Simple 语言定义

草案

2009 年 6 月

作者: Herbert Czymontek
翻译: Janeky(阿健)
Email:rojaneky@gmail.com

注意：此翻译版本可能不是最新的。详细信息参阅官方网站或发邮件查询

语法

Simple 语言采用“扩展巴克斯格式”（Extended Backus-Naur Form –EBNF）.以下一些标志将会被用到：

- ｜－用于分开一个个可供选择的对象，即“或”。
- （）－定义一个组合
- ［］－指定一个选择（ 0 或者 1 次）从里面定义的对象中选择 0 个或者 1 个
- ｛｝－指定重复对象（ 0 或者 n 次）

词法标志都将用黑体和蓝色突出显示，例如，数字 1 ： 1 。

词法结构

源文件是纯文本文件。
缺省的字符编码是　ＵＴＦ－８。
跟其他ＢＡＳＩＣ编程语言不同，Ｓｉｍｐｌｅ语言是区分大小写的。

行尾结束字符

Simple 语言由“行”（line）构成。所有的语法都由“行结束字符”区分开。以下的“行尾结束字符”能够被识别：

- Unicode 字符　＼u000D （ＣＲ－回车)。
- Unicode 字符　＼u000A (ＬＦ－换行)。
- Unicode 字符　＼u000D （ＣＲ－回车) 后面跟着＼u000A (ＬＦ－换行)
  （-译者注：Windows 环境）

EndOfLine := (CR | LF | CR LF)

可以用一个“行连续符”（Unicode \u005F (_下划线)）放在行结束字符前面取消换行功能。（注意，合法标识符的第一位不能是“_”）

空白符

以下的字符被称为空白符，用以区分一个个 token：

- Unicode \u0009 (TAB-tab 制表符)
- Unicode \u000B (VT –纵向制表符)

- Unicode \u000C (FF-换页符号)
- Unicode \u0020 (SP-空白键符)

Whitespace := (TAB | VT | FF | SP)

Tokens

从源文件读入的字符串被分成一系列的 tokens。Tokens 是语法的终结符。编译器总是先尝试寻找可能存在的最长字符串来定义一个 token。例如，输入字符串“a<<b”，将被分析成标识符 a，左移运算符<<和标识符 b。而字符串“a< <b”将被解析成标识符 a,小于号<,小于号<,和标识符 b

注释

注释等同于空白符。Simple 语言目前只是支持单行注释。注释以单引号 ’(Unicode \u0027)开始，以行尾结束符结束。注释不能以一个字符串常量开始。

Comment := ‘{任何的 Unicode 字符除了行尾结束符}EndOfLine

关键词

以下的字符系列都是关键词，不能用于定义标识符

Keywords := Alias | And | As | Boolean | ByRef | Byte | ByVal | Case | Const | Date | Dim | Double | Each | Else | ElseIf | End | Error | Event | Exit | For | Function | Get | If | In | Integer | Is | IsNot | Like | Long | Me | Static | Step | String | Sub | Then | To | TypeOf | Until | Variant | While | Xor

标识符

一个标识符是以 一个符合 Java 标准的字母符号（letter character），加上若干个符合 Java 标准的字母(letters)或者符合 Java 标准的数字符号(Java digit characters) 或者下划线 _ 。Java 字母就是 经过 java 方法 java.lang.Character.JavaLetter()计算能返回 true 的字符。而 Java 数字字符就是经过 java 方法 java.lang.Character.JavaDigit()计算能返回 true 的字符。

一个合法的标识符不能是关键词。

Identifier := JavaLetter { JavaLetter | JavaDigit | _}

Literals(常量，变量，字面值，类型？)

Literal 是指一种固定类型的值。Simple 语言有 interger(整型)，floating point(浮点型),Boolean（布尔型），string（字符型）和 object（对象型）

Literal := IntegerLiteral | FloatingPointLiteral | BooleanLiteral | StringLiteral | ObjectLiteral

整型（Integer Literals）

整型适用于 Inteter 类型或者 Long 类型精度计算的数值常量。可以用十进制或者十六进制来表示整型。整型通常是无符号的正数，如果需要表示负数，需要在前面增加负号“—”运算符。

IntegerLiteral := DecimalIntegerLiteral | HexIntegerLiteral
DecimalIntegerLiteral := **0** | NonZeroDigit {Digit}
NonZeroDigit := **1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
Digit := **0** | NonZeroDigit
HexIntegerLiteral := **&H** HexDigit{HexDigit}
HexDigit := Digit | **A | B | C | D | E | F**

浮点型（Floating Point Literals）

浮点型适用于 Single 类型或者 Double 类型精度计算的数值常量。浮点型通常是无符号的。如果要表示负数，需要加负号运算符“—”前缀

FloatingPointLiteral := DecimalIntegerLiteral . Digit { Digit }[Exponent]
Exponent := **E** [**+** | **-**] DecimalIntegerLiteral

布尔类型（Boolean Literals）

布尔类型是指 Boolean 修饰的，包括两个值 true 或者 false。

BooleanLiteral := **True | False**

字符串类型(String Literals)

字符串类型是指用双引号 **“**前后修饰的 Unicode 字符串。一个字符串类型就是用 String 修饰的量。如果我们要在一个字符串类型里面用到双引号本身，需要在双引号前面加反斜杠 “\”。同理，要表示反斜杠本身也需要加反斜杠。还有换行符（**\n**）,回车键（**\r**），制表符(**\t**) 和换页符（**\f** ）也是同样道理。

StringLiteral := **“** { StringCharacter } **”**
StringCharacter := InputCharacter except **“** and \ | EscapedCharacter
InputCharacter := any 16-bit Unicode character
EscapedCharacter := \\ | **\”| \n | \r | \t | \f**

对象类型（Object Literal）

如果一个 Object 类型对象不表示任何 object，可以用 **Nothing** 修饰

ObjectLiteral := Nothing

分隔符（Separators）

分隔符包括以下：

Separator := **(** | **)** | **,** | **:**

运算符（Operators）

Simple 语言运算符包括以下：

Operator := << | < | <= | = | <> | => | > | >> | & | + | - | \ | * | / | ^ | .

