

用 Python 进行 Curses 编程

作者: A.M. Kuchling, Eric S. Raymond

发布版本: 2.04

摘要

本文档介绍了如何使用 [curses](#) 扩展模块控制文本模式的显示。

curses 是什么？

curses 库为基于文本的终端提供了独立于终端的屏幕绘制和键盘处理功能；这些终端包括 VT100，Linux 控制台以及各种程序提供的模拟终端。显示终端支持各种控制代码以执行常见的操作，例如移动光标，滚动屏幕和擦除区域。不同的终端使用相差很大的代码，并且往往有自己的小怪癖。

在普遍使用图形显示的世界中，人们可能会问“为什么自找要麻烦”？毕竟字符单元显示终端确实是一种过时的技术，但是在某些领域中，能够用它们做花哨的事情仍然很有价值。一个小众市场是在不运行 X server 的小型或嵌入式 Unix 上。另一个是在提供图形支持之前，可能需要运行的工具，例如操作系统安装程序和内核配置程序。

curses 库提供了相当基础的功能，为程序员提供了包含多个非重叠文本窗口的显示的抽象。窗口的内容可以通过多种方式更改 --- 添加文本、擦除文本、更改其外观 --- 并且 curses 库将确定需要向终端发送哪些控制代码以产生正确的输出。curses 并没有提供很多用户界面概念如按钮、复选框或对话框等；如果你需要这些特性，请考虑某种用户界面库例如 [Urwid](#)。

curses 库最初是为 BSD Unix 编写的。后来 AT&T 的 Unix System V 版本加入了许多增强功能和新功能。如今 BSD curses 已不再维护，被 ncurses 取代，ncurses 是 AT&T 接口的开源实现。如果使用的是 Linux 或 FreeBSD 等开源 Unix 系统，则几乎肯定会使用 ncurses。由于大多数当前的商业 Unix 版本都基于 System V 代码，因此这里描述的所有功能可能都可用。但是，某些专有 Unix 所带来的较早版本的 curses 可能无法支持所有功能。

Python 的 Windows 版不包括 [curses](#) 模块。一个可用的移植版本是 [UniCurses](#)。

Python 的 curses 模块

此 Python 模块是对 curses 所提供的 C 函数的一个相当简单的包装器；如果你已经熟悉在 C 中进行 curses 编程，把这些知识转移到 Python 是非常容易的。最大的差异在于 Python 接口通过将不同的 C 函数比如 `addstr()`，`mvaddstr()` 和 `mvwaddstr()` 合并为一个 [addstr\(\)](#) 方法让事情变得更简单。你将在稍后看到更详细的介绍。

本 HOWTO 是关于使用 curses 和 Python 编写文本模式程序的概述。它并不被设计为一个 curses API 的完整指南；如需完整指南，请参见 ncurses 的 Python 库指南章节和 ncurses 的 C 手册页。相

对地，本 HOWTO 将会给你一些基本思路。

开始和结束 curses 应用程序

在做任何事情之前，curses 必须被初始化。这是通过调用 [initscr\(\)](#) 函数来完成的，它将确定终端的类型，向终端发送任何必须的设置代码，并创建多种内部数据结构。如果执行成功，`initscr()` 将返回一个代表整个屏幕的窗口对象；它通常会按照对应的 C 变量被称为 `stdscr`。

```
import curses
stdscr = curses.initscr()
```

使用 curses 的应用程序通常会关闭按键自动上屏，目的是读取按键并只在特定情况下展示它们。这需要调用函数 [noecho\(\)](#)：

```
curses.noecho()
```

应用程序也会广泛地需要立即响应按键，而不需要按下回车键；这被称为“cbreak”模式，与通常的缓冲输入模式相对：

```
curses.cbreak()
```

终端通常会以多字节转义序列的形式返回特殊按键，比如光标键和导航键比如 Page Up 键和 Home 键。尽管你可以编写你的程序来应对这些序列，curses 能够代替你做到这件事，返回一个特殊值比如 [curses.KEY_LEFT](#)。为了让 curses 做这项工作，你需要启用 keypad 模式：

```
stdscr.keypad(True)
```

终止一个 curses 应用程序比建立一个容易得多，你只需要调用：

```
curses.nocbreak()
stdscr.keypad(False)
curses.echo()
```

