Contents

[Matlab 1](#_Toc440974994)

[基本信息 1](#_Toc440974995)

[数据类型 2](#_Toc440974996)

[元胞数组 2](#_Toc440974997)

[结构数组 2](#_Toc440974998)

[矩阵: 2](#_Toc440974999)

[字符串：字符的行向量 3](#_Toc440975000)

[数值型 3](#_Toc440975001)

[逻辑型 4](#_Toc440975002)

[函数句柄 4](#_Toc440975003)

[函数使用技巧： 4](#_Toc440975004)

[全局变量 5](#_Toc440975005)

[二维绘图 6](#_Toc440975006)

[句柄图形对象 6](#_Toc440975007)

[Tcl/Tl 8](#_Toc440975008)

# Matlab

## 基本信息

自动补全函数名功能，键入一部分函数名，然后按tab键，弹出所有可能的已有函数名列表.

rootDirectory = matlabroot ; %调用matlabroot命令返回matlab的安装路径

filePath = fullfile(matlabroot, ‘toolbox’,’matlab’) ; %调用fullfile来产生完整路径

cd(filePath); %调用cd将当前目录切换到filePath路径下

addPath([filePath ‘/local/myfiles’]); %调用addPath在搜索路径下添加文件夹

which matlabrc.m; %调用which查看文件其所在的目录

[userview, sysview] = memory ; %用户信息和系统信息

pack; %将base空间变量以内存紧凑的形式整理，避免因存储空间的断续造成连续的内存地址不足

pause; %暂停程序执行

dirArr = dir(‘.\\*.m’) ; %获取当前目录下满足条件的子目录和文件,存在结构类型

for I = 3: size(dirArr)

isdir(dirArr{i}.name) ; %测试是否为目录，非目录，即文件

end

ls \*.doc; %列出当前目录满足条件的所有文件夹和文件名

dirArr = ls(‘..\\*.doc’) ;

for i=1:size(dirArray)

isdir(disArray(I, :) ); %测试是否为目录，非目录，即文件

end

cd; %更改或显示当前工作路径

fileName = ‘..\harbin.jpg’ ;

imfinfo(fileName) ; %图片信息

[X, map] = imread(filename) ; %读图像

Imshow(X); %显示图像

Image(X) ; %显示图像

Axis off ; %隐藏坐标轴

Imwrite(X, ‘..’harbin-copy.jpg’) ;

Image and imshow的区别：

Imshow不会扩展图像数据，即不会拉伸图像使其铺满坐标轴，而是改变坐标轴宽高比，使其适应图像数据。主要是无失真等比例显示图像

Image不会改变坐标轴的大小尺寸，而是扩展填充图像矩阵，使其铺满坐标轴区域。主要是用来显示行列分辨率差别很大的矩阵

低级文件的操作：

Fopen, fclose, fread, fwrite, fprintf, fscanf,具体查资料

## 数据类型

数值型，逻辑型，字符数组，结构数组，单元数组，函数句柄

## 元胞数组

存储不同类型的数据

cellArr{2,2} = “dd” ;

## 结构数组

同元胞数组一样，可以存放不同类型的数据。句柄结构就是采用结构数组

structArr(2,2).name = “qzlin” ;

structArr(2,2).number = “123” ;

## 矩阵:

Ind = find(X) ; %保存元素在矩阵中的线性索引值，因为在matlab中，矩阵是按列存储的，ind的值表示元素在矩阵中按列存储时的位置。

[m n] = find(X) ; %行列下标

[m n] = ind2sub(size(X), ind) ; %将线性索引转换为行列下标

Ind = sub2ind(size(X), m, n) ; %行列下标转换为线性索引

X(:, 1:2) = []; %删除第一列和第二列

Num = length(X) ; %矩阵行列数的最大值 max(size(X))

Num = size(X, 1) ; %矩阵的行数

Num = size(X, 2); %矩阵的列数

V = max(X) ; %取得矩阵每一列的最大值，组成行向量

V = max(X, [], 1)；%每列的最大值组成行向量, 容易记得，这是因为matlab矩阵按列存储，所以1,表示列处理，2表示行处理

V = max(X, [], 2); %每行的最大值组成列向量

预分配数组空间，非double型，需要明确指出，

X = zeros(100, 100, ‘uint8’) ; %要比X = uint8(zeros(100, 100))快很多

## 字符串：字符的行向量

Str = ‘abc’ ;

largeStr = [str ‘def’]