类型和值 （Types and Values）

基本类型

Type := NonArrayType | ArrayType
NonArrayType := **Boolean** | **Byte** | **Short** | **Integer** | **Long** | **Single** | **Double** | **String** | **Date** | **Variant** | ObjectType | ArrayType

Boolean 类型

Boolean 类型用来表示两个逻辑值： true 和 false。Simple 语言中定义为 **True** 和 **False**

Byte 类型

Byte 类型表示 8 位的有符号整数值，在 -128 到 127 之间

Short 类型

Short 类型表示 16 位的有符号整数值，在-32768 到 32767 之间

Integer 类型

Integer 类型表示 32 位的有符号整数值，在-2147483648 到 2147483647 之间

Long 类型

Long 类型表示 64 位的有符号整数值，从-9223372036854775808 到 9223372036854775808

Single 类型

Single 类型代表 32 位的单精度浮点数，遵循 IEEE 754 标准

Double 类型

Double 类型代表 64 位的双精度浮点数，同样遵循 IEEE 754 标准

String 类型

String 类型表示一系列的字符串（可能是空）

Date 类型

Date 类型跟 Java 中的 java.util.Calendar 类相匹配

Variant 类型

Variant 类型是一种分解的集合（disjoint union），可以表示任何顺序的其他数据类型（基本类型、数组，对象类型等）

Array 类型

ArrayType := NonArrayType **(** [Expression { **,** Expression }] | { **,** } **)**
Array 类型表示相同类型元素的集合，可以通过下标来访问。一个 Array 可以有一维或者多维（最多 256）

Array 可以先确定每维的元素个数，或者不确定（动态决定 size）

Object 类型

Object := **Object** | Identifier

Object 类型的值是其引用的 object 实例，或者表示没有任何实例，**Nothing**。继承（Inheritance）被认为是设计阶段（design time）的性质，意味着没有外在的语法来确定 object 的关系。除此之外的信息存储在源文件的一个特定区域（更多关于源文件格式的信息见附录）。Simple 支持接口的继承和实现。一个接口 Interface 对象只能定义常量和没有函数体的函数声明。

Object（接口除外）有两个预先定义的事件 event

- 当 Object 的任何成员被第一次引用时，Load 事件会发生
- 当配置一个 object 实例时（属于 New 运算符的一个执行阶段），Initialize 事件会发生

缺省值

下面展示了各种类型的缺省值，变量总是隐式地被预先分配缺省值（根据类型不同而不同）

Boolean	Byte	Short	Integer	Long	Single	Double	String	Date	Variant	Array	Object
False	0	0	0	0	0.0	0.0	“”	Nothing	any default value	Nothing	Nothing

类型转换

下表展示了类型之间转换的效果

From/To	Boolean	Byte	Short	Integer	Long	Single	Double	String	Date	Variant	Array	Object
Boolean		True -> -1 False -> 0	True -> -1 False -> 0	True -> -1 False -> 0	True -> -1 False -> 0	True -> -1.0 False -> 0.0	True -> -1.0 False -> 0.0	True -> "True" False -> "False"	Compile time error		Compile time error	Compile time error

Byte	0 -> False other values-> True							Integer value in decimal format	Compile time error		Compile time error	Compile time error
Short	0 -> False other values-> True	Possible loss of significant digits						Integer value in decimal format	Compile time error		Compile time error	Compile time error
Integer	0 -> False other values-> True	Possible loss of significant digits	Possible loss of significant digits					Integer value in decimal format	Compile time error		Compile time error	Compile time error
Long	0 -> False other values-> True	Possible loss of significant digits	Possible loss of significant digits	Possible loss of significant digits				Integer value in decimal format	Compile time error		Compile time error	Compile time error
Single	0 -> False other values-> True	Possible loss of significant digits or loss of precision	Possible loss of significant digits or loss of precision	Possible loss of significant digits or loss of precision	Possible loss of significant digits or loss of precision			See JavaDoc for the java.lang.Double method toString()	Compile time error		Compile time error	Compile time error
Double	0 -> False other values-> True	Possible loss of significant digits or loss of precision	Possible loss of significant digits or loss of precision	Possible loss of significant digits or loss of precision	Possible loss of significant digits or loss of precision	Possible loss of significant digits or loss of precision		See JavaDoc for the java.lang.Double method toString()	Compile time error		Compile time error	Compile time error
String	"False" -> False "True" -> True Other values ->	Numerical values, with possible loss of significant digits or	Numerical values, with possible loss of significant digits or	Numerical values, with possible loss of significant digits or	Numerical values, with possible loss of significant digits or	Numerical values, with possible loss of significant digits or	Numerical values, with possible loss of significant digits or		Compile time error		Compile time error	Compile time error

	Runtime error	loss of precision. Other values -> Runtime error	loss of precision. Other values -> Runtime error	loss of precision. Other values -> Runtime error	loss of precision. Other values -> Runtime error	loss of precision. Other values -> Runtime error	loss of precision. Other values -> Runtime error					
Date	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error			Compile time error	Compile time error
Variant	Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.		Depending on the type of the actual value. Runtime error if not possible.	Depending on the type of the actual value. Runtime error if not possible.
Array	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error		Element types must be identical. Dimensions must match, unless target is not dimensioned.	Compile time error
Object	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error	Compile time error		Compile time error	Assignment compatible with base object or interface object implemented by the object. Otherwise

												compiler time error.

Wider 类型和 Narrow 类型

对于数值和类型而已，一个 wider 类型比一个 narrower 类型拥有更大的值域范围。Narrower 类型转成 wider 类型，可能会被截断，造成精度缺失。当 Single 和 Double 转成 Long 或者 Integer，小数部分被截成 0。Boolean 类型可以看做是表示-1 和 0，是一个范围最小的数值类型

Narrow						**Wide**
Boolean	Byte	Short	Integer	Long	Single	Double

对于 object 类型，一个基本的 object 可以认为是 wider 类型，而一个派生 object 却是 narrow 类型。

程序构成单元 Program Units（aka 源文件）

SimpleProgramUnit := { Aliasdeclaration }
{Declaration }
PropertiesSection

一个 Simple 程序单元由一系列的声明构成。声明别名（alias declarations）仅限制在每个程序单元的开始处。所有的程序单元都以一个属性区域（如附录中描述）结束。这个熟悉区域由特定的工具维护，不能直接修改。
Simple 共有 3 种不同的程序单元，他们的区别在于各自的属性区域存储的内容不同

- Object 程序单元可以定义对象 object 或者从基类派生
- Interface 程序单元用于定义一个接口，不能包含任何的实现
- Form 程序单元用于定义一个用户接口 UI 的表单

所有的程序单元必须定义在一个包（package）内。一个 package 就是一个命名空间，用于区分相同名字的程序单元。包可以嵌套。一个嵌套包的包名由最外层的包名加上被嵌套的包名，它们之间用点“.”分隔开。

声明 Declaration

Declaration :=
VariableDeclaration | ConstantDeclaration | FunctionDeclaration | ProcedureDeclaration | PropertyDeclaration | EventDeclaration | EventHandlerDeclaration

不存在隐式的声明

变量声明 Variable Declaration

VariableDeclaration :=
[**Static**] **Dim** Identifier **As** Type { **,** Identifier **As** Type }

变量可以使一个对象数据成员，或者是局部变量。它们都将被初始化为各自类型的缺省值。对象数据成员可以声明为静态 Static，所有的该对象实例共同引用一个静态数据成员。而非静态（实例）成员都各自分配一个对象实例

```
‘正确例子：变量声明

Dim data1 As Integer          ‘实例数据成员
Static Dim data2 As Integer , data3 As Integer     ‘对象数据成员

Sub Foo( )
   Dim localData As Integer   ‘局部变量
End Sub
```

```
‘错误例子：变量声明
Sub Foo ( )
   Static Dim localData As Integer   ‘编译时期错误：不能有‘静态’局部变量
End Sub
```

常量声明

ConstantDeclaration :=
Const Identifier **As** Type = ConstantExpression { **,** Identifier **As** Type = ConstantExpression }

