NCURSES Programming HOWTO

中文版

第二版

新疆大学软件学院

顾宇 高江 侯瑞杰 石英 袁丽

The Messiah Team

NCURSES Programming HOWTO 中文版 翻译说明(第二版)：

原题：NCURSES Programming HOWTO V1.9
译题：NCURSES Programming HOWTO 中文版 第二版
原作：Pradeep Padala
来源：ibiblio（http://www.ibiblio.org/）
时间：2006 年 3 月 22 日
成员：顾宇（拜伦）、高江、侯瑞杰、石英、袁丽

修订说明：

在第一版发表以后，我收到了不少询问关于 NCURSES 中文的问题，眼下 NCURSES 对中文的支持不够好。同时我在进行 Poet 项目的过程中发现第一版的文档在翻译部分需要很多的改进。计划是利用这个寒假重新修订一下第一版的文档。但在 2005 年的 11 月份，我看到了 NCURSES Programming HOWTO 的最新版本，其中修改和补充了部分内容。因此借这次修订，把最新版本的 NCURSES Programming HOWTO 的更新内容也合并过来。感谢过去一年中大家对这份中文文档的肯定和支持。尤其是那些发送 E-mail 来给予我鼓励的朋友们。如果这部文档能使您受益，那将是我们最欣慰的事。

以下是更新的联系方式，同时第一版中的联系方式作废。
如果有任何的疑问，意见或建议，欢迎联系译者，我将会尽帮助您。

作者的联系方式：

作者的个人主页：http://www.eecs.umich.edu/~ppadala/
作者的电子邮件：ppadala@eecs.umich.edu.nospam

译者的联系方式：

译者的 Blog：http://blog.csdn.net/byronm
译者的项目站点：http://poet.cosoft.org.cn/
译者的电子邮件：byronm@sina.com

修订成员：

顾宇（新疆大学软件学院 软件 05 级 5 班）byronm@sina.com
高江（新疆大学软件学院 软件 05 级 7 班）star_gj@sina.com
侯瑞杰（新疆大学软件学院 软件 05 级 7 班）hourjworld@sina.com
石英（新疆大学软件学院 软件 05 级 7 班）quartz2006@sina.com
袁丽（新疆大学软件学院 软件 05 级 5 班）sail_land@sina.com

感谢新疆大学软件学院所有老师和同学对我这项工作的给予的支持和鼓励。
感谢对本文档第一版提出意见和批评的网友，是他们给了我修订第二版的动力。
感谢 Poet 小组的每个成员，他们给我了前进的信心和勇气。以及很多翻译上的帮助。
感谢猫扑(mop.com)中“开放源代码——中国”小圈子里的每个成员的支持。
感谢共创软件联盟(cosoft.org.cn)给我的项目提供了网上空间的支持。
感谢新老读者对本文的肯定和支持。有了你们的支持，才有这部文档的完善。
感谢我们所有的朋友、亲人。
本人水平有限，加上时间仓促，修订中难免会出现差错。如果有任何疑问或者意见请发 E-mail 告诉我。再次感谢您的支持和鼓励，我们会将这部文档做得更好。

NCURSES Programming HOWTO 中文版 翻译说明(第一版)：

原题：NCURSES Programming HOWTO
译题：NCURSES Programming HOWTO 中文版
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 用户群的壮大以及开放源代码运动的渐渐升温，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，方便了字符环境下应用程序界面的开发。
这是一份关于如何使用 NCURSES 库编写 Linux 控制台程序的 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。我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。

项目成员：（按加入项目时间先后）
Byron（byronm@sina.com）《NCURSES Programming HOWTO 中文版》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，那将是我们最欣慰的事。

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。

NCURSES Programming HOWTO

Pradeep Padala(p_padala@gmail.com)
V1.7.1 2005 年 6 月 20 日

各版本修订历史：

修订版本：1.9　　修订日期：2005 年 6 月 20 日　　修订人：P·Padala
NCURSES 的许可协议已经更换为 MIT-style 许可协议。示例程序也都遵循该许可协议。

修订版本：1.8　　修订日期：2005 年 6 月 17 日　　修订人：P·Padala
更新了很多地方，增加了参考和 Perl 语言的使用示例。修改了一些例子。对文字内容进行了大量语法上和风格上的修改。更新了 NCURSES 的历史。

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

修订版本：1.7　　修订日期：2002 年 6 月 25 日　　修订人：P·Padala
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。

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

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

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

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

修订版本：1.3　　修订日期：2001 年 7 月 24 日　　修订人：P·Padala
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

修订版本：1.2　　修订日期：2001 年 6 月 5 日　　修订人：P·Padala
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。

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

这部文档的写作目的是要成为一份 ncurses 以及子函数库的综合编程指南。我们将从一个简单的“Hello World”程序开始，循序渐进的学习掌握 ncurses 各方面更复杂、高级的内容。因此假定读者在此之前没有使用 ncurses 库的经验。如果有需要，请将注解发送至 p_padala@gmail.com

第一章 NCURSES 库简介

在使用电传终端作为计算机输出设备的那个年代，电传终端是和计算机分开放置的，并通过串行电缆连接。终端的配置通过发送一系列字节完成。所有的终端控制操作：改变光标在屏幕上的位置，清除屏幕某一区域的内容，屏幕滚动，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，也是通过这样一系列字节实现的。由于这些连续的控制字节以一个转义字符“0x1B”（即 ESC 键）作为起始字节，所以这种控制序列叫做转义序列。即使在当今，我们也可以向终端仿真程序发送转义序列，得到和当年电传打字终端相同的显示效果。

假设你想在终端（或终端仿真窗口）输出一段彩色的文字，可以将以下这段转义序列输入到你的字符控制台（或控制台窗口）：

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

在这里 “^[” 就是所谓的转义字符。（注意：“^[” 是一个字符，不是依次键入“^”和“[”字符。要输出“^[”，必须按下 Ctrl-V 再按下 ESC 键）执行以上的命令，就应该可以看见“In Color”变为红色了（译者注：试更改转义字符各分号间的参数，看看会有什么结果）。以后输出的文本信息都是这样的效果。 如果想返回原来的颜色设置可以使用以下的命令：

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

现在体会到这些神奇字符（转移序列）的作用了吗？然而这些奇怪的字符是不是很难理解呢？并且有时相同的转义序列在不同的终端上会有不同的显示结果。因此，UNIX 的设计者发明了一种叫做 termcap 的机制。

termcap 是一个列出特定终端的所有功能的文本文件，并且给出了实现对应功能的转义序列。然而在这种机制发明后的几年中，逐渐被 terminfo 机制取代。程序员使用 terminfo 时不用花过多的时间查阅 termcap。只需让程序查询 terminfo 的数据库就可得到控制字符，并将其发送到终端或终端仿真程序。

1.1 NCURSES 是什么？

你可能会疑惑，引入的这些技术术语是什么。假设在使用 termcap 或者 terminfo 的情况下，每个应用程序都在访问 terminfo 数据库并且做一些必要的处理（比如发送控制字符等等）。不久这些操作的复杂度将会变得难以控制。于是，curses 诞生了。curses 的命名是来自一个叫做 “cursor optimization”（光标最优化）的双关语（译者注：curses 本身有诅咒的意思）。curses 构成了一个工作在底层终端代码之上的封装，并向用户提供了一个灵活高效的 API（Application Programming Interface 应用程序接口）。它提供了移动光标，建立窗口，产生颜色，处理鼠标操作等功能。使程序员编写应用程序不需要关心那些底层的终端操作。

那么 ncurses 又是什么？ncurses 是最早的 System V Release 4.0 (SVr4)中 CURSES 的一个克隆。这是一个可自由配置的库，完全兼容旧版本的 curses。简而言之，它是一个管理应用程序在字符终端显示的函数库。当后面提到 curses 的时候，同时也可以和 NCURSES 互换。

关于 ncurses 详细的更新历史可以查阅 ncurses 源代码分发包中的 NEWS 文件。Thomas Dickey 是目前的维护人员。你可以通过 bug-ncurses@gnu.org联系维护人员

