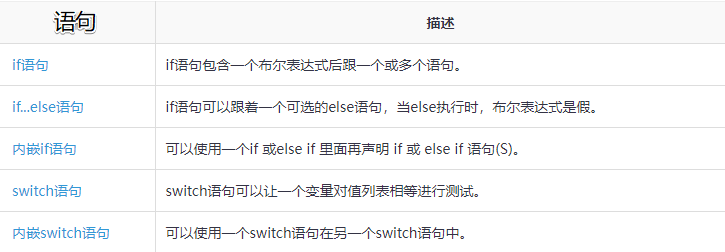
# Tcl判断语句

以下是大多数编程语言中的典型判断结构的一般形式：



Tcl语言在内部使用expr命令，因此我们不需要明确地使用expr语句。

Tcl语言提供以下类型的判断语句：



**？:操作符**

可以使用?:替代if … else语句

Exp1 ? Exp2 : Exp3;

等价于if (Exp1){

Exp2;

}else{

Exp3;

}

set a 10

set b [expr $a == 1 ? 20 : 30]

puts "Value of b is $b" ;#输出 30

set b [expr $a == 10 ? 20 : 30]

puts "Value of b is $b" :#输出20

## if语句

if语句包含一个布尔表达式后跟一个或多个语句。

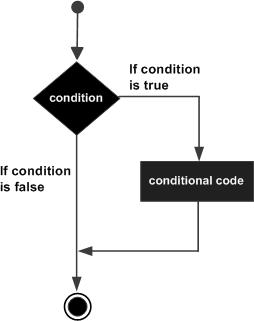
Tcl语言的if语句的语法是：

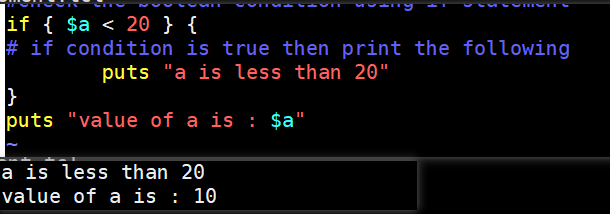
if {boolean\_expression} {

# statement(s) will execute if the boolean expression is true

}

If语句流程如下：





## If else语句

if语句可以跟着一个可选的else语句，else语句块执行时，布尔表达式值是假的。

Tcl语言的if ... else语句的语法是：

if {boolean\_expression} {

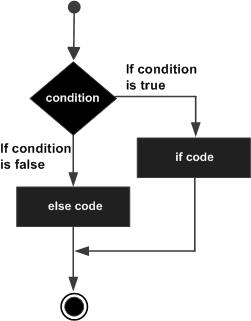
# statement(s) will execute if the boolean expression is true

} else {

# statement(s) will execute if the boolean expression is false

}

流程图如下：



if...else if...else 语句

if语句后可以跟可选的else if ... else语句，使用单个if 测试各种条件if...else if 声明是非常有用的。

当使用if …else if … else语句有几点要注意：

* 一个if可以有零或一个else，它必须跟在else if之后。
* 一个if语句可以有零到多个else if，并且它们必须在else之前。
* 一旦一个 else if 成功，任何剩余else if 或else 不会再被测试。

if...else if...else语句的语法是：

if {boolean\_expression 1} {

# Executes when the boolean expression 1 is true

} elseif {boolean\_expression 2} {

# Executes when the boolean expression 2 is true

} elseif {boolean\_expression 3} {

# Executes when the boolean expression 3 is true

} else {

# executes when the none of the above condition is true

}

## 嵌套if语句

嵌套if语句的语法如下：

if { boolean\_expression 1 } {

# Executes when the boolean expression 1 is true

if {boolean\_expression 2} {

# Executes when the boolean expression 2 is true

}

}

## Switch语句

switch语句可以让一个变量值的列表进行相等测试。每个值被称为一种情况(case)，该变量会被每个case检查。

Tcl语言未加大括号的switch语句的语法如下：

switch switchingString matchString1 {body1} matchString2 {body2} ... matchStringn {bodyn}

添加大括号的switch语句语法如下：

switch switchingString {

matchString1 {

body1

}

matchString2 {

body2

}

...

matchStringn {

bodyn

}

}

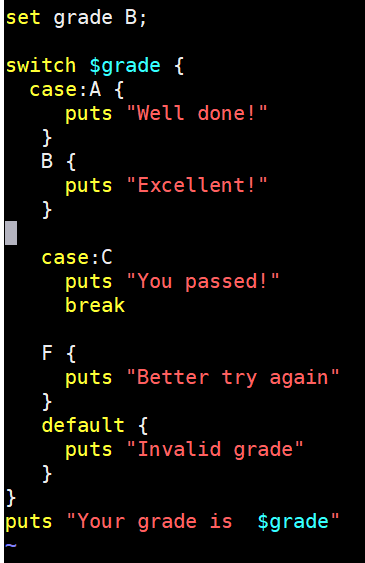
流程图如下：



以下规则适用于switch语句：

* 在switch语句中使用的switchingString通过比较matchString执行匹配的代码块
* 在一个switch内可以任何数量matchString块。
* switch语句可以有可选默认值，必须出现在switch语句的末尾。缺省情况下，可用于执行匹配时没有一个case是匹配的。
* switch语句中可以不写case关键字，也可以不写break。结果一样。

如：



## 嵌套switch语句

嵌套switch语句的语法如下：

switch switchingString {

matchString1 {

body1

switch switchingString {

matchString1 {

body1

}

matchString2 {

body2

}

...

