NCURSES Programming HOWTO
中文版

（简体中文）

NCURSES Programming HOWTO 中文版（简体中文）

翻译说明：

原作：Pradeep Padala
来源：The Linux Documentation Project（http://www.tldp.org/）
时间：2003 年 3 月 23 日
译者：拜伦（Byron）
E-mail：byronm@sina.com

译者说明：
鉴于国内对 Linux 和 Open Source（开放源代码）运动的需求渐渐升温，也为了普及 Linux 环境下编程，我选择了这份 NCURSES-HOWTO 文档进行写译工作。
NCURSES 库是由原 CURSES 库发展而来的，是（New CURSES）库的缩写形式，这个库提供了 C/C++对屏幕以及光标、鼠标访问的特性。使得由 C/C++简易写屏成为可能。由于 NCURSES 库提供了窗口等特性，使得 NCURSES 的用户群不断上升，这也是一个可喜的趋势。
这是一份关于如何使用 NCURSES 库编写程序的 HOWTO 文档，文档来源是 The Linux Documentation Project 的官方网站。这份文档详细的介绍了 NCURSES 库的使用。同时也是一份 NCURSES 教程，可以是你更快的进入到 NCURESE 和 Unix 的编程世界。
译者在翻译的时候，为了使译文更加符合中文语言环境的读者。作者对原作中的部分章节和语句进行了调整。

关于 NCURSES 库的更多的文档和软件都在作者自己的网站可以下载得到。
作者的个人主页：http://www.cise.ufl.edu/~ppadala/ncurses/NCURSES_HOWTO/
作者的电子邮件 p_padala@yahoo.com
或者联系译者：
译者的个人主页：http://poet.cosoft.org.cn/index.htm
译者的电子邮件：byronm@sina.com

项目成员：（按加入项目时间先后）
Byron（byronm@sina.com）《NCURSES Programming HOWTO 中文版（简体中文）》的项目发起人，负责了 1-15，19、20 章的翻译工作。
Simon Zhan（simonzhan@21cn.com）负责了所有章节翻译原稿的审校工作。
DreamXST（xstsky@163.com）负责了 16-18 章的翻译工作和 HTML 格式文档的排版工作。

在此，我谨代表我个人向 **Simon Zhan** 和 **DreamXST** 的工作表示十万分感谢。正是他们的工作，才有了今天这份文档中文版的诞生。也算是我们对中国开源社区做出的小小的贡献，如果这部文档能使您受益、那将是我们莫大的荣幸！
此外我要感谢所有关注这个项目的网友们。他们是（排名不分先后）：毛氏更伟、Posuring、心蓝、cizi、古岳、特洛伊、Li Yewang、iceiceice 和 Candy

如果发现了文档中的错误或者对此中文文档有任何的问题和建议，欢迎联系译者。
此份文档有配套的示例程序，可以在作者或者译者的个人站点下载。

翻译正文：

NCURSES Programming HOWTO 中译版

Pradeep Padala
p_padala@yahoo.com

V1.7.1 2002 年 6 月 25 日

版本修订历史：

版本：1.7.1　　修订日期：2002 年 6 月 25 日　　修订人：P·Padala
修订说明：为源代码建立的修订和说明加入 README 文件。

版本：1.7　　修订日期：2002 年 6 月 25 日　　修订人：P·Padala
修订说明：添加“其它格式”部分，并且给示例程序增添了许多有趣的特性。

版本：1.6.1　　修订日期：2002 年 2 月 24 日　　修订人：P·Padala
修订说明：移除了旧的 Changelog 部分，清理了 Makefile。

版本：1.6　　修订日期：2002 年 2 月 16 日　　修订人：P·Padala
修订说明：修改了很多的拼写错误，添加 ACS 变量部分

版本：1.5　　修订日期：2002 年 1 月 05 日　　修订人：P·Padala
修订说明：更改了 TOC 的结构

版本：1.3.1　　修订日期：2001 年 7 月 26 日　　修订人：P·Padala
修订说明：修正了“维护人员”段落，校订了稳定的发布版本。

版本：1.3　　修订日期：2001 年 7 月 24 日　　修订人：P·Padala
修订说明：给主文档添加了版权信息（LDP 许可），给程序添加了版权信息（GPL 许可）
修正了 printw 示例程序（printw_example）的错误

版本：1.2　　修订日期：2001 年 6 月 5 日　　修订人：P·Padala
修订说明：合并了 ravi 的改动。主要是介绍部分、菜单部分、表单部分和“Just For Fun”部分。

版本：1.1　　修订日期：2001 年 5 月 22 日　　修订人：P·Padala
修订说明：添加了“a word about window”部分，增添了 scanw 示例程序（scanw_example）。

这份文档被改写成了一份关于 NCURSES 库以及它的姊妹库的完整的编程参考。在这份文档中我们将以一个用 NCURSES 库编写的“Hello, World!”程序作为开始。循序渐进的讲解 NCURSES 库中更多的、更复杂的操作。所以在撰写这份编程指南的时候假定读者从未接触过 NCURSES 库。最新版本 NCURSES 库和 NCURSES 的文档都可以在作者的个人主页（http://www.cise.ufl.edu/~ppadala/ncurses/NCURSES_HOWTO/）找到。如果有任何的意见和建议也可以直接和作者（mailto:p_padala@yahoo.com）联系。

NCURSES Programming HOWTO 版权声明

NCURSES Programming HOWTO 中译版（简体中文）版权声明

第一章：NCURSES 库简介

1、介绍

在广泛使用电传打字机的那个年代，电传打字机作为输出终端，通过电缆和中央电脑连接。用户要向终端程序发送一系列特定的控制命令，才可以控制终端屏幕的输出。

比如要在改变光标在屏幕上的位置，清除屏幕某一区域的内容，卷动屏幕，切换显示模式，给文字添加下划线，改变字符的外观、颜色、亮度等等。这些控制都是通过一种叫做转义序列（escape sequence）的字符串实现的。叫做转义序列是因为这些连续字节都是以一个“0x1B”字符，即转义字符（按下 ESC 键所输入的字符）作为开头的字符串。即使在现在，我们也可以通过向终端仿真程序输入转义序列得到与当年电传打字终端同样的输出效果。

如果你想在终端（或者终端仿真程序）屏幕输出一段背景是彩色的文字，可以将以下这段转义序列输入到你的命令行提示符：

```
echo "^[[0;31;40mIn Color"
```

在这里 “^” 和 “[” 就是所谓的转义字符。（注意：在这里 “^[” 是一个字符。不是依次键入“^”和“[”字符。要打印出这个字符，你必须先按下 Ctrl+V，然后按下 ESC 键。）执行以上的命令后。你应该可以看见“In Color”的背景变为红色了。从此以后显示的文本信息都是以这样的效果输出的。

如果想终止这种效果并回到原来的方式可以使用以下的命令：

```
echo "^[[0;37;40m"
```

现在知道这些字符（转移序列）的作用了吗？（译者注：更改分号之间的参数，看看会有什么结果。）也许会和自己想像的不一样？可能是因为终端环境不同，终端环境取决于终端或操作系统的不同。（你不可能让一个黑白终端显示出彩色字符吧？）

为了避免这种不兼容情况的发生，能够在不同的终端上输出统一的结果。UNIX 的设计者发明了一种叫做 termcap 的机制。termcap 实际上是一个随同转义序列共同发布的文件。这个文件罗列出当前终端可以正确执行的所有转义序列，使用户输入转移序列的执行结果符合这个文件中的规定。但是，在这种机制发明后的几年中，一种叫做 terminfo 的机制逐渐取代 termcap。从此用户不用在编程时翻阅繁琐的 termcap 中的转义序列规定，仅需要通过访问 terminfo 的数据库就可以控制屏幕的输出了。

2、NCURSES 是什么？

你可能被以上的术语搅和的晕头转向了。假设在使用 terminfo 的情况下，让所有的应用程序访问 terminfo 数据库控制输出（比如发送控制字符，等等……）。不久这些调用代码将会使整个程序变得难以控制和管理。这些问题的出现导致了 CURSES 的诞生。CURSES 的命名是来自一个叫做 “cursor optimization”（光标最优化）的双关语。CURSES 库通过对终端原始控制代码（转义序列）的封装，向用户提供了一个灵活高效的 API（应用程序接口）。它提供了一套控制光标，建立窗口，改变前景背景颜色以及处理鼠标操作的函数。使用户在字符终端下编写应用程序时绕过了那些恼人的底层机制。

NCURSES 是一个从 System V Release 4.0 (SVr4) 中 CURSES 的克隆。这是一个可自

由配置的库，完全兼容旧版本的 CURSES。简而言之，他是一个可以使应用程序直接控制终端屏幕显示的库。当后面提到 CURSES 库的时候，同时也是 NCURSES 库。

NCURSES 包由 Pavel Curtis 发起，Zeyd Ben-Halim <zmbenhal@netcom.com>和 Eric S. Raymond <esr@snark.thyrsus.com>是最初的维护人员，他们在 1.8.1 及以后版本中增加了很多的新功能。Jürgen Pfeifer< juergen.pfeifer@gmx.net >撰写了菜单库和面板库。使菜单和面板机制同 Ada95 中捆绑的一样好。Thomas Dickey<dickey@herndon4.his.com>和 Jürgen Pfeifer 进行着 NCURSES 的维护工作。Florian La Roche <florian@gnu.org>持有 NCURSES 的版权，他在自由软件基金会（GNU）中担任 NCURSES 的维护工作。如果要联系现在的 NCURSES 的维护者请发电子邮件至 bug-ncurses@gnu.org。

3、NCURSES 可以做什么？

NCURSES 不仅仅只是封装了底层的终端功能，而且提供了一个相当稳固的工作框架（Framework）用以产生漂亮的界面。它包含了一些创建窗口的函数。而它的姊妹库 Menu、Panel 和 Form 则是对 CURSES 基础库的扩展。这些库一般都随同 CURSES 一起发行。我们可以建立一个同时包含多窗口（multiple windows）、菜单（menus）、面板（panels）和表单（forms）的应用程序。窗口可以被独立管理，例如让它卷动（scrollability）或者隐藏。

菜单（Menus）可以让用户建立命令选项，从而方便执行命令。而窗体（Forms）允许用户建立一些简单的数据输入和显示的窗口。面板（Panels）是 NCURSES 窗口管理功能的扩展，可以用它覆盖或堆积窗口。

以上这些就是 NCURSES 的一些基本介绍。在以后的章节里，我们将详细的介绍 NCURSES 库。

4、在哪能得到它（Where to get it？）

你现在应该知道 NCURSES 可以做什么了。在你使用之前必须先下载它。你在安装操作系统时（Linux）应该已经附带了。万一如果你的操作系统还没有安装 NCURSES 库，可以通过以下的途径可以下载得到：

下载并编译安装文件包：
你可以通过 ftp://ftp.gnu.org/pub/gnu/ncurses/ncurses.tar.gz 免费下载 NCURSES。也可以通过 GNU 的 FTP 目录：http://www.gnu.org/order/ftp.html 找到提供免费下载 NCURSES 文件包的站点。最新发布的稳定版本为：5.2 20001021.（译者翻译时已经有 5.4 版本的 NCURSES 下载了）

包裹文件中的 README 和 INSTALL 文件是安装 NCURSES 库的最主要资料。通常是这样安装 NCURSES 的：

```
tar zxvf   ncurses<version>.tar.gz      #解压缩并且释放 文件包
cd ncurses<version>                     #进入解压缩的目录（注意版本）
./configure                             #按照你的系统环境制作安装配置文件
make                                    #编译源代码并且编译 NCURSES 库
su root                                 #切换到 root 用户环境
make install                            #安装编译好的 NCURSES 库
```

使用 RPM 安装文件：
可以在 http://rpmfind.net 找到 NCURSES 的 RPM 格式安装包。可以在 root 模式下使用以下的命令安装：（以 5.2 版本为例）

rpm –vih 下载的 RPM 文件

1.4 本文档的写作意图和涵盖范围（Purpose/Scope of the document）

这份文档是一份完整描述 NCURSES 库以及其姊妹库（sister libraries）的编程参考。在这份文档中我们将以一个用 NCURSES 库编写的“Hello, World!”程序作为起点。循序渐进的讲解 NCURSES 库中更多更复杂的操作。所以在撰写这份文档的时候假定读者不了解 NCURSES 库。

1.5 关于文档中出现的程序（About the Programs）

这份文档中用到的所有程序已经被压缩成一个 tar.gz 的文件(需要相关软件解压缩)。可以通过 http://www.linuxdoc.org/HOWTO/NCURSES-Programming-HOWTO/ncurses_programs.tar.gz 下载到。以下是这个压缩包解压缩后的目录结构：

```
ncurses
    |
    |----> JustForFun          -- just for fun 部分的程序
    |----> basics              -- 一些基础部分的程序
    |----> demo                -- make 之后的程序
    |          |
    |          |----> exe      -- 一些已编译的可执行示例程序
    |----> forms               -- 和 form 库相关的程序
    |----> menus               -- 和 menus 库相关的程序
    |----> panels              -- 和 panels 库相关的程序
    |----> Makefile            -- 一级目录的下的 Makefile
    |----> README              -- 一级目录下的 README 文件包含程序说明。
    |----> COPYING             -- 程序版权信息文档。
```

这些目录下包扩以下的文件：

```
JustForFun
    |
    |----> hanoi.c             -- 汉诺塔示例
    |----> life.c              -- 生命演示游戏
    |----> magic.c             -- 一个魔方程序
    |----> queens.c            -- 八皇后问题
    |----> shuffle.c           -- 一个非常有意思的程序，只要你有时间去……
    |----> tt.c                -- 一个非常简单的打字练习程序

  basics
    |
    |----> acs_vars.c          -- 可选字符（ACS）变量示例
```

```
   |----> hello_world.c            -- 简单的“Hello, World!”程序
   |----> init_func_example.c      -- 初始化函数示例
   |----> key_code.c               -- 显示键盘字符代码的程序
   |----> mouse_menu.c             -- 一个可以使用鼠标访问的菜单
   |----> other_border.c           -- 展示与 box()函数不同的显示边框的其它函数
   |----> printw_example.c         -- 一个非常简单的使用 printw()函数的例子
   |----> scanw_example.c          -- 一个非常简单的使用 getstr()函数的例子
   |----> simple_attr.c            -- 一个在屏幕上打印 C 源程序注释的例子
   |----> simple_color.c           -- 一个简单的演示颜色的例子
   |----> simple_key.c             -- 一个可以用方向键访问的菜单的例子
   |----> temp_leave.c             -- 一个演示临时离开 CURSES 模式的例子
   |----> win_border.c             -- 展示窗口和边框的例子
   |----> with_chgat.c             -- chgat()函数使用的例子

forms
   |
   |----> form_attrib.c            -- 展示 field 属性的用法
   |----> form_options.c           -- 展示 field 选项的用法
   |----> form_simple.c            -- 一个简单的表单例子
   |----> form_win.c               -- 一个简单的窗口和表单联合使用的例子

menus
   |
   |----> menu_attrib.c            -- 展示菜单属性的用法
   |----> menu_item_data.c         -- 展示 item_name() 等等函数的用法
   |----> menu_multi_column.c      -- 建立多列菜单
   |----> menu_scroll.c            -- 展示菜单滚动的示例
   |----> menu_simple.c            -- 一个用方向键控制菜单的例子
   |----> menu_toggle.c            -- 建立多值菜单和解释 REQ_TOGGLE_ITEM
   |----> menu_userptr.c           -- 用户指针的用法
   |----> menu_win.c               -- 菜单和窗口结合的演示例子

panels
   |
   |----> panel_browse.c           -- 通过 tab 浏览展示用户指针的用法
   |----> panel_hide.c             -- 隐藏和取消隐藏面板的例子
   |----> panel_resize.c           -- 移动和改变面板大小的例子
   |----> panel_simple.c           -- 一个简单的面板使用例子
```

Makefile 包含在一级主目录里。它将会把所有的程序编译成可执行的文件。并把这些文件存在 demo/exec 目录下面。你也可以选择将其编译到别的目录下。每个目录下都有一个 README 文件详细描述了每个目录下的 C 源程序的内容。

对于每一个示例，我都给出了这些程序调用 NCURSES 目录下相关文件的路径名。

如果你希望在线阅读这些程序中某个单独的程序，可以通过浏览器访问一下网址：
http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/ncurses_programs/
所有以上发布的程序都遵照 GPL（http://www.gnu.org/copyleft/gpl.txt）许可协议发行，你可以在许可协议允许的范围内任意处置程序。

1.6 关于此文档的其他文件发布格式(Other Formats of the document)

这份 HOWTO 文档还有很多其它的文件发布格式，可以在 tldp.org 下载到。以下是关于这份文档其它格式的下载链接：

1.6.1 在 tldp.org 上的已发布格式链接：

Acrobat PDF 格式
PostScript 格式
多级 HTML 文档
只有一页的 HTML 文档

1.7 原文贡献者 （Credits）

感谢 Sharath（sharath_1@usa.net）和 Emre Akbas 编写了其中的一些段落。简介部分最初由 Sharath 写成。当重新写这部分的时候饮用了他最初完成的部分。Emre 帮忙编写了 printw()函数和 scanw()函数部分。

然后是 Ravi Parimi（parimi@ece.arizona.edu）。他是这个项目最初的实现者。在撰写这份文档时，他对这份文档提供了很多的意见。并且在文章撰写完毕后耐心的校对了整份文档。他还在 Linux 和 Solaris 平台上检查过文中使用的每一个程序。你可以从他的注记中发现你的问题。

1.8 远景规划（Wish List）

这些是正在进行或者将要进行的项目。如果你有项目或者想加入某个项目。请和 Padala 联系。

- 给最后讲述 form 库的一章增加更多的示例程序。（这个项目 Padala 正在进行）
- 准备一个演示程序展示文档中所有的示例。同时允许用户通过程序描述展示这些示例。让用户亲自编译并且察看这些程序。一个有对话框的用户界面更好。（这个项目 N.N.Ashok 正在进行）
- 加入 Debug 信息，使用 _trace，_tarcemouse 。
- 通过 NCURSES 库中的函数访问 termcap 或 tremios
- 使用户可以同时在两个终端上工作。
- 在“其它特色”章节（Miscellaneous features）中增加一些东西。

1.9 关于著作权（Copyright）

第二章：从 Hello World 程序开始

欢迎来到 NCURSES 的世界。在我们踏上学习 NCURSES 的路途之前，让我们先写一个小程序，来了解一下 NCURSES 的执行结构。

2.1 编译包含 NCURSES 函数的程序

如果要调用 NCURSES 库中的函数，你必须在代码中加载 ncurses.h 文件，就是在 C 或 C++ 程序中添加“#include <ncurses.h>”这一行。然后在连接程序中标记出 NCURSES。（注：NCURSES 库已经包含了“stdio.h”）

```
#include <ncurses.h>
。
。
。
编译和连接命令: gcc <程序文件> -lncurses
```

例 1：Hello World !!!程序

```
#include <ncurses.h>

int main()
{
    initscr();                  /* 初始化，进入 NCURSES 模式                */
    printw("Hello World !!!");  /* 在虚拟屏幕上打印 Hello, World!!!         */
    refresh();                  /* 将虚拟屏幕上的内容写到显示器上，并刷新   */
    getch();                    /* 等待用户输入                             */
    endwin();                   /* 退出 NCURSES 模式                        */
    return 0;
}
```

2.2 示例剖析

这个程序在显示器屏幕上打印“Hello World !!!”后等待用户按任意键退出。这个小程序展示了如何初始化并进入 curses 模式、处理屏幕操作和退出 curses 模式。让我们逐行的分析这个小程序：

2.2.1 关于 initscr() 函数

initscr()函数将终端屏幕初始化为 curses 模式。它用来清除屏幕上所有的字符，使屏幕变为空白，等待下一部处理。所以在调用其它的 NCURSES 函数前，都要先调用 initscr()函数初始化屏幕。这个函数初始化了 curses 系统并且为当前屏幕，也就是一个叫做被“stdscr”的窗口以及其它的数据结构分配内存。在以前的计算机上曾经出现过一个非常极端的例子：因为系统中的可用内存太小，以至于 initscr()函数无法分配足够的内存给相关的数据结构，从而导致 curses 系统初始化失败。我们将在以后的章节中详细介绍如何定制屏幕的初始化模式。

2.2.2 神秘的 refresh() 函数

第二行的 printw 函数用于把字符串“Hello，World！！！”输出到虚拟的屏幕上。这个函数用法上和 printf()函数很像。但是区别在于，printw 函数把字符串输出到被称作“stdscr”的虚拟坐标（0,0）上。从显示的结果来看，坐标（0,0）在屏幕的左上角上。

现在该说说这个神秘的 refresh()函数了。在我们使用 printw 函数打印“Hello World！！！”时，实际上这个数据被打印到一个叫作“stdscr”的虚拟窗口上，没有被直接输出到屏幕上。printw()函数的作用是不断将一些显示标记和相关的数据结构写在虚拟显示器上，并将这些数据写入 stdscr 的缓冲区内。所以，为了显示这些缓冲区中的数据我们必须使用 refresh（）函数告诉 curses 系统将缓冲区的内容输出到屏幕上。

这种机制可以使程序员不断在虚拟屏幕上写入数据，而调用 refresh（）函数时让一切看起来似乎是一次完成的。因为 refresh（）函数只核查窗口和数据中变动的部分，这种富有弹性的设计提供了一个高效的反馈机制。但是这有时很打击初学者的积极性。因为对于初学者来说忘记在输出后调用 refresh()函数是很恼人的错误。不过不用担心，因为很多人也经常犯这样的错误。

2.2.3 关于 endwin()函数

最后，别忘了要结束 curses 显示模式才可以返回到普通字符行模式。否则，在程序结束后你的终端可能会运转得不正常。endwin()函数释放了 curses 子系统和相关数据结构的内存，使你的终端回到普通字符模式。这个函数必须是在你完成所有的 curses 操作以后才可以调用。（译者注：如果你在 endwin()函数后再调用 curses 的函数。很显然，那些语句不会执行。说到这里，顺便提醒一下。如果你的程序不能正常地显示东西了。请务必看看 initscr()函数和 endwin()函数是不是在不该被调用的地方调用了。）

第三章：真正的起航

现在我们知道如何编写一个简单的 CURSES 程序，也知道了一个 CURSES 程序都需要哪些部分。接下来我们就要正式的进入 CURSES 高级功能的学习了。那里有很多的函数能够帮你定制出你自己需要的显示效果。也许会让你沮丧、也许会让你兴奋……但不管怎么样，好戏才刚刚开始。

那么，让我们开始吧……

第四章：初始化

我们现在知道了：在程序中调用 initscr()函数，会让屏幕初始化并进入 CURSES 模式。还有一些其它的函数可以根据我们自己的方案初始化 CURSES。不同的初始化函数可以让屏幕进入不同的显示模式。比如：终端模式（terminal mode）、彩色模式（color mode）、鼠标模式（mouse mode）等等……当然，我们还可以将这些模式混合起来……那么，让我们马上来看看一些经常使用的初始化函数：

