第12章 树 构 件

树构件一般用于显示分层结构的数据。树构件本身是 GtkTreeItem构件类型的垂直容器。树构件本身和分栏列表构件（ GtkCList）差别并不大：它们都是由容器构件派生而来的，其中与容器相关的函数在树构件和分栏列表构件中同样起作用。差别在于树构件可以嵌套另外的树构件。下面简要介绍怎么使用树构件。

树构件有自己的X窗口，缺省情况下是白色的背景。还有，绝大多数树的函数与分栏列表的函数工作方法是一样的。然而，树构件并不是从分栏列表构件派生而来的，因此，这些函数不可以互换。

12.1 创建新树构件

用下面函数创建树构件：

```
GtkWidget *gtk_tree_new( void );
```

像分栏列表构件一样，当项目添加到树构件时，或当子树扩展时，树构件会自动扩展。因为这个原因，所以需要将树构件组装到一个滚动窗口构件中。滚动窗口构件的缺省尺寸是很小的，因此一般对滚动窗口要用 gtk_widget_set_usize()函数设置它的缺省尺寸，以保证树构件足够大，可以看到其中的项目。

有了树构件后，就可以向其中添加项目了。现在，我们先介绍怎样创建树项构件：

```
GtkWidget *gtk_tree_item_new_with_label( gchar *label );
```

现在可以用下面的函数将树项构件添加到树构件中了：

```
void gtk_tree_append( GtkTree    *tree,
                      GtkWidget *tree_item );
void gtk_tree_prepend( GtkTree   *tree,
                       GtkWidget *tree_item );
```

注意，必须一次将全部树项添加到树构件中。没有与 gtk_list_*_items()等价的函数。

12.1.1 添加一个子树

子树与树构件的创建方法类似。子树要加到一个树构件的树项下面，使用以下函数：

```
void gtk_tree_item_set_subtree( GtkTreeItem *tree_item,
                                GtkWidget   *subtree );
```

在子树添加到一个树项之前或之后都不需要对子树调用 gtk_widget_show()函数。然而，在调用 gtk_tree_item_set_subtree()之前，必须将树项添加到一棵父树上。这是因为：从技术上来说，子树的父树并不是拥有它的树项构件（ GtkTreeItem），而是树项所在的树。

当将子树添加到树项时，一个“＋”或“－”符号会出现在旁边，用户可以点击它“展开”或“折叠”子树——也就是显示或隐藏子树。缺省状态树项是折叠起来的。 注意，在折叠树项时，所有子树的选中状态仍然保持原有状态，这可能是用户不希望看到的。

12.1.2 处理选中的列表

与分栏列表构件GtkCList一样，树构件类型也有一个选择域。可以用下面的函数设置选择模式以控制树构件的行为：

```
void gtk_tree_set_selection_mode( GtkTree          *tree,
                                  GtkSelectionMode  mode );
```

与各种选择模式相关的语义在分栏列表构件GtkCList中有详细的介绍。与GtkCList构件一样，当选择列表项或取消选择时，会引发select_child、unselect_child、和selection_changed信号。然而，要使用这些信号，需要知道树构件怎样引发信号，以及如何查找选中的列表项。

所有的树构件都有自己的X窗口，它们能够接受事件，例如鼠标点击。然而，要让树构件的选择类型为GTK_SELECTION_SINGLE和GTK_SELECTION_BROWSE的树能够正常动作，选中项的列表必须是针对树构件层次上最顶层的（也就是所谓的“根”树）。

因而，除非你已经知道它是“根”树，否则，直接访问任意一个树构件的选择状态并不是什么好主意。应该使用GTK_TREE_SELECTION (Tree)宏，它将给出一个指向“根”树的选择列表的指针。当然，如果选择模式是GTK_SELECTION_MULTIPLE（可以多选），这个选择列表也包含了其他不在所涉及到的子树下的树项。

最后，整个树都能引发select_child(理论上还有unselect_child)信号，但是，只有根树才能引发selection_changed信号。因而，如果想对一棵树和它的子树的select_child信号进行响应，必须对每一个子树调用gtk_signal_connect()函数。

12.1.3 树构件内部机制

Tree的结构定义是这个样子：

```
struct _GtkTree
{
  GtkContainer container;
  GList *children;
  GtkTree* root_tree; /* owner of selection list */
  GtkWidget* tree_owner;
  GList *selection;
  guint level;
  guint indent_value;
  guint current_indent;
  guint selection_mode : 2;
  guint view_mode : 1;
  guint view_line : 1;
};
```

