Ve编程语言语法介绍说明

关于Ve语言

Ve:Very Easy的简称 Ve是一个轻量级的数据处理编程语言。 把Ve语言看成是一个放大版的数学公式吧， Ve除了处理数字之外还可以处理逻辑，文本、集合等常见数据

基本类型

我们提供了最简单的数据单元：数字，整数，文本，是否

数字、整数

并没有提供float,double等数字类型。
如果当前的类型为数字类型，那么最终生成的其它的语言都会对应浮点型。 Ve支持科学计数法。

```
def a:int=300;
def b=3e2;
print(a==b);//true;
def c:number=100.2;
```

文本

使用双引号、单引号包起来的文本串
支持文本内部跨行显示
支持模板文本

```
def a:string='eeeee';
def b:string='eee
eee';
def c:string='两个@{a},@{b}';
def d='两个'+a+','+b;
print(d==c) //true
```

是否

表达逻辑真与假

```
def a:bool=true;
def b=false;
```

提供了处理复杂数据结的数据类型：对象、数组、列表

对象

对象类型类似于JSON

```
def data={ account:'kankan',pwd:'12345',sex:true};
def data:{ account:string,pwd:string,sex:bool}=={
account:'kankan',pwd:'12345',sex:true};
def data={
    account:string='kankan',
    pwd:string,
    sex:bool
}
print(data.pwd)//null
print(data.pwd)//null
print(data.sex)//null
```

数组

数组里面的数据类型必须一致

```
def a:int[]=[1,2,3];
def a:string[]=['1','2','3'];
def a:int[]=[1,2,'string'];//error 数组里面的每一项都应该是数字类型
def a:any[]=[1,2,'eee'];//ok ,数组里面的每一项都是any类型
def a=[1,2,3]//自动推断为int[]
def a=[1,2,'eee']//自动推为any[];
def a:{a:string,b:number}[]=[];//定义一个数组，里面的每一项都是{a:string,b:number}类
型
def a=int[1,2,3,4];
```

Any

any类型表示任意类型，也可以理解为无类型，所有的数据类型都继承该类型Any

```
def a:any=123;
def b:any=true;
```

枚举

枚举默认情况下，从0开始为元素编号。 你也可以手动的指定成员的数值。
被指定的元素值不参于元素编号。

```
enum Color {Red, Green, Blue}
def c:Color=Color.Red;
print(c);//0
```

```
print(c.toString()):://Red
//当然也可以指定值
enum Color {Red=2, Green=0, Blue=3}
print(Color.Red)//Red
print((int)Color.Red)// 2
print(Color.Red.toString());// Red
```

fun

fun类型

```
fun sum(a:number,b:number):number{
    return a+b;
}
fun action()
{
    //do something...
}
fun action(...args:string[]):void
{

}
fun action(a:string='eee',b:number='eesss')
{
   //如果返回值为void,可省略
}
```

null

```
def a=null;
print(a):// null
print(a.toString())//error...
print(a?.toString())//null
print(a is null)//true
a?=1;//如果a为空，则赋值为1
```

基于文本的类型

类型	名称	说明
日期	date	
正则	regex	
颜色	color	
网址	url	

类型	名称	说明
路径	path	
文件	file	
文件夹	folder	
图像	image	
字体	font	

```
def c=/2018-9-25/date;
//注意此时的date需要紧挨着
def url=/http://www.baidu.com/url;
def color=/#fff/color;
```

变量声明

def 声明

```
def a=10;
def a=20,b=30,c:color=/#fff/color;
```

const声明

const 声明的值在第一次赋值后，不可再重新赋值修改

```
const a=20;
a=30 //error
```

接口

```
interface LabelledValue {
  label: string;
}
```

类

```
class Greeter {
    greeting: string;
    new(message: string)
    {
```

```
        this.greeting = message;
    }
    greet():string
    {
        return "Hello, " + this.greeting;
    }
}
```

箭头函数

```
def add:(a:number,b:number)=>number=(a,b)=>a+b;
print add(1,2);//3
```

泛型

```
fun identity<T>(arg: T):T
{
    return arg;
}
//下面的泛型只能接收string
fun identity<T:string>(arg:T):T
{
    return arg;
}
```

