第 5 章 字符串与正则表达式

5.1 字符串

字符串属于不可变的有序组合数据类型，使用单引号、双引号、三单引号或者三双引号作为定界符，不同的定界符可以相互嵌套。Python 中没有独立的字符数据类型，一个字符也是字符串。

5.1.1 字符串的基本操作

1. 创建

创建字符串很简单，只要为变量分配一个值即可。例如：

```
>>> s1='SuZhou'          # 单引号
>>> s2="SuZhou"          # 双引号
>>> s3='''SuZhou'''      # 三单引号
>>> s4="""SuZhou"""      # 三双引号
>>> type(s1)
<class'str'>
>>> s5=''                # 空字符串
```

也可以使用不带参数的 str( ) 函数创建空字符串。例如：

```
>>> s6=str()             # str() 函数创建空字符串
>>> s6
''
```

通过 str( ) 函数也可以把任意对象转换为 str 对象。注意，其他组合类型转换为字符串类型的结果，分隔符后会自动产生空格。例如：

```
>>> s7=str(12345)
>>> s7
'12345'
>>> s8=[1,2,3,4,5]
>>> str(s8)              # str() 函数将列表数据中的方括号、逗号、空格都转换为结果字符
'[1,2,3,4,5]'
>>> s9={'a':1,'b':2}
>>> str(s9)
"{'a':1,'b':2}"
```

另外，若有两个紧邻的字符串，中间以空格分隔，则自动拼接成一个字符串。例如：

```
>>>'SuZhou' 'China'
'SuZhouChina'
>>> x='12'   '34'
>>> x
'1234'
```

2. 索引

字符串也是有序对象，支持使用索引访问指定位置的元素（字符）。例如：

```
>>> s1='SuZhou'
>>> s1[0]
'S'
```

字符串同样支持双向索引。例如：

```
>>> s1[-1]
'u'
```

注意：字符串属于不可变对象，所以不可以修改、添加、删除元素。例如：

```
>>> s1[2]='C'
Traceback(most recent call last):
  File "<pyshell#451>",line 1,in <module>
    s1[2]='C'
TypeError:'str' object does not support item assignment
```

3. 删除

与列表、元组一样，当不再使用字符串对象时，可以使用 del 命令删除整个字符串，但因字符串是不可变数据类型，不可以用 del 命令删除字符串中的部分字符。例如：

```
>>> s1='SuZhou'
>>> del s1[2]
Traceback(most recent call last):
  File "<pyshell#46>",line 1,in <module>
    del s1[2]
TypeError:'str' object doesn't support item deletion
>>> del s1                  # 删除整个字符串
>>> s1                      # s1 已被删除，访问时会抛出异常
Traceback(most recent call last):
  File "<pyshell#454>",line 1,in <module>
    s1
NameError:name 's1' is not defined
```

4. 切片

切片操作同样适用于字符串对象，但只能使用切片操作获取字符串中的元素，不能做任何添加、修改和删除元素的操作。例如：

```
>>> s1="SuZhou,China"
>>> s1[:7]
```

```
'SuZhou,'
>>> s1[:6]
'SuZhou'
>>> s1[8:]
'hina'
>>> s1[7:]
'China'
>>> s1[::2]
'SZo,hn'
```

5. 遍历

字符串是有序序列，遍历字符串中的元素有利用索引遍历和直接遍历两种方式。例如：

```
>>> s1='SuZhou'
'SuZhou'
>>> for i in range(len(s1)):                    # 根据字符串长度遍历，使用索引引用字符
    print(s1[i],end=' ')
S  u  Z  h  o  u
>>> for ch in s1:                               # 直接遍历用 in 引用字符
    print(ch,end=' ')
S  u  Z  h  o  u
```

6. 适用的运算符

运算符 +、+=、*、*=、in、<、<=、>、>=、==、!= 均可应用于字符串。+、+= 运算符用于字符串时，表示字符串的连接运算。*、*= 运算符用于重复字符串。成员测试符 in 则用于测试字符串中是否包含某个字符或字符串。例如：

```
>>> s1="SuZhou"
>>> s2="China"
>>> s1+s2                                       # 字符串连接
'SuZhouChina'
>>> s1+=s2                                      # 字符串连接，将结果赋予 s1
>>> s1
'SuZhouChina'
>>> s2*3
'ChinaChinaChina'
>>> s2 in s1                                    # 判断 s2 是否出现在 s1 中
True
>>>'w' in s1                                   # 判断字符 w 是否出现在 s1 中
False
```