来还原对终端作出的 curses 友好设置。然后，调用函数 [endwin\(\)](#) 来将终端还原到它的原始操作模式：

```
curses.endwin()
```

调试一个 curses 应用程序时常会发生，一个应用程序还未能还原终端到原本的状态就意外退出了，这会搅乱你的终端。在 Python 中这常常会发生你的代码中有 bug 并引发了一个未捕获的异常。当你尝试输入时按键不会上屏，这使得使用终端变得困难。

在 Python 中你可以避免这些复杂问题并让调试变得更简单，只需要导入 [curses.wrapper\(\)](#) 函数并像这样使用它：

```
from curses import wrapper

def main(stdscr):
    # 清空屏幕
```

```

stdscr.clear()

# 当 i == 10 时这将引发 ZeroDivisionError。
for i in range(0, 11):
    v = i-10
    stdscr.addstr(i, 0, '10 divided by {} is {}'.format(v, 10/v))

    stdscr.refresh()
    stdscr.getkey()

wrapper(main)

```

[wrapper\(\)](#) 函数接受一个可调用对象并进行上述的初始化过程，如果终端支持彩色还会初始化颜色。接下来 [wrapper\(\)](#) 会运行你提供的可调用对象。当该可调用对象返回时，[wrapper\(\)](#) 将恢复终端的初始状态。该可调用对象会在 [try...except](#) 再被调用以捕获异常，恢复终端状态，然后重新引发该异常。这样你的终端将不会在发生异常时处于不正常状态，你将能够读取异常的消息和回溯。

窗口和面板

窗口是 `curses` 中的基本抽象。一个窗口对象表示了屏幕上的一个矩形区域，并且提供方法来显示文本、擦除文本、允许用户输入字符串等等。

函数 [initscr\(\)](#) 返回的 `stdscr` 对象覆盖整个屏幕。许多程序可能只需要这一个窗口，但你可能希望把屏幕分割为多个更小的窗口，来分别重绘或者清除它们。函数 [newwin\(\)](#) 根据给定的尺寸创建一个新窗口，并返回这个新的窗口对象：

```

begin_x = 20; begin_y = 7
height = 5; width = 40
win = curses.newwin(height, width, begin_y, begin_x)

```

注意 `curses` 使用的坐标系统与寻常的不同。坐标始终是以 y,x 的顺序传递，并且左上角是坐标 (0,0)。这打破了正常的坐标处理约定，即 x 坐标在前。这是一个与其他计算机应用程序糟糕的差异，但这从 `curses` 最初被编写出来就已是它的一部分，现在想要修改它已为时已晚。

你的应用程序能够查明屏幕的尺寸，[curses.LINES](#) 和 [curses.COLS](#) 分别代表了 y 和 x 方向上的尺寸。合理的坐标应位于 (0,0) 到 ([curses.LINES](#) - 1, [curses.COLS](#) - 1) 范围内。

当你调用一个方法来显示或擦除文本时，效果并不会立即显示。相反，你必须调用窗口对象的 [refresh\(\)](#) 方法来更新屏幕。

这是因为 `curses` 最初是针对 300 波特的龟速终端连接编写的；在这些终端上，减少重绘屏幕的时间非常重要。相应地当你调用 [refresh\(\)](#) 时 `curses` 会累积对屏幕的修改并以最高效的方式显示它们。打个比方，如果你的程序在某个窗口内显示一些文本然后清空了该窗口，那么就没有必要发送这些原始文本因为它们从来都不可见。

在实践中，显式地告诉 `curses` 重绘一个窗口并不会真的让 `curses` 复杂多少。大部分程序会进行一系列活动，然后暂停并等待按键或者用户方的其他动作。你要做的事情就是保证屏幕在暂停并等待用户输入之前已被重绘，具体方式是首先调用 `stdscr.refresh()` 或其他相关窗口的 [refresh\(\)](#) 方法。

