tcl语法速查

tcl_synopsys

tcl_tessent

参考：

- tcl86_doc
- 正则表达式

tcl-tk入门经典_第二版_.pdf

tcl_and_the_tk_toolkit_john_k._ousterhout_.pdf

1. 查看tcl版本

进入tcl命令tclsh后，输入以下内容：

```
info patchlevel
```

安装tcl:

```
yum install -y tcl-devel.x86_64 tcl.x86_64
```

2. regexp正则表达式

语法：

```
regexp optionalSwitches patterns searchString fullMatch subMatch1 ...
subMatchn
```

-nocase：用于忽略大小写。

2.1 正则表达式规则

更详细的tcl正则表达式规则请查看 http://vmcc.vicp.net:9090/tcl86_doc/re_syntax.htm

tcl支持三种RE规则（BRE,ERE,ARE）默认是使用的ARE）

- BRE是basic REs）就是vim和grep使用正则表达式规则。
- ERE是extended REs） 是egrep使用的规则。
- ARE是在ERE的基础上再扩展，跟perl的有点相似，但细节有些区别。

```
# [[:<:]]和[[:>:]]与\<和\>和功能一样,用于匹配word边界
# 三种规则模式都支持[[:<:]] [[:>:]], 但仅BRE支持 \< \>

set a 123

## 使用ARE规则: --默认规则, 不用加任何修饰
if { [regexp {[[:<:]]123[[:>:]]} $a] } {
  puts "matched"
}

## 使用BRE规则： --加个(?b)修饰符就表示使用BRE规则
if { [regexp {(?b)23\>} $a] } {
  puts "matched"
}

## 使用ERE规则： --加个(?e)修饰符就表示使用ERE规则
if { [regexp {(?e)23[[:>:]]} $a] } {
  puts "matched"
}
```

2.2 基本用法

```
set name "day day up"

if [regexp -nocase {day} $name] {
    puts "matched"
}

if [regexp -nocase "day" $name] {
    puts "matched2"
}

if [regexp -nocase "cat" $name] {
    puts "matched3"
} else {
    puts "not matched3"
}
```

运行结果：

```
matched
matched2
not matched3
```

```
% set a "cdb"
cdb
% if {![regexp {c} $a]} { puts "haha" }
% set a "adb"
adb
```

```
% if {![regexp {c} $a]} { puts "haha" }
haha
%
```

2.3 {}与""的区别

- {}里面不能做变量替换，类似于像\s, [,]这种功能符不需要在前面加转义
- “ “ 里面支持变量替换，如果要输入功能符需要在前面加转义，即 \[表示为[, \]表示为]， 简单理解该表达式打印出来的值就是我们预期的值，所以在打印的时候要考虑转义。

```
% set a {\[}
\[
%
% set a "\\\["
\[
```

```
%
% set a "af"
af
% set b "af <= 3"
af <= 3
% regexp "af " $b
1
% regexp "$a " $b
1
% regexp "af\s" $b
0
% regexp "$a\s" $b
0
% regexp "af\\s" $b
1
% regexp {$a\s} $b
0
% regexp {\s} $b
1
% regexp "$a\\s<=" $b
1
% regexp "$a\\s*<=" $b
1
%
```

2.4 匹配多个条件

匹配多个条件时，直接使用 | 符号

```
% set a "hello you"
hello you
% if {[regexp {hello|you} $a]} {puts "hit"}
hit
```

```
% if {[regexp {hello|xou} $a]} {puts "hit"}
hit
% if {[regexp {hellof|xou} $a]} {puts "hit"}
% if {[regexp {hellof|you} $a]} {puts "hit"}
hit
%
```

2.5 submatch 子匹配

```
set name "day day up"

if [regexp -nocase {(day) (day) (up)} $name m1 m2 m3 m4] {
    puts "m1 = $m1, m2 = $m2, m3 = $m3, m4 = $m4"
}


if [regexp -nocase {(\w+) (\w+) (\w+)} $name m1 m2 m3 m4] {
    puts "m1 = $m1, m2 = $m2, m3 = $m3, m4 = $m4"
}
```

运行结果：

```
m1 = day day up, m2 = day, m3 = day, m4 = up
m1 = day day up, m2 = day, m3 = day, m4 = up
```

2.6 非贪婪匹配模式

非贪婪模式，就是在匹配满足的情况下，尽量少的匹配字符。

贪婪模式，就是尽量多在匹配字符，默认为贪婪模式。

贪婪匹配	非贪婪匹配	说明
*	*?	非贪婪模式就是在正常匹配模式后面加一个?号
+	+?	
?	??	
{m}	{m}?	
{m,}	{m,}?	
{m,n}	{m,n}?	