前面已经介绍过了直接访问树的选择状态的危险性。树构件定义中的其他部分也可以用宏或者函数访问。GTK_IS_ROOT_TREE (Tree)返回一个布尔值，指明一个树是否是“根”树，而GTK_TREE_ROOT_TREE (Tree)宏返回指定树的“根”树（GtkTree类型的值）(所以，要记住如果想使用gtk_widget_*()类型的函数，要用GTK_WIDGET (Tree)宏将对象转换为构件类型)。

与其直接访问Tree构件的子节点域，不如用GTK_CONTAINER (Tree)宏将树转换为一个

指针，然后将它传递到 gtk_container_children()函数中。这样将为原来的列表创建一个副本。使用完后要用g_free()函数释放它，或用破坏性的方法遍历它。如下所示：

```
    children = gtk_container_children (GTK_CONTAINER (tree));
    while (children) {
      do_something_nice (GTK_TREE_ITEM (children->data));
      children = g_list_remove_link (children, children);
}
```

上面的tree_owner域只在子树中有定义，它指向容纳这个子树的树项构件； level域指出特定树的嵌套层次，根部树的 level是0，每一个子树比它的双亲树的 level大1。这个域只有在树已在屏幕上绘出后才有设置值。

12.1.4 信号

```
void selection_changed( GtkTree *tree );
```

当树构件的选择区域发生变化时，会引发这个信号。也就是当选中树构件的子树或取消选择时。

```
void select_child( GtkTree   *tree,
                   GtkWidget *child );
```

当选中树构件的子树时，将引发这个信号。当调用 gtk_tree_select_item()和gtk_tree_select_child()、或当鼠标按钮按下，调用 gtk_tree_item_toggle()和gtk_item_toggle()函数时，也会引发该信号。当子树被添加到树上或者将子树从树上删除时会间接引发这个信号。

```
void unselect_child (GtkTree   *tree,
                     GtkWidget *child);
```

当树的子树要被取消选中时，引发这个信号。

12.1.5 函数和宏

```
guint gtk_tree_get_type( void );
```

返回GtkTree构件的类型标识符

```
GtkWidget* gtk_tree_new( void );
```

创建新树，并返回指向GtkWidget对象类型的指针。如果创建失败，返回 NULL。

```
void gtk_tree_append( GtkTree   *tree,
                      GtkWidget *tree_item );
```

将一个树项添加到树构件的后面。

```
void gtk_tree_prepend( GtkTree   *tree,
                       GtkWidget *tree_item );
```

将一个树项添加到树构件的前面。

```
void gtk_tree_insert( GtkTree   *tree,
                      GtkWidget *tree_item,
                      gint       position );
```

在树构件中的指定位置插入树项。

```
void gtk_tree_remove_items( GtkTree *tree,
                            GList   *items );
```

将一个树项列表(GList *形式的列表)从树构件中删除。注意，从一个树上删除一个树项会

解除它和它的子树，以及该子树的子树（如果有的话）的引用。如果只想删除一个树项，可以使用gtk_container_remove()函数。

```
void gtk_tree_clear_items( GtkTree *tree,
                           gint     start,
                           gint     end );
```

删除树构件中从start位置到end位置的树项。和上一个函数一样，它会解除树项及其子树的引用，因为这个函数只是构造一个列表，然后将列表传递给 gtk_tree_remove_items()函数。

```
void gtk_tree_select_item( GtkTree *tree,
                           gint     item );
```

对指定item位置的树项引发“ select_item ”信号，并选中它 (除非在信号处理函数中取消选择)。

```
void gtk_tree_unselect_item( GtkTree *tree,
                             gint     item );
```

对指定item位置的树项引发“ unselect_item ”信号，并取消选择。

```
void gtk_tree_select_child( GtkTree   *tree,
                            GtkWidget *tree_item );
```

对子树项引发“ select_item ”信号，并且选中它。

```
void gtk_tree_unselect_child( GtkTree   *tree,
                              GtkWidget *tree_item );
```

让子树项引发“ unselect_item ”信号，并取消选中状态。

```
gint gtk_tree_child_position( GtkTree   *tree,
                              GtkWidget *child );
```

返回一个子构件在树中的位置。如果子构件不在树中，将返回 -1。

```
void gtk_tree_set_selection_mode( GtkTree          *tree,
                                  GtkSelectionMode  mode );
```

