Jascal编程语言规范

2015/10/12

目录

[1. 概述 2-3](#_Toc432873881)

[1.1. 语言特性 2-3](#_Toc432873882)

[1.1.1. 运行时模型 2-3](#_Toc432873883)

[1.1.2. 面向对象 2-3](#_Toc432873884)

[1.1.3. RTTI 2-3](#_Toc432873885)

[1.1.4. 元组与多返回值 2-3](#_Toc432873886)

[1.1.5. 占位符 2-3](#_Toc432873887)

[1.1.6. 数组模型 2-4](#_Toc432873888)

[2. 定义 2-5](#_Toc432873889)

[2.1. 字符 2-5](#_Toc432873890)

[2.2. 记号 2-5](#_Toc432873891)

[2.3. 关键字 2-5](#_Toc432873892)

[2.4. 标识符 2-5](#_Toc432873893)

[2.5. 字面量 2-6](#_Toc432873894)

[3. 运算与运算符 3-7](#_Toc432873895)

[3.1. 算术运算 3-7](#_Toc432873896)

[3.2. 位运算 3-7](#_Toc432873897)

[3.3. 比较运算 3-7](#_Toc432873898)

[3.4. 逻辑运算 3-8](#_Toc432873899)

[3.5. 程序语义运算 3-8](#_Toc432873900)

[3.6. 优先级与结合性 3-8](#_Toc432873901)

[4. 字面量 4-9](#_Toc432873902)

[5. 表达式 5-10](#_Toc432873903)

[6. 语句 6-10](#_Toc432873904)

[6.1. 表达式语句 6-10](#_Toc432873905)

[6.2. 控制流语句 6-10](#_Toc432873906)

[6.2.1. if语句 6-11](#_Toc432873907)

[6.2.2. while语句 6-11](#_Toc432873908)

[6.2.3. for语句 6-11](#_Toc432873909)

[6.2.4. repeat语句 6-12](#_Toc432873910)

[6.2.5. case语句 6-12](#_Toc432873911)

# 概述

本文档定义了Jascal编程语言的词法、文法及语义。任何声明能够编译Jascal语言程序的编译器应当完全实现本文档中指定的内容，任何以Jascal语言书写的程序的最终运行效果应以本文档为准。使用在本文档中未指定的语义将导致与具体编译器实现有关的不确定行为。

## 语言特性

### 运行时模型

Jascal是编译式语言，以llvm为中间代码生成native code。

Jascal为强类型、安全类型语言，没有专门的指针类型，不允许其他类型与指针间的类型转换。Jascal运行时不包含垃圾回收器，程序员应自己负责使用删除对象运算符销毁不再使用的对象。

Jascal不负责数组边界检查和空指针、悬空指针检查，做出如上的无效访问可能导致与具体编译器实现和具体运行平台相关的不确定行为。

Jascal的函数调用规范由目标运行平台的ABI决定，函数调用传参中基本类型使用传值方式，对象类型使用传引用方式。

Jascal不负责运行时类型检查，但是支持RTTI

### 面向对象

Jascal是面向对象的编程语言，支持类、接口、继承、实现。

Jascal是严格面向对象的编程语言，所有函数必须为类的静态成员函数或成员函数。

Jascal类不支持静态初始化。

Jascal中所有成员函数都是虚函数。

### RTTI

Jascal支持RTTI，支持动态确定一个对象的实际类型，支持以某对象的实际类型动态创建新的对象。

Jascal的RTTI通过.class运算符实现，该运算符的返回值为一Class对象，实现了.getName()，.new()成员函数。但是没有Class类型，所以Class对象不能保存到变量或者以函数参数方式传递。