常量必须是一个对象数据成员，必须在声明的时候初始化赋值。

```
‘正确例子：常量声明

Const ZERO As Integer = 0
Const ONE As Integer = ZERO + 1 , TWO As Integer = ONE +1
```

```
‘错误例子：常量声明
Dim Zero As Integer
Const ZERO As Integer = Zero ‘编译时期错误：需要常量值

Sub Foo ( )
   Const LOCAL_ZERO As Integer = 0   ‘编译时期错误：不存在局部常量
End Sub
```

函数 Functions 和过程 Procedure 声明

FunctionDeclaration :=
[**Static**] **Function** Identifier **(** [FormalArguments] **) As** Type
Statements
End Function

ProcedureDeclaration :=
[**Static**] **Sub** Identifier **(** [FormalArguments] **)**
Statements
End Sub

FormalArguments := FormalArgument { **,** FormalArgument }
FormalArgument := [**ByRef | ByVal**] Identifier **As** Type

函数和过程都是对象成员，都是可嵌套的。两者之间的不同在于：函数有一个返回值而过程没有。函数隐式声明一个局部变量来保存返回值，该局部变量命名跟函数名相同。当函数结束时，该变量的值就是函数的返回结果值。

除非特别声明，参数默认是按值传递。关键字 **ByVal** 是可选的。当参数被关键字 **ByRef** 修饰时，就变成按引用传递。此时，当一个 l-value 按引用传递给参数时候，所有被调用实参的修改都将反映到 l-value 的值。如果一个 r-rvalue 按引用传递给参数，它将被当做按值传递

```
‘函数和过程声明例子

Sub ParenthesizedExpressionExamples ( )
   Dim par1 as Integer , par2 As Integer , par3 As Integer

   par1 = 1
   par2 = 2
   par3 = 3
   TestProcedure ( par1 , par2 , par3 , TestFunction ( ) )
   ‘ par1 还是 1，par2 还是 2，但是 par3 变成 4，第四个实际参数不是一个 lvalue-所以没有变化
End Sub

Sub TestProcedure ( par1 As Integer , ByVal par2 As Integer , ByRef par3 As Integer , ByRef par4 As Integer )
    par1 = 2
    par2 = 3
    par3 = 4
    par4 = 5
End Sub

Function TestFunction ( ) As Integer
    TestFunction = 4   ‘会返回 4
End Function
```

属性声明 Property Declaration

PropertyDeclaration :=

Property Identifier **As** Type
[**Get**
Statements
End Get]
[**Set**
Statements
End Set]
End Property

属性是对象成员。一个属性声明可以定义一个 getter 函数 和/或 一个 setter 函数。如果没有什么 setter 函数，那么该属性是只读的，不能放在一个赋值语句的左边被引用。如果不定义 getter，就变成只写的，只能在一个赋值语句的左边被引用

getter 函数隐式声明一个跟属性同名的局部变量。当 getter 函数结束时，该变量的值就是属性的值。setter 函数同样也有一个类似的局部变量。当 setter 开始执行时，这个变量值就被赋值成属性的值。一个属性的 getter 和 setter 函数不能再声明任何正式的参数

```
‘属性声明例子

Dim currentStatus As String    ‘Backing storage for Status property

Property Status As String
   Get
      Status = currentStatus    ‘返回 status
   End Get

   Set
      currentStatus = Status    ‘保持 status
      StatusLabel.Text = Status ‘更新  StatusLabel
   End Set
End Property
```

```
Sub SomeProcedure ( )
   Status = “Greate!”   将 currentStatus 赋值“Greate!”，同时显示 StatusLabel 的文本
End Sub
```

事件声明

EventDeclaration :=
Event Identifier **(** [FormalArguments] **)**
End Event

事件属于对象成员。事件声明定义一个事件，该事件可以被 RaiseEvent 语句触发

```
‘声明一个事件的例子

Event TestEvent ( str As String )
End Event
```

事件处理声明

EventHandlerDeclaration :=
Event Identifier **.** Identifier **(** [FormalArguments] **)**
[Statements]
End Event

事件处理声明同样属于对象成员。一个对象处理器跟一个过程（procedure）类似，但是不能被直接触发。事件处理器都是由定义该事件的对象的 RaiseEvent 语句调用。一个事件处理器定义了一个对象实例数据成员（第一个 **Event** 关键字前面标识符）和一个事件（第二个标识符）。实例数据成员必须在 event handler 所在的对象中声明。实例数据成员必须定义相关的事件。

```
‘定义事件处理器的相关例子
```

```
Dim Obj As ObjectWithEvent     ‘该变量包括了已定义一个 TestEvent 的对象

Event Obj . TestEvent ( str As String )
    StatusLabel = “Status : “ & str
End Event
```

别名声明 Alias 声明

AliasDeclaration :=
    Alias Identifier = QualifiedIdentifier

别名声明为一个已经存在的名字重新定义了别名。后面使用这个别名就等同于使用全名。别名声明只能在 变量，函数，过程，属性，事件声明前面出现。不能重定义一个别名，否则出现编译错误。
所有来自 com.google.devtools.simple.runtiom 或者 它的任何子包包的对象自动使用已经定义的别名

```
‘正确例子
‘你不仅可以用别名声明来什么较短的名称，也可以用来消除重名

Alias FirstObjectType = com.google.devtools.example1.ObjectType
Alias SecondObjectType = com.google.devtools.example2.ObjectType

Dim data1 As FirstObjectType
Dim data2 As SecondObjectType
```

```
‘错误例子
‘重复定义别名是不允许的

Alias Form = com.google.devtools.example1.objectType    ‘编译时期错误
```

```
Alias ObjectType = com.google.devtools.example1.ObjectType
Alias ObjectType = com.google.devtools.example2.ObjectType      ‘编译时期错误
```

作用域

在一个包（package）中定义的对象是全局可访问的。一个对象可以与其他包的对象同名，但是不能与同一个包的对象重名。

对象成员（数据成员，函数，过程，属性和事件）声明是全局可访问，可能只定义在 object 范围内。另外的同名对象成员不能定义在同一对象里。为此，函数，过程，属性和事件都不允许重载。派生类可以允许覆盖基类的函数（过程，属性和事件）。数据成员也是不允许在派生类中重复定义。

函数，过程和事件参数在其所在的函数作用域有效。不能在参数系列重复定义相同的参数。

局部变量可以在一个函数，过程，属性和事件的任何地方声明。它们的作用域从开始声明的地方，延续到其他能被继续访问的区域。同一个 statement block 不能重复声明局部变量。一个局部变量将会屏蔽掉所有外部的同名变量，包括 statement block 外的和对象的数据成员等。

```
‘作用域例子

Static Dim var1 As String
Static Dim var2 As Integer
Static Dim var3 As Object

Event ScopeTest . Load ( )
   var1 = “Not a number”
   var2 = 0
   var3 = Nothing
End Event
```

```
Static Sub TestProcedure ( var1 As Integer )
   Dim var2 As String

   Do
      Dim var2 As Object
      Dim var3 As String

      var1 = com.google.devtools.simple.smoketest.scopes.ScopeTest.var2
      var2 = ScopeTest . var3
   While False
End Sub
```

