KScript V1.0 使用手册

# 简介

KScript是使用C#为宿主语言开发的一款小型的动态类型脚本语言。其语法接近C系语言,其中很大一部分效仿了C#、Java、python,并具有一些创新的语法糖。

# 语法概述

**本节主要就KScript的变量定义、数据类型、各种运算符的使用、分支结构语句、循环结构语句、函数的定义和使用、部分原生函数的介绍、类的定义和使用、部分原生对象、内建类型的使用(包括String的高级操作)、外部代码文件/模块的使用、异常处理、匿名函数和lambda表达式、函数式编程及其相关库、反射特性、标准库介绍、拓展框架等 来让您熟悉并掌握KScript的使用方法和技巧。不用担心，这很简单，内容也不是很多。那么，开始吧。**

* **变量定义**

**KScript是一门动态语言，所以您无需为变量进行类型声明，变量可以在其生命周期中接受任何类型对象。请看下列语句:**

**var a = 1;**

**这样,就把1这个数字赋值给了变量a**

**当然，你可以省略关键字var:**

**a = 2;**

**你可以在一条语句中声明多个变量，可以同时绑定变量的值，对于未绑定值的变量(如下面的c)，其值为默认值null(表示一个空对象,将在后面介绍)**

**var b = a, c;**

**特别注意！以下的语句是不合法的:**

**a=1, b=2;**

**还有:**

**temp;**

**这句并不是用于声明temp变量,而是向程序的返回值缓存写入temp绑定的值。**

**还有一个值得注意的是,KScript的每条语句都需要以分号(;)结尾**

* **数据类型**

**KScript有一些内建的基本数据类型:**

**Num(数字类型)、Str(字符串类型)、Arr(数组类型)、Type(对象类别类型,用于静态数据的访问以及反射特性的实现)、**

**NameSpace(命名空间类型,用于模块的管理,不同的模块拥有间相互独立的一系列变量)**

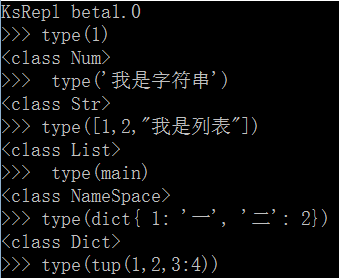
**内建数据结构(具体内容将在之后介绍):**

**List(列表类型)、Dict(字典类型)、Set(集合类型)、Tuple(元组类型)**

**·需要注意的是,Ks没有单独的布尔类型,而是用0表示false,非0(包括负数、小数)表示true。当然，你可以使用布尔常量(只是变量形式上像布尔型):true(1),false(0)**

**你可以使用原生函数type(obj)来查看一个对象的类型:**

**(以下为dos控制台下运行的解释器程序KsRepl。之后我们将使用专用的交互式解释器ksPerl和专用开发环境KsIDE编写、运行相关程序)**



* **运算符**

**运算符是一门语言的基础,Ks自然拥有丰富的运算符,并支持运算符重载,可以说非常灵活。**

**以下介绍部分运算符:**

**注:**

**·结合性指一个二目表达式运算时先运算哪一部分的子表达式,例如’+’是左结合的,所以1\*2+3\*4 =>(1\*2) + 3\*4; ’+=’是右结合的,所以 a+=6-7 => a += (6-7)**

**·优先级是指示不同运算符运算顺序的一组标量。比如’\*’比’+’优先级高,故**

**5/6+5\*9.2先计算5/6和5\*9.2,再进行加法运算**

**·短路性值一个布尔表达式会进行惰性求值,当真值可由第一个部分表达式确定时,后面的表达式将不予计算。例如true || read() => 1,并且不会再进行数据的读入; false && read() => 0,同上。**

**·对于位运算,会自动舍弃操作数的小数部分**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 运算符 | 名称 | 属性 | 说明 | 举例 | 优先级 |
| + | 加 | 左结合  可重载 | 相加或字符串连接操作 | 1+1 //2  “1”+”2a” // “12a” |  |
