkWidget *entry)
{
  gtk_entry_set_editable(GTK_ENTRY(entry),
                        GTK_TOGGLE_BUTTON(checkbutton)->active);
}

void entry_toggle_visibility (GtkWidget *checkbutton,
                            GtkWidget *entry)
{
  gtk_entry_set_visibility(GTK_ENTRY(entry),
                        GTK_TOGGLE_BUTTON(checkbutton)->active);
}

int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *vbox, *hbox;
    GtkWidget *entry;
    GtkWidget *button;
    GtkWidget *check;
    gtk_init (&argc, &argv);

    /* 创建主窗口*/
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_widget_set_usize( GTK_WIDGET (window), 200, 100);
    gtk_window_set_title(GTK_WINDOW (window), "GTK Entry");
    gtk_signal_connect(GTK_OBJECT (window), "delete_event",
                    (GtkSignalFunc) gtk_exit, NULL);
    vbox = gtk_vbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (vbox);
    entry = gtk_entry_new_with_max_length (50);
    gtk_signal_connect(GTK_OBJECT(entry), "activate",
                    GTK_SIGNAL_FUNC(enter_callback),
                    entry);
    gtk_entry_set_text (GTK_ENTRY (entry), "hello");
    gtk_entry_append_text (GTK_ENTRY (entry), " world");
    gtk_entry_select_region (GTK_ENTRY (entry),
                          0, GTK_ENTRY(entry)->text_length);
```

```
    gtk_box_pack_start (GTK_BOX (vbox), entry, TRUE, TRUE, 0);
    gtk_widget_show (entry);
    hbox = gtk_hbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (vbox), hbox);
    gtk_widget_show (hbox);

    check = gtk_check_button_new_with_label("Editable");
    gtk_box_pack_start (GTK_BOX (hbox), check, TRUE, TRUE, 0);
    gtk_signal_connect (GTK_OBJECT(check), "toggled",
                        GTK_SIGNAL_FUNC(entry_toggle_editable),
                        entry);
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
    gtk_widget_show (check);
    check = gtk_check_button_new_with_label("Visible");
    gtk_box_pack_start (GTK_BOX (hbox), check, TRUE, TRUE, 0);
    gtk_signal_connect (GTK_OBJECT(check), "toggled",
                        GTK_SIGNAL_FUNC(entry_toggle_visibility),
                        entry);
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
    gtk_widget_show (check);
    button = gtk_button_new_with_label ("Close");
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                               GTK_SIGNAL_FUNC(gtk_exit),
                               GTK_OBJECT (window));

    gtk_box_pack_start (GTK_BOX (vbox), button, TRUE, TRUE, 0);
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
    gtk_widget_grab_default (button);
    gtk_widget_show (button);
    gtk_widget_show(window);
    gtk_main();

    return(0);
}

 /*示例结束 */
```

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

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

编译应用程序，然后在shell提示符下输入./entry，执行结果如图9-5所示。当Editable检查按钮是按下状态时，可以编辑文本输入构件内的文本；如果该检查按钮是弹起的，不能对文本输入构件的文本进行编辑。Visible检查按钮是按下状态时，用户输入的字符会显示在文本输入构件上，否则不显示出来。

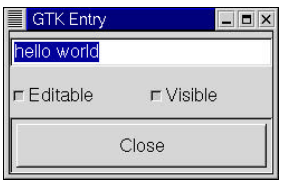

图9-5 文本输入构件GtkEntry

9.9 微调按钮构件GtkSpinButton

GtkSpinButton(微调按钮构件)通常用于让用户从一个取值范围内选择一个值。它由一个文本输入框和旁边的向上和向下两个按钮组成。点击某一个按钮会让文本输入框内的数值大小在一定范围内改变。文本输入框也可以直接进行编辑。

微调按钮构件允许其中的数值没有小数位或具有指定的小数位，并且数值可以按一种可配置的方式增加或减小。在按钮较长时间呈按下状态时，构件的数值会根据工具按下时间的长短加速变化。

微调按钮用一个调整对象来维护该按钮能够取值的范围。微调按钮构件因此而具有了很强大的功能。

下面是创建调整对象的函数。这里的用意是展示其中所包含的数值的意义：

```
GtkObject *gtk_adjustment_new( gfloat value,
                               gfloat lower,
                               gfloat upper,
                               gfloat step_increment,
                               gfloat page_increment,
                               gfloat page_size );
```

调整对象的这些属性在微调按钮构件中有如下用处：

value: 微调按钮构件的初值。

lower: 构件允许的最小值。

upper: 构件允许的最大值。

step_increment: 当鼠标左键按下时构件一次增加/减小的值。

page_increment: 当鼠标右键按下时构件一次增加/减小的值。

page_size: 没有用到。

另外，当用鼠标中间键点击向下或向上按钮时，可以直接跳到构件的 lower或upper值。

用下面的函数创建新微调按钮构件：

```
GtkWidget *gtk_spin_button_new( GtkAdjustment *adjustment,
                                gfloat         climb_rate,
                                guint          digits );
```

其中的climb_rate参数是介于0.0和1.0间的值，指明构件数值变化的加速度（长时间按住按钮，数值会加速变化）。第三个参数digits指定要显示的值的小数位数。

创建微调按钮构件之后，还可以用下面的函数对其重新配置：

```
void gtk_spin_button_configure( GtkSpinButton *spin_button,
                                GtkAdjustment *adjustment,
                                gfloat         climb_rate,
                                guint          digits );
```

其中spin_button参数就是要重新配置的参数。其他的参数与创建时的参数意思相同。

使用下面的函数可以设置或获取构件内部使用的调整对象：

```
void gtk_spin_button_set_adjustment( GtkSpinButton  *spin_button,
                                     GtkAdjustment  *adjustment );

GtkAdjustment adjustment(
                                      GtkSpinButton *spin_button );
```

显示数值的小数位数可以用下面的函数改变，其中 digits就是小数位数：

```
void gtk_spin_button_set_digits( GtkSpinButton *spin_button,
                                 guint          digits) ;
```

当前显示的构件数值可以用下面的函数改变：

```
void gtk_spin_button_set_value( GtkSpinButton *spin_button,
                                gfloat         value );
```

微调按钮构件的当前值可以以整数或浮点数的形式获得。使用下面的函数：

```
gfloat gtk_spin_button_get_value_as_float( GtkSpinButton *spin_button );
gint gtk_spin_button_get_value_as_int( GtkSpinButton *spin_button );
```

如果想以当前值为基数改变构件的值，可以使用下面的函数：

