第4章 构件定位

4.1 构件的显现、映射和显示

构件可以按它们是否有 GdkWindow窗口分类。有两种 Gtk+构件，一种有一个相关联的 GdkWindow窗口，另一种没有。大多数构件都有一个相关联的 GdkWindow窗口，构件就绘制在这个窗口上。这里的GdkWindow窗口和Gtk+里的GtkWindow窗口是不一样的。GdkWindow不是一个用户可见的对象，而是一个 X服务器用于划分屏幕的抽象概念。一个 GdkWindow窗口，对X服务器给出了关于将要显示的图形的结构信息。因为 X窗口系统是网络透明的，有可能X窗口的显示位置和X服务器不在同一台机器上，这样有助于减少网络流量。 GtkWindow是一个窗口构件，它是一个用户可见的对象。

还有一些构件，比如说 GtkLabel构件，没有与之相关联的 GdkWindow；它们被称为“无窗口构件”，并且是相对轻量级的。没有相关联窗口的构件绘制在它的父构件的 GdkWindow窗口上。一些操作，例如捕获一个事件，要求有一个 GdkWindow窗口，因此不能在无窗口构件上做这些操作。

构件要经过一系列与它们的GdkWindow相关的状态：

- 如果一个构件相应的 GdkWindow被创建出来，称为该构件被显现（ realize）。用 gtk_widget_realize()函数显现一个构件，用 gtk_widget_unrealize()函数反显现（unrealized）构件。因为X窗口必须有一个父窗口，如果一个构件已经显现，它的父窗口也必然已显现。
- 如果在构件的GdkWindow上调用了gdk_window_show()函数，称为该构件被映射（map）了。这意味着服务器已经要求在屏幕上显示这个构件的 GdkWindow窗口。很明显，GdkWindow窗口必须存在，也就是说，被映射的构件必然已被显现。
- 如果当一个构件的父构件被映射时，它也被映射到屏幕上，这个构件就是可见的。这意味着已经对该构件调用了 gtk_widget_show()函数。通过调用 gtk_widget_hide()函数，一个构件可以绘制为不可见的，这或者是取消未决的映射（已经确定了映射的时间，但还未映射），或者反映射该构件 (隐藏它的GdkWindow窗口)。因为顶级构件没有父构件，当它们一显示，它们同时就被映射了。

在典型的用户代码中，只需调用 gtk_widget_show()函数。这暗含当它的父构件一旦被显现和映射，该构件就被显现和映射。要理解的是： gtk_widget_show()函数并不会立即生效，这一点很重要，它仅仅是确定构件被显示出来的时间。也就是说，不用担心显示构件的顺序（不必一定要先显示子构件，再显示父构件）。但是，这时还不能立即访问这个构件的GdkWindow窗口。有时，又确实需要在映射之前访问构件的 GdkWindow窗口；在这样的情况下，要手工调用 gtk_widget_realize()函数来创建这个 GdkWindow。如果机会适当，gtk_widget_realize()函数还会显现构件的父构件。使用 gtk_widget_realize()函数的情况是不多见的，如果感觉到一定要这么做时，也许是使用了不正确的方法。

上面介绍了创建构件的过程。销毁构件自动地将以上事件的整个次序倒过来，递归取消子构件和构件本身的显现。

正如上面所提到的，对构件调用 gtk_widget_show()函数之后，构件并不一定会显示。只有当构件的所有父构件（直到最高级别的父构件）全部显示之后，它才会显示。因而，一般情况下，应该最后对最高级别的构件 GtkWindow调用gtk_widget_show()函数。否则，如果先显示高级别的构件，用户可能会看到窗口先出现在屏幕上，然后子构件一个一个显示在屏幕上。用户或许会觉得你的程序不够专业，甚至不正确。

下面是显现、映射和显示构件的相关函数。

函数列表： 显示/映射/显现构件

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

/* 显现一个构件，创建该构件的GdkWindow*/
void gtk_widget_realize(GtkWidget* widget)

/* 反显现构件，销毁该构件的GdkWindow*/
void gtk_widget_unrealize(GtkWidget* widget)

/*映射构件，构件的GdkWindow显示在窗口上*/
void gtk_widget_map(GtkWidget* widget)

/*反映射构件，隐藏构件的GdkWindow。注意，构件的GdkWindow还存在*/
void gtk_widget_unmap(GtkWidget* widget)

/*显示构件，当构件的父构件（向上递归直到最高级别构件）显示时，
 *构件将显示在屏幕上，*/
void gtk_widget_show(GtkWidget* widget)

/*隐藏构件，构件的GdkWindow依然存在*/
void gtk_widget_hide(GtkWidget* widget)
```

4.2 其他的构件概念

本节介绍了几个其他与 GtkWidget基类相关的概念，其中包括敏感性、焦点以及构件状态。

1. 敏感性

构件可以是敏感或不敏感的，不敏感的构件不能对输入进行响应。一般不敏感的构件是灰色的，不能接收键盘焦点。用 gtk_widget_set_sensitive()函数改变构件的敏感度。

函数列表： 改变敏感度

```
#include <gtk/gtkwidget.h>
/*设置构件的敏感性，widget参数是要设置的构件，setting设置为TRUE时，
 *构件是敏感的，setting设置是FALSE时，构件不敏感*/
void gtk_widget_set_sensitive(GtkWidget* widget,
                              gboolean setting)
```

构件缺省是敏感的。只有构件的所有容器是敏感的，构件才能是敏感的。也就是，可以通过设置容器的敏感性来让整个容器内的构件敏感 (或不敏感)。构件“真正的”敏感性，包括

它的父构件的状态，可以用 GTK_WIDGET_IS_SENSITIVE()宏测试。构件本身的敏感性，只与构件的父构件的敏感性有关，可以用 GTK_WIDGET_SENSITIVE()宏来查询。

宏列表： 敏感性