2.7 regsub正则表达式替换

语法：`regsub ?switches? exp string subSpec ?varName?`

当switches为
-all表示执行多次替换，类似于vim替换的g操作。
-nocase表示匹配时忽略大小写。

```
#  将abc替换为hello, 并且将替换后的内容保存到变量c
```

```
% regsub  {\w+} "abc def" "hello" c
1
% echo $c
"hello def"

#替换abc成hello
% regsub  {\w+} "abc def" "hello"
hello def

% set d [regsub  {\w+} "abc def" "hello" ]
hello def
% echo $d
"hello def"
%
```

2.8 使用变量作为正则表达式

```
%
%
% set c "da\[0\]\[2\]"
da[0][2]
% set c [regsub -all {\]} $c {\]}]
da[0\][2\]
% set c [regsub -all {\[} $c {\\[}]
da\[0\]\[2\]
%
% set b "da\[0\]\[2\] = 38"
da[0][2] = 38
% regexp $c $b  ; ##这里就是用$c里面的值来匹配$b里面的内容□ $c是一个变量，不是固定的string
1
%
```

2.9 使用\逃逸

```
%
% set c "da\[0\]\[2\]"
da[0][2]
% regsub -all {\]} $c {\\}
da[0\[2\
% regsub -all {\]} $c {\]}
da[0\][2\]
% regsub -all {\]} $c {\a}
da[0\a[2\a
%
% regsub -all {\[} $c {\\}
da\0]\2]
% regsub -all {\[} $c {\[}
da\[0]\[2]
```

```
% regsub -all {\[} $c {\a}
da\a0]\a2]
%
```

2.10 使用\1 \2

```
% set a "abc.ef"
abc.ef
% regsub {(.*c)} $a {\1}
abc.ef
% set ret [regsub {(.*c).*} $a {\1}]
abc
%
```

3. number

integer value: 335 (decimal), 0o517 (octal), 0x14f (hexadecimal), 0b101001111 (binary).

4. string

4.1 string length长度

```
% set len [string length "abc"]
3
% echo $len
```

4.2 string compare

string compare ?-nocase? ?-length length? string1 string2

返回结果为0表示相等， 返回-1, 表示string1 < string2， 返回1表示string1 > string2。

```
% string compare "abc" "abc"
0
% string compare "abc" "abc0"
-1
% string compare -nocase "abc" "ABc"
0
```

4.3 string cat

Concatenate the given strings just like placing them directly next to each other and return the resulting compound string. If no strings are present, the result is an empty string.

```
string cat ?string1? ?string2...?

set new_str [string cat "hello " "world"]
puts $new_str
#打印hello world
```

4.4 string replace

```
string replace string first last ?newstring?


string replace "0x55,hello" 0 3
#返回,hello


string replace "0x55,hello" 0 3 "0x33"
#将0x55替换为0x33
```

Removes a range of consecutive characters from string, If newstring is specified, then it is placed in the removed character range.

4.5 string range

```
string range string first last

string range "hello, world"  0 3

#返回hell
```

Returns a range of consecutive characters from string

4.6 string repeat

```
string repeat string count

string repeat "0" 4
```

Returns string repeated count number of times.

4.7 string 大小写转换

转大写

```
string toupper string ?first? ?last?
```

转小写

```
string tolower string ?first? ?last?
```

5. 布尔值

https://www.bilibili.com/read/cv16148884/

```
布尔值是编程语言中，普遍存在的一种数据类型。即便没有这种数据类型，也会有这个概念，比
如Tcl/Tk

布尔值表示"是、非"或者"真、假"这么一个成对的概念Tcl语言里都是字符串,
它通过一些特定字符表示布尔值。首先，很好理解的表示布尔值的单词（大小写均可）：

yes, no
true, false
on, off
数值1,0
```

```
% set a "true"
true
%  if {$a} {puts haha}
haha
% string is boolean "true"
1
#变量本身也是用string存储的，也可以直接和字符串比较
%  if {$a=="true"} {puts haha}
haha
```

```
## 布尔值单词的一部分，也被认为是布尔值
#以下都代表"真"
string is boolean "t"
string is boolean "tr"
string is boolean "tru"
string is boolean "ye"
#对应的f, fa, fal, fals, n 都代表"假"

#注意：字母 o 不会被认为是布尔值，因为on和off开头都是o
string is boolean o
>> 0
#但是of可以用来表示"假"，因为它是off的一部分
```

6. 进制转换

6.1 转为二进制数

```
% format "%b" 3
11
```

```
% format "%b" 0x3
11
% format "%b" 0x30
110000
```

6.2 转为16进制数

```
% format "%x" 0x30
30
% format "%x" 30
1e
% format "0x%x" 30
0x1e
%
```