包 引用

```
package nc{
    class classA
    {

    }
}
// 其它文件代码文件引用
use nc;
use nc.class1 classOne;
def a=new class1();
def b=new classOne();

//其它代码引用
use nc;
use nc.web;
use nc.web.class2 classSecond;
def a=new web.class2();
def a=new class2();
def a=new classSecond();
```

关键字

运算符	描述
def	
const	
if,else if,else	
while,do...while	
for,for in	
switch,switch case,default	
fun	
class,enum,interface,extend	
public,private,static,sealed	
package	
use	
get,set	
new,is,as,not,contain,start,end,match	

运算符

运算符是一种告诉编译器执行特定的数学或逻辑操作的符号。Ve 有丰富的内置运算符，分类如下：

- 算术运算符
- 关系运算符
- 逻辑运算符
- 赋值运算符
- 其他运算符

算术运算符

运算符	描述	实例
+	把两个操作数相加	A + B 将得到 30
-	从第一个操作数中减去第二个操作数	A - B 将得到 -10
*	把两个操作数相乘	A * B 将得到 200
/	分子除以分母	B / A 将得到 2
%	取模运算符，整除后的余数	B % A 将得到 0
++	自增运算符，整数值增加 1	A++ 将得到 11

运算符	描述	实例
--	自减运算符，整数值减少 1	A-- 将得到 9

关系运算符

运算符	描述	实例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(A == B) 不为真。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(A != B) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(A > B) 不为真。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(A < B) 为真。
>=	检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。	(A >= B) 不为真。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。	(A <= B) 为真。

逻辑运算符

运算符	描述	实例
&&	称为逻辑与运算符。如果两个操作数都非零，则条件为真。	(A && B) 为假。
||	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(A || B) 为真。
!	称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	!(A && B) 为真
^	异或，如果同时为真，则为假，否则为真	A^B

赋值运算符

运算符	描述	实例
=	简单的赋值运算符，把右边操作数的值赋给左边操作数	C = A + B 将把 A + B 的值赋给 C
+=	加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数	C += A 相当于 C = C + A
-=	减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数	C -= A 相当于 C = C - A
*=	乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数	C *= A 相当于 C = C * A
/=	除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数	C /= A 相当于 C = C / A

运算符	描述	实例
%=	求模且赋值运算符，求两个操作数的模赋值给左边操作数	C %= A 相当于 C = C % A
?=	null赋值	a?='123' 如果a为null,则a 当前取会为'123'

其他运算符

运算符	描述	实例
??	null条件	def b=a??'123';如果a为null,则b='123',否则b=a
?:	条件表达式	如果条件为真 ? 则为 X : 否则为 Y
is	判断对象是否为某一类型。	If(Ford is Car) // 检查 Ford 是否是 Car 类的一个对象。
as	强制转换，即使转换失败也不会抛出异常。	
match	文本格式判断	判断当前文本是否满格式 bool isMath=c match new Regex('[\d]+')
contain	是否包含	def a= '123' contain '1' def a=int[]{1,2,3} contain 1

Ve中的运算符优先级

类别	运算符	结合性
后缀	() [] -> . ++ - -	从左到右
一元	+ - ! ++ - - (type)	从右到左
乘除	* / %	从左到右
加减	+ -	从左到右
关系	< <= > >=	从左到右
相等	==, !=, is, as ,match contain,not is,not match,not contain	从左到右
逻辑与 AND	&&	从左到右
逻辑或 OR	||	从左到右
条件	?: ??	从右到左
赋值	= ?= += -= *= /= %	从右到左
逗号	,	从左到右

语句

名称	语法
for	`/for\((\{|((?!(statement)).)*(statement)?)/`
if	`/if\((\{|((?!(statement)).)*(statement)?)(eif\((\{|((?!(statement)).)*(statement)?))*(else(\{|((?!(statement)).)*(statement)?))?/`
switch	`/switch\(\{ /`
case	`/case((?!(case|default)).)* /`
return	`/return(((?!(statement)).)*(statement)?)/`
default	`/default((?!(case|default)).)*/`
do	`/do(\{|((?!(statement)).)*(statement)?)while\(/`
while	`/while\((\{|((?!(statement)).)*(statement)?)/`
fun	`/funword\(\(?\{/`
break	`/break/`
continue	`/continue/`