4.1 raw()和 cbreak()函数

通常情况下，用户输入的字符将被终端程序送入终端的缓冲区。但当用户输入换行符时，终端程序将会中断，同时输出当前的缓冲区内容并启用新行的输出缓冲。但是大多数程序需要当用户输入单个字符时，却希望这些字符能够立即显示在屏幕上。这两个函数就是用来禁用行缓冲（line buffering）。这两个函数所初始化的的模式同样可以用来给程序传送控制字符，比如：挂起（CTRL-Z）、中断或退出（CTRL-C）。区别在于，在 raw()函数模式下，这些字符将传送给程序去处理而不作为终端程序处理的信号。在 cbreak()模式下，这些控制字符将被认为是终端驱动程序中的控制字符，因而将这些字符传送给终端程序。我比较喜欢使用 raw()，那样可以进行更多的控制操作。

4.2 echo()和 noecho()函数

这两个函数控制用户输入的键盘回显。就是在运行程序的时候是否将输入的字符出现在屏幕上。比如你的程序在运行时你需要使用控制字符，但是你不想让控制字符出现在屏幕上，就可以使用这两个函数。也就是说当用户调用 getch()函数向程序输入数据时，你不想让他输入的字符出现在屏幕上。noecho()函数就可以不让控制字符（比如 CTRL-C）出现在屏幕上。大多数的交互式程序要进入控制模式的时候，一般都使用 echo()、noecho()函数初始化、关闭键盘回显。这样给了程序员更大的灵活性。

4.3 keypad()函数

这个函数允许使用功能键：F1、F2、方向键等等……几乎所有的交互式程序都使用这个函数。令用户使用方向键控制整个用户界面。使用 keypad(stdscr,TURE)就可以在“标准显示设备”（stdscr）上使用这些功能。在以后的章节中将详细讨论如何使用功能键。

4.4 halfdelay()函数

这个函数虽然不经常使用。但有时却非常有用。halfdelay()函数会启用**半延时模式**（half-delay mode）。和 cbreak()函数一样，当程序需要当用户输入这些字符时，它们能够立即显示在屏幕上。但是它要停滞一段限定时间（以 0.1 秒为单位）等待输入，如果没有有效的输入，返回 ERR。给 halfdelay()传递一个整型参数（以 0.1 秒为单位）。它就会按照参数中的时间等待用户输入。一般来说，这个函数在需要等待的输入的程序中可以被用到，如果用户没有及时做出响应，程序就可以去处理其它的事情了。最常见到的应用例子是在输入密码的出超时响应。

4.5 其它的初始化函数

上面的提到的这些函数可以定制 CURSES 在初始化后的行为。这些函数不能被广泛使用在程序的各个部分。所以，这些函数的调用要处在整个 CURSES 会话的开始部分。

4.6 一个例子

让我们写一个程序用来说明这些函数的用法。

例 2. 初始化函数用法的示例：

```
#include <ncurses.h>

int main()
{
    int ch;

    initscr();                  /* 开始 curses 模式          */
    raw();                      /* 禁用行缓冲                */
    keypad(stdscr, TRUE);       /* 开启功能键响应模式        */
    noecho();                   /* 当执行 getch()函数的时候关闭键盘回显 */
    printw("Type any character to see it in bold\n");
    ch = getch();
    /* 如果没有调用 raw()函数，必须按下 enter 键才可以执行下面的程序 */

    if(ch == KEY_F(1)) /* 如果没有调用 keypad（）初始化，将不会执行这条语句 */
        printw("F1 Key pressed");
    /* 如果没有使用 noecho() 函数，一些难看的控制字符将会被打印到屏幕上 */

    else
    {   printw("The pressed key is ");
        attron(A_BOLD);
        printw("%c", ch);
        attroff(A_BOLD);
    }
    refresh();          /* 将缓冲区的内容打印到显示器上 */
    getch();            /* 等待用户输入                 */
    endwin();           /* 结束 curses 模式             */
    return 0;
}
```

这个程序很简单，不需要太多的说明。但还是有一些前面没有提到的函数。getch()函数用来取得用户输入的信息。它和通常的 getchar()函数相似。但是 getch（）可以让我们在禁用行缓冲时避免在输入完成后还要按 enter 键。在后面的章节中我们也将讨论到这些输入函数。attron()和 attroff()函数作为切换开关用来开启和关闭字符的修饰效果。在这个例子中，它们使显示的字符字体加粗。在后面的章节中我们也将讨论到这些函数。

第五章：窗口机制简介

在我们进入数以万计的 CURSES 函数以前，让我们先了解一下窗口（windows）的知识。关于窗口机制的详细信息我们会在以后的章节中详细介绍。这一章只是一些基础的知识，为了给讲解输入输出函数作铺垫。当然，在现在就树立一个窗口的概念是很有好处的。

窗口实质上是由 CURSES 系统定义的一个假想的屏幕。这种窗口并不像 Windows 平台上的窗口，这些窗口没有边框。当 CURSES 初始化的时候，它会默认创建一个叫做 stdscr 的窗口。这个窗口的大小一般是 80 列，25 行的屏幕（根据显示器或者显卡的不同，可能会出现不同的大小）如果你运行简单的任务，比如打印几个字符串、输入一些数据等等……这样的单窗口完全可以满足你的需要。当然，你也可以通过窗口系统的函数创建你自己的窗口。

举个例子，如果你调用以下函数：

```
printw("Hi! There!");
refresh();
```

它会在 stdscr 上的当前光标位置输出“Hi! There!”。同样，调用 refresh()函数，它只更新 stdscr 上的缓冲区。

例如，你已经建立了一个叫做 win 的窗口。你要在这个窗口上输出以上的内容。只需要在普通的函数前添加 w 前缀就可以了。同时，函数中的参数也要相应的变化（要指明你所要显示信息的窗口）：

```
wprintw(win, "Hi There !!!");
wrefresh(win);
```

你将在这份文档的其余部分看到，这些函数有相同的命名规则。

```
printw(string);              /* 在 stdscr 的当前光标位置打印字符串 string */
mvprintw(y, x, string);      /* 将字符串 string 打印在坐标(y,x)处 */
wprintw(win, string);        /* 在窗口 win 的当前光标位置打印字符串 string */
mvwprintw(win, y, x, string);
/* 将光标移动到窗口 win 的(y,x)然后打印字符串 string */
```

没有 w 前缀的函数在调用时被扩展成以 stdscr 作为当前窗口的函数。

第六章：一些类似 printw()的输出函数

我猜你已经再也无法忍受对 CURSES 那些粗泛的了解了。我们已经通过一些例子，对 CURSES 系统及其函数有了基本的认识。从现在开始，要正式进入精彩的 curses 世界了。

在 CURSES 函数中有三类打印字符的函数，它们是：

addch()类函数集：将单一的字符打印到屏幕上，可以加字符修饰的一类函数。
printw()类函数集：和 printf()一样的具有格式化输出的一类函数。
addstr()类函数集：用于打印字符串的一类函数。

这几类函数可以交替的使用。关键问题是了解某一类函数具体的应用环境。让我们看看这几类函数之间使用的区别。

6.1 addch()类函数

addch()类函数用于在当前光标位置输入单个字符。并将光标移动到下一个位置。你可以使用这个函数打印一个字符。但在大多数情况下，这些函数可以顺便修饰这些字符。后面的章节我们会详细介绍**输出修饰**。如果一个字符使用了一种修饰效果（比如：粗体、反显……）当 CURSES 打印这个字符的时候就会应用这种字符修饰效果。

给字符关联属性有两种方法：

- 用**属性宏（通过“或”运算）**修饰单一字符。这些宏可以在头文件 ncurses.h 中找到。比如，你想使一个字符变量 ch 在输出的时候使变量中相应的字符同时以加粗（BOLD）和加下划线（UNDERLINE）的效果输出，你可以使用下面这种方法：

  ```
  addch(ch | A_BOLD | A_UNDERLINE);
  ```

- 使用像 attrset()、attron()、attroff()这样的**修饰函数**修饰。这些函数将在**文字修饰**一章介绍。简而言之，它们用来修饰当前窗口中的所有内容。一旦设置完成（attrset()函数被调用），则在相应窗口中输出的字符全部都会被修饰。直到关闭修饰（调用 attroff()函数）为止。

CURSES 又提供了一些可以在字符模式下作简单图形的特殊字符。你可以用它们绘制表格、或者水平、垂直线条等等。你可以在头文件 ncurses.h 中找到所有这些特殊字符。这些特殊字符都是以 ACS_作为开头声明的宏。

6.2. mvaddch(), waddch() 和 mvwaddch()函数

mvaddch()被用来将光标移动到指定位置然后输出字符。因而，下面的函数调用：

```
move(row,col);      /*将光标移动到 row 所指定的行和 col 所指定的列。*/
addch(ch);
```

可以用以下的等价函数取代：

```
mvaddch(row,col,ch);
```

waddch()函数和 addch()函数类似。区别在于，waddch()函数将字符输出到指定给定窗口的指定坐标处。（注：addch()将字符输出到标准输出 stdscr 上。）

同样的 mvwaddch()函数是把光标移动到指定窗口中的指定位置然后输出字符。

现在，我们了解了一些基本的输出函数。但是，如果我们要输出字符串，像这样一个字符一个字符输出是很烦人的。幸好，CURSES 为我们提供了和 printf()和 puts()一样方便的函数。

6.3. printw()类函数

这些函数的用法很像 printf()函数，但提供了可以在屏幕任意位置输出的能力。

6.3.1. printw()函数和 mvprintw()函数

printw()函数的用法绝大部分和 printf()函数相同。mvprintw()函数将光标移动到指定的位置然后打印内容。

6.3.2. wprintw() 函数和 mvwprintw 函数

这两个函数和以上两个函数类似。区别在于这两个函数将在指定的窗口输出的内容。而以上两个函数将内容输出到标准输出 stdscr 上。

6.3.3. vwprintw()函数

这个函数和 vprintf()相似。它用于打印变量表中所对应的变量。

6.3.4.一个简单的 printw()函数的使用例子（A Simple printw example）

例 3：一个简单的 printw()函数的使用例子

```
#include <ncurses.h>                    /* ncurses.h 已经包含了 stdio.h */
#include <string.h>

int main()
{
  char mesg[]="Just a string";          /* 将要被打印的字符串 */
  int row,col;                          /* 存储行号和列号的变量，用于指定光标位置
*/
  initscr();                            /* 进入 curses 模式 */
  getmaxyx(stdscr,row,col);             /* 取得 stdscr（标准输出设备）的行数和列数 */
  mvprintw(row/2,(col-strlen(mesg))/2,"%s",mesg);
 /*在屏幕的正中打印字符串 mesg*/
 mvprintw(row-2,0,"This screen has %d rows and %d columns\n",row,col);
 printw("Try resizing your window(if possible) and then run this program again");
 refresh();
 getch();
 endwin();

 return 0;
}
```

这个程序示范了 printw()类函数的一般用法，你只需要修改其中的行列坐标和要打印的字符串，这个程序就可以做你让它显示的。

这个程序引入了一个新函数 getmaxyx()。这是一个定义在 ncurses.h 中的宏，它会给出当前终端屏幕的行数和列数。所以，在窗口初始化更改窗口大小数据的时候它也会随着一起改变。getmaxyx()不是一个函数，因此我们不能向它传递指针，只可以传递一对整型变量。

6.4. addstr()类函数

addstr()函数用于输出字符串到指定窗口。这些函数的使用和 addch()函数的使用差不多，差别在于输出对象不是单个字符而是字符串。其它的扩展函数分别是 mvaddstr()、mvwaddstr()和 waddstr()，它们有着相同的 CURSES 命名规则和调用方法。（比如 mvaddstr() 函数就是分别调用了 move()和 addstr()函数）这个函数集中还有一个特殊函数 addnstr()，这个函数需要一个整型参数。用来打印字符串中和这个整型参数数目相同的字符。如果这个参数是负数，addnstr()将会打印整个字符串。

6.5.一些提醒

所有这些函数中使用坐标的时候是先引入 y 坐标，再引入 x 坐标。也就是先行后列，因为计算机字符模式是逐行显示的。很多初学者因为数学上的使用习惯而使用了先 x 后 y 的方式。如果你使用了很多行列坐标的操作，想想看将屏幕分割成不同的窗口，然后处理每一个单独的窗口。窗口的详细内容会在**窗口**一章中介绍。

第七章：关于类似 scanw()的输入函数

如果在屏幕上只是打印而没有输入的话，那样的程序是索然无味的。所以让我们来看看那些处理用户输入的函数。这些函数也被分为三类：

getch()类函数集：读取一个字符的一类函数。
scanw()类函数集：按照格式化读取输入的一类函数。
getstr()类函数集：读取字符串的一类函数。

7.1. getch()函数集

这些函数将从终端读入一个字符。但是在使用它的时候需要考虑有些微妙的情况：比如你没有事先调用 cbreak()函数的话，CURSES 将不会连续读取你输入的字符，除非你输入了换行符或者文末符。为了避免这种情况的出现，在需要即时显示输入字符的程序中，必须先调用 cbreak()函数。另外一个被广泛使用的函数是 noecho()。这个函数被调用时，用户输入的字符不会将显示在屏幕上。cbeak()和 noecho()是两个典型的用于键盘管理的函数。这类的函数将在**键盘管理**一章中说明。

7.2. scanw()函数集

这些函数用法大体上和 scanf()函数相似。只不过加入了能够在屏幕的任意位置格式化输入字符串的功能。

7.2.1. scanw()函数和 mvscanw()函数

scanw()函数的用法和 sscanf()函数的用法基本相同。实际上，在调用 scanw()函数时，是调用了 wgetstr()函数，并将 wgetstr()函数处理的数据结果传送到一个 scanw()调用中。（wgetstr()函数将在下面详细介绍，写到这里是为了结构整齐。）

7.2.2. wscanw()函数和 mvwscanw()函数

这两个函数的用法和以上两个函数相似。区别在于它们从一个窗口中读取数据。所以，它们需要提供指定窗口的指针。

7.2.3. vwscanw()函数（vwscanw()）

这个函数和 vprintf()相似。它用于输入变量表中所对应的变量。

7.3. getstr()函数集

这些函数用于从终端取得字符串。本质上，连续使用 getch()函数直到读到一个换行符（或文末符）和这个函数使用的结果是一样的。当健入后它会创建一个指向这个字符串的指

针。

7.4. 例子

例 4：一个简单的使用 scanw()函数的例子。

```
#include <ncurses.h>                    /* ncurses.h 已经包含了 stdio.h */
#include <string.h>

int main()
{
      char mesg[]="Enter a string: ";  /* 将要被打印的字符串信息 */
      char str[80];
      int row,col;                      /* 存储行号和列号的变量，用于指定光标位置
     */
      initscr();                        /* 进入 curses 模式 */
      getmaxyx(stdscr,row,col);         /* 取得 stdscr 的行数和列数 */
      mvprintw(row/2,(col-strlen(mesg))/2,"%s",mesg);
     /* 在屏幕的正中打印字符串 mesg */

      getstr(str);                      /* 将指针 str 指向读取的字符串 */
      mvprintw(LINES - 2, 0, "You Entered: %s", str);
      getch();
      endwin();

      return 0;
}
```

第八章：输出修饰

我们已经通过一些例子看到了修饰（Attributes）输出文字的效果。给某些文字加上修饰会使文字更加醒目和美观。在某些程度上也会增加输出信息的可读性。下面这个程序将会把一个 C 语言的源程序文件的注释部分用使用粗体（BOLD）输出

例 5：一个简单的文字修饰的例子：

```
#include <ncurses.h>

int main(int argc, char *argv[])
{
  int ch, prev;
  FILE *fp;
  int goto_prev = FALSE, y, x;
  if(argc != 2)
    {    printf("Usage: %s <a c file name>\n", argv[0]);
         exit(1);
    }
  fp = fopen(argv[1], "r");          /* 在这里检测文件是否成功打开 */
  if(fp == NULL)
    {    perror("Cannot open input file");
         exit(1);
    }
  initscr();                         /* 初始化并进入 CURSES 模式 */
  prev = EOF;
  while((ch = fgetc(fp)) != EOF)
  {
    if(prev == '/' && ch == '*')/* 当读到字符“/”和“*”的时候调用开启修饰函
数 */
    {
         attron(A_BOLD);         /* 将“/”和“*”及以后输出的文字字体加粗
    */
         goto_prev = TRUE;
    }
    if(goto_prev == TRUE)        /* 回到“/”和“*”之前开始输出 */
    {
         getyx(stdscr, y, x);
         move(y, x - 1);
         printw("%c%c", '/', ch);      /* 实际打印内容的部分 */
         ch = 'a';                     /* 避免下次读取变量错误，这里赋一个任意值
*/
         goto_prev = FALSE;            /* 让这段程序只运行一次 */
    }
```

```
    else printw("%c", ch);
    refresh();              /* 将缓冲区的内容刷新到屏幕上 */
    if(prev == '*' && ch == '/')
    attroff(A_BOLD);    /* 当读到字符“*”和“/”的时候调用修饰关闭函数*/
    prev = ch;
  }
    getch();
    endwin();               /* 结束并退出 Curses 模式 */
    return 0;
}
```

不用去理睬那些初始化部分和其它没用的部分。把注意集中在上面的这个 while 循环体中。这个循环读取文件中每个字符并寻找有“/*”（注释起始处标志）的地方。一旦找到，就会调用 attron()函数开始为输出文字加粗加亮。当找到“*/”（注释结束处标志）的地方，就会使用 attroff()函数关闭修饰效果。

这个程序介绍了两个十分有用的函数：getyx()和 move()。getyx()函数其实是一个定义在 ncurses.h 中的宏，它会给出当前光标的位置。由于 getyx()不是一个通常所指的函数，因此我们不能向它传递指针，只可以传递一对整型变量（前文提到过）。函数 move()将光标移动到指定位置。（译者注：在这里再次强调：所有这些函数中使用行列坐标的时候是先行列后列。就是先写 y 坐标，再写 x 坐标。）很多初学者因为数学上的使用习惯而使用了先行后列的方式。**（在这里一定要注意！）**

这个程序执行的任务非常简单，无需作过多的说明。这个程序对于分析 C 程序十分有帮助。你也可以试着将输出文字的颜色改变为其它颜色。也可以将这个程序扩展为分析其它语言程序的工具。

8.1 详细介绍

让我们来更多的了解一下输出修饰。attron()函数、attroff()函数和 attrset()函数以及他们的姊妹函数（sister functions）比如 attr_get()等等……可以用这些函数创造出生动有趣的显示效果。

attron()函数和 attroff()函数分别用来开启（on）或关闭（off）输出修饰。以下这些修饰属性已经定义在头文件 CURSES.h 中。可以在函数中使用：

A_NORMAL	普通字符输出 (不加亮显示)
A_STANDOUT	终端字符最亮
A_UNDERLINE	下划线
A_REVERSE	字符反白显示
A_BLINK	闪动显示
A_DIM	半亮显示
A_BOLD	加亮加粗
A_PROTECT	保护模式
A_INVIS	空白显示模式

A_ALTCHARSET 字符交替
A_CHARTEXT 字符掩盖
COLOR_PAIR(n) 前景、背景色设置

（注：经过译者试验（见试验程序 lst01.c），上述修饰中只有 A_NORMAL、A_BLINK、A_DIM、A_INVIS、A_BOLD、A_STANDOUT 和 A_REVERSE 有效。且 A_STANDOUT 和 A_REVERSE 具有相同的输出结果。也许是笔者的试验环境问题，也许是笔者的这个测试程序还不完备。还请读者能够在自己所使用的系统上测试，欢迎把测试结果给我。如果你有更好的测试程序，我更加希望你能通过 E-mail 寄给我。）

最后一个修饰是最吸引人的，它可以设置输出的字符及背景的颜色。颜色的设置将在后面章节详细介绍。

我们可以给一段输出同时设定多种修饰。这样可以得到多种结合的效果。如果你想反白显示字符并同时让字符闪烁。只需要在两种修饰属性间加一个“|”字符就可以了：

```
attron(A_REVERSE | A_BLINK);
```

8.2 attron()函数和 attrset()函数之比较

现在我们该讨论讨论 attron()函数和 attrset()函数之间的区别了。attrset()为整个窗口设置一种修饰属性。而 attron()函数只从被调用的地方开始设置。所以 attrset()会覆盖掉你先前为整个窗口设置的任何修饰属性。就像在窗口的开始处开启修饰，在窗口结尾处关闭修饰属性一样。这两种方法为我们管理输出修饰属性提供了更简单、更富有弹性的方法。但是，如果你粗心的话，可能会让整个屏幕变得十分杂乱无章，函数之间的调用会难以管理。这种情况，尤其在某些利用到菜单的程序中出现的十分普遍。所以，事先做好设计，然后按照设计去实施会效果会好得多。你可以经常使用 standend()函数关闭所有设置的修饰。这个函数的作用和 attrset(A_NORMAL)函数是相同的。

8.3 attr_get() 函数

attr_get()函数用来取得当前窗口的修饰属性设置以及背景、文字颜色。虽然这个函数不像上面的那些函数常用。但它却对检测屏幕区域的修饰属性设置很有用。当我们在屏幕输出一些混合的复杂修饰效果时，这个函数可以告诉我们每一个字符关联的修饰。这个函数必须在设置了 attrset()或者 attron()函数之后使用。

8.4 attr_ 类函数

这些函数都有 attr_的前缀，比如：attr_set()、attr_on()等等……它们的作用和上面的函数一样，只不过要在调用时添加一定的参数。

8.5 wattr_ 类函数（wattr functions）

这些函数的作用范围是某一个指定的窗口。而上面的函数只作用在默认的 stdscr 上。

8.6 chgat() 函数

chgar()函数被列在 curs_attr 的 man_page 的最末尾。事实上它是一个很有用的函数。它可以在不移动光标位置的情况下修改已输出的字符的修饰效果。它从光标当前位置处开始，以一个整型参数作为修改字符的个数。给这些字符设置某一种修饰属性。
当我们整型参数赋值为-1 时，它代表一行修饰。如果你想从当前光标位置使整个一行的输出修饰变为反白显示时，可以这样使用：

```
chgat(-1, A_REVERSE, 0, NULL);
```

这个函数经常被用来修改已输出的字符的修饰。当然，你也可以根据自己的需要选择要修改的起始点和终止点。

这一类的函数还包括 wchgat(), mvchgat()。它们的使用方法和差不多 chgat()，只不过 wchgat()要指定一个窗口。mvchgat()将光标先移动到指定位置，然后才执行剩下的修饰部分。同样的，chgat()是一个宏，只不过用 stdscr 替代了指定的窗口。大部分不带 w 前缀的函数都是以 stdscr 作为默认窗口的宏。

