TCL中的符号

> 置换

1. 变量置换 $

```
set a "snow"
puts $a
```

2. 命令置换 []

```
set a [expr 3 + 4]
puts $a
```

3. 反斜杠置换 \

   换行符、空格、[、]、$等会被TCL解释器当做特殊符号处理，使用\将其转为普通字符

```
set x 1.2
set y 1.3
puts "[expr $x + $y]"
puts "\[expr $x + $y\]"
puts "\[expr \$x + \$y]"
```

> 其他符号

1. 双引号 ""

   TCL解释器对双引号中的$和[]会进行变量置换和命令置换

2. 花括号 {}

   在 {} 中所有的特殊字符都按照普通字符处理，TCL解释器不会对其进行特殊处理

3. # 注释

TCL数据结构

> ***变量，数组，列表***

1. 变量 **某个容器的名称**

```
set <变量名> <变量值>
$<变量名> 取变量值
set a 2
puts ${a}_1     # 使用 ${a}_1 实现在变量内容后追加后缀
```

2. 数组 **存储一系列元素值，每一个元素通过数组名与元素名进行检索，类似于键值对**

```
set <数组名>(<元素名>) <元素值>
$<数组名>(<元素名>) 取数组中某元素值
set cell_1(pins) "A B C"
puts $cell_1(pins)
```

使用 *array* 指令获取数组信息

```
array size <数组名>     # 查看数组中元素个数
array name <数组名>     # 查看数组中各个元素名
```

3. 列表 **标量的有序集合**

```
set <列表名> {<元素1> <元素2> <元素3>}
$<列表名> 取列表值
```

TCL中有一系列关于列表的操作命令

```
命令                功能
concat          合并两个列表                concat $<list_name> $<list_name>
lindex          选取列表中的某一个元素       lindex $<list_name> <num>
llength         列表长度                    llength $<list_name>

lappend         在列表末端追加元素           lappend <list_name>
<num>/$<list_name>

lsort           列表排序                    lsort <开关> $<list_name>
            开关：缺省时按照ASCII码进行排序
             -real 按照浮点数值大小排序
             -unique 唯一化排序，删除重复元素
```

```
set list1 {buf1 buf2 bug3}
set list2 {bif1 bif2 bif3}
concat $list1 $list2
llength $list1
llength [concat $list1 $list2]
lindex $list1 1     # 此时输出为 buf2
lindex $list1 [expr [llength $list1] -1]    # 得到列表中最后一个元素的值

lsort
```

lappend <list_name> $<list_name> 这个语句表明TCL列表中可以包含列表

TCL支持的运算

1. 数学运算

+ - * /

> 数学运算指令 ***expr*** 将运算表达式求值
>
> 进行浮点数运算时 或结果应该为小数时，将运算表达式中任意一个数写成浮点形式即可

2. 逻辑运算

<= >= == !=

TCL控制流

if控制流

```
# 脚本语句的 { 一定要写在上一行，否则Tcl脚本解释器会认为当前行脚本已经执行结束，出现错误
if {判断条件} {
脚本语句
} elseif {判断条件} {
脚本语句
} else {
脚本语句
}
```

foreach循环指令

> 语法格式
>
> foreach 变量 列表 {循环主体}
>
> 功能：从第0个元素开始，依次取列表的元素，将其赋值给变量，然后执行循环主体一次，直到列表最后一个元素

```
set list1 {1 2 3}
foreach i $list1 {
# 循环主体
puts $i
}
```

循环控制指令 break & continue

```
foreach i $list {
if {} {
    continue
} elseif {} {
    break
}
}
```

循环控制指令 while

> 语法格式
>
> while {判断语句} {循环主体}
>
> 当while中的判断语句不满足时停止循环，此时while命令中断并返回一个空字符串

```
set i 3
while {$i > 0} {
    puts $i
    incr i -1; # set i [expr $i - 1]
}
```

循环控制指令 for

> 语法格式
>
> for {参数初始化} {判断语句} {重新初始化参数} {
>
> 循环主体
>
> }

```
for {set i 3} {$i > 0} {incr i -1} {
puts $i
}
```

过程函数

proc

> proc 函数名 参数列表 函数主题

```
proc add {a b} {
set sum [expr $a + $b]
return sum
}
add 3 4;

###################################
全局变量：  在所有过程之外定义的变量
局部变量：  在过程中定义的变量，只能在过程中被访问，过程退出后会被删除 (生存周期)
指令 global :可以在过程内部引用全部变量
# 在过程中引用全局变量需要使用 global 关键字
```