设置选择模式。模式可以是GTK_SELECTION_SINGLE (缺省值)，或GTK_SELECTION_BROWSE、GTK_SELECTION_MULTIPLE或GTK_SELECTION_EXTENDED。它只对根部树有定义，也只有对根部树有意义，因为只有根部树才会被选择。它对子树设置没有任何效果，并简单忽略设置值。

```
void gtk_tree_set_view_mode( GtkTree         *tree,
                             GtkTreeViewMode  mode );
```

设置“ 视图模式 ”—— 可以是GTK_TREE_VIEW_LINE (缺省值)或GTK_TREE_VIEW_ITEM。视图模式从一个树构件传递到它的子树，并且不能对某个子树单独设置 (这种说法也不完全正确)。

用术语解释“ 视图模式 ”相当暧昧，而在图中它决定当一个树的孩子被选中时，突出显示是什么样子。如果是 GTK_TREE_VIEW_LINE，整个树项会突出显示，如果是GTK_TREE_VIEW_ITEM，只有子构件(通常是标签)突出显示。

```
void gtk_tree_set_view_lines( GtkTree *tree,
                              guint    flag );
```

是否在树项间划线。Flag参数可以是TRUE —— 代表在树项间划线，或者FALSE，在树项间不划线。

```
GtkTree *GTK_TREE (gpointer obj);
```

将一个普通对象指针转换为GtkTree *指针。

```
GtkTreeClass *GTK_TREE_CLASS (gpointer class);
```

将一个普通类指针转换为GtkTreeClass *指针。

```
gint GTK_IS_TREE (gpointer obj);
```

判定一个普通指针是否指向一个GtkTree对象。

```
gint GTK_IS_ROOT_TREE (gpointer obj)
```

判定一个普通指针是否指向GtkTree构件以及它是否根树。虽然它可以接收任何指针，向函数传递一个并不指向树构件的指针也可能会引发问题。

```
GtkTree *GTK_TREE_ROOT_TREE (gpointer obj)
```

返回一个指向GtkTree构件的指针的根节点树。上面的警告同样适用。

```
GList *GTK_TREE_SELECTION( gpointer obj)
```

返回一个GtkTree构件的根树的选择列表。上面的警告同样适用。

12.2 树项构件GtkTreeItem

树项构件和分栏列表项构件一样，是从 GtkItem构件派生而来的，而GtkItem是从GtkBin构件派生而来的， 因此， GtkTreeItem是一种普通的容器，它只能包含一个子构件，且子构件可以是任何类型的。树项构件有一些额外的域，我们唯一需要关心的是 subtree —— 子树域。

TreeItem结构的定义是下面这样的：

```
struct _GtkTreeItem
{
  GtkItem item;
  GtkWidget *subtree;
  GtkWidget *pixmaps_box;
  GtkWidget *plus_pix_widget, *minus_pix_widget;

  GList *pixmaps;
  guint expanded : 1;
};
```

pixmaps_box域是一个事件盒构件，它捕捉对“＋”或“－”的点击，控制树的展开或折叠。pixmap域指向一个内部的数据结构。因为它总是可以使用 GTK_TREE_ITEM_SUBTREE (Item)以一种相对安全的方式获得树项构件的 subtree域，因而最好不要对树项构件的内部进行操作，除非你确实知道你在做什么。

因为它是直接从GtkItem构件派生而来的，所以它可以用GTK_ITEM (TreeItem) 宏转换为一个指针。树项通常带一个标签，因此可以用gtk_list_item_new_with_label()方便地创建新的树项。同样的效果可以用下面的代码获得。这些代码实际上是逐字从 gtk_tree_item_new_with _label()函数中复制而来：

```
tree_item = gtk_tree_item_new ();
label_widget = gtk_label_new (label);
gtk_misc_set_alignment (GTK_MISC (label_widget), 0.0, 0.5);

gtk_container_add (GTK_CONTAINER (tree_item), label_widget);
gtk_widget_show (label_widget);
```

因为上面的方法并不是强制性地将标签构件添加到树项里面，所以可以将一个水平组装盒构件（GtkHBox）或箭头构件（GtkArrow）甚至一个笔记本构件（GtkNotebook）(虽然这样应用程序看起来会很奇怪)添加到树项中。