### 元组与多返回值

Jascal以内置元组类型的方式支持多返回值，元组必须是确定项数与每一项的类型的，元组支持赋值与自动解包，但不支持以下标方式索引。

### 占位符

为了方便元组解包，引入\_作为占位符，所有向\_的赋值都会被忽略，所以可以书写如下形式的元组赋值：

[[x,\_,y]]=[[3,4,5]] // x=3, y=5

### 数组模型

Jascal的数组支持自定义下标，且必须指定自定义下标。

Jascal的数组支持动态上下标，但是动态下标时上标必须是动态的。

Jascal的动态数组作为单独的类型，静态数组可以向有相同的静态下标的动态上标数组转换，但不能向静态下标不同或动态下标数组转换。

Jascal区分多维数组与嵌套数组。

Jascal的多维静态数组会按照最后一维优先的顺序展开，即

a[0..3,0..4]在内存中的布局为a[0,0]，a[0,1]，a[0,2]，……

Jascal的数组不支持初始化。

# 定义

## 字符

字符是指输入流中的单个字符，包括英文字符、数字、符号、空白字符、其他字符。

英文字符包括从a到z及从A到Z的英文字母，区分大小写。

letter::="a..z,A..Z"

数字包括从0到9的阿拉伯数字。

digit::="0..9"

空白字符包括制表符、空格、回车、换行。

white::="\t" | "\n" | "\r" | " "

符号包括在关键字中出现的除了英文字符与数字之外的字符。

其他字符包括Unicode规范中的可见字符除去英文字符、数字和符号的部分。

## 记号

记号是指按指定词法分割后的输入流的单元，每个记号由一个或多个空白字符组成。

## 关键字

关键字（或称保留字）是指具有特殊意义的、用于描述程序语义结构的记号

Jascal语言使用如下关键字：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| using | module | begin | end | class |
| interface | extend | implements | static | private |
| public | protected | const | int | short |
| byte | unsigned | char | if | then |
| else | while | do | for | repeat |
| until | case | of | break | continue |
| return | float | double | true | false |
| nil | new | delete |  |  |
| ( | ) | + | - | \* |
| / | \*\* | [ | ] | [[ |
| ]] | . | : | :: | % |
| < | <= | > | >= | == |
| != | <> | | | & | ^ |
| || | && | ^^ | << | >> |
| += | -= | \*= | /= | \*\*= |
| %= | |= | &= | ^= | <<= |
| >>= | = | ||= | &&= | ^^= |
| ~ |  |  |  |  |

## 标识符

标识符是指满足如下条件的记号：

1. 以英文字母或下划线开头
2. 剩余部分由英文字符、数字或下划线组成
3. 不与任何一个关键字完全相同

identifier ::= ( letter | "\_" ) { letter | digit | "\_" }

## 字面量

字面量是指直接书写在程序里的、用于表示一个固定值或字符串的记号。

# 运算与运算符

Jascal支持多种数学上定义的算术运算、位运算、比较运算与逻辑运算，以及程序语义上的运算等。

所有二元运算符a?b都有与赋值运算相结合的版本a?=b，语义上等同于a=a?b。

## 算术运算

Jascal中算术运算与运算符的对应关系如下表

|  |  |  |  |
| --- | --- | --- | --- |
| 算术运算 | Jascal运算符 | 样例 | 样例结果 |
| 加法 | + | 1+2 | 3 |
| 减法 | - | 3-2 | 1 |
| 乘法 | \* | 2\*3 | 6 |
| 除法 | / | 3/2 | 1 |
| 乘方 | \*\* | 2\*\*3 | 8 |
| 取模 | % | 15%4 | 3 |
| 自增 | ++ |  |  |
| 自减 | -- |  |  |

注： 1、两个操作数都是整数时除法为整除

2、取模运算只能用于两个整数之间

3、自增自减运算只能用于整数

## 位运算

Jascal中位运算与运算符的对应关系如下表

|  |  |  |  |
| --- | --- | --- | --- |
| 位运算 | Jascal运算符 | 样例 | 样例结果 |
| 二进制或 | | | 3|10 | 11 |
| 二进制且 | & | 3&5 | 1 |
| 二进制异或 | ^ | 3^5 | 6 |
| 二进制取反 | ~ | ~3 | -4 |
| 二进制左移 | << | 3<<2 | 12 |
| 二进制右移 | >> | 15>>2 | 3 |