1.2 我们可以用 NCURSES 做什么？

ncurses 不仅仅封装了底层终端功能，而且提供了一个相当稳固的工作框架（Framework）可以在字符模式下产生美观的界面。它提供了一些创建窗口的函数。而它的姊妹库 Menu、Panel 和 Form 则对 curses 基础库及进行了扩展。这些扩展库通常都随同 curses 一起发行。我们可以建立一个同时包含多个窗口（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，例如让它滚动或者隐藏。

菜单（Menus）可以让用户建立命令选项，方便用户执行命令。而表单（Forms）允许用户建立一些简单的数据输入和输出的窗口。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 的简单介绍。在以后的章节里，我们将详细的介绍这些库。

1.3 在哪能得到它

现在你知道你可以用 ncurses 做什么了吧。使用 ncurses 之前你必须先安装它，通常在安装操作系统（Unix/Linux）时它已被安装。如果你的操作系统里没有 ncurses，你可以通过以下的途径安装：

编译 ncurses 包：
ncurses 可以从 ftp://ftp.gnu.org/pub/gnu/ncurses/ncurses.tar.gz获得。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 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）

tar 文件包中的 README 和 INSTALL 文件是安装 ncurses 库的主要资料。通常是按以下方法安装 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                                  # 获得 root 权限
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：

```
rpm –i <下载的 RPM 文件>
```

1.4 本文档的写作意图和涵盖范围

这部文档的写作目的是要成为一份 ncurses 以及子函数库的综合编程指南。我们将从一个简单的“Hello World”程序开始，循序渐进的学习掌握 ncurses 各方面更复杂、高级的内容。因此假定读者在此之前没有使用 ncurses 库的经验。

1.5 关于文档中出现的程序

这份文档中用到的所有程序已经被打包并压缩成一个tar.gz 文件。可以在 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                  -- 生命游戏
    |----> magic.c                 -- 数字幻方
    |----> queens.c                -- 皇后问题
    |----> shuffle.c               -- 智力拼图
    |----> 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

perl
|---->01-10.pl                     --前 10 个例子的 perl 语言版
```

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和 NCURSES 相同的许可协议（MIT-Style）， 这个协议使你可以做除了得到他们以外更多的事情。你可以更自由的使用它们。

1.6 关于此文档的其他文件发布格式

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 可以从从 tldp.org 得到的格式：

Acrobat PDF 格式
PostScript 格式
多页的 HTML 文档
单页的 HTML 文档

1.6.2 从源代码编译

如果以上链接出现问题，或者你想自己使用 sgml 进行实验，请遵循以下步骤：
1、从以下两个地址下载源代码包和 tar.gz 的示例程序压缩包：
http://cvsview.tldp.org/index.cgi/ldp/howto/docbook/ncurses-howto/NCURSES-Programming-HOWTO.sgml
http://cvsview.tldp.org/index.cgi/LDP/howto/docbook/NCURSESHOWTO/ncurses_programs.tar.gz
2、解压缩示例程序包：
tar zxvf ncurses_programs.tar.gz

3、用 jade 程序建立各种各式的文档，如果你想创建多页的 HTML 文档：
jade -t sgml -i html -d 风格样式表文件 NCURSES-Programming-HOWTO.sgml

如果需要 PDF 格式，首先要建立单页的 HTML 文档：
jade -t sgml -i html -d 风格样式表文件 -V nochunks NCURSES-Programming-HOWTO.sgml > NCURSES-ONE-BIG-FILE.html

然后利用这份文档和 htmldoc 工具创建 PDF 文档：
htmldoc --size universal -t pdf --firstpage p1 -f 输出的.pdf 文件全名 NCURSES-ONE-BIG-FILE.html

也可以用来生成 PS 文档：
htmldoc --size universal -t ps --firstpage p1 -f 输出的.ps 文件全名 NCURSES-ONE-BIG-FILE.html

详细信息请访问 LDP Author Guide (LDP 作者指南)可以获得更多信息，如果都失败了，请和作者 ppadala@gmail.com 联系，也可以联系译者 byronm@sina.com

1.7 原文贡献者

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。

Perl 语言版示例程序的提供者 Anuradha Ratnaweera（aratnaweera@virtusa.com）。

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

下面是正在进行或者将要进行的项目。如果你有新的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 版权声明

NCURSES Programming HOWTO 中文版（第二版）

版权声明

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，来了解一下应用 ncurses 库所编写程序的结构。

2.1 编译包含 NCURSES 库函数的程序

如果要调用 ncurses 库中的函数，你必须在代码中加载 ncurses.h 文件，就是要在 C 或 C++ 程序开头添加“#include <ncurses.h>”，然后在链接时标记-lncurses 参数。（注：ncurses 库已经包含“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       */
    getchar();                  /* 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，并等待下一部处理。所以在调用其它的 curses 函数前，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“stdscr”）和相关的数据结构分配内存。
在以前的计算机上曾经出现过一个极端的例子：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。在以后的章节中我们将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。这个函数在用法上和 printf()函数很像。不同的是：printw()函数把字符串输出到“stdscr”的虚拟窗口坐标（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，实际上这个消息打印到了一个叫作“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。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。

通过这种机制程序员能够不断在虚拟屏幕上写数据。然后调用 refresh（）函数让输出的操作看起来是一次完成的。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，很多人都会犯这样的错误。

2.2.3 关于 endwin()函数

最后，别忘了退出 curses 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占用的内存，使你能够正常返回控制台模式。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()函数后再调用其它的 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

现在我们知道了如何编写一个简单的 curses 程序，也知道了一个 curses 程序都由哪些部分组成。接下来我们就正式的进入 curses，了解每一部分的具体内容。curses 中有很多的函数能够帮你定制出你想要的显示效果。

那么，现在让我们开始吧……

第四章 初始化

我们现在知道在程序中调用 initscr()函数，屏幕就会初始化并进入 curses 模式。本章我们会介绍其它的初始化函数，这些函数可以根据我们自己的要求在初始化后定制 curses 会话（curses session）的功能及模式。例如：终端模式（terminal mode）、彩色显示模式（color mode）、鼠标操作模式（mouse mode）等。当然，我们还可以定制混合模式。这章就让我们来讨论这些在 initscr()函数之后调用的初始化函数。

4.1 raw()函数和 cbreak()函数

通常情况下，终端驱动程序会缓冲用户输入的字符，直到遇到换行符或回车符后，这些字符才可以被使用。但是大多数程序要求字符在输入时就可以被使用。 raw()和 cbreak()两个函数都可以禁止行缓冲（line buffering）。区别是：在 raw()函数模式下，处理挂起（CTRL-Z）、中断或退出（CTRL-C）等控制字符时，将直接传送给程序去处理而不产生终端信号；而在 cbreak()模式下，控制字符将被终端驱动程序解释成其它字符。就我个人而言，比较喜欢使用 raw()函数，因为用它可以做一些一般用户无法进行的控制操作。

4.2 echo()函数和 noecho()函数

这两个函数用来控制是否将从键盘输入的字符显示在终端上。调用 noecho()函数禁止输入的字符出现在屏幕上。也许程序员希望用户在进行控制操作时，需要屏蔽掉控制字符（如组合键操作），或调用 getch()函数读取键盘字符时，不想显示输入的字符（如在控制台输入登陆密码）。大多数的交互式应用程序在初始化时会调用 noecho()函数，用于在进行控制操作时不显示输入的控制字符。这两个函数给予程序员很大的灵活性，使程序员可以在窗口中的任意地方，实现输入字符的显示和屏蔽，而不需要刷新屏幕。

4.3 keypad()函数

这个函数允许使用功能键，例如：F1、F2、方向键等功能键。几乎所有的交互式程序都需要使用功能键，因为绝大多数用户界面主要用方向键进行操作。使用 keypad(stdscr,TURE)就为“标准屏幕”（stdscr）激活了功能键。你将会在以后的章节中学习到如何使用功能键。

4.4 halfdelay()函数

halfdelay()函数会启用半延时模式（half-delay mode）。和 cbreak()函数一样，当程序需要用户输入字符时，输入的每个字符都是可用的。给 halfdelay()传递一个整型参数（以 0.1 秒为单位），它就会在这个参数时间内等待输入。如果没有有效的输入，则返回 ERR。

一般来说，这个函数在需要等待输入的程序中会用到。如果用户没有在规定时间内给出有效输入，程序就可以去处理其它事情。最常见例子就是在输入密码时做出超时响应。

4.5 其它的初始化函数

上面提到的函数用来定制 curses 在初始化后的行为，因此这些函数只能在整个 curses 会话的开始部分（即初始化时）调用，而不能在程序的其它地方调用。

4.6 示例程序

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，
                                       我们必须按下 enter 键才可以将字符传递给程序*/
    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，但还是有一些在前面的章节没有介绍的函数。 getch()函数用来取得用户输入的字符，它等价于通常的 getchar()函数，只是我们在调用 getchar()函数时要禁止行缓冲以避免在输入完成后按enter 键。在后面的章节中我们将讨论到输入函数。attron()和 attroff()函数分别作为修饰的开头和结尾，修饰其间输出的字符。例如在本例中加粗（即使用 A_BOLD 作为函数参数）用户输入的字符。在后面的章节中我们也将会详细讨论这些函数。

第五章 窗口机制简介

在我们面对数以万计的 curses 函数之前，让我们先了解一下窗口机制。关于窗口机制的详细内容我们会在以后的章节中详细介绍，本章只介绍窗口的基本概念。

窗口是由 curses 系统定义的一个假想的屏幕，并不像 Windows 平台上的窗口那样具有边框。当 curses 初始化的时候，它会默认创建一个叫做 stdscr 的窗口。这个窗口的屏幕一般是 80 列，25 行（根据显示环境的不同，可能会出现不同的大小）。如果只是执行简单的任务，比如打印几个字符串、输入一些数据等等，这样的单窗口程序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 窗口上的当前光标位置输出“Hi! There!”。同样，调用 refresh()函数，只更新 stdscr 上的输出内容。

例如，你已经建立了一个叫做 win 的窗口并要输出以上内容，只需在以上的输出函数前添加 w 前缀就可以了。当然，函数中的参数也要做出相应的变化（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。每个相关函数都对应有三个甚至更多的处理函数。

