使用swig连接python与c

注：仅示例python3

swig官方文档：http://www.swig.org/Doc3.0/SWIGDocumentation.html#SWIG_nn3， 目前swig的最新版本已经到了4.0

1. python端发送bytes数据到c

通过python发一组数据到C，C调用底层硬件完成数据收尾操作，数据主要是char *的形式到c，python这边需要使用bytes对其相连。

```
from binascii import b2a_hex, a2b_hex

b = b'\x78\x79\x86\x88\xfa\x33\x45'    #以byte表示的bytes, binascii.a2b_hex之后的
print(f'b.hex() = {b.hex()}')

print(f'binascii.b2a_hex(b) = {binascii.b2a_hex(b)}')

b2a = b'78798688fa3345'    #以字符串形式表示的bytes, binascii.b2a_hex之后的
a2b = binascii.a2b_hex(b2a)
print(f'binascii.a2b_hex(b2a) = {binascii.a2b_hex(b2a)}')


io.py2c_test(a2b,7)    #以c的交互需要使用以byte(a2b)表示的bytes，不能是binascii.b2a_hex之后的。
```

在swig的.i文件中需要加入以下内容(仅支持python3)，表示c char*之间与python bytes相对而不是采用unicode，这一点相当重要，在python中不要使用字符串与C之间传递，而应该用bytes来传递，简单可行，不会发生字符转换规则以及check等等问题，latin-1转换>0x80的字符时有问题，utf8本身也不是属于byte0-255之间的字符定义）。

```
%begin %{
#define SWIG_PYTHON_STRICT_BYTE_CHAR
%}
```

c侧程序示例如下： 注：参数只能写成 char * ，不能写成unsigned char*, 但是可以在c内部通过强制转换将其转换成unsigned char *，

```
void py2c_test(char * tdi_bytes, int bytes_size)
{
    int i;
    printf("mpsse.c:: python to c bytes_size = %0d\n", bytes_size);
    for (i=0;i<bytes_size; i++) {
        printf("mpsse.c:: python to c (unsigned char ) = %0x tdi_bytes = %0x\n", (unsigned char )tdi_bytes[i], tdi_bytes[i]);
```

```
    }
}
```

2. c返回bytes数据给python

如果直接用c的char *传数据的话，会在遇到0时，自动把数据截止掉。举个例子：

c原来要返回的数据是0x55001233, python端会收到0x1233这个bytes数据，因为后面遇到0x00被截止掉。

所以C端返回这样的数据需要使用到struct, c端建立以下的struct:

```
typedef struct swig_bytes_data
{
        int size;
        char *bytes;
} swig_bytes_data;
```

c端要返回给python的数据，指定好正确的bytes和size之后python端即能接收到正常的数据。

然后在xxx.i文件中，指定typemap(out)

```
%typemap(out) swig_bytes_data
{
        $result = PyBytes_FromStringAndSize($1.bytes, $1.size);
        free($1.bytes);
}
```

2.1 语法参考

http://vmcc.vicp.net:9090/python/c-api/bytes.html

http://vmcc.vicp.net:9090/python/c-api/concrete.html#sequence-objects

2.2 相关function

```
PyObject* PyBytes_FromStringAndSize(const char *v, Py_ssize_t len)
//Return value: New reference.
//Return a new bytes object with a copy of the string v as value and length
len on success, and NULL on failure. If v is NULL, the contents of the bytes
object are uninitialized.



PyObject* PyByteArray_FromStringAndSize(const char *string, Py_ssize_t len)
//Return value: New reference.
//Create a new bytearray object from string and its length, len. On failure,
```

```
NULL is returned.



PyObject* PyUnicode_FromStringAndSize(const char *u, Py_ssize_t size)
//Return value: New reference.
//Create a Unicode object from the char buffer u. The bytes will be
interpreted as being UTF-8 encoded. The buffer is copied into the new
object. If the buffer is not NULL, the return value might be a shared
object, i.e. modification of the data is not allowed.

//If u is NULL, this function behaves like PyUnicode_FromUnicode() with the
buffer set to NULL. This usage is deprecated in favor of PyUnicode_New().
```