```
#include <gtk/gtkwidget.h>
GTK_WIDGET_IS_SENSITIVE(widget)
GTK_WIDGET_SENSITIVE(widget)
```

2. 焦点

某个时候，在一个顶级窗口中某个构件可能具有键盘焦点。顶级窗口接收到的任何键盘事件都被发送到这个构件。这一点很重要，因为在键盘上击键应该只有唯一的一种效果，例如，只能更改一个文本输入区域。

大多数构件在具有焦点时，会有一个视觉的指示。当使用缺省的 Gtk+主题（Theme）时，典型情况下，有焦点的构件有一个细黑框环绕着。用户可以用方向键或 Tab键在构件之间移动焦点。当用户用鼠标点击构件时，焦点也会移过去。

对键盘导航来说，焦点的概念是很重要的。例如，按下回车键或空格键会“激活”许多具有焦点的构件；可以用 Tab键在按钮之间移动，按下空格键激活这个按钮。

3. 独占

构件能够从其他构件中独占（ grab）鼠标指针和键盘。所谓独占，就是构件是“模态”的，用户只能向这个构件中输入字符，键盘焦点也不能改变到其他构件。独占输入的一个典型理由是：创建一个模态对话框时，如果窗口是独占的，则不能与其他的窗口交互。注意，还有另外一个Gdk级的“独占”。Gdk键盘和鼠标指针的独占发生在 X服务器范围内，也就是，其他应用程序不能接收到键盘和鼠标事件。构件独占是一个 Gtk+概念，它只独占同一个应用程序中的其他构件的事件。

4. 缺省

每个窗口至多有一个缺省构件。例如，典型情况下，对话框都有一个缺省按钮，当用户按回车键时，相当于点击了这个按钮。

5. 构件状态

构件的状态值决定了它们的外观：

- Normal：就是正常该有的样子。
- Active：例如，按钮正被按下，或检查按钮 (check box)正被选中。
- Prelight：鼠标指针越过一个构件 (典型情况，按下会有一些效果)。例如，当鼠标越过按钮时，按钮会“高亮显示”。
- Selected：构件是在一个列表中，或者是在其他类似状态，当前它是被选中的。
- Insensitive：构件是“灰色”的，不活动的，或者不响应。它不会对输入响应。

状态的准确含义及其视觉表达依赖于特定构件以及当前窗口管理器的主题（ Theme）。可以用GTK_WIDGET_STATE()宏存取构件的状态。这个宏返回以下几种常量之一：

GTK_STATE_NORMAL
GTK_STATE_ACTIVE
GTK_STATE_PRELIGHT
GTK_STATE_SELECTED
GTK_STATE_INSENSITIVE.

宏列表：状态存取函数

```
#include <gtk/gtkwidget.h>
GTK_WIDGET_STATE(widget)
```

4.3 构件的类型转换

在GTK中，所有构件的存储形式都是GtkWidget，但是许多函数都需要指向某种构件类型（比如GtkButton）的指针作为参数。虽然所有的构件都是从GtkWidget派生而来的，但是编译器并不能理解这种派生和继承关系。为此， GTK引进了一套类型转换系统。这些类型转换都是通过宏实现的。这些宏测试给定数据项的类型转换能力，并实现类型转换。下面几个宏是经常会碰到的：

GTK_WIDGET(widget)

GTK_OBJECT(object)

GTK_SIGNAL_FUNC(function)

GTK_CONTAINER(container)

GTK_WINDOW(window)

GTK_BOX(box)

所有的构件都是从Object基类派生而来的，这意味着可以在任何需要一个 GtkObject作为参数的函数中使用构件 —— 用GTK_OBJECT() 宏将构件转换为指向GtkObject类型的指针就可以了。

例如：