如果从一个子树中删除所有的树项，则这些树项会被销毁，并与子树解除父子关系，除非事前引用它。上层的树项会折叠起来，所以如果想要保留这些这些树项，应该做下面的工作：

```
gtk_widget_ref (tree);
owner = GTK_TREE(tree)->tree_owner;
gtk_container_remove (GTK_CONTAINER(tree), item);
if (tree->parent == NULL){
  gtk_tree_item_expand (GTK_TREE_ITEM(owner));
  gtk_tree_item_set_subtree (GTK_TREE_ITEM(owner), tree);
}
else
  gtk_widget_unref (tree);
```

可以对树项进行“拖放”，但要保证满足下面的条件：当调用 gtk_widget_dnd_drag_set()或gtk_widget_dnd_drop_set()函数时要拖曳的树项和放下的树项都已经添加到一个树上，并且每一个父构件都有自己的父构件，一直到顶级窗口或对话框窗口为止。否则，会发生很奇怪的事。

12.2.1 信号

树项构件GtkTreeItem从GtkItem构件中继承了select、deselect和toggle 信号。另外，它添加了两个自己的信号：expand和collapse。

```
void select( GtkItem *tree_item );
```

当一个item要被选中时引发这个信号，当用户点击它后也会引发，或当应用程序调用gtk_tree_item_select()、gtk_item_select()或gtk_tree_select_child()函数时。

```
void deselect( GtkItem *tree_item );
```

当取消选择树项时，或当用户在一个已选中的树项上点击时，会引发这个信号；当应用程序调用 gtk_tree_item_deselect()或gtk_item_deselect()函数时也会引发该信号。对树项来说，在调用 gtk_tree_unselect_child()和gtk_tree_select_child()函数时，也会引发该信号。

```
void toggle( GtkItem *tree_item );
```

当应用程序调用gtk_item_toggle()函数时，引发这个信号。效果是：如果该树项有一个父树的话，在树项上引发这个信号时，对树项的父树调用 gtk_tree_select_child()（绝不会调用gtk_tree_unselect_child()函数）；如果没有父树，高亮显示会反过来。

```
void expand( GtkTreeItem *tree_item );
```

当要展开一个树项的子树时，将引发整个信号。也就是当用户点击树项旁边的“＋”时，或当应用程序调用gtk_tree_item_expand()函数时引发。

```
void collapse( GtkTreeItem *tree_item );
```

当一个树项的子树要折叠时，引发这个信号。也就是，当用户点击树项旁边的“－”，或当应用程序调用gtk_tree_item_collapse()函数时引发。

12.2.2 函数和宏

```
guint gtk_tree_item_get_type( void );
```

返回“ GtkTreeItem ”的类型标识符。

```
GtkWidget* gtk_tree_item_new( void );
```

创建新的树项构件GtkTreeItem。返回一个指向GtkWidget对象的指针。如果创建失败，将返回NULL。

```
GtkWidget* gtk_tree_item_new_with_label (gchar       *label);
```

创建一个新的树项对象，该对象有一个唯一的子构件 GtkLabel。返回一个指向GtkWidget对象的指针。如果创建失败，则返回 NULL。

```
void gtk_tree_item_select( GtkTreeItem *tree_item );
```

这个函数是gtk_item_select (GTK_ITEM (tree_item))函数的一个封装，调用它时会引发select信号。

```
void gtk_tree_item_deselect( GtkTreeItem *tree_item );
```

这个函数基本上是gtk_item_deselect (GTK_ITEM (tree_item))函数调用的一个封装。调用此函数时会引发deselect信号。

```
void gtk_tree_item_set_subtree( GtkTreeItem *tree_item,
                                GtkWidget   *subtree );
```

这个函数将一个子树subtree添加到树项tree_item上。如果tree_item是展开的，会显示该子树；如果tree_item是折叠的，会隐藏该子树。还有， tree_item必须已经添加到树上。

```
void gtk_tree_item_remove_subtree( GtkTreeItem *tree_item );
```

删除树项的子树的所有树项 (解除引用，然后销毁它，从子树向下一直到最末端的树项)，然后删除该子树，并隐藏“ ＋/－ ”符号。

```
void gtk_tree_item_expand( GtkTreeItem *tree_item );
```

使树项tre_item引发expand信号，展开该树项。

```
void gtk_tree_item_collapse( GtkTreeItem *tree_item );
```

使树项tree_item引发collapse信号，将该树项折叠。

```
GtkTreeItem *GTK_TREE_ITEM (gpointer obj)
```

