

招财大龙猫

博客园 首页 新随笔 联系 订阅 管理

python与C语言调用模块 ctypes的详解

ctypes

ctypes是python的一个数据库，提供和C语言兼容的数据类型，可以直接调用动态链接库中的导出函数。
为了使用ctypes，必须依次完成以下步骤：

- 加载动态链接库
- 将python对象转换成ctypes所能识别的参数
- 使用ctypes所能识别的参数调用动态链接库中的函数

动态链接库加载方式有三种：

- cdll
- windll
- oledll

它们的不同之处在于：动态链接库中的函数所遵守的函数调用方式(calling convention)以及返回方式有所不同。

cdll用于加载遵循cdecl调用约定的动态链接库，windll用于加载遵循stdcall调用约定的动态链接库，oledll与windll完全相同，只是会默认从其导入的函数统一返回一个Windows HRESULT错误编码。

函数调用约定：函数调用约定指的是函数参数入栈的顺序，哪些参数入栈，哪些通过寄存器传值、函数返回时栈帧的回收方式(是由调用者负责清理，还是被调用者清理)、函数名称的修饰方法等等。常见的调用约定有cdecl和stdcall两种。在《程序员自我修养——链接、装载与库》一书的第10章有对函数调用约定的更详细介绍。

cdecl规定函数参数列表以从右到左的方式入栈，且由函数的调用者负责清除栈帧上的参数，stdcall的参数入栈方式与cdecl一致，但函数返回时是由被调用者自己负责清理栈帧。而且stdcall是Win32 API函数所使用的调用约定。

例子：

Linux下：

```
[jingjiang@iZ255w0dc5eZ ~]$ python2.7
Python 2.7.7 (default, May 8 2015, 00:24:34)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from ctypes import *
>>> libc = cdll.LoadLibrary("libc.so.6")
>>> libc.printf("%d", 2)
21
```

或者：

```
[jingjiang@iZ255w0dc5eZ ~]$ python2.7
Python 2.7.7 (default, May 8 2015, 00:24:34)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from ctypes import *
>>> libc = CDLL("libc.so.6")
>>> libc.printf("%d", 2)
21
```

其他例子：

```
[jingjiang@iZ255w0dc5eZ ~]$ python2.7
Python 2.7.7 (default, May 8 2015, 00:24:34)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from ctypes import *
>>> libc = CDLL("libc.so.6")
>>> libc.time( None )
1442212767
>>> libc.atoi( "123456" )
123456
```

一个完整的例子：

1. 编写动态链接库

```

// filename: foo.c

#include "stdio.h"

char* myprint(char *str)
{
    puts(str);
    return str;
}

float add(float a, float b)
{
    return a + b;
}

```

将foo.c编译为动态链接库：

```
gcc -fPIC -shared foo.c -o foo.so
```

2.使用ctypes调用foo.so

```

#coding:utf8

#FILENAME:foo.py

from ctypes import *

foo = CDLL('./foo.so')

myprint = foo.myprint
myprint.argtypes = [POINTER(c_char)] # 参数类型为char指针
myprint.restype = c_char_p # 返回类型为char指针
res = myprint('hello ctypes')
print(res)

add = foo.add
add.argtypes = [c_float, c_float] # 参数类型为两个float
add.restype = c_float # 返回类型为float
print(add(1.3, 1.2))

```

公告

昵称：招财大龙猫
园龄：3年7个月
粉丝：7
关注：0
+加关注

| 2022年8月 | | | | | | | > |
|---------|----|----|----|----|----|----|---|
| 日 | 一 | 二 | 三 | 四 | 五 | 六 | |
| 31 | 1 | 2 | 3 | 4 | 5 | 6 | |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 | |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | |
| 28 | 29 | 30 | 31 | 1 | 2 | 3 | |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 | |

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

python(7)
python RPC(3)
RabbitMQ(2)
SIP(1)
python AES(1)
nginx(1)
Fabric(1)
supervisor 参数(1)
linux(1)
mysql(1)
更多

随笔档案

2022年1月(5)
2021年10月(1)
2021年9月(2)
2021年6月(3)
2021年4月(2)
2020年11月(1)
2020年10月(1)
2020年9月(2)
2020年8月(1)
2019年10月(4)
2019年9月(1)
2019年8月(2)
2019年7月(1)
2019年6月(6)
2019年5月(13)
更多

阅读排行榜

- YOLOV3中Darknet中cfg文件说明和理解(14515)
- python 使用 with open() as 读写文件(14436)
- python基础之socket与socketserver(11101)
- YOLOv3 算法的详细说明(7007)
- yolov-7--解决报错:/bin/sh: 1: nvcc: not fo und make: *** [obj/convolutional_kernels.o] Error 127(5392)