```
 void gtk_spin_button_spin( GtkSpinButton *spin_button,
                            GtkSpinType    direction,
                            gfloat         increment );
```

其中，direction参数可以取下面的值：

GTK_SPIN_STEP_FORWARD

GTK_SPIN_STEP_BACKWARD

GTK_SPIN_PAGE_FORWARD

GTK_SPIN_PAGE_BACKWARD

GTK_SPIN_HOME

GTK_SPIN_END

GTK_SPIN_USER_DEFINED

这个函数中包含的一些功能将在下面详细介绍。其中的许多设置都使用了与微调按钮构件相关联的调整对象的值。

GTK_SPIN_STEP_FORWARD和GTK_SPIN_STEP_BACKWARD将构件的值按increment参数指定的数值增大或减小，除非 increment参数是0。这种情况下，构件的值将按与其相关联的调整对象的step_increment值改变。

GTK_SPIN_PAGE_FORWARD和GTK_SPIN_PAGE_BACKWARD按increment参数改变微调按钮构件的值。

GTK_SPIN_HOME将构件的值设置为相关联调整对象的范围的最小值。

GTK_SPIN_END 将构件的值设置为相关联调整对象的范围的最大值。

GTK_SPIN_USER_DEFINED 按指定的数值改变构件的数值。

下面介绍影响微调按钮构件的外观和行为的函数。

要介绍的第一个函数就是限制微调按钮构件的文本框只能输入数值。这样就阻止了用户输入任何非法的字符：

```
void gtk_spin_button_set_numeric( GtkSpinButton *spin_button,
                                  gboolean       numeric );
```

可以设置让微调按钮构件在upper和lower之间循环。也就是当达到最大值后再向上调整回到最小值，当达到最小值后再向下调整变为最大值。可以用下面的函数实现，其中 wrap可以设置为TRUE和FALSE：

```
void gtk_spin_button_set_wrap( GtkSpinButton *spin_button,
                               gboolean       wrap );
```

可以设置让微调按钮构件将其值圆整到最接近 step_increment的值（在该微调按钮构件使用的调整对象中设置的）。用下面的函数实现：

```
void gtk_spin_button_set_snap_to_ticks( GtkSpinButton  *spin_button,
                                        gboolean        snap_to_ticks );
```

微调按钮构件的更新方式可以用下面的函数改变：

```
void gtk_spin_button_set_update_policy( GtkSpinButton  *spin_button,
                                  GtkSpinButtonUpdatePolicy policy );
```

其中policy参数可以取GTK_UPDATE_ALWAYS或GTK_UPDATE_IF_VALID。

这些更新方式影响微调按钮构件在解析插入文本并将其值与调整对象的值同步时的行为。

在policy值为GTK_UPDATE_IF_VALID时，微调按钮构件只有在输入文本是其中调整对象指定范围内合法的值时才进行更新，否则文本会被重置为当前的值。

在policy值为GTK_UPDATE_ALWAYS时，在将文本转换为数值时忽略错误。

构件中使用的向上和向下的按钮的外观可以用下面的函数改变：

```
void gtk_spin_button_set_shadow_type( GtkSpinButton *spin_button,
                                      GtkShadowType  shadow_type );
```

与其他构件一样，shadow_type参数可以取以下的值：

GTK_SHADOW_IN

GTK_SHADOW_OUT

GTK_SHADOW_ETCHED_IN

GTK_SHADOW_ETCHED_OUT

最后，可以强行要求微调按钮构件更新自己：

```
void gtk_spin_button_update( GtkSpinButton  *spin_button );
```

下面是一个使用微调按钮构件的示例。

```
/* 微调按钮构件示例开始 spinbutton.c */

#include <gtk/gtk.h>

static GtkWidget *spinner1;

void toggle_snap( GtkWidget     *widget,
                  GtkSpinButton *spin )
{
  gtk_spin_button_set_snap_to_ticks (spin,
               GTK_TOGGLE_BUTTON (widget)->active);
}

void toggle_numeric( GtkWidget *widget,
                     GtkSpinButton *spin )
{
  gtk_spin_button_set_numeric (spin,
                  GTK_TOGGLE_BUTTON (widget)->active);
}

void change_digits( GtkWidget *widget,
                    GtkSpinButton *spin )
```

```
{
  gtk_spin_button_set_digits (GTK_SPIN_BUTTON (spinner1),
                              gtk_spin_button_get_value_as_int (spin));
}

void get_value( GtkWidget *widget,
                gpointer data )
{
  gchar buf[32];
  GtkLabel *label;
  GtkSpinButton *spin;

  spin = GTK_SPIN_BUTTON (spinner1);
  label = GTK_LABEL (gtk_object_get_user_data (GTK_OBJECT (widget)));
  if (GPOINTER_TO_INT (data) == 1)
    sprintf (buf, "%d", gtk_spin_button_get_value_as_int (spin));
  else
    sprintf (buf, "%0.*f", spin->digits,
             gtk_spin_button_get_value_as_float (spin));
  gtk_label_set_text (label, buf);
}


int main( int   argc,
          char *argv[] )
{
  GtkWidget *window;
  GtkWidget *frame;
  GtkWidget *hbox;
  GtkWidget *main_vbox;
  GtkWidget *vbox;
  GtkWidget *vbox2;
  GtkWidget *spinner2;
  GtkWidget *spinner;
  GtkWidget *button;
  GtkWidget *label;
  GtkWidget *val_label;
  GtkAdjustment *adj;

  /* 初始化GTK */
  gtk_init(&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC (gtk_main_quit),
                      NULL);

  gtk_window_set_title (GTK_WINDOW (window), "Spin Button");

  main_vbox = gtk_vbox_new (FALSE, 5);
  gtk_container_set_border_width (GTK_CONTAINER (main_vbox), 10);
```

```
gtk_container_add (GTK_CONTAINER (window), main_vbox);

frame = gtk_frame_new ("Not accelerated");
gtk_box_pack_start (GTK_BOX (main_vbox), frame, TRUE, TRUE, 0);

vbox = gtk_vbox_new (FALSE, 0);
gtk_container_set_border_width (GTK_CONTAINER (vbox), 5);
gtk_container_add (GTK_CONTAINER (frame), vbox);

/* 年、月、日微调器 */

hbox = gtk_hbox_new (FALSE, 0);
gtk_box_pack_start (GTK_BOX (vbox), hbox, TRUE, TRUE, 5);

vbox2 = gtk_vbox_new (FALSE, 0);
gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

label = gtk_label_new ("Day :");
gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

adj = (GtkAdjustment *) gtk_adjustment_new (1.0, 1.0, 31.0, 1.0,
                                            5.0, 0.0);
spinner = gtk_spin_button_new (adj, 0, 0);
gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner), TRUE);
gtk_spin_button_set_shadow_type (GTK_SPIN_BUTTON (spinner),
                                 GTK_SHADOW_OUT);
gtk_box_pack_start (GTK_BOX (vbox2), spinner, FALSE, TRUE, 0);

vbox2 = gtk_vbox_new (FALSE, 0);
gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

label = gtk_label_new ("Month :");
gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

adj = (GtkAdjustment *) gtk_adjustment_new (1.0, 1.0, 12.0, 1.0,
                                            5.0, 0.0);
spinner = gtk_spin_button_new (adj, 0, 0);
gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner), TRUE);
gtk_spin_button_set_shadow_type (GTK_SPIN_BUTTON (spinner),
                                 GTK_SHADOW_ETCHED_IN);
gtk_box_pack_start (GTK_BOX (vbox2), spinner, FALSE, TRUE, 0);

vbox2 = gtk_vbox_new (FALSE, 0);
gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

label = gtk_label_new ("Year :");
gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);
adj = (GtkAdjustment *) gtk_adjustment_new (1998.0, 0.0, 2100.0,
                                            1.0, 100.0, 0.0);
```

