# 1. JNA模拟char\*。

根据文档，最开始想到的就是用Java的字符串String模拟char\*，因为char\*是字符数组，可以理解成字符串。

比如在C语言中：

|  |
| --- |
| \_\_declspec(dllexport)  char\* getStr(char\* chs)  {  printf("%s\n", chs);  return chs;  } |

这样在Java中模拟的方法为：

|  |
| --- |
| String getStr(String chs); |

这样在Java中使用getStr()方法是能够正常调用和返回的。

但是，由于char\*也是指针类型，在C语言中，经常会用该参数输出数据。比如以下的方法：

|  |
| --- |
| \_\_declspec(dllexport)  void polishChs(char\* chs)  {  chs[0] = 'a';  } |

会将传入的char数组的第一个元素更改为字母’a’。也就是说传入参数的数据被改变了。但是如果在Java中用String类型模拟这个方法，比如：

|  |
| --- |
| void polishChs(String chs); |

那么调用polishChs()方法时，Java中这个参数的值是无法改变的，即无法接收到这个修改后的“传出参数”的值。因为Java中，本来在函数中就不能修改传入的String类型变量的值。这点有时会产生很大问题，即无法获得结果。

这种情况如何解决呢？

我们考虑到char\*实际是字符数组，而一个char类型变量占据8位（一个字节）的存储单元，因此，他实际上对应到Java中，就是字节数组而已！因此我们使用字节数组模拟。我们将方法模拟为：

|  |
| --- |
| **void** polishChs(**byte**[] chs); |

调用方法示例：

|  |
| --- |
| **byte**[] b = **new** **byte**[]{98, 99, 100}; // 相当于b c d三个字母。字节数组不能是null，字节数组的长度要根据需要来。  CAD.*instanceDll*.polishChs(b);  System.*out*.println(Arrays.*toString*(b)); |

输出的结果就是“[97, 99, 100]”了。完美。

## 2. Java模拟unsigned char\*

C语言常用于开发底层硬件，那么在实际中，就常会遇到unsigned类型变量。unsigned是无符号位的，但是Java中没有无符号的数据类型。

那么这么模拟呢？还是只要用具有相同位数的数据类型就可以了！比如unsigned char是无符号8位数据，那么Java中就用byte数据，他们的位数是一样的。但是unsigned char可表示0~255，byte表示的是-128~127的数据。那么当数据进行传递的时候，如果传入C函数是大于127的数据（但小于256），或者是C函数返回的数据是大于127的（但小于256），怎么搞？实际仔细想一想就有通用的方法：

（1）传入的数据大于127（但小于256）：直接将数字强转成byte数据类型即可，因为虽然强转后，在Java表现为负数，但是内存中的该数据单元是没有变化的（如果看过谭浩强C语言就知道了，或者知道补码等知识），因此，将这个“负数”传进C语言调用中，由于C接受的类型是unsigned char，因此C语言中实际处理的还是正数。强转的例子：

|  |
| --- |
| **byte** bt = (**byte**) 235; // 超过127要强转  System.*out*.println(bt); // 强转后，Java中输出的是-21 |

但将bt传入C语言的unsigned char参数，C语言接收到的数据还是235，可自行实验。

（2）那么C语言将接收到的235再用unsigned char返回呢，那么Java怎么得到235呢？如果Java用byte接收，那接收到的肯定是-21，因此，C语言只要用int类型接收即可。

综上所述，模拟unsigned char时，Java中的传入参数用byte模拟，如果传入的参数超过127但小于256，强转即可；而模拟传出参数时，只要用Java的int类型即可。

实际上，传入和传出参数都用int也是可以的，因为传入用int也没影响。那么可不可以传入和传出参数都用byte呢？如果用byte，那么Java中接收到数据后，如果是大于127但小于256的，则在Java中表现为负数，那么我们再把这个byte（负数）转成正的int呗！解决方法如下：

|  |
| --- |
| **byte** res = -21; // 示例，res的值就是C语言返回的大于127的数，在Java中表现为负数。  **int** realData = 0x00ff & res; // 用 0x00ff 和得到的结果进行“与”位运算  System.*out*.println(realData); // 得到的结果就是235 |