matchStringn {

bodyn

}

}

}

matchString2 {

body2

}

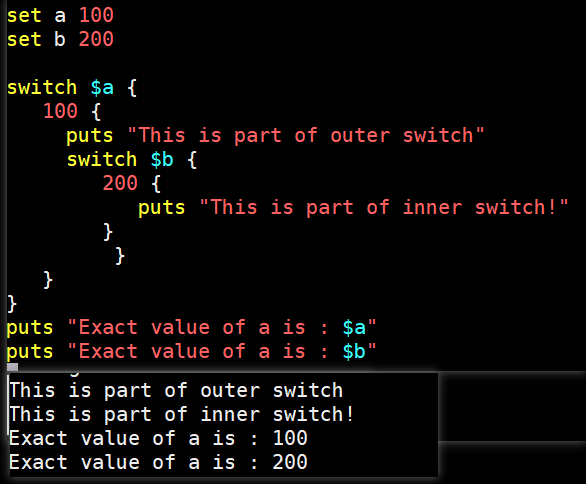
...

matchStringn {

bodyn

}

}



# Tcl循环

在一般情况下，语句是顺序执行的：如在函数内的第一条语句，首先执行，然后是第二条语句 ... 等等。有时候需要执行一个代码块多次，就需要一种循环语句。

循环语句可以执行语句或代码块多次，下面是在大多数编程语言循环语句的一般形式：



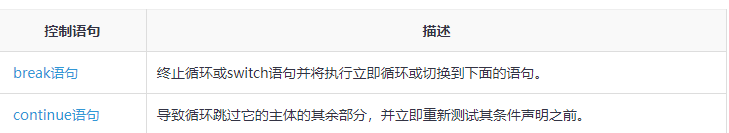
TCL语言中的循环语句如下：



## 循环控制语句

循环控制语句改变其正常的顺序执行。当执行离开范围，在该范围内创建的所有自动对象被销毁。

TCL支持下面的控制语句。



## 无限循环

如果条件永远不会为假那么一个循环就是无限循环。 while循环被经常用来实现此目的。可以通过条件表达式为1，来执行一个死循环。

while {1} {

puts "This loop will run forever."

}

TCL更普遍使用while {1} 构造以表示无限循环。

可以通过按 Ctrl+ C键 终止无限循环。

## While循环

只要给定的条件为真,执行目标语句声明多次。

while循环的语法是：

while {condition} {

statement(s)

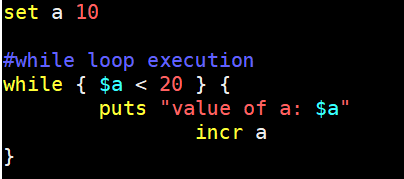
}

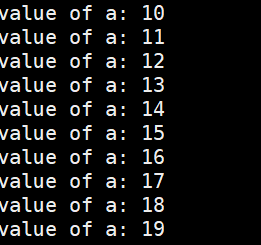
在这里，声明(S)可以是单个语句或语句块。所述条件可以是任何表达，真是指任何非零值。循环迭代当条件为真。

当条件为假，则程序控制进到紧接在循环之后的代码行。

流程如下：







## For循环

for循环的语法是：

for {initialization} {condition} {increment} {

statement(s);

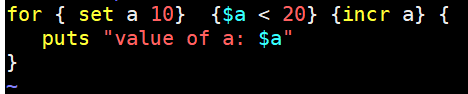
}

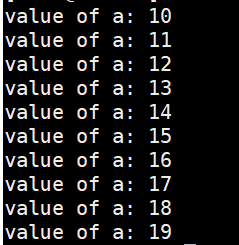
下面是一个for循环控制的流程：

* 初始化步骤首先执行，并且只有一次。这一步可以声明和初始化任何循环控制变量。不需要把一个声明放在这里，只要给定一个分号。
* 接着，条件condition进行了计算。如果为真，则执行循环体。如果是假的，循环体不执行，for循环之后的流程控制跳转到下一条语句。
* 循环的执行主体之后，控制流跳转回到增量语句。此语句可以更新任何循环控制变量。这个语句可以留空，只需要一个分号。
* 条件现在重新计算评估。如果条件为真，循环执行的过程中重新执行(循环体，然后增加步，然后再次计算条件)。之后当条件为假时，for循环终止。

流程图如下：







## 嵌套循环

TCL允许在一个循环内嵌套另一个循环

嵌套的for循环语句的语法如下：

for {initialization} {condition} {increment} {

for {initialization} {condition} {increment} {

statement(s);

}

statement(s);

}

嵌套while循环语句的语法如下：

while {condition} {

while {condition} {

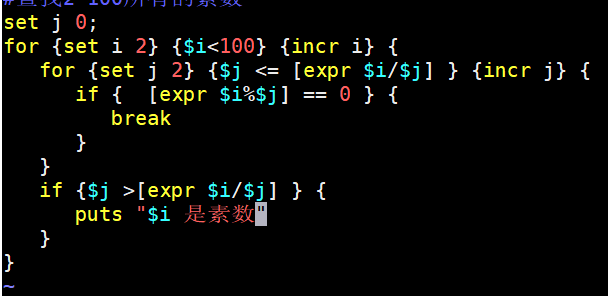
statement(s);

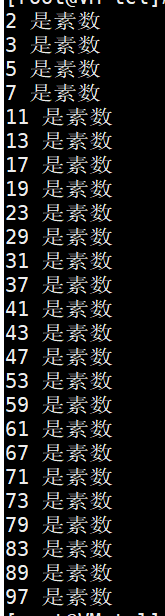
}

statement(s);

