字符串处理

当Mathematica和其他软件协作时，经常会用文件传输数据，或者通过命令行参数调用其它程序。在这种过程中，字符串的各种处理是经常用到的。
比如将Mathematica的表达式转换为其它程序的命令行参数，或者将输入或者输出调整成适当的格式等等。

In [ ]:

```
StringLength[{"我是一个字符串！", "I am a string!"}]
```

In [1]:

```
StringJoin["《一个人来到田纳西》\n\n", "毫无疑问\n", "我做的牛肉饼\n", "是全天下\n", "最好吃的"]
StringJoin[{"《一个人来到田纳西》\n\n", {"毫无疑问\n", {"我做的牛肉饼\n"}, "是全天下\n"},
  "最好吃的"}]
"《一个人来到田纳西》\n\n" <> "毫无疑问\n" <> "我做的牛肉饼\n" <> "是全天下\n" <> "最好吃的"
```

Out[1]:

《一个人来到田纳西》

毫无疑问
我做的牛肉饼
是全天下
最好吃的
《一个人来到田纳西》

毫无疑问
我做的牛肉饼
是全天下
最好吃的
《一个人来到田纳西》

毫无疑问
我做的牛肉饼
是全天下
最好吃的

In [4]:

```
alphabet = StringJoin[CharacterRange["a", "z"]]
StringTake[alphabet, 12]
StringDrop[alphabet, 12]
StringPart[alphabet, 12]
```

Out[4]:

```
abcdefghijklmnopqrstuvwxyz
abcdefghijkl
mnopqrstuvwxyz
l
```

In [8]:

```
StringTake[alphabet, {12, 16}]
StringPart[alphabet, {12, 16}]
```

Out[8]:

```
lmnop
{l, p}
```

- 字符串模式:

In [ ]:

```
MatchQ[{a, b, c, d}, {___, x_, x_, ___}]
MatchQ[{a, b, c, c, d}, {___, x_, x_, ___}]
```

In [ ]:

```
StringMatchQ["abcd", ___ ~~ x_ ~~ x_ ~~ ___]
StringMatchQ["abccd", ___ ~~ x_ ~~ x_ ~~ ___]
```

In [ ]:

```
StringFreeQ["aabbccdd", "bc" ~~ x_ ~~ "d"]
```

In [ ]:

```
StringCases["aabbccdd", x_ ~~ x_]
StringCases["aabbccdd", x_ ~~ x_ :> x ~~ x ~~ x]
```

In [ ]:

```
StringPosition["aabbccdd", x_ ~~ x_ ~~ y_ ~~ y_]
```

In [ ]:

```
StringCount["abcadcadcbqwertaac", "a"]
StringCount["abcadcadcbqwertaac", "a" ~~ _ ~~ "c"]
```

In [ ]:

```
StringReplace["abbaabbaa", "ab" -> "X"]
StringReplace["ababbabbaaababa", "ab" .. -> "X"]
```

In [ ]:

```
StringReplaceList["ccccc", "c" -> "XYX"]
StringReplaceList["abcdeabacde", {"abc" -> "X", "cde" -> "Y"}]
StringReplace["the cat in the hat", Except[Characters["aeiou"]] -> ""]
```

In [ ]:

```
StringSplit["a bbb  cccc aa   d"]
StringSplit["a----bbb---ccc--dddd", "--"]
StringSplit["a----bbb---ccc--dddd", "-" ..]
StringSplit["a--.-bbb- -|-ccc--dddd", ("-" | "." | "|" | " ") ..]

StringSplit[alphabet, "a" | "e" | "i" | "o" | "u"]
StringSplit[alphabet, {"a", "e", "i", "o", "u"}]
```

- 字符串模式的完整形式：

In [ ]:

```
FullForm["a" ~~ _ ~~ "b"]
FullForm["a" <> _ <> "b"]
FullForm["a" ~~ __]
FullForm["a" ~~ ___]
```

In [ ]:

```
FullForm["a" ~~ x_]
FullForm["a" ~~ x : _]
```

In [ ]:

```
FullForm["a" ..]
FullForm["a" ...]
```

In [ ]:

```
FullForm["a" | "e" | "i" | "o" | "u"]
FullForm[{"a", "e", "i", "o", "u"}]
```

- 其它字符串模式：

In [ ]:

```
StringSplit["a bbb \t cc\rcc aa \n  d", Whitespace]
```

In [ ]:

```
StringSplit["a384b6463fc216a5f8ecb6670f86456a",
 CharacterRange["0", "9"]]
StringSplit["a384b6463fc216a5f8ecb6670f86456a", DigitCharacter]
```

In [ ]:

```
StringSplit["a384b6463fc216a5f8ecb6670f86456a",
 CharacterRange["a", "z"]]
StringSplit["a384b6463fc216a5f8ecb6670f86456a", LetterCharacter]
```

In [ ]:

```
StringMatchQ["abaababb", StartOfString ~~ "a" ~~ __]
StringMatchQ["abaababb", __ ~~ "a" ~~ EndOfString]
```

In [ ]:

```
StringReplace["Deformations of semisimple bihamiltonian structures of \
hydrodynamic type", WordBoundary ~~ x_ :> ToUpperCase[x]]
```

In [ ]:

```
StringCases["aabbaccbcadddasadqweqer", Longest["a" ~~ ___ ~~ "a"]]
StringCases["aabbaccbcadddasadqweqer", Shortest["a" ~~ ___ ~~ "a"],
 Overlaps -> True]
```