6.3 转为10进制数

```
% format "%d" 0x30
48
% format "%d" 30
30
%
```

6.4 产生递进的二进制数

t3.tcl

```
for {set i 0} {$i <16} {incr i} {
    set hex_str [format "%04b" $i]
    puts -nonewline "hex_str = $hex_str"

    puts -nonewline "  bit\[3:0\] ="
    for {set j 3} {$j >= 0} {incr j -1} {
        set idx [expr 3-$j]
        set bit_str [string range $hex_str $idx $idx]
        puts -nonewline "  $bit_str"
    }
    puts ""
}
```

运行结果：

```
hex_str = 0000  bit[3:0] =  0  0  0  0
hex_str = 0001  bit[3:0] =  0  0  0  1
hex_str = 0010  bit[3:0] =  0  0  1  0
hex_str = 0011  bit[3:0] =  0  0  1  1
```

```
hex_str = 0100  bit[3:0] =  0  1  0  0
hex_str = 0101  bit[3:0] =  0  1  0  1
hex_str = 0110  bit[3:0] =  0  1  1  0
hex_str = 0111  bit[3:0] =  0  1  1  1
hex_str = 1000  bit[3:0] =  1  0  0  0
hex_str = 1001  bit[3:0] =  1  0  0  1
hex_str = 1010  bit[3:0] =  1  0  1  0
hex_str = 1011  bit[3:0] =  1  0  1  1
hex_str = 1100  bit[3:0] =  1  1  0  0
hex_str = 1101  bit[3:0] =  1  1  0  1
hex_str = 1110  bit[3:0] =  1  1  1  0
hex_str = 1111  bit[3:0] =  1  1  1  1
```

6.5 set_bits get_bits

```
proc h2b {h} {
    switch $h {
        0       { set ret 0000 }
        1       { set ret 0001 }
        2       { set ret 0010 }
        3       { set ret 0011 }
        4       { set ret 0100 }
        5       { set ret 0101 }
        6       { set ret 0110 }
        7       { set ret 0111 }
        8       { set ret 1000 }
        9       { set ret 1001 }
        a       { set ret 1010 }
        b       { set ret 1011 }
        c       { set ret 1100 }
        d       { set ret 1101 }
        e       { set ret 1110 }
        f       { set ret 1111 }
        default { set ret 0000 }
    }
    return $ret
}
proc hex2bin {hex_value} {
    set hex_value [regsub -nocase {0x} $hex_value {}]
    set len [string length $hex_value]
    set bins "0b"
    for {set i 0} {$i<$len} {incr i} {
        set tmp [string range $hex_value $i $i]
        set tmp [h2b $tmp]
        set bins "$bins$tmp"
    }
    return $bins
}
proc bin2hex {bins} {
    set bins [regsub -nocase {0b} $bins {}]
```

```
    set len [string length $bins]
    set padding ""
    if { [expr $len%4] != 0} {
        set padding [string repeat "0" [expr (4-$len%4)]]
    }

    set bins "$padding$bins"
    set len [string length $bins]
    set hexs "0x"
    for {set i 0} {$i<[expr $len/4]} {incr i} {
        set tmp [string range $bins [expr 0+$i*4] [expr 3+$i*4]]
        set tmp [format "%0x" "0b$tmp"]
        set hexs "$hexs$tmp"
    }
    return $hexs
}
proc expandbins {width bins} {
    set bins [regsub -nocase {0b} $bins {}]
    set len [string length $bins]
    if { $len > $width} {
        return [string range $bins [expr $len-$width] [expr $len-1]]
    } else {
        set padding [string repeat "0" [expr $width-$len]]
        return "$padding$bins"
    }
}

proc get_bins {width hex_value msb lsb} {
    set bins [hex2bin $hex_value]
    set bins [expandbins $width $bins]
    set bins [string range $bins [expr $width-$msb-1] [expr $width-$lsb-1]]

    return $bins
}
proc get_bits {width hex_value msb lsb} {
    set bins [get_bins $width $hex_value $msb $lsb]
    return [bin2hex $bins]
}

proc set_bits {width hex_value msb lsb new} {
    set bins [hex2bin $hex_value]
    set bins [expandbins $width $bins]

    set new_bins [get_bins [expr $msb-$lsb+1] $new [expr $msb-$lsb] 0]

    set replace_bins [string replace $bins [expr $width-$msb-1] [expr
$width-$lsb-1] $new_bins]
    return [bin2hex $replace_bins]
}
```

```
%
```

```
% get_bits 32 "0x33" 3 0
0x3
% get_bits 32 "0x33" 1 0
0x3
% get_bits 32 "0x33" 7 0
0x33
%
```

7. list

7.1 创建list

```
set list_a {1 2 3}
#or
set list_b [list 4 5 6]
```

7.2 list 长度

```
set len [llength $list_a]
```

7.3 list index索引

```
set item0 [lindex $list_a 0]

% set a {1 2 3 4 5}
1 2 3 4 5
% lindex $a 0
1
% lindex $a 1
2
% lindex $a end
5
% lindex $a end-1
4
% lindex $a end-2
3
```