```
spinner = gtk_spin_button_new (adj, 0, 0);
gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner), FALSE);
gtk_spin_button_set_shadow_type (GTK_SPIN_BUTTON (spinner),
                                 GTK_SHADOW_IN);
gtk_widget_set_usize (spinner, 55, 0);
gtk_box_pack_start (GTK_BOX (vbox2), spinner, FALSE, TRUE, 0);

frame = gtk_frame_new ("Accelerated");
gtk_box_pack_start (GTK_BOX (main_vbox), frame, TRUE, TRUE, 0);

vbox = gtk_vbox_new (FALSE, 0);
gtk_container_set_border_width (GTK_CONTAINER (vbox), 5);
gtk_container_add (GTK_CONTAINER (frame), vbox);

hbox = gtk_hbox_new (FALSE, 0);
gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, TRUE, 5);

vbox2 = gtk_vbox_new (FALSE, 0);
gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

label = gtk_label_new ("Value :");
gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

adj = (GtkAdjustment *) gtk_adjustment_new (0.0, -10000.0, 10000.0,
                                            0.5, 100.0, 0.0);
spinner1 = gtk_spin_button_new (adj, 1.0, 2);
gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner1), TRUE);
gtk_widget_set_usize (spinner1, 100, 0);
gtk_box_pack_start (GTK_BOX (vbox2), spinner1, FALSE, TRUE, 0);

vbox2 = gtk_vbox_new (FALSE, 0);
gtk_box_pack_start (GTK_BOX (hbox), vbox2, TRUE, TRUE, 5);

label = gtk_label_new ("Digits :");
gtk_misc_set_alignment (GTK_MISC (label), 0, 0.5);
gtk_box_pack_start (GTK_BOX (vbox2), label, FALSE, TRUE, 0);

adj = (GtkAdjustment *) gtk_adjustment_new (2, 1, 5, 1, 1, 0);
spinner2 = gtk_spin_button_new (adj, 0.0, 0);
gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinner2), TRUE);
gtk_signal_connect (GTK_OBJECT (adj), "value_changed",
                    GTK_SIGNAL_FUNC (change_digits),
                    (gpointer) spinner2);
gtk_box_pack_start (GTK_BOX (vbox2), spinner2, FALSE, TRUE, 0);

hbox = gtk_hbox_new (FALSE, 0);
gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, TRUE, 5);

button = gtk_check_button_new_with_label ("Snap to 0.5-ticks");
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (toggle_snap),
```

```
                    spinner1);
    gtk_box_pack_start (GTK_BOX (vbox), button, TRUE, TRUE, 0);
    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button), TRUE);

    button = gtk_check_button_new_with_label ("Numeric only input mode");
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (toggle_numeric),
                    spinner1);
    gtk_box_pack_start (GTK_BOX (vbox), button, TRUE, TRUE, 0);
    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button), TRUE);

    val_label = gtk_label_new ("");

    hbox = gtk_hbox_new (FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, TRUE, 5);
    button = gtk_button_new_with_label ("Value as Int");
    gtk_object_set_user_data (GTK_OBJECT (button), val_label);
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (get_value),
                    GINT_TO_POINTER (1));
    gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 5);

    button = gtk_button_new_with_label ("Value as Float");
    gtk_object_set_user_data (GTK_OBJECT (button), val_label);
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (get_value),
                    GINT_TO_POINTER (2));
    gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 5);

    gtk_box_pack_start (GTK_BOX (vbox), val_label, TRUE, TRUE, 0);
    gtk_label_set_text (GTK_LABEL (val_label), "0");
    hbox = gtk_hbox_new (FALSE, 0);
    gtk_box_pack_start (GTK_BOX (main_vbox), hbox, FALSE, TRUE, 0);
    button = gtk_button_new_with_label ("Close");
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                           GTK_SIGNAL_FUNC (gtk_widget_destroy),
                           GTK_OBJECT (window));
    gtk_box_pack_start (GTK_BOX (hbox), button, TRU
5);
    gtk_widget_show_all (window);  /* Enter the e
*/  gtk_main ();
    return(0);}

  /*示例结束 */
```

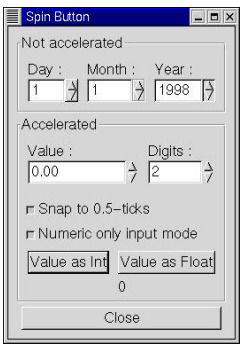

图9-6 微调按钮构件示例

图9-6是上面代码的运行结果。点击微调按钮右边的向上和向下箭头，前面的文本框的数值会随之而改变。

9.10 组合框GtkCombo

GtkCombo(组合框)是极为常见的构件，实际上它仅仅是其他构件的集合。从用户的观点来说，这个构件是由一个文本输入构件和一个下拉菜单组成的，用户可以从一个预先定义的

列表里面选择一个选项，同时，用户也可以直接在文本框里面输入文本。

下面是从定义组合框构件的结构里面摘取出来的，从中可以看到组合框构件是由什么构件组合形成的：

```
struct _GtkCombo
       GtkHBox hbox;
       GtkWidget *entry;
       GtkWidget *button;
       GtkWidget *popup;
       GtkWidget *popwin;
       GtkWidget *list;
       ...  };
```

可以看到，组合框构件有两个主要部分：一个输入框和一个列表。

用下面的函数创建组合框构件：

```
GtkWidget *gtk_combo_new( void );
```

现在，如果想设置显示在输入框部分中的字符串，可以直接操纵组合框构件内部的文本输入构件：

```
gtk_entry_set_text(GTK_ENTRY(GTK_COMBO(combo)->entry), "My String.");
```

要设置下拉列表中的值，可以使用下面的函数：

```
void gtk_combo_set_popdown_strings( GtkCombo *combo,
                                    GList    *strings );
```

在使用这个函数之前，先得将要添加的字符串组合成一个 GList链表。Glist是一个双向链表，是Glib的一部分。要做的就是设置一个Glist指针，其值设为NULL，然后用下面的函数将字符串追加到链表当中：

```
GList *g_list_append( GList *glist,
                      gpointer data );
```

要注意的是：一定要将Glist链表的初值设为NULL，必须将g_list_append函数返回的值赋给要操作的链表本身。

下面是一段典型的代码，用于创建一个选项列表：

```
   GList *glist=NULL;

   glist = g_list_append(glist, "String 1");
   glist = g_list_append(glist, "String 2");
   glist = g_list_append(glist, "String 3");
   glist = g_list_append(glist, "String 4");

   gtk_combo_set_popdown_strings( GTK_COMBO(combo), glist) ;
```

到这里为止，你现在已经可以使用设置的组合框构件了。有几个行为是可以改变的。下面是相关的函数：

