Atitit 编程语言语言规范总结

目录

[1. 语言规范 3](#_Toc2933)

[2. Types 3](#_Toc27461)

[2.1.1. Primitive types 3](#_Toc2010)

[2.1.2. Compound types 4](#_Toc23033)

[3. Statements 5](#_Toc6197)

[3.1.1. Empty statement 5](#_Toc31529)

[3.1.2. Labeled statement 6](#_Toc1799)

[3.1.3. Expression statement 8](#_Toc23601)

[3.1.4. If statement 8](#_Toc28053)

[3.1.5. Assert statement 8](#_Toc21814)

[3.1.6. Switch statement 8](#_Toc8089)

[3.1.7. While statement 10](#_Toc32395)

[3.1.8. Do statement 11](#_Toc8602)

[3.1.9. For statement 12](#_Toc18988)

[3.1.10. For-in statement 12](#_Toc26345)

[3.1.11. For-of statement 13](#_Toc10825)

[3.1.12. For-range statement 14](#_Toc15504)

[3.1.13. Break statement 15](#_Toc13560)

[3.1.14. Continue statement 16](#_Toc12473)

[3.1.15. Return statement 19](#_Toc30330)

[3.1.16. Throw statement 19](#_Toc601)

[3.1.17. Goto statement 20](#_Toc28355)

[4. Operators操作符 22](#_Toc17187)

[4.1.1. Unary operators 22](#_Toc23583)

[4.1.2. Binary operators 22](#_Toc26118)

[4.1.3. Ternary operators 23](#_Toc17454)

[5. Expressions表达式 24](#_Toc1743)

[5.1.1. Variable expression 24](#_Toc30093)

[5.1.2. Arithmetic expression 24](#_Toc23460)

[5.1.3. Relational expression 24](#_Toc20444)

[5.1.4. Function expression 24](#_Toc1897)

[5.1.5. Index expression 25](#_Toc19730)

[6. Scoping作用域 25](#_Toc31832)

[6.1.1. Expression scope 25](#_Toc24114)

[6.1.2. Block scope 25](#_Toc22610)

[6.1.3. Function scope 26](#_Toc32475)

[6.1.4. File scope 27](#_Toc12463)

[6.1.5. Module scope 27](#_Toc15962)

[6.1.6. Global scope 27](#_Toc12694)

[6.1.7. Packages 27](#_Toc12821)

# 语言规范

在语言学里有三个基本概念(也是三个分支)， ****syntax**** , ****semantics**** , ****pragmatics**** . 即语法，语义(编译结果)和语用(最佳实践, 标准库, 生态)。这里主要讲语法层面。

# Types

一门语言的规范，首先是类型(type), 声明(statements), 表达式(expressions)等， 然后是作用域(scoping)。前面的内容介绍了类型系统, 那么该类型系统定义了哪些类型，实现了哪些特性是我们首先要了解的。通常一门语言的语法规范会以grammar [6] 的形式来定义，例如golang中对于浮点数字的描述:

float\_lit = decimals "." [ decimals ] [ exponent ] |

decimals exponent |

"." decimals [ exponent ] .

decimals = decimal\_digit { decimal\_digit } .

exponent = ( "e" | "E" ) [ "+" | "-" ] decimals .

这段文本描述了浮点数字的书写规则，也是一棵语法树。 *grammar* 对于没有学过的人来说有些难懂，所以本文中用类型实例的方式表示。

### **Primitive types**

即基础类型, 或者叫内置(builtin)类型, 是程序以及 [复合类型](https://www.tuicool.com/choose-a-programming-language/zh.wikipedia.org/wiki/%E8%A4%87%E5%90%88%E9%A1%9E%E5%9E%8B) 的创建基础。



### **Compound types**

即复合类型， 也叫\*Composite type\*s. 复合类型可以由基础类型和复合类型所构成。



# Statements

statement的直译是声明，但在这里按照代码的逻辑单元来理解，一个statement是逻辑单元的开始或结束。在书籍中，通常描述为xxx语句，比如 ****if**** 语句。

### **Empty statement**

空语句, 不做任何事情。grammar规则:

EmptyStatement:

;

### **Labeled statement**

标签语句, 通常作为 ****goto**** , ****break**** , ****continue**** 的目标，例如c/c++里的 ****goto**** 语句，一个单词加上冒号即是 *labeled statement* 。下面代码中的标签语句 *LOOP:* 即是 ****goto**** 的目标。

/\*c\*/

return

它的grammar规则为:

LabeledStmt = Label ":" Statement .