例 6：mvchgat()用法示例：

```
#include <ncurses.h>

int main(int argc, char *argv[])
{
    initscr();                  /* 进入 curses 模式  */
    start_color();              /* 开启颜色管理功能  */

    init_pair(1, COLOR_CYAN, COLOR_BLACK);
    printw("A Big string which i didn't care to type fully ");
    mvchgat(0, 0, -1, A_BLINK, 1, NULL);
    /*
     *第一、二个参数表明了函数开始的位置
     *第三个参数是被改变修饰的字符的数目，-1 表示一整行
     *第四个参数是被改变的修饰名
     *第五个参数是颜色索引。颜色索引已经在 init_pair()中被初试化了
     *如果用 0 表示不使用颜色。
     *最后一个总是 NULL，没什么特别的意义。
     */
    refresh();
        getch();
    endwin();                   /* 结束 curses 模式   */
    return 0;
}
```

第九章：窗口机制

窗口（Window）机制是整个 CURSES 的核心概念。你应该已经通过前面的例子看到了所有的函数都是默认在输出“窗口”（stdscr）操作。即使如果你现在设计一个最简单的图形用户界面（GUI），你都需要用到窗口。使用窗口的一个最主要的原因是：通过窗口机制，你可以将屏幕分割为不同的部分，并且同时在不同的区域内分别操作。这样做的可以提高工作效率。另外一个原因是：你应当始终在你的程序中追求一种更好的、更易于管理的设计方式。如果你要设计一个大型的、复杂的用户界面，事先设计好这些部分将会提高你的办事效率。

9.1 基本概念

一个窗口的建立是通过一个叫做 newwin()的函数开始的。别以为你使用这个函数后会立即在屏幕上出现一个窗口。这个函数的作用是分配内存给窗口相关的数据结构。因此在 CURSES 里，窗口是一个假想的抽象概念。可以在屏幕中对这个假想中的窗口各个部分进行独立的控制。newwin()函数返回一个指向窗口的结构指针。这个指针可以被传送至一些类似于 wprintw()这样需要窗口参数的函数中。使用 delwin()函数可以删除一个窗口，并且释放存储窗口数据结构的内存和信息。

9.2 我要看见它！

有意思的是，我们创建了一个窗口却无法看见它。所以我们要做的是如何将窗口显示出来。box()函数可以用来在已经定义的窗口外围画上边框。让我们从以下这个例子中探索一下这些函数的详细信息。

例 7：带边框的窗口：

```
#include <ncurses.h>

WINDOW *create_newwin(int height, int width, int starty, int startx);
void destroy_win(WINDOW *local_win);

int main(int argc, char *argv[])
{   WINDOW *my_win;
    int startx, starty, width, height;
    int ch;

    initscr();                  /* 初始化并进入 curses 模式          */
    cbreak();                   /* 行缓冲禁止，传递所有控制信息  */
    keypad(stdscr, TRUE);       /* 程序需要使用 F1 功能键            */

    height = 3;
    width = 10;
```

```
    starty = (LINES - height) / 2;    /*计算窗口中心位置的行数 */
    startx = (COLS - width) / 2;      /*计算窗口中心位置的列数 */
    printw("Press F1 to exit");
    refresh();
    my_win = create_newwin(height, width, starty, startx);

    while((ch = getch()) != KEY_F(1))
    {   switch(ch)
        {   case KEY_LEFT:
                destroy_win(my_win);
                my_win = create_newwin(height, width, starty,--startx);
                break;
            case KEY_RIGHT:
                destroy_win(my_win);
                my_win = create_newwin(height, width, starty,++startx);
                break;
            case KEY_UP:
                destroy_win(my_win);
                my_win = create_newwin(height, width, --starty,startx);
                break;
            case KEY_DOWN:
                destroy_win(my_win);
                my_win = create_newwin(height, width, ++starty,startx);
                break;
        }
    }

    endwin();             /*结束 curses 模式      */
    return 0;
}

WINDOW *create_newwin(int height, int width, int starty, int startx)
{
    WINDOW *local_win;
    local_win = newwin(height, width, starty, startx);
    box(local_win, 0 , 0);          /* 0, 0 是字符默认的行列起始位置 */
    wrefresh(local_win);            /*刷新窗口缓冲，显示 box          */
    return local_win;
}

void destroy_win(WINDOW *local_win)
{/
* box(local_win, ' ', ' ');不会按照预期的那样清除窗口边框。 而是在窗口的四个角落留下残余字符*/
```

```
    wborder(local_win, ' ', ' ', ' ',' ',' ',' ',' ',' ');
    /*参数注解 9.3：
     * 1. win:当前操作的窗口
     * 2. ls:用于显示窗口左边界的字符
     * 3. rs:用于显示窗口右边界的字符
     * 4. ts:用于显示窗口上边界的字符
     * 5. bs:用于显示窗口下边界的字符
     * 6. tl:用于显示窗口左上角的字符
     * 7. tr:用于显示窗口右上角的字符
     * 8. bl:用于显示窗口左下角的字符
     * 9. br:用于显示窗口右下角的字符
     */
    wrefresh(local_win);
    delwin(local_win);
}
```

9.3 程序解析

别害怕，我知道这是一个很大的例子。但是我不得不在这里讲解一些很重要的东西。这个例子创建了一个窗口，并且可以使用键盘的方向键使窗口移动。它通过用户不断地按方向键，不断地删除旧窗口并且建立一个新的窗口。但不要超过窗口行列的限制，检查窗口的行列限制是一项很重要的工作。下面让我们逐行的分析这个例子。

creat_newwin()函数使用 newwin()函数建立一个窗口，并且使用 box()函数的窗口添加边框。**destory_win()**函数首先使用空白字符填充窗口，起到清除屏幕的作用。之后调用**delwin()**函数回收分配给窗口的内存。随着用户按下方向键**,startx**和**starty**的值就会不断改变并且建立一个新的窗口。

在**destory_win()**中，你可以看到我使用**wborder**替代**box**。这样做的原因我已经写到程序注释里面去了（我知道你刚才忽略了，现在赶紧去看看！）。**wborder()**函数可以用字符来绘制窗口的边框。这些边框是由四条线和四个角组成的。为了解释得更清楚一些，如果你已经按照如下的方向调用**wborader()**

```
wborder(win, '|', '|', '-', '-', '+', '+', '+', '+');
```

他所绘制的窗口会是以下这样子：

```
+----------------------+
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
+----------------------+
```

9.4 更多一点

你也可以在上面的的的例子中看到，我使用了 COLS 和 LINES 作为变量名。在 initscr()函数展初始化屏幕以后，这些变量中存储的数据就是屏幕初始化后的信息。这样做是为了方便标记窗口尺寸和计算出屏幕的中心位置坐标。getch()函数依然用来处理用户的键盘输入并且根据用户的输入作出程序中定义的处理。这种做法在交互式的图形界面应用程序中相当普遍。

9.5 其他的边框函数

上面这个例子所使用的方式：通过按下键盘上相应的按钮销毁一个窗口并建立一个新的窗口。这样的工作方式效率太低。下面让我们来写一些可以使窗口边框的使用更加有效率的程序。

下面这个程序使用 mvhline()和 mvvline()函数完成同样的效果。这两个函数非常简单，他们将在指定的位置绘制出指定大小的窗口。

例 8：绘制窗口边框的函数

```
#include <ncurses.h>

typedef struct _win_border_struct {
      chtype  ls, rs, ts, bs,
           tl, tr, bl, br;
}WIN_BORDER;
typedef struct _WIN_struct {

      int startx, starty;
      int height, width;
      WIN_BORDER border;
}WIN;

void init_win_params(WIN *p_win);
void print_win_params(WIN *p_win);
void create_box(WIN *win, int bool);

int main(int argc, char *argv[])
{     WIN win;
      int ch;

      initscr();              /*  初始化并进入 curses 模式          */
      start_color();          /*  开启彩色显示功能                  */
      cbreak();               /*  行缓冲禁止，传递所有控制信息  */
      keypad(stdscr, TRUE);       /*程序需要使用 F1 功能键  */
      noecho();
      init_pair(1, COLOR_CYAN, COLOR_BLACK);
```

```
    /* 以下是初始化窗口参数 */
    init_win_params(&win);
    print_win_params(&win);

    attron(COLOR_PAIR(1));
    printw("Press F1 to exit");
    refresh();
    attroff(COLOR_PAIR(1));

    create_box(&win, TRUE);
    while((ch = getch()) != KEY_F(1))
    {   switch(ch)
        {   case KEY_LEFT:
                create_box(&win, FALSE);
                --win.startx;
                create_box(&win, TRUE);
                break;
            case KEY_RIGHT:
                create_box(&win, FALSE);
                ++win.startx;
                create_box(&win, TRUE);
                break;
            case KEY_UP:
                create_box(&win, FALSE);
                --win.starty;
                create_box(&win, TRUE);
                break;
            case KEY_DOWN:
                create_box(&win, FALSE);
                ++win.starty;
                create_box(&win, TRUE);
                break;
        }
    }
    endwin();           /*结束 curses 模式          */
    return 0;
}
void init_win_params(WIN *p_win)
{
    p_win->height = 3;
    p_win->width = 10;
    p_win->starty = (LINES - p_win->height)/2;
    p_win->startx = (COLS - p_win->width)/2;
```

```
    p_win->border.ls = '|';
    p_win->border.rs = '|';
    p_win->border.ts = '-';
    p_win->border.bs = '-';
    p_win->border.tl = '+';
    p_win->border.tr = '+';
    p_win->border.bl = '+';
    p_win->border.br = '+';

}
void print_win_params(WIN *p_win)
{
#ifdef _DEBUG
    mvprintw(25, 0, "%d %d %d %d", p_win->startx, p_win->starty,
                p_win->width, p_win->height);
    refresh();
#endif
}
void create_box(WIN *p_win, int bool)
{   int i, j;
    int x, y, w, h;

    x = p_win->startx;
    y = p_win->starty;
    w = p_win->width;
    h = p_win->height;

    if(bool == TRUE)
    {   mvaddch(y, x, p_win->border.tl);
        mvaddch(y, x + w, p_win->border.tr);
        mvaddch(y + h, x, p_win->border.bl);
        mvaddch(y + h, x + w, p_win->border.br);
        mvhline(y, x + 1, p_win->border.ts, w - 1);
        mvhline(y + h, x + 1, p_win->border.bs, w - 1);
        mvvline(y + 1, x, p_win->border.ls, h - 1);
        mvvline(y + 1, x + w, p_win->border.rs, h - 1);
    }
    else
        for(j = y; j <= y + h; ++j)
            for(i = x; i <= x + w; ++i)
                mvaddch(j, i, ' ');

    refresh();
}
```

第十章：关于颜色系统

10.1 基础知识

生命中如果没有颜色的将是单调无趣的。CURSES 有一个非常不错的颜色处理机制。让我们通过以下这个小程序了解一下：

例 9：一个简单的颜色使用例子

```
#include <ncurses.h>

void print_in_middle(WINDOW *win, int starty, int startx, int width, char *string);
int main(int argc, char *argv[])
{   initscr();                  /*启动 curses 模式*/
    if(has_colors() == FALSE)
    {   endwin();
        printf("You terminal does not support color\n");
        exit(1);
    }
    start_color();              /*启动 color 机制 */
    init_pair(1, COLOR_RED, COLOR_BLACK);

    attron(COLOR_PAIR(1));
    print_in_middle(stdscr, LINES / 2, 0, 0, "Viola !!! In color ...");
    attroff(COLOR_PAIR(1));
        getch();
    endwin();
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char *string)
{   int length, x, y;
    float temp;
    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;
    length = strlen(string);
    temp = (width - length)/ 2;
```

```
    x = startx + (int)temp;
    mvwprintw(win, y, x, "%s", string);
    refresh();
}
```

通过这个例子你可以看到，如果要启动彩色机制，必须先调用 start_color()函数。这个函数被调用后，你就可以在你的终端屏幕上使用各种各样的颜色调整函数。如果您要检测你的屏幕的颜色显示能力，可以使用 has_colors()函数，如果你的终端屏幕不支持彩色显示，那么函数将返回 FLASE。

当 start_color()函数被调用后，CURSES 会初始化所有终端能够支持的颜色代码。这些颜色已经被定义成常量，可以通过访问这些常量（比如 COLOR_BLACK）来使用这些颜色。事实上在使用颜色时，你总是会成对的使用颜色常量设置颜色。这就意味着你必须给 init_pairs()传递一对参数用来定义屏幕的前景色和背景色。在这一对参数被应用后，这一对参数将被作为 COLOR_PAIR()函数的属性。这种方式也许会显得有点繁琐，但是这也是允许我们使管理屏幕颜色的一种更加简单的方法。你应该去看看“dailog”的源代码，顾名思义，“dailog”是一个用于在 shell 环境下显示对话框的程序。开发者已经定义了对话框的前景色和背景色。以便当我们需要在程序开始的时候初始化。通过访问一对变量来设置颜色属性，就像我们访问已定义的常量一样简单。
以下的这些颜色已经被预定义在 curses.h 里，你可以将它们当作颜色参数传递给相应的颜色函数。

COLOR_BLACK	**0**	**黑色**
COLOR_RED	**1**	**红色**
COLOR_GREEN	**2**	**绿色**
COLOR_YELLOW	**3**	**黄色**
COLOR_BLUE	**4**	**蓝色**
COLOR_MAGENTA	**5**	**洋红色**
COLOR_CYAN	**6**	**蓝绿色，青色**
COLOR_WHITE	**7**	**白色**

10.2 改变颜色定义

init_color()函数可以用来在初始化颜色的时候改变某个颜色的 RGB 值。比如你想减弱预定的红色设置。你可以这样调用 init_color()函数：

```
    init_color(COLOR_RED, 700, 0, 0);
    /* 参数 1      : 颜色名称
     * 参数 2, 3, 4 : 分别为 R(red),G(green),B(blue)的数值。最小值：0，最大值：
1000*/
```

如果你的显示终端无法改变颜色设置，函数将返回 ERR。can_change_color()函数可以用来监测你的终端是否可以支持这样的颜色改变。RGB 参数的值是 0 到 1000 的整数。默认的红色(RED)的定义是 R：1000，G：0，B：0 。

10.3 颜色定义内容（Color Content）

color_content()函数和 pair_content()函数可以用来查看 pair 变量中定义的内容。

第十一章：键盘管理

11.1 一些基础知识

一个图形用户界面（GUI）如果没有一个功能强大的用户对话接口的话，那它就是不完整的。一个 CURSES 程序应该对用户的键盘和鼠标操作很敏感。让我们先来了解一下你的键盘。

像你在以上看到的几乎所有的例子那样，得到用户的输入的键值非常简单。简单的取得用户按键信息的方法是使用 getch()函数。当你读取某些特殊的按键，例如换行键时，你应该激活 cbreak 模式。以便于读取功能键、方向键等等。

getch()返回一个整数对应键盘上相应的按键。如果你输入的是一个普通字符。这个整数就等价于这个字符。否则这个整数将和 curses.h 中定义的常量相匹配。例如如果用户按下了 F1，这个数字的返回值将是 265。这个值可以通过 curses.h 中内定义的宏 KEY_F()检测。这样可以更方便的管理键盘的输入。

例如，如果你这样调用 getch()函数：

```
int ch;
ch = getch();
```

getch()函数将等待用户输入。（除非你规定了延时时间）当你按下一个键，就会返回一个相应的整数。然后你可以检测这个值是否是你想要的。当然，这些相对应的值可以在 curses.h 中找到。

以下这段代码就可以用来监测键盘左方向键：

```
if(ch = = KEY_LEFT)
    printw("Left arrow is pressed\n");
```

让我们写一个可以通过上下键操纵的窗口菜单。

11.2.一个简单的使用键盘的例子

例 10： 一个读取键盘的例子

```
#include <stdio.h>
#include <ncurses.h>

#define WIDTH 30
#define HEIGHT 10
```

```
int startx = 0;
int starty = 0;
char *choices[] = {
                "Choice 1",
                "Choice 2",
                "Choice 3",
                "Choice 4",
                "Exit",
              };
int n_choices = sizeof(choices) / sizeof(char *);
void print_menu(WINDOW *menu_win, int highlight);

int main()
{   WINDOW *menu_win;
    int highlight = 1;
    int choice = 0;
    int c;

    initscr();
    clear();
    noecho();
    cbreak();   /*禁用行缓冲，直接传递所有输入 */
    startx = (80 - WIDTH) / 2;
    starty = (24 - HEIGHT) / 2;

    menu_win = newwin(HEIGHT, WIDTH, starty, startx);
    keypad(menu_win, TRUE);
    mvprintw(0, 0,"Use arrow keys to go up and down, Press enter to select a
choice");
    refresh();
    print_menu(menu_win, highlight);
    while(1)
    {   c = wgetch(menu_win);
        switch(c)
        {   case KEY_UP:
            if(highlight == 1)
                    highlight = n_choices;
                else
                    --highlight;
                break;
            case KEY_DOWN:
                if(highlight == n_choices)
                    highlight = 1;
                else
```

```
                    ++highlight;
                break;
            case 10:
                choice = highlight;
                break;
            default:
                mvprintw(24, 0, "Charcter pressed is = %3d Hopefully it can be printed
        as '%c'", c, c);
                refresh();
                break;
        }
        print_menu(menu_win, highlight);
        if(choice != 0)   /*用户必须要做一个选择*/
            break;
    }
    mvprintw(23, 0, "You chose choice %d with choice string %s\n", choice,
choices[choice - 1]);
    clrtoeol();
    refresh();
    endwin();
    return 0;
}
void print_menu(WINDOW *menu_win, int highlight)
{
    int x, y, i;
    x = 2;
    y = 2;
    box(menu_win, 0, 0);
    for(i = 0; i < n_choices; ++i)
    {   if(highlight == i + 1)      /*  使选择的字符串高亮显示  */
        {   wattron(menu_win, A_REVERSE);
            mvwprintw(menu_win, y, x, "%s", choices[i]);
            wattroff(menu_win, A_REVERSE);
        }
        else
            mvwprintw(menu_win, y, x, "%s", choices[i]);
        ++y;
    }
    wrefresh(menu_win);
}
```

第十二章：使用鼠标

你现在应该知道怎样取得键盘值了，让我们也来学习一下如何取得鼠标信号。很多的用户界面程序都被设计成允许使用键盘和鼠标共同操作。

12．1 基础知识（The Basics）

在使用鼠标之前,你要让鼠标接收的事件先要用 **mousemask( )**这个函数来激活.

```
mousemask(  mmask_t newmask,  /* 你想要监听的鼠标事件掩码 */
            mmask_t *oldmask ) /* 先前的鼠标事件掩码     */
```

上述函数中的第一个参数,就是你所要监听的事件的一个位掩码，默认情况下，在使用该函数之前。所有的鼠标事件接收状态都是未激活的。位掩码 ALL_MOUSE_EVENTS 可以让鼠标接收所有的事件。

这些是 NCURSES 定义的位掩码清单：（注意：不同的鼠标按键号码设置不同，使用前需要测试。一般情况下左键为 1 号，右键为二号）

掩码	对应事件
BUTTON1_PRESSED	鼠标 1 号键按下
BUTTON1_RELEASED	鼠标 1 号键释放
BUTTON1_CLICKED	鼠标 1 号键单击
BUTTON1_DOUBLE_CLICKED	鼠标 1 号键双击
BUTTON1_TRIPLE_CLICKED	鼠标 1 号键三击
BUTTON2_PRESSED	鼠标 2 号键按下
BUTTON2_RELEASED	鼠标 2 号键释放
BUTTON2_CLICKED	鼠标 2 号键单击
BUTTON2_DOUBLE_CLICKED	鼠标 2 号键双击
BUTTON2_TRIPLE_CLICKED	鼠标 2 号键三击
BUTTON3_PRESSED	鼠标 3 号键按下
BUTTON3_RELEASED	鼠标 3 号键释放
BUTTON3_CLICKED	鼠标 3 号键单击
BUTTON3_DOUBLE_CLICKED	鼠标 3 号键双击
BUTTON3_TRIPLE_CLICKED	鼠标 3 号键三击
BUTTON4_PRESSED	鼠标 4 号键按下
BUTTON4_RELEASED	鼠标 4 号键释放
BUTTON4_CLICKED	鼠标 4 号键单击
BUTTON4_DOUBLE_CLICKED	鼠标 4 号键双击
BUTTON4_TRIPLE_CLICKED	鼠标 4 号键三击
BUTTON_SHIFT	在鼠标事件发生时，伴随 Shift 键按下
BUTTON_CTRL	在鼠标事件发生时，伴随 Ctrl 键按下
BUTTON_ALT	在鼠标事件发生时，伴随 Alt 键按下
ALL_MOUSE_EVENTS	报告所有的鼠标事件
REPORT_MOUSE_POSITION	报告鼠标移动位置

12．2 取得鼠标事件（Getting the events）

当所有的鼠标监听事件被激活后。**getch()**一类的函数可以返回每次接收到的鼠标事件KEY_MOUSE。然后通过 **getmouse()**函数可以取得这些事件。

代码大概看起来是这样：

```
MEVENT event;

ch = getch();
if(ch == KEY_MOUSE)
    if(getmouse(&event) == OK)
    {
        .    /* 处理这个事件的代码 */
    }
。。。。。。
```

getmouse()函数将这个事件返回一个相应的指针。这个指针结构是这样的：

```
typedef struct
{
    short id;           /* ID 用来辨别不同的设备 */
    int x, y, z;        /* 事件发生的坐标 */
    mmask_t bstate;     /* 鼠标按键状态 */
}
```

Bstate 是我们关注的最主要变量，它返回了当前鼠标按键的状态。下面的这段代码可以让我们看看按下鼠标左键会出现什么：

```
if(event.bstate & BUTTON1_PRESSED)
    printw("Left Button Pressed");
```

