第 5 章 字符串与正则表达式

5.1 字 符 串

最早的字符串编码是美国标准信息交换码 ASCII，仅对 10 个数字、26 个大写英文字母、26 个小写英文字母及一些其他符号进行了编码。ASCII 采用 1 个字节来对字符进行编码，最多只能表示 256 个符号。

随着信息技术的发展和信息交换的需要，各国的文字都需要进行编码，不同的应用领域和场合对字符串编码的要求也略有不同，于是分别设计了不同的编码格式，常见的主要有 UTF-8、UTF-16、UTF-32、GB2312、GBK、CP936、base64、CP437 等。UTF-8 编码是国际通用的编码，以 1 个字节表示英语字符（兼容 ASCII），以 3 个字节表示中文，还有些语言的符号使用 2 个字节（如俄语和希腊语符号）或 4 个字节，UTF-8 对全世界所有国家需要用到的字符进行了编码。GB2312 是我国制定的中文编码，使用 1 个字节表示英语，2 个字节表示中文；GBK 是 GB2312 的扩充，而 CP936 是微软公司在 GBK 基础上开发的编码方式。GB2312、GBK 和 CP936 都是使用 2 个字节表示中文，UTF-8 使用 3 个字节表示中文。不同编码格式之间相差很大，采用不同的编码格式意味着不同的表示和存储形式，把同一字符存入文件时，写入的内容可能会不同，在理解其内容时必须了解编码规则并进行正确的解码。如果解码方法不正确就无法还原信息，从这个角度来讲，字符串编码也具有加密的效果。

Python 3.x 完全支持中文，使用 Unicode 编码格式，无论是一个数字、英文字母，还是一个汉字，都按一个字符对待和处理。例如，在 Python 3.5.1 中执行下面的代码，从代码中可以看到，在 Python 3.x 中甚至可以使用中文作为变量名。

```
>>>s='中国山东烟台'
>>>len(s)                          #字符串长度,或者包含的字符个数
6
>>>s='SDIBT'
>>>len(s)
5
>>>s='中国山东烟台 SDIBT'          #中文与英文字符同样对待,都算一个字符
>>>len(s)
11
>>>姓名='张三'                     #使用中文作为变量名
```

```
>>>年龄=40
>>>print(姓名)                          #输出变量的值
张三
>>>print(年龄)
40
```

小提示：在 Windows 平台上使用 Python 2.x 时，input()函数从键盘输入的字符串默认为 GBK 编码，而 Python 程序中的字符串编码则使用 # coding 显式地指定，常用的方式有：

```
#coding=utf-8
#coding:utf-8
#-*-coding:utf-8-*-
```

在 Python 中，字符串属于不可变序列类型，使用单引号（这是最常用的）、双引号、三单引号或三双引号作为界定符，并且不同的界定符之间可以互相嵌套。下面几种都是合法的 Python 字符串：

```
'abc'、'123'、'中国'、"Python"、'''Tom said,"Let's go"'''
```

除了支持序列通用方法（包括双向索引、比较大小、计算长度、元素访问、切片等操作）以外，字符串类型还支持一些特有的操作方法，如格式化、字符串查找、字符串替换（注意，不是原地替换）、排版等。但由于字符串属于不可变序列，不能直接对字符串对象进行元素增加、修改与删除等操作。另外，字符串对象提供的 replace()和 translate()方法也不是对原字符串直接进行修改替换，而是返回一个修改替换后的新字符串作为结果。

Python 支持短字符串驻留机制，对于短字符串，将其赋值给多个不同的对象时，内存中只有一个副本，多个对象共享该副本，与其他类型数具有相同的特点。然而，这一点并不适用于长字符串，长字符串不遵守驻留机制，下面的代码演示了短字符串和长字符串在这方面的区别。

```
>>>a='1234'
>>>b='1234'
>>>id(a)==id(b)                        #短字符串支持内存驻留机制
True
>>>a='1234'*50
>>>b='1234'*50
>>>id(a)==id(b)                        #长字符串不支持内存驻留机制
False
```

如果需要判断一个变量是否为字符串，可以使用内置方法 isinstance()或 type()。

```
>>>type('中国')
<class 'str'>
>>>type('中国'.encode('gbk'))          #编码成字节串，采用 GBK 编码格式
<class 'bytes'>
```

```
>>>bytes                                #bytes 是 Python 的内置类
<class 'bytes'>
>>>isinstance('中国', str)
True
>>>type('中国')==str
True
>>>type('中国'.encode())==bytes
True
>>>type('中国')==bytes
False
```

拓展知识：转义字符。如果大家学习过其他语言，应该了解转义字符的概念，可以跳过这部分内容。转义字符是指，在字符串中某些特定的符号前加一个斜线之后该字符将被解释为另外一种含义，不再表示本来的字符。常见的转义字符如表 5-1 所示。