评论排行榜

- YOLOV3中Darknet中cfg文件说明和理解(2)
- 关于win10深度学习安装配置 CUDA9.0+VS2017+Cudnn7.4.1.5+Anaconda3(cupy安装包)+python3.7+pycharm(2)
- 文竹越长越乱？教你7种修剪方法可保持文竹株形优美、健壮浓密(1)
- darknet53 yolo 下的识别训练(1)

推荐排行榜

- python基础之socket与socketserver(3)
- YOLOV3中Darknet中cfg文件说明和理解(2)
- telnet批量检测端口状态(linux)(2)
- python 使用 with open() as 读写文件(1)
- Tensorflow实战 手写数字识别(Tensorboard可视化)(1)

最新评论

1. Re:YOLOV3中Darknet中cfg文件说明和理解
@JY小脚 是的，是相乘的关系

—招财大龙猫
2. Re:YOLOV3中Darknet中cfg文件说明和理解
learning_rate=0.001 ★★一点小说明：实际学习率与GPU的个数有关，例如你的学习率设置为0.001，如果你有4块GPU，那 真实学习率为0.001/4 博主，这一点是不是写错了。 ...
—JY小脚丫

3. Re:关于win10深度学习安装配置
CUDA9.0+VS2017+Cudnn7.4.1.5+Anaconda3(cupy安装包)+python3.7+pycharm
@pengkai 是的，需要的...

执行:

```
1 [jingjiang@iZ25Sw0dc5eZ test]$ python2.6 foo.py
2 hello ctypes
3 hello ctypes
4 2.5
```

ctypes数据类型和C数据类型对照表

| C Type | Python Type | ctypes Type |
|-----------------------------|----------------------------|-------------|
| char | 1-character string | c_char |
| wchar_t | 1-character Unicode string | c_wchar |
| char | int/long | c_byte |
| char | int/long | c_ubyte |
| short | int/long | c_short |
| unsigned short | int/long | c_ushort |
| int | int/long | C_int |
| unsigned int | int/long | c_uint |
| long | int/long | c_long |
| unsigned long | int/long | c_ulong |
| long long | int/long | c_longlong |
| unsigned long long | int/long | c_ulonglong |
| float | float | c_float |
| double | float | c_double |
| char * (NULL terminated) | string or none | c_char_p |
| wchar_t * (NULL terminated) | unicode or none | c_wchar_p |
| void * | int/long or none | c_void_p |

查找动态链接库

```
1
2
3 >>> from ctypes.util import find_library
4 >>> find_library("m")
5 'libm.so.6'
6
7 >>> find_library("c")
8 'libc.so.6'
9
10 >>> find_library("bz2")
11 'libbz2.so.1.0'
```

函数返回类型

函数默认返回 C int 类型, 如果需要返回其他类型, 需要设置函数的 restype 属性。

```
1
2
3 >>> from ctypes import *
4 >>> from ctypes.util import find_library
5 >>> libc = cdll.LoadLibrary(find_library("c"))
6 >>> strchr = libc.strchr
7 >>> strchr("abcdef", ord("d"))
8 -808023673
9
10 >>> strchr.restype = c_char_p
11 >>> strchr("abcdef", ord("d"))
12 'de'
```

回调函数

- 定义回调函数类型, 类似于c中的函数指针, 比如: void (*callback)(void* arg1, void* arg2), 定义为: callback = CFUNCTYPE(None, c_voidp, c_voidp) None表示返回位是void, 也可以是其他类型. 剩余的两个参数与c中的回调参数一致。
- 定义python回调函数:

```
1 def _callback(arg1, arg2):
2     #do sth
3     # ...
4     #return sth
```

- 注册回调函数:

```
1 cb = callback(_callback)
```

另外, 使用ctypes可以避免GIL的问题。

一个例子:

```
1
2 //callback.c
3
4 #include "stdio.h"
5
6 void showNumber(int n, void (*print)())
7 {
8     (*print)(n);
9 }
10
```

编译成动态链接库:

```
1 gcc -fPIC -shared -o callback.so callback.c
```

编写测试代码:

```
1
2 #FILENAME:callback.py
3
4 from ctypes import *
5
6 _cb = CFUNCTYPE(None, c_int)
7
8 def pr(n):
9     print 'this is : %d' % n
10
11 cb = _cb(pr)
12
13 callback = CDLL("./callback.so")
14 showNumber = callback.showNumber
15 showNumber.argtypes = [c_int, c_void_p]
16 showNumber.restype = c_void_p
17
18 for i in range(10):
19     showNumber(i, cb)
```

4. Re:关于win10深度学习安装配置

CUDA9.0+VS2017+Cudnn7.4.1.5+Anaconda3(cupy安装包)+python3.7+pycharm

请问楼主, 如果我在Anaconda下创建了一个虚拟环境, 那么也需要安装Cuda toolkit吗?