}

可以把任何类型循环嵌套在其他类型循环内。例如，for循环可以是一个while循环的内循环，反之亦然。



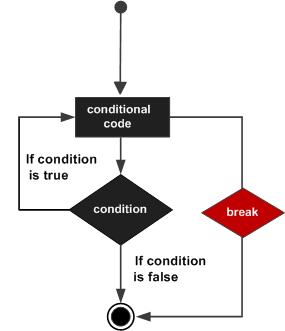


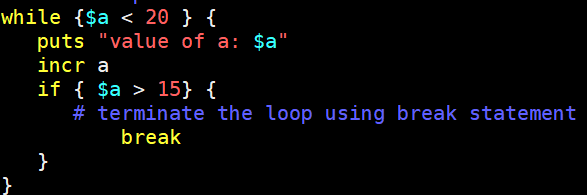
## break语句

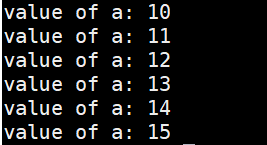
break语句用于终止循环。当循环中遇到break语句，循环立即终止，程序继续循环体后面的语句。

如果使用嵌套循环(即，一个循环在另一个循环体中)，break语句将停止最内层循环的执行，并继续执行外层循环。

流程如下：



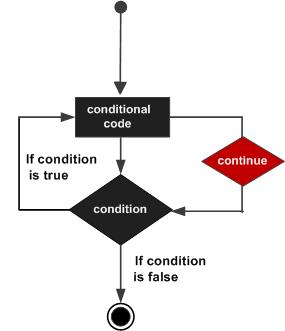


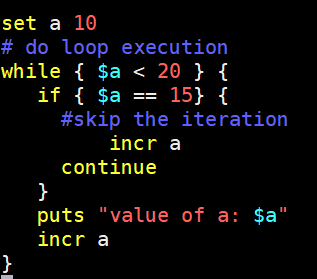


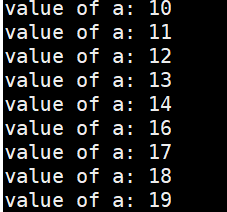
## continue语句

continue语句的工作有点像break语句。但不是强制终止，而是继续执行循环的下一个迭代，跳过中间的代码。

continue流程图如下：



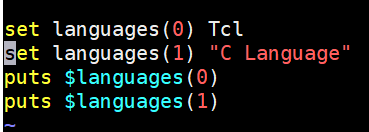


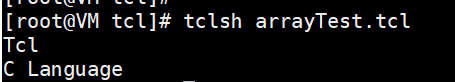


# Tcl数组

数组是一组使用索引对应元素的排列方式。常规数组的语法如下所示。

set ArrayName(Index) value





## 数组大小

计算数组大小的语法：

[array size variablename]

set languages(0) Tcl

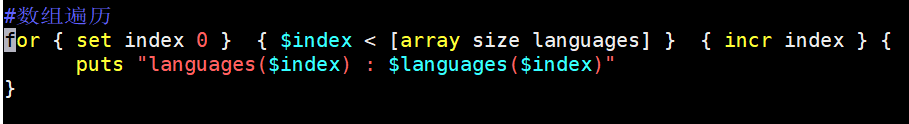
set languages(1) "C Language"

puts [array size languages]

输出2

## 遍历数组

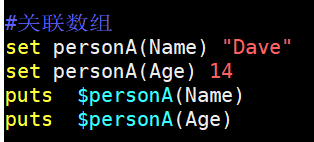
虽然数组索引可以是非连续的，像索引1，然后索引10等指定的值。但是如果它们是连续的，我们可以用数组迭代访问数组的元素。一个简单的数组迭代的打印元素，如下所示。





## 关联数组

在TCL，所有数组本质是相关联的。数组存储并没有任何具体的顺序进行检索。关联数组使用索引但不一定都是数字。一个简单的例子，关联数组与非数字索引如下所示。

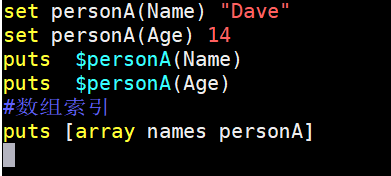




## 数组索引

用于检索数组索引的语法如下

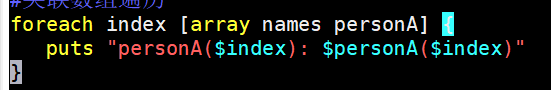
[array names variablename]





## 关联数组迭代

使用数组的索引来遍历数组





# TCL字符串

Tcl 中的原始数据类型是字符串。字符串中可以包含字母、数字、布尔值，甚至是二进制数据。 TCL使用16位Unicode字符，可以包含字母包括非拉丁字符，数字或标点符号。

布尔值的真值可以表示为1，yes 或 true； 0，no 或 false 为假值。

## 字符串表示

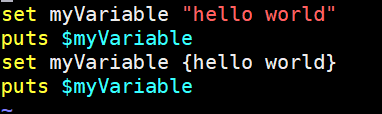
不同于其他语言，在TCL中只有一个单词时，不需要包含双引号。

set myVariable hello

puts $myVariable

输出hello

**当要表示多个单词，要使用双引号或大括号**





## 字符串转义

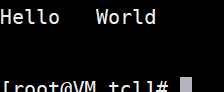
字符文字可以是纯字符（例如，"x"），转义序列（例如" \t"）或通用字符（例如" u02C0"）。

