字符编码

因为计算机只能处理数字，如果要处理文本，就必须把文本转换为数字才能处理。最早的计算机在设计时采用8个比特(bit)作为一个字节(byte),所以一个字节所能表示的最大整数是255,如果要表示更大的整数，就必须用更多的字节。比如两个字节可以表示的最大的整数是**65535**，四个字节可以表示的最大的整数是**4294967295**。

由于计算机是美国人发明的，因此最早只有127个字符被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为**ASCII**编码表。

但是要处理中文显然一个字节是不够的，至少需要两个字符，而且还不能和**ASCII**冲突，所以中国制定可**GB2312**编码，用来把中文编进去。

你可以想得的是，世界上有上百种语言，日本把日文编到**Shift_JIS**,韩国把韩文编到**Euc-kr**里，各国有各国的标准，就会不可避免地出现冲突，结果就是在多语言混合的文体里，显示出来会有乱码。

因此，Unicode应运而生。Unicode把所有的语言统一到一套编码里，这样就不会出现乱码了。

Unicode的标准也在不断发展，但最常用的是两个字节表示一个字符(如果要用到非常偏僻的字符就需要四个字节)。现代操作系统和大多数语言都支持Unicode。

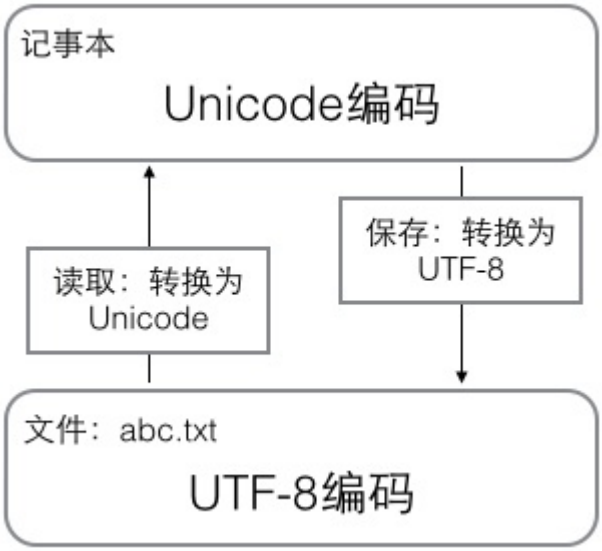
新的问题又出现：如果统一成Unicode编码，乱码问题从此消失，但是如果你写的文本基本是英文的话，用Unicode编码比ASCII编码要多一倍的空间，在存储和传输上就十分不划算。

所以，本着节约的精神，又出现了把Unicode编码转化为“可变长的编码”的UTF-8编码。UTF-8编码只是把一个Unicode字符根据不同的数字大小编成1-6个字节，常用的英文字母被编成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。但是你如果要传输的文本包含大量的英文字符的话，就用UTF-8编码就能节省空间：

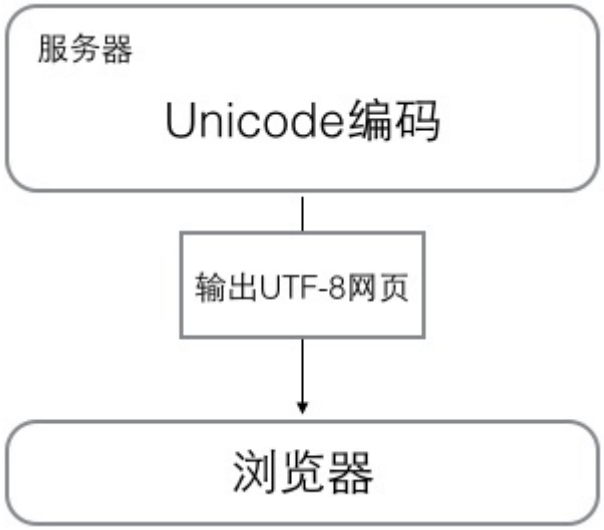
字符	ASCii	Unicode	UTF-8
A	01000001	00000000 01000001	01000001
中	x	01001110 00101101	11100100 10111000 10101101