字符串操作函数同c运行库字符串操作函数

isDir = isdir(‘D:\work’) ; %判断输入变量是否为文件夹

isEmpty = isempty(X) ; %判断数组是否为空

strcat与连接运算符[]区别：

strcat先将每个字符串尾部的空格去掉再连接，而[]会原封不动地将字符数组连接起来。

Str1 = ‘a ‘;

Str2 = ‘bc’ ;

[str1 str2] ; %结果为’abc’

Strcat(str1, str2) ; %结果为’a bc’

Str = ‘t=0:.1:2\*pi; x=sin(t);’ ;

eval(str) ; %执行字符串表达式（lamda表达式），注意只能在当前空间内

evalin(‘base’, str) ; %执行字符串表达式，evalin可以在基本工作空间内，也可以在当前空间

evalin(‘caller’, str) ; %注意在当前空间内

feval(@sin, 0:.1:2\*pi) ; %执行函数句柄@sin

str2num和str2double字符串与数值的转换,但二者有区别：

str2num为矩阵运算，可以生成数值矩阵

str2dboule为标量运算，只能生成一个数值。

Datestr(date) ; %获取日期作为字符串

如何批量产生字符串001.jpg,002.jpg,…100.jpg?

Str1 = sprint(‘%03d.jpg’, [1:100]) ; %格式化字符串，0表示前导填充0,3表示域宽，d表示十进制记数

Str2 = reshape(str1, 7, 100) ; %将字符串数组变成7\*100大小

picName = str2’ ;

for I = 1:size(picName, 1)

picName(I, :) ;

end

## 数值型

每种数据类型以矩阵或数组形式存在，最小尺寸是[0 0],能扩展任意大小的n维数组。

X = uint8([1 2 3]) .\* uint8(3) ; %同数据类型的整数或整数数组，运算结果的数据类型和操作数相同。

X = uint32([1 2 3]) .\* 75.49 ; %整数或整数数组与double型浮点数，运算结果的数据类型与整数操作数一样

Isinteger(x);

Isfloat(x) ;

Whos x; %显示数据类型

## 逻辑型

1,0表示真假，常用来表示掩模

x>0 ; %正数掩模

x(x>0); %提取正数

## 函数句柄

类似函数指针，包含函数的路径，函数名，类型以及可能存在的重载方法等映射信息，函数调用的优先原则：

变量 -> 子函数 -> 私有函数 -> 类构造函数 -> 重载方法 -> 当前目录中的同名函数 -> 路径中其他目录的函数。

（如果要查询同名函数中空间哪个被调用，可以用which函数查询）

函数句柄可用来标识子函数，私有函数和嵌套函数，这些函数对用户来说都是“隐藏”的

function out = myFunction(select)

switch select

case ‘case1’

out = @fun1 ;

case ‘case2’

out = @fun2 ;

otherwise

out = [] ;

end

function a = fun1(b,c)

…

function d = fun2(e,f)

…

fhandleInfo = functions(@myFun) ; %函数句柄的信息：函数名，类型，文件名等

Str = func2str(fhandle) ; %由函数句柄构造函数名字符串

Fhandle = str2func(‘str’) ; %由函数名字符串构造函数句柄

[y1, y2] = feval(fhandle, x1, x2) ; %执行函数句柄

## 函数使用技巧：

function [out1, out2, out3] = myfun(in1, in2) ;

调用函数时，其返回参数的个数可以少于函数定义的返回参数的个数，如：

[value1] = myfun(20, 30) ; %返回第一 个输出参数的值

[value1 value2] = myfun(20,30); %返回前两个输出参数的值

Varargin and varargout是可变长度的元胞数组，可以用来获取函数的输入和输出参数，nargin and nargout可以用来获取函数调用时输入和输出参数的个数，他们都是matlab预定义的专用参数。

Function [out1 out2 varargout] = myfun(in1, in2, varargin)

error(nargchk(2, 5, nargin)) ; %在函数入口处，验证输入参数个数

error(nargoutchk(2,5,nargin));%在函数入口处，验证输出参数个数

optionArgumentNum = size(varargin, 2) ; %取得可变参数个数，这里等价于nargin-2

nargin ; %函数总的输入参数个数

for I = 1:size(varargin, 2)

varargin{i} ; %获取可变输入参数

end

nargout ; %函数总的输出参数个数

for i=1:nargout-2

varargout{i} = …; %设置可变输出参数

end

open(D:\work\’matlab.mat’) ; %打开文件matlab.mat，当然还可以打开，\*.doc, \*.exe, \*.pdf, \*.ppt, \*.xml