在Tcl中有一些字符，当它们之前是反斜杠时，它们将具有特殊的含义，被用来表示类似换行符（\n）或制表符（\t）。

下面是一些转义字符：



puts "Hello\tWorld\n\n"将输出



## 字符串命令

常用命令列表列如下表：

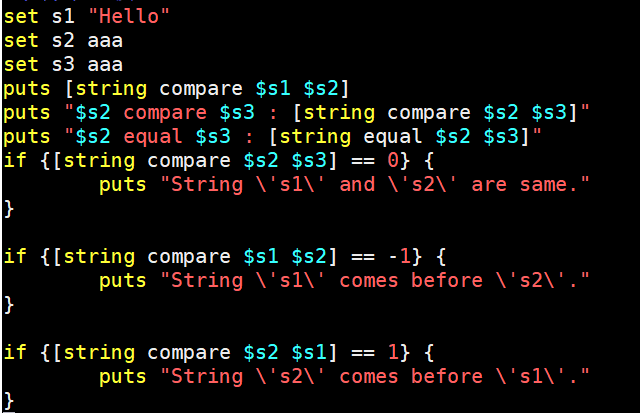


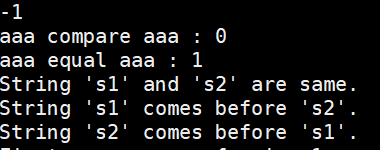
字符串比较

compare

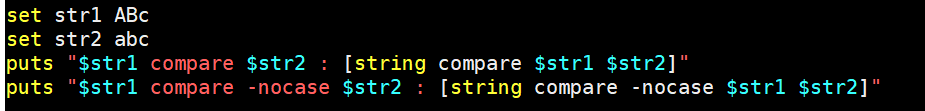
**如果第一个字符串在字典中先于第二个字符串，返回-1**；如果第一个字符串在字典中后于第二个字符串，返回1，如果两者相同，返回0

**string equal**则是对两个字符串进行简单的比较，如两者严格相同，则返回1，否则返回0（与string compare的返回值是不同的）



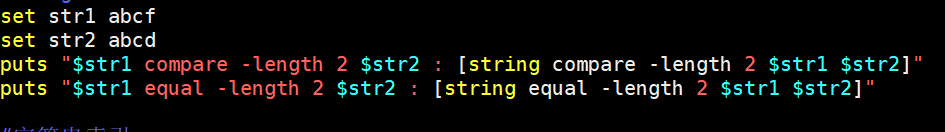


**string compare和stringequal都是区分大小写的，但如果添加选项-nocase，则不再区分大小写**





还可以**通过-length选项指定对字符串的前length个字符进行比较操作**





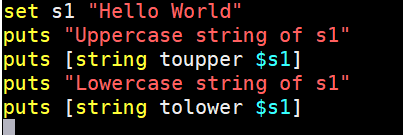
字符串比较的方法中string equal命令耗时最短。

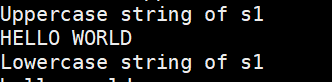
## 字符串长度



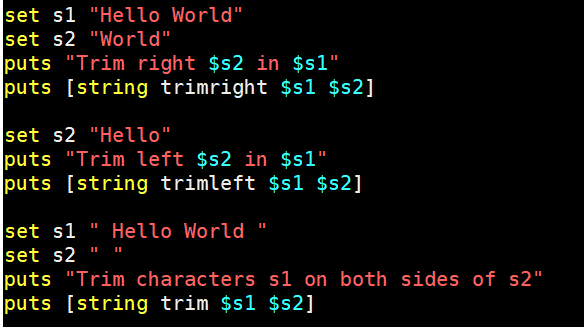


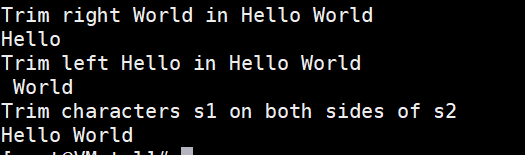
## 处理大小写



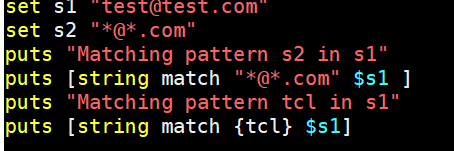


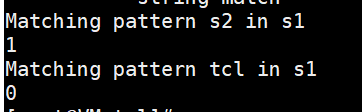
## 删除字符





## 匹配字符串





## append

set s1 "Hello"

append s1 " World"

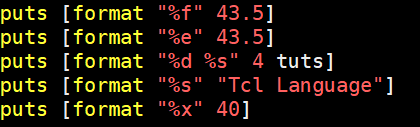
puts $s1

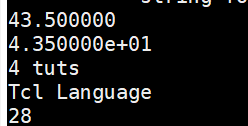
输出Hello World

## Format命令

Tcl显示格式format说明如下

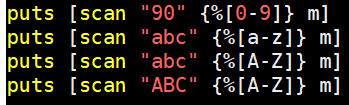






## Scan命令

scan命令用于根据格式说明符解析字符串





puts [scan "90" {%[0-9]}] ;# 90

puts [scan "90abc" {%[0-9]}] ;# 90

puts [scan "9a0abc" {%[0-9]}] ;# 9

puts [scan "90" {%[0-9]} m] ;# 1

# TCL列表

列表是Tcl的基本可用数据类型之一。它是用于表示项目的有序集合。它可以包括不同类型的在同一列表的项目。此外，一个列表可以包含另一个列表。

要注意的一个重要的事情是，要避免大的列表，在这种情况下，必要时可使用数组代替。

## 创建列表

一般语法如下。

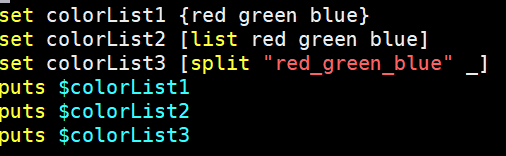
set listName { item1 item2 item3 .. itemn }

