

树莓派SDK的python调用说明与例程

文档编号：PY-N211

编写日期：2024年5月24日

编写人员：沈玉杰，深圳谱研互联科技有限公司

技术支持：沈玉杰，深圳谱研互联科技有限公司

 导读：本文介绍了如何在树莓派（arm64架构，ubuntu 64位系统）从近红外光谱仪或模块读取波长和强度值数据。第1章介绍了wrapper.py基本函数。第2章介绍了wrapper.py扩展包的生成与安装。第3章给出了调用示例main.py。第4章用于记录可能遇到的问题。

1 wrapper.py介绍

wrapper.py是已经封装好的python模块，内部开放了用于操作近红外光谱模块或光谱仪的几个必要函数。源码如下：

```
1 import ctypes
2
3 # 加载共享库, lib/libwrapper.so
4 libwrapper = ctypes.CDLL("lib/libwrapper.so")
5
6 # ----定义函数原型----
7 # libwrapper.so是由c语言封装的动态库, python调用库函数时, 需要先指定参数类型为ctypes.
8
9 # 连接光谱仪设备。返回值是已连接设备数量。
10 libwrapper.dlpConnect.argtypes = []
11 libwrapper.dlpConnect.restype = ctypes.c_int
12
13 # 打开已连接的光谱仪设备。参数是c_int型的配置索引；返回值<0打开失败，返回值>=0打开成功。
14 libwrapper.dlpOpenByUsb.argtypes = [ctypes.c_int]
15 libwrapper.dlpOpenByUsb.restype = ctypes.c_int
16
17 # 获取波长。参数是c_double型的指针 (POINTER) 和c_int型的波长个数；返回值<0打开失败，返
18 # 回值>=0打开成功。
19 libwrapper.dlpGetWavelengths.argtypes = [ctypes.POINTER(ctypes.c_double),
20                                         ctypes.c_int]
21 libwrapper.dlpGetWavelengths.restype = ctypes.c_int
```

```

21 # 获取强度值。参数是c_int型的指针 (POINTER) 和c_int型的数据点数；返回值<0打开失败，返回
    值>=0打开成功。
22 libwrapper.dlpGetIntensities.argtypes = [ctypes.c_int,
    ctypes.POINTER(ctypes.c_int), ctypes.c_int]
23 libwrapper.dlpGetIntensities.restype = ctypes.c_int
24
25 # ----/定义函数原型----
26
27 # ---调用函数---
28 # 连接光谱仪设备。返回值是已连接设备数量。
29 def dlpConnect():
30     return libwrapper.dlpConnect()
31
32 # 打开已连接的光谱仪设备。参数是c_int型的配置索引；返回值<0打开失败，返回值>=0打开成功。
33 def dlpOpenByUsb(index):
34     return libwrapper.dlpOpenByUsb(index)
35
36 # 获取波长。参数是c_double型的指针 (POINTER) 和c_int型的波长个数；返回值<0打开失败，返
    回值>=0打开成功。
37 def dlpGetWavelengths(wls, wlsNum):
38     return libwrapper.dlpGetWavelengths(wls, wlsNum)
39
40 # 获取强度值。参数是c_int型的指针 (POINTER) 和c_int型的数据点数；返回值<0打开失败，返回
    值>=0打开成功。
41 def dlpGetIntensities(activeIndex, intensities, wlsNum):
42     return libwrapper.dlpGetIntensities(activeIndex, intensities, wlsNum)
43
44 # ----/调用函数---

```

libwrapper.so是由c语言封装的动态库，python调用库函数时，需要先指定参数类型为ctypes。python通过导入ctypes模块即可指定库函数类型。

```
1 import ctypes
```

用户只需要调用4个必要函数，即可完成从光谱仪获取数据。由于波长是固定的，只在开始获取一次即可。函数名称和功能，在调用函数中有详细介绍，不再赘述。

2 模块安装

树莓派ubuntu 64位系统用户，拿到3个文件：

- 1) wrapper.py：封装好的库函数文件
- 2) setup.py：python扩展文件安装文件

3) lib/libwrapper.so: 库函数

4) main.py: 调用例程

新建文件夹, 如 `python_demo_v1.1`, 并给该文件夹设置读写权限, 把以上4个文件复制到相同目录。

/home/pi/codes/python_demo_v1.1/

名字	大小	已改变	权限	拥有者
..		2024/5/24 17:50:15	rwxr-xr-x	pi
lib		2024/5/25 10:51:33	rwxr-xr-x	pi
main.py	2 KB	2024/5/25 10:53:16	rw-r--r--	pi
setup.py	1 KB	2024/5/24 21:59:44	rw-r--r--	pi
wrapper.py	3 KB	2024/5/25 10:52:44	rw-r--r--	pi

构建setup.py文件:

```
1 python3 setup.py build
```

构建成功后, 用管理员安装构建的wrapper扩展包:

```
1 sudo python3 setup.py install
```

```
pi@raspberrypi:~/codes/python_demo_v1.1$ python3 setup.py build
running build
running build_ext
pi@raspberrypi:~/codes/python_demo_v1.1$ sudo python3 setup.py install
running install
running build
running build_ext
running install_lib
copying build/lib.linux-aarch64-3.7/wrapper.cpython-37m-aarch64-linux-gnu.so ->
/usr/local/lib/python3.7/dist-packages
running install_egg_info
Removing /usr/local/lib/python3.7/dist-packages/wrapper-1.0.egg-info
Writing /usr/local/lib/python3.7/dist-packages/wrapper-1.0.egg-info
pi@raspberrypi:~/codes/python_demo_v1.1$
```