表 5-1　常见的转义字符

转义字符	含　义
\b	退格，把光标移动到前一列位置
\f	换页符
\n	换行符
\r	回车
\t	水平制表符
\v	垂直制表符
\\	一个\
\'	单引号
\"	双引号
\ooo	3 位八进制数对应的字符
\xhh	2 位十六进制数对应的字符
\uhhhh	4 位十六进制数表示的 Unicode 字符

下面的代码演示了转义字符的用法：

```
>>>print('Hello\nWorld')             #包含转义字符的字符串
Hello
World
>>>oct(65)
'0o101'
>>>print('\101')                     #3 位八进制数对应的字符
A
>>>hex(65)
```

```
'0x41'
>>>print('\x41')                    #2 位十六进制数对应的字符
A
>>>ord('董')
33891
>>>hex(_)
'0x8463'
>>>print('\u8463')                  #4 位十六进制数表示的 Unicode 字符
董
```

5.1.1 字符串格式化的两种形式

如果需要将其他类型的数据转换为字符串，或者嵌入其他字符串或模板中再进行输出，就需要用到字符串格式化。Python 中字符串格式化的格式如图 5-1 所示，格式运算符%之前的部分为格式字符串，之后的部分为需要进行格式化的内容。

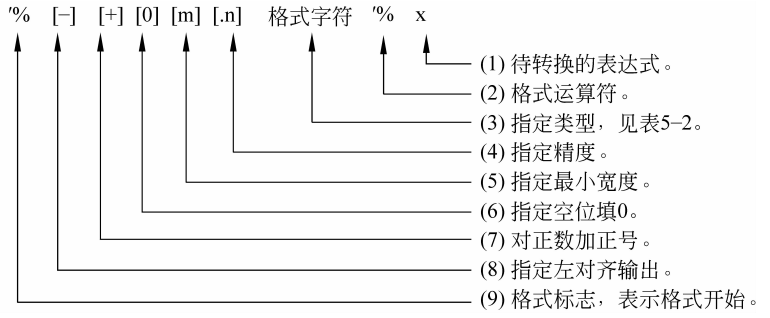

图 5-1 字符串格式化

Python 支持大量的格式字符，表 5-2 列出了比较常用的一部分。

表 5-2 格式字符

格式字符	说　明
%s	字符串(采用 str()的显示)
%r	字符串(采用 repr()的显示)
%c	单个字符
%b	二进制整数
%d	十进制整数
%i	十进制整数
%o	八进制整数
%x	十六进制整数
%e	指数(基底写为 e)

续表

格式字符	说　明
%E	指数(基底写为 E)
%f、%F	浮点数
%g	指数(e)或浮点数(根据显示长度)
%G	指数(E)或浮点数(根据显示长度)
%%	字符"%"

下面的代码演示了字符串格式化的用法：

```
>>>x=1235
>>>so="%o" %x
>>>so
'2323'
>>>sh="%x" %x
>>>sh
'4d3'
>>>se="%e" %x
>>>se
'1.235000e+03'
>>>"%s"%65                          #等价于 str()
'65'
>>>"%s"%65333
'65333'
>>>'%d,%c'%(65, 65)                 #使用元组对字符串进行格式化,按位置进行对应
'65,A'
>>>"%d"%"555"                       #试图将字符串转换为整数进行输出,抛出异常
Traceback(most recent call last):
  File "<pyshell#19>", line 1, in <module>
    "%d"%"555"
TypeError: %d format: a number is required, not str
>>>int('555')                       #可以使用 int()函数将合法的数字字符串转换为整数
555
>>>'%s'%[1, 2, 3]
'[1, 2, 3]'
>>>str((1, 2, 3))                   #可以使用 str()函数将任意类型数据转换为字符串
'(1, 2, 3)'
>>>str([1, 2, 3])
'[1, 2, 3]'
```

除了上面介绍的字符串格式化方法，目前 Python 社区更推荐使用 format()方法进行格式化，该方法更加灵活，不仅可以使用位置进行格式化，还支持使用与位置无关的参数名字来进行格式化，并且支持序列解包格式化字符串，为程序员提供了非常大的方便。