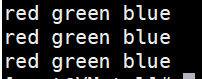
# or

set listName [list item1 item2 item3]

# or

set listName [split "items separated by a character" split\_character]





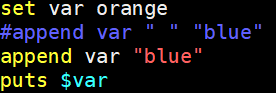
## 追加项目到列表

语法：append listName split\_character value

# or

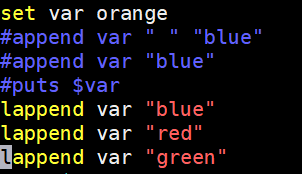
lappend listName value

append不会自动添加空格。



输出orangeblue

Lappend会自动在上个元素间添加空格





## 列表长度

语法如下。

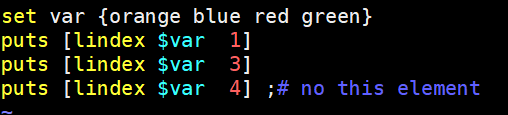
llength listName

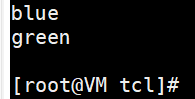


输出4

## 列表索引

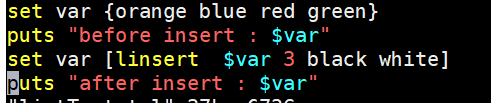
语法：lindex listname index

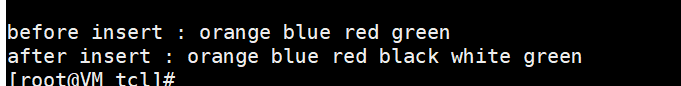




## 在索引处插入项

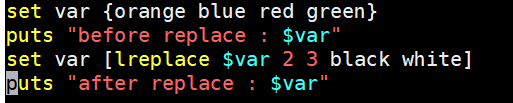
语法：linsert listname index value1 value2..valuen





## 替换指定索引处的元素

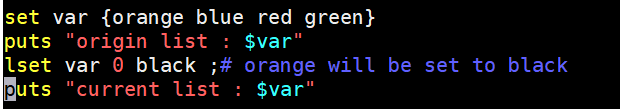
语法：lreplace listname firstindex lastindex value1 value2..valuen





## 设置指定索引处的元素

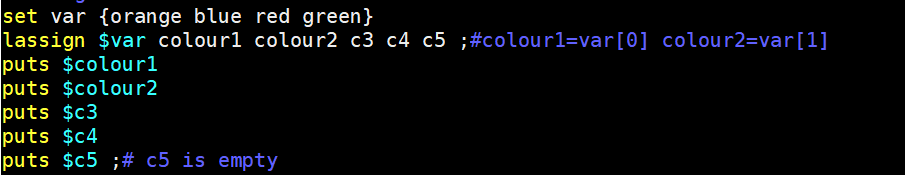
语法：lset listname index value

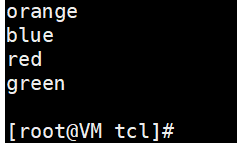




## 复制列表元素的值到变量

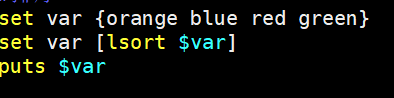
语法：lassign listname variable1 variable2.. variablen





## 列表排序

语法：lsort listname





输出的内容按照单词首字母进行排序。

# TCL字典

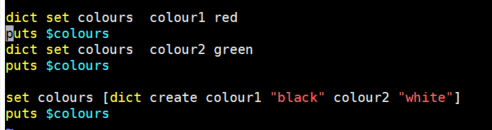
字典是用于映射键值对(key-value)的关系。常规字典的语法如下所示。

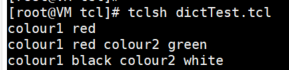
dict set dictname key value

# or

dict create dictname key1 value1 key2 value2 .. keyn valuen

创建字典：





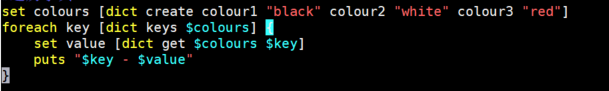
## 字典大小

语法：[dict size dictname]



输出2

## 遍历字典





## 根据键检索值

语法：[dict get $dictname keyname]



输出：black

## 获取字典所有的键

语法：[dict keys $dictname]





## 获取字典所有的值

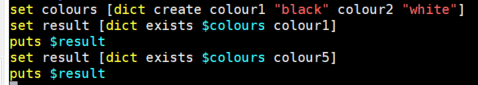
语法：[dict values $dictname]





## 判断键是否存在

语法：[dict exists $dictname keyname]



# TCL过程

Tcl过程类似于其他编程语言的函数，但又和函数定义不同。过程的定义使用{}而非()，这是最大的差别。

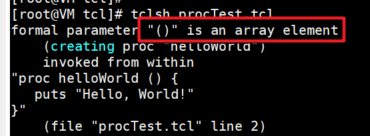
创建简单过程的语法如下：

proc procedureName {arguments} {

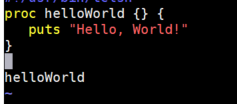
body

}

使用()会报错

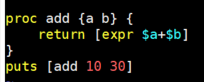


正确示例如下：



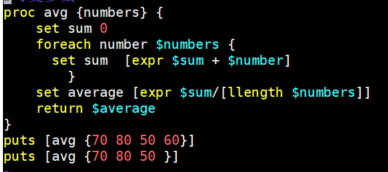
输出Hello, World!