```
gtk_signal_connect( GTK_OBJECT(button), "clicked",
                    GTK_SIGNAL_FUNC(callback_function), callback_data);
```

这里将“ button ”转换为一个GtkObject对象，并将回调函数名称转换为指向函数的指针。

许多构件也是容器，它们都是从 Container内派生而来。这类构件可以用一个GTK_CONTAINER宏将它转换为容器，传递到需要容器指针的函数中。

4.4 组装构件

前面介绍了用Gtk+/Gnome构件编写Linux应用程序的编程思想。归纳起来就是“ 事件驱动 ”。也就是，用Gtk+/Gnome构件创建应用程序界面，然后为构件的信号设置回调函数。当用户对界面进行操作时，会引发各种信号。如果某个信号，比如一个按钮的“ clicked ”信号连接了回调函数，就会调用这个回调函数。通过在回调函数里面使用代码控制构件的各种属性来与用户交互，或者对内存变量进行操作，实现程序的各种功能。因而，编程的核心就归结为怎样使用构件创建界面，怎样为构件的信号设置回调函数。

在Gtk+版的“ Hello World ”的程序中，我们使用 gtk_container_add( )函数将按钮添加到窗口上。如果界面很复杂，不止一个按钮，怎么办呢？怎样在代码中创建构件，怎样将构件在窗口上定位呢？ GTK使用一种称为组装的方法实现了这一点。

GTK是用C语言编写的。虽然没有使用 C++这样的面向对象的语言， GTK实现了自己的具有继承和派生特性的对象系统。在 Gtk+构件里面，所有的构件都是从GtkObject对象派生而来的。每种派生构件都继承了父构件接口，同时再实现自己的一些特有功能。这样做的好处是显而易见的，既然新构件的某些功能在已有构件中已经全部具有，为什么不将这个构件直接

拿来用呢？从已有构件派生一个构件，继承旧构件原有的接口，然后再实现那些旧构件不具有的接口和功能就可以创建一个新构件。

在用Gtk+构件创建程序界面时，用容器实现构件的定位。在 Gtk+中有两种容器，它们都是抽象的GtkContainer构件的子类。第一种类型的容器构件总是由 GtkBin(另一种抽象基类)派生而来。 GtkBin的派生类只能容纳一个子构件，它们为其子构件增加一些功能。例如，GtkButton就是一种GtkBin，它让其子构件成为一个可接受点击的按钮。 GtkFrame也是一种GtkBin，它在其子构件周围绘制一个边框。同样， GtkWindow让它的子构件显示在一个顶级窗口上。

第二种容器构件通常是直接从GtkContainer派生而来。这些构件可以有多个子构件，它们的作用就是管理布局。“管理布局”意味着这些容器为它们容纳的子构件分配大小尺寸和位置。例如， GtkVBox将它的子构件在一个垂直的栈内排列。 GtkTable构件可以让构件在一个表格上根据单元格定位。GtkFixed可以将子构件放在任意坐标位置。 GtkPacker允许你做Tk-风格的布局管理。

实际上，Gtk+中的大多数构件都是一个容器。这样也给予我们极大的灵活性。例如，如果我们想要一个带图片的按钮，因为 GtkButton是一个容器，只需将一幅图片添加到这个容器里面就可以了。对按钮中是否包含文本，文本和图片之间相对位置等都可以自由设置。使用类似的方法，可以设计出非常复杂，甚至非常古怪的外观布局。

本章介绍第二种容器构件。要生成所需要的布局，并且对构件的尺寸不使用任何硬性编码，需要理解怎样使用这些容器构件。最终的目标之一是避免对窗口尺寸、屏幕尺寸、构件外观、字体等因素做任何假设。如果这些因素发生变化，应用程序应该能够自动适应。

4.4.1 尺寸分配

一个窗口显示在屏幕上，如果用鼠标调整窗口的尺寸，窗口里面的构件会发生什么变化？这依赖于定位构件的容器构件以及构件的定位选项。窗口上的构件可能会按一定的规则改变大小。而这些规则又是通过什么方式实现的呢？这是通过一种称为“请求”和“分配”的协商机制实现的。

1. 请求

构件的请求由宽度和高度组成：构件需要的大小尺寸。它由一个 GtkRequisition结构表示：

```
typedef struct _GtkRequisition    GtkRequisition;
struct _GtkRequisition
{
  gint16 width;
  gint16 height;
};
```

不同的构件用不同的方式选择它们所需要的尺寸。例如， GtkLabel构件，请求足够的尺寸用以在标签上显示所有的文本。大多数容器基于它们的子构件的尺寸请求来请求所需尺寸。例如，如果在一个Box里放几个按钮， Box会要求有足够大的空间以容纳所有的按钮。

布局的第一阶段从一个顶级构件，比如说 GtkWindow，开始。因为它是一个容器，GtkWindow询问它的子构件的尺寸要求，子构件再询问它的子构件，依此递归。当所有的子构件都被询问过后， GtkWindow最后会从它的子构件得到一个 GtkRequisition值。这依赖于它

是如何配置的，GtkWindow也许会或者不会扩展大小以满足所有的尺寸要求。

2. 分配

布局的第二阶段从这一点开始。GtkWindow对可供子构件使用的空间做出一个决定，并向子构件传达这个决定。这就是子构件的尺寸分配，由以下结构表示：

```
typedef struct _GtkAllocation     GtkAllocation;
struct _GtkAllocation
{
  gint16 x;
  gint16 y;
  guint16 width;
  guint16 height;
};
```

width和height元素与GtkRequisition是一样的，它们代表子构件的大小。GtkAllocation结构也包含子构件相对它们父构件的坐标。GtkAllocation由它们的父构件容器分配给子构件。

构件应该尊重传给它们的GtkAllocation值。GtkRequisition仅仅是一个请求，构件必须能够应付任何尺寸。

了解了构件布局的过程，就很容易弄清楚容器在其中扮演的角色。它们的任务就是将每个构件的请求汇总成单个请求，并沿着构件树向上(按层次向上)传递，然后，将它们接收到的大小分配划分给子构件。具体怎样划分依赖于具体的容器。

4.4.2 GtkWindow构件

前面介绍了关于构件的概念、构件的尺寸分配等。创建用户界面的目的就是将各种构件在屏幕上布局，提供一个用户与应用程序交互的接口。大多数情况下，不会将构件直接绘制在屏幕上，而是绘制在一个窗口上。如果窗口的尺寸不变化，而是窗口在屏幕上移动，这些构件在窗口的相对位置一般是固定的，它们随着窗口一起移动。

在Gtk/Gnome应用程序中，GtkWindow构件是最大的容器。GtkWindow是从GtkBin派生而来的，它只能容纳一个子构件。因而，要在其中容纳多个构件，必须使用 GtkBox、GtkTable或者GtkFixed等构件来控制构件布局。

用下面的函数创建新窗口：