例如：

```
>>>print('{0:.3f}'.format(1/3))      #保留 3 位小数
0.333
>>>1/3
0.3333333333333333
>>>print("The number {0:,} in hex is: {0:#x}, in oct is {0:#o}".format(55))
The number 55 in hex is: 0x37, in oct is 0o67
>>>print ("The number {0:,} in hex is: {0:x}, the number {1} in oct is {1:o}".
format(5555, 55))
The number 5,555 in hex is: 15b3, the number 55 in oct is 67
>>>print ("The number {1} in hex is: {1:#x}, the number {0} in oct is {0:#o}".
format(5555, 55))
The number 55 in hex is: 0x37, the number 5555 in oct is 0o12663
>>>print("my name is {name}, my age is {age}, and my QQ is {qq}".format(name="
Dong", qq="306467355", age=38))
my name is Dong, my age is 38, and my QQ is 306467355
>>>position = (5, 8, 13)
>>>print("X:{0[0]};Y:{0[1]};Z:{0[2]}".format(position))
                                        #使用元组同时格式化多个值
X:5;Y:8;Z:13
>>>weather=[("Monday", "rain"),("Tuesday", "sunny"),("Wednesday", "sunny"),
("Thursday", "rain"),("Friday", "Cloudy")]
>>>formatter="Weather of '{0[0]}' is '{0[1]}'".format
>>>for item in map(formatter, weather):
    print(item)
```

上面最后一段代码也可以改为下面的写法：

```
>>>for item in weather:
    print(formatter(item))
```

运行结果为

```
Weather of 'Monday' is 'rain'
Weather of 'Tuesday' is 'sunny'
Weather of 'Wednesday' is 'sunny'
Weather of 'Thursday' is 'rain'
Weather of 'Friday' is 'Cloudy'
```

拓展知识：在字符串格式化方法 format()中常用的格式字符。在字符串格式化方法 format()中可以使用的格式主要有 b(二进制格式)、c(把整数转换成 Unicode 字符)、d(十进制格式)、o(八进制格式)、x(小写十六进制格式)、X(大写十六进制格式)、e/E(科学计数法格式)、f/F(固定长度的浮点数格式)、%(使用固定长度浮点数显示百分数)。

拓展知识：Python 标准库 string 还提供了用于字符串格式化的模板类

Template。例如：

```
>>>from string import Template
>>>t=Template('My name is ${name}, and is ${age} years old.')
                                    #创建模板
>>>d={'name':'Dong', 'age':39}
>>>t.substitute(d)                  #替换
'My name is Dong, and is 39 years old.'
>>>tt=Template('My name is $name, and is $age years old.')
>>>tt.substitute(d)
'My name is Dong, and is 39 years old.'
```

5.1.2　字符串常用方法

1. find()、rfind()、index()、rindex()、count()

find()和 rfind()方法分别用来查找一个字符串在另一个字符串指定范围(默认是整个字符串)中首次和最后一次出现的位置，如果不存在则返回－1；index()和 rindex()方法用来返回一个字符串在另一个字符串指定范围中首次和最后一次出现的位置，如果不存在则抛出异常；count()方法用来返回一个字符串在另一个字符串中出现的次数，如果不存在则返回 0。

```
>>>s="apple,peach,banana,peach,pear"
>>>s.find("peach")                  #返回第一次出现的位置
6
>>>s.find("peach", 7)               #从指定位置开始查找
19
>>>s.find("peach", 7, 20)           #在指定范围中进行查找
-1
>>>s.rfind('p')                     #从字符串尾部向前查找
25
>>>s.index('p')                     #返回首次出现的位置
1
>>>s.index('pe')
6
>>>s.index('pear')
25
>>>s.index('ppp')                   #指定子字符串不存在时抛出异常
Traceback(most recent call last):
  File "<pyshell#11>", line 1, in <module>
    s.index('ppp')
ValueError: substring not found
>>>s.count('p')                     #统计子字符串出现的次数
5
```

```
>>>s.count('pp')
1
>>>s.count('ppp')                                    #不存在时返回 0
0
```

拓展知识：实际开发时应优先考虑使用 Python 内置函数和内置对象的方法，运行速度快，并且运行稳定。例如，下面的代码用来检查长字符串中哪些位置上的字母是a，通过运行结果可以发现，使用字符串方法 find()的速度明显要比逐个字符比较快很多。

```
from string import ascii_letters
from random import choice
from time import time

letters=''.join([choice(ascii_letters)for i in range(999999)])
def positions_of_character(sentence, ch):    #使用字符串对象的 find()方法
    result=[]
    index=0
    index=sentence.find(ch, index+1)
    while index !=-1:
        result.append(index)
        index=sentence.find(ch, index+1)
    return result

def demo(s, c):                                    #普通方法，逐个字符比较
    result=[]
    for i,ch in enumerate(s):
        if ch ==c:
            result.append(i)
    return result

start=time()
positions=positions_of_character(letters, 'a')
print(time()-start)

start=time()
p=demo(letters, 'a')
print(time()-start)
```