至此, 我们在编写python程序时, 就可以直接导入wrapper.py包了。

3 调用示例

我们打开给出的例程main.py源码：

```
1 import ctypes
2 import wrapper
3
4 # 900-1700nm光谱模组是228个数据点
5 PIXEL_NUM = 228
6
7 print("\n");
8
9 # 调用 dlpConnect() 函数
10 result = wrapper.dlpConnect()
11 print("连接设备数量:{}\n".format(result))
12
13 # 调用 dlpOpenByUsb() 函数
14 index = 0 # USB 设备索引
15 result = wrapper.dlpOpenByUsb(index)
16 if(result>=0):
17     print("hid usb 打开成功!\n")
18 else:
19     print("hid usb 打开失败!\n")
20
21 # 创建用于存储波长数据的缓存
22 wls_buf = (ctypes.c_double * PIXEL_NUM)()
23
24 # 调用 dlpGetWavelengths() 函数
25 result = wrapper.dlpGetWavelengths(wls_buf, PIXEL_NUM)
26 if(result>=0):
27     for i in range(5):
28         wavelength = wls_buf[i]
29         print("{:.2f}\n".format(wavelength)) #打印前5个波长，保留两位小数
30 else:
31     print("获取波长失败!\n")
32
33 print("\n") # 波长和强度换行
34
35 # 创建用于存储强度数据的缓存
36 intensities_buf = (ctypes.c_int * PIXEL_NUM)()
37
38 # 调用 dlpGetIntensities() 函数
39 activeIndex = 0 # 激活的索引
40 result = wrapper.dlpGetIntensities(activeIndex, intensities_buf, PIXEL_NUM)
41 if(result>=0):
42     for i in range(5):
43         intensity = intensities_buf[i]
```

```
44         print("{:.2f}: {}".format(wavelength, intensity)) #打印前5个波长和对应强度  
    值  
45 else:  
46     print("获取强度失败!\n")
```

1) 导入所需模块

```
1 import ctypes  
2 import wrapper
```

2) 定义数据点数常量

```
1 # 900-1700nm光谱模组是228个数据点  
2 PIXEL_NUM = 228
```

3) 连接usb获取已连接光谱仪数量

```
1 # 调用 dlpConnect() 函数  
2 result = wrapper.dlpConnect()  
3 print("连接设备数量:{}\n".format(result))
```

4) 创建用于存储波长数据的缓存，即c语言类型的 double 指针，长度为 PIXEL_NUM；调用 获取波长函数，波长值均存入缓存 wls_buf 中；打印出波长前5个值

```
1 # 创建用于存储波长数据的缓冲区  
2 wls_buf = (ctypes.c_double * PIXEL_NUM)()  
3  
4 # 调用 dlpGetWavelengths() 函数  
5 result = wrapper.dlpGetWavelengths(wls_buf, PIXEL_NUM)  
6 if(result>=0):  
7     for i in range(5):  
8         wavelength = wls_buf[i]  
9         print("{:.2f}\n".format(wavelength)) #打印前5个波长，保留两位小数  
10 else:  
11     print("获取波长失败!\n")
```

5) 获取强度值，共计 PIXEL_NUM (即228) 个数据点。与获取波长类似，先创建c语言类型的 int 型指针，长度为 PIXEL_NUM；选择激活设置中的第1个配置，对应索引为0；调用获取强度函

数，强度值存储在 `intensities_buf` 缓存中；打印前5个波长和对应强度值。

```
1 # 创建用于存储强度数据的缓存
2 intensities_buf = (ctypes.c_int * PIXEL_NUM)()
3
4 # 调用 dlpGetIntensities() 函数
5 activeIndex = 0 # 激活的索引
6 result = wrapper.dlpGetIntensities(activeIndex, intensities_buf, PIXEL_NUM)
7 if(result>=0):
8     for i in range(5):
9         intensity = intensities_buf[i]
10        print("{:.2f}: {}".format(wavelength, intensity)) # 打印前5个波长和对应强度
    值
11 else:
12     print("获取强度失败!\n")
```

6) 用管理员执行main.py文件，打印对应结果

```
1 sudo python3 main.py
```

```
pi@raspberrypi:~/codes/python_demo_v1.1
pi@raspberrypi:~/codes/python_demo_v1.1 $ sudo python3 main.py

vendor_id: 0451
enumerate num: 1
连接设备数量:1

usb open vendor_id: 0451
devHandle is ok!
hid usb 打开成功!

GetWavelengths times: 2691
900.85
904.74
909.92
913.80
917.67

917.67: 425
917.67: 577
917.67: 262
917.67: 549
917.67: 216
pi@raspberrypi:~/codes/python_demo_v1.1 $
```

注意：

- 1) 由于涉及到对usb hid设备的调用，这里需要使用管理员权限执行main.py。
- 2) 获取波长和强度值时，均会有一段扫描时间。该时间与配置信息中设置的平均次数有关，平均次数越大，扫描时间越长。

4 常见问题

4.1 提示找不到udev相关信息

udev是Linux系统中用于管理设备节点的守护进程，由于涉及到对usb hid接口的调用，需要系统在系统中安装udev驱动包。在基于Debian的系统（如Ubuntu）上，您可以使用以下命令安装udev的开发文件和库：

```
1 sudo apt-get update  
2 sudo apt-get install libudev-dev
```