7.4 list sort排序

```
set list_a_sorted [lsort $list_a]
```

```
% set a {a0 b0 b2 b3 b10 b20 b22 b30}
a0 b0 b2 b3 b10 b20 b22 b30
% lsort $a
```

```
a0 b0 b10 b2 b20 b22 b3 b30
% lsort -dictionary $a
a0 b0 b2 b3 b10 b20 b22 b30
%
```

7.5 list lappend追加

可以用作push功能使用

```
lappend listName value
```

7.6 lassign

```
% split {a:2} {:}
a 2
% lassign  [split {a:2} {:}]  m1 m2
% puts $m1
a
% puts $m2
2
```

7.7 list 嵌套

```
% set a {1 2}
1 2
% set b {3 4}
3 4
% set c {}
% lappend c $a
{1 2}
% lappend c $b
{1 2} {3 4}
% echo $c
"{1 2} {3 4}"
% foreach item $c { puts "-----"; foreach i $item {puts $i}}
-----
1
2
-----
3
4
% concat $a $b $c
1 2 3 4 {1 2} {3 4}
% llength [concat $a $b $c]
6
```

```
#注意lrange返回的是由first 到last元素组成的数据，如果原来的元素是数组，会被转换成字符串。
% foreach i [lrange $c 0 0] { puts $i}
1 2
% set a [lrange $c 0 0]
{1 2}
% lindex $a 0; # a只有一个元素，那就是“1 2”，a已经不是list了。
1 2
```

7.8 list concat

```
% concat a b {c d e}
a b c d e

% set a {1 2 3}
1 2 3
% set b {u v w}
u v w
% concat $a $b
1 2 3 u v w
% concat a b $b
a b u v w
```

7.9 list 删除第一个元素

可以用作pop功能使用

```
set listName [lrange $listName 1 end]
```

7.10 linsert 插入

语法：

```
linsert list index ?element element ...?
```

This command produces a new list from list by inserting all of the element arguments just before the index'th element of list.

举例：注意，linsert后面需要使用$带变量名。

```
set oldList {the fox jumps over the dog}
set midList [linsert $oldList 1 quick]
set newList [linsert $midList end-1 lazy]
# The old lists still exist though...
set newerList [linsert [linsert $oldList end-1 quick] 1 lazy]
```

7.11 lreplace

lreplace — Replace elements in a list with new elements

lreplace list first last ?element element ...?

Replacing an element of a list with another:

```
% lreplace {a b c d e} 1 1 foo
a foo c d e
```

Replacing two elements of a list with three:

```
% lreplace {a b c d e} 1 2 three more elements
a three more elements d e
```

Deleting the last element from a list in a variable:

```
% set var {a b c d e}
a b c d e
% set var [lreplace $var end end]
a b c d
```

7.12 lrange

lrange — Return one or more adjacent elements from a list

lrange list first last

EXAMPLES

Selecting the first two elements:

```
% lrange {a b c d e} 0 1
a b
```

Selecting the last three elements:

```
% lrange {a b c d e} end-2 end
c d e
```

Selecting everything except the first and last element:

```
% lrange {a b c d e} 1 end-1
b c d
```

7.13 split string to list

split — Split a string into a proper Tcl list

```
% set a "hello world"
hello world
% split $a { }
hello world
% llength [split $a { }]
2
% foreach item [split $a { }] {
puts $item
}
hello
world
```

7.14 技巧

目前还没有找到在递归调用中返回return list的方法， 目前自己使用的是替代方案，采用一个global的list，然后在递归函数里直接修改该list即可。

8. array数组

8.1 创建数组

语法：`set ArrayName(Index) value`

```
set ar(0) 1
set ar(1) 2
set ar(2) 3

puts $ar(0)
puts $ar(1)
puts $ar(2)
```

8.2 array set

初始化设置array, 参数必须是要偶数个，name-value是成对出现的

```
% array set c {a 1 b 2 c 3 d 4}
% echo $c(a)
1
% echo $c(b)
2
% echo $c(c)
```

```
3
% echo $c(d)
4
```

8.3 array size

```
% array size c
6
```

8.4 array name

用于搜索array里面是否出现xx name关键字

```
array names arrayName ?mode? ?pattern?
Returns a list containing the names of all of the elements in the array that
match pattern.
Mode may be one of -exact, -glob, or -regexp.   -glob is default
```

```
% array set c {a 1 b 2 c 3 d 4}
% echo $c(a)
1
% echo $c(b)
2
% echo $c(c)
3
% echo $c(d)
4
% set c(a1) 1.1
1.1
% set c(b1) 2.1
2.1
% array name c -regexp a
a a1
% array name c -regexp b
b b1
% array name c -exact a
a
% array name c -exact a1
a1
% array name c -exact b
b
% array name c -exact b1
b1
% array name c -exact a2   #  a2就不存在
% if {[array name c -exact a2] == ""} {  echo "not exist" }
"not exist"
%
% if {[array name c -exact a1] == "a1"} {  echo "exist" }
```

```
exist
%



% array name c a
a
% array name c a1
a1
% array name c a*
a a1
%
```

