Zsh 开发指南： 数组

其实字符串在 zsh 中也可以当字符数组操作，但很少有需要把字符串当数组来处理的场景。本篇中主要讲的是字符串数组，复杂度要比单个字符串高一些。--陌辞寒

导读

了解完结构比较简单的字符串后，我们来看更复杂一些的数组。其实字符串在 zsh 中也可以当字符数组操作，但很少有需要把字符串当数组来处理的场景。本篇中主要讲的是字符串数组，复杂度要比单个字符串高一些。

在实际的脚本编写中，较少需要处理单个的字符串。往往需要处理从各个地方过来的大量文本，不可避免会用到数组。用好数组，会让文本处理工作事半功倍。

本篇只涉及数组的基础用法。

数组定义

数组可以直接赋值使用，不需要提前声明。等号和小括号之间不能有空格，小括号中的元素以空格隔开。

```
% array=(a bc ccc dddd)
# 用 $array 即可访问数组全部元素，输出时元素以空格分隔
% echo $array
a bc ccc dddd

# 使用 print -l 可以每行输出一个元素
% print -l $array
a
bc
ccc
dddd

# 输出数组中的元素个数，用法和取字符串长度一样
% echo $#array
4

# 包含带空格的字符串
% array=(a "bc ccc" dddd)
% print -l $array
a
bc ccc
dddd

# 可以换行赋值，但如果行中间有空格，依然需要加引号
% array=(
> a
> bb
> "c c c"
> dddd
> )
```

元素读写

```
% array=(a bc ccc dddd)

# 用法和取字符串的第几个字符一样，从 1 开始算
% echo $array[3]
ccc
# -1 依然是最后一个元素，-2 是倒数第二个，以此类推
% echo $array[-1]
dddd

% array[3]=CCC

# 如果赋值的内容是一个空的小括号，则删除该元素
% array[2]=()

% print -l $array
a
CCC
dddd

# 用 += 为数组添加一个新元素
% array+=eeeee
% print -l $array
a
CCC
dddd
eeeee

# 用 unset 可以删除整个数组
% unset array

# array 变量变成未定义状态
% echo $+array
0
```

数组拼接

```
% array1=(a b c d)
% array2=(1 2 3 4)

# 用 += 拼接数组
% array1+=(e f g)
% echo $array1
```

```
a b c d e f g

# 拼接另一个数组，小括号不可以省略，否则 array1 会被转成一个字符串
% array2+=($array1)
% echo $#array2
11

# 去掉小扩号后，array1 被转成了一个字符串
% array2+=$array1
% echo $#array2
12
% echo $array2[12]
a b c d e f g


# 字符串可以直接拼接数组而转化成数组
% str=abcd
% str+=(1234)

% echo $#str
2
```

数组遍历

```
% array1=(a bb ccc dddd)
% array2=(1 2 3)

# 用 for 可以直接遍历数组，小括号不可省略
% for i ($array1) {
> echo $i
> }
a
bb
ccc
dddd

# 小括号里可以放多个数组，依次遍历
% for i ($array1 $array2) {
> echo $i
> }
a
bb
```

```
ccc
dddd
1
2
3
```

数组切片

数组切片和字符串切片操作方法完全相同。

```
% array=(a bb ccc dddd)

% echo $array[2,3]
bb ccc

# 依然可以多对多地替换元素
% array[3,-1]=(1 2 3 4)
% echo $array
a bb 1 2 3 4

# 也可以使用另一种语法，不建议使用
% echo ${array:0:3}
a bb 1
```

元素查找

数组的元素查找方法，和字符串的子字符串查找语法一样。

```
% array=(a bb ccc dddd ccc)

# 用小 i 输出从左到右第一次匹配到的元素位置
% echo $array[(i)ccc]
3

# 如果找不到，返回数组大小 + 1
% echo $array[(i)xxx]
6

```

```
# 用大 I 输出从右到左第一次匹配到的元素位置
% echo $array[(I)ccc]
5

# 如果找不到，返回 0
% echo $array[(I)xxx]
0

# 可以用大 I 判断是否存在元素
% (($array[(I)dddd])) && echo good
good

% (($array[(I)xxx])) && echo good


% array=(aaa bbb aab bbc)
# n:2: 从指定的位置开始查找
% echo ${array[(in:2:)aa*]}
3
```

元素排序

```
% array=(aa CCC b DD e 000 AA 3 aa 22)

# 用小写字母 o 升序排列，从小到大
% echo ${(o)array}
000 22 3 aa aa AA b CCC DD e

# 用大写字母 O 降序排列，从大到小
% echo ${(O)array}
e DD CCC b AA aa aa 3 22 000

# 加 i 的话大小写不敏感
% echo ${(oi)array}
000 22 3 aa AA aa b CCC DD e


% array=(cc aaa b 12 115 90)
# 加 n 的话按数字大小顺序排
% echo ${(on)array}
12 90 115 aaa b cc

```

```
21. # 0a 用于反转数组元素的排列顺序
22. % echo ${(0a)array}
23. 90 115 12 b aaa cc
```

去除重复元素

```
1.  % array=(ddd a bb a ccc bb ddd)
2.
3.  % echo ${(u)array}
4.  ddd a bb ccc
```