! start notepad.exe 1.txt ; %调用外部程序，打开文件，同时返回

Winopen(1.txt) ; %调用window外壳命令，打开文件，和双击该文件效果一样

## 全局变量

首先要定义全局变量，如果在函数栈空间里要调用全局变量，需要先声明，才能调用,全局变量一经定义，一直存在，直到用户调用clear global 命令才能清除

Global globalVar1

Function myfunction1

Global globalVar1

globalVar1 = 10 ;

clear global globalVar1 ; %清除该全局变量

switch expression

case val1

statements1 ;

case val2

statements2 ;

otherwise

statementsn ;

end

注意：expression可以为数值，字符或字符串，val1,val2可以数值，字符，字符串，多个数值的组合，多个数值之间用｛｝如case {3, 4}

## 二维绘图

X = [-pi : 0.01 : pi]’ ;

Plot([x x], [sin(x) cos(x)]) ; 同时绘制多条曲线，当X,Y为维数相同的实数矩阵时，每列绘制一条曲线。

Stem([x x], [x^2 x^3]) ; %火柴杆图

Subplot(3, 2, [5 6]) ; %绘制图，占用5和6索引的空间

如何legend, title, texlabel,text,xlabel,ylabel见资料

Gtext({‘I am ‘, ‘qzlin’}) ; %在鼠标单击处放置两行字符串

Annotation(‘line’, x, y) ; %画线，从点[x(1) y(1)] -> [x(2) y(2)]， 还可以显示文本框，圆，矩形等

## 句柄图形对象

在matlab中，由图形命令产生的每一个对象都是图形对象，可以被单独操作，图形包括很多对象，它们组合在一起，形成有意义的图形，图形对象按父对象和子对象组成层次结构。

根（屏幕）

图形窗口

坐标轴

面板和按钮组

UI控件

工具栏

右键菜单

表格

菜单

组

核心

图像

光线

曲面

矩形

面片

线条

文本

uibuttongroup

root

figure

axes

uipanel

uicontrol

uitoolbar

uicontextmenu

uitable

uimenu

hpgroup

核心

image

light

surface

rectangle

surpface

line

text

Root，计算机屏幕，是所有其他对象的父对象，主要保存一些系统状态和设置信息（类比面向对象object）

图形窗口对象，也称为figure对象，它表示整个图形窗口，是根的直接子对象，

第3层为坐标轴对象和用户接口对象，是figure的直接子对象

坐标轴对象是核心对象和组对象的父对象，用于数据的可视化

用户接口对象（也称为UI对象）用于matlab与用户之间的交互操作

第4层包括核心对象和组对象

核心对象为所有绘图的基本元素。

组对象为多个核心对象组合而成的坐标轴子对象。例如，图形的注释(annotation)，插图（legend），直方图(bar),火柴杆图(stem)

对象都有一个独一无二的句柄，通过操作句柄，可查看对象所有属性或修改大部分属性。根对象句柄值为0，图形窗口的句柄值默认为正整数，其他对象的句柄值为系统随机产生的正数。

每个图形对象都有一个属性列表，记录了该图形对象所有的信息。这个属性列表实质上是一个结构体，字段名为对象的属性名，字段值为对象的属性值。

objProperty = get([h1 h2 …]) ; %获取对象数组[h…]的属性列表

objFactory = get(0,’Factory’) ; %获取GUI对象所有属性的出厂值，只读

objDefault = get(h, ‘Default’) ; %返回对象h的所有默认属性值。

Set([h0 h1], ‘Units’, ‘normalized’) ; %设置图形对象的属性值

Set(h) ; %查看对象的所有可设属性值

%设置对象的属性，采用结构体设置法。

objPro = get(figure) ;

objPro.Units = ‘pixels’ ;

objPro.MenuBar = ‘none’ ;

get(gcf) ; %gcf获取当前图形窗口的句柄，

findobj ; %返回根对象及其子对象的句柄

delete(gcf) ; %delete删除文件或图形对象

clf(h, ‘reset’) ; %删除窗口中的对象，重设窗口的属性，并返回窗口的有效句柄

%控制程序执行

Uiwait(h); %暂停程序执行直至uiresume(h)

Uiresume(h) ;

Waitfor(h) ;

Waitforbuttonpress ; %暂停当前窗口内按键或鼠标单击

Pause ; %程序暂停执行，直到键盘按下任意键

Pause(n) ; %程序暂停n秒，

[x, y, button] = ginput ; %返回当前坐标轴内所选点的xy坐标,鼠标单击类型或键盘按键名称,坐标位置是相对于左下角，左键返回1,中键返回2，右键返回3

How to draw circle, eclipse?