<、<=、>、>=、==、!= 用于比较字符串的大小，确定字符串的大小时逐个字符比较大小，当被比较的两个字符能区分大小时，结束比较；若一直不能区分大小，且字符个数相等，则结果为相等，否则，字符个数多的字符串大，字符个数少的字符串小。字符的大小由其编码确定，例如，西文字符以字符对应的 ASCII 码值确定大小。以下是与字符串相关的运算。

```
>>> s1="SuZhou"
>>> s2="China"
>>> s1>s2                          # 关系运算符用于比较两个字符串的大小
True
>>> s1>"SuZhouT"                   # 若前面字符都相等，则长大短小
False
```

7. 内置函数对字符串的操作

字符串是一个由字符组成的组合型数据，因此 4.1.2 节中介绍的适用于组合数据类型的内置函数，除了 sum( ) 外都可以用于字符串。例如：

```
>>> s1="SuZhou"
>>> len(s1)                        # 求 s1 的长度（字符个数）
6
>>> max(s1)                        # 求 s1 中的最大元素（字符）
'u'
>>> min(s1)                        # 求 s1 中的最小元素（字符）
'S'
>>> sorted(s1)                     # 对 s1 中的元素（字符）排序，默认为升序
['S','Z','h','o','u','u']
>>> list(reversed(s1))             # 对 s1 中的元素（字符）逆序
['u','o','h','Z','u','S']
>>> list(zip(s1,"China"))          # 使用 list() 方法将 zip 对象转换为列表
[('S','C'),('u','h'),('Z','i'),('h','n'),('o','a')]
>>> list(enumerate(s1))            # 使用 list() 方法将 enumerate 对象转换为列表
[(0,'S'),(1,'u'),(2,'Z'),(3,'h'),(4,'o'),(5,'u')]
>>> all(s1)                        # 判断 s1 中所有元素（字符）是否都等价于 True
True
>>> any(s1)                        # 判断 s1 中是否有元素（字符）等价于 True
True
```

当所有元素都是字符串的两个列表或元组比较大小时，按索引次序依次比较相同索引的元素大小，即比较同一位置上的字符串大小。例如：

```
>>> list1=['123','9','67','3962']
>>> list1>['123','821']
True
```

上述比较运算中，首先比较两个列表的第一个元素 '123' 和 '123'，无法比出大小，就再比较 '9' 和 '821'，字符串的大小需要逐个字符进行比较，因为 '9' 的编码大于 '8' 的编码，可得 '9' 大于 '821'，因此 list1 大于 ['123','821']。

max( )、min( )、sorted( ) 函数的结果也采用类似的处理方式。例如：

```
>>> list1=['123','9','67','3962']
>>> max(list1)
'9'
>>> min(list2)
'123'
```

```
>>> sorted(list2)
['123','3962','67','9']
```

另外，内置函数 eval( ) 可将字符串转换为表达式并进行求值运算。例如：

```
>>> s1="3+4"
>>> eval(s1)                          # 将 s1 转换为表达式 3+4，并求出值
7
>>> s2=eval(s1)*2
>>> s2
14
>>> x=10
>>> y=20
>>> eval("x*y")                       # 将 s1 转换为表达式 x*y，并求出值
200
```

与字符串相关的内置函数还有 ord( ) 和 chr( )，ord( ) 可将字符转换为一个整数，chr( ) 则将一个整数转换为字符。例如：

```
>>> ord('A')                          # 转换结果是 'A' 字符在 ASCII 码表中的编码值
65
>>> chr(65)
'A'
>>> chr(ord('A')+32)                  # 可以将大写字符 'A' 转换为小写字符 'a'
'a'
>>> ord('苏')                         # 汉字的编码是 UTF-8，见 5.1.2 节的介绍
33487
>>> chr(33487)
'苏'
>>> chr(33488)
'茏'
>>> ord('苏州')                       # ord() 函数只能处理单个字符
Traceback(most recent call last):
  File "<pyshell#91>",line 1,in <module>
    ord('苏州')
TypeError:ord() expected a character,but string of length 2 found
```

5.1.2 字符串编码

最早的字符编码是 ASCII（美国标准信息交换码），共编码了 128 个字符（包括 10 个数字、26 个大写字母、26 个小写字母及一些控制符）。ASCII 采用 1 字节对字符进行编码，其编码字符仅考虑了当时美国的需求。