```
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)坐标处然后打印字符串 string */
```

由此可以看出，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 输出函数

我猜你已经等不及去尝试一些具体的操作。在之前的章节，我们通过一些例子，对整个 curses 系统及其函数有了大概的认识。现在，让我们一起走进 curses 的精彩世界。

在 curses 函数中有三类输出函数，它们分别是：

1. **addch()系列**：将单一的字符打印到屏幕上，可以附加加字符修饰参数的一类函数。
2. **printw()系列**：和 printf()一样的具有格式化输出的一类函数。
3. **addstr()系列**：打印字符串的一类函数。

这几类函数可以交替使用。关键是要了解各类函数的具体应用环境。让我们来看看这些函数。

6.1 addch()系列函数

addch()函数用于在当前光标位置输入单个字符，并将光标右移一位。你可以使用这个函数输出一个字符，并给其添加修饰。在后面的章节中我们将会对其做出详细的介绍。
如果一个字符关联有修饰效果（比如：粗体、反色等等），那么当 curses 输出这个字符的同时就会应用相关的修饰。

给单个字符关联属性有两种方法：

- 使用修饰宏（通过“或”运算符）修饰。这些修饰宏可以在头文件 ncurses.h 中找到。比如，你想输出一个具有加粗（BOLD）和加下划线（UNDERLINE）效果的字符变量 ch，可以使用下面这种方法：

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

- 使用 attrset()、attron()、attroff()修饰函数。简而言之，它们将当前的修饰关联于给定的窗口。一旦设置完成，则在相应窗口中输出的字符都会被修饰，直到关闭窗口。这些函数将在文字修饰一章介绍。

另外，curses 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、水平、垂直线条等等，这些特殊字符都是以 ACS_作为开头声明的宏并保存在头文件 ncurses.h 里。

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

mvaddch()用于将光标移动到指定位置输出字符。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。不同的是，waddch()函数是将字符输出到指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处输出字符。

现在，我们熟悉了一些基本的输出函数。但是，如果我们要输出字符串，像这样一个一个 的输出字符是很烦人的。幸好，ncurses 为我们提供了像 printf()和 puts()一样方便的函数。

6.3 printw()系列函数

这些函数的用法和我们熟悉的 printf()函数相似，但增加了可以在屏幕任意位置输出的功能。

6.3.1 printw()函数和 mvprintw()函数

这两个函数的绝大部分用法和 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。如果你想先移动光标，再调用 printw()函数，也就是说先调用 move()函数，然后调用 printw()函数。我不知道为什么有些人非要使用这两个函数代替一个函数，当然决定权在你手里。

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，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

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;
}
```

上面这个程序展示了使用 printw()系列函数输出字符是多么简单。你只需要修改要输出的行列坐标和要打印的信息，它就会按你的需要输出。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，可以给出指定窗口的行列数目。getmaxyx()是通过修改所给变量实现这一功能的。getmaxyx()不是一个函数，因此不能以指针作为参数，只能通过两个整型变量来实现。

6.4 addstr()系列函数

addstr()函数用于在指定窗口输出字符串，如同连续使用 addch()函数来输出指定字符串中的每个字符。实际上，也就是所有 addstr()系列输出函数的事实。addstr()系列函数还包括 mvaddstr()、mvwaddstr()和 waddstr()，它们有着相同的 curses 命名规则和调用方法（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()，它需要一个整型参数 n，用来打印字符串中的前 n 个字符。如果这个参数是负数，addnstr()将会打印整个字符串。

6.5 提醒

所有这些函数中使用坐标时，先给定 y 坐标，再给定 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，而错误的先 x 后 y。如果你进行了过多行列坐标的操作，想想看将屏幕分割成几个不同的窗口，然后独立处理每一个单独窗口的情形。窗口的详细内容会在窗口一章中介绍。

第七章 输入函数

如果你的程序只有输出而没有输入，那是非常单调的。让我们来看看处理用户输入的函数。输入函数也被分为三种：

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

7.1 getch()系列函数

这个函数用于从键盘读入一个字符。但是在使用它的时候需要考虑一些微妙的情况：例如你没有事先调用 cbreak()函数的话，curses 不会连续读取你输入的字符，除非在输入这些字符之前遇到了换行符或者文末符（EOF）。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。另外一个比较广泛的做法是使用 noecho()函数。这个函数被调用时，用户输入的字符不会立即显示在屏幕上。cbeak()和 noecho()是两个用来管理键盘输入的典型函数。键盘管理的具体内容将在键盘管理一章中介绍。

7.2 scanw()系列函数

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读入格式化字符串的功能。

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()函数将在本章后半部分介绍，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，它们需要指定窗口的指针作为第一个参数。