12．3 把它们放在一起

让我们做一个有鼠标交互操作的菜单程序。为了让例子看起来更简单，去掉了键盘操作。

例 11：用鼠标访问菜单

```
#include <ncurses.h>

#define WIDTH 30
#define HEIGHT 10

int startx = 0;
int starty = 0;

char *choices[] = {
            "Choice 1",
```

```
                "Choice 2",
                "Choice 3",
                "Choice 4",
                "Exit",
              };

int n_choices = sizeof(choices) / sizeof(char *);

void print_menu(WINDOW *menu_win, int highlight);
void report_choice(int mouse_x, int mouse_y, int *p_choice);

int main()
{    int c, choice = 0;
     WINDOW *menu_win;
     MEVENT event;

     /* 初始化 curses */
     initscr();
     clear();
     noecho();
     cbreak();    /* 禁用行缓冲，直接传递所有的信号 */

     /* 将窗口放在屏幕中央 */
     startx = (80 - WIDTH) / 2;
     starty = (24 - HEIGHT) / 2;

     attron(A_REVERSE);
     mvprintw(23, 1, "Click on Exit to quit (Works best in a virtual console)");
     refresh();
     attroff(A_REVERSE);

     /* 首先显示菜单 */
     menu_win = newwin(HEIGHT, WIDTH, starty, startx);
     print_menu(menu_win, 1);
     /* 监听所有的鼠标事件 */
     mousemask(ALL_MOUSE_EVENTS, NULL);

     while(1)
     {    c = wgetch(menu_win);
          switch(c)
          {    case KEY_MOUSE:
               if(getmouse(&event) == OK)
               {    /* 用户按下鼠标左键 */
                    if(event.bstate & BUTTON1_PRESSED)
```

```
                {   report_choice(event.x + 1, event.y + 1, &choice);
                    if(choice == -1) /* 退出选项 */
                        goto end;
                    mvprintw(22, 1, "Choice made is : %d String Chosen is \"%10s\"", choice, choices[choice - 1]);
                    refresh();
                }
            }
            print_menu(menu_win, choice);
            break;
        }
    }
end:
    endwin();
    return 0;
}


void print_menu(WINDOW *menu_win, int highlight)
{
    int x, y, i;

    x = 2;
    y = 2;
    box(menu_win, 0, 0);
    for(i = 0; i < n_choices; ++i)
    {   if(highlight == i + 1)
        {   wattron(menu_win, A_REVERSE);
            mvwprintw(menu_win, y, x, "%s", choices[i]);
            wattroff(menu_win, A_REVERSE);
        }
        else
            mvwprintw(menu_win, y, x, "%s", choices[i]);
        ++y;
    }
    wrefresh(menu_win);
}

/* 报告鼠标所在位置的菜单选项 */
void report_choice(int mouse_x, int mouse_y, int *p_choice)
{   int i,j, choice;

    i = startx + 2;
    j = starty + 3;
```

```
    for(choice = 0; choice < n_choices; ++choice)
    if(mouse_y == j + choice && mouse_x >= i && mouse_x <= i +
strlen(choices[choice]))
        {   if(choice == n_choices - 1)
                *p_choice = -1;
            else
                *p_choice = choice + 1;
            break;
        }
}
```

12．4 一些辅助函数说明

mouse_trafo() 函数和 **wmouse_trafo()**函数用来转换鼠标坐标到相对应的屏幕坐标。想得到详细资料可以阅读 curs_mouse(3X)的 man 文档页。

为了识别鼠标的单击，**mouseinterval()**函数可以设置鼠标按键按下和释放的间隔时间（按千分之一秒计）。这个函数返回当前设置的间隔时间，默认间隔时间是五分之一秒。

第十三章：屏幕操作

在这一章里， 我们将看到一些控制屏幕的高级函数。顺便写一些非常有意思的程序。这些程序经常被用于编写基于控制台的游戏。

13．1 getyx()函数集

getyx() 函数可以用来找到当前光标的位置。它还可以根据你设置的屏幕的横、纵坐标移动你的当前光标。getyx()是一个定义的宏，所以你不能向它传送变量的地址。你只能这样调用它：

```
getyx(win, y, x);
 /*     win:  窗口的指针
  *     y, x:  光标坐标的 y, x 值将被赋到这两个变量
  */
```

getparyx()用于取得子窗口相对于主窗口的起始坐标。这个函数对于更新子窗口的信息非常有用。当设计类例如多级菜单这样的特殊组件时，存储菜单坐标就会显得十分困难。所以他使用坐标的方式会轻而易举的解决这个问题：在子窗口中建立这个菜单，然后通过**getparyx()**函数设置这个子窗口对应的起始坐标。

getbegyx()函数和 **getmaxyx()**函数用于存储当前窗口的起始坐标和结束坐标。通过对这两个函数可以高效的管理主窗口和子窗口的大小和位置。

13．2 屏幕转储

对于游戏开发而言，存储和恢复屏幕的某个区域是十分重要的。**scr_dump()**函数可以当前的屏幕存入一个指定的文件中（函数原型：**scr_dump(const char *file)** ——译者注）。之后可以通过 **scr_restore()**调用转储文件来函数恢复屏幕（函数原型：**scr_restore(const char *file)** ——译者注）。在游戏设计中这两个函数可以用来使游戏在场景切换时快速移动。

13．3 窗口转储

窗口转储同屏幕转储的原理一样。**getwin()**函数（函数原型：**getwin(FILE * filep)** ——译者注）用来将窗口内容存储到一个指定的文件中。**putwin()**函数（函数原型：**putwin(WINDOW *win, FILE * filep)** ——译者注）则调用相应文件恢复窗口。

copywin()可以将一个窗口的内容拷贝到另外一个窗口。它将源窗口指定矩形区域的所有内容复制到目的窗口指定的矩形区域。最后的参数用来选择要重写整个目的窗口，还是覆盖所选择的目的窗口的矩形区域。如果参数为真，那么就是非破坏性的。（以下为函数的原型——译者注）

```
int  copywin(
        const WINDOW *src,      /*源窗口指针*/
        WINDOW   *dst,          /*目的窗口指针*/
        int  sminrow,           /*源窗口所选矩形区域的最小行数*/
        int  smincol,           /*源窗口所选矩形区域的最小列数*/
```

```
int  dminrow,          /*目的窗口所选矩形区域的最小行数*/
int  dmincol,          /*目的窗口所选矩形区域的最小列数*/
int  dmaxrow,          /*目的窗口所选矩形区域的最大行数*/
int  dmaxcol,          /*目的窗口所选矩形区域的最达列数*/
int  over)             /*是否覆盖目的窗口*/
```

第十四章：其他特色

现在你了解的函数足够可以写一个非常不错的 CURSES 程序了，而且你还可以加很多的花样。这里还有一些在很多情况下都很有用的函数，我们来看看其中一些函数吧。

14．1 curs_set()函数（curs_set()）

这个函数用来设制光标是否可见。它的参数可以是：**0**（不可见），**1**（可见），**2**（完全可见）

14．2 临时离开 Curses 模式（Temporarily Leaving Curses mode）

有时候你也许会想暂时离开 CURSES 模式，回到行缓冲模式下做些其他的事。在这种情况下，你首先要调用 **def_prog_mode()**函数存储 tty 模式下的信息，然后使用 **end_win()**函数退出 CURSES 模式。就可以回到最初的 tty 模式。如果想要恢复，就可以调用**reset_prog_mode()**函数，这个函数会通过被**def_prog_mode()**函数存储的信息重新读入虚拟的屏幕上，这时别忘了使用 **refresh()**函数刷新，你才看到你真正回到了 CURSES 模式。让我们通过一个小例子了解一下这些函数的用法：

例 12：临时离开模式

```
#include <ncurses.h>

int main()
{
    initscr();                  /* 启动 CURSES 模式 */
    printw("Hello World !!!\n"); /* 打印 Hello World!!! */
    refresh();                  /* 让虚拟显示器的内容显示到屏幕上 */
    def_prog_mode();            /* 存储当前 tty 模式 */
    endwin();                   /* 临时退出 CURSES 模式 */
    system("/bin/sh");          /* 返回普通的行缓冲模式 */
    reset_prog_mode();          /* 返回到 def_prog_mode()存储的 tty 模式 */
    refresh();                  /* 使用 refresh() 函数恢复屏幕的内容 */
    printw("Another String\n"); /* 完全返回 CURSES 模式    */
    refresh();                  /* 别忘了刷新屏幕 */
    endwin();                   /* 退出 CURSES 模式 */
    return 0;
}
```

14．3 ACS_变量（ACS_ variables）

如果你曾经在 DOS 下编写过程序，你应该知道扩展字符集。但是它们只能在某些终端上才是可以显示出来的。例如 NCURSES 的 **box()**函数（译者注：这个函数用来绘制一个矩形框）就使用了这些字符。所有这些字符都是 ACS_变量，所谓 ACS，就是 **A**lternative **C**haracter **S**et（可选字符集）。你可以注意到在以前的程序中多多少少都用到这些有意思的字符。这有一个程序分别介绍这些字符：

例 13：ACS 变量介绍例子

```
#include <ncurses.h>

int main()
{
initscr();
printw("Upper left corner          ");              addch(ACS_ULCORNER);
printw("\n");
printw("Lower left corner          "); addch(ACS_LLCORNER); printw("\n");
printw("Lower right corner         ");              addch(ACS_LRCORNER);
printw("\n");
printw("Tee pointing right         "); addch(ACS_LTEE); printw("\n");
printw("Tee pointing left          "); addch(ACS_RTEE); printw("\n");
printw("Tee pointing up            "); addch(ACS_BTEE); printw("\n");
printw("Tee pointing down          "); addch(ACS_TTEE); printw("\n");
printw("Horizontal line            "); addch(ACS_HLINE); printw("\n");
printw("Vertical line              "); addch(ACS_VLINE); printw("\n");
printw("Large Plus or cross over   "); addch(ACS_PLUS); printw("\n");
printw("Scan Line 1                "); addch(ACS_S1); printw("\n");
printw("Scan Line 3                "); addch(ACS_S3); printw("\n");
printw("Scan Line 7                "); addch(ACS_S7); printw("\n");
printw("Scan Line 9                "); addch(ACS_S9); printw("\n");
printw("Diamond                    "); addch(ACS_DIAMOND); printw("\n");
printw("Checker board (stipple)    "); addch(ACS_CKBOARD); printw("\n");
printw("Degree Symbol              "); addch(ACS_DEGREE); printw("\n");
printw("Plus/Minus Symbol          "); addch(ACS_PLMINUS); printw("\n");
printw("Bullet                     "); addch(ACS_BULLET); printw("\n");
printw("Arrow Pointing Left        "); addch(ACS_LARROW); printw("\n");
printw("Arrow Pointing Right       "); addch(ACS_RARROW); printw("\n");
printw("Arrow Pointing Down        "); addch(ACS_DARROW); printw("\n");
printw("Arrow Pointing Up          "); addch(ACS_UARROW); printw("\n");
printw("Board of squares           "); addch(ACS_BOARD); printw("\n");
printw("Lantern Symbol             "); addch(ACS_LANTERN); printw("\n");
printw("Solid Square Block         "); addch(ACS_BLOCK); printw("\n");
printw("Less/Equal sign            "); addch(ACS_LEQUAL); printw("\n");
printw("Greater/Equal sign         "); addch(ACS_GEQUAL); printw("\n");
printw("Pi                         "); addch(ACS_PI); printw("\n");
printw("Not equal                  "); addch(ACS_NEQUAL); printw("\n");
printw("UK pound sign              "); addch(ACS_STERLING); printw("\n");
refresh();
getch();
endwin();
return 0;
}
```

第十五章：扩展库

除了 curses 主函数库以外，以下还有一些扩展的库，这些库可以增强 CURSE 的性能并添加了很多新特性。以下的章节就要分别介绍这三个独立发布的扩展库，它们分别是 panel（面板扩展库），menu（菜单扩展库），form（表单扩展库）。

第十六章：面板库

现在你应该已经很精通 curses 库了，你可能会想试着做一些更大的项目。为了让界面看起来更专业，于是你就创建了许多重叠的窗口。但很不幸的是，你很快会发现它们变得难以管理，多次的更新窗口使开发变成了一场恶梦。如果你没有按照适当的顺序刷新那些窗口的话，它们就会给你带来很多麻烦。

不过别失望，面板库（Panel Library）提供了一个很好的解决方案。用 ncureses 的开发者的话来说就是：

*如果你的界面设计需要使窗口在运行的时候置底或者置顶，你就会发现不光显示正确的结果很困难，而且为此付出得代价很大。这时候 **Panels** 库就可以派上用场了。*

如果你要处理很多重叠的窗口，选择 panels 库绝对没错。通过一系列的 wnoutrefresh()函数和 doupdate()函数调用可以大大减少工作量。面板库实时控制着窗口的顺序信息，并且可以处理好重叠顺序，而且可以适时的更新屏幕。读到这里，你是不是又开始激动了呢？那还等什么呢，让我们继续！

16.1 基础知识

面板对象实际上也是一个窗口，它被认为是一块包含所有面板对象的一个平台。这个平台可以看作是一个栈，栈顶的面板是完全可见的，其它的面板就被“压”在下面。所有面板根据所处栈的位置不同从而决定它是否可见。基本思想就是：创建一个栈来保存那些重叠的面板，然后使用面板库来正确的显示它们。可以调用一个类似于 refresh（）的函数来以正确的显示顺序显示这些面板。 面板库提供了一些可以隐藏、显示、移动和改变其大小等操作的函数。这样，我们就可以通过调用面板库提供的函数来解决重叠的窗口的问题了。

一般的，一个面板程序的设计流程如下：

1、 创建要添加到面板里面去的窗口（使用 **newwin（）**）。
2、 创建一个有合适可见顺序的面板，根据需要的可见顺序把它们压进栈。函数 **new_panel()** 可以用来创建面板。
3、 调用 **update_panels()** 把面板按正确的顺序写到虚拟屏幕上，调用 **doupdate()**让它在现实器上显示。
4、 调用 **show_panel(), hide_panel(), move_panel()**等函数来对面板进行操作。可以使用一些例如 **panel_hidden()** 和 **panel_window()**之类的辅助函数。你也可以使用用户指针来存储为一个定制面板的数据。而且可以使用函数 **set_panel_userptr()** 和 **panel_userptr()** 来设置和取得一个面板的用户指针。
5、 所有的工作都做完后用 **del_panel()** 删除这个面板。

接下来我们通过一些程序来加深一下对这些概念的理解。下面的将要看到的程序创建了 3 个重叠的面板并把它们按次序显示在屏幕上。

16.2 编译包含面板库的程序

要使用面板库里的函数，你首先要把 panel.h 这个头文件包含到你的代码中。如果要把编

译并连接与面板库相关的程序的话就要同时添加-lpanel 和 –lncurses 两个参数。

```
#include <panel.h>
```

.
.
.

编译和连接: gcc <program file> -lpanel –lncurses

例 14. 一个有关面板库的基础例子

```
#include <panel.h>

int main()
{
        WINDOW *my_wins[3];
        PANEL  *my_panels[3];
        int lines = 10, cols = 40, y = 2, x = 4, i;

        initscr();
        cbreak();
        noecho();
        /* 为每个面板创建窗口 */
        my_wins[0] = newwin(lines, cols, y, x);
        my_wins[1] = newwin(lines, cols, y + 1, x + 5);
        my_wins[2] = newwin(lines, cols, y + 2, x + 10);

        /* 为窗口添加创建边框以便你能看到面板的效果 */
        for(i = 0; i < 3; ++i)
                box(my_wins[i], 0, 0);

        /* 按自底向上的顺序，把给每个窗口添加进一个面板 */
        my_panels[0] = new_panel(my_wins[0]); /* 把面板 0 压进栈，叠放顺序:
stdscr-0 */
        my_panels[1] = new_panel(my_wins[1]); /* 把面板 1 压进栈，叠放顺序:
stdscr-0-1 */
        my_panels[2] = new_panel(my_wins[2]); /* 把面板 2 压进栈，叠放顺序:
stdscr-0-1-2 */

        /* 更新栈的顺序。把面板 2 置于栈顶  */
        update_panels();

        /* 在屏幕上显示 */
        doupdate();

        getch();
        endwin();
```

```
}
```

如你所见，上面的这个程序就是按照前面所讲的那个流程进行的。用 **newwin()** 创建窗口，然后用 **new_panel()**把它们添加到面板库里面去，当我们把那些面板一个一个压进栈的时候，panels 栈也就随之更新了。调用 **update_panels()** 和 **doupdate()** 就可以让它们在屏幕上显示出来。

16.3 面板窗口浏览

下面给出一个稍微复杂点的例子。这个程序创建了 3 个窗口，通过使用<TAB>键可以使它们循环置顶显示。让我们来看一下代码：

例 15 一个面板窗口浏览的例子

```
#include <panel.h>
#define NLINES 10
#define NCOLS 40

void init_wins(WINDOW **wins, int n);
void win_show(WINDOW *win, char *label, int label_color);
void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);

int main()
{   WINDOW *my_wins[3];
    PANEL  *my_panels[3];
    PANEL  *top;
    int ch;

    /*初始化 curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* 初始化所有的颜色 */
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_GREEN, COLOR_BLACK);
    init_pair(3, COLOR_BLUE, COLOR_BLACK);
    init_pair(4, COLOR_CYAN, COLOR_BLACK);

    init_wins(my_wins, 3);

    /* 按自底向上的顺序，把每个窗口添加进一个面板 */
    my_panels[0] = new_panel(my_wins[0]);/* 把面板 0 压入栈， 顺序: stdscr-0
```

```
*/
    my_panels[1] = new_panel(my_wins[1]); /* 把面板 1 压入栈，顺序:
stdscr-0-1 */
    my_panels[2] = new_panel(my_wins[2]); /* 把面板 2 压入栈，顺序:
stdscr-0-1-2 */

    /* 为下一个面板建立用户指针 */
    set_panel_userptr(my_panels[0], my_panels[1]);
    set_panel_userptr(my_panels[1], my_panels[2]);
    set_panel_userptr(my_panels[2], my_panels[0]);

    /* 更新面板栈的顺序。把面板 2 置于栈顶*/
    update_panels();

    /* 在屏幕上显示*/
    attron(COLOR_PAIR(4));
    mvprintw(LINES - 2, 0, "Use tab to browse through the windows (F1 to Exit)");
    attroff(COLOR_PAIR(4));
    doupdate();

    top = my_panels[2];
    while((ch = getch()) != KEY_F(1))
    {   switch(ch)
        {   case 9:
                top = (PANEL *)panel_userptr(top);
                top_panel(top);
                break;
        }
        update_panels();
        doupdate();
    }
    endwin();
    return 0;
}

/* 显示所有的窗口 */
void init_wins(WINDOW **wins, int n)
{   int x, y, i;
    char label[80];

    y = 2;
    x = 10;
    for(i = 0; i < n; ++i)
    {   wins[i] = newwin(NLINES, NCOLS, y, x);
```

```
            sprintf(label, "Window Number %d", i + 1);
            win_show(wins[i], label, i + 1);
            y += 3;
            x += 7;
        }
    }

    /* 用一个边框和控件显示所有的窗口 */
    void win_show(WINDOW *win, char *label, int label_color)
    {   int startx, starty, height, width;

        getbegyx(win, starty, startx);
        getmaxyx(win, height, width);

        box(win, 0, 0);
        mvwaddch(win, 2, 0, ACS_LTEE);
        mvwhline(win, 2, 1, ACS_HLINE, width - 2);
        mvwaddch(win, 2, width - 1, ACS_RTEE);

        print_in_middle(win, 1, 0, width, label, COLOR_PAIR(label_color));
    }

    void print_in_middle(WINDOW *win, int starty, int startx, int width, char
    *string, chtype color)
    {   int length, x, y;
        float temp;

        if(win == NULL)
            win = stdscr;
        getyx(win, y, x);
        if(startx != 0)
            x = startx;
        if(starty != 0)
            y = starty;
        if(width == 0)
            width = 80;

        length = strlen(string);
        temp = (width - length)/ 2;
        x = startx + (int)temp;
        wattron(win, color);
        mvwprintw(win, y, x, "%s", string);
        wattroff(win, color);
        refresh();
```

```
}
```

16.4 使用用户指针

在上面的例子中，为了在这个循环例查找下一个窗口我使用了用户指针。我们可以通过指定一个用户指针给面板增加自定义信息，这个指针可以指向你想要存储的任何信息。在这个例子中，我往指针里面存储了循环中的下一个面板。一个面板的用户指针可以用函数 **set_panel_userptr()** 来设定。用面板作为函数 **panel_userptr()** 参变量可以访问它，函数将返回这个面板的用户指针。在循环中查找下一个面板结束后，通过使用函数 **top_panel()** 可以将其置于顶层。用给定的面板作为参变量，这个函数将会把这个面板置于面板栈的顶层。

16.5 移动面板和改变面板的大小

函数 **move_panel()**可以将面板移动到你想移动的地方。它不改变面板在栈里的位置。务必使用 **move_panel()**而不是用来移动窗口的函数 **mvwin()** 来移动面板。

改变面板的大小有一点点复杂，没有哪个函数可以用来直接改变和面板关联的窗口的大小。要改变面板的大小，只有创建一个大小为你所需的新窗口。用函数 **replace_panel()**可以替换和相应面板关联的窗口。但是别忘了要删除旧的窗口。和面板关联的窗口可以使用函数 **panel_window()**来创建。

下面这个程序就用简单的方式体现了这个概念。同样，你可以用<TAB>键来使它们循环。要改变大小或移动当前的面板你可以按下‘ r’ 或‘ m ’键。然后使用方向键来改变大小和移动。在具体操作时，为获得需要的结果这个例子利用了用户已有的数据。