上述的方法就是将Java中的byte看成无符号的数转成Java中的int。

为什么用“0x00ff”，因为每个十六进制位表示内存中的4位，后两个ff就可对应一个byte数据的内存。这样用0x00ff与byte数据进行位运算，原来的byte低位就原封不懂保存到int类型数据的低位（ff（就是八个1）和原byte中数据进行与操作，原来的数据不会改变，原来是0，与后还是0，原来是1，与后还是1），高位还是0，这样就转成了正数的int类型，因为不论如何，Java中的byte是不能显示为正数的。

这样就完全了解了如何进行unsigned的处理，int类型和byte类型都可以使用。

对于其他基本的数据类型，自行考虑解决，基本思路就是这样，需要考虑底层的数据存储。

说明：上述也能使用Java的char进行模拟（注意Java中char是两个字节的，因此该转换的还是要转换）。

现在再来模拟unsigned char\*数据类型。前提是之前的内容都要看过，前面所有提到的不是无谓的，是有一定的联系的，以便更好理解。

unsigned char\*类型，相当于是unsigned和char\*的“结合”，即是char类型数组，并且是无符号的，因此较复杂。由于unsigned char\*也要考虑该方法的参数既是传入参数，又是传出参数。例如：

|  |
| --- |
| \_\_declspec(dllexport)  void polishUChs(unsigned char \* uchs)  {  uchs[0] = 'a';  uchs[1] = 'b';  } |

那么这时肯定是用数组了。仔细想想，当然是用byte数组或者是int数组（或者是char数组呢）。

实践证明，使用int数组和char数组是不行的，因为在Java中，char是占两个字节（Unicode编码），int占四个字节，如果传入int数组或者char数组，那么C语言中会把int数组中的一个元素看成是uchs数组中的四个元素，同样会把char数组中的一个元素当成uchs中的两个元素。

因此对于数组来说，必须要使用位数相同的数据类型，即这里必须要使用byte[]数组类型，传入时，超过127则强转，获得传出参数后，每个byte数组的元素通过和“0x00ff”进行与运算即可。

Java模拟示例：

|  |
| --- |
| **package** demo.dll;  **import** com.sun.jna.Library;  **import** com.sun.jna.Native;  **public** **interface** CAD **extends** Library {  // ConsoleApplication2是DLL文件名  CAD *instanceDll* = (CAD) Native.*loadLibrary*("ConsoleApplication2", CAD.**class**);  **void** polishUChs(**byte**[] uchs);  } |

Java调用示例：

|  |
| --- |
| // 无符号byte的范围是0x00~0xff(255)，在Java中，超过0x7f(127)就要强转。  **byte**[] uchs = **new** **byte**[]{0x02, 0x04, 0x7f, (**byte**) 0x81};  CAD.*instanceDll*.polishUChs(uchs);  // 遍历被polishUChs()改变后的uchs，处理成int数组就好了  **int**[] realArr = **new** **int**[4]; // 用于存储4个数  **for** (**int** i = 0; i < uchs.length; i++) {  realArr[i] = 0x00ff & uchs[i];  }  // 输出int数组  System.*out*.println(Arrays.*toString*(realArr)); // 结果是[97, 98, 127, 129]。没问题 |

这样就完美解决了。

总结分析：

1. 这里主要验证的就是传入和传出的正确性。这个正确了，基本就确定无误了。

2. 遇到其他数据类型，也要考虑底层的数据的结构，明白之后将会豁然开朗，举一反三。我做了这个实践就是这样的感觉。所以大家自己可以试试。（所以说，谭浩强的那本C语言还是有价值的）至于一些更复杂的，比如char\*\*什么玩意的，自己研究吧，我现在也没研究，C语言不是非常非常好。希望要有举一反三的思想，多考虑数据的底层，因为是C嘛。

完结。