随着计算机的发展与普及，各国的文字都需要编码才能进行信息交换。于是各个国家或地区也都纷纷设计了自己的文字编码。目前国内常用的编码有扩充 ASCII、GB 2312、GBK、UTF-8 等。不同的编码规则意味着字符的不同表示和存储形式，因此，即使同一个字符，因使用的编码不一样，存入计算机时其内容也可能不一样。

GB 2312 是我国最早制定的中文编码标准，GBK（K 表示扩充）是对 GB 2312 的扩

充。这两种编码的中文汉字都采用 2 字节表示。UTF-8 也称万国码，是针对 Unicode 的一种编码方式。UTF-8 为了节省资源，采用变长编码，编码长度为 1~6 字节不等。对于中文字符，UTF-8 则使用 3 字节来表示。

Python 3.x 默认采用 UTF-8 编码格式，能支持中文，无论是一个数字、英文字符，还是汉字，都按一个字符来处理。如果想使用其他编码方式，可以使用字符串的 encode( ) 方法来指定。encode( ) 编码后得到的是 bytes 对象，而 bytes 对象可以通过 decode( ) 方法按对应的编码格式解码为 str 字符串。bytes 是一种特殊的字符类型，称为字节串，其每个元素是一个 8 位二进制数据。创建字节串时，只要在字符串前面加上字母 b 即可。例如：

```
>>> s1="SuZhou"                          # 创建字符串对象 s1
>>> b1=b"SuZhou"                         # 创建字节串对象 b1
>>> type(s1)                             # str 类型
<class'str'>
>>> type(b1)                             # bytes 类型
<class'bytes'>
>>> len(b1)
6
```

中文字符串的长度是所包含汉字字符的个数。例如：

```
>>> s2=" 苏州 "                          # 创建字符串对象
>>> len(s2)
2
>>> b2=s2.encode()                       # 默认使用 UTF-8 对汉字编码，每个汉字 3 字节
>>> b2
b'\xe8\x8b\x8f\xe5\xb7\x9e'
>>> len(b2)
6
>>> b2.decode()                          # 使用默认的 UTF-8 解码
' 苏州 '
```

以上 b2 中的 \x 是转义符标志，例如 \xe8 指字节内容为十六进制的 E8，连续的每 3 字节构成 1 个汉字字符。如果用 GBK 编码，则是 2 字节构成一个汉字。例如：

```
>>> " 苏州 ".encode('gbk')               # 使用 GBK 对汉字编码，每个汉字 2 字节
b'\xcb\xd5\xd6\xdd'
>>> b'\xcb\xd5\xd6\xdd'.decode()         # 默认使用 UTF-8 解码，会抛出异常
Traceback(most recent call last):
  File "<pyshell#472>",line 1,in <module>
    b'\xcb\xd5\xd6\xdd'.decode()
UnicodeDecodeError:'utf-8' codec can't decode byte 0xcb in position
0:invalid continuation byte
>>> b'\xcb\xd5\xd6\xdd'.decode('gbk')        # 使用 GBK 解码则正确
' 苏州 '
```

另外还需要注意，中文字符串前面不能加前缀 b。例如：

```
>>> b'苏州'
SyntaxError:bytes can only contain ASCII literal characters.
```

5.1.3 字符串的方法

Python 字符串提供了大量的方法，可以应用这些方法进行字符串的查找、替换和排版等操作。由于字符串属于不可变对象，因此只要涉及字符串修改的方法都返回修改后的新字符串，而原字符串是不会有任何改动的，都是非原地操作。