--pengkai

5. Re:darknet53 yolo 下的识别训练

博主, 我想请教以下, 我用cpu测试darknet时, 显示 Loading weights from yolov3.weights...Done 可是下面就没有任何输出了 也没有产生预测结果图像 光标一直...

--Wang_mh

执行：

```
$ python2.7 callback.py
this is : 0
this is : 1
this is : 2
this is : 3
this is : 4
this is : 5
this is : 6
this is : 7
this is : 8
this is : 9
```

结构体和联合

union (联合体 共用体)

1.union中可以定义多个成员,union的大小由最大的成员的大小决定。

2.union成员共享同一块大小的内存,一次只能使用其中的一个成员。

3.对某一个成员赋值,会覆盖其他成员的值(也不奇怪,因为他们共享一块内存,但前提是成员所占字节数相同,当成员所占字节数不同时只会覆盖相应字节上的值,比如对char成员赋值就不会把整个int成员覆盖掉,因为char只占一个字节,而int占四个字节)

4.联合体union的存放顺序是所有成员都从低地址开始存放的。

结构体和联合必须从Structure和Union继承。子类必须定义__fields__属性。__fields__属性必须是一个二元组的列表,包含field的名称和field的类型。field类型必须是一个ctypes的类型。例如:c_int,或者其他继承自ctypes的类型。例如:结构体,联合,数组,指针。

```
from ctypes import *

class Point(Structure):
    __fields__ = [
        ("x", c_int),
        ("y", c_int),
    ]

    def __str__(self):
        return "x={0.x}, y={0.y}".format(self)

point1 = Point(x=10, y=20)
print "point1:", point1

class Rect(Structure):
    __fields__ = [
        ("upperleft", Point),
        ("lowerright", Point),
    ]

    def __str__(self):
        return "upperleft:[{0.upperleft}], lowerright:[{0.lowerright}]".format(self)

rect1 = Rect(upperleft=Point(x=1, y=2), lowerright=Point(x=3, y=4))
print "rect1:", rect1
```

运行：

```
python test.py
point1: x=10, y=20
rect1: upperleft:[x=1, y=2], lowerright:[x=3, y=4]
```

数组

数组定义很简单,比如:定义一个有10个Point元素的数组。

TenPointsArrayType = Point * 10。

初始化和使用数组：

```
from ctypes import *

TenIntegersArrayType = c_int * 10
array1 = TenIntegersArrayType(*range(1, 11))print array1

for i in array1:
    print i
```

运行：

```
$ python2.7 array.py
<_main_.c_int_Array_10 object at 0x7fad0d7394d0>
1
2
3
4
5
6
7
8
9
10
```

指针

pointer()可以创建一个指针,Pointer实例有一个contents属性,返回指针指向的内容。

```
>>> from ctypes import *
>>> i = c_int(42)
>>> p = pointer(i)
>>> p<_main_.LP_c_int object at 0x7f413081d560>
>>> p.contents
c_int(42)
>>>
```

可以改变指针指向的内容

```
>>> i = c_int(99)
>>> p.contents = i
>>> p.contents
c_int(99)
```


可以按数组的方式访问,并改变值

```
>>> p[0]
```

```
99
>>> p[0] = 22
>>> i
c_int(22)
```


传递指针或引用

很多情况下, c函数需要传递指针或引用, ctypes也完美支持这一点。
byref()用来传递引用参数, pointer()也可以完成同样的工作, 但是pointer会创建一个实际的指针对象, 如果你不需要一个指针对象, 用byref()会快很多。

```

>>> from ctypes import *
>>> i = c_int()
>>> f = c_float()
>>> s = create_string_buffer('\000' * 32) >>> print i.value, f.value, repr(s.value)
0 0.0 ''
>>> libc = CDLL("libc.so.6")
>>> libc.sscanf("1 3.14 Hello", "%d %f %s", byref(i), byref(f), s)
3
>>> print i.value, f.value, repr(s.value)
1 3.1400001049 'Hello'
```


