# 字符串

《java编程思想》字符串一章的第一句话：“**可以证明，字符串是计算机程序设计中最常见的行为。**”

符号的有序组合形成表达式，每个表达式表示一个对象或认识。而表达式本身正是字符串的子集。字符串表达式是特殊的表达式

## 字符串计算

### 字符串长度

string.length

注意:与java不同，length没有参数，也没有()

使用方法：

ask(“abc”.length) //=3

### 字符串相加

string1+string2

如："abc"+"def" //="abcdef"

在会话状态：

ask("abc"+"def") //="abcdef"

### 子字符串

substring(start,end)

与java等取子字符串函数不同，hiwii取子字符串是包括end字符串。而在java中，substring函数是不包括位置end的字符的。

比如，java中：

String a = "abcde".substring(0,2) //结果是"ab"

而hwii中

String a = "abcde".substring(0,2) //结果是"abc"

做这样改动的目的是：java中如果要取start直到字符串结束的substring，end取值从逻辑上已经超出了字符串长度，如：

String tail = "abcde".substring(2,5) //结果是"cde"。但最后一个字符'e'的位置是4。由于java的substring不包括end字符，因此必须让end=5才能返回要求的结果。

允许end=5并没有什么，但是java中，如果要取一个char，这样的位置就会报错。

char tail = "abcde".charAt(5) //系统抛出异常

### 截取子串

格式：cut(position,len,Boolean forward)

表示向前或向后获得一个子串，截取长度为len。

forward=true向前，false向后。forward参数可以省略，默认是true。

这个和substring的不同在于，substring的参数是开始和结束位置，而cut参数是开始位置和截取长度，而且cut可以选择向前或向后截取。

### 取字符

格式：string.charAt(pos)

pos必须是大于等于0的整数。

使用方法:

ask(“abc”.charAt(0)) //='a'

### to表达式

string.toExpression

如：

“3".toExpression //结果是integer 3

"3" + "4" //结果是"34"

"3".toExpression + "4".toExpression

"3+4".toExpression //结果是 3 + 4,不是7，因为没有进行计算

lambda(,"3+4".toExpression).run()

## 字符串比较

字符串支持六种比较判断(=,!=,>,<,>=,<=)

## 字符串操作

java中认为，字符串是不可变的。

### reverse

string.reverse

# 正则表达式

正则表达式与java正则表达式有较大差别。

传统正则表达式来源于perl，是一串特别定义的字符串组成。如果要善于使用这些reg，必须牢牢记住很多特殊字符的使用，比如常见的'?','+','\*'都是这些特殊字符。

按照hiwii语言原则，除了分隔符外，所有计算操作符必然与一个函数一一对应。因此，一组有效好记的函数应该是正则表达式的基础，而不是超过一页的符号使用说明和规则。

传统正则表达式的匹配原则一般是默认是否匹配给定的整个字符串，即是否匹配是和字符串相关的，或者有些书认为的，匹配是从右向左匹配的。

而hiwii语言中，正则表达式是一组字符串集合的共同特征，一个字符串是否与一个正则表达式匹配的含义是：一个字符串是否属于这个正则表达式表示的字符串集合。

## 特殊正则表达式

包括：anyString/operator/line/character/word/digit/number/

hiwii语言使用的表达式也是一组特殊正则表达式，包括：identifier/function/ParenExpression/BracketExpression/Block/subjectOperation/subjectVerb/subjectPositive/subjectNegative都是特殊正则表达式。

## 简单正则表达式

reg(String)

简单正则表达式表示一个精确匹配。

## 选择正则表达式

alternate(reg1，reg2...)

表示可选项只要有一项匹配，则

## 重复正则表达式

重复正则表达式又分为以下几种：

### 恰好n次重复

repeatNTimes(reg,n)

### 贪婪n次重复

repeatNGreedy(reg,n)

### 懒惰n次重复

repeatNReluctant(reg,n)

### 贪婪区间重复

repeatRangeGreedy(reg,min,max)

String(abababab) match repeat("ab",2,3)

满足部分是String(ababab)

表示从n1到n2(n1<=n2)重复次数。如果n1省略，默认值为0。如果n2省略，默认值为无穷大。

## 复合正则表达式

compound(reg1,reg2……)

## 否定正则表达式

regNot(

简单、选择、重复、复合、嵌套正则表达式可以返回一个字符串的匹配部分，而否定正则表达式不能返回匹配部分。

## 字符串和正则表达式

### guide

boolean string$guide(reg,int position，boolean forward)

forward默认为true。表示能否向前或向后获得匹配

//返回字符串开始直到匹配结束，如果greedy则匹配直到不能匹配为止，reluctant表达式匹配一旦符合，则返回匹配结束地址。

### count计算

int string.count(reg,int position,Boolean forward)

forward默认为true。表示能否向前或向后获得匹配.

