幻尔科技

Jetson Nano 版本开发教程

V1.0

Hiwonder 官方网站

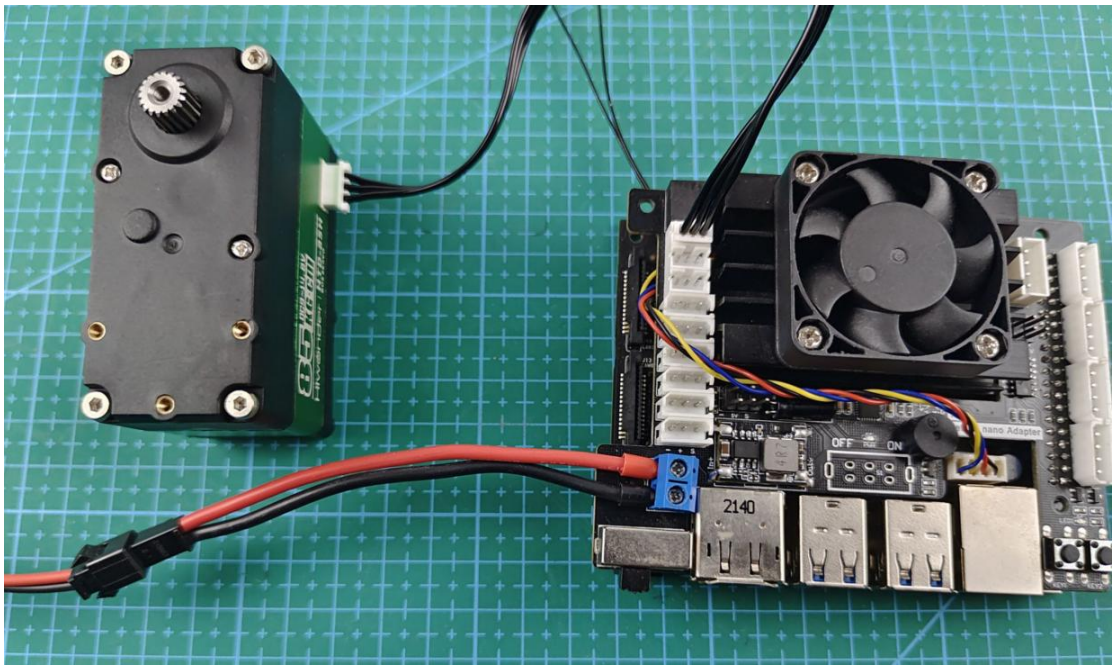
版本号	修改日期	修改摘要
V1.0	20230923	初次发布

目录

1.准备工作

1.1 接线说明

本节示例使用的是 Jetson Nano 主板和 Jetson 扩展板，通过 11.1V 6000mAh 锂电池来供电。将总线舵机连接至 Jetson 扩展板任意一个总线接口。



注意：

① 如使用我司锂电池，连接锂电池对接线请以红接＋，黑接-接到 DC 接口。

② 如对接线未连接锂电池，请勿与电池对接线直接对接，避免正负极发生短路从而造成短路。

1.2 环境配置

在电脑端安装 NoMachine，软件包位于“**2 软件工具->远程桌面连接工具**”下。关于 NoMachine 的详细使用，可在对应目录下进行学习。

将程序和库文件 SDK，拖动到 Jetson 系统镜像内，这里以放置在桌面为例进行。**注意：库文件要同程序放在同一目录下。**

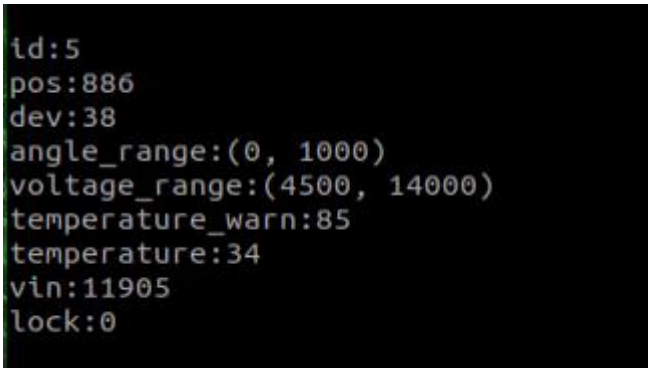
打开命令行终端，输入给文件添加执行权限的指令“**chmod a+x serial_servo sdk/**”。

```
hiwonder@hiwonder:~/Desktop$ chmod a+x serial_servo sdk/
hiwonder@hiwonder:~/Desktop$
```