正则

\w 匹配任意字符（字母、数字、下划线）

\d 匹配一个数字

***** 零次或多次匹配

+ 一次或多次匹配

? 零次或一次匹配

\s 空位

\S 非空位置

. 任意一个字符

符号	功能
*	零次或多次匹配
+	一次或者多次匹配
?	零次或者一次匹配

正则匹配指令 - regexp

> 语法格式
>
> ```
> % regexp {<正则>} "<正文字符串>" <变量0> <变量1> ...
> # 默认使用变量0 存放正则表达式匹配到的子串
> # 若在正则表达式中使用了 () 进行子串捕获，则捕获到的子串存放于 变量1 变量2... 中
> ```
>
> 功能： 在字符串中使用正则表达式匹配

```
################################################
switches : -nocase 将字符串中的大写都当作小写处理
exp 正则表达式
string 用来进行匹配的字符串
matchstring 用正则表达式匹配的所有字符串
sub 正则表达式中的子表达式匹配的字符串
################################################
% regexp {\w+\d+} "abc456"  # 至少一个字母与至少一个数字
1
% regexp {^\d.*\d$} "1 dfsal 1" # 以数字开头 以数字结尾
1
```

捕获变量

- 使用()捕获变量

  ```
  # 将字符串"Snow is 30 years old"中的30捕获出来
  % regexp {\s(\d+).*} "Snow is 30 years old" total age
  1   # 使用括号 ()捕获到的子串存放到了 age变量中，而 {}中正则表达式匹配到的子串存放到了total变量中
  % puts $total
   30 years old
  % puts $age
  30
  ```

- 一次性捕获多个字符串 ***在正则表达式中使用多个括号()***

文本处理

- open - 打开文件

  ```
  open <file_name> <parameter>
  # parameter 打开方式 r / w
  # 指令存在返回值 返回值为文件ID 一般记为 fileId，通过set命令存放于其他变量中
  ```

- gets - 读fileId标识文件的下一行，并把该行赋值给变量，并返回该行的字符数(文件尾部返回-1)

```
gets fileId <Var>
# 使用 Var存放读到的字符串
```

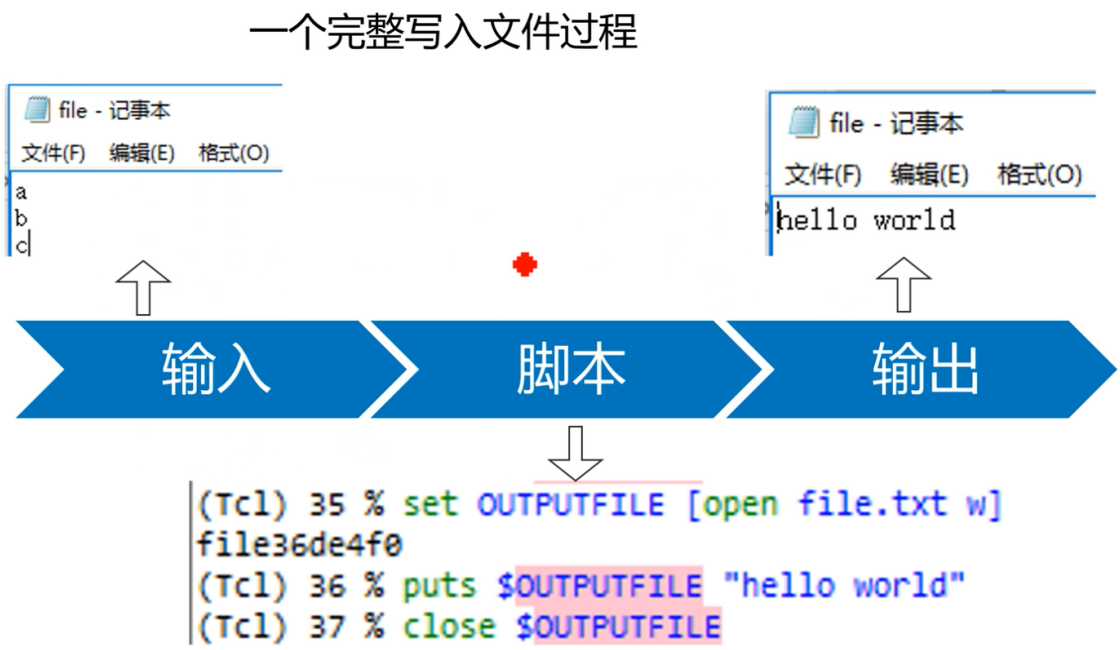
- close - 关闭文件

```
close fileId
```

读入整个文件的过程

```
% set INPUTFILE [open file.txt r] # 使用INPUTFILE变量存放文件ID
% while {[gets $INPUTFILE line] >= 0} { # 读到文件尾部返回值为-1 故使用 >=0判断是否读完
puts $line
}
% close $INPUTFILE  # 关闭文件
```

写入文件的过程




```
puts <fileId> <字符串> # 根据文件句柄指向 在文件末尾追加字符串
```

文件操作结束后一定要使用*close*命令关闭文件

现有文本file.txt其内容如下。请写一TCL脚本求出所有Slack值之和。

```
Slack = -0.051
Slack = -0.234
Slack = -0.311
Slack = -0.056
Slack = -0.434
Slack = -0.316
Slack = -0.151
Slack = -0.524
```

```
set sum 0
set INFILE [open file.txt r]
while {[gets $INFILE line] >= 0} {
if {[regexp {^Slack\s+=\s+(-?\d+\.?\d+)}
$line total slack]} {
set sum [expr $sum +$slack]
}
}
close $INFILE
puts $sum
```