8.5 array迭代

```
for {set i 0} {$i < [array size ar]} {incr i} {
    puts "ar $i : $ar($i)"
}
```

8.6 array 非数字index

```
% set TD(a)   a
a
% set TD(a,b) a,b
a,b
% array size TD
2
% set id [array startsearch TD]
s-1-TD
% array nextelement TD $id
a
% array nextelement TD $id
a,b
% array nextelement TD $id
% array donesearch TD $id
%
```

8.7 生成连续数字list

tcl语言并没有原生类似perl 1..10产生10个连续数字的语法，可以使用类似以下proc的方式产生。

```
proc range {start end} {
    set list {}
    for {set i $start} {$i <= $end} {incr i} {
        lappend list $i
    }
    return $list
```

```
}
set numbers [range 1 10]  ;#生成1到10的列表
```

9. dict字典

语法

```
dict set dictname key value
# or
dict create key1 value1 key2 value2 .. keyn valuen
```

9.1 创建dict

```
#创建一个空DICT

%set mdict [dict create]
%dict keys $mdict


#创建dict,并赋值
dict set ages zhangsan 30; 往ages添加zhangsan
dict set ages lisi 40; 往ages添加lisi

#or

set weigths [dict create "a" 0.2 "b" 0.3 "c" 0.4 "d" 0.66]
```

字符串方式直接创建dict

```
set mydict {
    a {
        num1 {1}
        num2 {2}
    }
    b {
        num3 {3}
        num4 {4}
    }
}
set a    [dict get $mydict a]
set num1 [dict get $a num1]
set num2 [dict get $a num2]
set b    [dict get $mydict b]
set num3 [dict get $b num3]
set num4 [dict get $b num4]
```

9.2 dict unset 删除某个key

```
dict unset dictionaryVariable key ?key ...?

dict unset ages lisi; #删除lisi
```

This operation (the companion to dict set) takes the name of a variable containing a dictionary value and places an updated dictionary value in that variable that does not contain a mapping for the given key. Where multiple keys are present, this describes a path through nested dictionaries to the mapping to remove. At least one key must be specified, but the last key on the key-path need not exist. All other components on the path must exist. The updated dictionary value is returned.

9.3 dict size大小

```
dict size $weigths
```

9.4 dict for

```
dict for {key value} $dic {
    puts "$key--$value"
}
```

9.5 dict get value

```
dict get $weigths a


foreach {key value} [dict get $weigths] {
    puts "$key--$value"
}

set value [dict get $weigths "aa"]
```

9.6 dict keys

```
dict keys $weigths
```

9.7 dict迭代

```
foreach item [dict keys $weigths] {
    set value [dict get $weigths $item]
    puts $value
```

```
}
```

9.8 dict exists

```
dict exists $weigths a
```

9.9 dict keys sort排序

```
lsort [dict keys $weigths]
```

9.10 dict 按值排序

```
set d {a 10 b 3 c 25 d 1}

# 1. 生成嵌套列表 {{a 10} {b 3} {c 25} {d 1}}
set pairs {}
dict for {k v} $d {
    lappend pairs [list $k $v]
}

# 2. 按第2列（值）数字排序
set sortedPairs [lsort -integer -index 1 $pairs]

# 3. 展开回扁平字典列表
set flat [concat {*}$sortedPairs]
set newDict [dict create {*}$flat]
puts $newDict
```

封装成proc

```
proc sortDictByVal {dict args} {
    set pairs {}
    dict for {k v} $dict {lappend pairs [list $k $v]}
    return [concat {*}[lsort {*}$args -index 1 $pairs]]
}
# 调用示例
set sortedDict [sortDictByVal $d -integer -decreasing]
```

9.11 dict merge 相当于copy

```
dict merge ?dictionaryValue ...?

set dict_a [dict merge $dict_b]
set dict_a [dict merge $dict_b $dict_c]
```

Return a dictionary that contains the contents of each of the dictionaryValue arguments. Where two

(or more) dictionaries contain a mapping for the same key, the resulting dictionary maps that key to the value according to the last dictionary on the command line containing a mapping for that key.