2.案例开发

2.1 案例 1 总线舵机信息读取

本案例通过终端窗口显示出总线舵机 ID、位置、温度等相关信息。



2.1.1 运行程序

(1) 打开终端，输入切换到程序所在目录的指令：

“**cd Desktop/serial_servo/**”，按下回车。

```
hiwonder@hiwonder:~$ cd Desktop/serial_servo/
hiwonder@hiwonder:~/Desktop/serial_servo$
```

(2) 输入执行本案例程序的指令：

“**python3 serial_servo_status.py**”。

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_status.py
```

2.1.2 实现效果

程序运行后，终端打印画面会滚屏打印舵机各项状态信息。

```
id:5
pos:886
dev:38
angle_range:(0, 1000)
voltage_range:(4500, 14000)
temperature_warn:85
temperature:34
vin:11863
lock:0
```

各项信息的具体含义如下：

① id:舵机 ID。在这里示例为 5。

② pos：舵机当前所在的位置。在这里示例为 886。

③ dev：舵机偏差。在这里示例为 38。

④ angle range：舵机限位范围。在这里示例为 0-1000。

⑤ voltage range：舵机电压范围。在这里示例为 4.5~14V。

⑥ temperature warn：舵机过温报警阈值。在这里示例为 85℃。

⑦ temperature：舵机当前温度。在这里示例为 34℃。

⑧ vin：舵机当前电压值。在这里示例为 11.863V。

⑨ lock：舵机是否为上电状态。在这里示例为 0，代表为掉电状态。当为 1，即上电状态。

2.1.3 案例程序简要分析

- **导入必要的模块**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

首先，导入 sys 和 time 模块，以及 hiwonder_servo_controller 模块。通过导入

该模块，我们可以使用其中定义的函数和类来控制舵机。

- **创建舵机控制对象**

```
servo_control =
hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1',
115200)
```

通过实例化 HiwonderServoController 类创建一个 servo_control 对象，传入的参数是串行舵机的设备路径和波特率。

- **获取舵机状态信息并打印**

```
            servo_id = servo_control.get_servo_id()
            pos = servo_control.get_servo_position(servo_id)
            dev = servo_control.get_servo_deviation(servo_id)
            if dev > 125:
                dev = -(0xff - (dev - 1))
            angle_range = servo_control.get_servo_range(servo_id)
            vin_range = servo_control.get_servo_vin_range(servo_id)
            temperature_warn =
servo_control.get_servo_temp_range(servo_id)
            temperature = servo_control.get_servo_temp(servo_id)
            vin = servo_control.get_servo_vin(servo_id)
            load_state = servo_control.get_servo_load_state(servo_id)
```

进入循环函数后，通过调用 servo_control 对象的方法，获取舵机的各种状态信息。这些状态信息包括舵机 ID、位置、偏差、角度范围、电压范围、过温报警阈值、当前温度、电压以及舵机上电状态。

在获取后，可将上面的这些参数打印出来。

```
            print('id:%s'%(str(servo_id).ljust(3)))
            print('pos:%s'%(str(pos).ljust(4)))
            print('dev:%s'%(str(dev).ljust(4)))
```

```
            print('angle_range:%s'%(str(angle_range).ljust(4)))
            print('voltage_range:%s'%(str(vin_range).ljust(5)))
            print('temperature_warn:%s'%(str(temperature_warn).ljust(4))
)
            print('temperature:%s'%(str(temperature).ljust(4)))
            print('vin:%s'%(str(vin).ljust(4)))
            print('lock:%s'%(str(load_state).ljust(4)))
            print('')
```

将获取到的舵机状态信息打印出来。每条打印语句使用字符串格式化和 ljust()方法，以对齐输出结果。

- **设置中断标志位**

