# 15.16 不确定编码格式的C字符串¶

## 问题¶

你要在C和Python直接来回转换字符串，但是C中的编码格式并不确定。 例如，可能C中的数据期望是UTF-8，但是并没有强制它必须是。 你想编写代码来以一种优雅的方式处理这些不合格数据，这样就不会让Python奔溃或者破坏进程中的字符串数据。

## 解决方案¶

下面是一些C的数据和一个函数来演示这个问题：

```
/* Some dubious string data (malformed UTF-8) */
const char *sdata = "Spicy Jalape\xc3\xb1o\xae";
int slen = 16;

/* Output character data */
void print_chars(char *s, int len) {
  int n = 0;
  while (n < len) {
    printf("%2x ", (unsigned char) s[n]);
    n++;
  }
  printf("\n");
}
```

在这个代码中，字符串 `sdata` 包含了UTF-8和不合格数据。 不过，如果用户在C中调用 `print_chars(sdata, slen)` ，它缺能正常工作。 现在假设你想将 `sdata` 的内容转换为一个Python字符串。 进一步假设你在后面还想通过一个扩展将那个字符串传个 `print_chars()` 函数。 下面是一种用来保护原始数据的方法，就算它编码有问题。

```
/* Return the C string back to Python */
static PyObject *py_retstr(PyObject *self, PyObject *args) {
  if (!PyArg_ParseTuple(args, "")) {
    return NULL;
  }
  return PyUnicode_Decode(sdata, slen, "utf-8", "surrogateescape");
}

/* Wrapper for the print_chars() function */
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  PyObject *obj, *bytes;
  char *s = 0;
  Py_ssize_t   len;

  if (!PyArg_ParseTuple(args, "U", &obj)) {
    return NULL;
  }

  if ((bytes = PyUnicode_AsEncodedString(obj,"utf-8","surrogateescape"))
      == NULL) {
    return NULL;
  }
  PyBytes_AsStringAndSize(bytes, &s, &len);
  print_chars(s, len);
  Py_DECREF(bytes);
```

```
 Py_RETURN_NONE;
}
```

如果你在Python中尝试这些函数，下面是运行效果：

```
>>> s = retstr()
>>> s
'Spicy Jalapeño\udcae'
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f ae
>>>
```

仔细观察结果你会发现，不合格字符串被编码到一个Python字符串中，并且并没有产生错误， 并且当它被回传给C的时候，被转换为和之前原始C字符串一样的字节。

## 讨论¶

本节展示了在扩展模块中处理字符串时会配到的一个棘手又很恼火的问题。 也就是说，在扩展中的C字符串可能不会严格遵循Python所期望的Unicode编码/解码规则。 因此，很可能一些不合格C数据传递到Python中去。 一个很好的例子就是涉及到底层系统调用比如文件名这样的字符串。 例如，如果一个系统调用返回给解释器一个损坏的字符串，不能被正确解码的时候会怎样呢？

一般来讲，可以通过制定一些错误策略比如严格、忽略、替代或其他类似的来处理Unicode错误。 不过，这些策略的一个缺点是它们永久性破坏了原始字符串的内容。 例如，如果例子中的不合格数据使用这些策略之一解码，你会得到下面这样的结果：

```
>>> raw = b'Spicy Jalape\xc3\xb1o\xae'
>>> raw.decode('utf-8','ignore')
'Spicy Jalapeño'
>>> raw.decode('utf-8','replace')
'Spicy Jalapeño?'
>>>
```

`surrogateescape` 错误处理策略会将所有不可解码字节转化为一个代理对的低位字节（udcXX中XX是原始字节值）。 例如：

```
>>> raw.decode('utf-8','surrogateescape')
'Spicy Jalapeño\udcae'
>>>
```

单独的低位代理字符比如 `\udcae` 在Unicode中是非法的。 因此，这个字符串就是一个非法表示。 实际上，如果你将它传个一个执行输出的函数，你会得到一个错误：

```
>>> s = raw.decode('utf-8', 'surrogateescape')
>>> print(s)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'utf-8' codec can't encode character '\udcae'
in position 14: surrogates not allowed
>>>
```

然而，允许代理转换的关键点在于从C传给Python又回传给C的不合格字符串不会有任何数据丢失。 当这个字符串再次使用 `surrogateescape` 编码时，代理字符会转换回原始字节。例如：

```
>>> s
'Spicy Jalapeño\udcae'
>>> s.encode('utf-8','surrogateescape')
b'Spicy Jalape\xc3\xb1o\xae'
>>>
```

作为一般准则，最好避免代理编码——如果你正确的使用了编码，那么你的代码就值得信赖。 不过，有时候确实会出现你并不能控制数据编码并且你又不能忽略或替换坏数据，因为其他函数可能会用到它。 那么就可以使用本节的技术了。

最后一点要注意的是，Python中许多面向系统的函数，特别是和文件名、环境变量和命令行参数相关的 都会使用代理编码。例如，如果你使用像 `os.listdir()` 这样的函数， 传入一个包含了不可解码文件名的目录的话，它会返回一个代理转换后的字符串。 参考5.15的相关章节。

PEP 383 中有更多关于本机提到的以及和surrogateescape错误处理相关的信息。