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()系列函数

这些函数用于从终端读取字符串。本质上，这个函数执行的任务和连续用 getch()函数读取字符的功能相同：在遇到回车符、新行符和文末符时将用户指针指向该字符串。

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文字修饰（Attributes）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“*”的时候开启修饰*/
      {
            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);/* 打印实际内容的部分 */
           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“/”的时候关闭修饰*/
    prev = ch;
  }
  getch();
  endwin();                     /* 结束并退出 curses 模式 */
  return 0;
}
```

我们把注意集中在上面这段代码的 while 循环体中。这个循环体读取文件中的每个字符并寻找有“/*”（注释起始处标志）的地方。一旦找到，就会调用 attron()函数开始启动文字加粗的修饰。当找到“*/”（注释结束处标志）的地方，就会调用 attroff()函数停止为后续文字继续添加修饰。

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，需要注意的是我们不能用指针作为参数，只能传递一对整型变量（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 先行后列，即先写 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列后行的方式。**（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 语言源代码十分有帮助。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

最后一个修饰是最吸引人的，它可以设置输出的字符的颜色以及背景的颜色。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所有修饰属性。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，使得函数之间的调用会难以管理。这种情况，尤其在某些用到菜单系统的程序中十分普遍。所以，事先做好设计，然后按照设计去实施效果会好得多。另外，你可以经常使用 standend()函数关闭所有设置的修饰。这个函数和 attrset(A_NORMAL)函数的作用是相同的。

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。注意：这个函数必须在调用了 attrset()或者 attron()之后才能使用。

8.4 attr_ 类函数

这些函数都有 attr_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

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。

当我们给整型参数赋值为-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 chgat()差不多，只不过 wchgat()要指定一个窗口，而 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，没什么特殊含义。
            */
    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 系统的核心。通过前面的例子我们看到了基于“标准窗口”（stdscr）的一些操作函数。即使设计一个最简单的图形用户界面（GUI），都需要用到窗口。你可能需要将屏幕分成几个部分并分别处理，然而，将屏幕拆分成各个窗口，然后独立处理每个窗口是比较高效的方法。使用窗口的另外一个重要原因是：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，事先设计好各个部分将会提高你的编程效率。

9.1 基本概念

一个窗口是通过调用 newwin()函数建立的。但当你调用这个函数后屏幕上并不会有任何变化。因为这个函数的实际作用是给用来操作窗口的结构体分配内存，这个结构体包含了窗口的大小、起始坐标等信息。可见，在 curses 里，窗口是一个假想的概念，用来独立处理屏幕上的各个部分。newwin()函数返回一个指向窗口结构的指针，像 wprintw()等函数都需要以窗口指针作为参数。delwin()函数可以删除一个窗口，并释放用来存储窗口结构的内存和信息。

9.2 显示窗口

可惜的是，当我们创建了一个窗口之后却无法看见它，所以我们现在要做的就是让窗口显示出来。box()函数可以在已定义的窗口外围画上边框。现在让我们看看下面程序中的函数：

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，这的确是一个很大的程序，但它确实讲解了一些很重要的东西：它创建了一个窗口，并且可以使用方向键来移动它。当用户按下方向健的时候，它会删除现有的窗口并在下一个位置建立新窗口，这样就实现了窗口移动的效果。注意：移动窗口时不能超过窗口行列的限制。下面就让我们逐行的分析这个程序：

creat_newwin()函数使用 newwin()函数创建了一个窗口，并且使用 box()函数给窗口添加了边框。destory_win()函数首先使用空白字符填充窗口，从而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以新坐标为起点建立一个新窗口。

在 destory_win()中，我们使用了 wborder 来替代 box。这样做的原因已经写到程序注释里了（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。为了理解得更明白一些，你可以试着这样调用 wborader()函数：

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

他所绘制的窗口会是以下这样子：

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

9.4 更多的说明

从上面的例子中还可以看到，函数使用了 COLS 和 LINES 作为变量名。在 initscr()函数初始化屏幕以后，这两个变量分别存储屏幕初始化后的行数和列数。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。同时，根据用户的输入做出程序中定义的操作。这种做法在交互式的图形界面应用程序中非常普遍。

9.5 其它的边框函数

上面这个例子所使用的方式是通过按下键盘上相应的按钮撤消一个窗口并建立一个新的窗口，但是这样的工作方式效率太低。现在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，它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);
      /* 以下代码初始化窗口的参数 */
      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

如果生命中没有颜色将会单调无趣。curses 有一个非常不错的颜色处理机制。让我们通过以下程序来了解一下颜色系统：

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，要启动彩色机制，必须先调用 start_color()函数，之后就可以在终端屏幕上调用其它处理颜色的函数。如果要检测当前屏幕是否支持彩色显示，可以调用 has_colors()函数，如果终端屏幕不支持彩色显示，那么该函数将返回 FLASE。

在调用 start_color()函数后，curses 就初始化了当前终端支持的所有颜色。然后就可通过像 COLOR_BLACK 这样的宏调用各种颜色。你现在如果要使用颜色，就必须成对定义前景色和背景色。所有的颜色都是这样使用的。这意味着你必须用 init_pair()函数给每一对颜色编号并为其设置前景色和背景色。之后这个编号就作为调用颜色对的参数，传递给 COLOR_PAIR()函数，用来调用你已定义的颜色对。也许一开始你会觉得这样做很麻烦，但它会让你很轻松管理颜色对。“dialog”就使用了这种管理颜色的方法。你可以通过源代码了解这个用 Shell 脚本编写的对话框。开发者应该在程序开始的部分定义并且初始化所需要使用的颜色对常量。这样做会让我们更容易的设置颜色属性。

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。你就可以这样调用此函数：

```
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(COLOR_RED)的定义是 R：1000，G：0，B：0 。

10.3 颜色定义内容

color_content()函数和 pair_content()函数可以用来查看当前颜色的设置。

第十一章 键盘管理

11.1 基础知识

每一个 GUI（图形用户界面）都有强大的用户交互界面。一个 curses 程序应该对用户的输入（仅通过键盘和鼠标）及时的做出反应。那就让我们从处理键盘开始。

就像前面章节中的例子那样，很容易就能取得用户的输入。一个最简单的方法是使用 getch() 函数。如果你喜欢处理单个按键，而不是处理一行的话（经常以回车键作为一行结束标志），你应该在读取按键之前激活 cbreak 模式。如果要读取功能键则应该激活 keypad。这两个函数的用法详见**初始化**一章。

getch()返回一个整数来对应键盘上的按键。如果你输入的是一个普通字符，这个整数就等价于该字符。如果是其它字符，它就返回一个在 curses.h 中已定义的常量相匹配的值。例如用户按下了 F1，返回的整数将是 265，该值可以通过 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，就会返回相应的整数，之后你可以检测这个值是否是你要按的键。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

你现在已经知道如何取得键盘的输入了，那现在让我们也来取得鼠标的输入。因为很多用户界面程序都支持使用键盘和鼠标的共同操作。

12．1 基础知识

在使用鼠标之前，首先要调用 mousemask( )这个函数来激活你想要接收的鼠标事件。

```
mousemask(   mmask_t newmask,          /*  你想要监听的鼠标事件掩码  */
             mmask_t *oldmask )        /*  旧版本使用的鼠标事件掩码   */
```

上述函数中的第一个参数，就是你所要监听的事件的位掩码，默认情况下，在使用该函数之前，所有鼠标事件的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。

下面是 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 2 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

当所有的鼠标监听事件被激活后。getch()一类的函数在每次接收到的鼠标事件时可以返回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

能够使用鼠标操作的程序是非常棒的，让我们做用鼠标操作的菜单程序。为了让例子看起来更有针对性，这个程序中去掉了键盘操作：

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(intmouse_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)的联机 man 文档。

mouseinterval()函数用来设置识别鼠标单击的间隔时间（即按键按下和弹起的时间，按千分之一秒计），并返回上一次设置的间隔时间，默认的间隔时间是五分之一秒，即 mouseinterval(200)。

第十三章 屏幕操作

在这一章我们将了解一些管理屏幕的函数。这些函数经常被用于编写基于控制台的游戏。顺便让我们写一些非常有意思的程序。

13．1 getyx()系列函数

getyx() 函数可以用来取得当前光标的位置。并把它存储在传递给它的两个变量中。getyx()是一个宏，所以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主窗口的起始坐标，它在更新子窗口时经常使用。当设计一个多级菜单时，如果用存储菜单坐标方法来处理，就变得非常困难。然而使用 getparyx()函数找到该菜单的相对坐标的方案就显的比较简单。getbegyx()函数和 getmaxyx()函数用于以同样的方式存储当前窗口的起始和结束坐标，可以高效地管理窗口和子窗口。

13．2 屏幕转储

开发游戏的时候，通常存储和恢复屏幕是十分必要的。 scr_dump()函数可以把当前屏幕的内容存入指定文件，即以文件名作为函数的参数（函数原型：scr_dump(const char *file) ——译者注）。而通过 scr_restore()函数调用屏幕数据文件来恢复屏幕（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快速切换游戏场景。

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）则调用相应的文件来恢复窗口。

copywin()可以将一个窗口拷贝到另一个窗口，即将源窗口矩形区域（由参数指定）中的内容复制到目标窗口指定的矩形区域（由参数指定）里。而最后的参数用来选择是否要覆盖目的窗口：如果参数为 TRUE，那么就会覆盖目标窗口的内容；如果参数为 FLASE，那么就会重写目的窗口中的内容。以下为函数的原型：

```
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大列数 */
        int   over)             /* 是否覆盖目的窗口 */
```