```
        except KeyboardInterrupt:
            break
```

最后设置按键，当终端界面打印时，按下 Ctrl+C，捕获 KeyboardInterrupt 异常，跳出循环，结束程序的执行。

2.2 案例 2 总线舵机 ID 设置

2.2.1 运行程序

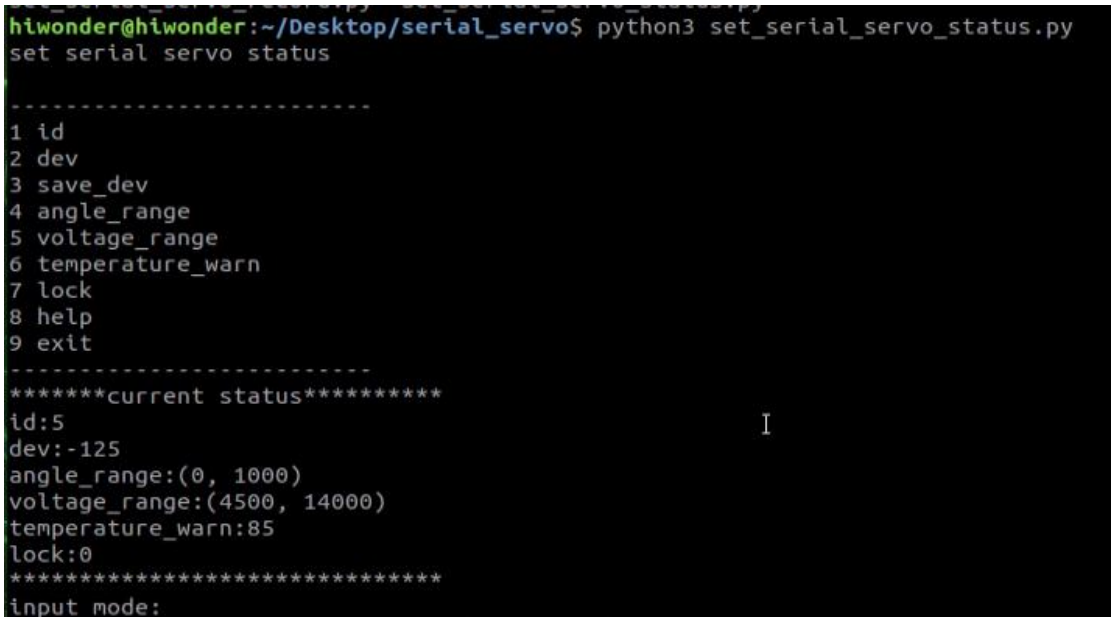
(1) 打开终端，输入切换到程序所在目录的指令：

“**cd Desktop/serial_servo/**”，按下回车。