```
void gtk_combo_set_use_arrows( GtkCombo *combo,
                               gint      val );

void gtk_combo_set_use_arrows_always( GtkCombo *combo,
                                      gint      val );
void gtk_combo_set_case_sensitive( GtkCombo *combo,
                                   gint      val );
```

gtk_combo_set_use_arrows()让用户用上/下方向键改变文本输入构件内的值。这并没有改变列表的值，只是用列表中的下一个列表项替换了文本输入框中的文本（向上则取上一个值，向下则取下一个值）。这是通过搜索当前项在列表中的位置并选择前一项 /下一项来实现的。通常，在一个输入框中方向键是用来改变焦点的（也可以用 TAB键）。注意，如果当前项是列表的最后一项，按向下的方向键会改变焦点的位置 (这对列表在第一项时按向上方向键也适用)。

如果当前值并不在列表中，则不能使用 gtk_combo_set_use_arrows()函数。

同样地，gtk_combo_set_use_arrows_always()允许使用上/下方向键在下拉列表中选取列表项，但是它在列表项中循环，也就是当列表项位于第一个表项时按向上方向键，会跳到最后一个，当列表项位于最后一个表项时按向下方向键，会跳到第一个。这样可以完全禁止使用方向键改变焦点。

gtk_combo_set_case_sensitive()函数切换GTK是否以大小写敏感的方式搜索其中的列表项。这一般用在内部文本输入构件中的文本查找组合框构件中的列表值。可以将其设置为大小写敏感或不敏感。如果用户同时按下“ Alt ”和“ Tab ”键，组合框构件还可以用来完成当前输入。注意，窗口管理器也要使用这种组合键方式，将会忽略 GTK中这个组合键的使用。

注意，我们使用的是组合框构件，它能够为我们从一个下拉列表中选择一个选项。这一点是很直截了当的。大多数时候，你可能很关心怎样从其中的文本输入构件中获取数据。组合框构件内部的文本输入构件可以用 GTK_ENTRY(GTK_COMBO(combo)->entry)访问。一般想要做的两件主要工作一个是连接一个 activate，当用户按回车键时能够进行响应，另一个就是读出其中的文本。第一件工作可以用下面的方法实现：

```
gtk_signal_connect(GTK_OBJECT(GTK_COMB(combo)->entry), "activate",
                   GTK_SIGNAL_FUNC (my_callback_function), my_data);
```

可以使用下面的函数在任意时候取得文本输入构件中的文本：

```
gchar *gtk_entry_get_text(GtkEntry *entry);
```

具体做法如下：

```
    char *string;
    string = gtk_entry_get_text(GTK_ENTRY(GTK_COMBO(combo)->entry));
```

这就是取得文本输入框中字符串的方法。

```
void gtk_combo_disable_activate(GtkCombo *combo);
```

这个函数禁用组合框构件内部的文本输入构件（ GtkEntry）的activate信号。

9.11 日历构件GtkCalendar

GtkCalendar(日历构件)显示一个月历视图，可以在上面方便地选择年份、月份和日期。这样，如果要做与日期相关的编程，不再需要考虑复杂的历法问题。日历构件本身外观也很漂亮，创建和使用都非常简单。同时，日历构件 GtkCalendar不存在2000年问题。如果要开发一个日程管理等类的软件，这个构件是一个很好的选择。

创建日历构件的方法和其他构件的类似：

```
GtkWidget *gtk_calendar_new();
```

有时候，需要同时对构件的外观和内容做很多的修改。这时候可能会引起构件的多次更新，导致屏幕闪烁。可以在修改之前使用一个函数将构件“ 冻结 ”，然后在修改完成之后再用

一个函数将构件“解冻”。这样，构件在整个过程中只做一次更新。

这个函数将构件“冻结”：

```
void gtk_calendar_freeze( GtkCalendar *Calendar );
```

这个函数将构件“解冻”：

```
void gtk_calendar_thaw  ( GtkCalendar *Calendar );
```

这两个函数和其他构件（比如GtkText）的冻结/解冻函数作用完全一样。

日历构件有几个选项，可以用来改变构件的外观和操作方式。使用下面的函数可以改变这些选项：

```
void gtk_calendar_display_options( GtkCalendar  *calendar,
                                   GtkCalendarDisplayOptions  flags );
```

函数中的flags参数可以将下面的五种选项中的一个或者多个用逻辑位或（|）操作符组合起来：

GTK_CALENDAR_SHOW_HEADING：这个选项指定在绘制日历构件时，应该显示月份和年份。

GTK_CALENDAR_SHOW_DAY_NAMES：这个选项指定用三个字母的缩写显示每一天是星期几（比如MON、TUE等）。

GTK_CALENDAR_NO_MONTH_CHANGE：这个选项指定用户不应该也不能够改变显示的月份。如果只想显示某个特定的月份，则可以使用这个选项。比如，如果在窗口上同时为一年的12个月分别设置一个日历构件时。

GTK_CALENDAR_SHOW_WEEK_NUMBERS：这个选项指定应该在构件的左边显示每一周在全年的周序号（一年是52个周，元月1日是第1周，12月31日是第52周）。

GTK_CALENDAR_WEEK_START_MONDAY：这个选项指定在日历构件中每一周是从星期一开始而不是从星期天开始。缺省设置是从星期天开始。此选项只影响日期在构件中从左到右的排列顺序。

下面的函数用于设置当前要显示的日期：

```
gint gtk_calendar_select_month( GtkCalendar *calendar,
                                guint        month,
                                guint        year );

void gtk_calendar_select_day( GtkCalendar *calendar,
                              guint        day );
```

gtk_calendar_select_month()的返回值是一个布尔值，指示设置是否成功。如果设置一个非法值，比如31月，则会返回一个FALSE值。

使用gtk_calendar_select_day()函数，如果day参数指定的日期是合法的，会在日历构件中选中该日期。如果day参数的值是0，将清除当前的选择。

除了可以选中一个日期以外，在一个月中可以有任意个日期被“标记”。被“标记”的日期会在日历构件中高亮显示。下面的函数用于标记日期和取消标记：

```
gint gtk_calendar_mark_day( GtkCalendar *calendar,
                            guint        day);

gint gtk_calendar_unmark_day( GtkCalendar *calendar,
                              guint        day);
```

```
void gtk_calendar_clear_marks( GtkCalendar *calendar);
```

当前标记的日期存储在一个GtkCalendar结构的数组中。数组的长度是31，这样，要想知道某个特定的日期是否被标记，可以访问数值中相应的元素(注意，在C语言中，数值是从0开始编号的)。例如：

```
GtkCalendar *calendar;
calendar = gtk_calendar_new();
...
/* 当月7日被标记了吗？*/
if (calendar->marked_date[7-1])
   /* 若执行此处的代码，表明7日已经被标记 */
```

注意，在月份和年份变化时，被标记的日期是不会变化的。

下面的函数用于取得当前选中的年/月/日值：