表达式 Expressions

操作符优先级

操作符必须按照他们的优先级顺序使用。拥有高优先级的操作符先执行运算。

操作符优先级（从高到低）

- ^ 幂运算
- + - 正数和负数运算
- / 乘法和除法
- \ 整除
- Mod 求模
- + - 加法和减法
- & 字符串连接运算
- << >> 位移运算
- < <= > >= = <> Is IsNot Like TypeOf… Is 比较运算
- Not 逻辑和位运算求反
- And 逻辑和位运算求结合
- Or Xor 逻辑和位运算求析取
- = 赋值运算

结合性 Associativity

如果表达式包含了多个相同优先级的运算，那么按照相同级别从左至右依次运算

Simple 表达式

PrimaryExpression := **Me** | Identifier | QualifiedIdentifier | Literal | PraenthesizedExpression | CallOrArrayExpression | NewExpression

Me 表达式

Me 关键字可以用在实例函数，过程，属性和事件中，表示对当前实例对象的引用。

标识符 Identifiers

标识符是指语言词法规定中定义的标志（例如 variables ,contants, namespace types,function,procedures,properties 和 events）

已验证的标识符 Qualified Identifiers

QualifiedIdentifier := [PrimaryExpression .] Identifier

有两种形式的已验证标识符。第一种用来消除疑义当它可能指向多个标识（symbol） 。 这种情况下 primary 表达式指向一个包名。包名必须是唯一的。另一种情况，primary 表达式指向一个对象的实例而这个标识符表示其中的对象成员

字面量/常量表达式（Literals/Constant）

ConstantExpression := Expression

一个常量表达式是指语言词法规定中定义的连续字面量与其他合适操作符结合的表达式。

调用和数组表达式

CallOrArrayExpression := QualifiedIdentifier **(** [ActualArguments] **)**
ActualArguments := Expression { **,** Expression }

一个合法的标识符后面跟着括号，可能是一个函数调用或者是数组表达式，这取决于这个标识符所关联的类型。
如果这个标识符关联一个数组类型，那么后面一定跟着一串索引供数组访问。索引的个数必须跟数组的维数相匹配。每个单独的索引必须是合适的 Integer 类型

如果这个标识符是一个函数或者过程名字，后面一定紧跟着一系列的函数参数。实际参数的个数必须符合形式参数的个数。每个实参的类型也必须符合函数定义时候形参的类型。
参数都是从左至右依次验证。

调用的类型取决于函数或过程本身。如果一个函数或过程是一个对象函数或过程，它们可以直接被调用。如果函数或过程是一个实例函数或过程，那么它将通过一个对象实例来调用。当这个实例对象是一个 interface 对象，运行时实例对象的实际类型是通过名称来进行 lookup

```
‘调用和数组表达式例子

Dim Array As Integer( 2 , 3 )

Sub CallAndArrayExpressionExamples ( )
    Array ( 1 , 2 ) = 3
    Procedure ( Array ( 1 , 2 ) , “4”)
End Sub

Sub Procedure ( arg1 As Integer , arg2 As Integer )
End Sub
```

圆括号表达式

ParenthesizedExpression := **(** Expression **)**

圆括号可以用于更改 （Override） 左结合性 和 操作符的优先级

```
‘圆括号表达式例子

Sub ParenthesizedExpressionExample ( )
    Dim result As Integer

    ‘运算符优先级
    result = 2 ^ 4 + 1    ‘结果是 17
    result = 2 ^ ( 4 + 1) ‘结果是 32

    ‘左结合性
    result = 2 – 4 + 1
    result = 2 – ( 4 + 1 )
End Sub
```

New 表达式

NewExpression := **New** QualifiedIdentifier [**On** QualifiedIdentifier] | **New** QualifiedIdentifier (Expression { **,** Expression } **)**

New 表达式用于为一个对象分配内存和初始化，或者动态确定数组实例的大小。
对一个对象分配内存和初始化对象数据成员的值（根据类型赋缺省值），New 表达式会触发对象实例的 initialization 事件。 如果被分配资源的对象是一个组件，组件名称后面必须跟关键字 **On** 和放置组件的容器。组件先放置入容器，然后 initialization 事件才发生。
当对一个动态数组分配资源时，每一维的大小将通过尝试传递给 **New** 操作符。分配数组资源结束后，每个数组元素都进行了初始化（根据类型赋缺省值）

```
‘ New 操作符表达式例子

Sub NewExpressionExamples ( )
    Dim label As Label
    Dim obj As Object
    Dim array1 As Integer ( ) , array2 As Integer ( , )

    label = New Label On MainForm
    obj = New SampleObject
    array1 = New Integer ( 5 )
    array2 = New Integer ( 2 , 3 )
End Sub
```

复合表达式

Expression := LogicalOrBitOpExpression

复合表达式包括一个操作符和若干个操作数

求幂操作符

ExponentiationExpression := PrimaryExpression | PrimaryExpression ^ ExponentiationExpression

二元操作符 ^ 用于右操作数个左操作数相乘 即 power（左操作数，右操作数）
如果需要(除了 Double)，操作数会在运算之前被强制转换成 Double 类型。运算结果是 Double 类型
浮点数的求幂运算符合 IEEE 754 数值规则
除了类型转换的错误，求幂运算本身不产生运行时错误，尽管可能存在上界下界溢出，精度缺失等情况。

```
‘求幂操作表达式运算

Sub ExponentiationExpressionExamples ( )
```

```
Dim result As Integer
   result = 2 ^ 4 + 1
   result = 2 ^ ( 4 + 1 )
End Sub
```

正号和负号操作符

IdentityNegationExpression := ExponentiationExpression | IdentityNegationOperator IdentityNegationExpression

IdentityNegationOperator := + | -

一元操作符 " ＋ " 不改变操作数的值。而“－”使操作数的值求反。如果需要，在运算之前，操作数需要先转换成数值类型（Byte ,Short , Integer , Long , Single or Double ）。运算结果的类型跟操作数的类型相同（或转换类型后的操作数类型）

浮点数的求反参照 IEEE 754 数值规则。
除了类型转换错误，该操作符本身不产生任何运行时错误，尽管可能造成精度缺失。

```
+ / -操作符表达式例子

Sub NegationExpressionExample ( )
    Dim result As Integer

    result = 4
    result = +result
    result = - result
End Sub
```

乘法和除法操作符

MultiplicationExpression := IdentityNegationExpression | IdentityNegationExpression MultiplicationOperator MultiplicationExpression

MultiplicationOperator := * | /

二元运算符“*”对两个操作数执行乘法运算。
如果需要，操作数在运算之前将会被转换成数值类型（Byte ,Short , Integer , Long , Single or Double 等）。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。运算结果的类型跟操作数的 common 类型相同。
浮点数的除法运算参照 IEEE 754 数值规则。
除了类型转换的错误，该操作符不产生任何运行时错误，尽管可能出现溢出，精度缺失等情况。