例 16、 一个移动和改变面板大小的例子

```
#include <panel.h>

typedef struct _PANEL_DATA {
    int x, y, w, h;
    char label[80];
    int label_color;
    PANEL *next;
}PANEL_DATA;

#define NLINES 10
#define NCOLS 40

void init_wins(WINDOW **wins, int n);
void win_show(WINDOW *win, char *label, int label_color);
void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);
void set_user_ptrs(PANEL **panels, int n);

int main()
{   WINDOW *my_wins[3];
```

```
    PANEL  *my_panels[3];
    PANEL_DATA  *top;
    PANEL *stack_top;
    WINDOW *temp_win, *old_win;
    int ch;
    int newx, newy, neww, newh;
    int size = FALSE, move = FALSE;

    /* 初始化 curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* 初始化所有的颜色 */
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_GREEN, COLOR_BLACK);
    init_pair(3, COLOR_BLUE, COLOR_BLACK);
    init_pair(4, COLOR_CYAN, COLOR_BLACK);

    init_wins(my_wins, 3);

    /* 更新面板栈的顺序。把面板 2 置于栈顶*/
    my_panels[0] = new_panel(my_wins[0]);  /* 把面板 0 压入栈，顺序:
stdscr-0 */
    my_panels[1] = new_panel(my_wins[1]);  /* 把面板 1 压入栈。顺序:
stdscr-0-1 */
    my_panels[2] = new_panel(my_wins[2]);  /* 把面板 2 压入栈,顺序:
stdscr-0-1-2 */

    set_user_ptrs(my_panels, 3);
    /* 更新面板栈的顺序。把面板 2 置于栈顶 */
    update_panels();

    /* 在屏幕上显示出来 */
    attron(COLOR_PAIR(4));
    mvprintw(LINES - 3, 0, "Use 'm' for moving, 'r' for resizing");
    mvprintw(LINES - 2, 0, "Use tab to browse through the windows (F1 to Exit)");
    attroff(COLOR_PAIR(4));
    doupdate();

    stack_top = my_panels[2];
    top = (PANEL_DATA *)panel_userptr(stack_top);
```

```
    newx = top->x;
    newy = top->y;
    neww = top->w;
    newh = top->h;
    while((ch = getch()) != KEY_F(1))
    {   switch(ch)
        {   case 9:      /* Tab 对应编号 */
                top = (PANEL_DATA *)panel_userptr(stack_top);
                top_panel(top->next);
                stack_top = top->next;
                top = (PANEL_DATA *)panel_userptr(stack_top);
                newx = top->x;
                newy = top->y;
                neww = top->w;
                newh = top->h;
                break;
            case 'r': /* 改变大小*/
                size = TRUE;
                attron(COLOR_PAIR(4));
                mvprintw(LINES - 4, 0, "Entered Resizing :Use Arrow Keys to
resize and press <ENTER> to end resizing");
                refresh();
                attroff(COLOR_PAIR(4));
                break;
            case 'm':    /* 移动 */
                attron(COLOR_PAIR(4));
                mvprintw(LINES - 4, 0, "Entered Moving: Use Arrow Keys to
Move and press <ENTER> to end moving");
                refresh();
                attroff(COLOR_PAIR(4));
                move = TRUE;
                break;
            case KEY_LEFT:
                if(size == TRUE)
                {   --newx;
                    ++neww;
                }
                if(move == TRUE)
                    --newx;
                break;
            case KEY_RIGHT:
                if(size == TRUE)
                {   ++newx;
                    --neww;
```

```
                }
                if(move == TRUE)
                    ++newx;
                break;
            case KEY_UP:
                if(size == TRUE)
                {   --newy;
                    ++newh;
                }
                if(move == TRUE)
                    --newy;
                break;
            case KEY_DOWN:
                if(size == TRUE)
                {   ++newy;
                    --newh;
                }
                if(move == TRUE)
                    ++newy;
                break;
            case 10:    /* Enter 对应编号 */
                move(LINES - 4, 0);
                clrtoeol();
                refresh();
                if(size == TRUE)
                {   old_win = panel_window(stack_top);
                    temp_win = newwin(newh, neww, newy, newx);
                    replace_panel(stack_top, temp_win);
                    win_show(temp_win, top->label, top->label_color);
                    delwin(old_win);
                    size = FALSE;
                }
                if(move == TRUE)
                {   move_panel(stack_top, newy, newx);
                    move = FALSE;
                }
                break;

        }
        attron(COLOR_PAIR(4));
        mvprintw(LINES - 3, 0, "Use 'm' for moving, 'r' for resizing");
        mvprintw(LINES - 2, 0, "Use tab to browse through the windows (F1 to
Exit)");
            attroff(COLOR_PAIR(4));
```

```
            refresh();
        update_panels();
        doupdate();
    }
    endwin();
    return 0;
}

/* 显示所有的窗口 */
void init_wins(WINDOW **wins, int n)
{   int x, y, i;
    char label[80];

    y = 2;
    x = 10;
    for(i = 0; i < n; ++i)
    {   wins[i] = newwin(NLINES, NCOLS, y, x);
        sprintf(label, "Window Number %d", i + 1);
        win_show(wins[i], label, i + 1);
        y += 3;
        x += 7;
    }
}

/* 把每个面板设置为 PANEL_DATA 结构 */
void set_user_ptrs(PANEL **panels, int n)
{   PANEL_DATA *ptrs;
    WINDOW *win;
    int x, y, w, h, i;
    char temp[80];

    ptrs = (PANEL_DATA *)calloc(n, sizeof(PANEL_DATA));

    for(i = 0;i < n; ++i)
    {   win = panel_window(panels[i]);
        getbegyx(win, y, x);
        getmaxyx(win, h, w);
        ptrs[i].x = x;
        ptrs[i].y = y;
        ptrs[i].w = w;
        ptrs[i].h = h;
        sprintf(temp, "Window Number %d", i + 1);
        strcpy(ptrs[i].label, temp);
        ptrs[i].label_color = i + 1;
```

```
        if(i + 1 == n)
            ptrs[i].next = panels[0];
        else
            ptrs[i].next = panels[i + 1];
        set_panel_userptr(panels[i], &ptrs[i]);
    }
}
/* 用一个边框和标题栏来显示窗口 */
void win_show(WINDOW *win, char *label, int label_color)
{   int startx, starty, height, width;

    getbegyx(win, starty, startx);
    getmaxyx(win, height, width);

    box(win, 0, 0);
    mvwaddch(win, 2, 0, ACS_LTEE);
    mvwhline(win, 2, 1, ACS_HLINE, width - 2);
    mvwaddch(win, 2, width - 1, ACS_RTEE);

    print_in_middle(win, 1, 0, width, label, COLOR_PAIR(label_color));
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{   int length, x, y;
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;

    length = strlen(string);
    temp = (width - length)/ 2;
    x = startx + (int)temp;
    wattron(win, color);
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
```

```
}
```

我们把注意力集中在主循环上。一旦程序捕获某个键被按下，它就执行相应的动作。如果按下了‘r’键，“更改大小”模式就启动了。用户可以通过按方向键来更改面板的大小。当用户按<ENTER>键时，就确定了面板的新尺寸，程序就依照上面介绍的方法更改大小。不过在“更改大小”模式中，程序并没有真正显示出面板更改大小后的边框。这就留给读者一个作业：用点来显示新尺寸面板的边框。
当用户按‘m’键时“移动面板”模式就启动了。这个操作会比“更改大小”简单一点。随着方向键的按下，新面板的位置也随之移动，当<ENTER>键按下时，程序就调用 panel()函数把面板移动到光标的当前位置
在这个示范程序中，用户数据就代表 PANEL_DATA，在查找面板的相关信息时扮演重要角色。就象在说明中所写，PANEL_DATA 保存了面板的尺寸，标题，标题颜色和循环中指向下一个面板的指针。

16.6 隐藏和显示面板

隐藏一个面板可以使用函数 hide_panel()。这个函数仅仅是把它从面板栈中移走，因此，要在屏幕上隐藏的话只要调用函数 update_panels() 和 doupdate()就可以了。它不会破坏面板结构和这个隐藏的面板。函数 show_panel()可以让它重新显示。

下面的程序显示了如何隐藏面板。按键 ’a’ 或 ’b’ 或 ‘c’来分别实现显示或隐藏第一，二，三个窗口，其它的依此类推。它使用了一个用户给定的一个带有隐藏的变量的数据，用来跟踪窗口是否是隐藏的。因为某些原因，按理用来告诉用户一个面板是否隐藏的函数 panel_hidden 没有起作用。Michael Andres 在这里提供了一个错误报告。

例 17、 一个隐藏和显示面板的例子

```
#include <panel.h>

typedef struct _PANEL_DATA {
    int hide;     /* 如果面板是隐藏的时候为真   */
}PANEL_DATA;

#define NLINES 10
#define NCOLS 40

void init_wins(WINDOW **wins, int n);
void win_show(WINDOW *win, char *label, int label_color);
void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);

int main()
{   WINDOW *my_wins[3];
    PANEL  *my_panels[3];
    PANEL_DATA panel_datas[3];
    PANEL_DATA *temp;
```

```
    int ch;

    /* 初始化 curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* 初始化所有的颜色 */
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_GREEN, COLOR_BLACK);
    init_pair(3, COLOR_BLUE, COLOR_BLACK);
    init_pair(4, COLOR_CYAN, COLOR_BLACK);

    init_wins(my_wins, 3);

    /* 更新面板栈的顺序。把面板 2 置于栈顶 */
    my_panels[0] = new_panel(my_wins[0]);  /* Push 0, order: stdscr-0 */
    my_panels[1] = new_panel(my_wins[1]);  /* Push 1, order: stdscr-0-1
*/
    my_panels[2] = new_panel(my_wins[2]);  /* Push 2, order: stdscr-0-1-2
*/

    /* 初始化所有的面板并都设为非隐藏的 */
    panel_datas[0].hide = FALSE;
    panel_datas[1].hide = FALSE;
    panel_datas[2].hide = FALSE;

    set_panel_userptr(my_panels[0], &panel_datas[0]);
    set_panel_userptr(my_panels[1], &panel_datas[1]);
    set_panel_userptr(my_panels[2], &panel_datas[2]);

    /* 更新面板栈的顺序，第二个面板将置于栈顶 */
    update_panels();

    /* 在屏幕上显示 */
    attron(COLOR_PAIR(4));
    mvprintw(LINES - 3, 0, "Show or Hide a window with 'a'(first window)
'b'(Second Window)  'c'(Third Window)");
    mvprintw(LINES - 2, 0, "F1 to Exit");

    attroff(COLOR_PAIR(4));
    doupdate();
```

```
    while((ch = getch()) != KEY_F(1))
    {   switch(ch)
        {   case 'a':
                temp = (PANEL_DATA *)panel_userptr(my_panels[0]);
                if(temp->hide == FALSE)
                {   hide_panel(my_panels[0]);
                    temp->hide = TRUE;
                }
                else
                {   show_panel(my_panels[0]);
                    temp->hide = FALSE;
                }
                break;
            case 'b':
                temp = (PANEL_DATA *)panel_userptr(my_panels[1]);
                if(temp->hide == FALSE)
                {   hide_panel(my_panels[1]);
                    temp->hide = TRUE;
                }
                else
                {   show_panel(my_panels[1]);
                    temp->hide = FALSE;
                }
                break;
            case 'c':
                temp = (PANEL_DATA *)panel_userptr(my_panels[2]);
                if(temp->hide == FALSE)
                {   hide_panel(my_panels[2]);
                    temp->hide = TRUE;
                }
                else
                {   show_panel(my_panels[2]);
                    temp->hide = FALSE;
                }
                break;
        }
        update_panels();
        doupdate();
    }
    endwin();
    return 0;
}
```

```
/* 显示所有窗口 */
void init_wins(WINDOW **wins, int n)
{    int x, y, i;
     char label[80];

     y = 2;
     x = 10;
     for(i = 0; i < n; ++i)
     {    wins[i] = newwin(NLINES, NCOLS, y, x);
          sprintf(label, "Window Number %d", i + 1);
          win_show(wins[i], label, i + 1);
          y += 3;
          x += 7;
     }
}

/* 通过边框和标题显示窗口 */
void win_show(WINDOW *win, char *label, int label_color)
{    int startx, starty, height, width;

     getbegyx(win, starty, startx);
     getmaxyx(win, height, width);

     box(win, 0, 0);
     mvwaddch(win, 2, 0, ACS_LTEE);
     mvwhline(win, 2, 1, ACS_HLINE, width - 2);
     mvwaddch(win, 2, width - 1, ACS_RTEE);

     print_in_middle(win, 1, 0, width, label, COLOR_PAIR(label_color));
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{    int length, x, y;
     float temp;

     if(win == NULL)
          win = stdscr;
     getyx(win, y, x);
     if(startx != 0)
          x = startx;
     if(starty != 0)
          y = starty;
     if(width == 0)
```

```
        width = 80;

    length = strlen(string);
    temp = (width - length)/ 2;
    x = startx + (int)temp;
    wattron(win, color);
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
}
```

16.7、 panel_above()和 panel_below()类函数

函数 panel_above() 和 panel_below() 可以用来查看某一个面板的上面和下面的面板。如果函数的参变量为 NULL，它们就分别返回一个指向最底层和最上层面板的指针。

第十七章：菜单库

菜单库（Menu）很好的扩展了 cureses 基础库。你可以通过这个库所提供的函数方便的创建菜单。同时你一定会定制它的显示效果，让它看起来更美观。那么，我们来看看这个库。

菜单就是一个显示屏幕，里面包含着帮助用户选择的特定子菜单项。简而言之，菜单就是一个菜单项的集合，使你可以方便的从中选择相应的菜单命令。有些读者还不了解多选菜单的强大功能。菜单库提供编制多选菜单的功能。这个我们稍后讨论，现在我们了解一下菜单库的基础知识。

17．1．基础知识

要创建菜单，首先要建立菜单项，之然后通过菜单把菜单项显示出来。接下来，就由功能强大的 **menu_driver()**函数来处理用户的操作。对于处理包含菜单的程序来说，这个函数是一个非常强大的工具。

一个菜单程序的控制流程大致如下：

1. 初始化 curses
2. 用函数 **new_item()**创建菜单项。同时要为菜单项指定名称用来描述相应功能。
3. 用函数 **new_menu()**创建菜单，同时指定要添加的菜单项。
4. 用函数 **menu_post()**递送菜单并刷新屏幕
5. 用一个循环处理用户的菜单请求。并用 **menu_driver()**函数对菜单做必要的更新。
6. 用 **menu_unpost()**取消菜单递送。
7. 用 **free_menu()**释放分配给菜单的内存
8. 用 **free_item()**释放分配给菜单项的内存
9. 结束 curses

现在我们看一个简单的菜单示例程序，这个程序可以用方向键来更新当前菜单项。

17．2 编译包含菜单库的程序

要使用菜单库中的函数，首先要把 menu.h 这个文件包含进去。在编译和连接时就要同时添加-lmenu 和-lncurses 两个参数。

```
#include <menu.h>
    .
    .
.
```

编译和连接: gcc <程序> -lmenu –lncurses

例 18 菜单基础知识示例

```
#include <curses.h>
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD       4
```

```
char *choices[] = {
                        "Choice 1",
                        "Choice 2",
                        "Choice 3",
                        "Choice 4",
                        "Exit",
                  };

int main()
{   ITEM **my_items;
    int c;
    MENU *my_menu;
    int n_choices, i;
    ITEM *cur_item;

    initscr();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));

    for(i = 0; i < n_choices; ++i)
            my_items[i] = new_item(choices[i], choices[i]);
    my_items[n_choices] = (ITEM *)NULL;

    my_menu = new_menu((ITEM **)my_items);
    mvprintw(LINES - 2, 0, "F1 to Exit");
    post_menu(my_menu);
    refresh();

    while((c = getch()) != KEY_F(1))
    {   switch(c)
        {   case KEY_DOWN:
                menu_driver(my_menu, REQ_DOWN_ITEM);
                break;
            case KEY_UP:
                menu_driver(my_menu, REQ_UP_ITEM);
                break;
        }
    }

    free_item(my_items[0]);
```

```
    free_item(my_items[1]);
    free_menu(my_menu);
    endwin();
}
```

这个程序演示了用菜单库创建菜单的基本步骤。首先我们用 **new_item()**建立菜单项，然后用 **new_menu()**函数把它们添加到菜单。当递送菜单并刷新屏幕后，主循环就开始处理了。它读入取用户的输入并进行相应的操作。函数 **menu_driver()**是处理菜单系统的主函数。这个函数的第二个参数用来说明每个菜单的相应操作。**menu_driver()**函数根据参数执行相应的任务。参数的值可以是菜单导览请求，一个 ASCII 码或与鼠标事件相关的一个特定的 KEY_MOUSE 值。

menu_driver()函数可以接受以下导览请求：（就是第二个参数）

REQ_LEFT_ITEM	左移一个菜单项。
REQ_RIGHT_ITEM	右移一个菜单项。
REQ_UP_ITEM	上移一个菜单项。
REQ_DOWN_ITEM	下移一个菜单项。
REQ_SCR_ULINE	向上滚动一行。
REQ_SCR_DLINE	向下滚动一行。
REQ_SCR_DPAGE	下翻一页。
REQ_SCR_UPAGE	上翻一页
REQ_FIRST_ITEM	跳到首项。
REQ_LAST_ITEM	跳到最末一项。
REQ_NEXT_ITEM	跳到下一项。
REQ_PREV_ITEM	跳到上一项。
REQ_TOGGLE_ITEM	选择/取消选择一项。
REQ_CLEAR_PATTERN	清空菜单模式缓冲区。
REQ_BACK_PATTERN	删除菜单模式缓冲区的前面一个字符。
REQ_NEXT_MATCH	跳到下一个与模式匹配的项。
REQ_PREV_MATCH	跳到上一个与模式匹配的项。

千万不要被这么多的操作请求吓倒，稍后我们会一个一个地讲解。在这个例子中，最有趣的是**REQ_UP_ITEM**和**REQ_DOWN_ITEM**。当这两个选项传给**menu_driver()**函数时，**menu_driver()**函数将会通过重新刷新屏幕上移或下移一个菜单项。

17.3 Menu Driver：菜单系统的灵魂

如你在上面的例子中所看到的，menu_driver 在更新菜单时有着举足轻重的作用。所以了解它的各个选项和它们的作用就很有必要了。前面已经解释了，menu_driver()的第二个参数可以是一个移动请求，一个可打印的字符或 KEY_MOUSE 键值。我们来剖析一下各个移动请求的区别。

REQ_LEFT_ITEM 和 REQ_RIGHT_ITEM

一个菜单可以用多列的方式显示菜单项，这可以用函数 menu_format()来实现。当显示一个多列菜单时，这两个移动请求可使 menu drive 在当前菜单项位置左移或右移一个菜单项。

REQ_UP_ITEM 和 REQ_DOWN_ITEM
这两个移动请求在上面的例子中已经出现过。这两个移动请求传递给 menu_driver 时，menu_driver 从当前菜单项上移或下移一个菜单项。

REQ_SCR_* 之类的移动请求
REQ_SCR_ULINE, REQ_SCR_DLINE, REQ_SCR_DPAGE 和 REQ_SCR_UPAGE 是有关屏幕滚动的移动请求。如果所有的菜单项在当前菜单子窗口中显示不完，菜单就是可滚动的了。这些请求传给 menu_driver 时，分别向上/下滚动一行/页。

REQ_FIRST_ITEM, REQ_LAST_ITEM, REQ_NEXT_ITEM 和 REQ_PREV_ITEM
这些移动请求请求的功能从名字上就可以明显的看出来。分别是移动到第一个菜单项、第二个菜单项、下一个菜单项和前一个菜单项。

REQ_TOGGLE_ITEM
当使用这个移动请求时，当前菜单项会被锁定。这个选项仅用于一个多层菜单。因此使用它时须关闭 O_ONEVALUE。这个选项可以通过 **set_menu_opts()**函数关闭或启用。

样式匹配请求
每个菜单都有一个与之关联的匹配缓冲区。通过这个缓冲区，用户可以通过输入 ASCII 字符样式查找与之匹配的菜单项。任何传给 menu_driver 的 ASCII 字符都会被送入样式匹配缓冲区，同时它试着从菜单项列表中查找最近的匹配项。然后立刻从当前菜单项跳转到那个与匹配样式最近的菜单项上去。REQ_CLEAR_PATTERN 请求用来清空匹配缓冲区。REQ_BACK_PATTERN 清除模式缓冲区的前一个匹配样式。如果有多个匹配样式，这些匹配样式就可以通过 REQ_NEXT_PATTERN 和 REQ_PREV_PATTERN 切换，这两个选项分别从当前菜单项移至下/上一个匹配样式。

鼠标事件请求
如果有鼠标事件请求，则根据鼠标的位置来产生相应的事件。在 man 帮助页是这样解释这些行为的：

> *如果第二个参数是 KEY_MOUSE 的键值，相应的鼠标事件就会转化为上面已经定义的请求。现在你只须在用户窗口点击鼠标（如：在菜单显示区或窗口）。如果你是在菜单显示区单击的，将生成 REQ_SCR_ULINE 选项，如果是双击，就会生成 REQ_SCR_UPAGE，如果单击三次，则生成 REQ_FIRST_ITEM。如果你是在菜单显示区域的下方单击的，将生成 REQ_SCR_DLINE，若是双击，就会生成 REQ_SCR_DPAGE，若是单击三次的话，则生成 REQ_LAST_ITEM。如果你单击菜单显示区内的某个菜单项，菜单的提示光标就定位在那个菜单项上了。*

后面的程序中将会对以上的移动请求作适当的讲解。

17.4 含菜单窗口

每个已创建的菜单都对应着一个窗口和一个子窗口。菜单窗口显示菜单的标题或边框线。菜单子窗口显示当前可选的菜单项。但在上面的那个例子里我们并没有指定窗口或子窗口。当窗口未被指定时，stdscr 将作为主窗口。然后菜单系统根据将要显示的菜单项规划计算子窗口的大小。这些菜单项就在规划好的子窗口中显示出来。接下来让我们就来通过

玩转这些窗口：打印一个有边框线和标题的菜单。

例 19. 一个菜单窗口用法的例子

```
#include <menu.h>
#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD       4

char *choices[] = {
                        "Choice 1",
                        "Choice 2",
                        "Choice 3",
                        "Choice 4",
                        "Exit",
                        (char *)NULL,
                  };
void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);

int main()
{   ITEM **my_items;
    int c;
    MENU *my_menu;
    WINDOW *my_menu_win;
    int n_choices, i;

    /* 初始化 curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    init_pair(1, COLOR_RED, COLOR_BLACK);

    /* 创建菜单项 */
    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices, sizeof(ITEM *));
    for(i = 0; i < n_choices; ++i)
        my_items[i] = new_item(choices[i], choices[i]);

    /* 创建菜单 */
    my_menu = new_menu((ITEM **)my_items);

    /* 创建与菜单相关联的窗口*/
    my_menu_win = newwin(10, 40, 4, 4);
    keypad(my_menu_win, TRUE);
```

```
    /* 设置主窗口和子窗口 */
    set_menu_win(my_menu, my_menu_win);
    set_menu_sub(my_menu, derwin(my_menu_win, 6, 38, 3, 1));

    /* 设置字符串的标记为 " * " */
    set_menu_mark(my_menu, " * ");

    /* 在主窗口的边界打印边框线和标题 */
    box(my_menu_win, 0, 0);
    print_in_middle(my_menu_win, 1, 0, 40, "My Menu", COLOR_PAIR(1));
    mvwaddch(my_menu_win, 2, 0, ACS_LTEE);
    mvwhline(my_menu_win, 2, 1, ACS_HLINE, 38);
    mvwaddch(my_menu_win, 2, 39, ACS_RTEE);
    mvprintw(LINES - 2, 0, "F1 to exit");
    refresh();

    /* 递送菜单 */
    post_menu(my_menu);
    wrefresh(my_menu_win);

    while((c = wgetch(my_menu_win)) != KEY_F(1))
    {       switch(c)
            {   case KEY_DOWN:
                menu_driver(my_menu, REQ_DOWN_ITEM);
                break;
                case KEY_UP:
                menu_driver(my_menu, REQ_UP_ITEM);
                break;
            }
            wrefresh(my_menu_win);
    }

    /* 取消递送并释放占用的内存 */
    unpost_menu(my_menu);
    free_menu(my_menu);
    for(i = 0; i < n_choices; ++i)
    free_item(my_items[i]);
    endwin();
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{   int length, x, y;
```

```
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;

    length = strlen(string);
    temp = (width - length)/ 2;
    x = startx + (int)temp;
    wattron(win, color);
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
}
```

这个例子创建了这样一个菜单：有标题、边框，用一根线分隔开标题和菜单项。如你所见，使用函数 set_menu_win()来把一个窗口附加到一个菜单。同样也把子窗口附加进来。这样就可以在子窗口中显示菜单和菜单项了。你也可以使用 set_menu_mark()来设置标志串，让它显示在在所选菜单项的左边。

17.5. 滚动菜单

如果设置的子窗口面积不够显示所有的菜单项，菜单将变成可滚动的。在当前列的最后一个菜单项时，如果你把参数 REQ_DOWN_ITEM 传递过去，它将会变成 REQ_SCR_DLINE，将菜单滚动一个菜单项。你也可以手动地用 REQ_SCR_操作来滚动菜单。我们来看看怎么做。