```
hiwonder@hiwonder:~$ cd Desktop/serial_servo/
hiwonder@hiwonder:~/Desktop/serial_servo$
```

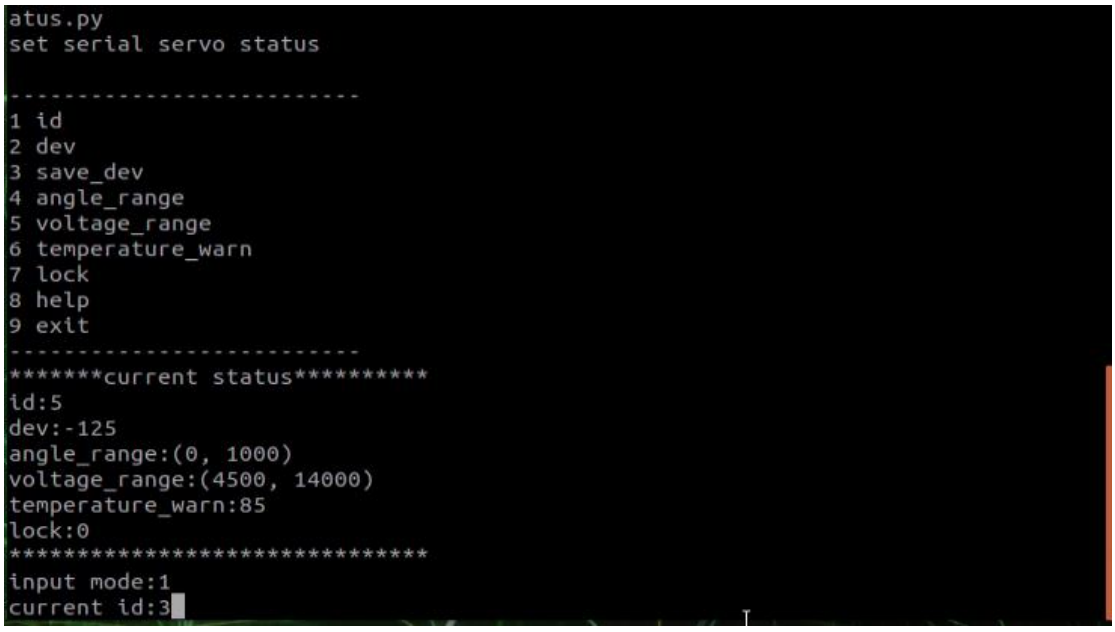
(2) 输入设置总线舵机 ID 的指令：

“**python3 set_serial_servo_status.py** ”。



（3）在指令提示框，先输入“1”，进入 ID 设置模式，然后输入 ID 号，程序里读取当前舵机 ID 为 5，那么这里以更改 3 为例，输入完成后按下回车即可。

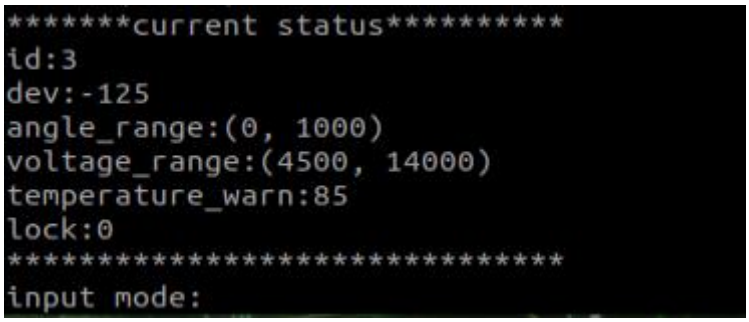
注意：ID 号设置范围为 0-253。



（4）如需退出设置，按下“Ctrl+C”即可。

2.2.2 实现效果

修改完成后，该回传界面将直接显示当前设置的 ID。



2.2.3 案例程序简要分析

- **导入必要的模块**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

首先，导入 sys 和 time 模块，以及 hiwonder_servo_controller 模块。通过导入该模块，我们可以使用其中定义的函数和类来控制舵机。

- **创建舵机控制对象**

```
servo_control =
hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1',
115200)
```

通过实例化 HiwonderServoController 类创建一个 servo_control 对象，传入的参数是串行舵机的设备路径和波特率。

- **进入主循环函数**

```
servo_control =
hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1',
115200)while True:
    try:
        get_status()
        mode = int(input('input mode:'))
        if mode == 1:
```

```
            oldid = int(input('current id:'))
            newid = int(input('new id(0-253):'))
            servo_control.set_servo_id(oldid, newid)
        elif mode == 2:

            servo_id = int(input('servo id:'))
            dev = int(input('deviation(-125~125):'))
            if dev < 0:
                dev = 0xff + dev + 1
                    servo_control.set_servo_deviation(servo_id, dev)
                                #跳过中间代码
        elif mode == 9:
            break
        else:
            print('error mode')
    except KeyboardInterrupt:
        break
```

在主循环中，程序首先调用 get_status()函数打印舵机的当前状态。然后，通过输入模式选择用户想要执行的操作。根据不同的模式，程序执行相应的代码块来实现舵机的设置和控制。

- **打印菜单选项**

```
print('''
--------------------------
1 id
2 dev
3 save_dev
4 angle_range
5 voltage_range
```

```
6 temperature_warn
7 lock
8 help
9 exit
--------------------------''')
```

用于打印菜单选项，提供了不同的操作模式供用户选择。

- **设置中断标志位**

```
        except KeyboardInterrupt:
            break
```

最后设置按键，当终端界面打印时，按下 Ctrl+C，捕获 KeyboardInterrupt 异常，跳出循环，结束程序的执行。

2.3 案例 3 控制总线舵机转动

2.3.1 运行程序

(1) 打开终端，输入切换到程序所在目录的指令：

“**cd Desktop/serial_servo/**”，按下回车。