Theta = linspace(0, 2\*pi, 100) ;

X = x0 + r\*cos(theta) ;

Y = y0 + r\*sin(theta) ;

Plot(x, y) ;

%根据参数方程 画椭圆

Theta = linspace(0, 2\*pi, 100) ;

Phi = 30/180 \* pi ; %方向角

X = x0 + r\*cos(theta+phi) ;

Y = y0 +r\*sin(theta+phi) ;

%根据解析方程画椭圆(X0,Y0,LongAxis,ShortAxis,Angle)

t1=0:.02:pi;  
t2=pi:.02:2\*pi;  
z1=exp(i\*t1);  
z2=exp(i\*t2);  
z1=(LongAxis\*real(z1)+i\*ShortAxis\*imag(z1))\*exp(i\*Angle);  
z2=(LongAxis\*real(z2)+i\*ShortAxis\*imag(z2))\*exp(i\*Angle);  
z1=z1+Xcenter+Ycenter\*i;  
z2=z2+Xcenter+Ycenter\*i;  
plot(z1,'r')  
hold on  
plot(z2,'r')  
hold off  
grid on

# Tcl/Tl

Tcl(tool command language)是一种用于控制和扩展应用程序的动态语言（也称脚本语言），它的解释器是c函数库，可以很容易地整合到应用程序中。

Tk扩展了Tcl内核的功能，增加了构建用户界面的命令

优点：与低级的工具集相比，要使用Tcl and Tk所需要学习的知识更少，代码数量和开发时间减少90% ; Tcl是解释语言，使用Tcl应用程序时，可以在运行中生成和使用新的脚本，而无需重新编译和重启应用程序；Tcl还可以让应用程序很容易地拥有强大的脚本语言，要为一个已经存在的应用程序添加脚本能力，只需要实现几条新的Tcl命令，用来为应用程序提供相应的基本功能。

tclsh = tcl shell for tcl

wish = windowing shell for tk

语法：

每一条命令都由一个或多个单词组成，单词之间用空格或制表符隔开。

命令由换行符或分号隔开

每条Tcl命令都返回一个结果，如果该命令没有有意义的结果，它会返回一个空字符串。

Tcl处理一条命令分两步：解析和执行。

在解析阶段，Tcl解释器应用解释命令的十多条规则，将命令分解为单词，并执行替换，在每条例命令进行解析的方法是完全一样的，在解析阶段，Tcl解释器不认为各单词的值有任何具体意义，Tcl只是进行一系列简单的字符串操作。

在执行阶段，命令中的各个单词都有了具体的意义，Tcl把第一个单词作为命令名称，检查这个命令是否已经定义，并且查找完成该命令功能的命令过程。如果命令已经定义，Tcl解释器就调用该命令过程，把命令中全部单词传递给该过程。

Tcl解析一条命令时，只从左向右解析一次，进行一轮替换。每一个字符只会被扫描一次。

每一个字符只会发生一层替换，而不会对替换后的结果再进行一次扫描替换。

Tcl解释器在解析阶段执行替换规划，替换可以发生在命令中的任何一个单词上，包括命令名本身，在一个单词中也可以进行任意多个替换

变换替换：由$符号引发，将Tcl变量的值插入单词中。

变量替换可以在一个单词中的任何位置进行，可以进行任意多次。如expr $result\*$base ;

变量名由$符号后面所有数字、字母以及下划线组成。

变量替换可以用于产生新的名称：如下.b1 .b2 .b3 .b4 .b5

foreach i{1 2 3 4 5} {

button .b$i

}

命令替换：由[]表示，把一个单词的部分或全部替换为一个命令的结果。[]内的字符必须构成有效的Tcl脚本，脚本可以包含任意多条命令，命令之间用换行符隔开，也可以用分号隔开。括号和它们之间的所有字符会被替换为脚本的结果。