## 比较运算

Jascal中比较运算与运算符的对应关系如下表

|  |  |  |  |
| --- | --- | --- | --- |
| 比较运算 | Jascal运算符 | 样例 | 样例结果 |
| 小于 | < | 1<2 | true |
| 小于等于 | <= | 1<=2 | true |
| 大于 | > | 1>2 | false |
| 大于等于 | >= | 1>=2 | false |
| 等于 | == | 1==2 | false |
| 不等于 | !=（或<>） | 1!=2 | true |

## 逻辑运算

Jascal中逻辑运算与运算符的对应关系如下表

|  |  |  |  |
| --- | --- | --- | --- |
| 逻辑运算 | Jascal运算符 | 样例 | 样例结果 |
| 逻辑或 | || | true||false | true |
| 逻辑且 | && | true&&false | false |
| 逻辑非 | ! | !true | false |
| 逻辑异或 | ^^ | true^^true | false |

## 程序语义运算

Jascal中包括如下用于程序语义的运算

|  |  |
| --- | --- |
| 运算 | Jascal运算符 |
| 赋值 | = |
| 访问成员 | . |
| 访问数组 | [] |
| 括号 | () |

## 优先级与结合性

Jascal中的运算遵从如下优先级与结合性

|  |  |
| --- | --- |
| 优先级 | Jascal运算符 |
| （高） | () |
|  | . |
|  | [] |
|  | ++、-- |
|  | ^= |
|  | &= |
|  | |= |
|  | ~ |
|  | ^ |
|  | & |
|  | | |
|  | <<、<<=、>>、>>= |
|  | \*\*= |
|  | \*=、/=、%= |
|  | +=、-= |
|  | \*\* |
|  | \*、/、% |
|  | +、- |
|  | <、>、<=、>=、!=、<> |
|  | ! |
|  | ^^= |
|  | &&= |
|  | ||= |
|  | ^^ |
|  | && |
|  | || |
| （低） | = |

除了=、^=、&=、|=等具有赋值效果的运算符为右结合外其他运算符均为左结合。

# 字面量

字面量包括字面数字、字面字符串与字面布尔值

字面数字包括字面整数与字面浮点数。

字面整数包括字面二进制整数、字面八进制整数、字面十进制整数、字面十六进制整数。

字面二进制整数为前部由0和1组成、最后一个字符为b的记号。

字面八进制整数为由0开始，剩余部分由0到7组成的记号。

字面十进制整数为+0、-0、由+或-开始的剩余部分由0到9组成的记号、由1到9开始的剩余部分由0到9组成的记号。

字面十六进制整数为由0x或0X开始的剩余部分由0到9和a到f和A到F组成的记号。

literal ::= literal\_int | literal\_float | literal\_double | literal\_string | literal\_bool

literal\_int ::=

[ "-" | "+" ] ( "0..9" { "0..9" } ) |

( "0" | ( "1..9" { "0..9" } ) ) |

( "0x" | "0X" ) ( "0..9,a..f,A..F" ) { "0..9,a..f,A..F" } |

"0,1" { "0,1" } "b"

literal\_float ::=

[ "-" | "+" ] "0" "." "0..9" { "0..9" } "f" |

[ "-" | "+" ] "." "0..9" { "0..9" } "f" |

[ "-" | "+" ] "1..9" { "0..9" } "." "0..9" { "0..9" } "f"

literal\_double ::=

[ "-" | "+" ] "0" "." "0..9" { "0..9" } |

[ "-" | "+" ] "." "0..9" { "0..9" } |

[ "-" | "+" ] "1..9" { "0..9" } "." "0..9" { "0..9" }

literal\_string ::= "\"" { character } "\""

literal\_bool ::= "true" | "false"

