15.13 传递NULL结尾的字符串给C函数库¶

问题¶

你要写一个扩展模块，需要传递一个NULL结尾的字符串给C函数库。 不过，你不是很确定怎样使用Python的Unicode字符串去实现它。

解决方案¶

许多C函数库包含一些操作NULL结尾的字符串，被声明类型为 `char *`. 考虑如下的C函数，我们用来做演示和测试用的：

```
void print_chars(char *s) {
    while (*s) {
        printf("%2x ", (unsigned char) *s);

        s++;
    }
    printf("\n");
}
```

此函数会打印被传进来字符串的每个字符的十六进制表示，这样的话可以很容易的进行调试了。例如：

```
print_chars("Hello");   // Outputs: 48 65 6c 6c 6f
```

对于在Python中调用这样的C函数，你有几种选择。 首先，你可以通过调用 `PyArg_ParseTuple()` 并指定”y“转换码来限制它只能操作字节，如下：

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  char *s;

  if (!PyArg_ParseTuple(args, "y", &s)) {
    return NULL;
  }
  print_chars(s);
  Py_RETURN_NONE;
}
```

结果函数的使用方法如下。仔细观察嵌入了NULL字节的字符串以及Unicode支持是怎样被拒绝的：

```
>>> print_chars(b'Hello World')
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>> print_chars(b'Hello\x00World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be bytes without null bytes, not bytes
>>> print_chars('Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' does not support the buffer interface
>>>
```

如果你想传递Unicode字符串，在 `PyArg_ParseTuple()` 中使用”s“格式码，如下：

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  char *s;

  if (!PyArg_ParseTuple(args, "s", &s)) {
    return NULL;
  }
  print_chars(s);
  Py_RETURN_NONE;
}
```

当被使用的时候，它会自动将所有字符串转换为以NULL结尾的UTF-8编码。例如：

```
>>> print_chars('Hello World')
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>> print_chars('Spicy Jalape\u00f1o')  # Note: UTF-8 encoding
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> print_chars('Hello\x00World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str without null characters, not str
>>> print_chars(b'Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not bytes
>>>
```

如果因为某些原因，你要直接使用 `PyObject *` 而不能使用 `PyArg_ParseTuple()`， 下面的例子向你展示了怎样从字节和字符串对象中检查和提取一个合适的 `char *` 引用：

```
/* Some Python Object (obtained somehow) */
PyObject *obj;

/* Conversion from bytes */
{
   char *s;
   s = PyBytes_AsString(o);
   if (!s) {
      return NULL;   /* TypeError already raised */
   }
   print_chars(s);
}

/* Conversion to UTF-8 bytes from a string */
{
   PyObject *bytes;
   char *s;
   if (!PyUnicode_Check(obj)) {
       PyErr_SetString(PyExc_TypeError, "Expected string");
       return NULL;
   }
   bytes = PyUnicode_AsUTF8String(obj);
   s = PyBytes_AsString(bytes);
   print_chars(s);
   Py_DECREF(bytes);
}
```

前面两种转换都可以确保是NULL结尾的数据， 但是它们并不检查字符串中间是否嵌入了NULL字节。 因此，如果这个很重要的话，那你需要自己去做检查了。

讨论¶

如果可能的话，你应该避免去写一些依赖于NULL结尾的字符串，因为Python并没有这个需要。 最好结合使用一个指针和长度值来处理字符串。 不过，有时候你必须去处理C语言遗留代码时就没得选择了。

尽管很容易使用，但是很容易忽视的一个问题是在 `PyArg_ParseTuple()` 中使用“s”格式化码会有内存损耗。 但你需要使用这种转换的时候，一个UTF-8字符串被创建并永久附加在原始字符串对象上面。 如果原始字符串包含非ASCII字符的话，就会导致字符串的尺寸增到一直到被垃圾回收。例如：

```
>>> import sys
>>> s = 'Spicy Jalape\u00f1o'
>>> sys.getsizeof(s)
87
>>> print_chars(s)     # Passing string
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> sys.getsizeof(s)   # Notice increased size
103
>>>
```

如果你在乎这个内存的损耗，你最好重写你的C扩展代码，让它使用 `PyUnicode_AsUTF8String()` 函数。如下：

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  PyObject *o, *bytes;
  char *s;

  if (!PyArg_ParseTuple(args, "U", &o)) {
    return NULL;
  }
  bytes = PyUnicode_AsUTF8String(o);
  s = PyBytes_AsString(bytes);
  print_chars(s);
  Py_DECREF(bytes);
  Py_RETURN_NONE;
}
```

通过这个修改，一个UTF-8编码的字符串根据需要被创建，然后在使用过后被丢弃。下面是修订后的效果：

```
>>> import sys
>>> s = 'Spicy Jalape\u00f1o'
>>> sys.getsizeof(s)
87
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> sys.getsizeof(s)
87
>>>
```

如果你试着传递NULL结尾字符串给ctypes包装过的函数， 要注意的是ctypes只能允许传递字节，并且它不会检查中间嵌入的NULL字节。例如：

```
>>> import ctypes
>>> lib = ctypes.cdll.LoadLibrary("./libsample.so")
>>> print_chars = lib.print_chars
>>> print_chars.argtypes = (ctypes.c_char_p,)
>>> print_chars(b'Hello World')
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>> print_chars(b'Hello\x00World')
48 65 6c 6c 6f
>>> print_chars('Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ctypes.ArgumentError: argument 1: <class 'TypeError'>: wrong type
>>>
```

如果你想传递字符串而不是字节，你需要先执行手动的UTF-8编码。例如：

```
>>> print_chars('Hello World'.encode('utf-8'))
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>>
```

对于其他扩展工具（比如Swig、Cython）， 在你使用它们传递字符串给C代码时要先好好学习相应的东西了。