```
hiwonder@hiwonder:~$ cd Desktop/serial_servo/
hiwonder@hiwonder:~/Desktop/serial_servo$
```

(2) 输入总线舵机转动的指令“**python3 serial_servo_move_demo.py** ”。

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_move_demo.py
serial servo move between 500 - 1000
```

2.3.2 实现效果

程序运行后，舵机会从位置 0-1000 以 1 秒为间隔来回转动。

2.3.3 案例程序简要分析

- **导入必要的模块**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

首先，导入 sys 和 time 模块，以及 hiwonder_servo_controller 模块。通过导入该模块，我们可以使用其中定义的函数和类来控制舵机。

- **创建舵机控制对象**

```
servo_control = hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1', 115200)
```

通过实例化 HiwonderServoController 类创建一个 servo_control 对象，传入的参数是串行舵机的设备路径和波特率。

- **进入主循环函数**

```
while True:
    try:
        servo_id = 6  # 舵机 ID (0-253)
        position = 500  # 位置 (0-1000)
        duration = 500  # 时间 (20-30000)
        servo_control.set_servo_position(servo_id, position, duration)
        time.sleep(duration/1000.0 + 0.1)  # 等待直到舵机运动完成，额外增加 100ms 的延迟
        servo_id = 1
        position = 1000
        duration = 500
        servo_control.set_servo_position(servo_id, position, duration)
```

```
        time.sleep(duration/1000.0 + 0.1)
    except KeyboardInterrupt:
        servo_id = 6
        position = 0
        duration = 500
        servo_control.set_servo_position(servo_id, position, duration)
        break
```

在主循环中，程序通过调用 set_servo_position()方法来设置舵机的位置和运行时间。首先，将舵机 ID 设置为 1，位置设置为 500，运行时间设置为 500ms，然后调用 set_servo_position()方法来控制舵机运动到指定位置。

接着，通过 time.sleep()方法等待舵机运动完成，为了确保舵机完全到达目标位置，额外增加了 100ms 的延迟。然后，将舵机位置设置为 1000，再次调用 set_servo_position()方法来控制舵机运动到新的位置。

- **设置中断标志位**

```
        except KeyboardInterrupt:
            break
```

最后设置按键，当终端界面打印时，按下 Ctrl+C，捕获 KeyboardInterrupt 异常，跳出循环，结束程序的执行。

2.4 案例 4 调节总线舵机速度

2.4.1 运行程序

(1) 打开终端，输入切换到程序所在目录的指令：

“**cd Desktop/serial_servo/**”，按下回车。

(2) 输入总线舵机转动的指令“**python3 serial_servo_speed_demo.py** ”。

```
hiwonder@hiwonder:~/Desktop/serial_servo$ python3 serial_servo_speed_demo.py
serial servo move at different speed
```

2.4.2 实现效果

程序运行后舵机现象如下：

① 舵机从位置 500 开始；

② 以 0.5 秒的时间转动到位置 1000；

③ 以 1.5 秒的时间转动到位置 500；

④ 以 2.5 秒的时间转动到位置 0；

⑤ 以 3.5 秒的时间转动到位置 500。

2.4.3 案例程序简要分析

- **导入必要的模块**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

首先，导入 sys 和 time 模块，以及 hiwonder_servo_controller 模块。通过导入该模块，我们可以使用其中定义的函数和类来控制舵机。

- **创建舵机控制对象**

```
servo_control =
hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1',
115200)
```

通过实例化 HiwonderServoController 类创建一个 servo_control 对象，传入的参数是串行舵机的设备路径和波特率。

- **控制舵机速度**

```
servo_id = 6  # 舵机 ID (0-253)
position = 500  # 位置 (0-1000)
duration = 1000  # 时间 (20-30000)
servo_control.set_servo_position(servo_id, position, duration)
time.sleep(duration/1000.0 + 0.1)

servo_id = 6
position = 1000
duration = 500
servo_control.set_servo_position(servo_id, position, duration)
time.sleep(duration/1000.0 + 0.1)

servo_id = 6  # 舵机 ID (0-253)
position = 500  # 位置 (0-1000)
duration = 1000  # 时间 (20-30000)
servo_control.set_servo_position(servo_id, position, duration)
time.sleep(duration/1000.0 + 0.1)

servo_id = 6
position = 0
duration = 500
servo_control.set_servo_position(servo_id, position, duration)
time.sleep(duration/1000.0 + 0.1)
```