一个面板是一种特殊的窗口，它可以比实际的显示屏幕更大，并且能只显示它的一部分。创建面板需要指定面板的高度和宽度，但刷新一个面板需要给出屏幕坐标和面板的需要显示的局部。

```
pad = curses.newpad(100, 100)
# 这个循环使用字母填充 pad; addch() 函数将在下一部分解释
for y in range(0, 99):
    for x in range(0, 99):
        pad.addch(y,x, ord('a') + (x*x+y*y) % 26)

# 在屏幕中央显示 pad 的某个区域。
# (0,0) : 要显示的 pad 区域的左上角坐标。
# (5,5) : 要填充到窗口的左上角坐标。
# (20, 75) : 填充到窗口的右下角坐标。
pad.refresh( 0,0, 5,5, 20,75)
```

此 refresh() 调用位在屏幕坐标 (5,5) 到坐标 (20,75) 的矩形范围内显示面板的一个部分；被显示部分在面板上的坐标是 (0,0)。除了上述差异，面板非常像是普通窗口并支持相同的方法。

如果你在屏幕上有多个窗口和面板那么有个更高效的方式来更新屏幕并防止屏幕的每部分被更新时出现烦人的屏幕闪烁。refresh() 实际上做了两件事：

1. 调用每个窗口的 [noutrefresh\(\)](#) 方法来更新一个表达屏幕期望状态的底层的数据结构。
2. 调用函数 [doupdate\(\)](#) 来改变物理屏幕来符合这个数据结构中记录的期望状态。

你可以改为在多个窗口上调用 noutrefresh() 来更新该数据结构，然后调用 doupdate() 来更新屏幕。

显示文字

从一名 C 程序员的视角来看，curses 有时看起来就像是一堆函数组成的迷宫，每个都有细微的差异。举个例子，addstr() 是在 stdscr 窗口的当前光标位置显示一个字符串，而 mvaddstr() 则是在显示字符串之前先移动到给定的 y,x 坐标。waddstr() 与 addstr() 很像，但允许指定一个要使用的窗口而不是默认使用 stdscr。mvwaddstr() 允许同时指定一个窗口和一个坐标。

幸运的是，Python 接口隐藏了所有这些细节。stdscr 和其他任何窗口一样是一个窗口对象，并且诸如 [addstr\(\)](#) 之类的方法接受多种参数形式。通常有四种形式。

形式	描述
str 或 ch	在当前位置显示字符串 str 或字符 ch
str 或 ch, attr	在当前位置使用 attr 属性显示字符串 str 或字符 ch
y, x, str 或 ch	移动到窗口内的 y,x 位置，并显示 str 或 ch
y, x, str 或 ch, attr	移至窗口内的 y,x 位置，并使用 attr 属性显示 str 或 ch

属性允许以突出显示形态显示文本，比如加粗、下划线、反相或添加颜色。这些属性将来下一小节细说。

`addstr()` 方法接受一个 Python 字符串或字节串作为要显示的值。字节串的内容会被原样发送到终端。字符串会使用窗口的 `encoding` 属性值指定的编码格式编码为字节串；该值默认为 `locale.getencoding()` 所返回的系统编码格式。

方法 `addch()` 接受一个字符，可以是长度为 1 的字符串，长度为 1 的字节串或者一个整数。

对于特殊扩展字符有一些常量，这些常量是大于 255 的整数。比如，`ACS_PLMINUS` 是一个“加减”符号，`ACS_ULCORNER` 是一个框的左上角（方便绘制边界）。你也可以使用正确的 Unicode 字符。

窗口会记住上次操作之后光标所在位置，所以如果你忽略 y,x 坐标，字符串和字符会出现在上次操作结束的位置。你也可以通过 `move(y,x)` 的方法来移动光标。因为一些终端始终会显示一个闪烁的光标，你可能会想要保证光标处于一些不会让人感到分心的位置。在看似随机的位置出现一个闪烁的光标会令人非常迷惑。

如果你的应用程序完全不需要一个闪烁的光标，你可以调用 `curs_set(False)` 来使它隐形。为与旧版本 `curses` 的兼容性的关系，有函数 `leaveok(bool)` 作为 `curs_set()` 的等价替换。如果 `bool` 是真值，`curses` 库会尝试移除闪烁光标，并且你也不必担心它会留在一些奇怪的位置。

属性和颜色

字符可以以不同的方式显示。基于文本的应用程序常常以反相显示状态行，一个文本查看器可能需要突出显示某些单词。为了支持这种用法，`curses` 允许你为屏幕上的每个单元指定一个属性值。