将一个指针转换为GtkTreeItem *。

```
GtkTreeItemClass *GTK_TREE_ITEM_CLASS (gpointer obj)
```

将一个指针转换为GtkTreeItemClass类。

```
gint GTK_IS_TREE_ITEM (gpointer obj)
```

判定一个普通指针是否指向一个GtkTreeItem对象。

```
GtkWidget GTK_TREE_ITEM_SUBTREE (gpointer obj)
```

返回一个树项的子树(obj参数应该是指向一个GtkTreeItem对象的指针)。

12.3 树构件示例

下面的代码是树构件的一个示例。它在窗口中添加了一个树构件，并为所有相关对象的信号设置了回调函数。尝试一下，看看这些信号是怎样引发的，能用来做些什么。

```
/* 树构件示例开始tree.c */
```

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

/* for all the GtkItem:: and GtkTreeItem:: signals */
static void cb_itemsignal (GtkWidget *item, gchar *signame)
{
  gchar *name;
  GtkLabel *label;

  /* 它是从GtkBin派生而来，所以它只能有一个子构件 */
  label = GTK_LABEL (GTK_BIN (item)->child);
  /* 取得标签的文本*/
  gtk_label_get (label, &name);
  /*获取树项所在树的层次*/
  g_print ("%s called for item %s->%p, level %d\n", signame, name,
           item, GTK_TREE (item->parent)->level);
}

/* 注意，这个函数没有被调用过*/
static void cb_unselect_child (GtkWidget *root_tree, GtkWidget *child,
                               GtkWidget *subtree)
{
  g_print ("unselect_child called for root tree %p, subtree %p, child %p\n",
           root_tree, subtree, child);
}

/* 注意这个函数在用户点击树项时调用，不管它是否已经被选中 */
static void cb_select_child (GtkWidget *root_tree, GtkWidget *child,
                             GtkWidget *subtree)
{
  g_print ("select_child called for root tree %p, subtree %p, child %p\n",
           root_tree, subtree, child);
}

static void cb_selection_changed (GtkWidget *tree)
{
  GList *i;
    g_print ("selection_change called for tree %p\n", tree);
  g_print ("selected objects are:\n");

  i = GTK_TREE_SELECTION(tree);
  while (i){
    gchar *name;
    GtkLabel *label;
    GtkWidget *item;

    /* 从列表节点获得一个指向GtkWidget类型的指针*/
    item = GTK_WIDGET (i->data);
    label = GTK_LABEL (GTK_BIN (item)->child);
    gtk_label_get (label, &name);
    g_print ("\t%s on level %d\n", name, GTK_TREE
             (item->parent)->level);
```

```
    i = i->next;
  }
}

int main (int argc, char *argv[])
{
  GtkWidget *window, *scrolled_win, *tree;
  static gchar *itemnames[] = {"Foo", "Bar", "Baz", "Quux",
                               "Maurice"};
  gint i;

  gtk_init (&argc, &argv);

  /* a generic toplevel window */
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_signal_connect (GTK_OBJECT(window), "delete_event",
                      GTK_SIGNAL_FUNC (gtk_main_quit), NULL);
  gtk_container_set_border_width (GTK_CONTAINER(window), 5);

  /* 创建一个滚动窗口 */
  scrolled_win = gtk_scrolled_window_new (NULL, NULL);
  gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_win),
                                  GTK_POLICY_AUTOMATIC,
                                  GTK_POLICY_AUTOMATIC);
  gtk_widget_set_usize (scrolled_win, 150, 200);
  gtk_container_add (GTK_CONTAINER(window), scrolled_win);
  gtk_widget_show (scrolled_win);

  /* 创建一个根树*/
  tree = gtk_tree_new();
  g_print ("root tree is %p\n", tree);
  /* connect all GtkTree:: signals */
  gtk_signal_connect (GTK_OBJECT(tree), "select_child",
                      GTK_SIGNAL_FUNC(cb_select_child), tree);
  gtk_signal_connect (GTK_OBJECT(tree), "unselect_child",
                      GTK_SIGNAL_FUNC(cb_unselect_child), tree);
  gtk_signal_connect (GTK_OBJECT(tree), "selection_changed",
                      GTK_SIGNAL_FUNC(cb_selection_changed), tree);
  /* 将这个树加到滚动窗口中*/
  gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW(scrolled_win),
                                         tree);
  /* 设置选择模式 */
  gtk_tree_set_selection_mode (GTK_TREE(tree),
                               GTK_SELECTION_MULTIPLE);
  /* 显示树构件 */
  gtk_widget_show (tree);