## 过程的多个参数





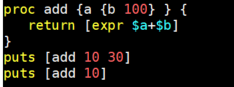
## 过程的可变参数





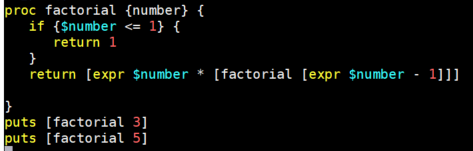
## 默认参数

默认参数是用来提供一种可如果未提供任何参数值时，可以使用默认值。有时称为隐式参数，一个例子如下所示。





## 递归过程





# TCL包

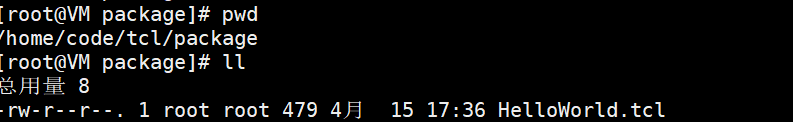
软件包用于创建可重用的代码单元。包由一组提供特定功能的文件组成。该文件集合由一个包名称标识，并且可以具有多个相同文件的版本。该包可以是Tcl脚本，二进制库或两者的组合的集合。

包使用命名空间的概念来避免变量名和过程名称的冲突。

## 创建包

包可以以最小的两个文件的帮助下被创建。一个文件包含包代码。另一个文件包含声明包的索引文件包。

1.在包文件夹/home/code/tcl/package创建代码，如HelloWorld.tcl



内容如下：

# Create the namespace

namespace eval ::HelloWorld {

# Export MyProcedure

namespace export MyProcedure

# My Variables

set version 1.0

set MyDescription "HelloWorld"

# Variable for the path of the script

variable home [file join [pwd] [file dirname [info script]]]

}

# Definition of the procedure MyProcedure

proc ::HelloWorld::MyProcedure {} {

puts $HelloWorld::MyDescription

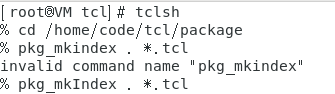
}

package provide HelloWorld $HelloWorld::version

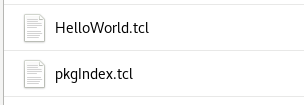
package require Tcl 8.5

2.创建包索引

打开tclsh。切换到HelloWorld目录，并使用pkg\_mkIndex命令创建索引文件



执行完会在当前目录生成一个pkgIndex.tcl文件



3.添加目录到AUTOPATH

使用lappend命令来添加程序包到全局列表



4.添加软件包



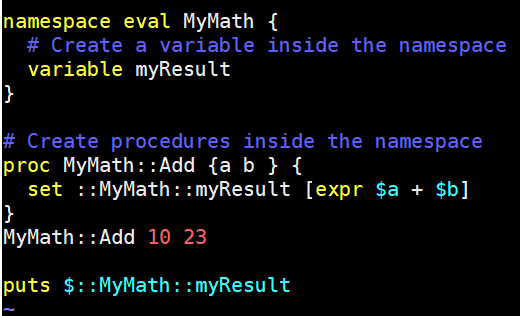
5.调用过程



# TCL命名空间

命名空间是用于组变量和过程的一组标识符的容器。命名空间可从Tcl 8.0版开始使用。引入命名空间之前，有一个全局范围。现在有了命名空间，可以分区全局范围。

## 创建命名空间

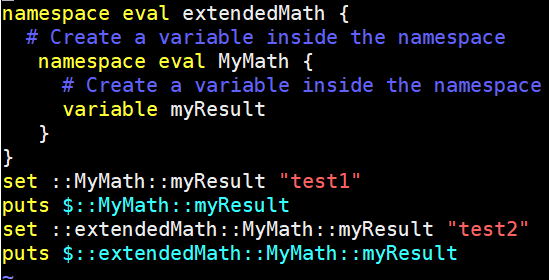


执行输出



## 嵌套命名空间

TCL允许命名空间的嵌套。一个简单的例子如下：





## 导入和导出命名空间

上面的例子可以看到，使用了大量的范围解析运算符，它们的使用使程序变得更复杂。可以通过导入和导出命名空间避免这种情况。

## 删除命名空间

通过使用forget子命令删除导入的命名空间

namespace forget MyMath::\*

# TCL文件IO

TCL支持使用内置命令对文件处理，如：open, read, puts, gets 和 close.

## 打开文件

Tcl使用open命令在Tcl中打开文件。打开文件的语法如下

open fileName accessMode

文件名是字符串，accessMode可以是以下值之一：



## 关闭文件

关闭文件使用close命令，语法如下

close fileName

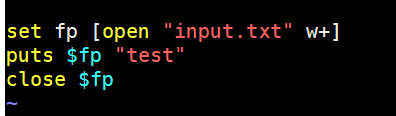
当程序使用完一个已经打开的文件之后，必须关闭。在大多数情况下，文件不需要被显式地关闭;它们会自动关闭，文件对象会自动终止。

## 写入文件

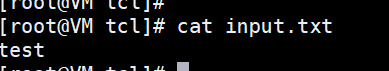
puts命令用于写入一个打开的文件。

puts $filename "text to write"

