目 录

[第1章 Python 3的bytes/str之别 1](#_Toc480364421)

[1.1 字符编码 1](#_Toc480364422)

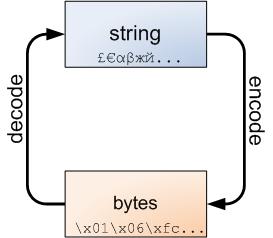
[1.2 bytes和str的区别和共同点 2](#_Toc480364423)

[1.3 实例 4](#_Toc480364424)

# Python 3的bytes/str之别

Python 3最重要的新特性之一是对字符串和二进制数据流做了明确的区分。文本总是Unicode，由str类型表示，二进制数据则由bytes类型表示。Python 3不会以任意隐式的方式混用str和bytes，，你不能拼接字符串和字节流，也无法在字节流里搜索字符串（反之亦然），也不能将字符串传入参数为字节流的函数（反之亦然）。下面让我们深入分析一下二者的区别和联系。

不管怎样，字符串和字节包之间的界线是必然的，下面的图解非常重要，务请牢记于心：



## 字符编码

谈到Python3.x中bytes类型和str类型，就不得不先说说编码的事情。

　　在计算机历史的早期，美国为代表的英语系国家主导了整个计算机行业，26个英文字母组成了多样的英语单词、语句、文章。因此，最早的字符编码规范是ASCII码，一种8位即1个字节的编码规范，它可以涵盖整个英语系的编码需要。那么，编码是什么？编码是字符的二进制表示方法！我们都知道，所有的英文字符、标点符号、特殊字符、汉字、片假名等等最终存储在磁盘上都是01010101这类东西。在计算机内部，读取和存储数据归根结底，处理的都是0/1组成的比特流。可问题是，人类看不懂，看不懂，看不懂这些比特流！于是出现了字符编码，它是个翻译机，在计算机内部某个地方，透明的帮我们将比特流翻译成人类可以直接理解的文字。对于一般用户，不需要知道这个过程是什么原理，是怎么执行的。但是对于程序员却是个必须搞清楚的问题。

　　以ASCII编码为例，它规定1个字节8个比特位代表1个字符的编码，也就是“00000000”这么宽，一个一个字节的解读。例如：01000001表示大写字母A，有时我们会“偷懒"的用65这个十进制来表示A在ASCII中的编码。8个比特位，可以没有重复的表示2的8次方也就是255个字符。

　　后来，计算机得到普及，中文、日文、韩文等等国家的文字需要在计算机内表示，ASCII的255位远远不够，于是标准组织制定出了叫做UNICODE的万国码，它规定任何一个字符（不管哪国的）至少以2个字节表示，可以更多。其中，英文字母就是用2个字节，而汉字是3个字节。这个编码虽然很好，满足了所有人的要求，但是它不兼容ASCII，同时还占用较多的空间和内存。因为，在计算机世界更多的字符是英文字母，明明可以1个字节就能够表示，非要用2个。

　　于是UTF-8编码应运而生，它规定英文字母系列用1个字节表示，汉字用3个字节表示等等。因此，它兼容ASCII，可以解码早期的文档。UTF-8很快就得到了广泛的应用。

　　在编码的曲折发展历程中，我国还创造了自己的编码方式，例如GBK，GB2312，BIG5。他们只局限于在国内使用，不被国外认可。在GBK编码中，中文汉字占2个字节。

## bytes和str的区别和共同点

让我们转回bytes和str的身上。bytes是一种比特流，它的存在形式是01010001110这种。我们无论是在写代码，还是阅读文章的过程中，肯定不会有人直接阅读这种比特流，它必须有一个编码方式，使得它变成有意义的比特流，而不是一堆无意义的01组合。因为编码方式的不同，对这个比特流的解读也会不同，对实际使用造成了很大的困扰。下面让我们看看Python是如何处理这一系列编码问题的：

>>> s = "中文"

>>> s

'中文'

>>> type(s)

<class 'str'>

>>> b = bytes(s,encoding='utf-8')

>>> b

b'\xe4\xb8\xad\xe6\x96\x87'

>>> type(b)

<class 'bytes'>

从例子可以看出，s是个字符串类型。Python有个内置函数bytes()可以将字符串转换成bytes类型，b实际上是一串01的组合，但为了在ide环境中让我们相对直观的观察，它被表现成了b'\xe4\xb8\xad\xe6\x96\x87'这种形式，开头的b表示这是一个bytes类型。\xe4是十六进制的表示方式，它占用1个字节的长度，因此”中文“被编码成utf-8后，我们可以数得出一共用了6个字节，每个汉字占用3个，这印证了上面的论述。在使用内置函数bytes()的时候，必须明确encoding的参数，不可省略。

　　我们都知道，字符串类（str）里有一个encode()方法,它是从字符串向比特流的编码过程。而bytes类型恰好有个decode()方法，它是从比特流向字符串解码的过程。除此之外，我们查看Python源码会发现bytes和str拥有几乎一模一样的方法列表，最大的区别就是encode和decode。

　　从实质上来说，字符串在磁盘上的保存形式也是01的组合，也需要编码解码。

　　如果，上面的阐述还不能让你搞清楚两者的区别，那么记住下面两几句话：

　　1.在将字符串存入磁盘和从磁盘读取字符串的过程中，Python自动地帮你完成了编码和解码的工作，你不需要关心它的过程。

　　2.使用bytes类型，实质上是告诉Python，不需要它帮你自动地完成编码和解码的工作，而是用户自己手动进行，并指定编码格式。

　　3.Python已经严格区分了bytes和str两种数据类型，你不能在需要bytes类型参数的时候使用str参数，反之亦然。这点在读写磁盘文件时容易碰到。

　　在bytes和str的互相转换过程中，实际就是编码解码的过程，必须显式地指定编码格式。

>>> b

b'\xe4\xb8\xad\xe6\x96\x87'

>>> type(b)

<class 'bytes'>

>>> s1 = str(b)

>>> s1

"b'\\xe4\\xb8\\xad\\xe6\\x96\\x87'"

>>> type(s1)

<class 'str'>

>>> s1 = str(b,encoding='utf-8')

>>> s1

'中文'

>>> type(s1)

<class 'str'>

我们再把字符串s1，转换成gbk编码的bytes类型：

>>> s1

'中文'

>>> type(s1)

<class 'str'>

>>> b = bytes(s1,encoding='gbk')

>>> b

b'\xd6\xd0\xce\xc4'

## 实例

msg = "我爱北京"

print(msg.encode(encoding="utf-8")) #如果不指定编码格式，默认以系统的为准

print(msg.encode(encoding="utf-8").decode())

输出结果如下：

b'\xe6\x88\x91\xe7\x88\xb1\xe5\x8c\x97\xe4\xba\xac'

我爱北京