使用连续字符或者数值构造数组

```
1.  # 大括号中的逗号分隔的字符串会被展开
2.  % array=(aa{bb,cc,11}) && echo $array
3.  aabb aacc aa11
4.
5.  # .. 会将前后的数组连续展开
6.  % array=(aa{1..3}) && echo $array
7.  aa1 aa2 aa3
8.
9.  # 第二个 .. 后的数字是展开的间隔
10. % array=(aa{15..19..2}) && echo $array
11. aa15 aa17 aa19
12.
13. # 也可以从大到小展开
14. % array=(aa{19..15..2}) && echo $array
15. aa19 aa17 aa15
16.
17. # 可以添加一个或多个前导 0
18. % array=(aa{01..03}) && echo $array
19. aa01 aa02 aa03
20.
21. # 单个字母也可以像数值那样展开，多个字母不行
22. % array=(aa{a..c}) && echo $array
23. aaa aab aac
24.
25. # 字母是按 ASCII 码的顺序展开的
26. % array=(aa{Y..c}) && echo $array
27. aaY aaZ aa[ aa\ aa] aa^ aa_ aa` aaa aab aac
28.
```

```
29.
30. # 这些用法都可以用在 for 循环里边
31. % for i (aa{a..c}) {
32. > echo $i
33. > }
34. aaa
35. aab
36. aac
```

从字符串构造数组

```
1.  % str="a bb ccc dddd"
2.
3.  # ${=str} 可以将 str 内容按空格切分成数组
4.  % array=(${=str})
5.  % print -l $array[2,3]
6.  bb
7.  ccc
8.
9.
10. % str="a:bb:ccc:dddd"
11. # 如果是其他分隔符，可以设置 IFS 环境变量指定
12. % IFS=:
13. % array=(${=str})
14. % print -l $array[2,3]
15. bb
16. ccc
17.
18.
19. % str="a\nbb\nccc\ndddd"
20. # 如果是其他分隔符，也可以用 (s:x:) 指定
21. % array=(${(s:\n:)str})
22. % print -l $array[2,3]
23. bb
24. ccc
25.
26.
27. % str="a##bb##ccc##dddd"
28. # 分隔符可以是多个字符
29. % array=(${(s:##:)str})
30. % print -l $array[2,3]
31. bb
```

```
ccc


% str="a:bb:ccc:dddd"
# 如果分隔符是 :，可以 (s.:.)
% array=(${(s.:.)str})
% print -l $array[2,3]
bb
ccc
```

从文件构造数组

`test.txt` 内容。

```
a
bb
ccc
dddd
```

每行一个元素。

```
# f 的功能是将字符串以换行符分隔成数组
# 双引号不可省略，不然会变成一个字符串，引号也可以加在 ${ } 上
% array=(${(f)"$(<test.txt)"})
% print -l $array
a
bb
ccc
dddd

# 不加引号的效果
% array=(${(f)$(<test.txt)})
% print -l $array
a bb ccc dddd


# 从文件构造数组，并将每行按分隔符 : 分隔后输出所有列
for i (${(f)"$(<test.txt)"}) {
        array=(${(s.:.)i})
        echo $array[1,-1]
```

```
20. }
```

从文件列表构造数组

```
1.  # 这里的 * 即上一篇讲的通配符，所有的用法都可以在这里使用。
2.  % array=(/usr/bin/vim*)
3.  % print -l $array
4.  /usr/bin/vim
5.  /usr/bin/vimdiff
6.  /usr/bin/vimtutor
7.
8.  # 要比 ls /usr/bin/[a-b]?? | wc -l 快很多
9.  % array=(/usr/bin/[a-b]??) && print $#array
10. 3
```

数组交集差集

```
1.  % array1=(1 2 3)
2.  % array2=(1 2 4)
3.
4.  # 两个数组的交集，只输出两个数组都有的元素
5.  % echo ${array1:*array2}
6.  1 2
7.
8.  # 两个数组的差集，只输出 array1 中有，而 array2 中没有的元素
9.  % echo ${array1:|array2}
10. 3
11.
12. # 如果有重复元素，不会去重
13. % array1=(1 1 2 3 3)
14. % array2=(4 4 1 1 2 2)
15. % echo ${array1:*array2}
16. 1 1 2
```

数组交叉合并

```
1.  % array1=(a b c d)
2.  % array2=(1 2 3)
3.
4.  # 从 array1 取一个，再从 array2 取一个，以此类推，一个数组取完了就结束
```

```
% echo ${array1:^array2}
a 1 b 2 c 3

# 如果用 :^^，只有一个数组取完了的话，继续从头取，直到第二个数组也取完了
% echo ${array1:^^array2}
a 1 b 2 c 3 d 1
```

对数组中的字符串进行统一的处理

一些处理字符串的方法（主要是各种形式的截取、替换、转换等等），也可以用在数组上，效果是对数组中所有元素统一处理。

```
% array=(/a/b.htm /a/c /a/b/c.txt)

# :t 是取字符串中的文件名，可以用在数组上，取所有元素的文件名
% print -l ${array:t}
b.htm
c
c.txt

# :e 是取扩展名，如果没有没有扩展名，结果数组中不会添加空字符串
% print -l ${array:e}
htm
txt

# 字符串替换等操作也可以对数组使用，替换所有字符串
% print -l ${array/a/j}
/j/b.txt
/j/c
/j/b/c.txt
```

`:#` 也可以在数组上用，但更实用一些。

```
% array=(aaa bbb ccc)

# :# 是排除匹配到的元素，类似 grep -v
% print ${array:#a*}
bbb ccc

# 前边加 (M)，是反转后边的效果，即只输出匹配到的元素，类似 grep
```

```
% print ${(M)array:#a*}
aaa

# 多个操作可以同时进行，(U) 是把字符串转成大写字母
% print ${(UM)array:#a*}
AAA
```

总结

本篇讲的是数组的基础用法，还有很多复杂的操作方法，以后会提到。