用 Python 进行 Curses 编程

發🄵 *3.8.3*

Guido van Rossum
and the Python development team

6 月 30, 2020

Python Software Foundation
Email: docs@python.org

Contents

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。

1 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---以及 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 并没有提供诸多用户界面概念，例如按钮，复选框或对话框。如果需要这些功能，请考虑用户界面库，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。

Windows 版本的 Python 不包含 `curses` 模块。提供了一个名为 UniCurses 的移植版本。也可以尝试使用 Fredrik Lundh 编写 the Console module，它使用了与 curses 不相同的 API，但提供了可光标定位的文本输出，完全支持鼠标和键盘输入。

1.1 Python 的 curses 模块

此 Python 模块相当简单地封装了 curses 提供的 C 函数；如果你已经熟悉在 C 语言中使用 curses 编程，把这些知识转移的 Python 是非常容易的。最大的差异在于 Python 中的接口通过把不同的 C 函数合并来让事情变得更简单，比如 `addstr()`、`mvaddstr()` 和 `mvwaddstr()` 三个 C 函数被并入 `addstr()` 这一个方法。下文中会描述更多的细节。

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。

2 开始和结束 curses 应用程序

在做任何事情之前，curses 必须先被初始化。可以通过调用函数 `initscr()` 来实现，它将查明终端的类型，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。如果此操作成功，`initscr()` 将会返回一个代表整个屏幕的窗口对象；它通常会遵循对应的 C 变量名被称作 `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 中这常常会发生在你的代码中有 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):
    # Clear screen
    stdscr.clear()

    # This raises ZeroDivisionError when i == 10.
    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)
```

函数 `wrapper()` 接受一个可调用对象并首先进行上述初始化过程，在终端支持着色时还会初始化颜色。接着 `wrapper()` 运行你提供的可调用对象。当该可调用对象返回时，`wrapper()` 会还原终端到初始状态。该可调用对象会在 `try...except` 这样的结构内被调用，当它捕获到异常时，会先还原终端再重新引发这个异常。所以你的终端不会因为异常而被留在一个搞笑的状态，你也可以正常阅读异常消息和回溯信息。

3 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，来分别重绘或者清楚它们。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 最初是为 300 波特的龟速终端连接编写的；在这些终端上，压制重绘屏幕的时间就非常重要。相对地，当你调用 `refresh()` 时，curses 会累积屏幕的修改并以效率最高的方式显示它们。打个比方，如果你的程序在一个窗口内显示一些文本然后清楚了这个窗口，那么这些原始文本不需要被发送，因为它们甚至不曾能被看见。

在实践中，显式地告诉 curses 来重绘一个窗口并不会太复杂化 curses 编程。大部分程序会显示一堆内容然后等待按键或者其他某些用户侧动作。你要做的事情就是，保证屏幕在暂停并等待用户输入前被重绘，只需要先调用 `stdscr.refresh()` 或者其他相关窗口的 `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)
# These loops fill the pad with letters; addch() is
# explained in the next section
for y in range(0, 99):
    for x in range(0, 99):
        pad.addch(y,x, ord('a') + (x*x+y*y) % 26)

# Displays a section of the pad in the middle of the screen.
# (0,0) : coordinate of upper-left corner of pad area to display.
# (5,5) : coordinate of upper-left corner of window area to be filled
#         with pad content.
# (20, 75) : coordinate of lower-right corner of window area to be
#          : filled with pad content.
pad.refresh( 0,0, 5,5, 20,75)
```

此 `refresh()` 调用会在屏幕坐标 (5,5) 到坐标 (20,75) 的矩形范围内显示面板的一个部分，被显示的部分在面板上的坐标是 (0,0)。除了上述差异，面板与常规的窗口相同，也支持相同的方法。

如果你在屏幕上有多个窗口和面板，有一个更有效率的方法来更新窗口，避免每个部分单独更新时烦人的屏幕闪烁。`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。

你可以改为调用在多个窗口上 noutrefresh() 方法来更新该数据结构，然后调用函数 doupdate() 来更新屏幕。

4 显示文字

从一名 C 语言程序员的视角来看，curses 有时看起来就像是一堆略有差异的函数组成的扭曲迷宫。举个例子，addstr() 在 stdscr 窗口的当前光标位置显示一个字符串，而 mvaddstr() 则是先移动到一个给定的 y,x 坐标再显示字符串。waddstr() 与 addstr() 类似，但允许指定一个窗口而非默认的 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。

方法 addstr() 接受一个 Python 字符串或字节串作为用于显示的值。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 属性值编码为字节，它默认为 locale.getpreferredencoding() 返回的系统默认编码。

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。

4.1 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
A_BLINK	闪烁文字
A_BOLD	超亮或粗体文字
A_DIM	半明亮的文字
A_REVERSE	反相显示文本
A_STANDOUT	可用的最佳突出显示模式
A_UNDERLINE	带下划线的文字