```
GtkWidget*  gtk_window_new  (GtkWindowType type);
```

对于应用程序的窗口，type一般是GTK_WINDOW_TOPLEVEL。

下面的函数用于设置窗口的标题：

```
void gtk_window_set_title  (GtkWindow *window,
                            const gchar *title);
```

其中window是要设置标题的窗口，title是一个字符串（窗口标题）。

下面的函数用于设置顶级窗口在处理它的尺寸请求、以及用户调整尺寸时的行为：

```
void        gtk_window_set_policy          (GtkWindow *window,
                                            gint allow_shrink,
                                            gint allow_grow,
                                            gint auto_shrink);
```

注意，不要随意使用这个函数，否则，可能会使窗口的行为变得很古怪。一般只使用下面两种调用方法：

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

用户可调整窗口的尺寸。

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

窗口的尺寸是由程序控制的，只与窗口的子构件的当前尺寸相匹配。

第一种情况是缺省值，也就是说缺省的窗口就是这样的。

下面的函数用于设置窗口是否可调整尺寸。其中 setting是布尔值，当其值为TRUE时窗口尺寸可调，为FALSE时窗口大小不可调整：

```
void gtk_window_set_user_resizeable(GtkWidget* window, gboolean setting);
```

最好只使用下面两种形式设置窗口行为。GTK+ 1.4可能会用gtk_window_set_user_resizeable()函数替换gtk_window_set_policy()函数。

顶级窗口总是改变自己的尺寸以保证它的子构件能够接受到它们的请求值。这意味着如果添加一个子构件，顶级窗口会扩大以容纳它们。不过，如果窗口的尺寸太大，它不会自动缩小以适应子构件的尺寸请求。当 gtk_window_set_policy()函数中的auto_shrink参数设置为TRUE、子构件的空白区域太多时，窗口将自动缩小以适应子构件的大小。 auto_shrink参数通常用在上面提到的两种常见模式中的第二种。也就是说，如果想要让窗口总是根据程序的运行情况自动调整大小，将auto_shrink设置为TRUE。

注意 如果allow_shrink和allow_grow参数都设置为FALSE，auto_shrink参数没有任何作用。

前面提到的两种情况都没有将 allow_shrink参数设置为TRUE。如果allow_shrink设置为TRUE，用户能够缩小窗口的尺寸，使它的子构件不能接收到全部的尺寸请求。通常这是一个糟糕的主意，因为这样将使大多数构件的外观显示不正确。此外，如果由于某种原因，窗口的尺寸得到重新计算， GTK+倾向于重新扩展窗口。因而， allow_shrink参数应该总设置为FALSE。

使用allow_shrink时，存在的实际问题是一些特殊构件总是需要太多的空间，所以用户不能充分缩小窗口的大小。也许应该对子构件调用 gtk_widget_set_usize()函数，并迫使它的尺寸请求足够大。最好的办法是调用 gtk_window_set_default_size()函数设置构件的缺省尺寸，以便子构件获得比它请求的值更大的分配值。

```
void        gtk_window_set_default_size     (GtkWindow *window,
                                             gint width,
                                             gint height);
```

其中width和height是要设置的窗口的缺省宽度和高度。

用下面的构件向窗口中添加子构件：

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

窗口最常用的两个信号是 delete_event和destroy。当使用窗口管理器关闭窗口（点击窗口标题条上的“×”按钮），或者在窗口的某个构件上调用 gtk_widget_destroy()函数时，将引发delete_event信号。如果在delete_event信号处理函数中返回 FALSE，GTK将引发destroy信号。返回TRUE意味着不需要销毁窗口。对某些窗口，比如对话框，一般不在 delete_event信号中销毁窗口。返回FALSE，窗口会被信号销毁。

一般应该为窗口的delete_event信号设置一个回调函数，对该信号进行处理。否则，只要

用户点击窗口标题条上的“ × ”按钮，窗口就会关闭。

4.4.3 GtkBox

有两种GtkBox(组装盒)： GtkHBox（水平组装盒）和 GtkVBox（垂直组装盒）。一个GtkBox可以管理一行(GtkHBox)或一列 (GtkVBox)构件。对GtkHBox来说，所有的构件都分配了同样的高度，组装盒的作用就是在构件间分配可用空间。 GtkHBox还随意用一些可用宽度在构件间预留间隙 (称为“ 间距 ”)。GtkVBox的作用是一样的，不过是在垂直的方向上 (也就是，它分配可用的高度而不是宽度)。GtkBox是一个抽象的基类， GtkVBox和GtkHBox差不多可以完全使用它的接口。组装盒是最有用的容器构件。

用下面函数列表中的构建函数创建新的 GtkBox。组装盒构建函数带两个参数。homogeneous 参数如果为 TRUE（同质），意味着所有的子构件都被分配同样数量的空间。spacing参数指定每个子构件之间的间距。组装盒创建之后，还有函数可以改变 spacing参数值，切换构件的同质属性。

函数列表： GtkHBox构建函数

```
#include <gtk/gtkhbox.h>
GtkWidget* gtk_hbox_new(gboolean homogeneous,
           gint spacing)
```

函数列表： GtkVBox构建函数

```
#include <gtk/gtkvbox.h>
GtkWidget* gtk_vbox_new(gboolean homogeneous,
           gint spacing)
```

有两个基本的函数添加子构件到组装盒中，下面的函数列表中已将它们列出。

函数列表： 在GtkBox中添加构件