例 20. 一个滚动菜单的例子

```
#include <curses.h>
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD       4

char *choices[] = {
                        "Choice 1",
                        "Choice 2",
                        "Choice 3",
                        "Choice 4",
                        "Choice 5",
                        "Choice 6",
```

```
                        "Choice 7",
                        "Choice 8",
                        "Choice 9",
                        "Choice 10",
                        "Exit",
                        (char *)NULL,
                };
void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);

int main()
{   ITEM **my_items;
    int c;
    MENU *my_menu;
        WINDOW *my_menu_win;
        int n_choices, i;

    /* 初始化 curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_CYAN, COLOR_BLACK);

    /* 创建菜单项 */
    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices, sizeof(ITEM *));
    for(i = 0; i < n_choices; ++i)
        my_items[i] = new_item(choices[i], choices[i]);

    /* 创建菜单 */
    my_menu = new_menu((ITEM **)my_items);

    /* 创建与菜单相关联的窗口 */
    my_menu_win = newwin(10, 40, 4, 4);
    keypad(my_menu_win, TRUE);

    /* 设置主窗口和子窗口 */
    set_menu_win(my_menu, my_menu_win);
    set_menu_sub(my_menu, derwin(my_menu_win, 6, 38, 3, 1));
    set_menu_format(my_menu, 5, 1);
```

```
    /* 设置标志串为" * " */
    set_menu_mark(my_menu, " * ");

    /* 在主窗口的边界打印边框线和标题 */
    box(my_menu_win, 0, 0);
    print_in_middle(my_menu_win, 1, 0, 40, "My Menu", COLOR_PAIR(1));
    mvwaddch(my_menu_win, 2, 0, ACS_LTEE);
    mvwhline(my_menu_win, 2, 1, ACS_HLINE, 38);
    mvwaddch(my_menu_win, 2, 39, ACS_RTEE);

    /* 传递菜单 */
    post_menu(my_menu);
    wrefresh(my_menu_win);

    attron(COLOR_PAIR(2));
    mvprintw(LINES - 2, 0, "Use PageUp and PageDown to scoll down or up a page of
items");
    mvprintw(LINES - 1, 0, "Arrow Keys to navigate (F1 to Exit)");
    attroff(COLOR_PAIR(2));
    refresh();

    while((c = wgetch(my_menu_win)) != KEY_F(1))
    {       switch(c)
            {   case KEY_DOWN:
                    menu_driver(my_menu, REQ_DOWN_ITEM);
                    break;
                case KEY_UP:
                    menu_driver(my_menu, REQ_UP_ITEM);
                    break;
                case KEY_NPAGE:
                    menu_driver(my_menu, REQ_SCR_DPAGE);
                    break;
                case KEY_PPAGE:
                    menu_driver(my_menu, REQ_SCR_UPAGE);
                    break;
        }
                    wrefresh(my_menu_win);
    }
    /* 取消传递并且释放占用的内存 */
    unpost_menu(my_menu);
    free_menu(my_menu);
    for(i = 0; i < n_choices; ++i)
        free_item(my_items[i]);
    endwin();
```

```
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{    int length, x, y;
     float temp;

     if(win == NULL)
          win = stdscr;
     getyx(win, y, x);
     if(startx != 0)
          x = startx;
     if(starty != 0)
          y = starty;
     if(width == 0)
          width = 80;

     length = strlen(string);
     temp = (width - length)/ 2;
     x = startx + (int)temp;
     wattron(win, color);
     mvwprintw(win, y, x, "%s", string);
     wattroff(win, color);
     refresh();
}
```

这个程序本身已经有了很好地注解。在这个例子中，菜单项的数量增加到了 10 个，大于我们设置的子窗口的大小——6项，这个信息可以通过函数**set_menu_format()**显式地传递给菜单系统。在这里我们指定了在一页中可以显示的行数和列数。如果小于子窗口的高度，我们可以任意地指定每页可以显示的行数和列数。如果用户按了键PAGE UP或PAGE DOWN，请求 REQ_SCR_UPAGE 和 REQ_SCR_PATTERN 就传递给了 **menu_driver()**，菜单则会滚动。

17.6. 多列菜单

在上面的例子中你知道了函数 **set_menu_format()**的用法。我并没有提及列变量（第三个参数）的用法。如果子窗口够宽的话，你可以选择显示每行显示多个菜单项。这可以在列变量里面设置。为简化问题起见，下面的这个例子没有对菜单项进行描述。

例 21. 一个多列菜单的例子

```
#include <curses.h>
#include <menu.h>
#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD       4

char *choices[] = {
                    "Choice 1", "Choice 2", "Choice 3", "Choice 4", "Choice 5",
```

```
                "Choice 6", "Choice 7", "Choice 8", "Choice 9", "Choice 10",
                "Choice 11", "Choice 12", "Choice 13", "Choice 14", "Choice 15",
                "Choice 16", "Choice 17", "Choice 18", "Choice 19", "Choice 20",
                "Exit",
                (char *)NULL,
        };

int main()
{   ITEM **my_items;
    int c;
    MENU *my_menu;
    WINDOW *my_menu_win;
    int n_choices, i;

    /* 初始化 curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_CYAN, COLOR_BLACK);

    /* 创建菜单项 */
        n_choices = ARRAY_SIZE(choices);
        my_items = (ITEM **)calloc(n_choices, sizeof(ITEM *));
        for(i = 0; i < n_choices; ++i)
                my_items[i] = new_item(choices[i], choices[i]);

    /* 创建菜单 */
    my_menu = new_menu((ITEM **)my_items);

    /* 设置菜单选项，不显示描述 */
    menu_opts_off(my_menu, O_SHOWDESC);

    /* 创建与菜单相关联的窗口 */
    my_menu_win = newwin(10, 70, 4, 4);
    keypad(my_menu_win, TRUE);

    /* 设置主窗口和子窗口 */
    set_menu_win(my_menu, my_menu_win);
    set_menu_sub(my_menu, derwin(my_menu_win, 6, 68, 3, 1));
    set_menu_format(my_menu, 5, 3);
    set_menu_mark(my_menu, " * ");
```

```
/* 在主窗口的边界打印边框线和标题 */
box(my_menu_win, 0, 0);

attron(COLOR_PAIR(2));
mvprintw(LINES - 3, 0, "Use PageUp and PageDown to scroll");
mvprintw(LINES - 2, 0, "Use Arrow Keys to navigate (F1 to Exit)");
attroff(COLOR_PAIR(2));
refresh();

/* 传递菜单 */
post_menu(my_menu);
wrefresh(my_menu_win);

while((c = wgetch(my_menu_win)) != KEY_F(1))
{       switch(c)
        {   case KEY_DOWN:
            menu_driver(my_menu, REQ_DOWN_ITEM);
            break;
        case KEY_UP:
            menu_driver(my_menu, REQ_UP_ITEM);
            break;
        case KEY_LEFT:
            menu_driver(my_menu, REQ_LEFT_ITEM);
            break;
        case KEY_RIGHT:
            menu_driver(my_menu, REQ_RIGHT_ITEM);
            break;
        case KEY_NPAGE:
            menu_driver(my_menu, REQ_SCR_DPAGE);
            break;
        case KEY_PPAGE:
            menu_driver(my_menu, REQ_SCR_UPAGE);
            break;
    }
    wrefresh(my_menu_win);
}

/* 取消传递并释放占用的内存 */
unpost_menu(my_menu);
free_menu(my_menu);
for(i = 0; i < n_choices; ++i)
free_item(my_items[i]);
endwin();
```

```
}
```

我们仔细看一下函数 **set_menu_format()**的调用。它指定行数为 3，因此每行就显示 3 个菜单项。我们已经用函数 **menu_opts_off()** 关闭了描述显示。其他几个函数：**set_menu_opts()**，**menu_opts_on()**和 **menu_opts()**也可以用来控制菜单选项。下面的菜单选项可以用来指定：

O_ONEVALUE
在菜单中只能选一个菜单项。

O_SHOWDESC
当菜单被传递时显示对菜单项的描述。

O_ROWMAJOR
以行为主序显示菜单。

O_IGNORECASE
在样式匹配时忽略大小写。

O_SHOWMATCH
样式匹配时光标移到菜单项上。

O_NONCYCLIC
上移或下移一个菜单项时，不能跳到菜单另一端的菜单项上去。

默认情况下，这些选项都是打开的。你可以通过函数 **menu_opts_on()**和 **menu_opts_off()** 用来开关某个属性。你也可以使用函数 **set_menu_opts()**直接指定这些选项。这个函数的参数应该是上面那些常量的或（OR）值。函数 **menu_opts()**可以用来查看当前菜单的选项。

17.7. 多选菜单

把选项 O_ONEVALUE 关闭之后会是什么情况呢？菜单变成了多项选择的了。这意味着你可以同时选择多个菜单项。这就把我们的注意力转向了 REQ_TOGGLE_ITEM 请求的用法，一起来看看吧：

例 22. 一个多选菜单的例子

```
#include <curses.h>
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD       4

char *choices[] = {
                        "Choice 1",
                        "Choice 2",
                        "Choice 3",
                        "Choice 4",
```

```
                        "Choice 5",
                        "Choice 6",
                        "Choice 7",
                        "Exit",
                };

int main()
{   ITEM **my_items;
    int c;
    MENU *my_menu;
    int n_choices, i;
    ITEM *cur_item;

    /* 初始化 curses */
    initscr();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* 初始化菜单菜单项 */
    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));
    for(i = 0; i < n_choices; ++i)
    my_items[i] = new_item(choices[i], choices[i]);
    my_items[n_choices] = (ITEM *)NULL;

    my_menu = new_menu((ITEM **)my_items);

    /* 设置菜单为多值菜单 */
    menu_opts_off(my_menu, O_ONEVALUE);

    mvprintw(LINES - 3, 0, "Use <SPACE> to select or unselect an item.");
    mvprintw(LINES - 2, 0, "<ENTER> to see presently selected items(F1 to Exit)");
    post_menu(my_menu);
    refresh();

    while((c = getch()) != KEY_F(1))
    {       switch(c)
            {   case KEY_DOWN:
                menu_driver(my_menu, REQ_DOWN_ITEM);
                break;
            case KEY_UP:
                menu_driver(my_menu, REQ_UP_ITEM);
                break;
```

```
            case ' ':
                menu_driver(my_menu, REQ_TOGGLE_ITEM);
                break;
            case 10:    /* Enter 键*/
            {   char temp[200];
                ITEM **items;

                items = menu_items(my_menu);
                temp[0] = '\0';
                for(i = 0; i < item_count(my_menu); ++i)
                    if(item_value(items[i]) == TRUE)
                    {   strcat(temp, item_name(items[i]));
                        strcat(temp, " ");
                    }
                move(20, 0);
                clrtoeol();
                mvprintw(20, 0, temp);
                refresh();
            }
            break;
        }
    }

    free_item(my_items[0]);
    free_item(my_items[1]);
    free_menu(my_menu);
    endwin();
}
```

Woo！出现了好多新函数。我们一个一个来看吧。先看 REQ_TOGGLE_ITEM。在一个多选菜单里，用户可以选择多个菜单项。请求 REQ_TOGGLE_ITEM 锁定当前选择的菜单项。在这种情况下，当按下空格键时，请求 REQ_TOGGLE_ITEM 就被传给 menu_driver，从而锁定某个菜单项。

如果用户按下了<ENTER>键，则显示当前已选的菜单项。首先，用函数 **menu_items()**找到与菜单相关联的菜单项。然后循环核对这些菜单项，看某个菜单项是否已选。如果一个菜单项已选，函数 **item_value()**返回 TRUE。函数 **item_count()**返回菜单的菜单项数。菜单项的名字可以通过函数 **item_name()**找到。你也可以用函数 **item_description()**来得知一个菜单项的相关描述。

17.8 菜单选项

好了，现在你一定非常想在你的菜单里弄出点花样来，且使得菜单功能更加强大。我知道，你要让菜单变成彩色的！！！你希望创建一个好看一点的菜单，就像这里的文本模式 DOS 游戏。函数 **set_menu_fore()**和 **set_menu_back()**可以用来改变选中和未选中菜单项的属性。这两个函数的名字可能会引起误会。它们并不会改变菜单的前景色与背景色哦！

函数 **set_menu_grey()**可以用来设置菜单中未选菜单项的属性。这把我们引向了仅为单个

菜单项的进行设置的选项 O_SELECTABLE。我们可以用函数 **item_opts_off()**来关闭这个选项，然后那个菜单项就不可选了。在菜单窗口中它就变成灰色的。让我们把这些概念运用到下面的这个例子中：

例 23. 一个菜单选项的例子

```
#include <menu.h>
#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD       4

char *choices[] = {
                        "Choice 1",
                        "Choice 2",
                        "Choice 3",
                        "Choice 4",
                        "Choice 5",
                        "Choice 6",
                        "Choice 7",
                        "Exit",
                  };

int main()
{    ITEM **my_items;
     int c;
     MENU *my_menu;
     int n_choices, i;
     ITEM *cur_item;

     /* 初始化 curses */
     initscr();
     start_color();
     cbreak();
     noecho();
     keypad(stdscr, TRUE);
     init_pair(1, COLOR_RED, COLOR_BLACK);
     init_pair(2, COLOR_GREEN, COLOR_BLACK);
     init_pair(3, COLOR_MAGENTA, COLOR_BLACK);

     /* 初始化菜单项 */
     n_choices = ARRAY_SIZE(choices);
     my_items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));
     for(i = 0; i < n_choices; ++i)
          my_items[i] = new_item(choices[i], choices[i]);

     my_items[n_choices] = (ITEM *)NULL;
     item_opts_off(my_items[3], O_SELECTABLE);
```

```
    item_opts_off(my_items[6], O_SELECTABLE);

    /* 创建菜单 */
    my_menu = new_menu((ITEM **)my_items);

    /* 为菜单设置前景和背景 */
    set_menu_fore(my_menu, COLOR_PAIR(1) | A_REVERSE);
    set_menu_back(my_menu, COLOR_PAIR(2));
    set_menu_grey(my_menu, COLOR_PAIR(3));

    /* 传递菜单 */
    mvprintw(LINES - 3, 0, "Press <ENTER> to see the option selected");
    mvprintw(LINES - 2, 0, "Up and Down arrow keys to naviage (F1 to Exit)");
    post_menu(my_menu);
    refresh();

    while((c = getch()) != KEY_F(1))
    {       switch(c)
            {   case KEY_DOWN:
                menu_driver(my_menu, REQ_DOWN_ITEM);
                break;
            case KEY_UP:
                menu_driver(my_menu, REQ_UP_ITEM);
                break;
            case 10: /* Enter 键*/
                move(20, 0);
                clrtoeol();
                mvprintw(20, 0, "Item selected is : %s",
                item_name(current_item(my_menu)));
                pos_menu_cursor(my_menu);
                break;
        }
    }
    unpost_menu(my_menu);
    for(i = 0; i < n_choices; ++i)
    free_item(my_items[i]);
    free_menu(my_menu);
    endwin();
}
```

17.9. 大有作为的用户指针

我们可以把一个用户指针和菜单中的一个菜单项联系起来。它的工作原理与面板里的用户指针一样，并不触及菜单系统。你可以在那里面存储任何你想要的东西。我经常用它来存储所选菜单选项要执行的函数（已选定并且用户按下了<ENTER>键）。

例 24. 菜单用户指针的用法

```
#include <curses.h>
#include <menu.h>
#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD       4

char *choices[] = {
                        "Choice 1",
                        "Choice 2",
                        "Choice 3",
                        "Choice 4",
                        "Choice 5",
                        "Choice 6",
                        "Choice 7",
                        "Exit",
                  };
void func(char *name);

int main()
{    ITEM **my_items;
     int c;
     MENU *my_menu;
         int n_choices, i;
     ITEM *cur_item;

     /* 初始化 curses */
     initscr();
     start_color();
     cbreak();
     noecho();
     keypad(stdscr, TRUE);
     init_pair(1, COLOR_RED, COLOR_BLACK);
     init_pair(2, COLOR_GREEN, COLOR_BLACK);
     init_pair(3, COLOR_MAGENTA, COLOR_BLACK);

     /* 初始化菜单项 */
     n_choices = ARRAY_SIZE(choices);
     my_items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));
     for(i = 0; i < n_choices; ++i)
     {
          my_items[i] = new_item(choices[i], choices[i]);
          /* 设置用户指针 */
          set_item_userptr(my_items[i], func);
     }
```

```
    my_items[n_choices] = (ITEM *)NULL;

    /* 创建菜单 */
    my_menu = new_menu((ITEM **)my_items);

    /* 传递菜单 */
    mvprintw(LINES - 3, 0, "Press <ENTER> to see the option selected");
    mvprintw(LINES - 2, 0, "Up and Down arrow keys to naviage (F1 to Exit)");
    post_menu(my_menu);
    refresh();

    while((c = getch()) != KEY_F(1))
    {       switch(c)
            {   case KEY_DOWN:
                    menu_driver(my_menu, REQ_DOWN_ITEM);
                    break;
                case KEY_UP:
                    menu_driver(my_menu, REQ_UP_ITEM);
                    break;
                case 10: /* Enter 键 */
                {   ITEM *cur;
                    void (*p)(char *);

                    cur = current_item(my_menu);
                    p = item_userptr(cur);
                    p((char *)item_name(cur));
                    pos_menu_cursor(my_menu);
                    break;
                }
                break;
        }
    }
    unpost_menu(my_menu);
    for(i = 0; i < n_choices; ++i)
    free_item(my_items[i]);
    free_menu(my_menu);
    endwin();
}

void func(char *name)
{   move(20, 0);
    clrtoeol();
    mvprintw(20, 0, "Item selected is : %s", name);
}
```

第十八章 表单库

当你看到网页上那些处理用户数据的表单后，你肯定很想在纯文本模式下创建一个那样的表单。用纯 ncurses 创建那样的表单十分困难。但是表单库（Form Library）为我们提供了一个基础的编写框架，使我们可以很容易地创建和维护表单。它包含了很多的函数，分别用来管理，动态扩展表单域等等。我们将通过后面的内容来掌握它。

表单就是表单域的集合，也许这些表单域是一个标签，也许是一项数据的输入框。表单库也提供把表单划分为页面的函数。

18.1. 基础知识

表单的创建步骤与菜单的创建大致相同：
首先，用 **new_field()**创建与表单相关联的表单域。你可以设置表单域的修饰选项，这样的界面显示出来会更加美观。离开表单域以前，这些对表单域的修饰属性都是有效的。
然后，表单域添加进表单，整个表单就会显示到屏幕上，准备读取输入信息。与 **menu_driver()**相似，表单由 **form_driver()**操纵。我们可以传递请求给 **form_driver()**来移动显示焦点（校者注：即表示被选定，通常是高亮条等）到某个表单域，或将光标移动到表单域的结尾等等。用户在表单域内输入并确认后结束，表单就不再显示，并释放占用过的内存。

一个表单程序的流程大致如下：
初始化 curses
通过 **new_field()**创建表单域。你可以指定域的高度，宽度以及它在表单中的位置。
指定表单域所作用的表单，并用 **new_form()**创建表单。
用 **form_post()**来递送表单，并刷新屏幕。
用一个循环来处理用户请求，通过 **form_driver()**对表单做相应的更新。
用 **form_unpost()**取消表单的递送。
用 **free_form()**释放已分配给表单的内存。
释放已分配给菜单项的内存。
结束 curses

如你所见，表单库的用法与跟菜单库的用法很像。下面的几个例子将会让我们领略表单处理的其他方面。先让我们从一个简单的例子开始吧：

18.2. 编译包含表单库的程序

要使用表单库中的函数，必须要把 form.h 头文件包含进源程序代码。在编译和连接时要同时添加 –lform 和 –lncurses 两个选项。

```
#include <form.h>
    .
    .
    .
```

编译和链接: gcc <程序文件> -lform –lncurses

例 25. 表单库基础知识

```
#include <form.h>

int main()
{   FIELD *field[3];
    FORM  *my_form;
    int ch;

    /* 初始化 curses */
    initscr();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* 初始化表单域 */
    field[0] = new_field(1, 10, 4, 18, 0, 0);
    field[1] = new_field(1, 10, 6, 18, 0, 0);
    field[2] = NULL;

    /* 设置表单域 */
    set_field_back(field[0], A_UNDERLINE);    /* 为选项打印一条下滑线 */
    field_opts_off(field[0], O_AUTOSKIP);
    /* 在域（输入框）填满后光标不会自动跳到下一个表单域 */
    set_field_back(field[1], A_UNDERLINE);
    field_opts_off(field[1], O_AUTOSKIP);

    /* 创建并递送表单 */
    my_form = new_form(field);
    post_form(my_form);
    refresh();

    mvprintw(4, 10, "Value 1:");
    mvprintw(6, 10, "Value 2:");
    refresh();

    /* 用循环获取用户请求 */
    while((ch = getch()) != KEY_F(1))
    {   switch(ch)
        {   case KEY_DOWN:
                /* 跳至下一个表单域 */
                form_driver(my_form, REQ_NEXT_FIELD);
                /* 跳到当前缓冲的末尾 */
```

```
                /* 精确地在输入最后一个后字符跳出这个表单域 */
                form_driver(my_form, REQ_END_LINE);
                break;
            case KEY_UP:
                /* 移动到前一个表单域 */
                form_driver(my_form, REQ_PREV_FIELD);
                form_driver(my_form, REQ_END_LINE);
                break;
            default:
                /* 如果输入的是普通字符，就把它打印出来 */
                form_driver(my_form, ch);
                break;
        }
    }
    /* 取消表单并释放内存 */
    unpost_form(my_form);
    free_form(my_form);
    free_field(field[0]);
    free_field(field[1]);

    endwin();
    return 0;
}
```

上面的程序比较简单易懂。它首先调用 **new_field()**函数创建了两个表单域。**new_field()**的六个参数分别确定了表单域的高、宽、起始位置的纵坐标、横坐标、不在显示的行数，以及附加的工作缓冲区。其中第五个参数（不显示的行数）决定了表单域哪些部分的是可见的。如果是 0，则显示整个表单域。当用户访问的表单域超出了显示范围，那么表单将变成可滚动的。之后，表单库分别为每个表单分配了缓冲区，用来存储用户输入的数据。**new_field()**的最后一个参数用来分配额外的缓冲区。这些缓冲区可以用来做其它的事情。

当创建表单域后，用 **set_field_back()**函数可以增添背景修饰效果。选项 **AUTOSKIP** 已用 **field_opts_off()**关闭。如果这个选项是打开的，一旦当前的表单域被输入的数据填满，则光标自动跳转到下一个表单域。

给表单添加完表单域后，就把表单递送出去。接着，通过对 **form_driver()**发送相应的操作请求， while 循环里将处理用户输入的信息。有关**form_driver()**的细节将在后面的部分详细解释。

18.3. 玩转表单域

每个表单域关联了大量的属性。那样可意使你定制它们从而得到你想要的效果。是不是很有趣？那还等什么呢？

18.3.1. 获取域的大小和位置

我们可以用函数 **field_info()**得到已创建表单域的参数。该函数返回表单域的高、宽、起始位置的纵坐标、横坐标、不显示的行数和附加的缓冲区。它的顺序跟 **new_field()**的相反：

```
int field_info( FIELD *field,                  /* 要获取信息的表单域*/
                int *height, *int width,       /* 域的高、宽 */
                int *top, int *left,           /* 起点的 y 坐标，起点的 x 坐标 */
                int *offscreen,                /* 不在显示范围内的行数 */
                int *nbuf);                    /* 附加缓冲区的大小 */
```