双引号引用：取消""内的命令分隔符如空格，分号的特殊解释。""本身并不是单词的一部分，它们只是定界符而已。变量替换、命令替换以及反斜线替换在双引号中正常进行。如

set msg "a is $a; the square of a is [expr $a\*$a]"

大括号引用：取消{}内的所有特殊字符的特殊意义。变量替换、命令替换以及反斜线替换在大括号中都不会进行。也就是说，如果一个单词以左大括号开头，那么直到与它配对的右大括号为止，所有字符都将被原封不动地识别为这个单词的值。这个单词中不会发生任何替换。

应用场合：在过程定义和命令流控制中，经常会用到嵌套的大括号来把一个或多个参数表达为需要处理的脚本。因为某段脚本需要作为一个参数出现，它就必须被大括号括起来。而在这个作为参数的脚本中，又可能包含其他的控制流命令，这些命令的一些参数也可能是脚本。

反斜线替换：用于向单词中插入像换行符这样的特殊字符，以及像[、$这样的会被Tcl解析器认为是有特殊含义的字符。反斜线接换行符可以用于将一条很长的命令写成几行，如：

pack .base .label1 .power .label2 .result \

-side left -padx 1m -pady 1m

替换最好的用法就是最简单的用法。如果在同一条命令中使用太多的替换，特别是如果使用太多反斜线，代码会变得不可读。也不可靠，最好把那条繁琐的命令分解为多条命令，以更简单的层次关系管理参数（Tcl提供了一些命令如format,subst and list用于协助处理复杂的替换情况）。另一个值得考虑的方法是建立“过程”，将复杂的操作隔离开来。这样编写的脚本通常比“内联”所有操作的脚本更易读，更容易维护。

注释：注意注释符必须出现在Tcl预期将获得命令的第一个字符的位置上。如

# This is a comment

set a 100 ; # This is a comment

proc countdown {x} {

puts "Running countdown"

# while { $x >= 0 } {

# puts "x = $x"

# incr x -1

# }

或者采用如下注释

proc countdown {x} {

puts "Running countdown"

if 0 {

while { $x >= 0 } {

puts "x = $x"

incr x -1

}

}

如果Tcl是动态语言，创建变量，数组等，如果不存在，则创建，如果存在则显示

变量：

set a 12.6 ; #set可以用来创建、读取和修改变量

unset a -nocomplain ; #删除变量a

#在变量的末尾添加文本， #因为在脚本层次上所有的值都表示为字符串，Tcl解释器会自动为变量管理内在，因此无需进行变量声明。仅在需要时才进行字符串表达形式与内部形式之间的转换，Tcl的性能因此#有了显著的提升。

set msg ""

foreach i {1 2 3 4 5} {

append msg "$i squared is [expr $i\*$i]\n"

}

incr a 12 ; # a += 12

关联数组：

set earnings(Jan) 87966

set earnings(Feb) 95400

多维数组：

set matrix(1,1) 140

set matrix(1,2) 218

array exists earnings ; #数组是否存在

array size earnings ; #返回数组的元素个数

array get earnings ; #返回数组内容字典

array names earnings ; #返回数组的元素名列表

array set earnings dictionary ; #将字典dictionary并入earnings中

编历

set yearTotal 0

foreach i [array names earnings] {

set yearTotal [expr $yearTotal+$earnings($i)]

}

set cell $matrix($i,$j) ;

Tcl库的预定义变量

puts "The command name is \"$argv0\""; #脚本文件名存放在变量argv0中

puts "There were $argc arguments: $argv"; #命令行参数个数argc，命令行参数以列表形式存放在变量argv中。

puts "Your home directory is $env(HOME) ; #env数组变量，其元素是所有过程的环境变量。

#tcl\_platform是一个数组变量，其元素是对应用程序正在运行的平台的描述

puts $tcl\_platform(platform) ;

puts $tcl\_platform(os) ;

puts $tcl\_platform(machine) ;

表达式：

expr sin(0.2) \* rand() ; 将它的参数作为表达式处理， 获得表达式的值并把这个值作为字符串返回

expr 2\*\*3 ; #2的3次方

expr pow(2,3) ; #2的3次方

hypot($x,$y) ; #(x,y)的平方根

只要可能，Tcl就把表达式作为数字处理，只有在使用字符串比较操作符时，或在使用关系操作符全至少一个操作数无法理解为字符时才进行字符串操作。Tcl将整数存储为C的long类型，实数存诸为C的double类型。如果一个操作符的各个操作数类型不同，Tcl会自动进行类型转换。