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()
```

As I said before, a color pair consists of a foreground and background color. The `init_pair(n, f, b)` function changes the definition of color pair *n*, to foreground color f and background color b. Color pair 0 is hard-wired to white on black, and cannot be changed.

Colors are numbered, and `start_color()` initializes 8 basic colors when it activates color mode. They are: 0:black, 1:red, 2:green, 3:yellow, 4:blue, 5:magenta, 6:cyan, and 7:white. The `curses` module defines named constants for each of these colors: `curses.COLOR_BLACK`, `curses.COLOR_RED`, and so forth.

Let's put all this together. To change color 1 to red text on a white background, you would call:

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

When you change a color pair, any text already displayed using that color pair will change to the new colors. You can also display new text in this color with:

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

Very fancy terminals can change the definitions of the actual colors to a given RGB value. This lets you change color 1, which is usually red, to purple or blue or any other color you like. Unfortunately, the Linux console doesn't support this, so I'm unable to try it out, and can't provide any examples. You can check if your terminal can do this by calling `can_change_color()`, which returns `True` if the capability is there. If you're lucky enough to have such a talented terminal, consult your system's man pages for more information.

5 用户输入

The C curses library offers only very simple input mechanisms. Python's `curses` module adds a basic text-input widget. (Other libraries such as Urwid have more extensive collections of widgets.)

There are two methods for getting input from a window:

- `getch()` refreshes the screen and then waits for the user to hit a key, displaying the key if `echo()` has been called earlier. You can optionally specify a coordinate to which the cursor should be moved before pausing.

- `getkey()` does the same thing but converts the integer to a string. Individual characters are returned as 1-character strings, and special keys such as function keys return longer strings containing a key name such as `KEY_UP` or `^G`.

It's possible to not wait for the user using the `nodelay()` window method. After `nodelay(True)`, `getch()` and `getkey()` for the window become non-blocking. To signal that no input is ready, `getch()` returns `curses.ERR` (a value of -1) and `getkey()` raises an exception. There's also a `halfdelay()` function, which can be used to (in effect) set a timer on each `getch()`; if no input becomes available within a specified delay (measured in tenths of a second), curses raises an exception.

The `getch()` method returns an integer; if it's between 0 and 255, it represents the ASCII code of the key pressed. Values greater than 255 are special keys such as Page Up, Home, or the cursor keys. You can compare the value returned to constants such as `curses.KEY_PPAGE`, `curses.KEY_HOME`, or `curses.KEY_LEFT`. The main loop of your program may look something like this:

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

The `curses.ascii` module supplies ASCII class membership functions that take either integer or 1-character string arguments; these may be useful in writing more readable tests for such loops. It also supplies conversion functions that take either integer or 1-character-string arguments and return the same type. For example, `curses.ascii.ctrl()` returns the control character corresponding to its argument.

There's also a method to retrieve an entire string, `getstr()`. It isn't used very often, because its functionality is quite limited; the only editing keys available are the backspace key and the Enter key, which terminates the string. It can optionally be limited to a fixed number of characters.

```
curses.echo()             # Enable echoing of characters

# Get a 15-character string, with the cursor on the top line
s = stdscr.getstr(0,0, 15)
```

The `curses.textpad` module supplies a text box that supports an Emacs-like set of keybindings. Various methods of the `Textbox` class support editing with input validation and gathering the edit results either with or without trailing spaces. Here's an example:

```
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)

    # Let the user edit until Ctrl-G is struck.
    box.edit()
```

(下页继续)

(繼續上一頁)

```
    # Get resulting contents
    message = box.gather()
```

See the library documentation on `curses.textpad` for more details.

6 更多的信息

This HOWTO doesn't cover some advanced topics, such as reading the contents of the screen or capturing mouse events from an xterm instance, but the Python library page for the `curses` module is now reasonably complete. You should browse it next.

If you're in doubt about the detailed behavior of the curses functions, consult the manual pages for your curses implementation, whether it's ncurses or a proprietary Unix vendor's. The manual pages will document any quirks, and provide complete lists of all the functions, attributes, and `ACS_*` characters available to you.

Because the curses API is so large, some functions aren't supported in the Python interface. Often this isn't because they're difficult to implement, but because no one has needed them yet. Also, Python doesn't yet support the menu library associated with ncurses. Patches adding support for these would be welcome; see the Python Developer's Guide to learn more about submitting patches to Python.

- Writing Programs with NCURSES: a lengthy tutorial for C programmers.
- ncurses 手册主页 <https://linux.die.net/man/3/ncurses>‘_
- ncurses 常见问题 <http://invisible-island.net/ncurses/ncurses.faq.html>‘_
- "Use curses... don't swear": video of a PyCon 2013 talk on controlling terminals using curses or Urwid.
- "Console Applications with Urwid": video of a PyCon CA 2012 talk demonstrating some applications written using Urwid.