```
#include <gtk/gtkbox.h>
void gtk_box_pack_start(GtkBox* box,
                GtkWidget* child,
                gboolean expand,
                gboolean fill,
                gint padding)

void
gtk_box_pack_end(GtkBox* box,
               GtkWidget* child,
               gboolean expand,
               gboolean fill,
               gint padding)
```

一个组装盒可以包含两套构件。第一套是从组装盒的“ 头部 ”(顶部或左边)组装的；第二种是从“ 尾部 ”(底部或者右边)开始组装。如果将三个构件从盒子的“ 头部 ”开始组装到盒子里，组装的第一个构件会出现在盒子的最上面或者最左边，第二个构件紧接着第一个构件，第三个最接近盒子的中心。如果接着将三个按钮组从组装盒的“ 尾部 ”组装到盒子里，第

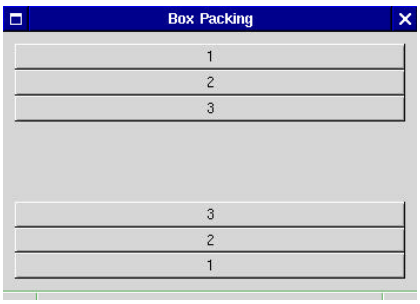

图4-1 组装到GtkVBox中的按钮

一个出现在盒子的最下边或者最右边，第二个紧挨着它，第三个最靠近盒子的中心。所有六个构件都组装之后，从上到下或者从左到右的次序是： 1，2，3，3，2，1。图4-1显示了这个例子。组装次序只对盒子的每个末端是重要的，也就是，可以使用不同的组装方法，取得相同的结果。

1 GtkBox布局细节

组装过程受三个参数影响，它们对从“ 头部 ”和“ 尾部 ”组装都是一样的。这些参数的含义很复杂，因为它们与盒子的homogeneous 设置以及每个构件相互作用。

下面介绍GtkBox怎样计算它的相关“ 方向 ”(对GtkHBox是width，对GtkVBox是height)上的尺寸请求：

1) 每个子构件的总尺寸请求等于每个子构件的尺寸请求加上两倍用于组装子构件的填充量值。子构件的填充量是在子构件两边的空白空间。简而言之，子构件尺寸 =(子构件的尺寸请求)＋2×(子构件的填充量)。

2) 如果盒子是同质的，整个盒子的基准尺寸请求等于最大的子构件尺寸请求 (请求数＋ 填充量)乘以子构件的数目。在同质的盒子中，所有的子构件都与最大的子构件一样大的。

3) 如果盒子不是同质的，整个盒子的基准尺寸请求等于每个子构件的尺寸请求 (请求数＋填充量)之和。

4) 组装盒范围内的spacing(间距)设置决定在子构件之间留多大的间距，所以这个值要乘以子构件数目减1，并加到基准尺寸请求中。注意，间距并不属于子构件，它是子构件之间的空白空间，不受expand和fill参数的影响。另一方面， padding(填充量)是环绕每个子构件的空间，受子构件的组装参数的影响。

5) 所有的容器都有一个“ 边框宽度 ”设置；它会将两倍的边框宽度(代表两边的边框宽度)加到尺寸请求中。这样， GtkBox的总的尺寸请求就是： (子构件尺寸之和)＋spacing×(子构件数－1)＋2×(边框宽度)。

在计算它的尺寸请求，将请求传送到它的父容器中之后， GtkBox会接收到它的尺寸分配，并将尺寸按下面的规则在子构件之间分配：

1) 边框宽度和构件间的间距从分配值中减去，剩余的部分是子构件自己的可用空间。这块空间分为两块：子构件实际要求的数量 (子构件的请求值和填充量)，以及“ 额外空间 ”。额外空间 = (分配尺寸) － (子构件尺寸之和)。

2) 如果盒子不是同质的，“ 额外 ”空间在按expand参数设置为TRUE时在子构件之间划分。这些子构件会占满可用空间。如果没有子构件能够展开，额外空间将用于在盒子中央（在从头组装和从尾部组装的构件之间）增加更多的空间。

3) 如果盒子是同质的，额外空间根据需要分配。那些请求较多空间的构件将得到较少的额外空间，让每个构件都占据同样的空间。同质的组装盒忽略 expand参数---额外空间分配给所有的子构件，而不是可扩展的构件。

4) 当构件获得一些额外空间时，有两种可能。在子构件周围增加更多的填充量，或者将子构件本身扩展。 fill参数决定哪种可能会发生。如果 fill是TRUE，子构件展开填充空间 ---也就是，整个空间变成子构件的分配值；如果 fill是FALSE，增加子构件的填充量，填满额外空间，只给子构件分配它请求的空间。注意，如果 expand设置为FALSE并且盒子也不是同质的，fill没有效果，因为子构件永远也不会接受到额外空间。

相信很少有人能记住这么罗嗦的规则。幸好， Gtk+教程的作者将这么多的设置归纳为5种情形，下面我们一步一步看。

2. 非同质组装盒的组装模式

在非同质的组装盒中有三种组装方法。第一种，将所有的构件用它们的正常尺寸组装到盒子的尾部。这意味着将expand参数设置为FALSE：

```
gtk_box_pack_start(GTK_BOX(box),
                   child,
                   FALSE, FALSE, 0);
```

结果显示在图4-2中。在这里，只有expand参数管用，没有子构件会接受额外空间，所以即使fill设置为TRUE它们也不能充满剩余空间。

图4-2 非同质，expand = FALSE

第二种方法，可以在GtkBox中扩展构件，让它们保持它们的正常尺寸，如图 4-3所示，这意味着将expand参数设置为TRUE：