首先，将舵机 ID 设置为 6，位置设置为 500，运行时间设置为 1000ms，然后调用 set_servo_position()方法来控制舵机运动到指定位置。

接着，通过 time.sleep()方法等待舵机运动完成，为了确保舵机完全到达目标位置，

额外增加了 100ms 的延迟。然后，将舵机位置设置为 1000，再次调用 set_servo_position()方法来控制舵机运动到新的位置。同样地，通过 time.sleep()方法等待舵机运动完成。接下来，将舵机位置设置为 500，再次调用 set_servo_position()方法来控制舵机运动到新的位置。

最后，将舵机位置设置为 0，再次调用 set_servo_position()方法来控制舵机运动到新的位置。在每次舵机运动之后，都通过 time.sleep()方法等待舵机运动完成。

- **设置中断标志位**

```
    except KeyboardInterrupt:
        break
```

最后设置按键，当终端界面打印时，按下 Ctrl+C，捕获 KeyboardInterrupt 异常，跳出循环，结束程序的执行。

2.5 案例 5 示教记录操作

2.5.1 运行程序

(1) 打开终端，输入切换到程序所在目录的指令：

“**cd Desktop/serial_servo/**”，按下回车。

(2) 输入总线舵机转动的指令“ **python3 set_serial_servo_record.py** ”。

2.5.2 实现效果

程序运行后，终端打印出舵机的 ID 编号，舵机回到 500 脉冲宽度的位置，打印出“**Start turning the servo**”信息后开始舵机开始掉电，接着我们可以用手掰动舵臂，之后舵机

会记录下当前的位置，返回 500 脉冲宽度的位置之后再回到刚刚我们用手掰动的位置。

2.5.3 案例程序简要分析

- **导入必要的模块**

```
import sys
import time
sys.path.append("..")
from sdk import hiwonder_servo_controller
```

首先，导入 sys 和 time 模块，以及 hiwonder_servo_controller 模块。通过导入该模块，我们可以使用其中定义的函数和类来控制舵机。

- **创建舵机控制对象**

```
servo_control = hiwonder_servo_controller.HiwonderServoController('/dev/ttyTHS1', 115200)
```

通过实例化 HiwonderServoController 类创建一个 servo_control 对象，传入的参数是串行舵机的设备路径和波特率。

- **获取舵机 ID 并打印**

```
servo_id = servo_control.get_servo_id()
print('id:',servo_id)
```

调用 get_servo_id()获取舵机的 ID，并将其打印出来。

- **控制舵机回到中位**

```
servo_control.set_servo_position(servo_id, 500, 500)
```

调用 set_servo_position()将舵机移动到位置 500 回中，并设置运行时间为 500ms。

- **舵机掉电设置**

```
servo_control.unload_servo(servo_id, False)
```

调用 unload_servo()将舵机掉电，以此可以手掰。

- **获取输入模式**

```
mode = int(input('input mode:'))
```

使用 input()函数获取用户输入的模式，并将其转换为整数型。

- **根据获取的模式来执行操作**

```
if mode == 0:
    servo_control.get_servo_position(servo_id)
    pos = servo_control.get_servo_position(servo_id)
    servo_control.set_servo_position(servo_id, 0, 500)
    time.sleep(3)
    servo_control.set_servo_position(servo_id, pos, 500)
```

如果模式为 0，首先调用 get_servo_position()获取舵机当前位置，然后将舵机位置设置为 0，并等待 3 秒，最后将舵机位置恢复为之前的位置。

这里用户同时可以根据需要修改舵机 ID。

- **设置中断标志位**

```
    except KeyboardInterrupt:
        break
```

最后设置按键，当终端界面打印时，按下 Ctrl+C，捕获 KeyboardInterrupt 异常，跳出循环，结束程序的执行。