运行结果如下：

```
0.009000539779663086
0.08400487899780273
```

速度居然相差 10 倍左右，看来内置对象提供的方法还真是不错，简直是人见人爱，花见花开。但是不要高兴太早，一切都是相对的，这世间没有绝对得好，也没有绝对得坏，内

置对象的某些方法也不是在任何场合都能保证最优。例如把上面代码中的

```
letters=''.join([choice(ascii_letters)for i in range(999999)])
```

改为

```
letters=''.join([choice('ab')for i in range(999999)])
```

然后再次运行，会发现结果与上面的代码恰好相反，逐个比较的方法又比使用 find()方法快了很多。稍加分析可以发现，上面两段代码是完全一样的，只是所查找数据的密度不一样，处理速度却有着翻天覆地的变化。所以说，首先要分析待处理的数据有什么样的特点（包括组成元素、分布情况等），然后才能设计最优的算法并采用最高效的方法。但一般情况下，Python 内置函数、内置对象的方法和标准库对象的效率要高于自己编写的代码。

2. split()、rsplit()、partition()、rpartition()

split()和 rsplit()方法分别用来以指定字符为分隔符，从字符串左端和右端开始将其分隔成多个字符串，并返回包含分隔结果的列表；partition()和 rpartition()用来以指定字符串为分隔符将原字符串分隔为3部分，即分隔符之前的字符串、分隔符字符串和分隔符之后的字符串，如果指定的分隔符不在原字符串中，则返回原字符串和两个空字符串。

```
>>>s="apple,peach,banana,pear"
>>>li=s.split(",")                          #使用逗号进行分隔
>>>li
["apple", "peach", "banana", "pear"]
>>>s.partition(',')                         #从左侧使用逗号进行切分
('apple', ',', 'peach,banana,pear')
>>>s.rpartition(',')                        #从右侧使用逗号进行切分
('apple,peach,banana', ',', 'pear')
>>>s.rpartition('banana')                   #使用字符串作为分隔符
('apple,peach,', 'banana', ',pear')
>>>s="2014-10-31"
>>>t=s.split("-")                           #使用指定字符作为分隔符
>>>t
['2014', '10', '31']
>>>list(map(int, t))                        #将分隔结果转换为整数
[2014, 10, 31]
```

对于 split()和 rsplit()方法，如果不指定分隔符，则字符串中的任何空白符号（包括空格、换行符、制表符等）的连续出现都将被认为是分隔符，返回包含最终分隔结果的列表。

```
>>>s='hello world \n\n My name is Dong   '
>>>s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
>>>s='\n\nhello world \n\n\n My name is Dong   '
```

```
>>>s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
>>>s='\n\nhello\t\t world \n\n\n My name\t is Dong   '
>>>s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
```

另外，split()和 rsplit()方法还允许指定最大分隔次数（注意，不是必须分隔这么多次），例如：

```
>>>s='\n\nhello\t\t world \n\n\n My name is Dong   '
>>>s.split(maxsplit=1)                    #分隔 1 次
['hello', 'world \n\n\n My name is Dong   ']
>>>s.rsplit(maxsplit=1)
['\n\nhello\t\t world \n\n\n My name is', 'Dong']
>>>s.split(maxsplit=2)
['hello', 'world', 'My name is Dong   ']
>>>s.rsplit(maxsplit=2)
['\n\nhello\t\t world \n\n\n My name', 'is', 'Dong']
>>>s.split(maxsplit=5)
['hello', 'world', 'My', 'name', 'is', 'Dong   ']
>>>s.split(maxsplit=6)
['hello', 'world', 'My', 'name', 'is', 'Dong']
>>>s.split(maxsplit=10)              #最大分隔次数大于实际可分隔次数时，自动忽略
['hello', 'world', 'My', 'name', 'is', 'Dong']
```

小提示：调用 split()方法并且不传递任何参数时，将使用任何空白字符作为分隔符，如果字符串存在连续的空白字符，split()方法将自动忽略；明确传递参数指定 split()使用的分隔符时，情况略有不同。

```
>>>'a,,,bb,,ccc'.split(',')              #每个逗号都被作为独立的分隔符
['a', '', '', 'bb', '', 'ccc']
>>>'a\t\t\tbb\t\tccc'.split('\t')        #每个制表符都被作为独立的分隔符
['a', '', '', 'bb', '', 'ccc']
>>>'a\t\t\tbb\t\tccc'.split()           #连续多个制表符被作为一个分隔符
['a', 'bb', 'ccc']
```

3. join()

与 split()相反，join()方法用来将列表中多个字符串进行连接，并在相邻两个字符串之间插入指定字符。

```
>>>li=["apple", "peach", "banana", "pear"]
>>>sep=","
>>>s=sep.join(li)                         #使用逗号作为连接符
>>>s
```