9.12 dict 包 list

```
% set a {1 2}
1 2
% set b {3 4}
3 4
% set mdict [dict create]
% dict set mdict key_0 $a
key_0 {1 2}
% dict set mdict key_1 $b
key_0 {1 2} key_1 {3 4}
% foreach key [dict keys $mdict] { foreach i [dict get $mdict $key] { puts
"$key . $i"} }
key_0 . 1
key_0 . 2
key_1 . 3
key_1 . 4
```

10. 条件判断 & loop循环控制

10.1 if elseif else

```
if { $x>0 } {
    puts "x>0"
} elseif { $x□□1 } {
    puts "x==0"
} else {
    puts "other"
}
```

```
set isTrue [expr {
    [info exists ::env(SOME_ENV_VAR)] &&
    [string is true -strict $::env(SOME_ENV_VAR)]
}]


set a "word"
set b "base"
if { [expr [regexp {wo} $a] && [regexp {ba} $b] ] } {
    puts "haha"
}
```

10.2 while

```
set i 8
while { $i>=0 } {
    puts "$i"
    incr i -1
}

set i 8
while { $i>=0 } {
    puts "$i"
    incr i -1
    if { $i == 2 } {
        break
        #continue
    }
}
```

10.3 for

```
set b " "
set loop_times 8
for {set i $loop_times} {$i>=0} {incr i -1} {
      lappend b $i
}
```

10.4 foreach

```
set list_b {1 2}
foreach i $list_b {
    puts "$i"
}
```

10.5 switch

```
switch $x {
    a -
    b {incr t1}
    c {incr t2}
    d { [expr 1*2] }
    default {incr t3}
}
```

其中 a 的后面跟一个'一'表示使用和下一个模式相同的脚本。

switch看起来还有行数限制，如果switch结构行数过多，可能导致语法错误，此时建议修改下写法，比如

在switch结构里使用proc调用，这样代码显得更紧凑。

11. proc 过程函数控制

11.1 无参数

```
set xxx_var 0x3

proc helloWorld {} {
    global xxx_var
    puts "Hello, World!"
    puts "xxx_var = $xxx_var"
}
helloWorld

#  proc中，可以使用global xxx_var来使用proc之外的变量$xxx_var
```

11.2 带参数

```
proc add {a b} {
   return [expr $a+$b]
}
puts [add 10 30]
```

11.3 参数带默认值

```
proc add {a {b 100} } {
   return [expr $a+$b]
}
puts [add 10 30]
puts [add 10]
```

11.4 可变个数参数

```
##########用法1,传入proc的是一个list变量
% proc calc_sum {numbers} {
    set sum 0
    foreach number $numbers {
      set sum  [expr $sum + $number]
        }
    return $sum
}
% puts [calc_sum {70 80 50 60}]
260
% calc_sum 70 80
```

```
wrong # args: should be "calc_sum numbers"
% calc_sum 70
70

##########用法2, 参数以args结尾, args就代表不确定数量的多个参数

% proc calc_sum2 {args} {
    set sum 0
    foreach number $args {
      set sum  [expr $sum + $number]
        }
    return $sum
}
%
% calc_sum2 2
2
% calc_sum2 2 3
5
% calc_sum2 2 3 6
11
% calc_sum2 {70 80}
missing operator at _@_
in expression "0 + 70 _@_80"
```

12. 文件

12.1 文件操作模式

模式	说明
r	只读
w	只写
a	追加写
r+	读写，文件必须存在
w+	读写，如果文件存在，就变成0文件。如果文件不存在就创建文件
a+	读写，如果文件存在，只能追加写。如果文件不存在就创建文件

12.2 写文件

```
set fp [open "input.txt" w]
puts $fp "test0"
puts $fp "test1"
close $fp
```

12.3 读文件

12.3.1 采用read方法

read — Read from a channel 一下把内容全部读出来

语法：`read channelId numChars`

```
set fp [open "input.txt" r]
set file_data [read $fp]
puts "readout: $file_data"
close $fp
```

12.3.2 采用get一行方法

gets — Read a line from a channel 一行一行地读

语法：`gets channelId ?varName?`

```
set fp [open "input.txt" r]
while { [gets $fp data] >= 0 } {
   puts "readout2: $data"
}
close $fp
```

12.4 判断文件

12.4.1 是否可执行

```
file executable name
```

Returns 1 if file name is executable by the current user, 0 otherwise. On Windows, which does not have an executable attribute, the command treats all directories and any files with extensions exe, com, cmd or bat as executable.

12.4.2 是否存在

```
file exists name
```

Returns 1 if file name exists and the current user has search privileges for the directories leading to it, 0 otherwise.

12.4.3 是否是目录

```
file isdirectory name
```

Returns 1 if file name is a directory, 0 otherwise.

12.4.4 是否是普通文件

```
file isfile name
```

Returns 1 if file name is a regular file, 0 otherwise.