Label = identifier .

*labeled statement* 作为 ****goto**** 的目标大家应该见的很多，这里再举一个作为 ****continue**** 的目标的例子，便于理解。

// golang

}

### **Expression statement**

某些 *expression* (表达式), 比如赋值，函数调用等，可以作为一个 *statement* ，称之为 *expression statement*

### **If statement**

我们常说的 ****if**** 条件语句，也叫 ****If-then-else**** , 如果条件满足则执行此逻辑, 否则执行它的 ****else****(如果存在)逻辑.

// golang

if x > max {

x = max

}

### **Assert statement**

断言语句， ****assert**** 加真假值表达式, 如果表达式结果为 *false* , 程序退出。

// c

assert( size <= LIMIT );

不是所有语言都有断言语句.

### **Switch statement**

switch条件语句，判断表达式的值，满足不同的条件执行时执行不同的逻辑, 当所有条件都不满足时，执行默认逻辑(如果存在).

}

对于golang, *switch statement* 分成了两类, 一类是常规的 *expression switch* (上面的例子), 一类是 *type switch* (下面的例子)

// golang type switch

}

### **While statement**

while循环语句, 重复判断bool表达式的值，如果为真则执行，直到表达式的值为假

}

tip: 在golang里， ****for**** 语句加上bool表达式，可以实现 ****while**** 语句

}

### **Do statement**

do语句，和 ****while**** 配合使用，先执行一次逻辑，再判断条件

// c

### **For statement**

for语句, 一般由三部分组成，初始化表达式，条件表达式和一个末尾表达式(post statement)，初始化表达式只在开始时执行一次，条件表达式用来判断本次执行是否退出，末尾表达式在每次 ****for**** 语句的代码逻辑执行完后都会执行。

### **For-in statement**

for语句常用来迭代，于是出现了多个变种， ****for-in**** 常见于脚本语言，如 *TypeScipt* ， *Groovy* , 用于迭代可枚举的(enumerable types)数据类型

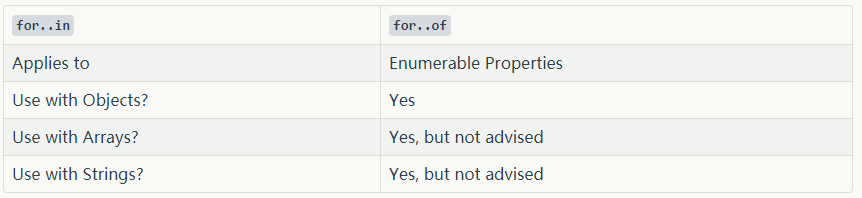
这里，x被赋值为person这个map的key。当遍历的对象是map时，x为key，当遍历的对象是数组时，x为索引下标

### **For-of statement**

for语句变种, 类似于 ****for-in**** , 用来迭代可迭代的(iterable)数据类型, 比如数组和字符串

这里变量v被赋值为foo函数返回的对象。被赋值的变量总是可迭代类型里的元素。

For-invs ****For-of****



### **For-range statement**

for语句变种，在golang中用来迭代数组, 或者map

### **Break statement**

跳出语句，用于立即跳出一个逻辑单元，当不配合 *labeled statement* 使用时，立即(abruptly)跳出最里层的一个封闭(enclosing)逻辑单元, 如 ****switch**** , ****do**** , ****while**** , ****for**** . 当配合 *labeled statement*使用时，立即跳出label标定的层级的封闭逻辑单元。

// c

*labeled break* 已在 *labeled statement* 中说明，这里不再赘述.

### **Continue statement**

continue语句，立即结束当前层级的逻辑，跳转到 ****for**** 循环的末尾表达式，开始下一次循环.

continue和 *labeled statement* 配合使用时，不仅会结束当前层级的逻辑，还会跳转到 *label* 标签指定的位置。我们看下下面的代码逻辑.