```
void gtk_calendar_get_date( GtkCalendar *calendar,
                            guint       *year,
                            guint       *month,
                            guint       *day );
```

使用这个函数时，需要先声明几个guint类型的变量——传递给函数的year、month和day参数。所需要的返回值就存放在这几个变量中。如果将某一个参数设置为 NULL，则不返回该值。

日历构件能够引发许多信号，用于指示日期被选中以及选择发生的变化。信号的意义很容易理解。信号名称如下：

```
month_changed            选择月份变化*/
day_selected             选择日期变化*/
day_selected_double_click 选中日期并以鼠标双击*/
prev_month               选择前一月*/
next_month               选择下一月*/
prev_year                选择前一年*/
next_year                选择下一年*/
```

下面是一个日历构件的示例，运用了上面介绍的各项特性。

```
/* 日历构件示例开始 calendar.c */
#include <gtk/gtk.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

#define DEF_PAD 10
#define DEF_PAD_SMALL 5

#define TM_YEAR_BASE 1900

typedef struct _CalendarData {
  GtkWidget *flag_checkboxes[5];
  gboolean  settings[5];
  gchar     *font;
  GtkWidget *font_dialog;
  GtkWidget *window;
```

```
  GtkWidget *prev2_sig;
  GtkWidget *prev_sig;
  GtkWidget *last_sig;
  GtkWidget *month;
} CalendarData;

enum {
  calendar_show_header,
  calendar_show_days,
  calendar_month_change,
  calendar_show_week,
  calendar_monday_first
};

/*
 * GtkCalendar日历构件
 */

void calendar_date_to_string( CalendarData *data,
                              char         *buffer,
                              gint          buff_len )
{
  struct tm tm;
  time_t time;

  memset (&tm, 0, sizeof (tm));
  gtk_calendar_get_date (GTK_CALENDAR(data->window),
                         &tm.tm_year, &tm.tm_mon, &tm.tm_mday);
  tm.tm_year -= TM_YEAR_BASE;
  time = mktime(&tm);
  strftime (buffer, buff_len-1, "%x", gmtime(&time));
}

void calendar_set_signal_strings( char         *sig_str,
                                  CalendarData *data)
{
  gchar *prev_sig;

  gtk_label_get (GTK_LABEL (data->prev_sig), &prev_sig);
  gtk_label_set (GTK_LABEL (data->prev2_sig), prev_sig);

  gtk_label_get (GTK_LABEL (data->last_sig), &prev_sig);
  gtk_label_set (GTK_LABEL (data->prev_sig), prev_sig);
  gtk_label_set (GTK_LABEL (data->last_sig), sig_str);
}

void calendar_month_changed( GtkWidget    *widget,
                             CalendarData *data )
{
  char buffer[256] = "month_changed: ";
```

```
  calendar_date_to_string (data, buffer+15, 256-15);
  calendar_set_signal_strings (buffer, data);
}

void calendar_day_selected( GtkWidget    *widget,
                            CalendarData *data )
{
  char buffer[256] = "day_selected: ";

  calendar_date_to_string (data, buffer+14, 256-14);
  calendar_set_signal_strings (buffer, data);
}

void calendar_day_selected_double_click( GtkWidget    *widget,
                                         CalendarData *data )
{
  struct tm tm;
  char buffer[256] = "day_selected_double_click: ";

  calendar_date_to_string (data, buffer+27, 256-27);
  calendar_set_signal_strings (buffer, data);

  memset (&tm, 0, sizeof (tm));
  gtk_calendar_get_date (GTK_CALENDAR(data->window),
                         &tm.tm_year, &tm.tm_mon, &tm.tm_mday);
  tm.tm_year -= TM_YEAR_BASE;

  if(GTK_CALENDAR(data->window)->marked_date[tm.tm_mday-1] == 0) {
    gtk_calendar_mark_day(GTK_CALENDAR(data->window),tm.tm_mday);
  } else
    gtk_calendar_unmark_day(GTK_CALENDAR(data->window),tm.tm_mday);
  }
}

void calendar_prev_month( GtkWidget    *widget,
                          CalendarData *data )
{
  char buffer[256] = "prev_month: ";

  calendar_date_to_string (data, buffer+12, 256-12);
  calendar_set_signal_strings (buffer, data);
}

void calendar_next_month( GtkWidget    *widget,
                          CalendarData *data )
{
  char buffer[256] = "next_month: ";

  calendar_date_to_string (data, buffer+12, 256-12);
  calendar_set_signal_strings (buffer, data);
}
```

```
void calendar_prev_year( GtkWidget    *widget,
                         CalendarData *data )
{
  char buffer[256] = "prev_year: ";

  calendar_date_to_string (data, buffer+11, 256-11);
  calendar_set_signal_strings (buffer, data);
}

void calendar_next_year( GtkWidget    *widget,
                         CalendarData *data )
{
  char buffer[256] = "next_year: ";

  calendar_date_to_string (data, buffer+11, 256-11);
  calendar_set_signal_strings (buffer, data);
}

void calendar_set_flags( CalendarData *calendar )
{
  gint i;
  gint options=0;
  for (i=0;i<5;i++)
    if (calendar->settings[i])
      {
        options=options + (1<<i);
      }
  if (calendar->window)
    gtk_calendar_display_options (GTK_CALENDAR (calendar->window), options);
}

void calendar_toggle_flag( GtkWidget    *toggle,
                           CalendarData *calendar )
{
  gint i;
  gint j;
  j=0;
  for (i=0; i<5; i++)
    if (calendar->flag_checkboxes[i] == toggle)
      j = i;

  calendar->settings[j]=!calendar->settings[j];
  calendar_set_flags(calendar);

}

void calendar_font_selection_ok( GtkWidget    *button,
                                 CalendarData *calendar )
{
  GtkStyle *style;
  GdkFont  *font;
```

```
      calendar->font = gtk_font_selection_dialog_get_font_name(
                          GTK_FONT_SELECTION_DIALOG (calendar->font_dialog));
      if (calendar->window)
        {

font=gtk_font_selection_dialog_get_font(GTK_FONT_SELECTION_DIALOG(calendar-
    >font_dialog));
          if (font)
            {
              style = gtk_style_copy (gtk_widget_get_style (calendar->window));
              gdk_font_unref (style->font);
              style->font = font;
              gdk_font_ref (style->font);
              gtk_widget_set_style (calendar->window, style);
            }
        }
    }

    void calendar_select_font( GtkWidget    *button,
                               CalendarData *calendar )
    {
      GtkWidget *window;

      if (!calendar->font_dialog) {
        window = gtk_font_selection_dialog_new ("Font Selection Dialog");
        g_return_if_fail(GTK_IS_FONT_SELECTION_DIALOG(window));
        calendar->font_dialog = window;

        gtk_window_position (GTK_WINDOW (window), GTK_WIN_POS_MOUSE);

        gtk_signal_connect (GTK_OBJECT (window), "destroy",
                            GTK_SIGNAL_FUNC (gtk_widget_destroyed),
                            &calendar->font_dialog);

       gtk_signal_connect (GTK_OBJECT (GTK_FONT_SELECTION_DIALOG (window)-
>ok_button),
        "clicked", GTK_SIGNAL_FUNC(calendar_font_selection_ok),
                            calendar);
        gtk_signal_connect_object (GTK_OBJECT (GTK_FONT_SELECTION_DIALOG (window)-
>cancel_button),
                                   "clicked",
                                   GTK_SIGNAL_FUNC (gtk_widget_destroy),
                                   GTK_OBJECT (calendar->font_dialog));
      }
      window=calendar->font_dialog;
      if (!GTK_WIDGET_VISIBLE (window))
        gtk_widget_show (window);
      else
        gtk_widget_destroy (window);
    }
    void create_calendar()
```