18.3.2. 移动表单域

通过函数 **move_field()**可以移动表单域的位置。

```
int move_field(   FIELD *field,                /* 要移动的表单域*/
                  int top, int left);          /* 新位置的起点坐标（先行后列） */
```

同样的，改变后的位置也可以通过 **field_info()**查询到。

18.3.3. 设置表单域的对齐方式

输入表单域数据的对齐方式可以用函数 **set_field_just()**设置。

```
int set_field_just(FIELD *field,               /* 要更改的表单域 */
                   int justmode);              /* 要设置的对齐方式 */

int field_just(FIELD *field);                  /* 查询表单域的对齐方式 */
```

这两个函数可以用来设置或者返回相应表单域对齐方式的值，分别为为 **NO_JUSTIFICATION**（没有对齐方式），**JUSTIFY_RIGHT**（右对齐），**JUSTIFY_LEFT**（左对齐） 或者 **JUSTIFY_CENTER**（居中对齐）。

18.3.4. 表单域的显示修饰

正像前面提到的那样，在上面的例子中，表单域的修饰效果可以用 **set_field_fore()**和 **set_field_back()**两个函数来设置。这两个函数分别设置表单域的前景和背景修饰。你也可以指定用字符填充该表单域的空白处。通过调用函数 **set_field_pad()**你就可以设置背景填充字符了。默认背景填充字符为空格。函数 **field_fore()**，**field_back()**，**field_pad()**可以用来查询当前表单域的前景、背景修饰和背景填充字符。以下是这些函数的原形：

```
int set_field_fore(FIELD *field,               /* 要设置的表单域 */
                   chtype attr);               /* 被设置表单域的前景修饰属性 */

chtype field_fore(FIELD *field);               /* 要查询的表单域 */
                                               /* 返回该表单域的前景修饰属性 */

int set_field_back(FIELD *field,               /* 要设置的表单域 */
                   chtype attr);               /* 被设置表单域的背景修饰属性 */

chtype field_back(FIELD *field);               /* 要查询的表单域 */
                                               /* 返回该表单域的背景修饰属性 */

int set_field_pad(FIELD *field,                /* 要设置的表单域*/
```

```
                    int pad);              /* 要设置表单域的背景填充字符 */
chtype field_pad(FIELD *field);            /* 要查询的表单域*/
                                           /* 返回该表单域的背景填充字符 */
```

尽管上面的函数看起来很简单，但用 **set_field_fore()**来设置颜色对初学者来说可能很困难。我先解释一下表单域的前景和背景修饰。前景修饰与字符有关。即在表单域内以**set_field_for()**设置字符的修饰效果。背景修饰用来填充表单域的背景，不管表单域内是否有字符。由于颜色通常是成对定义，怎样才能正确显示包含颜色设置的表单域呢？下面的例子很清楚地解释了颜色定义。

例 26. 一个有关表单域属性的例子

```
#include <form.h>
int main()
{   FIELD *field[3];
    FORM *my_form;
    int ch;

    /* 初始化 curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* 初始化颜色 */
    init_pair(1, COLOR_WHITE, COLOR_BLUE);
    init_pair(2, COLOR_WHITE, COLOR_BLUE);

    /* 初始化表单域 */
    field[0] = new_field(1, 10, 4, 18, 0, 0);
    field[1] = new_field(1, 10, 6, 18, 0, 0);
    field[2] = NULL;

    /* 设置表单域选项 */
    set_field_fore(field[0], COLOR_PAIR(1));   /* 设置表单域背景色为蓝色 */
    set_field_back(field[0], COLOR_PAIR(2));  /* 白色为前景色（字符为白色）
*/
    field_opts_off(field[0], O_AUTOSKIP);
    /* 当这个域填满时不自动跳到下一个表单域*/

    set_field_back(field[1], A_UNDERLINE);
    field_opts_off(field[1], O_AUTOSKIP);

    /* 创建表单并递送 */
    my_form = new_form(field);
```

```
    post_form(my_form);
    refresh();

    set_current_field(my_form, field[0]); /* 把此表单域设为当前的活动域 */
    mvprintw(4, 10, "Value 1:");
    mvprintw(6, 10, "Value 2:");
    mvprintw(LINES - 2, 0, "Use UP, DOWN arrow keys to switch between fields");
    refresh();

    /* 用循环获取用户请求 */
    while((ch = getch()) != KEY_F(1))
    {   switch(ch)
        {   case KEY_DOWN:
                /* 跳到下一个表单域 */
                form_driver(my_form, REQ_NEXT_FIELD);
                /* 跳到当前缓冲区的末尾 */
                /* 在输入最后一个字符后跳出 */
                form_driver(my_form, REQ_END_LINE);
                break;

            case KEY_UP:
                /* 跳到前一个表单域 */
                form_driver(my_form, REQ_PREV_FIELD);
                form_driver(my_form, REQ_END_LINE);
                break;

            default:
                /* 如果是正常的字符，则读入并打印 */
                form_driver(my_form, ch);
                break;
        }
    }

    /* 取消表单递送并释放内存 */
    unpost_form(my_form);
    free_form(my_form);
    free_field(field[0]);
    free_field(field[1]);

    endwin();
    return 0;
}
```

仔细研究这些颜色变量对并深刻的理解前景和背景修饰。在我的程序里面，若要设置颜色

属性，我一般只用 **set_field_back()**来设置背景修饰。cureses 不允许自定义颜色。

18.3.5. 表单域属性

表单的处理是通过对表单域设定属性实现的，有大量的表单域属性对应表单处理的各个方面。你可以用下面的函数为表单域设定属性：

```
int set_field_opts(FIELD *field,          /* 要变更的表单域 */
                   int attr);             /* 要设定的属性 */

int field_opts_on(FIELD *field,           /* 要改变选项的表单域 */
                  int attr);              /* 要启用的属性 */

int field_opts_off(FIELD *field,          /* 要改变选项的表单域 */
                   int attr);             /* 要关闭的属性 */

int field_opts(FIELD *field);             /* 要查询的表单域 */
```

函数 **set_field_opts()**可以直接用来设置表单域属性。要启用或关闭某些属性的话，你可以用函数 **field_opts_on()**或 **field_opts_off()**来完成。你可以随时用函数 **field_opts()**来查询某个表单域的属性。下面列出了所有可用的属性，默认情况下，它们都是已启用的。

O_VISIBLE

控制表单域是否在屏幕上可见。可以用来显示或隐藏某些表单域以及附加表单域。

O_ACTIVE

控制表单域在表单处理时是否是激活状态（如：类似 Tab 键在域之间跳转访问）。这样可以使应用程序创建带有可变缓冲区的标签或起始表单域。

O_PUBLIC

控制输入时是否在表单域中显示输入数据。如果关闭了某个表单域的这个属性，表单系统仍然允许编辑这个表单域的数据。但不会在这个可视表单域中显示出来，同时光标也不会移动。你可以通过关闭 O_PUBLIC 选项来定义一个密码域。

O_EDIT

控制表单域的数据能否被修改。当这个选项关闭时，除了 **REQ_PREV_CHOICE** 和 **REQ_NEXT_CHOICE** 以外，其它所有的编辑请求都将失效。可以设置类似这样的只读表单域用来显示帮助信息。

O_WRAP

控制多行表单域的自动换行。通常，当一个单词（用空格作为单词之间的间隔）的任意一个字符到了当前行的末尾时，整个单词会自动换行显示（如果有下一行的话）。当这个选项关闭时，这个单词会在行尾处断开。

O_BLANK

清空表单域。当此选项打开后，在表单域的起始位置输入字符时，程序会自动清空整个域

(除了刚输入的这个字符)。

O_AUTOSKIP
控制当前表单域填满时是否自动跳转到下一个表单域。通常，当用户的输入的数据超出了的一个表单域所能容纳的数据时，光标会自动跳转到下一个表单域。如果这个选项已关闭，则光标停在表单域的末尾。对于没有大小限制的动态表单域，这个选项会被忽略。

O_NULLOK
是否检查空白表单域的有效性，默认为不检查。用户可以在退出时让一个表单域为空，而不用像通常那样检查这个表单域是否有效。如果某个表单域的这个选项是关闭的，退出程序时就要检查表单域的有效性了。

O_PASSOK
控制表单有效性检查发生的时间。默认是在表单域的数据被修改后检查。设置以后，用户在退出表单域后检查表单域的有效性。当你需要在处理表单时同时更改表单域的有效性检查函数，设置 **O_PASSOK** 选项就对了。

O_STATIC
控制表单域的长度是否固定为初始长度。如果关闭这个选项，则表单域就变成动态的。系统会根据输入数据的长度自动调整表单域大小。

如果当前的表单域为选定状态时，表单域的属性不能修改。但是，可以在表单域递送后修改属性。

这些属性都是位掩码，可以用“或”逻辑运算符组合属性。你已经看见了关闭 **O_AUTOSKIP** 属性后的效果。下面的这个例子说明了更多属性的用法。其他的属性会在后面适当的地方说明。

例 27. 表单域属性用法示例

```
#include <form.h>

#define STARTX 15
#define STARTY 4
#define WIDTH 25

#define N_FIELDS 3

int main()
{   FIELD *field[N_FIELDS];
    FORM  *my_form;
    int ch, i;
    /* 初始化 curses */
    initscr();
    cbreak();
```

```
noecho();
keypad(stdscr, TRUE);

/* 初始化表单域 */
for(i = 0; i < N_FIELDS - 1; ++i)
    field[i] = new_field(1, WIDTH, STARTY + i * 2, STARTX, 0, 0);
field[N_FIELDS - 1] = NULL;

/* 设置表单域选项 */
set_field_back(field[1], A_UNDERLINE);     /* 为选项打印下划线 */
field_opts_off(field[0], O_ACTIVE);        /* 这个表单域是静态标签 */
field_opts_off(field[1], O_PUBLIC);        /* 这个表单域类似密码域 */
field_opts_off(field[1], O_AUTOSKIP);
/* 防止在输入完最后一个字符后还是在同一个域输入*/

/* 创建表单并递送 */
my_form = new_form(field);
post_form(my_form);
refresh();

set_field_just(field[0], JUSTIFY_CENTER); /* 表单域输入时居中对齐 */
set_field_buffer(field[0], 0, "This is a static Field "); /* 初始化表单域 */
mvprintw(STARTY, STARTX - 10, "Field 1:");
mvprintw(STARTY + 2, STARTX - 10, "Field 2:");
refresh();

/* 循环读取用户请求 */
while((ch = getch()) != KEY_F(1))
{   switch(ch)
    {   case KEY_DOWN:
            /* 跳到下一个表单域 */
            form_driver(my_form, REQ_NEXT_FIELD);
            /* 跳到当前缓冲的末尾 */
            /* 在最后一个字符处跳出 */
            form_driver(my_form, REQ_END_LINE);
            break;
        case KEY_UP:
            /* 跳到前一个表单域 */
            form_driver(my_form, REQ_PREV_FIELD);
            form_driver(my_form, REQ_END_LINE);
            break;
        default:
            /* 若是普通的字符，则打印出来 */
            form_driver(my_form, ch);
```

```
                break;
            }
        }

        /* 取消表单传递并释放内存 */
        unpost_form(my_form);
        free_form(my_form);
        free_field(field[0]);
        free_field(field[1]);

        endwin();
        return 0;
}
```

这个例子虽然用处不大，但它演示了这些属性的用法和效果。如果恰到好处地使用这些属性，可以在有效地在表单中显示信息。第二个表单域关闭了 **O_PUBLIC** 选项，所以你看不到输入的字符。

18.3.6. 获取表单域的状态

表单域的状态用来标记表单域的是否被修改。初始值为 **FALSE**，当用户在表单域中输入数据，同时也修改了数据缓冲区后，该值就变为 **TRUE**。因此，可以通过查询一个表单域的状态来看检测它是否被修改了。下面的函数支持以上的功能。

```
int set_field_status(FIELD *field,          /* 要改变状态的表单域 */
                     int status);           /* 要设置的表单域状态 */

int field_status(FIELD *field);             /* 要取得状态的表单域*/
```

你的程序最好能够在离开一个表单域的时候检查一下表单域的状态。因为表单域的有效性还没有确定，所以数据缓冲区可能还没有更新。为确保返回正确的表单域状态，最好在下面三种情况调用 **field_status()**函数：（1）、退出表单域并检查有效性时；（2）、在表单域或表单初始化或退出时；（3）、Form Driver 处理 **REQ_VALIDATION** 请求后。

18.3.7、用户指针

每个表单域的结构都包含一个用户指针，这个指针可以使用户做很多事情。它不能被表单库直接访问，但可以被用户任意使用。下面的函数可以设置和读取用户指针。

```
int set_field_userptr(FIELD *field,
                      char *userptr);       /* 与这个表单域关联的用户指针 */
char *field_userptr(FIELD *field);          /* 要获取指针的表单域 */
```

18.3.8、动态表单域

假如你需要一个可以动态改变大小的表单域的话，那么这正是你需要用到的功能。当用户输入的数据大小超过该表单域的原大小时，文本框的尺寸就会动态的改变。根据文本框的位置，文本框可以自动的水平或垂直滚动，以适应新增添的数据。

为使一个表单域的宽度可以动态地改变，应该先关闭 **O_STATIC** 选项。这可以通过调用下面函数完成：

```
field_opts_off(field_pointer, O_STATIC);
```

但是通常不允许一个表单域无限制地增长。可以设置表单域动态增长的最大宽度：

```
int set_max_field(FIELD *field,          /* 要进行设置的表单域 */
                  int max_growth);       /* 表单域动态增长的最大宽度 */
```

可以通过下面的函数访问动态增长的表单域：

```
int dynamic_field_info( FIELD *field,    /* 要进行查询的表单域 */
                        int  *prows,     /* 表单域的行数 */
                        int  *pcols,     /* 表单域的列数 */
                        int   *pmax)     /* 表单域动态增长的最大宽度 */
```

作者建议使用这个函数来检查动态表单域的属性。

再次调用 **new_field()**函数，函数将会自动创建高度为一个字符的单行表单域。创建初始的高度大于一个字符的表单域自动被定义为多行表单域。

一个关闭了 **O_STATIC** 选项的单行表单域（动态增长的表单域）只包含一行，但是如果用户输入的数据的大小超过初始的大小的话，表单域宽度可以自动增长。显示的宽度仍然是固定的，但是可以通过水平滚动来查看超出部分的数据。

如同单行表单域一样，一个关闭了 **O_STATIC** 选项的多行表单域（动态增长的表单域）包含固定的宽度，但是如果用户输入的数据大小超过初始的大小的话，行数可以自动增长。可显示的宽度仍然是固定的，但可以通过纵向滚动来查看超出部分的数据。

上面两段非常形象的描述了表单域的动态增长行为。

下面介绍表单库的其他行为：

1、 如果某个表单域的 **O_STATIC** 选项关闭，也没有设置表单域动态增长的最大限度，**O_AUTOSKIP** 选项将被忽略。通常情况下，当用户在表单域的最后一个位置输入数据，**O_AUTOSKIP** 会自动向 Form Driver 发送 **REQ_NEXT_FIELD** 请求。但是，在一个没有设定最大增长值的动态表单域里，就没有最末的位置。但是如果定义了最大增长值，如果表单域自动增长到最大限度，**O_AUTOSKIP** 选项也将正常工作。

2、 如果 **O_STATIC** 选项关闭，表单域的对齐请求也将被忽略。通常的，**set_field_just** 可以通过 **JUSTIFY_LEFT**，**JUSTIFY_RIGHT**，**JUSTIFY_CENTER** 来设置单行域的对齐方式。但是从定义中我们知道：一个动态增长的单行表单域包含比初始长度更多的数据，并且可以通过水平滚动查看这些数据。因此，这些都超过了 **set_field_just** 作用的范围。所以函数 **field_jus** 不会改变返回的结果。

3、 如果 **O_STATIC** 选项关闭，并且没有为表单域定义最大动态增长范围，对 Form Driver 发送的 **REQ_NEW_LINE** 请求将会忽略，但 **O_NL_OVERLOAD** 选项正常执行。这时，如果表单选项 **O_NL_OVERLOAD** 是打开的，如果在表单域的最后一行发送 **REQ_NEW_LINE** 请求系统将会隐式地生成并发送一个 **REQ_NEXT_FIELD** 请求。

如果一个表单域可以无限制地动态增长，则没有最后一行。因此发送 **REQ_NEW_LINE** 请求就不会隐式生成 **REQ_NEXT_FIELD** 请求了。如果指定了最大动态增长的范围，并且 **O_NL_OVERLOAD** 选项是打开的。如果在表单域自动增长到最大，且用户停留在最后一行的时候就可以发送 **REQ_NEW_LINE** 请求，系统会隐式的产生并发送 **REQ_NEXT_FIELD** 请求。

4、 调用库函数 **dup_field()**仍然会正常工作；它将复制这个表单域，包括当前缓冲区的大小和表单域的内容。如果指定了表单域的最大动态增长值，也将会一起被复制。

5、 调用库函数 **link_field()**仍然会正常工作；它将复制表单域的所有修饰属性和所连接表单域的共享缓冲区。如果 **O_STATIC** 选项被随后的某个表单域的共享缓冲区修改了，并且输入的数据超出了当前缓冲区能够容纳的数据，那么系统对此的反应将取决于当前域的属性。

6、 调用库函数 **field_info()**仍然会正常工作；变量 **nrow** 将包含开始调用 **new_field** 时的值。用户应该用上面叙述的 **dynamic_field_info** 函数来查询缓冲区的当前大小。

上面的有些要点只有详细了解释 Form Driver 后才好理解。我们会在后面的几个部分深入讲解 Form Driver。

18.4、表单窗口

表单窗口的概念与菜单窗口的很相似。每个表单与一个主窗口和一个子窗口关联。主窗口用来显示标题、边框以及其他的东西。子窗口包含所有的表单域并且根据它们的位置显示它们。这使得创建样式灵活的表单变得很容易。

由于这些操作与菜单窗口的创建很相似。所以使用的函数大致相同，工作原理也大致相同。所以这里只给出一个例子，并不做过多的解释。

例 28. 一个表单窗口例子

```
#include <form.h>

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);
int main()
{
    FIELD *field[3];
    FORM  *my_form;
    WINDOW *my_form_win;
    int ch, rows, cols;

    /* 初始化 curses */
    initscr();
    start_color();
    cbreak();
```

```
    noecho();
    keypad(stdscr, TRUE);

    /* 初始化颜色对 */
    init_pair(1, COLOR_RED, COLOR_BLACK);

    /* 初始化域 */
    field[0] = new_field(1, 10, 6, 1, 0, 0);
    field[1] = new_field(1, 10, 8, 1, 0, 0);
    field[2] = NULL;

    /* 设置表单域选项*/
    set_field_back(field[0], A_UNDERLINE);
    field_opts_off(field[0], O_AUTOSKIP);
    /* 如果当前表单域已满，不跳自动跳到下一个表单域 */

    set_field_back(field[1], A_UNDERLINE);
    field_opts_off(field[1], O_AUTOSKIP);

    /* 创建表单并传递 */
    my_form = new_form(field);

    /* 计算表单所需要的面积大小 */
    scale_form(my_form, &rows, &cols);

    /* 创建与表单相关联的窗口 */
        my_form_win = newwin(rows + 4, cols + 4, 4, 4);
        keypad(my_form_win, TRUE);

    /* 设置主窗口和子窗口 */
        set_form_win(my_form, my_form_win);
        set_form_sub(my_form, derwin(my_form_win, rows, cols, 2, 2));

    /* 在窗口的四周显示边框和标题 */
       box(my_form_win, 0, 0);
print_in_middle(my_form_win, 1, 0, cols + 4, "My Form", COLOR_PAIR(1));
    post_form(my_form);
    wrefresh(my_form_win);
    mvprintw(LINES - 2, 0, "Use UP, DOWN arrow keys to switch between fields");
    refresh();

    /* 循环以获取用户请求 */
    while((ch = wgetch(my_form_win)) != KEY_F(1))
    {   switch(ch)
```

```
        {   case KEY_DOWN:
                /* 跳到下一个域 */
                form_driver(my_form, REQ_NEXT_FIELD);
                /* 跳到当前缓冲区的末尾 */
                /* 在最后一个字符处跳出 */
                form_driver(my_form, REQ_END_LINE);
                break;
            case KEY_UP:
                /* 跳到上一个域 */
                form_driver(my_form, REQ_PREV_FIELD);
                form_driver(my_form, REQ_END_LINE);
                break;
            default:
                /* 如果是一个普通的字符，则打印出来 */
                form_driver(my_form, ch);
                break;
        }
    }
    /* 取消表单递送并释放内存 */
    unpost_form(my_form);
    free_form(my_form);
    free_field(field[0]);
    free_field(field[1]);
    endwin();
    return 0;
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{   int length, x, y;
    float temp;
    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;
    length = strlen(string);
    temp = (width - length)/ 2;
    x = startx + (int)temp;
    wattron(win, color);
```

```
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
}
```

18.5、表单域输入的有效性检查

默认情况下，表单域接收任何用户输入的数据。但也可以为表单域附加输入有效性检查。这样的话，如果域里面有和检查项不相符的数据，用户就跳不出这个域了。有些检查类型也支持即时检查——即在用户输入字符的同时进行检查。

可以用下面的函数给表单域附加输入有效性检查：

```
int set_field_type(FIELD *field,          /* 要附加输入检查的表单域 */
                   FIELDTYPE *ftype,      /* 要关联的检查类型 */
                   ...);                  /* 其他参数 */
```

一旦为表单域附加检查类型，表单域的有效性检查类型可以这样查询：

```
FIELDTYPE *field_type(FIELD *field);      /* 要查询的表单域 */
```

Form Driver 只检查由用户从终端输入数据的有效性。在下面的情况下，不进行有效性检查：

- 应用程序通过调用 set_field_buffer 改变表单域的数据；
- 已连接的表单域间接的被修改了——被其他连接到这个域的表单域修改。

下面是已定义的检查类型。你也可以自定义检查类型，虽然可以让程序更敏感、更智能，但也很麻烦。

TYPE_ALPHA（字符输入检查模式）

这个检查模式只接受字母，不接受空格，数字，特殊字符。这个检查模式让表单域在输入时就对输入的字符进行检查。这个检查模式可以这样设置：

```
int set_field_type(FIELD *field,          /* 要附加输入检查的表单域 */
                   TYPE_ ALPHA            /* 要附加的检查模式 */
                   int width);            /* 表单域的最大宽度 */
```

参数 **width** 指定检查数据的最小宽度。用户必须输入至少一个字符，才可以离开这个表单域。通常你会把表单域的宽度设置为这个宽度；但是如果这个最小宽度大于表单域的宽度，那么有效性检查的结果总会是“无效”。把最小宽度设置为 0 可以使有效性检查在任何时候结束。