字符串对象的常用方法如表 5-1 所示。

表 5-1 字符串对象的常用方法

方 法	功 能 说 明
str.capitalize( )	将字符串 str 的第一个字母变成大写，其他字母都变成小写
str.center(width[,fillchar])	返回一个宽度为 width 且原字符串 str 居中的字符串。fillchar 为填充字符，默认为空格
str.count(sub,start=0,end=len(string))	统计字符串 str 里某个子串 sub 出现的次数。可选参数 start 和 end 为在字符串中搜索的开始与结束位置
str.encode(encoding='UTF-8',errors='strict')	以指定的编码格式编码字符串。参数 errors 可以指定不同的错误处理方案
str.find(str1,beg=0,end=len(string))	返回子字符串 str1 在 str 中的起始索引位置。参数 beg（开始）和 end（结束，不含 end）可以指定范围。无论是否使用参数 beg 和 end，返回的都是整个字符串中的起始位置。如果不包含子字符串 str1，则返回 −1
str.rfind(str1,beg=0, end=len(string))	返回字符串 str1 最后一次出现的位置，如果没有匹配项则返回 −1。可以指定可选参数 [beg:end] 设置查找区间
str.index(str1,beg=0,end=len(string))	返回字符串 str1 出现的位置，可以指定可选参数 [beg:end] 设置查找区间。该方法与 find() 方法一样，只是如果 str1 不在 str 中则会报一个异常
str.rindex(str1,beg=0, end=len(string))	返回子字符串 str1 最后出现的位置，如果没有匹配的字符串会报异常。可以指定可选参数 [beg:end] 设置查找区间
str.isalnum( )	检测字符串是否由字母和数字组成
str.isalpha( )	检测字符串是否只由字母组成
str.isdigit( )	检测字符串是否只由数字组成
str.islower( )	检测字符串是否只由小写字母组成
str.isspace( )	检测字符串是否只由空白字符组成
str.isupper( )	检测字符串中所有的字母是否都为大写
str.join(sequence)	将组合数据类型 sequence 中的字符串元素，以指定的字符串 str 连接生成一个新的字符串
str.ljust(width[,fillchar])	返回一个原字符串 str 左对齐，并使用空格填充至指定长度 width 的新字符串。参数 fillchar 可以指定其他填充字符。如果指定的长度小于原字符串的长度则返回原字符串

续表

方　法	功 能 说 明
str.rjust(width[,fillchar])	返回一个原字符串 str 右对齐，并使用空格填充至长度 width 的新字符串。参数 fillchar 可以指定其他填充字符。如果指定的长度小于字符串的长度则返回原字符串
str.lower( )	将字符串中所有字母转换为小写
str.upper( )	将字符串中所有字母转换为大写
str.title( )	将所有单词的首字母转换为大写，其余字母均为小写
str.swapcase( )	对字符串的大小写字母进行互换
str.replace(old,new[,max])	把字符串 str 中的 old（旧子字符串）替换成 new（新子字符串），如果指定第三个参数 max，则替换不超过 max 次
str.split(str1="",num=str.count(str))	通过指定分隔符 str1 对字符串进行从左分隔并返回包含结果的列表。如果参数 num 有指定值，则最多仅分隔为 num+1 个子字符串
str.rsplit(str1="",num=str.count(str))	通过指定分隔符 str1 对字符串进行从右分隔并返回包含结果的列表。如果参数 num 有指定值，则最多仅分隔为 num+1 个子字符串
str.splitlines([keepends=False])	按照行 ('\r','\r\n',\n') 分隔，返回一个包含各行作为元素的列表。参数 keepends 默认值为 False，结果子串不包含换行符；如果值为 True，则保留换行符
str.startswith(substr,beg=0,end=len(string))	检查字符串是否以指定子字符串 substr 开头，如果是则返回 True，否则返回 False。如果参数 beg 和 end 有指定值，则在指定范围内检查
str.endswith(suffix[,start[,end]])	判断字符串是否以指定后缀结尾，如果以指定后缀结尾则返回 True，否则返回 False。可选参数 start 与 end 为检索字符串的开始与结束位置
str.strip([chars])	移除字符串头尾指定的字符（默认为空白字符）
str.lstrip([chars])	移除字符串左边的指定字符（默认为空白字符）
str.rstrip([chars])	移除字符串末尾的指定字符（默认为空白字符）
str.zfill(width)	返回指定长度的字符串，原字符串右对齐，前面填充 0
str.maketrans(intab,outtab)	返回一个字符映射的转换表。第一个参数 intab 是字符串，表示需要转换的字符；第二个参数 outtab 也是字符串，表示转换的目标
str.translate(table)	根据参数 table 给出的字符映射表转换字符串 str 中的字符，table 由 maketrans() 方法创建
str.format(…)	返回按照给定参数进行格式化后的字符串副本

下面具体介绍一些常用方法的用法。

1. lower( )、upper( )、title( )、capitalize( )、swapcase( )

这几个方法主要用来对字符串中的字母进行各种大小写的转换，都返回新字符串，而不会对原字符串进行任何修改。例如：

```
>>> s1="Soochow University is Beautiful"
>>> s1.lower()                          # 返回全部的小写字符串
'soochow university is beautiful'
>>> s1.upper()                          # 返回全部的大写字符串
'SOOCHOW UNIVERSITY IS BEAUTIFUL'
>>> s1.title()                          # 返回每个单词首字母变为大写的字符串
'Soochow University Is Beautiful'
>>> s1.capitalize()                     # 返回首字母变为大写的字符串
'Soochow university is beautiful'
>>> s1.swapcase()                       # 返回大小写字母互换后的字符串
'sOOCHOW uNIVERSITY IS bEAUTIFUL'
```