```
{
  GtkWidget *window;
  GtkWidget *vbox, *vbox2, *vbox3;
  GtkWidget *hbox;
  GtkWidget *hbbox;
  GtkWidget *calendar;
  GtkWidget *toggle;
  GtkWidget *button;
  GtkWidget *frame;
  GtkWidget *separator;
  GtkWidget *label;
  GtkWidget *bbox;
  static CalendarData calendar_data;
  gint i;

  struct {
    char *label;
  } flags[] =
    {
      { "Show Heading" },
      { "Show Day Names" },
      { "No Month Change" },
      { "Show Week Numbers" },
      { "Week Start Monday" }
    };


  calendar_data.window = NULL;
  calendar_data.font = NULL;
  calendar_data.font_dialog = NULL;

  for (i=0; i<5; i++) {
    calendar_data.settings[i]=0;
  }

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "GtkCalendar Example");
  gtk_container_border_width (GTK_CONTAINER (window), 5);
  gtk_signal_connect(GTK_OBJECT(window), "destroy",
                     GTK_SIGNAL_FUNC(gtk_main_quit),
                     NULL);
  gtk_signal_connect(GTK_OBJECT(window), "delete-event",
                     GTK_SIGNAL_FUNC(gtk_false),
                     NULL);

  gtk_window_set_policy(GTK_WINDOW(window), FALSE, FALSE, TRUE);

  vbox = gtk_vbox_new(FALSE, DEF_PAD);
  gtk_container_add (GTK_CONTAINER (window), vbox);
  /*
```

```
 * 顶级窗口，其中包含日历构件，设置flags选项的按钮和设置字体的按钮
 */

hbox = gtk_hbox_new(FALSE, DEF_PAD);
gtk_box_pack_start (GTK_BOX(vbox), hbox, TRUE, TRUE, DEF_PAD);
hbbox = gtk_hbutton_box_new();
gtk_box_pack_start(GTK_BOX(hbox), hbbox, FALSE, FALSE, DEF_PAD);
gtk_button_box_set_layout(GTK_BUTTON_BOX(hbbox), (GTK_BUTTONBOX_SPREAD);
gtk_button_box_set_spacing(GTK_BUTTON_BOX(hbbox), 5);

/* 日历构件 */
frame = gtk_frame_new("Calendar");
gtk_box_pack_start(GTK_BOX(hbbox), frame, FALSE, TRUE, DEF_PAD);
calendar=gtk_calendar_new();
calendar_data.window = calendar;
calendar_set_flags(&calendar_data);
gtk_calendar_mark_day ( GTK_CALENDAR(calendar), 19);
gtk_container_add( GTK_CONTAINER( frame), calendar);
gtk_signal_connect (GTK_OBJECT (calendar), "month_changed",
                    GTK_SIGNAL_FUNC (calendar_month_changed),
                    &calendar_data);
gtk_signal_connect (GTK_OBJECT (calendar), "day_selected",
                    GTK_SIGNAL_FUNC (calendar_day_selected),
                    &calendar_data);
gtk_signal_connect (GTK_OBJECT (calendar), "day_selected_double_click",
                    GTK_SIGNAL_FUNC (calendar_day_selected_double_click),
                    &calendar_data);
gtk_signal_connect (GTK_OBJECT (calendar), "prev_month",
                    GTK_SIGNAL_FUNC (calendar_prev_month),
                    &calendar_data);
gtk_signal_connect (GTK_OBJECT (calendar), "next_month",
                    GTK_SIGNAL_FUNC (calendar_next_month),
                    &calendar_data);
gtk_signal_connect (GTK_OBJECT (calendar), "prev_year",
                    GTK_SIGNAL_FUNC (calendar_prev_year),
                    &calendar_data);
gtk_signal_connect (GTK_OBJECT (calendar), "next_year",
                    GTK_SIGNAL_FUNC (calendar_next_year),
                    &calendar_data);


separator = gtk_vseparator_new ();
gtk_box_pack_start (GTK_BOX (hbox), separator, FALSE, TRUE, 0);

vbox2 = gtk_vbox_new(FALSE, DEF_PAD);
gtk_box_pack_start(GTK_BOX(hbox), vbox2, FALSE, FALSE, DEF_PAD);

frame = gtk_frame_new("Flags");
gtk_box_pack_start(GTK_BOX(vbox2), frame, TRUE, TRUE, DEF_PAD);
vbox3 = gtk_vbox_new(TRUE, DEF_PAD_SMALL);
gtk_container_add(GTK_CONTAINER(frame), vbox3);
```

```
    for (i = 0; i < 5; i++)
      {
        toggle = gtk_check_button_new_with_label(flags[i].label);
        gtk_signal_connect (GTK_OBJECT (toggle),
                            "toggled",
                            GTK_SIGNAL_FUNC(calendar_toggle_flag),
                            &calendar_data);
        gtk_box_pack_start (GTK_BOX (vbox3), toggle, TRUE, TRUE, 0);
        calendar_data.flag_checkboxes[i]=toggle;
      }
    /* 创建一个按钮，用于设置字体 */
    button = gtk_button_new_with_label("Font...");
    gtk_signal_connect (GTK_OBJECT (button),
                        "clicked",
                        GTK_SIGNAL_FUNC(calendar_select_font),
                        &calendar_data);
    gtk_box_pack_start (GTK_BOX (vbox2), button, FALSE, FALSE, 0);

    frame = gtk_frame_new("Signal events");
    gtk_box_pack_start(GTK_BOX(vbox), frame, TRUE, TRUE, DEF_PAD);

    vbox2 = gtk_vbox_new(TRUE, DEF_PAD_SMALL);
    gtk_container_add(GTK_CONTAINER(frame), vbox2);

    hbox = gtk_hbox_new (FALSE, 3);
    gtk_box_pack_start (GTK_BOX (vbox2), hbox, FALSE, TRUE, 0);
    label = gtk_label_new ("Signal:");
    gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, TRUE, 0);
    calendar_data.last_sig = gtk_label_new ("");
    gtk_box_pack_start (GTK_BOX (hbox), calendar_data.last_sig, FALSE, TRUE, 0);

    hbox = gtk_hbox_new (FALSE, 3);
    gtk_box_pack_start (GTK_BOX (vbox2), hbox, FALSE, TRUE, 0);
    label = gtk_label_new ("Previous signal:");
    gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, TRUE, 0);
    calendar_data.prev_sig = gtk_label_new ("");
    gtk_box_pack_start (GTK_BOX (hbox), calendar_data.prev_sig, FALSE, TRUE, 0);

    hbox = gtk_hbox_new (FALSE, 3);
    gtk_box_pack_start (GTK_BOX (vbox2), hbox, FALSE, TRUE, 0);
    label = gtk_label_new ("Second previous signal:");
    gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, TRUE, 0);
    calendar_data.prev2_sig = gtk_label_new ("");
    gtk_box_pack_start (GTK_BOX (hbox), calendar_data.prev2_sig, FALSE, TRUE,
0);

    bbox = gtk_hbutton_box_new ();
    gtk_box_pack_start (GTK_BOX (vbox), bbox, FALSE, FALSE, 0);
    gtk_button_box_set_layout(GTK_BUTTON_BOX(bbox), GTK_BUTTONBOX_END);
    button = gtk_button_new_with_label ("Close");
```