字符串操作：

Tcl脚本层次所有的值都是字符串，避免空格换行等，可以用双引号

set str "Sample string"

string index $str 3 ; #第四个字符p

string index $str end-1; #倒数第二个字符

string range $str 3 end ; #第四到结尾的所有字符

string length $str ; #字符个数

string toupper $str ;

string tolower $str ;

string trim aaxxaxaba abc ; # 删除开头和结尾出现的字符a,b,c，从而返回字符串xxax

string trimleft aaxxxaba abc ; #删除开头出现的字符a,b,c，从而返回字符串xxaxaba

string trimright aaxxxaba abc ;

string repeat "abc" 2 ; #返回字符串为abcabc

string first th "There is the tub" ; #返回子字符串th在长字符串中的最左边位置

string last th "There is the tub" ;

string compare Michigan Michigan ; #返回-1,0,1分别对应str1在字典上<,=,>str2.

string equal Michigan Michigan ; #相同为1,不相同为0

string replace "I am qzlin" 5 end tom ; #返回字符串置换,结果为"I am tom"

string is digit 1234; #确定字符串值类型是否正确

format "the result is %.3f" [expr sqrt(10)] ; #相当于C库的sprintf，同c库支持的所有转换符，很多时候使用printf and sprintf是为了把二进制数值转换为字符串，从而进行字符串替换操作。Tcl不需要这一转化，因为这些值已经存储为字符串形式了，字符串替换替换操作可以直接由Tcl解释器完成。在Tcl使用format，通常是为了改变一个值的格式，以改善显示效果。

列表：

列表允许把任意数量的值集合在一起，把集合作为一个实体传递，从集合中取得各成员的值。例如一个组内的所有用户、一个文件夹中的所有文件以及一个组件的所有选项。

set myList {John Anne Mary Jim} ;

lindex $myList 0 ; #列表索引

set elements {{a b} {c {d e f}} g} ;

lindex #elements {1 1 2} ; #通过多个索引值从子列表中取得元素f

list {a b c} {d e} f {g h i} ; #生成列表

concat $myList $elements ; #将几个列表中的所有元素串接为一个大列表

lrepeat 2 a b c ; #获得列表{a b c a b c }

lrange $myList 1 2 ; #返回列表某范围

linsert $myList 2 Tom ; #将元素插入到指定索引值对应的原列表中的元素之前

lreplace $myList 3 3 ; #将第四个元素删除

lreplace #myList 1 2 bob jack ; #将第2，3个元素替换

lset $myList 0 qzlin ; #修改元素值

lappend $myList dog cat ; #在列表结尾添加新元素

laaign {a b c} x y z ; #将列表中的值分发到一个或多个变量中。如果变量名比列表中的元素个数多，多余的变量会被设置为空字符串。

lsearch $myList Tom ; #在列表中查找指定的元素。

lsort $myList ; #按字典排序

lsort -integer {10 1 2} ; #整数排序

lsort -real {0.3 2.0 5.0} ; #实数排序

lsort -integer -index 1 {{First 24} {Second 18} {Third 30}} ; #-index选项允许指定子列表中元素的索引，根据指定的元素对子列表进行排序

split "tom, 35, vp marketing" , ; #将字符串按指定的分隔符分成几个部分于列表中。结果是tom 35 {vp marketing}

join {{} usr include sys types.h} / ; #将列表元素按指定的分隔符合并为字符串。结果是/usr/include/sys/types.h

字典：

有序的集合，字典会维护关键字的顺序，并且在需要时按顺序遍历它们。

字典和数组有一些根本性的不同，数组是变量的无序集合，而不是值的集合，而且不能嵌套

字典中的每个关键字都必须是独一无二的。关键字和关联值都可以是任意的值。

在Tcl脚本中使用字典时，它常常被大括号括起来，对于比较复杂的，可以使用换行把各个关键词-关联值对分隔开

set employees {

0001 {

firstname Joe

surname Schoe

title Mr

}

0002 {

firstname Ann

surname Huan

title Miss

}

}

dict get $employees 0002 firstname ; #嵌套的字典中获取一个值就变得更方便，可以像给出文件系统中的一条路径一样给出嵌套字典中的一条路径

set example {

firstname Ann

initial E

surname Huan

}

dict replace $example initial Y ; #将E替换为Y

dict set $example initial Y ; #将initial键值设为Y

dict remove $example initial ; #将initial E字典键值对去除

dict unset $example initial ; #将initial键值对去除

set colors1 {

foreground white

background black

}

set colors2 {

highlight red

foreground green

}

dict merge $colors1 $colors2; #融合多个字典

dict size $colors1 ; #确定字典元素的个数

dict exists $example firstname ;

dict keys $example ; #获取所有关键字的列表

dict values $example ; #获取所有值的列表

遍历：

dict for {key value} $colors1 {

puts [format "%s: %s" $key $value]