*eg.*去除一段源程序中所有的空行和 C 语言风格的注释

In [ ]:

```
cSource = "#include <stdio.h>\n\n/*adasd*/ /*asdasd/*adasd*/ada*/ /*/* \
asadsad */ */ /* adasd */\n\nint main() {\n\n    printf(\"Hello, \
World!\");\n    return 0;\n\n}"
```

In [ ]:

```
FixedPoint[
 StringReplace[#, {
    "\n" .. :> "\n",
    StartOfLine ~~ " " .. ~~ EndOfLine :> "",
    "/*" ~~ x___ ~~ "*/" /; StringFreeQ[x, "/*" | "*/"] :> ""
    }] &,
 cSource
 ]
```

Mathematica还支持PCRE（Perl Compatible Regular Expressions）语法的正则表达式：

In [ ]:

```
"c" the literal character c
"." any character except newline
"\"[\!\(\*SubscriptBox[\(c\), \(1\)]\)\!\(\*SubscriptBox[\(c\), \(2\)]\)\[Ellipsis]]\"" any of the
"\"[\!\(\*SubscriptBox[\(c\), \(1\)]\)-\!\(\*SubscriptBox[\(c\), \(2\)]\)]\""   any character in t
"\"[^\!\(\*SubscriptBox[\(c\), \(1\)]\)\!\(\*SubscriptBox[\(c\), \(2\)]\)\[Ellipsis]]\""    any cha
"p*"    p repeated zero or more times
"p+"    p repeated one or more times
"p?"    zero or one occurrence of p
"p{m,n}"    p repeated between m and n times
p*?, p+?, p??   the shortest consistent strings that match
p*+,p++, p?+    possessive match
"\"(\!\(\*SubscriptBox[\(p\), \(1\)]\)\!\(\*SubscriptBox[\(p\), \(2\)]\)\[Ellipsis])\"" strings mat
"\"\!\(\*SubscriptBox[\(p\), \(1\)]\)|\!\(\*SubscriptBox[\(p\), \(2\)]\)\"" strings matching Subs
```

In [ ]:

```
"\\\\d" digit 0\[Dash]9
"\\\\D" nondigit
"\\\\s" space, newline, tab, or other whitespace character
"\\\\S" non-whitespace character
"\\\\w" word character (letter, digit, or _)
"\\\\W" nonword character
"[[:class:]]"  characters in a named class
"[^[:class:]]" characters not in a named class
```

In [ ]:

```
^   the beginning of the string (or line)
$   the end of the string (or line)
\\ A    the beginning of the string
\\ z    the end of the string
\\ Z    the end of the string (allowing for a single newline character first)
\\ b    word boundary
\\ B    anywhere except a word boundary
```

In [ ]:

```
(?i)    treat uppercase and lowercase as equivalent (ignore case)
(?m)    make ^ and $ match start and end of lines (multiline mode)
(?s)    allow . to match newline
(?x)    disregard all whitespace and treat everything between "#" and "\n" as comments
(?-\[Backslash]#c)  unset options
```

In [ ]:

```
(?=p)   the following text must match p
(?!p)   the following text cannot match p
(?<= p) the preceding text must match p
(?<!p)  the preceding text cannot match p
```

提速技巧：

- 字符串模式 vs 正则表达式

因为Mathematica处理字符串模式时都是直接将它翻译为正则表达式，所以一般来说这两者的效率没有区别。但是如果有很多字符串表达式，那么翻译的开销可能会增大，所以这时候直接写正则表达式，免去这笔开销，会提高一定的效率。

在表达能力方面，两者差不多，不过仍有一些情况不能互相翻译。例如，含有条件判断的字符串模式（即 /; 和 ?）就不能翻译到正则表达式。反之，正则表达式中的某些用法也不能直接翻译成字符串模式，需要配合Mathematica中的其它函数才能实现。

- 处理字符串表（List of strings）时一次喂

In [ ]:

```
test = Table[
   StringJoin[{"a", "c", "g", "t"}[[#]] & /@
     Table[RandomInteger[{1, 4}], {10}]], {100000}];
```

In [ ]:

```
(* 用 Select 的话，将对 test 的每个元素调用一次 StringMatchQ，造成额外的开销 *)
Select[test, StringMatchQ[#, "a" ~~ ___ ~~ "ggg" ~~ ___] &] // Timing

(* 这里利用 StringMatchQ 的 Listable 属性，只用了一次调用 *)
Pick[test, StringMatchQ[test, "a" ~~ ___ ~~ "ggg" ~~ ___]] // Timing

(* 另一种避免使用 StringMatchQ 的解决方法 *)
Flatten[StringCases[test,
   StartOfString ~~ "a" ~~ ___ ~~ "ggg" ~~ ___ ~~
    EndOfString]] // Timing
```

- 将一般的表达式搜索转换为字符串搜索

In [ ]:

```
test = Range[1000];
test[[{50, 75}]] = 5;

Position[test, 5]

MatchQ[test, {___, x_, ___, x_, ___, x_, ___}] // Timing
```

In [ ]:

```
teststr = FromCharacterCode[test];
StringPosition[teststr, FromCharacterCode[5]]

StringMatchQ[teststr,
  StringExpression[___, x_, ___, x_, ___, x_, ___]] // Timing
```