封装

public：所有对象都可以访问；如果对象没有任何修饰，则默认是public
private：对象本身在对象内部可以访问；
static :所有对象共用

Ve函数

Ve中定义函数 当定义一个方法时，从根本上说是在声明它的结构的元素。在 Ve中，定义方法的语法如下：

(Parameter List) { Method Body } 下面是方法的各个元素：
Access Specifier：访问修饰符，这个决定了变量或方法对于另一个类的可见性。
Return type：返回类型，一个方法可以返回一个值。返回类型是方法返回的值的数据类型。如果方法不返回任何值，则返回类型为 void。返回类型为void时，可省略
Method name：方法名称，是一个唯一的标识符，且是大小写敏感的。它不能与类中声明的其他标识符相同。
Parameter list：参数列表，使用圆括号括起来，该参数是用来传递和接收方法的数据。参数列表是指方法的参数类型、顺序和数量。参数是可选的，也就是说，一个方法可能不包含参数。
Method body：方法主体，包含了完成任务所需的指令集。
实例

```
class NumberManipulator
{
   public FindMax( num1:int,num2:int):int
   {
      /* 局部变量声明 */
      def result:int;
      if (num1 > num2)
         result = num1;
```

```
        else
           result = num2;

        return result;
    }
    ...
}
```

继承与多态

基类和派生类
一个类可以派生自多个类或接口，这意味着它可以从多个基类或接口继承数据和函数。 Ve 中创建派生类的语法如下：

```
<acess-specifier> class <base_class>
{
  ...
}
class <derived_class> : <base_class>
{
 ...
}
```

多态

```
  class Printdata
   {
      print(int i)
      {
         print("Printing int: @{i}" );
      }
      print(string s)
      {
         print("Printing string: @{s}");
      }
   }
```

Ve 异常处理

假设一个块将出现异常，一个方法使用 try 和 catch 关键字捕获异常。try/catch 块内的代码为受保护的代码，使用 try/catch 语法如下所示：

```
try
{
   // 引起异常的语句
}
```

```
catch(ExceptionName e1)
{
   // 错误处理代码
}
finally
{
   // 要执行的语句
}
```

Ve 中的异常类

异常类	描述
Ve.IndexOutOfRangeException	处理当方法指向超出范围的数组索引时生成的错误。
Ve.ArrayTypeMismatchException	处理当数组类型不匹配时生成的错误。
Ve.NullReferenceException	处理当依从一个空对象时生成的错误。
Ve.DivideByZeroException	处理当除以零时生成的错误。
Ve.InvalidCastException	处理在类型转换期间生成的错误。

Ve 特性（Attribute）

Ve特性

```
class String{
    [Translate('string.replace")]
    replace(old:string,newStr:string)(string);
}
//调用说明
def a='字符替换说明';
def newStr='eeee';
string result;
print(result)//'字符替换eeee'
```

属性、索引器、委托、匿名方法

访问器（Accessors） 属性（Property）的访问器（accessor）包含有助于获取（读取或计算）或设置（写入）属性的可执行语句。访问器（accessor）声明可包含一个 get 访问器、一个 set 访问器，或者同时包含二者。例如：

```
//声明类型为 string 的 Code 属性
private code:string;
Code:string
{
   get
   {
```

```
        return code;
    }
    set
    {
        code = value;
    }
}
```

函数、委托、匿名函数

```
class classA {
    //申明了委托，委托可以看定义了一函数类型但没有实现过程
    //相当于定义了类型，但没赋值
     ab:(a:number,b:number)=>number;
     task(fx:(a:number,b:number)=>number):
     {
         return fx(10,300);
     }
}
fun abc(a:int,b:numbr):number
{
    return a+b;
}
fun abcd(a:int,b:number):number
{
    return a-b;
}
def a=new classA;
a.ab=abc;
print(a.ab(10,200));//210
a.ab=abcd;
print(a.ab(10,200))//-190
print(a.task(abc))//310
print(a.task(abcd))//-290
print(a.task((x,y)=>x*y))//3100
```