# 表达式

表达式由字面量、标识符与运算符组成，以上述规定的优先级与结合性进行运算，运算过程中出现错误则运算不会继续下去，但是已经执行了一半的中间计算的副作用不会消除。

所有的逻辑运算具有短路作用。

expression ::=

( expression

( "+" | "-" | "\*" | "/" | "\*\*" | "%"

| "<" | "<=" | ">" | ">=" | "==" | "!=" | "<>"

| "|" | "&" | "^"

| "||" | "&&" | "^^"

| "<<" | ">>"

| "+=" | "-=" | "\*=" | "/=" | "\*\*=" | "%="

| "|=" | "&=" | "^="

| "<<=" | ">>="

| "=" )

expression ) |

"(" expression ")" |

( "!" | "~" | "++" | "--" | "-" ) expression |

expression ( "++" | "--" ) |

identifier |

class\_name "::" identifier |

class\_name "." identifier |

function\_call |

literal |

"new" class\_name "(" [ expression { "," expression } ] ")" |

"new" type\_name { array\_specifier } "[" expression { "," expression } "]" |

"delete" expression

# 语句

Jascal中的语句分为表达式语句、控制流语句和块语句。

## 表达式语句

表达式可以单独构成一个语句，整个表达式的最终结果将被直接丢弃。

## 控制流语句

控制流语句用于实现分支、循环、函数调用与返回

### if语句

if语句用于实现分支，由if关键字，(关键字，分支条件表达式，)关键字，then关键字，条件成立时执行的语句及可选的else关键字和条件不成立时执行的语句组成。

if语句要求分支条件必须为类型为bool的表达式。

if语句首先将分支条件表达式求值，依据条件成立与否跳转到条件成立语句或条件不成立语句，如果条件不成立且未指定条件不成立语句，则直接跳转到if语句的下一条语句。条件成立语句或条件不成立语句执行结束后总会跳到if语句的下一条语句。

if\_statement ::=

"if" "(" expression ")" "then" statement\_terminator |

"if" "(" expression ")" "then" statement "else" statement\_terminator

### while语句

while语句用于实现循环，由while关键字，(关键字，循环条件表达式，)关键字，do关键字和循环体语句组成。

while语句要求循环条件表达式必须为类型为bool的表达式。

while语句首先将循环条件表达式求值，如果条件成立则跳转到循环体语句，循环体执行结束后跳回while语句重新求值；如果条件不成立则跳转到while语句的下一条语句。

while\_statement ::= "while" "(" expression ")" "do" statement

while\_statement\_terminator ::= "while" "(" expression ")" "do" statement\_terminator

### for语句

for语句用于实现循环，由for关键字，(关键字，初始化表达式或初始化变量声明，;关键字，循环条件表达式，;关键字，循环更新表达式，)关键字，do关键字和循环体语句组成。

for语句要求循环条件表达式必须为类型为bool的表达式。

for语句的执行流程如下：

1. 执行初始化表达式，或按照初始化变量声明的语义定义新变量，新的变量的作用域仅限于for语句中
2. 将循环条件表达式求值，如果条件不成立则跳转到for语句的下一条语句
3. 执行循环体语句
4. 将循环更新表达式求值，表达式的最终结果被丢弃
5. 跳转到2)

在循环体语句中可以使用break语句终止循环或使用continue语句直接跳转到4)

for\_statement ::= "for" "(" [ variable\_defination | expression ] ";" [ expression ] ";"

[ expression ] ")" "do" statement

for\_statement\_terminator ::= "for" "(" [ variable\_defination | expression ] ";"

[ expression ] ";" [ expression ] ")" "do" statement\_terminator

### repeat语句

repeat语句用于实现循环，由repeat关键字，循环体语句列表，until关键字，(关键字，循环终止条件表达式，)关键字组成。

repeat语句首先执行循环体，然后将循环终止条件表达式求值，如果条件成立则跳转到repeat语句的下一句，否则再次执行循环体并检查循环终止条件。

repeat\_statement ::= "repeat" statement\_list "until" "(" expression ")"

### case语句