```
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (gtk_main_quit),
                      NULL);
  gtk_container_add (GTK_CONTAINER (bbox), button);
  GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
  gtk_widget_grab_default (button);

  gtk_widget_show_all(window);
}


int main(int   argc,
         char *argv[] )
{
  gtk_set_locale ();
  gtk_init (&argc, &argv);

  create_calendar();

  gtk_main();

  return(0);
}
/* 示例结束 */
```

示例的运行效果如图9-7所示。其中的Flags框架中的几个检查按钮用于设置日历构件的属性。尝试点击这几个检查按钮，看看有什么效果。 Font...按钮用于设置日历构件的字体。

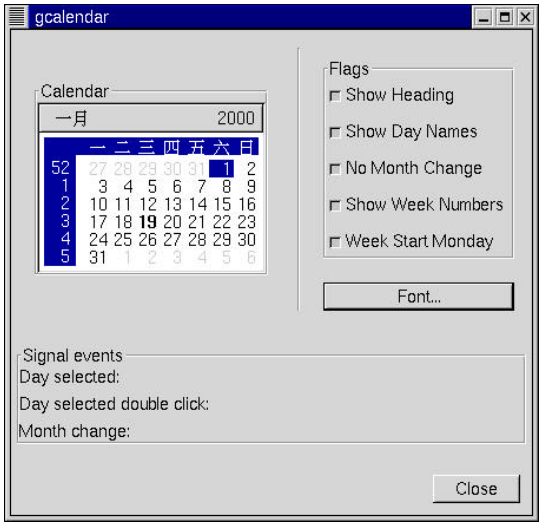

图9-7 日历构件GtkCalendar

9.12 颜色选择构件GtkColorSelect

GtkColorSelect(颜色选择构件)是一个用来交互式地选择颜色的构件。这个组合构件让用户通过操纵RGB值（红绿蓝）和HSV值（色调、饱和度、数值）来选择颜色。这是通过调整

GtkSlider构件的值或者文本输入构件的值，或者从一个色调/饱和度/数值条上选择相应的颜色来实现的。你还可以通过它来设置颜色的透明性。

目前，颜色选择构件只能引发一种信号： color_changed。它是在构件内的颜色值发生变化时，或者当用户通过 gtk_color_selection_set_color()函数显式设置构件的颜色值时引发。

现在可以看一下颜色选择构能够为我们提供一些什么。这个构件有两种风格：

gtk_color_selection和gtk_color_selection_dialog。

```
GtkWidget *gtk_color_selection_new( void );
```

这个函数很少用到。它创建一个孤立的颜色选择构件，并需要将其放在某个窗口上。颜色选择构件是从GtkVBox构件派生的。

```
GtkWidget *gtk_color_selection_dialog_new( const gchar *title );
```

这是最常用的颜色选择构件的构建函数，它创建一个颜色选择对话框。它内部有一个框架构件，框架构件中包含了一个颜色选择构件、一个垂直分隔线构件、一个 GtkHBox构件以及Ok、Cancel、Help三个按钮。你可以通过访问颜色选择对话框构件结构中的 ok_button、cancel_button和help_button构件来访问它们。 例如：

```
GTK_COLOR_SELECTION_DIALOG(colorseldialog)->ok_button)

void gtk_color_selection_set_update_policy( GtkColorSelection *colorsel,
                                            GtkUpdateType      policy );
```

这个函数设置颜色选择构件的更新方式。缺省的更新方式是GTK_UPDATE_CONTINUOUS，这意味着当用户拖动对话框内部的滑块，或者按下鼠标键并在色调/饱和度轮形图或颜色值条上拖动时，当前颜色值将连续更新。如果碰到性能问题，可以将它设置为 GTK_UPDATE_DISCONTINUOUS或GTK_UPDATE_DELAYED。

```
void gtk_color_selection_set_opacity( GtkColorSelection *colorsel,
                                      gint               use_opacity );
```

颜色选择构件支持调整颜色的不透明性 (一般也称为alpha通道)。缺省值是禁用这个特性。调用下面的函数，将 use_opacity设置为TRUE启用该特性。同样， use_opacity设置为FALSE时将禁用此特性。

```
void gtk_color_selection_set_color( GtkColorSelection *colorsel,
                                    gdouble           *color );
```

可以调用这个函数显式地设置颜色选择构件的当前颜色。其中的 color参数是一个指向颜色值（gdouble类型）的数组。数组的长度依赖于是否启用了不透明性。数组的第一位包含红色值， 1是绿色， 2是蓝色， 3是不透明性 (只有不透明性启用时才有，创建前面的gtk_color_selection_set_opacity()函数)。所有的值都介于0.0和1.0之间。

```
void gtk_color_selection_get_color( GtkColorSelection *colorsel,
                                    gdouble           *color );
```

当需要查询当前颜色值时，典型情况是接收到一个 color_changed信号时，使用这个函数。其中，color值是一个指向颜色数组的指针。

下面是一个简单的示例，它演示了如何使用颜色选择对话框构件。这个程序显示了一个包含绘图区的窗口。点击它会打开一个颜色选择对话框，改变颜色选择对话框中的颜色，会

改变绘图区的背景色。

```
/* 颜色选择对话框示例开始colorsel.c */

#include <glib.h>
#include <gdk/gdk.h>
#include <gtk/gtk.h>

GtkWidget *colorseldlg = NULL;
GtkWidget *drawingarea = NULL;

/* 颜色改变信号的处理函数*/
void color_changed_cb (GtkWidget *widget, GtkColorSelection *colorsel)
{
  gdouble color[3];
  GdkColor gdk_color;
  GdkColormap *colormap;

  /* 获得绘图区的色彩表 */
  colormap = gdk_window_get_colormap (drawingarea->window);

  /* 获得当前颜色 */
  gtk_color_selection_get_color (colorsel,color);

  /* 将颜色值转换为16位无符号整数(0..65535)并
   * 插入到GdkColor结构中 */
  gdk_color.red = (guint16)(color[0]*65535.0);
  gdk_color.green = (guint16)(color[1]*65535.0);
  gdk_color.blue = (guint16)(color[2]*65535.0);