  for (i = 0; i < 5; i++){
    GtkWidget *subtree, *item;
    gint j;
```

```
/* 创建一个树项*/
item = gtk_tree_item_new_with_label (itemnames[i]);
/* 为所有的树项信号设置回调函数*/
gtk_signal_connect (GTK_OBJECT(item), "select",
                    GTK_SIGNAL_FUNC(cb_itemsignal), "select");
gtk_signal_connect (GTK_OBJECT(item), "deselect",
                    GTK_SIGNAL_FUNC(cb_itemsignal), "deselect");
gtk_signal_connect (GTK_OBJECT(item), "toggle",
                    GTK_SIGNAL_FUNC(cb_itemsignal), "toggle");
gtk_signal_connect (GTK_OBJECT(item), "expand",
                    GTK_SIGNAL_FUNC(cb_itemsignal), "expand");
gtk_signal_connect (GTK_OBJECT(item), "collapse",
                    GTK_SIGNAL_FUNC(cb_itemsignal), "collapse");
/* 将树项添加到树上*/
gtk_tree_append (GTK_TREE(tree), item);
/* Show it - this can be done at any time */
gtk_widget_show (item);
/* 为树项创建子树*/
subtree = gtk_tree_new();
g_print ("-> item %s->%p, subtree %p\n", itemnames[i], item,
         subtree);

/* 如果想要这些信号对子树的子构件也起作用，下面的代码就是必要的。
 * 注意，"selection_change"信号总会在根树上引发*/
gtk_signal_connect (GTK_OBJECT(subtree), "select_child",
                    GTK_SIGNAL_FUNC(cb_select_child), subtree);
gtk_signal_connect (GTK_OBJECT(subtree), "unselect_child",
                    GTK_SIGNAL_FUNC(cb_unselect_child), subtree);
/* 下面一句代码没有效果，因为对子树来说，它已经被全部忽略了。 */
gtk_tree_set_selection_mode (GTK_TREE(subtree),
                             GTK_SELECTION_SINGLE);
/* 下面的代码也没有作用，不过理由与上面不一样：
 * 树构件的"view_mode"和"view_line"值是从根
 * 树开始从上往下传递的,所以，在后面设置有可能
 * 会产生不可预料的结果*/
gtk_tree_set_view_mode (GTK_TREE(subtree), GTK_TREE_VIEW_ITEM);
/* 设置树项的子树。注意，只有将树项加到父树上之后才能做这件事 */
gtk_tree_item_set_subtree (GTK_TREE_ITEM(item), subtree);

for (j = 0; j < 5; j++){
  GtkWidget *subitem;

  /* 创建一个子树 */
  subitem = gtk_tree_item_new_with_label (itemnames[j]);
  /* 连接所有树项的回调函数*/
  gtk_signal_connect (GTK_OBJECT(subitem), "select",
                      GTK_SIGNAL_FUNC(cb_itemsignal), "select");
  gtk_signal_connect (GTK_OBJECT(subitem), "deselect",
                      GTK_SIGNAL_FUNC(cb_itemsignal), "deselect");
  gtk_signal_connect (GTK_OBJECT(subitem), "toggle",
```

```
                        GTK_SIGNAL_FUNC(cb_itemsignal), "toggle");
      gtk_signal_connect (GTK_OBJECT(subitem), "expand",
                        GTK_SIGNAL_FUNC(cb_itemsignal), "expand");
      gtk_signal_connect (GTK_OBJECT(subitem), "collapse",
                        GTK_SIGNAL_FUNC(cb_itemsignal), "collapse");
      g_print ("-> -> item %s->%p\n", itemnames[j], subitem);
      /* 将它加到父树构件上*/
      gtk_tree_append (GTK_TREE(subtree), subitem);
      /* 显示这个构件 */
      gtk_widget_show (subitem);
    }
  }

  /* 显示窗口，然后进入主循环 */
  gtk_widget_show (window);
  gtk_main();
  return 0;
}
/* 示例结束 */
```

此示例的运行结构如图12-1所示。

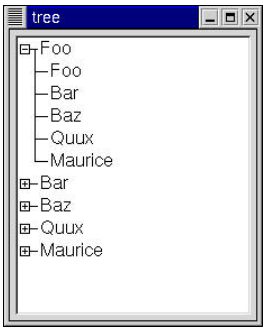

图12-1 树构件示例