}

流程控制：

if {$x < 0} {

set x 0 ;

}

if命令获取两个参数，第一个参数是表达式，第二个参数是Tcl脚本

建议总是把表达式和脚本放在大括号中，这样直到命令执行前都不会有替换发生，而且每一个左大括号都必须在它的前一个单词的同一行，因为换行符就是命令分隔符。

switch $x {

a { incr t1 }

b { incr t2 }

c { incr t3 }

}

#倒转列表

#while

set b {}

set i [expr [llength $a] - 1]

while {$i >= 0} {

lappend b {lindex $a $i}

incr i -1

}

#for

set b {}

for {set i [expr [llength $a] - 1]} {$i >= 0} {incr i -1} {

lappend b [lindex $a $i]

}

#foreach

set b {}

foreach i $a {

set b [linsert $b 0 $i]

}

break and continue 同C

#eval 是用于创建和运行Tcl脚本的通用构造块，它接受任意多个参数，把它们用分隔符串接起来，然后把串接的结果作为一个Tcl脚本处理。所有的Tcl解析规则都正常应用于这个脚本，因此这个脚本可以饮食多个命令，展开为多行，包含注释等。

eval 的一个用途是生成命令，把它们存放在变量中，然后再把这个变量作为Tcl脚本运行

set reset { set a 0; set b 0; set c 0}

eval $reset

eval另一个用途是强制进行另一级的解析

set vars {a b c d}

eval unset $vars ; # unset a b c d

或

unset {\*}$vars

过程：

proc plus {a b} {

return [expr {$a + $b}] ;

}

过程名为plus，参数列表为{a b}, Tcl脚本在大括号内

在proc命令完成后，新的命令plus就存在了，可以像任何其他Tcl命令一样调用。plus命令的返回值就是它的过程块中最后一个命令的返回值或者return 返回值

proc fac {x} {

if { $x <= 1} {

return 1

}

return [expr $x\* [fac [expr $x-1]]]

}

在过程可以使用global命令引用全局变量

global x y

过程无参数列表，即参数列表为空

proc printVars {} {

global a b

puts "a is $a, b is $b"

}

过程参数列表默认值

proc inc {value {increment 1} {

return [expr $value + $increment] ;

}

过程可变参数列表目:如果参数列表中的最后一个元素是特殊名称args,那么调用过程中可以给出可变数量个参数，args会变设置成一个列表，其元素就是这些更多的参数，如果没有更多的参数，args会设置为空字符串

proc sum {args} {

set total 0

foreach val $args {

set total [expr $total+$val]

}

return $total

}

Tcl只支持参数的传值调用，Tcl不支持指针和引用类型

但可以使用upvar命令访问当前过程的上下文范围之外的变量的通用机制。可以用于访问全局变量、命令空间变量以及其他活动中的过程中的局部变量。它更常见的使用则是模拟传引用调用的行为，这对数组特别有用

如果arr是一个数组，就不能像myproc $arr这样把它传给myproc过程，因为并没有对应整个数组的值，只有对应各外数组元素的值，但是，可以把数组的名字传给过程，如myproc arr,然后使用upvar 命令在过程中访问数组的元素

proc printArray {\_arr} {

upvar $\_arr arr

foreach el [lsort [array names arr]] {

puts "$el = $arr($el)"

}

}

应用匿名过程：

Tcl过程不仅提供模块化编程，还可以提升性能，因为过程块在Tcl内部会同高效率的二进制代码表示，过程块还提供一个局部范围，但如果过程只被使用一次或有限几次，比如常见的例子是代码回调，如对组件命令和文件事件处理程序的回调，以及只用一次转换函数，那么可以应用匿名过程。

set states {California Delaware Hawaii Indiana Iowa}

lsort -command {

apply { {e1 e2} {

expr [string length $e1] - [string length $e2]

} }

} $states