可改变内容的字符串

如果需要可改变内容的字符串, 需要使用 createstringbuffer()


```

>>> from ctypes import *
>>> p = create_string_buffer(3) # create a 3 byte buffer, initialized to NUL bytes
>>> print sizeof(p), repr(p.raw)
3 '/x00/x00/x00'>>> p = create_string_buffer("Hello") # create a buffer containing a NUL terminated string
>>> print sizeof(p), repr(p.raw)
6 'Hello/x00'
>>> print repr(p.value)
'Hello'
>>> p = create_string_buffer("Hello", 10) # create a 10 byte buffer
>>> print sizeof(p), repr(p.raw)
10 'Hello/x00/x00/x00/x00/x00'
>>> p.value = "Hi"
>>> print sizeof(p), repr(p.raw)
10 'Hi/x001o/x00/x00/x00/x00/x00'
>>>
```

赋值给c_char_p, c_wchar_p, c_void_p

只改变他们指向的内存地址, 而不是改变内存的内容

```

>>> s = "Hello, World"
>>> c_s = c_char_p(s)
>>> print c_s
c_char_p('Hello, World')>>> c_s.value = "Hi, there"
>>> print c_s
c_char_p('Hi, there')
>>> print s
# first string is unchanged
Hello, World
>>>
```

数据都可以改变

```

>>> i = c_int(42)
>>> print i
c_int(42)
>>> print i.value42
>>> i.value = -99
>>> print i.value
-99
>>>
```

使用中遇到的一些问题

1: 当动态库的导出函数返回char *的时候, 如何释放内存
如果把restype设置为c_char_p, ctypes会返回一个常规的Python字符串对象, 一种简单的方式就是使用void *和强制转换结果。

```


string.c:

#include
#include
#include

char *get(void)
{
    char *buf = "Hello World";
    char *new_buf = strdup(buf);
    printf("allocated address: %p\n", new_buf);
    return new_buf;
}


void freeem(char *ptr)
{
    printf("freeing address: %p\n", ptr);
    free(ptr);
}
```

Python使用:

```

from ctypes import *

lib = cdll.LoadLibrary('./string.so')
lib.freeem.argtypes = c_void_p,
lib.freeem.restype = None
lib.get.argtypes = []
lib.get.restype = c_void_p

>>> ptr = lib.get()
allocated address: 0x9facd8
>>> hex(ptr)
'0x9facd8'
>>> cast(ptr, c_char_p).value
```

```
"Hello World"
>>> lib.freeme(ptr)
freeing address: 0x9facad8

```

也可以使用c_char_p的子类, 因为ctypes不会对简单类型的子类调用getfunc:

```
class c_char_p_sub(c_char_p):
    pass

lib.get.restype = c_char_p_sub
```

value属性会返回字符串. 在这个例子中, 可以把freeme的参数改为更通用的c_void_p, 它接受任何指针类型或整型地址。


2: 如何把含有'\0'的char*转换成python字符串
[How do you convert a char * with 0-value bytes into a python string?](#)

参考资料

- python ctypes库中动态链接库加载方式
- 用python ctypes调用动态链接库
- ctypes 使用方法与说明
- C语言union(联合体 共用体)
- Python ctypes: how to free memory? Getting invalid pointer error

标签: python

好文置顶 关注我 收藏该文  

 招财大龙猫
粉丝 - 7 关注 - 0
[+加关注](#)


0
推荐

0
反对

« 上一篇: 运维部署神器 Fabric 支持 Python3
» 下一篇: nginx 高并发下有配置说明和示例

posted @ 2020-11-18 14:13 招财大龙猫 阅读(3684) 评论(0) 编辑 收藏 举报

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 登录后才能查看或发表评论, 立即 [登录](#) 或者 [注册](#) 博客园首页

【社区专享】0成本搭建支持连麦、虚拟人、即时通信的直播间

 ZEGO 即购

社区开发者专享

0 成本搭建支持连麦、虚拟人、即时通信的直播间

低延迟 • 不卡顿 • 易接入

编辑推荐:

- 使用 CSS 构建强大且酷炫的粒子动画
- [C#]GDI+之鼠标交互: 原理、示例、一步步深入、性能优化
- 一文带你弄懂 CDN 技术的原理
- 妙用 CSS 构建花式透视背景效果
- .NET IoT 入门指南: 基于 GPS 的 NTP 时间同步服务器

 SQL 专家云

SQL Server 数据库可视化、智能化运维平台

可随时随地监控数据库、性能监控、性能优化、智能告警、报表生成、自动故障排查

最新新闻

- TIOBE 编程语言排行榜 “哄”
 - 英特尔开源行业首个路径引导库, 将与 Blender 集成
 - TIOBE 8 月榜单: Python 市占达新高, Carbon 位列第 192
 - 阿里巴巴云原生大数据运维平台SREWorks正式开源
 - 为何AlphaFold不会对新药研制产生革命性影响?
- » 更多新闻...