TYPE_ALNUM（字符和数字输入检查模式）

这个检查模式同时接受字母和数字，不接受空格，特殊字符这个检查模式让表单域在输入时就对输入的字符进行检查。这个检查模式可以这样设置：

```
int set_field_type(FIELD *field,          /* 要附加输入检查的表单域 */
                   TYPE_ AKLUNM           /* 要附加的检查模式 */
                   int width);            /* 表单域的最大宽度 */
```

参数 **width** 设置所检查数据的最小宽度。同 TYPE_ALPHA 方式一样，通常你会把表单域的宽度设置为这个宽度；但是如果这个最小宽度大于表单域的宽度，那么有效性检查的结果总会是“无效”。把最小宽度设置为 0 可以使有效性检查在任何时候结束。

TYPE_ENUM（限定匹配项检查模式）
这种检查模式检查输入到表单域的数据是否在一个给定限定的匹配项集合内，这个集合可以是一系列的字符串（例如：美国各州的仅有两个字母的邮政代码）。这个检查方式可以这样设置：

```
int set_field_type(FIELD *field,          /* 要附加输入检查的表单域 */
                   TYPE_ENUM,             /* 要附加的检查模式 */
                   char **valuelist;      /* 限定值的列表 */
                   int checkcase;         /* 是否大小写敏感 */
                   int checkunique);      /* 是否检查匹配项唯一性 */
```

参数 valuelist 必须指向一系列以 **NULL** 字符结束的有效的字符串。如果参数 **checkcase** 为真，输入到表单域的字符串将区分大小写。

当用户退出一个附加 **TYPE_ENUM** 检查的表单域时，程序将试着自动将缓冲区内的数据补充完整数据之有效。当然，输入完整的供选的字符串是一般的做法。但是也可以输入一个供选字符串的前几个字符，表单会将其自动补充完整为有效数据。

默认情况下，如果你输入字符串的前几个字符，并且这几个字符在供选集合中有很多个匹配项，这前几个字符将会被匹配为第一个匹配字符串。不过，如果参数 **checkunique** 为真的话，要让输入有效，就要求匹配项在集合内是唯一的。

REQ_NEXT_CHOICE 和 **REQ_PREV_CHOICE** 请求对这些表单域会有特殊用处。

TYPE_INTEGER
这种检查方式检查表单域的值是否是一个整数。这个检查方式可以这样设置：

```
int set_field_type(FIELD *field,          /* 要附加输入检查的表单域 */
                   TYPE_INTEGER,          /* 要附加的检查模式 */
                   int padding,           /* 填充为 0 */
                   int vmin, int vmax);   /* 有效值的范围 */
```

在这种情况下的有效字符为整数，也可以是带“－”号的负整数。当用户离开这个表单域之后才检查输入值是否在规定范围之内。如果限制范围的最大值小于或者等于最小值，整数值的限制范围将被忽略。如果输入的整数通过了范围检查，函数将会自动地在这个值**前面**填充尽可能多的 0，让其变为填充变量。

通过 C 库函数 **atoi（3）**可以很方便的解释 **TYPE_INTEGER** 类型值的缓冲区。

TYPE_NUMERIC
这种检查方式检查表单域的值是否是一个十进制小数。这个检查方式可以这样设置：

```
int set_field_type(FIELD *field,          /* 要附加输入检查的表单域 */
```

```
                TYPE_NUMERIC,          /* 要附加的检查模式 */
                int padding,           /* 要设置的精度 */
                int vmin, int vmax);   /* 有效值的范围 */
```

在这种情况下的有效字符为实数也可以是带“－”号的负数，也可以带小数点。同样的，当退出这个表单域的时候才检查输入值是否在规定范围之内。如果限制范围的最大值小于或者等于最小值，整数值的限制范围将被忽略。如果输入的整数通过了范围检查，函数将会自动地在这个值**后面**填充尽可能多的 0，让其变为填充变量。

通过 C 库函数 **atoi（3）**可以很方便的解释 **TYPE_NUMERIC** 类型值的缓冲区。

TYPE_REGEXP

这种检查方式检查表单域的值是否是一个与常规表达式相匹配的数据。这个检查方式可以这样设置：

```
int set_field_type(FIELD *field,          /* 要设置检查方式的域 */
                   TYPE_REGEXP,           /* 要关联的类型 */
                   char *regexp);         /* 要匹配的表达式 */
```

常规表达式的语法即 **regcomp(3)**。常规表达式是否匹配的检查在退出表单域时执行。

18.6、Form Driver：表单系统的灵魂

和菜单系统一样，**form_driver()**在表单系统中扮演同样重要的角色。各种类型的请求都是经过 **form_driver()**处理的：

```
int form_driver(FORM *form,       /* 要进行操作的表单 */
                int request)      /* 表单请求项 */
```

正像你在前面的有些例子中看到的，必须在一个循环中取得用户的输入，然后再确定这是一个数据还是一个表单处理请求。若是表单处理请求，则将信号传递给 **form_driver()**，由它完成相应的工作。这些请求项可以大致地分为下面几类。下面将解释这些不同的请求和它们的用法：

18.6.1、页面导航请求

这些请求用来在表单里面实现翻页，并控制页面的显示。一个表单可以由多个页面组成。如果你的一个表单中含有很多表单域和逻辑分段，你就可以把这个表单设置为多页的。函数 **set_new_page()**可以为指定的表单域增添一个新页面。

```
int set_new_page(FIELD *field,          /* 要设置新页或取消设置的表单域 */
          bool new_page_flag);          /* 若要设置断点，则应该为 TRUE */
```

下面的导航请求允许你在不同的表单页面中跳转：

- **REQ_NEXT_PAGE**　跳到表单下一页
- **REQ_PREV_PAGE**　跳到表单上一页
- **REQ_FIRST_PAGE**　跳到表单第一页
- **REQ_LAST_PAGE**　跳到表单最后一页

这些请求把页面列表视作一个循环。也就是说，在最后一页发送 **REQ_NEXT_PAGE** 请求使得表单页从最后一页跳到第一页，**REQ_PREV_PAGE** 使得表单页从第一页跳到最后一页。

18.6.2、表单域间移动请求

下面的这些请求处理光标在不同表单域间移动，这些表单域必须是同一个页面内的。

- **REQ_NEXT_FIELD** 跳到下一个表单域
- **REQ_PREV_FIELD** 跳到上一个表单域
- **REQ_FIRST_FIELD** 跳到第一个表单域
- **REQ_LAST_FIELD** 跳到最后一个表单域

- **REQ_SNEXT_FIELD** 跳到有序表单域的下一个表单域
- **REQ_SPREV_FIELD** 跳到有序表单域的上一个表单域
- **REQ_SFIRST_FIELD** 跳到有序表单域的第一个表单域
- **REQ_SLAST_FIELD** 跳到有序表单域的最后一个表单域

- **REQ_LEFT_FIELD** 向左跳到上一个表单域
- **REQ_RIGHT_FIELD** 向右跳到下一个表单域
- **REQ_UP_FIELD** 向上跳到上一个表单域
- **REQ_DOWN_FIELD** 向下跳到下一个表单域

这些请求把同一页内的表单域视为一个循环。也就是说，**REQ_NEXT_FIELD** 可以使光标从最后一个表单域跳到第一个表单域，**REQ_PREV_FIELD** 可以使光标从第一个域跳到最后一个域。对于这些请求(包括 **REQ_FIRST_FIELD** 和 **REQ_LAST_FIELD**)来说，表单域的顺序就是表单数组中其指针的顺序（用 **new_form()**或 **set_form_fields()**设置）。

同样也可以按屏幕位置的来依次遍历这些表单域，顺序是自左至右，由上到下。可以通过使用第二组移动请求（即 **REQ_S** 开头的请求）完成。

最后，也可以使用方向键在表单域间移动。要完成这项功能，可以使用上面四组请求中的第三组。注意：为完成这个功能，这些请求引用的表单位置是指表单的左上角。

例如，假设你有一个多行表单域 B 和两个与 B 在同一行的单行表单域 A 和 C。而且 A 在 B 的左边，C 在 B 的右边。**REQ_MOVE_RIGHT** 请求只在 A,B,C 的第一行在同一行的情况下才能使光标从 A 移动到 B；否则将跳过 B 而移动到 C。

18.6.3、域内导航请求

下面的这些请求使光标在当前表单域内移动.

- **REQ_NEXT_CHAR** 跳到下一个字符处
- **REQ_PREV_CHAR** 跳到前一个字符处
- **REQ_NEXT_LINE** 跳到下一行
- **REQ_PREV_LINE** 跳到前一行
- **REQ_BEG_FIELD** 跳到表单域的开始
- **REQ_END_FIELD** 跳到表单域的结尾

- **REQ_BEG_LINE** 跳到行首
- **REQ_END_LINE** 跳到行尾
- **REQ_LEFT_CHAR** 跳到表单域的左边
- **REQ_RIGHT_CHAR** 跳到表单域的右边
- **REQ_UP_CHAR** 跳到表单域的上边
- **REQ_DOWN_CHAR** 跳到表单域的下边

前后单词之间都用空格来分开。如果要让光标移动到起始处或末尾处（在表单域内或行内），相应的命令会在当前范围内查找第一个字符或最后一个字符。

18.6.4、滚动请求

增大后的动态表单域和指定显示行数的表单域都是可滚动的。单行表单域以水平方式滚动；多行表单域以垂直方式滚动。一般滚动都是因为光标移动（为使光标可见而滚动）引起的。你也可以使用下面的请求来显式地请求滚动：

- **REQ_SCR_FLINE** 纵向向前滚动一行
- **REQ_SCR_BLINE** 纵向向后滚动一行
- **REQ_SCR_FPAGE** 纵向向前滚动一页
- **REQ_SCR_BPAGE** 纵向向后滚动一页
- **REQ_SCR_FHPAGE** 纵向向前滚动半页
- **REQ_SCR_BHPAGE** 纵向向后滚动半页
- **REQ_SCR_FCHAR** 纵向向前滚动一个字符
- **REQ_SCR_BCHAR** 纵向向后滚动一个字符
- **REQ_SCR_HFLINE** 纵向向前滚动一个域宽
- **REQ_SCR_HBLINE** 纵向向后滚动一个域宽
- **REQ_SCR_HFHALF** 纵向向前滚动半个域宽
- **REQ_SCR_HBHALF** 纵向向后滚动半个域宽

在执行滚动时，表单域的一页就是它的可视部分的高度。

18.6.5、编辑请求

当你传递一个 ASCII 字符给 Form Driver 时，它被视做一个添加到表单域的数据缓冲区中的字符。是插入模式还是替换模式由表单域的编辑模式决定（默认模式是插入模式）。
下面的请求支持表单域编辑并且可以改变编辑模式：

- **REQ_INS_MODE** 设置为插入模式
- **REQ_OVL_MODE** 设置为替换模式
- **REQ_NEW_LINE** 新行请求（看下面的解释）
- **REQ_INS_CHAR** 在当前字符处插入一个空格
- **REQ_INS_LINE** 在当前字符处插入一个空行
- **REQ_DEL_CHAR** 删除光标所在处的字符
- **REQ_DEL_PREV** 删除光标所在处的前一个字
- **REQ_DEL_LINE** 删除光标所在行
- **REQ_DEL_WORD** 删除光标所在的字
- **REQ_CLR_EOL** 清除当前位置到行尾的字符

- **REQ_CLR_EOF** 清除当前位置到域尾的字符
- **REQ_CLEAR_FIELD** 清空整个域

REQ_NEW_LINE 请求和 **REQ_DEL_PREV** 请求比较复杂，而且部分被其他表单选项控制。尤其是在光标处于表单域的开头或最后一行发送这类请求时，这里需要特别的说明：

首先看一下 REQ_NEW_LINE：
通常 **REQ_NEW_LINE** 的作用是在插入模式时中断光标当前编辑位置的所在行。把当前光标后面的部分作为一个新行插入，并且光标移到新行的开头（你可以认为是在表单域缓冲区插入一行）。

通常 **REQ_NEW_LINE** 在替换模式下会将当前光标位置到行尾的所有内容清除。并且将光标移动到新行的开头。

但是，在表单域的开头或最后一行发送 **REQ_NEW_LINE** 请求时，它相当于 **REQ_NEXT_FIELD** 的功能。若选项 **O_NL_OVERLOAD** 关闭，这个特殊行为就会禁止。

接下来看一下 REQ_DEL_PREV：
通常 **REQ_DEL_PREV** 是删除前一个字符。如果是在插入模式下并且光标在行首，当前行的内容将会接到上一行上去，这时它完成的功能是把当前行的内容接到上一行，并删除当前行（你可以认为是在表单域缓冲区中删除一新行）。

但是如果是在表单域的开头，**REQ_DEL_PREV** 则被当成 **REQ_PREV_FIELD** 处理。

如果选项 **O_BS_OVERLOAD** 关闭，这个特殊行为就会禁止，Form Driver 就会返回 **E_REQUEST_DENIED**。

18.6.6、顺序请求

如果你的表单域类型是有序的，并且已经关联了取得前一个值和后一个值的函数。以下两个请求可以将前一个值或后一个值放入缓冲区：

- **REQ_NEXT_CHOICE** 把当前值的后一个值放在缓冲区
- **REQ_PREV_CHOICE** 把当前值的前一个值放在缓冲区

对于内置域类型，只有 TYPE_ENUM 有内置的函数用来读取前面一个值或后面一个值。当定义了一个你自己的表单域类型时（参看自定义有效检查类型），你可以关联你自己的函数。

18.6.7、应用程序命令

表单请求用大于 KEY_MAX 且小于等于常数 MAX_COMMAND 的整数表示。在这个范围内的值将被 **form_driver()**忽略。这使得其他应用程序任意使用它们。它可以被看成是应用程序中特定的行为代码，并通过该代码采取相应的行为。

第十九章：开发工具库和扩展组件库

通过以上章节，你现在知道 CURSES 姊妹库的威力了吧？你现在一定卷起袖子准备开发一个规模宏大的项目了。不过先打住…如果只是直接使用 NCURSES 库或者其姊妹库，编写这些代码并维护其中的窗口组件仍然是件相当痛苦的事。以下会介绍一些组件工具库和扩展组件库。这些库中包含了一些可以直接使用的工具和常用组件，可以用来替代你自己编写的组件，你可以直接使用它们，或从这些组件的代码中汲取编程灵感，甚至还可以自己来扩展这些组件的功能。

19.1. CDK (Curses Development Kit)

在这里先引用作者的话：

*CDK 是 CURSES 开发包（*Curses Development Kit*）的缩写，目前包含 21 个可以用来迅速开发 CURESES 程序的常用组件。*

这个工具包提供了一些可以直接应用到程序中的常用组件。这些组件的源代码都有良好的编码风格和非常详细的文档。示例目录中的例子会给 CURSES 的初学者很好的开始。CDK 可以从官方网站 http://www.vexus.ca/products/CDK/cdk.tar.gz 下载（译者注：也可以从 Poet 项目的站点 http://poet.cosoft.org.cn/downloads/develop/cdk.tar.gz 下载）。你可以根据 tar 包中的 README 文件提示安装它。

19.1.1. 组件列表

下面这个列表就是 CDK 中的组件列表，旁边有这个组件的说明：

组件	简要说明
Alphalist	允许用户从一个单词列表中选择一个单词，并且能够通过键入单词的前几个字符来缩短搜索列表。
Buttonbox	创建按钮。
Calendar	创建一个简单日历。
Dialog	创建一个让用户通过按钮响应的对话框。
Entry	允许用户可以输入各种类型的数据。
File Selector	一个用 CDK 基本组件建立组成的文件选择器。这个例子展示了怎样用 CDK 基本组件创建一个复杂的对话框。
Graph	画一个图标
Histogram	画一个柱状图
Item List	创建一个可以供用户选择的弹出式区域。对于选择日期来说是个非常有用的组件。
Label	显示一个弹出式信息框，它的标签可以被看作窗口的一部分。
Marquee	显示一个可以滚动的选取框。
Matrix	创建一个由很多选项组合的复杂矩阵。
Menu	创建一个下拉式菜单。
Multiple Line Entry	一个多行的输入区域，对于输入多行的数据（例如描述文本）很有

	用。
Radio List	创建一个单选按钮列表。
Scale	产生一个数值范围让用户在限定的范围内(用方向键等)从中选定一个数值。
Scrolling List	创建一个滚动列表/菜单。
Scrolling Window	创建一个可滚动的日志文件查看器。你可以在它运行时向窗口中添加数据。这是一个很不错的用来显示某些进度的组件。（类似一个控制台窗口）
Selection List	创建一个多选项菜单。
Slider	像 Scale 组件一样，这个组件提供一个可视的滑块用来显示数值。
Template	创建一个预先设置输入文字的敏感区域，常被用作输入日期、电话号码等预先设置好格式的输入区域。
Viewer	这是一个文件/信息查看器。在你要从某个文件读取信息时将会非常有用。

19.1.2. 一些吸引人的功能

CDK 除了让基于 CURSES 的程序便于开发和维护。CDK 还解决了很多复杂的 CURSES 处理问题。例如打印混合颜色的字符串，使字符串看起来更雅致。你也可以向 CDK 的某些函数传递带修饰标记字符串：

例如：
"</B/1>This line should have a yellow foreground and a blue background.<!1>"

当你将这些参数传给 newCDKLabel()函数，它将打印一行蓝色背景的黄色字符串。当然，还有很多修饰标记，让字符看起来更加美观和精致。你可以通过查阅 man 文档中的 cdk_display(3X)页找到更多详细的资料。这页给出了很多非常不错的例子来说明它的用法。

19.1.3. 总结

总而言之，CDK 包含了很多编码风格良好的扩展组件。适当的使用这些组件可以很方便的构建一个强大、稳定的工作框架。尤其是编写一个比较复杂的图形用户界面。

19.2. 对话组件

很久以前，也就是 1994 年的 9 月，那时 Linux 还鲜有人知，Jeff Tranter 就在 Linux Journal 发表了一篇关于人机对话的文章。这篇文章的开头是这样写的：

Linux 是基于 Unix 的操作系统，但是它的内核和应用程序中很多独特且实用的功能却超过了 Unix 下某些常用的东西。其中一个鲜为人知的亮点就是“dialog”：一个可以不用编写 shell 脚本就可以创建专业效果的对话框的部件。这篇文章仅仅作为一个指南，介绍 dialog 并通过例子告诉你何时何地可以用到它……

就像他说明的，dialog 是一个真正的亮点。它可以简单的创造出专业效果的对话框。它可以创建多种多样的对话框，比如：带菜单的、带选项列表的等等。它应该是缺省安装的，如果你的机器上没有，你可以点击 ibiblio linux archive.下载

上面的文章给出了 dialog 的作用和能力的概览。man 文档还有更多关于 dialog 详细的资料。它可以应用在更多的情况下。Linux 内核的文本模式就是一个很好应用 dialog 的例子。Linux 的内核用不同版本的 dialog 定制自己所需要的东西。

dialog 最初是为 shell 脚本代码的使用而设计的。如果你想将它应用到你的 C 程序中，你可以使用 lobdialog。但是关于 libdialog 的文档非常稀少。最权威的参考是这个库的 dialog.h 头文件。你也许需要去改动它已得到自己想要的输出。非常幸运，它的源代码很容易定制。我已经多次成功地通过自己修改这些代码应用于不同的应用场合。

19.3. Perl 中的 CURSES 模块

Perl 中的 CURSES 模块：Curses::Form 和 Curses::Widgets 用来从 Perl 中访问 Curses。如果你已经安装了 Curses 和 Perl，你就可以从 CPAN 模块页的“Curses”的类别中下载这三个压缩的模块。当你安装这些模块后，就可以像在 Perl 中用其他模块那样使用。如果你需要更多详细资料可以参看 perlmod man 文档。以上的模块都包含很不错的说明文档，而且包含了一些演示脚本来测试这些功能。虽然 widgets 模块还很原始，但这些模块提供从 Perl 中访问 curses 很好的途径。

如果需要更多的资料可以参看 man 页 Curses(3)，Curses::Form(3)和 Curses::Widgets(3)这些页只有安装以上的模块才可以被安装。

第二十章：快乐至上

这一章将介绍这部文档中的示例程序中 JustForFun 目录下的程序。这些程序是我凭兴趣写的，但是并不是说它们是很好的编程范例，也不是使用 ncurses 的最佳方法或诀窍。它们为初学者提供了一些点子，同时允许您为这部文档添加更多的示例程序。如果你无意中用 curses 编写了一个又简单又有趣的程序，而且希望他们加入这个章节，可以联系作者。（译者注：用英文联系原文作者可以将您的程序添加到原始的英文版文档，如果您想将您的程序仅包含到中文版文档，可以联系译者）

20.1. 生命游戏(The Game of Life)

生命游戏是一个神奇的数学游戏，用 Paul Callahan 的话说就是：

> *生命游戏(或者是简单生命)不是一个通常定义的游戏。在这个游戏中，没有玩家、没有输也没有赢，当这些“碎片”被放置到起点位置，规则将确定后面发生的任何事。然而，人生总是充满了惊奇！在很多情况下，看见开始位置（或者开始的样子）和看到未来发生的事是不可能的。遵循游戏的规则是发现起点和看到未来的唯一方法。*

这个程序开始以一个反转的类似U样子开始，它会告诉你生命是多么精彩。你有很多的空间可以用来改进这个程序。你可以让用户通过输入选择他的生命模式，甚至可以从文件中读取这些样式。你可以改变规则并在大量的变化得到乐趣。在google上可以查到很多和生命游戏（game of life）的信息。

文件路径： JustForFun/life.c

20.2. 幻方(Magic Square)

幻方，另一个神奇的数学游戏。它很容易了解，但很难满足它的条件。在幻方中要让每行的和、每列的和、以及每个斜行的和必须相等。
这个程序运行时需要附加一个奇数作为参数。

文件路径： JustForFun/magic.c

20.3. 汉诺塔(Towers of Hanoi)

著名的汉诺塔模拟程序。目的是将第一个柱子上的所有盘子按从上到下、从小到大顺序移动到最后一个柱子上，中间有一个柱子用来临时放置盘子。无论怎么放，放盘子的时候不能让较大的盘子压在较小的盘子上。

20.4. “八皇后”问题（Queens Puzzle）

这是最著名的八皇后问题。这个问题要求在国际象棋的棋盘上摆放 8 个皇后，而且这八个皇后不会相互攻击。这个程序用一个非常简单的返回算法来解决这个问题。

文件路径：JustForFun/queens.c

20.5. 纸牌（Shuffle）

一个有趣的游戏，可以用来消磨时间。

文件路径：JustForFun/shuffle.c

20.6 打字练习（Typing Tutor）

一个简单的打字训练程序，我创建了更多其它的东西以便于使用。如果对键盘不熟悉，但是你知道使用键盘的正确姿势。这个程序会很有用的。

文件路径：JustForFun/tt.c