二元运算符“/”将执行左操作数除以右操作数
如果需要，操作数在运算之前将会被转换成数值类型（Byte ,Short , Integer , Long , Single or Double ）。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。运算结果的类型跟操作数的 common 类型相同。
浮点数的除法运算参照 IEEE 754 数值规则。
除了类型转换的错误，当右操作数为 0 时会产生运行时错误，其他情况不产生运行时错误，尽管可能出现溢出，精度缺失等情况。

```
‘除法和除法运算符表达式例子

Sub MultiplicationExpressionExamples ( )
    Dim result As Integer

    result = 12 / 2 * 6          ‘结果 36
    result = 12 / ( 2 * 6 )      ‘结果 1
    result = 12.5 / 0            ‘运行时错误
End Sub
```

整除操作符

IntegerDivisionExpression := MultiplicationExpression | MultiplicationExpression \ IntegerDivisionExpression

二元操作符“\”用于左操作符整除右操作符
如果需要，操作数在运算之前将会被转换成数值类型（Byte ,Short , Integer , Long , Single or Double ）。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。运算结果的类型是 Integer
浮点数的除法运算参照 IEEE 754 数值规则。
除了类型转换的错误，当右操作数为 0 时会产生运行时错误，其他情况不产生运行时错误，尽管可能出现溢出，精度缺失等情况。

```
‘整除操作符表达式
Sub IntegerDivisionExpression ( )
    Dim result As Integer

    result = 12.5 \ “3.1”   ‘结果是 4
    result = 12.5 \ 0       ‘运行时错误
End Sub
```

求模操作符

ModuloExpression := IntegerDivisionExpression | IntegerDivisionExpression **Mod** ModuloExpression

二元操作符 **Mod** 是指 求 左操作数 除以 右操作数 后得到的余数，可以用公式定义 （left / right）*right + (left Mod right) = left
如果需要，操作数在运算之前将会被转换成数值类型（Byte ,Short , Integer , Long , Single or Double ）。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。运算结果的类型跟操作数的 common 类型相同。
浮点数的除法运算参照 IEEE 754 数值规则。
除了类型转换的错误，当右操作数为 0 时会产生运行时错误，其他情况不产生运行时错误，尽管可能出现溢出，精度缺失等情况。

```
‘求模操作表达式例子

Sub ModuloExpressionExample ( )
    Dim result As Integer

    result = 12 Mode 3   ‘0
    result = 11 Mode 3   ‘2
    result = -11 Mod 3   ‘-2
    result = 12.5 Mod 0  ‘会产生一个运行时错误
End Sub
```

加法和减法操作符

AditonExpression := ModuloExpression | ModuloExpression AdditionOperator AdditionExpression

AdditionOperator := + | -

二元运算“+”对两个操作数执行加法运算，“-”对两个操作数执行减法运算（右减左）
如果需要，操作数在运算之前将会被转换成数值类型（Byte ,Short , Integer , Long , Single or Double ）。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。运算结果的类型跟操作数的 common 类型相同。
浮点数的除法运算参照 IEEE 754 数值规则。
除了类型转换的错误，该操作符不产生任何运行时错误，尽管可能出现溢出，精度缺失等情况。

```
‘加法减法操作符表达式

Sub AditonExpressionExample (arg As String )
    Dim result As Integer

    result = 2 + “5” – 1 ‘结果是 6
    result = result + arg     ‘可能出现运行时错误，如果 arg 不能转换成一个数字
End Sub
```

字符串连接操作符

StringConcatenationExpression := AdditionExpression | AdditionExpression **&** StringConcatenationExpression

二元操作符“**&**”将两个操作数（字符串）连接成一个字符串。如果需要，操作数在运算之前将会被转换成 Sting 类型。运算结果是 String 类型。除了类型转换的错误，该操作符不能产生任何运行时错误

```
‘字符串连接符例子

Sub StringConcatentionExpressionExamples ( )
```

```
Dim result As String

result = “abc”&2+”5”   ‘结果是 7
End Sub
```

位移操作符

BitShiftOperator StringConcatenationExpression
BitShiftOperator := << | >>

二元操作符<<将左操作数 左移 右操作数 位，而>>则是带符号右移。
如果需要，操作数在运算之前将会被转换成数值类型（Byte ,Short , Integer , Long , Single or Double ）。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。运算结果的类型跟操作数的 common 类型相同。
除了类型转换的错误，该操作符不产生任何运行时错误

```
‘位移操作符表达式例子

Sub BitShiftExpressionExamples ( )
    Dim result As Integer

    result = &H12348080
    result =result << 16   ‘结果是 &H80800000
    result =result >> 16   ‘结果是 &HFFFF8080
End Sub
```

比较，Is, IsNot , Like And TypeOf …Is 操作符

ComparisonExpression := BitShiftExpression | BitShiftExpression ComparisonOperator ComparisonExpresion
ComparisonOperator := OrderedComparisonOperator | **Is** | **IsNot** | **Like** | **TypeOf** PrimaryExpression **Is** Type
OrderedComparisonOperator := < | <= | = | < > | >= | >

二元运算符< 比较左操作数是否在数值上小于右操作数。 <= 比较左操作符在数值上是否小于或者等于右操作数。>, >=操作符功能类似。如果需要，操作数在运算前将会被转成一个 integer 类型（Byte，Short，Integer or Long）或者 String 类型。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。如果两个操作数的类型都是 String，那么它们将按照词典编纂顺序比较（根据 Java 方法 java.lang.String.**compareTo()**）.
=比较两个操作数在数值上是否相等。<>比较两个操作数在数值上是否不同。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。

Is 比较左操作数是否等于右操作数。**IsNot** 比较左操作数不等于右操作数。注意，操作数必须是 Object 或者 Array 类型。

Like 表示左操作数是否符合右操作数给出的正则表达式。正则表达式跟 Java 中定义的 java.util.regex.Pattern 相同。如果需要，操作数在运算之前会被转换成 String 类型

TypeOf。。。Is 比较左操作数是否跟右操作数类型相同
操作符的结果是 Boolean。
除了转换错误，该操作符不会产生任何运行时错误

```
‘比较操作符表达式例子

Sub ComparisonExpressionExamples ( )
    Dim result As Boolean

    result = “bar” < “foo”   ‘结果是 True
    result = 2 < “one”       ‘结果是 True
    result = “bar”=”foo”     ‘False
    result = result <> True  ‘True
End Sub

Sub IsExpressionExample ( )
    Dim result As Boolean
    Dim obj1 As Object , obj2 As Object
```

```
    obj1 = SomeObject
    obj2 = SomeObject
    result = obj1 Is obj2      'True

    obj2 = DifferentObject
    result = obj1 Is obj2      'False
End Sub

Sub LikeExpressionExamples ( )
    Dim result As Boolean

    result ="foof " like "f.*f"    'True
    result = "goof" like "f.*f"    'False
End Sub

Sub TypeOfExpressionExamples ( )
    Dim result As Boolean
    Dim obj As Object
    Dim var As Variant

    obj=Nothing
    var=1

    result = obj TypeOf Integer 'False
    result = obj TypeOf Integer() 'False
    result = obj TypeOf Object    'True
    result = obj TypeOf Foo       'True

    obj = SomeObject
    result =obj TypeOf Foo    'True 当 obj 是 Foo 类型或者 Foo 的基类

    result = var TypeOf Integer    'True
    result = var TypeOf String     'result is True
```

```
    result = var TypeOf Object    'False
End Sub
```