2. strip( )、lstrip( )、rstrip( )

这三个方法分别用来删除字符串两端、左端和右端连续的空白字符或指定字符，都返回新字符串，而不是原地操作。以 lstrip( ) 为例，删除时，从左端第 1 个字符开始，判断该字符是否为要删除字符，如果是则继续判断下一个字符，如果一直是，则一直删除，直到遇到一个不是要删除的字符为止。例如：

```
>>> s1="    Soochow University is Beautiful    "
>>> s1.strip()                          # 删除两端空白字符
'Soochow University is Beautiful'
>>> s1="\t\nSoochow University is Beautiful\t\n"
>>> s1.strip()                          # \t(Tab) 和 \n( 换行 ) 也被认为是空白字符
'Soochow University is Beautiful'
>>> s1="__:Soochow University is Beautiful:__"
>>> s1.strip("_")                       # 删除两端指定字符 _
':Soochow University is Beautiful:'
>>> s1.strip(":_")                      # 删除两端指定字符 : 和 _
'Soochow University is Beautiful'
>>> s1.lstrip(":_")                     # 删除左端指定字符 : 和 _
'Soochow University is Beautiful:__'
>>> s1.rstrip("_")                      # 删除右端指定字符 _
'__:Soochow University is Beautiful:'
```

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

字符串对象的 find( ) 和 index( ) 方法都是用来查找一个字符串在另一个字符串指定范围中首次出现的位置，区别在于当查询结果不存在该字符串时，find( ) 返回 −1，而 index( ) 则抛出异常。rfind( )、rindex( ) 和 find( )、index( ) 类似，区别只在于这两个方法查找的是最后出现的位置。count( ) 方法用来统计并返回一个字符串在另一个字符串中出现的次数，如果没有出现，则返回 0。例如：

```
>>> s1="Because had because,so had so"
>>> s1.find("a")            # s1 中第一次出现 a 的下标位置
3
>>> s1.find("had")          # s1 中第一次出现 had 的下标位置
8
```

```
>>> s1.find("had",11)       # 从下标位置 11 开始查找 s1 中第一次出现 had 的位置
24
>>> s1.find("had",11,22)    # 在 s1 下标范围 [11,22) 中没有查到 had
-1
>>> s1.rfind("had")         # s1 中最后一次出现 had 的下标位置，即从后向前查找
24
>>> s1.rfind("had",5,15)    # 在 s1 下标范围 [5,15) 中查找中最后一次出现 had 的位置
8
>>> s1.index("had")          # s1 中第一次出现 had 的下标位置
8
>>> s1.rindex("had")         # s1 中最后一次出现 had 的下标位置
24
>>> s1.index("have")         # s1 中找不到 have，抛出异常
Traceback(most recent call last):
  File "<pyshell#655>",line 1,in <module>
    s1.index("have")
ValueError:substring not found
>>> s1.count("a")            # 统计 a 在 s1 中出现的次数
4
>>> s1.count("so")           # 统计 so 在 s1 中出现的次数
2
>>> s1.count("have")         # 统计 have 在 s1 中出现的次数，若不存在则返回 0
0
>>> s1.count('so',11,22)     # 从下标位置 [11,22) 范围中统计 so 出现的次数
0
>>> s1.count('so',22)        # 从下标位置 22 开始统计 so 出现的次数
1
```

4. split( )、rsplit( )、splitlines( )

字符串对象的 split( ) 方法用指定分隔符对字符串从左往右分隔，返回包含结果的列表，且可以指定最大分隔次数。例如：

```
>>> s1="Because had because,so had so"
>>> s1.split(",")            # 使用逗号分隔
['Because had because','so had so']
```

如果不指定分隔符，则默认使用空白字符（包括空格、换行、制表符等）的连续出现作为分隔符，连续出现是指多个连续的空白字符作为一个分隔符。例如：

```
>>> s1="Because had because,so had so"
>>> s1.split()               # 默认使用空白字符分隔
['Because','had','because,','so','had','so']
>>> s2="\nBecause\nhad\nbecause,\t\t\n\nso\nhad\nso\n"
>>> s2.split()               # 默认使用空白字符分隔，逗号后的 \t\t\n\n 是一个分隔符
['Because','had','because,','so','had','so']
>>> s2.split('\n')           # 使用 \n 作为分隔符，结果会多出三个空字符串 ''
['','Because','had','because,\t\t','','so','had','so','']
```

在分隔的同时，还可以指定最大分隔次数，分隔后的子字符串为最大分隔次数 +1，