第十四章 其它特色

现在你所掌握的函数可以写出一个非常不错的 curses 程序了。这里还有一些很有趣的函数可以为你的程序增色。

14．1 curs_set()函数

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

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它的事。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，让你回到最初的 tty 模式。如果你结束了 tty 模式下的工作，想要返回 curses 模式，就需要调用 reset_prog_mode()函数，它会将 def_prog_mode()函数保存的信息重新读入到虚拟的屏幕上。之后必须通过 refresh()函数刷新屏幕，才可以返回到原先保存的 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_常量

如果你曾经在 DOS 下编写过程序，就应该知道扩展字符集。但是这些字符集中的字符只能在少数的终端上显示。例如 NCURSES 的 box()函数（译者注：这个函数用来绘制一个矩形框）就使用了这些扩展字符。所有这些字符都是以 ACS_作为前缀的常量，所谓 ACS，就

是 Alternative Character Set（可选字符集）的缩写。你可以注意到在以前的程序中多多少少都用到了这些有意思的字符。下面这个程序分别介绍这些字符：

例 13：ACS 常量介绍例子

```
#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

curses 函数除了主函数库外，还有一些具有很多新功能和特性的字符文本模式的扩展库。以下章节将分别介绍与 curses 一起发布的三个扩展库（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

在精通 curses 库后，你可能会想尝试着做一些更大的项目。为了让界面看起来更专业，你可能会创建许多重叠的窗口，但不幸的是，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得异常困难。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时置底或者置顶，虽然会为此付出很大代价，但是想要显示正确的结果仍然很困难。这时候 Panels 库就可以派上用场了。

如果你要处理重叠的窗口，panels 库就派上用场了。它使程序员避免使用大量的 wnoutrefresh()和 doupdate()函数来处理这个问题，并且减轻了由底向上正确处理这些信息的负担。这个库维持着窗口重叠关系以及窗口顺序，并在适当的时候更新它们。

那么还等什么呢？让我们开始吧！

16.1 基础知识

面板对象实际上是一个窗口，和其它窗口一样被隐含地处理为容器的一部分。这个容器实际上是一个栈，栈顶的面板是完全可见的。而其它面板在栈中所处的位置，决定了它们是否可见。其基本思想是：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。例如调用一个类似于 refresh（）函数就按正确的顺序来显示这些面板。面板库提供了一系列隐藏、显示、移动、改变大小等面板操作的函数，使操作重叠的窗口变得更加简单。

通常，一个面板程序的设计流程如下：

1. 使用 newwin()函数创建一个窗口，它将添加到面板里。
2. 创建面板（利用所创建的窗口）并将面板依据用户指定的可见顺序压进栈。调用 new_panel()函数即可创建该面板。
3. 调用 update_panels()函数就可将面板按正确的顺序写入虚拟屏幕，调用 doupdate()函数就能让面板显示出来。
4. show_panel(), hide_panel(), move_panel()等函数分别用来对面板进行显示、隐藏、移动等操作时，可以使用 panel_hidden()和 panel_window()这两个辅助函数。你也可以使用用户指针来存储面板的数据，set_panel_userptr() 和 panel_userptr()函数分别用来设置和取得一个面板的用户指针。
5. 当一个面板使用完毕后，用 del_panel()函数就可删除指定的面板。

现在让我们通过一些程序来加深对这些概念的理解。下面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，同时编译并连接与面板库相关的程序必须添加-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，为每一个面板关联一个窗口*/
        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，这个程序就是按照 16.1 所介绍的流程进行的。用 newwin()函数来创建窗口，然后通过 new_panel()函数把窗口添加到 panels 栈里面，当那些面板一个一个压进栈的时候，栈就随之更新了。最后调用 update_panels()函数和 doupdate()函数就可以让它们在屏幕上显示出来。

16.3 面板浏览

下面给出一个较复杂的例子。它创建了 3 个窗口，通过<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

在上面例子中，使用用户指针在循环里查找下一个要显示的面板。我们可以通过指定一个用户指针给面板添加自定义信息，这个指针可以指向你想要存储的任何信息。在这个例子中，我们用指针存储了循环中下一个要显示的面板。其中，用户指针可以用 set_panel_userptr()函数设定。要想访问某个面板的用户指针，就必须以该面板作为 panel_userprt()函数的参数，函数就会返回该面板的用户指针。结束面板的查找后， top_panel()函数就会将其置于面板栈的顶层。要想将任意一个面板置于面板栈的顶层，只需将该面板作为 top_panel()函数的参数。

16.5 移动面板和改变面板的大小

move_panel()函数可将面板移动到屏幕上指定的位置，而不是改变面板在栈中的位置。确保在移动面板窗口时使用 move_panel()函数，而不是 mvwin()函数。

改变面板大小有点儿复杂，因为没有一个函数可以直接改变面板所关联窗口的大小。一个可替代的解决方案是按照所需的大小创建一个新窗口，再调用 replace_panel()函数来替换相应面板上的关联窗口。别忘了替换后删除原窗口，使用 panel_window() 函数可以找到与该面板关联的窗口。

下面这个程序就体现了这种思想。你可以像先前那样，用<Tab>键循环查看窗口。如果要改变当前面板大小或移动当前面板的位置，就要分别按下‘r’或‘m’键，然后使用方向键来调节，最后以<Enter>键确定大小或者位置.。这个例子利用用户数据保存程序运行的必要数据。

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':  /* 改变面板大小*/
                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':/* 移动面板 */
                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();
}
```

让我们把注意力集中在主 while 循环体上。一旦该循环发现某个键被按下，程序就会执行该键相应的处理。当按下‘r’键时，程序就会执行“更改大小”操作，同时你就可以通过方向键更改面板的大小，然后按<Enter>键确定大小。在“更改大小”模式下，程序不会显示窗口是怎样更改大小的。请读者思考一下如何用“点”来打印更改大小后窗口的边框。

当按下‘m’键时，程序就会执行“移动面板”操作。这个操作比“更改大小”简单一点。按下方向键的同时，面板的位置随之改变，当按下<Enter>键时，程序就会调用 move_panel()函数把面板固定到当前光标的位置。在这个程序中， PANEL_DATA 就是所谓的用户数据，它在查找面板的相关信息时扮演重要角色，正如说明中所说的那样，PANEL_DATA 保存了面板的尺寸，标题，标题颜色以及指向下一个面板的指针。

16.6 隐藏和显示面板

使用 hide_panel()函数可以隐藏面板，它仅仅是把面板从面板栈中移走，不会破坏被隐藏面板所关联的结构。而要在屏幕上隐藏面板需要调用 update_panels()函数和 doupdate()函数。此外，show_panel()函数可以让显示已隐藏面板。

以下程序展示了面板的隐藏。按下'a' 或 'b' 或 'c' 键分别显示或隐藏第一、二、三个窗口。使用了用户数据中一个叫做 hide 的变量来跟踪窗口，标记出该窗口是否被隐藏。由于某些原因，panel_hidden()函数（它用来告诉用户一个面板是否隐藏）不能够正常工作。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]);
    /* 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*/

    /* 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

panel_above()和 panel_below()函数可以分别用来查看某个面板的上层和下层面板。如果参数为 NULL，它们就分别返回面板栈中最上层和最下层面板的指针。

第十七章 菜单库

菜单库对 cureses 基础库进行了很好的扩展。你可以通过这个库所提供的函数方便的创建菜单。如果你想让它更美观，可以定制它的显示效果。下面我们就来看看这个库。

菜单是一个用来帮助用户选择子菜单项的屏幕。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。 curses 菜单库还提供编制多选菜单的功能。有些读者可能不了解多选菜单。这个我们稍后讨论，我们先来了解一下菜单库的基础知识。

17．1 基础知识

要创建菜单，你首先要建立菜单项，然后发送并显示菜单。接下来，所有处理用户响应的工作就交给功能强大的 menu_driver()函数来完成。这个函数是整个菜单库的核心。

一个菜单程序大致的控制流程如下：

1．初始化 curses。
2．用函数 **new_item()**创建菜单项，同时为菜单项指定名称并且描述其相应的功能。
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 **post_menu()**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 **unpost_menu()**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菜单的示例程序，它是用方向键来更新当前菜单项的。