逻辑和位操作符

LogicalOrBitOpExpression := ComparisonExpression | ComparisonExpression LogicalOrBitOperator LogicalOrBitExpression | **Not** LogicalOrBitOpExpression
LogicalOrBitOperator := **And | Or | Xor**

二元操作符 **And** 对它的操作数执行 AND 位运算。二元操作符 **Or** 对它的操作数执行包括的（inclusive）OR 位运算。二元操作符 **Xor** 对它的操作数执行互斥的（exclusive）OR 位运算（异或）。如果需要，操作数在运算之前将会被转换成数值类型（Byte ,Short , Integer , Long , Single or Double ）。如果两个操作数的类型不同，narrower 类型将会被转成 wider 类型。运算结果的类型跟操作数的 common 类型相同。

一元操作符 **Not** 对它的操作数进行求反位运算。如果需要，操作数在运算之前将会被转换成数值类型（Byte ,Short , Integer , Long , Single or Double ）。
忽略可能存在的错误，逻辑和位操作符本身不产生任何运行时错误
对于 Boolean 类型的操作数，把他们当做只有一位操作数进行运算。

```
'逻辑位操作符表达式例子
Sub BitExpressionExamples ( )
    Dim result As Integer

    result = &H12348080
    result = result And &H0000FFFF    '&H00008080
    result = result Or &H80800000     '&H80808080
    result = result Xor &H0000FFFF    '&H80807F7F
    result = Not result               '&H7F7F8080
End Sub

Sub LogicalExpressionExample ( )
    Dim result As Boolean
```

```
    result = False
    result = result And True     'False
    result = result Or True      'True
    result = result Xor True     'False
    result = Not result          'True
End Sub
```

赋值操作符

AssignmentExpression := LValue = Expression
LValue := QualifiedIdentifier | CallOrArrayExpression

一元操作符 = 将右操作数赋值给左操作数。左操作数必须是可更改的，意味着它是一个变量（局部变量或者数据成员）或者数组成员。如果需要，右操作数将会被转换成跟左操作数相同类型。这个转换必须是赋值可兼容的。忽略可能的转换错误，赋值操作符本身不会产生任何运行时错误。

```
'赋值表达式例子

Sub GoodExpressionExample ( )
    Dim i As Integer
    Dim a As Integer( 2 )

    i = "1"   '赋值给变量
    a(i) = i  '赋值给数组成员
End Sub

'赋值表达式错误例子

Function One ( ) As Integer
    One =1
End Function
```

```
Sub BadExpressionExample ( )
    Dim i As Integer

    One( ) = 2 ‘编译时期错误，不能赋值给一个函数
    i+1 = 3    ‘编译时期错误，不能赋值给一个表达式
End Sub
```

语句（Statements）

Statements := Statement | Statements
Statement := DoWhileStatement | DoUntilStatement | ExpressionStatement | ExitStatement | ForEachStatement | ForToStatement | IfStatement | OnErrorStatement | RaiseEventStatement | REMStatement | SelectStatement | WhileStatement | VariableDeclaration

函数体，事件，属性和过程等都包括了一系列的语句和变量声明

Do…While 语句

DoWhileStatement :=
    Do
        Statements
    While Expression

Do…While 重复执行 Do 后面的语句块，直到 While 表达式为 **True**。Do 后面的语句块至少执行一次，因为相关的条件语句都是在语句块后面才计算。可以在语句块里面执行一个 Exit 语句阻止 Do。。。While 的继续执行

```
‘Do…While 语句例子

Sub DoWhileExample ( )
    Dim counter As Integer
    ‘counter 为 0
    Do
        counter = counter + 1
```

```
    While counter < 20
    ‘counter is 20
End Sub
```

Do…Util 语句

DoUntilSatement :=
    Do
        Statements
    Until Expression

Do…Until 重复执行 Do 后面的语句块，直至 Until 后面的表达式为 **True**。Do 后面的语句块至少执行一次，因为相关的条件语句都是在语句块后面才计算。可以在语句块里面执行一个 Exit 语句阻止 Do。。。Until 的继续执行

```
‘Do…Until 语句例子

Sub DoUntilExample ( )
    Dim counter As Integer
    ‘conter  为  0
    Do
       counter = counter + 1
    Until counter >= 20
    ‘ counter  变成 20
End Sub
```

表达式语句

ExpressionStatement := AssignmentExpression | CallOrArrayExpression

表达式语句包括赋值表达式和调用（call）表达式

```
‘表达式语句例子
```

```
Function One( ) As Integer
    One = 1
End Function

Sub GoodExpressionExample ( )
      Dim i As Integer
      Dim a As Integer( 2 )

      i = "1"     '赋值给变量
      a ( i ) = One ( ) '赋值给数组成员
      One ( )           '函数调用 =Ok  虽然  结果没有用到
End Sub
```

```
'表达式语句错误例子

Sub BadExpressionExample ( )
    Dim a As Integer ( 2 )

    a( i )  '编译时期错误：既不是一个赋值表达式也不是一个调用表达式 –只是一个数组访问
End Sub
```

Exit 语句

ExistStatement :=
Exit [**Do** | **Event** | **For** | **Function** | **Property** | **Sub** | **While**]

Exit 语句用于退出父级循环语句，或者退出函数，过程，属性和事件。
如果 **Exith** 后面没有跟其他的关键字，编译器会查找周围最靠近的循环语句。如果后面有一个关键字，那么循环将会在 Exit 语句执行时退出。如果没有找到循环，Exit 所在的函数，过程，属性或者时间将结束。
如果 Exit 后面跟着 **Do** ,**For** 或者 **While**，编译器将会寻找周围最近的循环语句。如果找到，循环将会在 Exit 语句执行时退出。如果没有找到循环，将会报错。
如果 Exit 后面跟着 **Event** ，**Fuction** , **Property** 或者 **Sub**，那么 **Exit** 所在的函数，过程，属性，或者事件都将结束

```
‘Exit 语句例子
Fuction ExitFromDoWhileExample (exitFromFunction As Boolean ) As Boolean
 Do
     If exitFromFunction Then
         ExitFromWhileExample = False
         Exit Function
     Else
         ExitFromDoWhileExample = True
         Exit      ‘退出循环，但不是退出函数
     End If
  While True
End Function
```

For Each 语句

ForEachStatement :=
    For Each <identifier> **In** <expression>
        Statements
    Next [<identifier>]