属性值是一个整数，它的每一个二进制位代表一个不同的属性。你可以尝试以多种不属性位组合来显示文本，但 `curses` 不保证所有的组合都是有效的，或者看上去有明显不同。这一点取决于用户终端的能力，所以最稳妥的方式是只采用最常见的有效属性，见下表。

属性	描述
<code>A_BLINK</code>	闪烁文本
<code>A_BOLD</code>	超亮或粗体文本
<code>A_DIM</code>	半明亮文本
<code>A_REVERSE</code>	反相显示文本
<code>A_STANDOUT</code>	可用的最佳突出显示模式
<code>A_UNDERLINE</code>	带下划线的文本

所以，为了在屏幕顶部显示一个反相的状态行，你可以这么编写：

```
stdscr.addstr(0, 0, "Current mode: Typing mode",
               curses.A_REVERSE)
stdscr.refresh()
```

`curses` 库还支持在提供了颜色功能的终端上显示颜色的功能。最常见的提供颜色的终端很可能是 Linux 控制台，采用了 `xterms` 配色方案。

为了使用颜色，你必须在调用完函数 `initscr()` 后尽快调用函数 `start_color()`，来初始化默认颜色集 (`curses.wrapper()` 函数自动完成了这一点)。当它完成后，如果使用中的终端支持显示颜色，`has_colors()` 会返回真值。（注意：`curses` 使用美式拼写 “color”，而不是英式 / 加拿大拼写 “colour”。如果你习惯了英式拼写，你需要避免自己在这些函数上拼写错误。）

`curses` 库维护一个有限数量的颜色对，包括一个前景（文本）色和一个背景色。你可以使用函数 `color_pair()` 获得一个颜色对对应的属性值。它可以通过按位或运算与其他属性，比如 `A_REVERSE` 组合。但再说明一遍，这种组合并不保证在所有终端上都有效。

一个样例，用 1 号颜色对显示一行文本：

```
stdscr.addstr("Pretty text", curses.color_pair(1))
stdscr.refresh()
```

如前所述，颜色对由前景色和背景色组成。`init_pair(n, f, b)` 函数可改变颜色对 n 的定义为前景色 f 和背景色 b 。颜色对 0 硬编码为黑底白字，不能改变。

颜色已经被编号，并且当其激活 color 模式时 `start_color()` 会初始化 8 种基本颜色。它们是：0:black, 1:red, 2:green, 3:yellow, 4:blue, 5:magenta, 6:cyan 和 7:white。`curses` 模块为这些颜色定义了相应的名称常量：`curses.COLOR_BLACK`, `curses.COLOR_RED` 等等。

让我们来做个综合练习。要将颜色 1 改为红色文本白色背景，你应当调用：

```
curses.init_pair(1, curses.COLOR_RED, curses.COLOR_WHITE)
```

当你改变一个颜色对时，任何已经使用该颜色对来显示的文本将会更改为新的颜色。你还可以这样来显示新颜色的文本：

```
stdscr.addstr(0,0, "RED ALERT!", curses.color_pair(1))
```

某些非常花哨的终端可以将实际颜色定义修改为给定的 RGB 值。这允许你将通常为红色的 1 号颜色改成紫色或蓝色或者任何你喜欢的颜色。不幸的是，Linux 控制台不支持此特性，所以我无法尝试它，也无法提供任何示例。想要检查你的终端是否能做到你可以调用 `can_change_color()`，如果有此功能则它将返回 `True`。如果你幸运地拥有一个如此优秀的终端，请查询你的系统的帮助页面来了了解详情。

用户输入

C `curses` 库只提供了非常简单的输入机制。Python 的 `curses` 模块增加了一个基本的文本输入控件。（其他的库如 `Urwid` 拥有更丰富的控件集。）

有两个方法可以从窗口获取输入：

- `getch()` 会刷新屏幕然后等待用户按键，如果之前调用过 `echo()` 还会显示所按的键。你还可以选择指定一个坐标以便在暂停之前让光标移动到哪里。
- `getkey()` 将做同样的事但是会把整数转换为字符串。每个字符将返回为长度为 1 个字符的字符串，特殊键例如函数键将返回包含键名的较长字符串例如 `KEY_UP` 或 `^G`。