12.5 文件路径展开

12.5.1 带*号匹配展开为绝对路径

```
glob – Return names of files that match patterns

set abs_path [glob ./out/linux_*/*/]
```

12.5.2 相对路径替换为绝对路径

file normalize name

Returns a unique normalized path representation for the file-system object (file, directory, link, etc), whose string value can be used as a unique identifier for it. A normalized path is an absolute path which has all “../” and “./” removed. Also it is one which is in the “standard” format for the native platform. On Unix, this means the segments leading up to the path must be free of symbolic links/aliases (but the very last path component may be a symbolic link), and on Windows it also means we want the long form with that form's case-dependence (which gives us a unique, case-dependent path). The one exception concerning the last link in the path is necessary, because Tcl or the user may wish to operate on the actual symbolic link itself (for example file delete, file rename, file copy are defined to operate on symbolic links, not on the things that they point to).

```
set abs_path [file normalize ../../]
```

13. 随机数

```
    set rand_data0 [expr int([expr rand()*10000])]
    set rand_data1 [expr int([expr rand()*10000])]
    set rand_data2 [expr int([expr rand()*10000])]
    set rand_data3 [expr int([expr rand()*10000])]
    set rand_data4 [expr int([expr rand()*10000])]
```

14. 调用系统shell命令

```
exec uname -a
exec mkdir dir_a/subdir_b -p
```

15. info命令

```
[info script]  # 用于显示当前脚本文件路径。

[info procs ?pattern?]
# If pattern is not specified, returns a list of all the names of Tcl
command procedures in the current namespace.
# If pattern is specified, only those procedure names in the current
namespace matching pattern are returned.
#包括用户自己写的proc函数
```

15.1 判断变量是否存在

```
% unset a
% if {[info exists a] && $a == 1} { echo "haha" }
% set a 1
1
% if {[info exists a] && $a == 1} { echo "haha" }
haha
% set a 0
0
% if {[info exists a] && $a == 1} { echo "haha" }
%
```

16. 时间

```
clock microseconds      返回毫秒计时
clock seconds          返回秒计时
```

```
set time1 [clock microseconds]
puts "time1 = $time1"

set time2 [clock microseconds]
puts "time1 = $time2"

set delta_time [expr $time2 - $time1]
puts "delta_time = $delta_time"


#获取系统时间
% clock format [clock seconds] -format {%Y%m%d-%H:%M}
20221122-10:28
%
```

17. tcl command alias别名

tcl支持一个命令有多个alias别名，比如一般来说echo和puts的行为是一样的，但是标准tcl不支持echo命令，可以使用alias来实现这一点。

```
interp alias {} echo {} puts;#为标准tcl puts命令添加一个alias 别名，取名为echo


# other alias exapmles
interp alias {} getIndex {} lsearch {alpha beta gamma delta}

set idx [getIndex delta]
```

18. json

参考：
https://core.tcl-lang.org/tcllib/doc/tcllib-1-18/embedded/www/tcllib/files/modules/json/json.html#1

https://core.tcl-lang.org/tcllib/doc/trunk/embedded/md/toc.md

18.1 json2tcl

json::json2dict工作得还是很好的

```
package require json

set jsonStr { \
  { "photos": { "page": 1, "pages": "726", "perpage": 3, "total": "7257",
    "photo": [
      { "id": "6974156079", "owner": "74957296", "secret": "005d743f82",
"server": "7197", "farm": 8, "title": "Kenya", "ispublic": 1, "isfriend": 0,
"isfamily": 0 },
      { "id": "6822988100", "owner": "52857411", "secret": "56630c18e8",
"server": "7183", "farm": 8, "title": "Gedi", "ispublic": 1, "isfriend": 0,
"isfamily": 0 },
      { "id": "6822909640", "owner": "52857411", "secret": "f4e392ea36",
"server": "7063", "farm": 8, "title": "Local", "ispublic": 1, "isfriend": 0,
"isfamily": 0 }
    ] }, "stat": "ok" }
}

set d1 [json::json2dict $jsonStr]
puts "d1 is :"
puts $d1


foreach {key value} [dict get $d1] {
```

```
    set is [string is list $value]
    puts "$key -- $is -- "
    if {$key == "photos"} {
        foreach {key1 value1} [dict get $value] {
            puts "  $key1 -- $value1"

            if {$key1 == "photo"} {
                foreach value2 $value1 {
                    puts "    ==========$value2"
                    foreach {key3 value3} [dict get $value2] {
                        puts "    =========   $key3 -- $value3"
                    }
                }
            }
        }
    }
    #puts "$key--$value"
}
```