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 文件包含进去。编译并连接时要同时添加-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。首先用 new_item()函数建立菜单项，然后用 new_menu()函数把这些菜单项添加到菜单。当递送了菜单并刷新屏幕后，主循环就开始处理。它读取用户的输入并进行相应的操作。函数 menu_driver()是菜单系统的核心函数。这个函数的第二个参数是相应菜单操作宏。 menu_driver()函数根据参数执行相应的操作。参数的值可以是菜单的导航请求、一个 ASCII 码或与鼠标事件相关的一个特定 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：菜单系统的核心

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。前面已经解释过， menu_driver()的第二个参数可以是一个导航请求。一个可打印的字符（ASCII 码）或 KEY_MOUSE 键值。我们来剖析一下各个导航请求：

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。将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_系列的移动请求

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。当这些请求传给 menu_driver 时，将分别向上/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、最后一个菜单项、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（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样式匹配缓冲区。通过这个缓冲区，用户可以通过输入 ASCII 字符串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，它同时试着从菜单项列表中查找与之最近的匹配项。然后立刻从当前菜单项跳转到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** 清除模式缓冲区中前一个匹配样式。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** 进行切换，这两个选项分别从当前菜单项移至下或上一个匹配样式。

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 帮助页中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导航请求作详细的讲解。

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。而在上面的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 将作为菜单窗口。然后菜单系统根据将要显示的菜单项计算子窗口的大小。菜单项就在这些规划好的子窗口中显示出来。让我们利用这些窗口，来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，以及一根用来分隔开标题和菜单项的线。如你所见，使用 set_menu_win()函数把菜单附加到一个窗口上，之后使用 set_menu_sub()函数把菜单的子窗口也附加到这个窗口上，菜单项就可以在子窗口中显示。使用 set_menu_mark()函数可以来设置标志串，标志串就会出现在所选菜单项的左边。

17.5 滚动菜单

如果设置的子窗口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处传递 REQ_DOWN_ITEM 参数，它将会变成 REQ_SCR_DLINE，并滚动到下一个菜单项。你也可以手动的用 REQ_SCR_操作来滚动菜单。现在让我们来看看怎样实现菜单滚动。

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();
}
```

在本例中，菜单项的数目增加到了 10 个，超出了我们预先设置的显示数目——6 项。这个信息可以通过函数 set_menu_format()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菜单项数目如果用户按下 PAGE UP 或 PAGE DOWN 键，相应的菜单请求(REQ_SCR_UPAGE 和 REQ_SCR_PATTERN)将传递给 menu_driver()函数，因此菜单就变成可滚动的了。

17．6 多列菜单

在上面的例子中你已经了解到函数 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，你可以选择让每行显示多个菜单项，这可以在列变量里设置。简单其见，本例省去了菜单项的描述。

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();
}
```

我们来仔细看一下 set_menu_format()函数的调用，它指定列数为 3，即每行就显示 3 列菜单项。我们已经用 menu_opts_off()函数关闭了描述信息的显示。set_menu_opts(), menu_opts_on()和 menu_opts()也是用来设置菜单选项的函数。

以下就是一些常用的菜单选项：

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。你可以通过调用 menu_opts_on()和 menu_opts_off()函数用来开关某个选项，也可以调用 set_menu_opts()函数来直接指定这些选项。如果想同时打开多个选项，应该用“|”（或）分隔参数中的不同选项。调用 menu_opts()函数可以查看当前菜单的选项。

17.7 多选菜单

把选项 O_ONEVALUE 关闭之后，菜单变成多选菜单，你就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，现在一起来看看吧：

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。让我们一个一个来看吧。先来看看 REQ_TOGGLE_ITEM 请求：在一个多选菜单里，用户可以选择多个菜单项， REQ_TOGGLE_ITEM 请求用来锁定当前已选择的菜单项。在这种情况下，当按下空格键时， REQ_TOGGLE_ITEM 请求就被传给 menu_driver，当前菜单项就被锁定。

如果用户按下<ENTER>键，就会显示当前已选的菜单项。首先，通过 menu_items()函数可找出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被选。如果一个菜单项已选，item_value()函数返回 TRUE。item_count()函数用来返回菜单的菜单项数；item_name()函数用来找菜单项的名字； item_description()函数用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菜单里弄出点花样来，菜单功能要更加强大且是彩色的。就像这里的文本模式 DOS 游戏。set_menu_fore()和 set_menu_back()函数可以分别改变选中和未选中菜单项属性，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！

set_menu_grey()函数可以用来设置菜单中不可选菜单项的显示属性。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 item_opts_off()函数来关闭这个选项，然后被设置的菜单项就不可选了，且在菜单窗口中是灰色的。现在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 实用的用户指针

我们可以使用用户指针关联菜单中的每一个菜单项，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你想要存储的任何信息。我经常用它来存储所选菜单选项要执行的函数（该菜单项已被选并且用户按下了<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

当你看到网页上处理用户数据的表单（Forms）时，你肯定很想在字符控制台模式下创建一个那样的表单。用普通的 ncurses 函数创建那样的表单十分困难。而表单库（Form Library）为我们提供了一个基本的框架，使我们可以很容易地创建和维护表单。它包含了很多用来管理，动态扩展表单域的函数。本章就来介绍表单库。

表单是表单域的集合，这些表单域可以是一个标签或一个数据输入框。表单库提供把一个表单划分为多个页面的函数。

18.1 基础知识

表单的创建步骤与菜单的创建步骤大致相同：

首先，用 new_field()函数创建与表单相关联的表单域。你可以设置修饰表单域的选项，使它更美观。在操作焦点（译者注：即表示被选定，通常是高亮条等）离开表单域之前，这些对表单域的修饰属性都是有效的。

然后，表单域被添加进表单，整个表单就会传递到屏幕上，并准备读取信息。与 menu_driver()函数相似，表单由 form_driver()函数操纵。我们可以传递操作请求给 form_driver()函数来实现移动操作焦点到某个表单域，或将光标移动到表单域的结尾等操作。用户在表单域输入数据并确认输入后，表单就不再被传递，并释放所占用的内存。

编写含有表单的程序，一般步骤如下：

1. 初始化并进入 curses
2. 用 new_field()函数创建表单域。你可以指定域的高度，宽度以及它在表单中的位置。
3. 指定已创建的表单域所作用的表单，并用 new_form()函数创建表单。
4. 用 post_form()函数来递送表单，并刷新屏幕。
5. 用一个循环来处理用户请求，通过 form_driver()函数对表单做相应的更新。
6. 用 unpost_form()函数取消表单的递送。
7. 用 free_form()函数释放已分配给表单的内存。
8. 用 free_field()函数释放已分配给菜单项的内存。
9. 退出 curses 模式

如你所见，表单库的用法与跟菜单库很相似。下面的几个例子将带我们领略表单的各个方面。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：

```
#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表单域的高、宽，起始位置的横、纵坐标，不显示数据的行数，和附加的工作缓冲区。其中第五个参数（不显示的行数）决定表单域中哪些部分是可见的，如果是 0，则显示整个表单域，否则当用户访问的表单域超出显示范围时，表单就变成可滚动的。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。最后的一个参数（附加的工作缓冲区）用来分配额外的缓冲区。它可以用来做你想做的任何事情。

当表单域创建后，用 set_field_back()函数可以增添背景修饰效果。在本例中，表单域添加了下划线。AUTOSKIP 选项已用 field_opts_off()函数关闭。如果这个选项是打开的，当前的表单域一旦被输入的数据填满，光标就自动跳转到下一个表单域。给表单添加完表单域后，表单就被递送出去。接着，用户输入的信息将在 while 循环中作为对 form_driver()函数发送操作请求来处理。

有关 form_driver()的细节将在后面的部分详细解释。

18.3 表单域控制

每个表单域关联着大量的属性，你可以操作这些属性来实现想要的效果。是不是很有趣，那么你还等什么呢？

18.3.1 获取域的大小和位置

我们可以用 field_info()函数来重新得到在创建表单域时所设置的参数。该函数返回表单域的高、宽，起始位置的横、纵坐标、不显示的行数和附加的缓冲区。它和 new_field()函数的功能相反：

```
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()函数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()函数查询到。

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()函数设置。

```
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);                    /* 返回表单域的对齐方式 */
```