```
gtk_box_pack_start(GTK_BOX(box),
                   child,
                   TRUE, FALSE, 0);
```

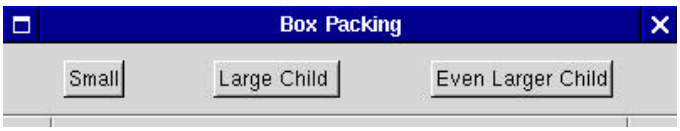

图4-3 非同质，expand = TRUE 和fill = FALSE

最后，可以将fill参数设置为TRUE，在盒子中填充构件(让最大的子构件有更大的空间)：

```
gtk_box_pack_start(GTK_BOX(box),
                   child,
                   TRUE, TRUE, 0);
```

这种设置的效果见图4-4。

图4-4 非同质，expand = TRUE 和 fill = TRUE

3. 同质GtkBox组装模式

组装一个同质的GtkBox只有两种有意义的方法。expand参数是与同质的GtkBox是无关的，所以这两种情况是对应于fill参数的不同设置。

如果fill是FALSE，将会得到图4-5的结果。注意，组装盒逻辑划分为三个部分，但是只有最大的子构件才占据了它的整个空间。其他构件只填满了空间的三分之一。如果 fill是TRUE，

将得到图4-6所示的结果，所有的构件都是一样大的。

图4-5 同质的GtkBox，fill = FALSE

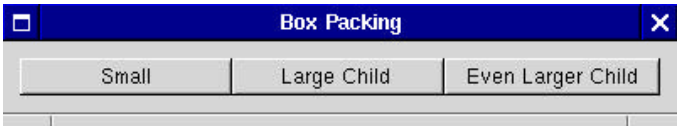

图4-6 同质的GtkBox，fill = TRUE

4. 组装摘要

图4-7同时显示了5种组装技巧。它们都是组装到一个同质的GtkVBox中，fill设置为TRUE，构件之间的间距是两个像素。从这应该能够理解它们的相对效果。要记住，还可以改变padding和spacing参数来增加或减少构件之间的空白空间。

最后一点：注意expand和fill参数只在盒子的尺寸比它要求的尺寸大时才有用。也就是说，这些参数决定额外的空间如何分配。典型情况下，当用户调整窗口的尺寸，让它比缺省尺寸大时，额外空间才显示出来。因而，应该尽量尝试调整窗口的尺寸，以保证构件是正确组装的。

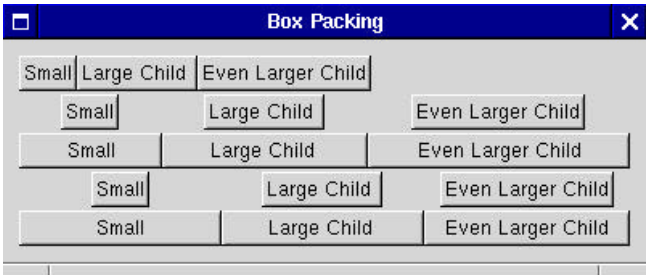

图4-7 五种用组装盒组装构件的方法

4.4.4 表格构件GtkTable

GtkTable（表格构件）是很常用的用于定位的构件。我们用表格构件创建一个网格，把构件放在网格里。构件可以在网格中占据任意多个格子。

用gtk_table_new创建一个表格构件：

```
GtkWidget *gtk_table_new( gint rows,
                          gint columns,
                          gint homogeneous);
```

第一个参数是表格的行数，第二个参数是表格的列数。

hmomgeneous参数与表格如何划分尺寸有关。如果 homogeneous参数是TRUE，表格的所有格子都是同样大的，并且，格子的大小等于表格中最大构件的大小。如果 homogeneous是FALSE，表格的大小由同一行上最高构件和同一列最宽的构件确定。行和列从 0到n编号，其

中n是调用gtk_table_new函数创建表格时指定的行或列数。这样，如果明确指定 rows = 2和columns = 2，外观就像图4-8所示一样：

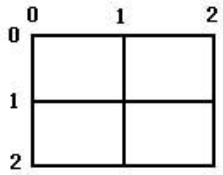

图4-8 2×2表格示意图

注意 坐标系统的原点在表格的左上角。

要将构件放到表格中，可以使用下列函数：

```
void gtk_table_attach( GtkTable  *table,
                       GtkWidget *child,
                       gint       left_attach,
                       gint       right_attach,
                       gint       top_attach,
                       gint       bottom_attach,
                       gint       xoptions,
                       gint       yoptions,
                       gint       xpadding,
                       gint       ypadding );
```

第一个参数“ table ”是前面创建的表格，第二个参数“ child ”是你希望放置在表格里的构件。left_attach和right_attach参数明确指定构件放在什么地方，并且占据多少个格子。

如果要将按钮放在2×2表格下面一行的右边的格子里面，并且想要它填充整个格子，可以设置为：left_attach= 1, right_attach = 2, top_attach = 1, bottom_attach =2。

现在，如果想让一个 widget占据2×2表格中的上面一行，应该使 left_attach = 0, right_attach = 2, top_attach = 0, bottom_attach =1。

xoptions和 yoptions用于指定组装选项，可以按“ 位或 ”设置以允许多重选项。这些选项是:

- GTK_FILL：如果表格的格子比 widget大，并且指定了GTK_FILL,构件将扩大并占满所有可用空间。
- GTK_SHRINK：如果表格比它所要求的空间还小， (通常由用户调整窗口的尺寸)，那么构件会被放在窗口的底部以外的区域，无法看见。如果指定 GTK_SHRINK，构件将和表格一同缩小。
- GTK_EXPAND：让表格使用窗口的所有保留空间。

padding参数和GtkBox里的padding一样，在构件的周围产生一个空白的区域。

gtk_table_attach()函数选项较多。为了方便，有下面的快捷函数：