18.2 tcl2json

以下这段代码是直接可以使用的。来自于

https://rosettacode.org/wiki/Rosetta_Code

https://rosettacode.org/wiki/JSON#Tcl

```
package require Tcl 8.6
package require json::write

proc tcl2json value {
    # Guess the type of the value; deep *UNSUPPORTED* magic!
    regexp {^value is a (.*?) with a refcount} \
    [::tcl::unsupported::representation $value] -> type

    switch $type {
    string {
        return [json::write string $value]
    }
    dict {
        return [json::write object {*}[
        dict map {k v} $value {tcl2json $v}]]
    }
    list {
        return [json::write array {*}[lmap v $value {tcl2json $v}]]
    }
    int - double {
        return [expr {$value}]
    }
    booleanString {
```

```
        return [expr {$value ? "true" : "false"}]
    }
    default {
        # Some other type; do some guessing...
        if {$value eq "null"} {
        # Tcl has *no* null value at all; empty strings are semantically
      # different and absent variables aren't values. So cheat!
        return $value
        } elseif {[string is integer -strict $value]} {
        return [expr {$value}]
        } elseif {[string is double -strict $value]} {
        return [expr {$value}]
        } elseif {[string is boolean -strict $value]} {
        return [expr {$value ? "true" : "false"}]
        }
        return [json::write string $value]
    }
    }
}



set d [dict create blue [list 1 2] ocean water book {a b c d}]
puts [tcl2json $d]




dict set ages zhangsan 30
dict set ages lili 28

dict set addr shu  "guanzhou"
dict set addr lv  "suzhou"
dict set addr cc  [list cc1 cc2]
#set addr [list xx0 xx1]

dict set ages misc $addr
puts [tcl2json $ages]
```

输出：输出格式优美，可读性强。

```
{
    "blue"  : ["1","2"],
    "ocean" : "water",
    "book"  : "a b c d"
}
{
    "zhangsan" : 30,
    "lili"     : 28,
    "misc"     : {
        "shu" : "guanzhou",
```

```
        "lv"   : "suzhou",
        "cc"   : ["cc1","cc2"]
    }
}
```

18.3 printtcl

此处是为了打印tcl变量的值，支持普通变量的dict & list 但其实使用json格式来展示打印会更好 tcl2json

```
package require json

set jsonStr { \
  { "photos": { "page": 1, "pages": "726", "perpage": 3, "total": "7257",
    "photo": [
      { "id": "6974156079", "owner": "74957296", "secret": "005d743f82",
"server": "7197", "farm": 8, "title": "Kenya", "ispublic": 1, "isfriend": 0,
"isfamily": 0 },
      { "id": "6822988100", "owner": "52857411", "secret": "56630c18e8",
"server": "7183", "farm": 8, "title": "Gedi", "ispublic": 1, "isfriend": 0,
"isfamily": 0 },
      { "id": "6822909640", "owner": "52857411", "secret": "f4e392ea36",
"server": "7063", "farm": 8, "title": "Local", "ispublic": 1, "isfriend": 0,
"isfamily": 0 }
    ] }, "stat": "ok" }
}

set d1 [json::json2dict $jsonStr]
puts "d1 is :"
puts $d1

proc printtcl {value {prefix ""}} {
    # Guess the type of the value; deep *UNSUPPORTED* magic!
    regexp {^value is a (.*?) with a refcount} \
    [::tcl::unsupported::representation $value] -> type

    switch $type {
    dict {
        dict map {k v} $value {
          puts "$prefix\key:$k"
          printtcl $v "$prefix  "
        }
    }
    list {
        lmap v $value {
            puts "$prefix\list item:\["
            printtcl $v "$prefix  "
            puts "$prefix\list item:\]"
        }
    }
    default {
```

```
        puts "$prefix$value"
    }
    }
}
printtcl $d1
```

输出：

```
d1 is :
photos {page 1 pages 726 perpage 3 total 7257 photo {{id 6974156079 owner
74957296 secret 005d743f82 server 7197 farm 8 title Kenya ispublic 1
isfriend 0 isfamily 0} {id 6822988100 owner 52857411 secret 56630c18e8
server 7183 farm 8 title Gedi ispublic 1 isfriend 0 isfamily 0} {id
6822909640 owner 52857411 secret f4e392ea36 server 7063 farm 8 title Local
ispublic 1 isfriend 0 isfamily 0}}} stat ok
key:photos
  key:page
    1
  key:pages
    726
  key:perpage
    3
  key:total
    7257
  key:photo
    list item:[
      key:id
        6974156079
      key:owner
        74957296
      key:secret
        005d743f82
      key:server
        7197
      key:farm
        8
      key:title
        Kenya
      key:ispublic
        1
      key:isfriend
        0
      key:isfamily
        0
    list item:]
    list item:[
      key:id
        6822988100
      key:owner
        52857411
```

```
      key:secret
        56630c18e8
      key:server
        7183
      key:farm
        8
      key:title
        Gedi
      key:ispublic
        1
      key:isfriend
        0
      key:isfamily
        0
    list item:]
    list item:[
      key:id
        6822909640
      key:owner
        52857411
      key:secret
        f4e392ea36
      key:server
        7063
      key:farm
        8
      key:title
        Local
      key:ispublic
        1
      key:isfriend
        0
      key:isfamily
        0
    list item:]
key:stat
  ok
```

19. 其它注意

1. 标准tcl不支持echo命令（打印信息请使用puts命令），有些工具支持可能是支持自己作了功能增强。