  /* 分配颜色 */
  gdk_color_alloc (colormap, &gdk_color);

  /* 设置窗口的背景颜色 */
  gdk_window_set_background (drawingarea->window, &gdk_color);

  /* 清除窗口 */
  gdk_window_clear (drawingarea->window);
}

/* 绘图区事件处理函数*/

gint area_event (GtkWidget *widget, GdkEvent *event, gpointer client_data)
{
  gint handled = FALSE;
  GtkWidget *colorsel;

  /*检查是否接收到一个按键按下事件 */

  if (event->type == GDK_BUTTON_PRESS && colorseldlg == NULL)
    {
```

```
        /* 接收到了一个事件，但是还没有创建颜色选择对话框 */
        handled = TRUE;

        /* 创建颜色选择对话框 */
        colorseldlg = gtk_color_selection_dialog_new("Select background color");

        /* 获取颜色选择构件*/
        colorsel = GTK_COLOR_SELECTION_DIALOG(colorseldlg)->colorsel;

        /* 为"color_changed"信号设置回调函数，将用户数据设置为
         * 颜色选择构件 */
        gtk_signal_connect(GTK_OBJECT(colorsel), "color_changed",
          (GtkSignalFunc)color_changed_cb, (gpointer)colorsel);

        /* 显示对话框 */

        gtk_widget_show(colorseldlg);
      }

  return handled;
}

/* 关闭、退出事件处理函数 */

void destroy_window (GtkWidget *widget, gpointer client_data)
{
  gtk_main_quit ();
}

/* 主函数 */
gint main (gint argc, gchar *argv[])
{
  GtkWidget *window;

  /* 初始化GTK */
  gtk_init (&argc,&argv);

  /* 创建顶级窗口，设置标题，以及窗口是否可缩放 */
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW(window), "Color selection test");
  gtk_window_set_policy (GTK_WINDOW(window), TRUE, TRUE, TRUE);

  /* 为"delete"和"destroy"事件设置回调函数*/
  gtk_signal_connect (GTK_OBJECT(window), "delete_event",
    (GtkSignalFunc)destroy_window, (gpointer)window);

  gtk_signal_connect (GTK_OBJECT(window), "destroy",
    (GtkSignalFunc)destroy_window, (gpointer)window);

  /* 创建绘图区，设置尺寸，捕获鼠标按键事件 */
  drawingarea = gtk_drawing_area_new ();
```

```
    gtk_drawing_area_size (GTK_DRAWING_AREA(drawingarea), 200, 200);
    gtk_widget_set_events (drawingarea, GDK_BUTTON_PRESS_MASK);

    gtk_signal_connect (GTK_OBJECT(drawingarea), "event",
      (GtkSignalFunc)area_event, (gpointer)drawingarea);

    /* 将绘图区添加到窗口中，然后显示它们 */
    gtk_container_add (GTK_CONTAINER(window), drawingarea);

    gtk_widget_show (drawingarea);
    gtk_widget_show (window);

    /* 进入主循环 */
    gtk_main ();
    return(0);
}
/* 示例结束 */
```

上面代码的运行效果如图 9-8所示。在程序窗口上点击，就会弹出一个颜色选择对话框，选择的颜色会实时地显示在窗口中。

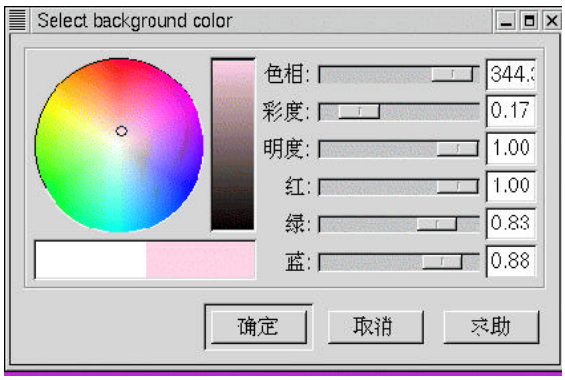

图9-8 颜色选择构件

9.13 文件选择构件GtkFileSelect

GtkFileSelect(文件选择构件)是一种快速、简单的显示文件对话框的方法。它带有“ Ok ”、“ Cancel ”、“ Help ”按钮，可以极大地减少编程时间。

可以用下面的方法创建文件选择构件：

```
GtkWidget *gtk_file_selection_new( gchar *title );
```

要设置文件名，例如，要在打开时指向指定目录，或者给定一个缺省文件名，可以使用下面的函数：

```
void gtk_file_selection_set_filename( GtkFileSelection *filesel,
                                      gchar            *filename );
```

要获取用户输入或点击选中的文本，可以使用下面的函数：

```
gchar *gtk_file_selection_get_filename( GtkFileSelection *filesel );
```

还有几个指向文件选择构件内部的构件的指针，它们是：

dir_list

file_list

selection_entry

selection_text

main_vbox

ok_button

cancel_button

help_button

在为文件选择构件的信号设置回调函数时，极有可能用到 ok_button、cancel_button和help_button三个指针。

下面的例子是来自testgtk.c中的一段代码。在这个例子中， Help按钮出现在屏幕上，但是它什么也不做，因为没有为它的信号设置回调函数。

```
/* 文件选择构件示例开始 filesel.c */

#include <gtk/gtk.h>

/* 获得文件名，并将它打印到控制台上 */
void file_ok_sel (GtkWidget *w, GtkFileSelection *fs)
{
    g_print ("%s\n", gtk_file_selection_get_filename (
            GTK_FILE_SELECTION (fs)));
}

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

int main (int argc, char *argv[])
{
    GtkWidget *filew;
        gtk_init (&argc, &argv);

    /* 创建一个新的文件选择构件 */
    filew = gtk_file_selection_new ("File selection");
        gtk_signal_connect (GTK_OBJECT (filew), "destroy",
                        (GtkSignalFunc) destroy, &filew);
    /* 为ok_button按钮设置回调函数，连接到file_ok_sel函数 */
    gtk_signal_connect (GTK_OBJECT (GTK_FILE_SELECTION (filew)->ok_button),
                        "clicked", (GtkSignalFunc) file_ok_sel, filew );

    /* 为cancel_button设置回调函数，销毁构件*/
    gtk_signal_connect_object (GTK_OBJECT (GTK_FILE_SELECTION
                                           (filew)->cancel_button),
```

```
                                    "clicked", (GtkSignalFunc) gtk_widget_destroy,
                                    GTK_OBJECT (filew));

    /* 设置一个文件名，作为它的缺省文件名 */
    gtk_file_selection_set_filename (GTK_FILE_SELECTION(filew),
                                     "penguin.png");

    gtk_widget_show(filew);
    gtk_main ();
    return 0;
}
/* 示例结束 */
```

编译后，在shell提示符下输入./filesel，运行结果如图9-9所示。这个构件可以用于打开和保存文件。

图9-9 文件选择构件