| - | 减 | 左结合  可重载 | 相减 | -2.25 – 6 // -8.25 |  |
| \* | 乘 | 左结合  可重载 | 相乘、运算符重复、拉锁(Zip)运算 | 2.5\*6 // 15  “a”\*3// “aaa”  [1,2,3]\*[4,5,6]  //[(1,4)(2,5)(3,6)]  3 \* [2,3] //[6,9] |  |
| / | 除(以) | 左结合  可重载 | 相除 | 5/4 //1.25 |  |
| \*\* | 双乘 | 左结合  可重载 | 乘方 | 2\*\*10 //1024 |  |
| % | 模运算 | 左结合 | 除法取余 | 6 % 5 //1 |  |
| & | 按位与 |  |  |  |  |
| | | 按位或 |  |  |  |  |
| ^ | 按位异或 |  |  |  |  |
| >> | 右移位 |  |  |  |  |
| << | 左移位 |  |  |  |  |
| || | 逻辑或 |  |  |  |  |
| && | 逻辑与 |  |  |  |  |
| (以上任意符号)= | 复合运算 | 右结合 | 先计算,后赋值 | a = 1;  a+=3;//a==4 |  |
| == | 相等比较 | 左结合 | 比较两个对象是否相等,返回真值(若为非值类型对象,则只有统一地址上的对象才相等) | 1.25==1.25//1  ‘123’ == ‘123’//1  a = Class.new;  b = Class.new;  a==b //0 |  |
| != | 不等比较 | 左结合 | 比较两个对象是否不相等 | ‘123’ != ‘123’//0 |  |
| < | 小于 | 左结合 | 比较左对象是否小于右对象 |  |  |
| > | 大于 | 左结合 | 比较左对象是否大于右对象 |  |  |
| <= | 小于等于 | 左结合 | 比较左对象是否小于或等于右对象 |  |  |
| >= | 大于等于 | 左结合 | 比较左对象是否大于或等于右对象 |  |  |
| : | 成对 | 左结合 | 由左右两边的表达式组成一个二元的元组。若子表达式为元组,则会拆解放入新元组中 | 2:3 // (2,3)  (1,2):3:4 //(1,2,3,4) |  |
| in | 内含 |  | 检查一个对象是否包含在右边的可迭代型对象中；检查一个数是否在右侧的区间对象中 | ‘12’ in ‘123’//1  in #(1,3]//1  ‘q’ in [1,2,’q’]//1  ‘s’ in set{1,’s’}//1 |  |
| is | 类同 |  | 检查左侧对象的类型是否与右侧的类型对象相同或为此类型的子类 | main is Namespace//1  ‘as’+23 is Str//1  dog is Dog //1  dog is Animal //1 |  |
| ! | 非 | 单目 |  |  |  |
| ~ | 按位取反 | 单目 |  | ~1 // -2 |  |
| & | 取地址 | 单目 |  | &a // 1072693248  &’123a’ == &’123a’//1 |  |
| @ | 绝对值 | 单目 |  | @(1-2)//1 |  |
| - | 相反数 | 单目 |  | -5 // -5  @-5 //5 |  |
| ?: | 条件运算符 | 三目运算符  Non短路性(待修复) | 检查?左边的逻辑表达式的真值。若为真则返回：左边结果,否则返回右边结果。这是一个复合的运算符,’?’右侧的运算结果实质上为一个二元组 | 1==1.0?’123’+’a’:’12’  //123a |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