当column等于4时，结束逻辑，此时不是跳转到当前层级的 *post statement* , 而是跳转到RawLoop, 所以输出结果应该为:

1-2

3-2

5-2

不仅4没有输出，6也被 ****continue**** 掉了.

### **Return statement**

return语句跳出当前函数，回到函数的调用方, 同时将一个或者多个返回值传给调用方。本应出现在最后一行的 ****return**** 语句，在没有返回值的情况下，可以省略。 *Groovy* 语言的 ****return**** 语句是可选的。

### **Throw statement**

在一些语言中比如 *Java* , *JavaScript* 等使用 ****throw**** 语句来抛出错误，以便上层的调用方能够通过 ****try-catch-throw**** 的方式捕捉并处理。未捕捉的 ****throw**** 语句会导致线程/进程终止。对于 *Java* ,****throw**** 的的对象必须是 *Exception* 或者其子类，对于 *JavaScript* , throw的对象可以是任意类型.

### **Goto statement**

goto语句和 *labeled statement* 配合使用，用于逻辑跳转，程序执行流程会直接跳转到标签处. *goto statement* 只有部分语言提供，而且写法也有不同，比如对于 *golang* , *labeled statement* 必须在 *goto statement* 之前, 而 *C* 语言则无此限制。

以上都是常见的语句，除此之外，语言也会有实现一些非常规的语句，比如 *golang* 的 ****defer**** 语句。

# Operators操作符

在介绍表达式之前，先介绍它的组成元素之一，操作符。其中，操作符分为一元(unary)操作符，二元(binary)操作符和三元操作符. 优先级决定了在多个操作符同时出现时，先使用哪个来求值。数字越大，优先级越高。

### **Unary operators**

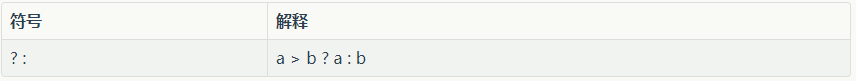


### **Binary operators**



### **Ternary operators**

在计算机中也叫条件运算符(conditional operator)



# **Expressions**表达式

在编程语言里一个表达式(expression [7] )是由一个或多个常量，变量，操作符或函数组成。通过对表达式求值(evaluate)来得到一个新的值，这个新的值可以是基础类型，也可以是复合类型。表达式在运算时，会进行类型推断和类型检查。从之前讲的内容可以看出， *statements* 的作用主要是控制代码逻辑的执行顺序， *expressions* 则是具体的代码逻辑.

### **Variable expression**

// golang

c := a + b

### **Arithmetic expression**

2 + 3

### **Relational expression**

3 != 4

### **Function expression**

// golang

v := f(1)

### **Index expression**

a[x]

表达式并没有严格的分类，各个语言也不尽相同，这里仅列举了部分例子来说明。

# **Scoping**作用域

即作用域, 作用域是名称(比如变量的声明)和其实体(entity, 比如变量的定义)的绑定规则。作用域约束了实体的作用范围，保证程序是无歧义的。

### **Expression scope**

实体仅在表达式内可用。

// c

({ int x = f(); x \* x; })

临时变量 *x* 接受函数的返回值并平方，这样避免两次调用函数 *f* .

### **Block scope**

通常编程语言都会使用花括号 {} 来将代码包裹成块(block), 在block内声明的实体，仅在block内有效。

### **Function scope**

在函数内声明的实体，仅在函数内有效。

为了不和 *block scope* 混淆，这里用 *python* 的例子。

### **File scope**

在代码文件内声明的全局变量，仅在当前文件内有效。

### **Module scope**

在某些现代语言中，一个实体可以在一个模块内的各个文件内有效，比如 *golang* . 部分语言，一个文件就是一个独立的module，此时，也属于 *file scope* .

### **Global scope**

在所有模块，所有文件内都有效的实体称为全局实体，此类作用域属于 *global scope* . 在编程实践当中，应尽量避免使用。

### **Packages**

一个复杂程序一般会由多个包(或者叫模块)组成, 这种机制能让程序的结构和逻辑更加清晰可读，提高代码的复用能力，也可以借助 *module scope* 来避免同名之间的冲突。这里仅列举几种常见的包引入方式。

// Java 包名是一个层级结构

顺便说下python里 *from* 和常规 *import* 之间，在包的的使用上会有差异。