使用 `nodelay()` 窗口方法可以不等待用户操作。在 `nodelay(True)` 之后，窗口的 `getch()` 和 `getkey()` 将成为非阻塞的。为表明输入未就绪，`getch()` 会返回 `curses.ERR` (值为 -1) 并且 `getkey()` 会引发异常。此外还有 `halfdelay()` 函数，它可被用来 (实际地) 在每个 `getch()` 上设置一个计时器；如果在指定的延迟内 (以十分之一秒为单位) 输入还不可用，`curses` 将引发异常。

`getch()` 方法返回一个整数；如果其值在 0 到 255 之间，它代表所按的键的 ASCII 码。大于 255 的值为特殊键例如 Page Up, Home 或方向键等。你可以将返回的值与 `curses.KEY_PPAGE`, `curses.KEY_HOME` 或 `curses.KEY_LEFT` 等常量做比较。你的程序的主循环看起来可能会像这样:

```
while True:
    c = stdscr.getch()
    if c == ord('p'):
        PrintDocument()
    elif c == ord('q'):
        break # 退出 while 循环
    elif c == curses.KEY_HOME:
        x = y = 0
```

`curses.ascii` 模块提供了一些 ASCII 类成员函数，它们接受整数或长度为 1 个字符的字符串参数；这些函数在为这样的循环编写更具可读性的测试时可能会很有用。它还提供了一些转换函数，它们接受整数或长度为 1 个字符的字符串参数并返回同样的类型。例如，`curses.ascii.ctrl()` 返回与其参数相对应的控制字符。

还有一个可以提取整个字符串的方法 `getstr()`。它并不经常被使用，因为它的功能相当受限；可用的编辑键只有 Backspace 和 Enter 键，它们会结束字符串。也可以选择限制为固定数量的字符。

```
curses.echo()           # 启用字符回显

# 获取一个 15 个字符的字符串，光标位于顶端行
s = stdscr.getstr(0,0, 15)
```

`curses.textpad` 模块提供了一个文本框，它支持类似 Emacs 的键绑定集。`Textbox` 类的各种方法支持带输入验证的编辑及包含或不包含末尾空格地收集编辑结果。下面是一个例子:

```
import curses
from curses.textpad import Textbox, rectangle

def main(stdscr):
    stdscr.addstr(0, 0, "Enter IM message: (hit Ctrl-G to send)")

    editwin = curses.newwin(5,30, 2,1)
    rectangle(stdscr, 1,0, 1+5+1, 1+30+1)
    stdscr.refresh()

    box = Textbox(editwin)

    # 让用户编辑直到按下 Ctrl-G。
    box.edit()

    # 获取结果内容
    message = box.gather()
```

请查看 [curses.textpad](#) 的库文档了解更多细节。

更多的信息

本 HOWTO 没有涵盖一些进阶主题，例如读取屏幕的内容或从 xterm 实例捕获鼠标事件等，但是 [curses](#) 模块的 Python 库文档页面现在已相当完善。接下来你应当去浏览一下其中的内容。

如果你对 curses 函数的细节行为有疑问，请查看你的 curses 具体实现的指南页面不论它是 ncurses 还是特定 Unix 厂商的版本。指南页面将写明各种怪异问题，并为你提供所有函数、属性及可用 [ACS](#) * 字符的完整列表。

由于 curses API 是如此的庞大，某些函数并不被 Python 接口所支持。这往往不是因为它们难以实现，而是因为还没有人需要它们。此外，Python 尚不支持与 ncurses 相关联的菜单库。欢迎提供添加这些功能的补丁；请参阅 [Python 开发者指南](#) 了解有关为 Python 提交补丁的详情。

- [Writing Programs with NCURSES](#): 一个面向 C 程序员的详细教程。
- [ncurses 手册主页](#)
- [ncurses 常见问题](#)
- ["使用 curses... 请勿爆粗"](#): 一场有关使用 curses 或 Urwid 来控制终端的 PyCon 2013 演讲的视频。
- ["使用 Urwid 的控制台应用程序"](#): 一场演示使用 Urwid 编写应用程序的 PyCon CA 2012 演讲的视频。