For Each 语句执行 **In** 后面的语句块，In 后面的表达式给出的数组或者集合的每个元素都将执行一次。每个数组或者集合元素都被分配相关的循环变量。循环变量一定是跟数组或者集合元素可兼容的。Exit 语句可以使 For Each 语句退出。循环的终点是 **Next** 关键字，后面跟着可选的循环变量标识符

```
‘For Each 语句例子
Dim c As Collection
Dim a As Integer ( 2 )

Event ForEachExample.Initialize( )
    ‘初始化集合
    c = New Collection
```

```
        c.Add ( “one” )
        c.Add ( 2 )
        c.Add (Me)
End Event

‘一个集合的 For Each 例子

Function ForEachCollectionExample ( ) As Integer
    Dim v As Variant
    For Each v In GetCollection ( )
        ForEachCollectionExample = ForEachCollectionExample + 1
    Next v
End Function

‘一个数组 For Each 语句例子
Function ForEachArrayExample ( ) As Integer
    Dim i As Integer
    For Each i In GetArray ( )
        ForEachArrayExample = ForEachArrayExample + 1
    Next i
End Function

Function GetCollection( ) As Collection
        GetCollection = c
End Function

Function GetArray ( ) As Integer ( )
        GetArray = A
End Function
```

For … To 语句

ForToStatement :=

For<identifier> = Expression **To** Expression [**Step** Expression]

Statements

Next [<identifier>]

For…To 语句将一直执行语句块，直到 For 后面的循环变量等于或者超过 **To** 后面的表达式的值。

在循环执行开始时，循环变量被赋值成 = 和 **To** 之间的表达式的值。每执行一次循环，循环变量都会增加 1。如果有关键字 Step，循环变量每次将递增 n（n 为 **Step** 后面表达式的值）。对于一个正的 step 表达式，如果表达式的值大于或者等于 end expression(To 后面的 Expression 值)，循环将结束。对于一个负的 step 表达式，如果表达式的值小于或者等于 end expression，循环将结束。end expression 和 上面提到的 n 都只计算一次，不会每次循环都计算。注意到 For。。。To 语句不隐式定义循环变量。循环变量必须跟初始值，结束值和 step 值相兼容。Exit 语言同样可以退出循环。循环的终点是 **Next** 关键字，后面跟着可选的循环变量标识符。

```
‘ For … To 表达式例子

Function ForIntegerExample ( start As Integer , stop As Integer ) As Integer
    Dim i As Integer
    For i = start To stop
        ForIntegerExample = ForIntegerExample + i
    Next i
End Function

Function ForWithStepDoubleExample (start As Double, stop As Double, step As Double )
As Double
        Dim d As Double
        For d = start To stop Step step
            ForWithStepDoubleExample = ForWithStepDoubleExample + d
        Next    ‘没有循环变量名 d 也是 ok 的
End Function
```

If 语句

```
IfStatement :=
        If Expression Then Statement [ Else Statement ]
    |
        If Expression Then
            Statements
        { ElseIf Expression Then
            Statements }
        [ Else
```

Statements]
End If

If 语句按照条件执行相关的语句。如果 **If** 关键字后面的语句计算结果为 True(可能需要类型转换)，那么关键字 Then 后面的语句将被执行，If 语句终止。反之，如果 If 关键字后面的计算结果为 False，并且存在可选的 ElseIf 语句，那么所有 **ElseIf** 后面的语句被依次计算，直到有一个为 True，该 ElseIf 对应的 Then 后面的语句将被执行，ElseIf 结束。最后，如果没有 If 或者 ElseIf 后面的语句计算结果为 Tue，而存在 **Else** 关键字，此时 **Else** 后面的语句将被执行。

```
‘各种 If 语句例子
‘下面的常量用来当做 test 函数的返回值，他们标志哪个语句被真正执行了

Const NONE As String = “”
Const THEN As String = “Then”
Const ELSETHE As String = “ElseThen”
Const ELSE As String = “Else”

‘只有简单一条 If 语句没有 Else
Function SingleLineIfThenExample(testValue As Boolean) As String
    If testValue Then SingleLineIfThenExample = THEN
End Function

‘一条 If 语句和一条 Else
Function SingleLineIfThenElseExample ( testValue As Boolean ) As String
    If testValue Then SingleLineIfThenElseExample = THEN
    Else
        SingleLineIfThenElseExample = ELSE
End Function

‘多条 If 语句，没有 Else 语句
Function IfThenExample ( testValueForThen As Boolean ) As String
    If testValueForThen  Then
        IfThenExample = THEN
    End If
End Function
```

```
Function IfThenElseExample (testValueForThen As Boolean ) As String
    If testValueForThen Then
        IfThenElseExample = THEN
    Else
        IfThenElseExample = ELSE
    End If
End Function

‘多条 If 语句和 ElseIf 语句，没有 Else 语句
Function IfThenElseIfExample (testValueForIf As Boolean , testValueForElseIf As Boolean )
As String

    If testValueForIf Then
        IfThenElseIfExample = THEN
    ElseIf testValueForElseIf Then
        IfThenElseIfExample = ELSETHEN
    End If
End Function

‘多条 If 和 ElseIf 语句，加上 Else
Function IfThenElseIfElseExample (testValueForIf As Boolean , testValueForElseIf As
Boolean ) As String
    If testValueForIf Then
        IfThenElseIfElseExample =THEN
    ElseIf testValueForElseIf Then
        IfThenElseIfElseExample =ELSETHEN
    Else
        IfThenElseIfElseExampel = ELSE
    End If
End Function
```

On Error 语句

```
OnErrorStatement :=
    On Error
        { Case Types
          Statements }
        [ Case Eles
          Statements }
```

End Error

Types := Type { , Type }

OnError 语句用于处理运行时错误。它必须放在函数，过程，时间处理或者属性的 getter 和 setter 的末尾。这也意味着每个函数只能有一个 OnError 语句。如果函数的其他语句没有产生运行时错误，那么 OnError 语句永远不会被执行

如果一个运行时异常被引发，在运行时异常发生的地方,OnError 语句会继续执行。如果函数本身没有定义 OnError 语句，那么系统将会寻找一个外层调用函数的 OnError 语句。重复这个过程，直到寻找到一个 OnError 语句，或者到达 call stack 的底部，此时应用程序会终止这个运行时错误。如果找到一个 OnError 语句，但是这个语句没有提供出来如何处理这种运行时错误，这个运行时错误将被传给调用函数。

一个 OnError 语句会在 Case 语句后定义一个错误处理器来处理一系列的运行时错误类型。特殊的 CaseElse 语句会被处理任何的运行时错误

重复在一个 OnError 语句中为一个相同的 runtime error 定义多个 error handler 将会造成编译错误

```
‘OnError 语句例子
‘函数将会返回 True 如果访问给定的数组时产生一个 runtime error

Function OnErrorExample ( array As Integer( ) , index As Integer ) As Boolean
    Dim value As Integer
    value = array ( index )

    On Error
        Case AssertionFailure , UninitializedInstanceError
          ‘不会发生，只是为了演示 OnError 语句的语法
        Case ArrayIndexOutOfBoundsError
          ‘当数组越界时候函数返回 True
           OnErrorExample = True
        Case Else
          ‘这个也不会发生，只是为了演示 OnError 的语法而已

     EndError
End Function
```