```
void gtk_table_attach_defaults( GtkTable  *table,
                                GtkWidget *widget,
                                gint       left_attach,
                                gint       right_attach,
                                gint       top_attach,
                                gint       bottom_attach );
```

X和 Y选项默认是 GTK_FILL | GTK_EXPAND,并且X和Y的padding参数设为0，其余的参数与前面的函数一样。

我们也可以用gtk_table_set_row_spacing()和gtk_table_set_col_spacing()函数设置行、列间距。所设置的间距位于指定的行或列之间。

```
void gtk_table_set_row_spacing( GtkTable *table,
                                gint row,
```

```
                                gint spacing);
void gtk_table_set_col_spacing ( GtkTable *table,
                                 gint column,
                                 gint spacing);
```

注意 对列来说，空间位于列的右边，对行来说，它位于行的下边。

也可以为所有的行或/和列设置相同的间隔。

```
void gtk_table_set_row_spacings( GtkTable *table,
                                 gint spacing);
void gtk_table_set_col_spacings( GtkTable *table,
                                 gint spacing);
```

注意 在上面的函数中，最后一行和最后一列并没有设置任何空间。

表格组装示例

这里我们将制作一个窗口，上面放一个 2×2表格，在表格中放三个按钮。第一个和第二个按钮放在上面一排。第三个按钮(退出按钮)被放置在下面一排，跨越两列。

这里是源代码:

```
/* 表格示例table.c */

#include <gtk/gtk.h>

/* 回调函数
 * 将传递过来的数据打印到控制台上 */
void callback( GtkWidget *widget,
               gpointer   data )
{
    g_print ("Hello again - %s was pressed\n", (char *) data);
}

/* 这个回调函数退出程序 */
void delete_event( GtkWidget *widget,
                   GdkEvent  *event,
                   gpointer   data )
{
    gtk_main_quit ();
}

int main( int   argc,
          char *argv[] )
{
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *table;

    gtk_init (&argc, &argv);

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

```
/* 设置窗口标题 */
gtk_window_set_title (GTK_WINDOW (window), "Table");

/* 设置delete_event信号的回调函数，立即退出GTK*/
gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                    GTK_SIGNAL_FUNC (delete_event), NULL);

/* 设置窗口的边框宽度 */
gtk_container_set_border_width (GTK_CONTAINER (window), 20);

/* 创建一个2×2的表格 */
table = gtk_table_new (2, 2, TRUE);

/* 将窗口放在主窗口上*/
gtk_container_add (GTK_CONTAINER (window), table);

/* 创建第一个按钮 */
button = gtk_button_new_with_label ("button 1");

/* 点击按钮时，调用"callback"
 * 以一个指针"button 1作为它的参数*/
gtk_signal_connect (GTK_OBJECT (button), "clicked",
          GTK_SIGNAL_FUNC (callback), (gpointer) "button 1");

/* 将第一个按钮插入到表格的左上角格子中 */
gtk_table_attach_defaults (GTK_TABLE(table), button, 0, 1, 0, 1);
gtk_widget_show (button);

/* 创建第二个按钮 */
button = gtk_button_new_with_label ("button 2");
/* 点击这个按钮时，调用"callback"函数
 * 以指针"button 2为参数 */
gtk_signal_connect (GTK_OBJECT (button), "clicked",
          GTK_SIGNAL_FUNC (callback), (gpointer) "button 2");
/* 将第二个按钮插入到表格构件的右上角格子中 */
gtk_table_attach_defaults (GTK_TABLE(table), button, 1, 2, 0, 1);
gtk_widget_show (button);

/* 创建"Quit"按钮*/
button = gtk_button_new_with_label ("Quit");

/* 点击"Quit"按钮时，调用"delete_event"函数退出程序 */
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (delete_event), NULL);

/* 将退出按钮插入到表格构件下面的两个格子中 */
gtk_table_attach_defaults (GTK_TABLE(table), button, 0, 2, 1, 2);

gtk_widget_show (button);
gtk_widget_show (table);
```

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

图4-9 用表格定位构件

编译它，然后启动table应用程序。运行结果见图 4-9。尝试一下调整窗口的尺寸，看一看按钮如何变化。

4.4.5 固定容器构件GtkFixed

GtkFixed(固定容器构件)允许将构件放在窗口的固定位置，这个位置是相对于窗口的左上角的。构件的位置可以动态改变。

使用GtkFixed为构件定位，在大多数情况下都是不可取的。因为当用户调整窗口尺寸时，构件不能适应窗口的尺寸变化。当然，你可以在窗口尺寸变化的时候采取行动，调整构件的位置和大小。

只有三个与固定容器构件相关的函数：

```
GtkWidget* gtk_fixed_new( void );
void gtk_fixed_put( GtkFixed  *fixed,
                    GtkWidget *widget,
                    gint16     x,
                    gint16     y );
void gtk_fixed_move( GtkFixed  *fixed,
                     GtkWidget *widget,
                     gint16     x,
                     gint16     y );
```