这两个函数可以用来设置或者返回相应表单域对齐方式，它们是：NO_JUSTIFICATION（不对齐），JUSTIFY_RIGHT（靠右对齐），JUSTIFY_LEFT（靠左对齐）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，表单域的显示属性可以用 set_field_fore()和 set_field_back()两个函数来设置，它们分别设置表单域的前景和背景修饰属性。你也可以用指定字符（pad）填充该表单域的空白处作为修饰。通过调用 set_field_pad()函数可以设

置背景填充字符，默认填充字符是空格符。field_fore()、field_back()、field_pad()函数可以用来查询当前表单域的前景、背景修饰参数和背景填充字符。以下是这些函数的函数原形：

```
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()函数设置前景、背景的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属性，前景属性与字符有关。即在表单域内的字符是以 set_field_for()函数所设置的修饰效果打印的。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，如何才能正确显示包含颜色设置的表单域呢？下面的例子清楚地解释了颜色的定义方法。

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();
/* 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;
}
```

你通过实际操作，可以更好的理解颜色定义以及前景和背景的修饰。在我的程序里面，若要设置颜色属性，我一般只用 set_field_back()函数来设置背景修饰。在 cureses 里不允许单独定义颜色，颜色必须成对定义。

18.3.5 表单域属性

表单库提供了大量的表单设置的选项，这些选项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,          /* 要设置属性的表单域 */
                   int attr);            /* 要设定的属性 */

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

int field_opts_off(FIELD*field,          /* 要改变属性的表单域 */
                   int attr);            /* 要关闭的属性 */
int field_opts(FIELD*field);             /* 要查询的表单域 */
```

set_field_opts()函数可以直接设置表单域属性。而启用或关闭某些表单域属性的时，就要用到 field_opts_on()或 field_opts_off()函数。你可以随时用 field_opts()函数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，它们都已启用。

O_VISIBLE

控制表单域在屏幕上是否可见。根据上一级表单域的值来显示或隐藏表单域。

O_ACTIVE

控制表单域在表单处理时是否是激活状态，这样的操作类似 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跳转标签或起始表单域，缓冲区的大小由应用程序自身给定，而不是用户。

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，表单系统仍然允许在这个表单域中编辑数据。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。例如，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。当这个选项被关闭时，除了**REQ_PREV_CHOICE** 和 **REQ_NEXT_CHOICE** 请求以外其它所有的编辑请求都将失效。你可以设置这样的只读表单域来显示帮助信息。

O_WRAP
控制多行表单域的自动换行功能。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）。当这个选项被关闭时，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，如果在表单域的起始位置输入字符时，程序会自动清空整个域(正在输入的这个字符除外)。

O_AUTOSKIP
该选项控制表单域填满时，光标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
控制检查空白表单域是否是有效输入，默认是不检查。用户可以在退出时，可以使一个表单域为空，而不去检查这个表单域是否是有效输入。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空白表单域是否是有效输入。

O_PASSOK
控制表单域的有效性检查是发生在每次退出表单域时，还是在表单域的数据被修改之后。默认是在表单域的数据被修改后检查。该选项被设置后，用户在退出表单域后将会检查表单域的有效性。如果你想在表单域被修改时，用你自己的检查函数检查表单域数据的有效性。设置 O_PASSOK 选项会很有帮助。

O_STATIC
控制表单域是否固定为初始大小。如果关闭这个选项，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，表单域的属性不能被修改。但可以在表单域递送后修改其属性。

这些属性都是**位掩码**，很显然可以用“按位或”逻辑运算符组合。你已经看见了如何关闭 O_AUTOSKIP 属性。下面的这个例子将说明更多的属性用法。其它的属性会在别的地方说明。

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;
}
```

虽然这个例子用处不大，但却展示了这些属性的用法。如果适当的使用这些属性，它们可以在表单中有效地显示信息。第二个表单域关闭了 O_PUBLIC 选项，所以不会显示你所输入的字符。

18.3.6 获取表单域的状态

表单域的状态用来标记表单域是否被修改。它的初始值为 FALSE（即未被修改状态），当用户在表单域中输入数据，并修改了数据缓冲区后，该值就变为 TRUE（即已修改状态）。可以通过下面的函数检查一个表单域的状态来检测它是否被修改了。

```
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名 */

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，所以数据缓冲区有可能还没有更新。为确保返回正确的表单域状态，最好在下面三种情况下调用 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，它可以使用户做很多事情，而表单库不能使用它。下面两个函数分别用来设置和获取用户指针。

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

char *field_userptr(FIEL D *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地改变大小表单域的话，表单库正好提供了这个功能。当用户输入的数据大小超过该表单域的原大小时，文本框的尺寸就会相应的动态改变。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。

为使一个表单域的大小可以动态地改变，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 */
```

虽然也可以使用 field_info()函数，但我建议使用这个函数来取得动态表单域的属性。

再次调用 new_field()函数，它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将会被自动定义为多行表单域。

一个关闭了 O_STATIC 选项的单行表单域（可以动态增长的单行表单域），虽然只有一行，但是如果用户输入的数据超过初始大小的话，表单域宽度将自动增长。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（动态增长的多行表单域）包含固定的宽度，但是如果用户输入的数据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后，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。而在定义了最大增长值的动态表单域里，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、 同理，如果 O_STATIC 选项关闭，表单域的对齐请求也将被忽略。通常，set_field_just () 函数可以通过 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了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 field_just()函数返回的值没有变化。

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 请求将会被忽略，但 O_NL_OVERLOAD 选项正常执行。通常，如果表单选项 O_NL_OVERLOAD 是打开的，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。一旦指定了最大动态增长的范围，并且 O_NL_OVERLOAD 选项是打开的，当表单域自动增长到最大，且用户停留在最后一行时，发送 REQ_NEW_LINE 请求就会隐式生成 REQ_NEXT_FIELD 请求了。

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()函数时的值。用户应该通过上面叙述的 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相关联。主窗口用来显示标题、边框以及其它的东西。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操作很相似，所以使用的函数和工作原理大致相同。所以这里并不做过多的解释。

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。也可以为表单域附加输入有效性的检查。这样的话，如果域里面有和检查项不相符的数据，输入数据将会失效。有些检查选项也支持即时检查——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 只检查由终端用户输入数据的有效性。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()函数改变表单域的数据；
- 已连接的表单域间接的被修改了——被其他连接到该域的表单域修改。

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（字母输入检查模式）

这个检查模式只接受字母，不接受空格、数字、特殊字符（表单域将在输入时对输入的字 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，（表单域将在输入时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）。这种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就会区分大小写。

当用户退出一个附加 TYPE_ENUM 检查的表单域时，程序会自动将缓冲区内的数据补充完整并使之有效。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 checkunique 参数为真并要求输入有效的话，则要求匹配项在集合内是唯一的。

REQ_NEXT_CHOICE 和 REQ_PREV_CHOICE 请求对这些表单域将会特别有用。

TYPE_INTEGER（整数检查模式）
这种检查方式检查表单域的值是否是一个整数。它可以这样设置：

```
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“－”号的整数（即负整数）。当用户离开这个表单域后才会检查输入值是否在规定范围内。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。如果输入的整数在限制范围内，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。

TYPE_NUMERIC（数值检查模式）
这种检查方式检查表单域的值是否是一个十进制小数或整数。它可以这样设置：

```
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，有效字符可以为实数也可以是带“－”号的正实数（即负实数）。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。如果输入的整数在限制范围之内，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。

TYPE_NUMERIC 类型值的缓冲区可以用 C 语言的库函数 atoi（3）解释。

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，它可以这样设置：

```
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：表单系统的核心

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

```
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表单域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()函数，由它来完成相应的工作。