RaiseEvent 语句

RaiseEventStatement :=
RaiseEvent Identifier **(** [ActualArguments] **)**

RaiseEvent 语句会调用关键字 **RaiseEvent** 后面 Indentifier 所指的 Event handler。RaiseEvent 语句只能用在定义了相关 event 的实例函数，过程，属性中。

```
‘调用一个本地事件例子

‘定义一个本地事件（带一个整数参数）
Event TextEvent ( x As Integer )
End Event

‘调用本地事件，传入参数
Sub RaiseEventExample ( x As Integer )
      RaiseEvent TestEvent (x)
End Sub
```

Select 语句

SelectStatement :=
Select Expression
{ **Case** CaseExpressions
Statements }
[**Case Else**
Statements]
End Select

CaseExpressions := CaseExpression { **,** CaseExpression }
CaseExpression := Expression | **Is** OrederComparisonOperator Expression | Expression **To** Expression

Slelect 语句允许根据条件执行相关语句。首先，**Select** 后面的表达式会被计算，接下来计算第一个 Case 的表达式。如果两者相等，执行该 case 的语句。然后 Select 语句结束。其他的 case 语句都不执行。

如果没有符合的 case 语句，任何 Select 的 Statements 都不执行。Select 后面的 expression 和每个 case 的 expression 必须类型兼容

Case 表达式可以有一个或多个括号分隔的表达式。有两种特殊的表达式。第一种是一个范围表达式 range expression，包括两个 expression，第一个是下界，第二个是上界，被 **To** 关键字区分。只要 Select 的 expression 在这个区间，那么 case 符合。另一种形式是 comparison expression，Is 关键字为首，后面跟着一个比较操作符和一个表达式。如果 Select expression 和 comparison expression 的比较结果为 True，这个 case 符合条件。

最后，如果 Case 后面跟着 Else 关键字 的语句会被执行，无需考虑 Select 的 expression 值。注意这个特殊的 case 必须是最后一个 case

```
‘Select 语句例子 ：将字符串转换成数字
Dim SpecialNumber As String

Function StringToNumbers( number As String ) As Integer
   Select number
      Case “Zero”
           StringsToNumbers = 0
      Case “One”,SpecialNumer
           StringsToNumbers = 1
      Case Else
           StringsToNumbers = 2
    End Select
End Function

‘Select 语句例子：将数字转换成字符串
Function NumbersToStrings ( number As Integer ) As String
   Select number
      Case Is < 0
          NumbersToStrings = “Negative number “
      Case 0
          NumbersToStrings = “Zero”
      Case 1
```

```
            NumbersToStrings =”One”

        Case 2 To 1000
            NumbersToStrings = “Between 2 and 1000”
        Case Else
            NumbersToStrings = “Big numbers “
    End Select
End Function
```

While 语句

```
WhileStatement :=
        While Expression
            Statements
        End While
```

While 语句会重复执行它的语句块，直到 While 后面的表达式值为 False 才结束。While 语句可以用 Exit 语句终止循环

```
‘While 语句例子

Sub WhileExample ( )
  Dim counter As Integer
    ‘counter 缺省值 0
  While counter < 20
     counter = counter + 1
  End While
     ‘counter 变成 20
End Sub
```

附录

深入 Simple 源文件 – 性质区域 Properties Section

如果你只是对如何使用 Simple 语言感兴趣，可以略过这一部分

每个源文件末尾都有一个性质区域。Form 源文件定义了 form 使用的组件和这些组件的性质。Object 源文件定义了基类（如果存在）和实现的所有接口。Interface 源文件只是简单说明本身是一个接口

支持 Simple 语言的编辑器应该隐藏源文件的性质区域。对于 form 源文件，它应该提供一个可视化的设计器，运行添加和删除组件到 form。还应该为组件提供属性编辑器。对于 Object 源文件，它应该提供属性编辑器来设置基类和所有实现的接口。无须为 interface 源文件提供特殊支持。

性质区域关键词

PropertiesSectionKeywords := **$As** | **$Define** | **$End** | **$Form** | **$Interface** | **$Object** | **$Properties** | **$Source**

Form 源文件

FormSourcePropertiesSection :=

$Proerties

$Source $Form

$Define Identifier **$As Form**

{ ComponentPropertyOrNestedComponent }

$ End Define

$End $Properties

ComponentPropertyOrNestedComponent :=

Identifier = Literal

|

$Define Identifier **$As** QualifiedIdentifier

{ ComponentPropertyOrNextedComponent }
$End $Define

只有这些属性的值（容器实现时定义的）被改变时候，才需要保存。更多的信息请参考“容器编写指南”

```
‘form 源文件性质区域例子
$Properties
    $Source $Form
    $Define Main $As Form
        Title =”hello Word”
        $Define ByeButton $As Button
            Text = “Bye…”
        $End Define
    $End $Define
$End $Properties
```

Object 源文件

FormSourcePropertiesSection :=
$Properties
$Source $Object
[**BaseObject** = QualifiedIdentifier]
{ **ImplementsInterface** = QualifiedIndentifier }
$End $Properties

```
‘Object 源文件性质区域例子

$Properties
    $Source $Object
    BaseObject = com.youdomain.Foo
    ImplementsInterface = com.somedomain.Interface1
```

```
    ImplementsInterface = com.otherdomain.Interface2
$End $Properties
```

Interface Source Files

InterfaceSourcePropertiesSection :=

$Properties

$Source $Interface

$End $Properties

```
‘interface  源文件性质区域例子

$Properties
    $Source $Interface
$End $Properties
```

下页还有哦

译后感

翻译这篇文档，感觉一个字“很痛苦”。虽然英语自认为还算可以，平时学习工作阅读文档没有任何障碍。毕竟，to know what it’s and to describe what it’s 完全是两码事。我现在由衷佩服那些翻译好的专家了，再也不骂那些翻译不好的书籍了。

接触这门语言，是在八月初吧。看了新闻，便好奇去 look。结果引发了我差不多十年前的回忆。当时刚上初中，山村学校竟然破天荒地开设电脑课了，还发了几本书。学的都是一些基本操作，可惜五笔从那时开始到现在都没学会，泪流满面啊。其中印象深刻的是书中讲述的 Basic 语言，当时程序的魔力就吸引了我。从此走上了 IT 这条不归路。真是少壮不努力，老大干 IT。

Simple 语言是属于 Basic 语系的，所有看到那些代码，感觉似曾相识。相信你也曾经接触过吧。耗时一个星期，每天晚上下班后就翻译，不敢说信达雅，只能算勉强能看，甚至可能会对你的阅读造成误解，在此深表歉意，希望你能对我的英文翻译水平多多包涵。

我还要感谢我大学时候的《编译原理》老师 leeman，这篇文档的关于“语言定义”部分让我想起你当时的教诲，可以我不是好学生，学得很差。

以后我会陆续发布关于 Simple 的中文资料，希望能给大家一些帮助。

不需要对我表示任何感谢，因为我也经常在网上得到你无偿提供的资源。如果非要有点表示的话，希望你对这份文档提出任何修改意见或提供更多有益的资源。

关于文档的任何问题，以及Simple的任何问题，或者任何不是问题的问题，都欢迎交流。可以叫我阿健。我的email : rojaneky@gmail.com

2009-8-9