一个简单写入文件的例子如下所示

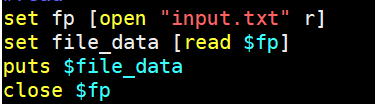


执行完后会在当前目录下生成一个input.txt文件

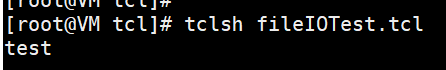


## 读取文件

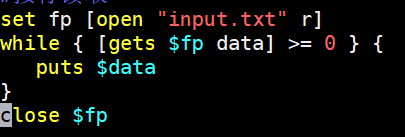
语法：set file\_data [read $fp]

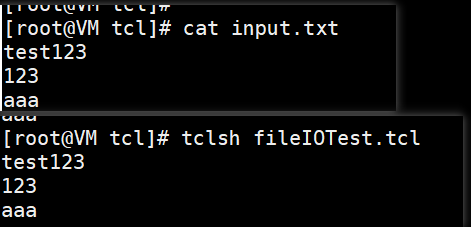


Read会一次读取文件全部内容。



按行读取：





# TCL错误处理

Tcl的错误处理设置有error 和 catch命令。

**Error语法**

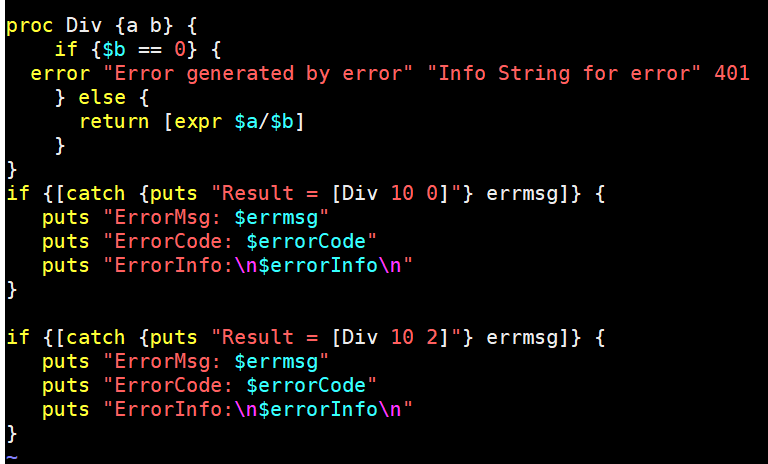
error message info code

在上面的 error命令语法，message是错误信息，info是在全局变量errorInfo中设置，code是在errorCode设置的全局变量。

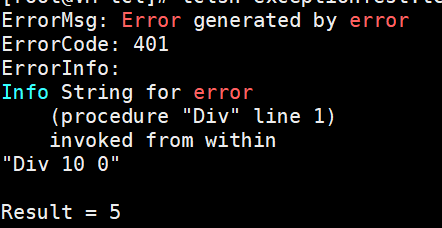
**Catch语法**

catch script resultVarName

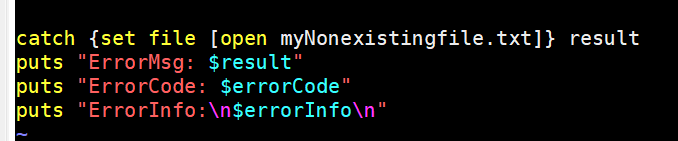
在上述catch语法中，脚本是要执行的代码，resultVarName保存结果。如果没有错误catch命令返回0，如果有一个错误，返回1。

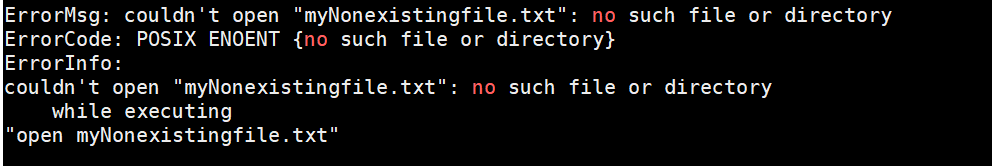


**执行输出**



如上所示我们可以创建自己的自定义错误消息。同样地，也能够捕捉由Tcl所产生的错误。一个例子如下所示





# TCL内置函数

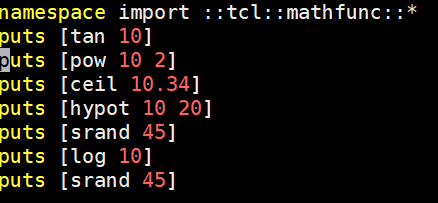
Tcl提供了一些内置的函数，用于各种操作。主要包括

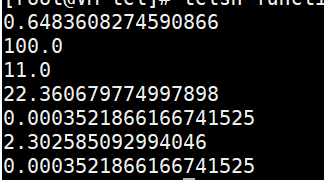
* 列表处理函数。
* 字符串处理函数。
* 数组处理函数。
* 字典处理函数。
* 文件I/O处理函数。
* 命名空间和包处理函数。
* 数学处理函数。
* 操作系统处理函数。