* **分支结构:**

**分支结构是结构化代码的基础,有了分支结构,我们可以定义多个逻辑块,**

**根据条件执行不同的逻辑。**

**分支结构有以下两种:条件语句、模式匹配语句。**

1. **条件语句**

**格式:**

**If(逻辑表达式) { //如果**

**代码**

**}**

**else if(逻辑表达式) { //或如果**

**…**

**}**

**(支持零个或多个else if分支)**

**other { //否则(上面所有条件都不匹配时)**

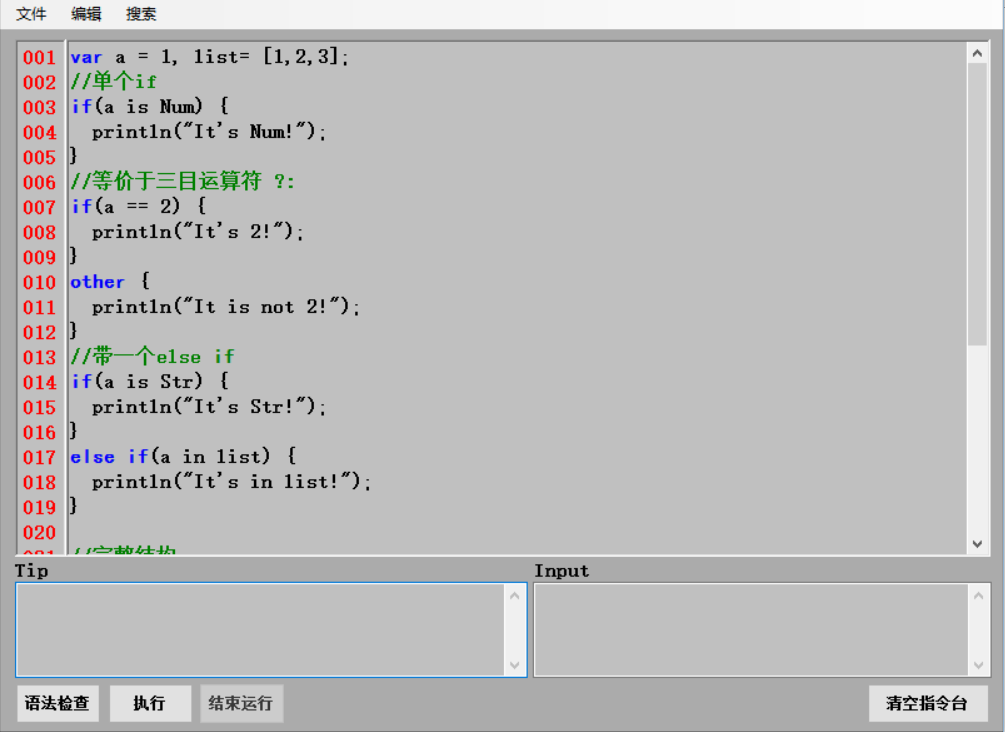
**…**

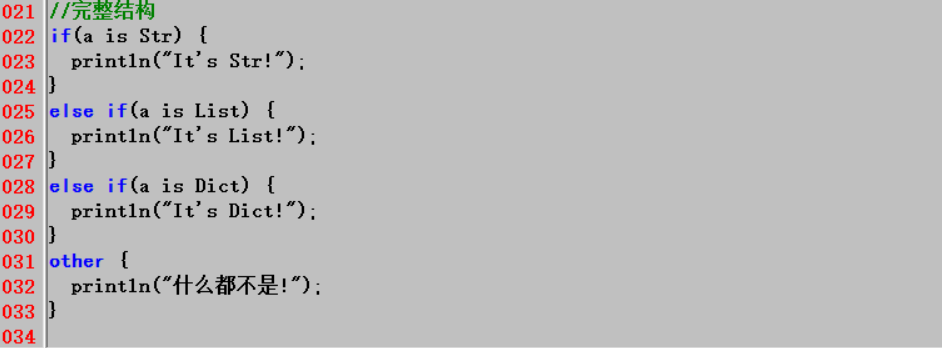
**}**

**(支持零个或一个other语句)**

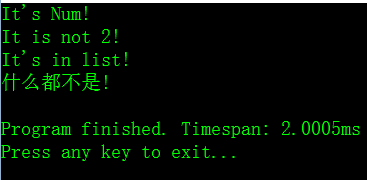
**注意,条件语句同样也是短路性的,即只有上一个条件不符合时,才会执行下面的判断。**

**例程：**





**运行结果:**

****

1. **模式匹配语句**

**格式:**

**match 对象、表达式**

**when 对象 {…}**

**when …**

**default {…}**

**注意,ks模式匹配可能与其他语言的switch语句有所不同,它不是向下跌落的,只能匹配一个条件,但不需要像其他语言那样用break结束标签跌落。(今后会进行研究，作出调整)**

**例程:**