其中：

gtk_fixed_new函数用于创建新的固定容器构件。

gtk_fixed_put函数将构件放在由x，y指定的位置。

gtk_fixed_move 函数将指定构件移动到新位置。

注意，这几个函数只是将构件定位，构件将以缺省尺寸显示。如果要想指定构件的长度和宽度，可以使用下面的函数。

```
#include <gtk/gtkwidget.h>
void gtk_widget_set_uposition(GtkWidget* widget,
                      gint x,
                      gint y)
void gtk_widget_set_usize(GtkWidget* widget,
                    gint width,
                    gint height)
```

这两个参数中的参数x、y、width和height可以是-1，这时对这个参数使用缺省值。

下面的例子演示了怎样使用固定容器构件。

```
/* 固定容器示例fixed.c */
#include <gtk/gtk.h>

gint x=50;
gint y=50;
```

```
/* 这个回调函数将按钮移动到固定容器的新位置 */
void move_button( GtkWidget *widget,
                  GtkWidget *fixed )
{
  x = (x+30)%300;
  y = (y+50)%300;
  gtk_fixed_move( GTK_FIXED(fixed), widget, x, y);
}

int main( int   argc,
          char *argv[] )
{
  GtkWidget *window;
  GtkWidget *fixed;
  GtkWidget *button;
  gint i;

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

  /* 创建一个新按钮 */
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "Fixed Container");

  /* 为窗口的"destory"事件设置一个信号处理函数 */
  gtk_signal_connect (GTK_OBJECT (window), "destroy",
                      GTK_SIGNAL_FUNC (gtk_main_quit), NULL);

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

  /* 创建一个固定容器构件*/
  fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), fixed);

  gtk_widget_show(fixed);

  for (i = 1 ; i <= 3 ; i++) {
    /* 创建一个标题为"Press Me的新按钮 */
    button = gtk_button_new_with_label ("Press me");

    /* 当按钮接收到"clicked"信号时, 调用move_button()函数*/
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (move_button), fixed);

    /* 将按钮组装到一个固定容器构件中*/
    gtk_fixed_put (GTK_FIXED (fixed), button, i*50, i*50);

    /* 最后一步是显示新建的构件 */
    gtk_widget_show (button);
```

```
    }

    /* 显示窗口 */
    gtk_widget_show (window);

    /* 进入主循环*/
    gtk_main ();

    return(0);
}

/* 示例结束 */
```

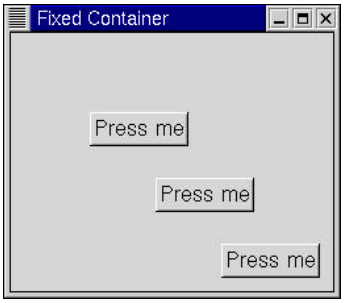

图4-10 GtkFixed构件

上面的代码的执行效果见图4-10。点击“Press Me”按钮，按钮会在窗口上移动（实际上是在GtkFixed上移动）。

4.4.6 布局容器构件GtkLayout

GtkLayout(布局容器构件)与固定容器构件类似，不过它可以在一个无限的滚动区域定位构件(其实也不能大于4294967296像素)。在X系统中，窗口的宽度和高度只能限于在32767像素以内。布局容器构件使用一些特殊的技巧越过这种限制。所以，当在滚动区域内有很多子构件时，可以让滚动更平滑。

用以下函数创建布局容器构件：

```
GtkWidget *gtk_layout_new( GtkAdjustment *hadjustment,
                           GtkAdjustment *vadjustment );
```

可以看到指定布局容器构件滚动时要使用的调整对象。关于“调整对象”，我们在后面的章节另有介绍。

用以下函数在布局容器构件内添加和移动子构件。将构件widget放在布局容器构件中坐标为x、y的位置：

```
void gtk_layout_put( GtkLayout *layout,
                     GtkWidget *widget,
                     gint       x,
                     gint       y );
```

将构件widget移动到布局容器构件中坐标为x、y的位置：

```
void gtk_layout_move( GtkLayout *layout,
                      GtkWidget *widget,
                      gint       x,
                      gint       y );
```

布局容器构件的尺寸可以用以下函数指定：

```
void gtk_layout_set_size( GtkLayout *layout,
                          guint      width,
                          guint      height );
```

布局容器构件是Gtk构件中一种会重绘自身的构件，当使用上面的函数改变了构件内的对象时，它会在屏幕上重绘自身。当想对布局容器构件做较大的变化时，可以用下面的函数“冻结”或“解冻”该构件，这样会禁止或启用布局容器构件重绘自身，能防止窗口闪烁。

“冻结”布局容器构件：

```
void gtk_layout_freeze( GtkLayout *layout );
```

“解冻”布局容器构件：

```
void gtk_layout_thaw( GtkLayout *layout );
```

最后4个函数用于操纵垂直和水平的调整对象。

取得布局构件的水平调整对象：

```
GtkAdjustment* gtk_layout_get_hadjustment( GtkLayout *layout );
```

取得布局构件的垂直调整对象：

```
GtkAdjustment* gtk_layout_get_vadjustment( GtkLayout *layout );
```

设置布局构件的水平调整对象：

```
void gtk_layout_set_hadjustment( GtkLayout     *layout,
                                 GtkAdjustment *adjustment );
```

设置布局构件的垂直调整对象：

```
void gtk_layout_set_vadjustment( GtkLayout     *layout,
                                 GtkAdjustment *adjustment);
```