表单请求可以大致地分为下面几类。让我们一起来看：

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，你就可以将这个表单设置为多页的。set_new_page()函数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 */
```

下面的导航请求用来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，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，这些表单域必须在同一个页面内。

- **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表单域。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列表中的指针顺序（用 new_form()或 set_form_fields()函数设置）。

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。）

假设你有一个多行表单域 B，以及两个与 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_NEXT_WORD** 移动到下一个单词
- **REQ_PREV_WORD** 移动到前一个单词
- **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** 跳到表单域的左边的字符
- **REQ_RIGHT_CHAR** 跳到表单域的右边的字符
- **REQ_UP_CHAR** 跳到表单域的上边的字符
- **REQ_DOWN_CHAR** 跳到表单域的下边的字符

用两个字符之间的空格作为单词之间的分隔。如果要让光标移动到该行或该域的起始或末尾处，相应的命令就会在当前范围内查找第一个或最后一个字符。

18.6.4 滚动请求

动态增长大小的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的移动（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** 横向向前滚动一个字符
- **REQ_SCR_BCHAR** 横向向后滚动一个字符
- **REQ_SCR_HFLINE** 横向向前滚动一个域宽
- **REQ_SCR_HBLINE** 横向向后滚动一个域宽
- **REQ_SCR_HFHALF** 横向向前滚动半个域宽
- **REQ_SCR_HBHALF** 横向向后滚动半个域宽

在执行滚动时，表单域一页的高度就是它可视部分的高度。

18.6.5 编辑请求

当你传递一个 ASCII 字符给 Form Driver 时，它将会被添加到表单域的数据缓冲区中去。是插入模式还是替换模式是由表单域的编辑模式决定的（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符
- **REQ_DEL_LINE** 删除光标所在行
- **REQ_DEL_WORD** 删除光标所在的字符

·**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，而且部分请求由一对表单选项控制。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，在此需要特别的说明一下：

首先看一下 REQ_NEW_LINE 请求：
通常 REQ_NEW_LINE 请求的作用是在插入模式下中断光标所在的编辑行。即把当前光标后面的部分作为一个新行插入，并使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 请求在改写模式下会将把从当前光标的位置到其所在行行尾位置的所有内容清除。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，它的功能相当于 REQ_NEXT_FIELD 请求的功能。若选项 O_NL_OVERLOAD 关闭，这个特殊行为就会被禁止。

接下来看一下 REQ_DEL_PREV 请求：
通常 REQ_DEL_PREV 请求的功能是删除前一个字符。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 请求则被当成 REQ_PREV_FIELD 请求处理。

如果 O_BS_OVERLOAD 选项被关闭，这个特殊行为就会被禁止， Form Driver 就会返回 E_REQUEST_DENIED 请求。

18.6.6 顺序请求

如果你的表单域类型是有序类型（TYPE_ENUM），并且已经关联了取得有序类型值的函数（一般包含取得前一个值和后一个值的两个函数）。以下请求会将取得的有序类型值放入缓冲区：

·**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

对于内置的表单域类型，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。当你自己定义了一个表单域类型时（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

表单请求是大于 KEY_MAX 且小于等于常数 MAX_COMMAND 的整数。在这个范围内的值将被 form_driver()函数忽略。这就使得其它应用程序可以任意使用范围外的值。它可以被看成是应用程序中特定行为的代码，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 及其姊妹库的强大功能了吧。你是否有开发一个规模宏大项目的冲动呢？不过请等一下，如果只是直接使用 NCURSES 库及其姊妹库，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情。下面会介绍一些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，你可以用来替代自己编写的组件，或直接使用它们，或从这些组件的代码中汲取编程灵感，甚至还可以扩展这些组件的功能。

19.1 CDK (Curses Development Kit)

在这里先引用 CDK 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 curses 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具有良好的编码风格和非常详尽的文档。示例目录中的例子会给 curses 的初学者一个很好的学习起点。CDK 可以从官方网站 http://invisible-island.net/cdk/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 提示来安装它。

19.1.1 组件列表

下面这个列表就是 CDK 中的组件列表，并附带有相应的说明：

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以外，还解决了很多复杂的 curses 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>"
```

若将这条字符串作为 newCDKLabel()函数的参数，它将打印一行蓝色背景的黄色字符串。当然，还有更多的修饰标记可供选择。这些选项让字符看起来更加美观、精致。你可以通过查阅的 cdk_display(3X)页找到更多详细的资料。man 文档中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。尤其是在编写一个比较复杂的图形用户界面的时候。

19.2 dialog 组件

在 1994 年的 9 月，Linux 还鲜为人知， 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 并通过例子告诉你在何时何地可以用到它……

就像他所说的，dialog 组件是个亮点，它可以方便的创造出专业效果的对话框，也可以创建多种多样的对话框。比如：带菜单的对话框、带选项列表的对话框……它在安装 Linux 的时候已经默认安装了，如果你的电脑上没有，你可以在 Thomas Dickey 的个人站点（http://invisible-island.net/dialog/）下载。

上面给出了 dialog 的作用和功能的概括。man 文档中还有更多关于 dialog 详细的资料，它可以在很多情况下应用。Linux 内核文本模式就是一个很好的应用了 dialog 的例子。Linux 内核用已修改的 dialog 版本定制它所需要的功能。

dialog 最初为 shell 脚本代码的使用而设计的。你如果想将它应用到 C 程序中，可以使用 lobdialog。但是关于 libdialog 的文档非常少，最权威的参考是 dialog.h 文件。你可能需要去改动它得到自己想要的输出。libdialog 库源代码很容易自己定制。我已经在一些场合修改它以符合自己的需要。

19.3 Perl 中的 CURSES 模块

perl 中的 curses 模块：Curses::Form 和 Curses::Widgets 提供了从 perl 访问 curses 的途径。如果你已经安装了 curses 和 perl，就可以从 CPAN 模块页的“Curses”类别页中下载到这些已压缩的模块。在你安装这些模块后，就可以像在 perl 中使用其它模块那样使用它们。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。widgets 模块虽然比较初级，但它提供了从 perl 中访问 curses 良好的途径。

Anuradha Ratnaweera 为本文档的示例程序提供了 perl 语言版本。这些程序源代码可以在示例程序压缩包的 perl 目录下找到。
如果需要更多的资料可以参看 man 文档页 Curses(3)，Curses::Form(3) 和 Curses::Widgets(3)，这些 man 文档页在安装以上的模块的时候附带安装。

第二十章 快乐至上

这一章将介绍这部文档的示例中 JustForFun 目录下的程序。这些程序都是我凭个人兴趣写的，它们并不是很好的编程范例，也不是使用 ncurses 的最佳方法和诀窍。用它们只是为初学者提供一些点子，所以欢迎您为其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，而且希望把它们加入这个章节，可以联系作者。（译者注：联系原文作者可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，请联系译者）

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“细胞”被放置到起点位置，由演变规则确定后面发生的事情。然而，生命总是充满了惊奇！在很多情况下，看见开始位置（或者开始的样子）同时看到未来发生的事 是不可能的。遵循游戏的演变规则是从开始看到未来的唯一方法。

这个程序从一个反转的 U 开始，它会告诉你生命是多么神奇。你有很多空间可以用来改进这个程序。你可以让用户通过输入来选择它的生命模式，甚至可以从文件中读取这些模式。你可以改变规则，玩出更多的花样。在 google 上可以查到很多和生命游戏（game of life）有关的信息。
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。它很容易理解，但很难填入合适的数字满足它的条件：每行、每列、每个斜行的和必须相等。这个程序运行时必须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，中间借助一个柱子临时放置盘子。要求是：小盘子在上，大盘子在下。
文件路径：JustForFun/hanoi.c

20.4 皇后问题（Queens Puzzle）

著名的皇后问题。这个问题要求在国际象棋的棋盘上摆放 N 个皇后，让它们不会相互攻击。这个程序用了非常简单的返回算法。
文件路径：JustForFun/queens.c

20.5 智力拼图（Shuffle）

一个可以用来消磨时间的有趣游戏。你要通过移动空位，将格子里的数字按照由小到大的顺序排列。

文件路径：JustForFun/shuffle.c

20.6 打字练习（Typing Tutor）

一个训练打字的简单程序。如果你对键盘不熟悉（缺乏键盘练习），但是知道使用键盘的正确姿势。这个程序会对你提高打字速度很有帮助。

文件路径：JustForFun/tt.c

第二十一章 参考

关于 NCURSES 详细内容请参考：

NCURSES 的 man 联机文档。

NCURSES 的 FAQ：

http://invisible-island.net/ncurses/ncurses.faq.html

《Writing programs with NCURSES》

作者： Eric Raymond 、 Zeyd M. Ben-Halim

http://invisible-island.net/ncurses/ncurses-intro.html

-这份文档虽然有些陈旧.但这份文档内容和结构是我撰写这部 HOWTO 文档的灵感源泉。