返回向前或向后匹配子串的长度，如果没有获得匹配，则返回值为-1。

### matches

boolean string$matches(reg)

表示字符串是否匹配正则表达式

### locate

int n = string.locate(pos,reg,forward)

从position位置开始向后匹配。

# 程序执行

计算、判断和操作往往不是一个动作能够完成的，一般需要顺序的执行多个动作才能完成一次计算、判断和操作。这种顺序执行的过程就称作程序执行。

block的顺序执行即传统意义上的程序执行。程序分为操作、计算和判断。操作的执行可以由计算和判断支持，但计算和判断的执行不能调用任何操作。

## 分支

系统定义了三种基本分支表达式：

分支表达式可以单独使用，也可以作为操作、判断和计算表达式使用。单独使用即作为block中的一个表达式单独出现，如：

if(3>2,break)

作为操作表达式：

对象#分支表达式；如：车#if(状态正常,启动)

作为计算表达式：

对象.分支表达式；如：车.if(张三健康, 张三, 李四)//选择车的司机

作为计算表达式：

对象$分支表达式；如：车$if(没有故障灯亮,正常,异常)

对象!分支表达式；

### if分支

if(condition,expression[,expression])

if分支表达式由2个或三个参数构成，第一个参数用于表示条件，第二个表示表示执行表达式，第三个表达式是可选的，表示else表达式，即条件不满足的情况下执行的表达式。

比如：if(3>2, command1, commandElse)

### choose分支

choose(case(cond1):expression1,case(cond2):expression2....else:expression)

这是一个多分支结构，同样以函数形式为表达式。

每个case表达式中的参数表示判断条件，其后表示执行表示。

判断按顺序依次进行，如果满足条件，则执行对应的表达式，然后退出。

如：

choose(case(i>3):echo("abcd"),

case(i=3):echo("abc"),

else:echo("ab")

)

### switch分支

switch(operation, case(result1):expression, case(result1):expression....else:expression)

与choose一样表示多分支结构，与choose不同的是，第一个参数表示一个计算表达式，case中可以有多个表达式。执行与choose分支一样，遇到满足条件的case语句则执行，然后退出分支执行。如：

switch(i,

case(0,1,2):echo("ab"),

case(3):echo("abc"),

else:echo("abcd")

)

## 循环

### while循环

while(condition,Expression);

### 容器迭代

each(i:collection,Expression)

collection必须是List或有限集合(Set)

### for循环

for(var1=initV,condition,iteration，expression)

初始表达式：空|赋值表达式|[多个赋值表达式]

迭代执行部分只运行修改初始定义的变量。

这个循环语句继承了java、C/C++的特点，但增加了一些限制。

可以使用空条件循环：

for(,,,statement);//这条语句表示不断循环，直到遇到break语句退出。

它与下面语句相同。

while(true,statement)

hiwii程序执行体不能执行循环变量的修改操作，循环变量的操作只能在iteration部分完成,而且循环变量的操作部分不能修改外部变量。

下面是一个java的例子。

for(int i=0;i<10;){

i++;//内部修改变量值在java中是允许的。而hiwii不允许

}

另一个java例子：

int a=0;

for(;a<10;a++){

//迭代执行部分运行修改外部迭代变量，java中允许。hiwii不允许

}

## 程序控制语句

### break语句

与java语言类似，用于中断循环。只能用于循环体中，否则系统会报错。

如：

{

break; //系统报错，break没有处于循环体中

}

### skip语句

skip语句表示结束当前模块的执行。

在java、c等大多数语言中，并没有结束模块执行的语句。在java、c中，有continue语句，而continue语句只用于循环中，表示结束当前循环进入下一循环。然而java、c既然提供了模块执行的方式，却没有提供提前退出模块的方法，而必须完全执行模块中的每一条指令，这会给用户带来一定的迷惑。

现在有了skip语句，那么当在循环中执行模块时，skip就可以完全取代continue语句。在循环中执行模块，表示循环建立模块执行环境，并在本次循环结束后退出模块执行环境。

skip语句必须在模块中执行，如果直接在会话中执行，系统会报错，如下：

>skip //系统返回错误

> {ask(1);skip;ask(2);} //只执行skip之前的指令

1

### return

用于Action程序中，表示结束action。

### return(expression)

这个语句和return语句的使用完全不同。return语句用于action Block中，该语句用于计算中，表示结束计算并返回计算结果。

### decide(expression)

参数是判断表达式，判定返回yes或no。

### throw(Exception)

表示结束当前程序，并抛出异常。