## 数学函数



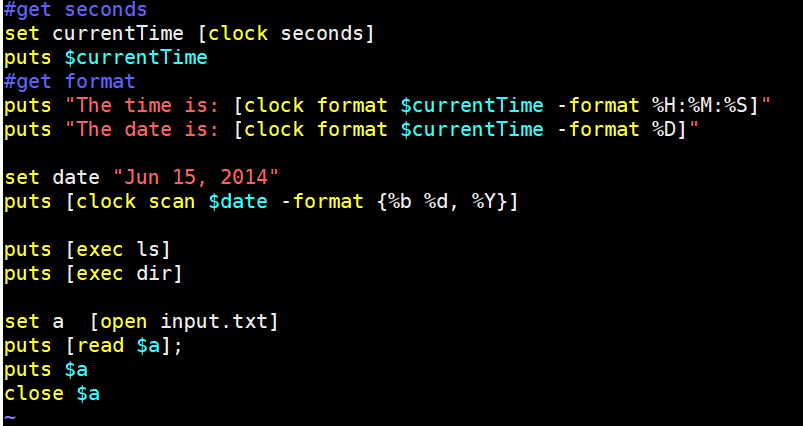
使用内置数学函数的例子如下：

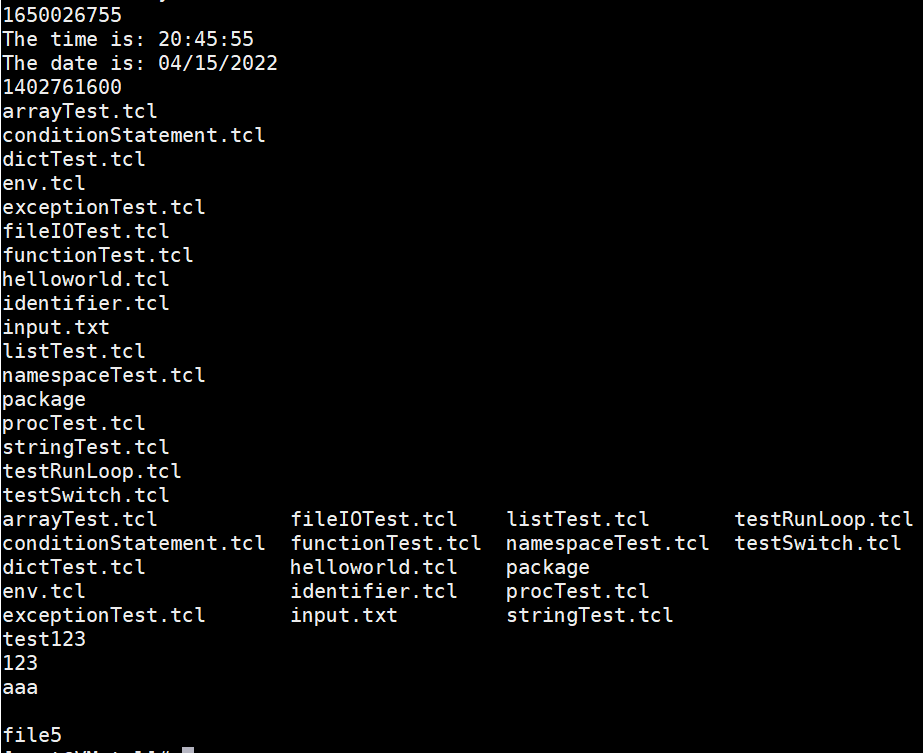




## 系统函数

* **clock seconds**， 以秒为单位返回当前时间。
* **clock format**，将秒数格式化为日期和时间。
* **clock scan**，可扫描输入字符串并将其转换成秒。
* **open**，用于打开文件。
* **exec** ，用于执行系统命令。
* **close**，用于关闭文件。





日期和时间的格式化字符如下：



# TCL正则表达式

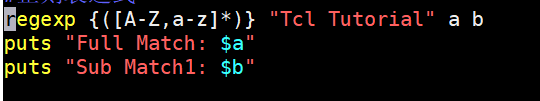
“regexp”命令用于匹配Tcl中的正则表达式。正则表达式是包含搜索模式的一系列字符。它由多个规则组成，如下：

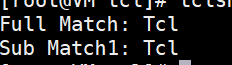


**语法：**

regexp optionalSwitches patterns searchString fullMatch subMatch1 ... subMatchn

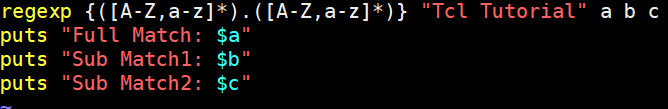
看一个简单的例子，当遇到除了字母外任何其他字符时，搜索将被停止并返回。

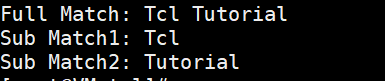




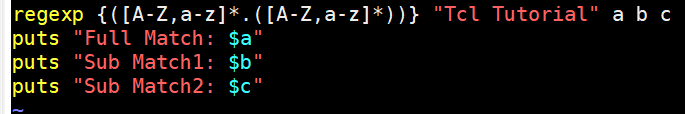
## 多元组匹配

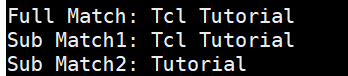
下面的例子演示了如何匹配多个元组





一个子模式包含多个模式

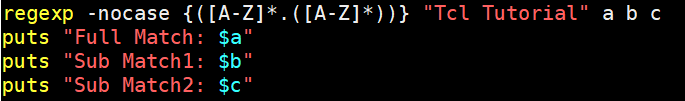


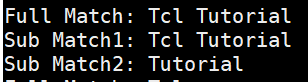


## 正则表达式可选命令

在Tcl中提供的参数有

* **-nocase :** 用于忽略大小写。
* **-indices :** 匹配的子模式，而不是匹配的字符存储的位置。
* **-line :**新行敏感匹配。换行后忽略字符。
* **-start index :**搜索模式开始设置偏移
* **-- :**标志着开关的结束





# Git仓库

<https://github.com/Angeliau/TCL-demo.git>