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或传输时，就转换为UTF-8编码。

用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符存到内存里，保存的时候再把Unicode转换为UTF-8保存到文件：



浏览网页的时候，服务器会吧动态生成的Unicode内容转换为UTF-8在传输到浏览器：



所以你看到很多网页的源码上回有类似**meta charset="UTF-8"** 的信息，表示该网页正是使用的UTF-8编码。

Python的字符串

Python3中，字符串是以Unicode编码的，也就是说，Python3的字符串支持多种语言：

```
>>> print('包含中文的str')
包含中文的str
```

对于单个字符的编码，Python提供了**ord()** 函数获取字符的整数表示，**chr()** 函数把编码转换为相应的字符：

```
>>> ord('A')
65
>>> ord('中')
20013
>>> chr(66)
'B'
>>> chr(25991)
'文'
```

如果知道字符的整数编码，还可以用十六进制这么写str:

```
>>> '\u4e2d\u6587'
'中文'
```

由于Python的字符串类型是str,在内存中以Unicode表示，一个字符对应若干个字节。如果要在网络上传输，或者保存到磁盘上，就需要把str变成

以字节为单位的bytes。

Python对bytes类型的数据用带b前缀的单引号或双引号表示;

```
x = b'abc'
```

要注意区分'ABC'和b'ABC',前者是str,后者虽然内容显示和前者一样，但是bytes的每个字符都只占用一个字节。 以Unicode表示的str通过encode()方法可以编码为制定的bytes,例如：

```
>>> 'ABC'.encode('ascii')
b'ABC'
>>> 'ABC'.encode('Unicode')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    'ABC'.encode('Unicode')
LookupError: unknown encoding: Unicode
>>> LookupError: unknown encoding: Unicode
SyntaxError: invalid syntax
>>> 'ABC'.encode()
b'ABC'
>>> '中文'.encode('ascii')
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    '中文'.encode('ascii')
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1: ordinal not in range(128)
>>> '中文'.encode('utf-8')
b'\xe4\xb8\xad\xe6\x96\x87'
```

字节变字符decode()：

```
>>> b'ABC'.decode('ascii')
'ABC'
>>> b'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')
'中文'
```

要计算**str**包含多少个函数，可以使用**len()** 函数。

```
>>> len('ABC')
3
>>> len('中文')
2
```

len() 函数计算的是str的字符数，如果换成bytes,**len()** 函数就计算字节数

```
>>> len(b'ABC')
3
>>> len('中文'.encode('utf-8'))
6
```

由于Python源代码也是一个文件，所以，当你的源代码中包含中文的时候，在保存源代码时，就务必指定保存为UTF-8编码。当Python解释器读取源代码时，为了让它按UTF-8编码读取，我们通常在文件开头写上这两行：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

第一行注释是为了告诉Linux/OS X系统，这是一个Python可执行程序，Windows系统会忽略这个注释； 第二行注释是为了告诉python3解释器，按照UTF-8编码读取源代码，否则，你在源代码中写的中文可能会有乱码。

格式化

在Python中，采用的格式化方式和C语言中一致：

```
>>> 'Hello,%s' %'world'
'Hello,world'
>>> 'Hi,%s,you have $%d.' %('qxh',1000000)
'Hi,qxh,you have $1000000.'
```

常用的占位符有：

%d	整数
%f	浮点数
%s	字符串
%x	十六进制整数

如果你不太确定应该用什么，%s永远起作用，它会把任何数据类型转换为字符串：

```
>>> 'Age: %s. Gender:%s' %(25,True)
'Age: 25. Gender:True'
```

有些时候，字符串里面的%s是一个普通字符怎么办？这个时候就需要转义，用%%来表示一个%:

```
>>> 'growth rate: %d %%' %7
'growth rate: 7 %'
```

练习：

```
>>> s1 = 72
>>> s2 = 85
>>> r=(s2-s1)/s1 * 100
>>> print('小明成绩提高了%2.1f%%' %r)
小明成绩提高了18.1%
```