# 语法

**实际使用时，请用“e”代替“exp”关键字，请用“re”代替“return”关键字**

## 函数

对于COOL而言，函数是其最重要的部分。此节首先提供了一些简单代码以便读者熟悉COOL函数的风格，然后介绍了返回表达式的函数与返回运算值的函数、附加权重的函数、正向函数与逆向函数、函数权重、函数逆推。这些构成了COOL的推理框架。

## 定义函数

在COOL中，函数由函数声明和函数体两部分组成，一个表示相加的函数可以用如代码1的方式进行定义：

|  |
| --- |
| **代码 1** 函数声明示例 |
|  |
|  |
|  |

其中“@”用以修饰后面的为函数声明而非函数调用。调用函数时需要去掉“@”：

|  |
| --- |
| **代码2** 函数调用示例 |
|  |

函数在调用时会优先传入实际参数的引用而非复制，例如代码3：

|  |
| --- |
| **代码3** 函数在调用时会优先传入实际参数的引用 |
|  |
|  |
|  |
|  |

代码3执行结束后值增加1。在该段代码中，允许函数参数嵌入函数名字（谓语）中，以增强表达能力。

此外，COOL提供了一些无需定义即可使用的内置函数，用于提供基础的数学运算功能。

## 返回表达式的函数与返回运算值的函数

通过在函数声明加上属性“exp”来表明此函数的返回是一个表达式。这种函数用来表述表达式的变换规则，用户不能直接调用返回表达式函数。

例如，代码4通过函数描述乘法分配律的逆运算：

|  |
| --- |
| **代码4** 返回表达式的函数示例 |
|  |
|  |
|  |

其中，紧跟“@”的一对大括号及其内的表达式被称为函数声明作用域，内部的唯一表达式被称为函数声明表达式。其实2.1.1节的函数示例中函数名称也位于其函数声明作用域中，但是作用域边界的大括号为书写方便省略掉了。

如果函数声明没有属性“exp”则此函数返回的是运算值。用户只能直接调用返回运算值的函数。

## 正向函数与逆向函数

正向函数与逆向函数均针对返回运算值的函数而言；返回表达式的函数不具备此属性。

正向函数，即所有函数参数均确定，函数返回值待定的函数。正向函数的执行过程即利用输入参数推导函数的返回值，例如代码1。

逆向函数，即函数返回值确定，函数输入参数中存在待定参数的函数。逆向函数的执行过程即利用函数的返回值与确定的输入参数计算待定的输入参数。

例如，代码5提供了计算二次函数一个解所需的逆向函数：

|  |
| --- |
| **代码5** 逆向函数示例 |
|  |
|  |
|  |
|  |
|  |
|  |

其中，“$”符号作用于参数x，表明参数x在该表达式中待定，对于一个表达式中多次出现的变量，只需要使用“$”修饰一次即可。函数体中出现的变量“ans”代表的函数的返回值，为已知参数，可以不使用。

用户调用逆向函数时，需要用“$”修饰所要推导的待定变量，例如求解以a为未知数的一元二次方程（代码6）：

|  |
| --- |
| **代码6** 调用逆向函数代码示例 |
|  |

需要将a用“$”修饰。

## 函数权重

函数权重被用于控制推理系统的推理方向。用户可以为那些更容易导向正确推理结果的变换（即函数）赋予更大的权重，不容易导向正确推理结果的变换赋予更小的权重。函数的权重在函数声明时确定，位于“@”与函数声明作用域之间，例如代码7中定义了一个权重为10的函数：

|  |
| --- |
| **代码7** 具有权重的函数示例 |
|  |
|  |
|  |

不赋予权重的函数（如代码1）其权重为0。

## 函数逆推

函数逆推是一种通过正向函数定义逆向函数的特殊方式。例如代码8：

|  |
| --- |
| **代码8** 利用正向函数定义逆向函数的示例 |
|  |
|  |
|  |

其中“=>”表示推导。在本文实现此功能的算法中使用这种方法定义逆向函数的必要条件是：逆向函数的函数声明中的所有参数与正向函数的函数声明中的所有参数的名称必须一一对应；逆向函数中的待定参数及依赖此参数的变量只参与正向函数的线性结构，不参与函数体的循环、分支结构；正向函数的函数体代码不修改域外参数；正向函数的函数体中不存在同名不同作用域的变量。由于是必要条件，这也意味着即使满足这些要求也有可能无法完成逆推。

## 变量

## 变量的声明与类型

COOL的非临时变量在使用前必须被声明。声明变量a，并赋初值1的示例如代码9：

|  |
| --- |
| **代码9**变量声明示例 |
|  |

变量的类型由最近一次所赋值的类型决定。COOL支持的基础数据类型为浮点数和字符串；当不对变量赋值时，其类型默认为浮点数。

## 变量的访问

变量从其被声明处至其所在作用域结束处之间的代码段中均可被访问。当一个表达式需要使用某个变量时，其优先使用当前作用域的变量。用户可以通过使用“out”修饰此变量以强行使表达式使用上层作用域的变量。例如代码10：

|  |
| --- |
| **代码10**“out”使用示例 |
|  |
|  |
|  |
|  |
|  |

其计算结果（内层的最终值）为2。

当在函数声明作用域中使用“out”修饰参数时，此参数将不再是形式参数，而是函数声明作用域外的实际参数。例如代码11的函数声明中的：

|  |
| --- |
| **代码11**函数声明中的“out”使用示例 |
|  |
|  |
|  |
|  |
|  |

## 条件语句

## 循环

COOL支持while循环结构，示例如代码12：

|  |
| --- |
| **代码12** while循环示例 |
|  |
|  |
|  |

## 分支

COOL的分支结构，示例如代码13：

|  |
| --- |
| **代码13**分支结构示例 |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

## 注释语句

注释风格与C语言相同，如代码14：

|  |
| --- |
| **代码14**注释示例 |
|  |
|  |

## 完整代码示例1

求解以下数学问题：

对两个数字量，依次进行以下操作：

1、将，求和，结果记为；

2、修改的值，使其满足的值等于；

3、求解变量，满足等于;

4、对，，求和，得到最终结果；

已知第四步得到的结果为，且的初始值为3，则的初始值为多少？

求解问题的完整的COOL代码如代码15，涉及了返回表达式的函数与返回运算值的函数、正向函数与逆向函数、函数权重、函数逆推。

|  |
| --- |
| **代码15**完整代码示例1 |
|  |
|  |
|  |
|  |
| *定义加法交换律，函数声明被属性exp修饰，表明此函数是返回表达式的函数，对于返回表达式的函数，其函数参数a、b被“#”修饰，表明无论a、b是否待定，均可应用加法交换律；* |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| *“-->”表示输出，意为将x值以默认方式输出* |

## 类

解决相似问题的规则可以单独封装成类，通过继承用户可以更灵活地复用、修改和拓展规则，实现对复杂问题的分治处理和程序的模块化开发。

## 定义类

在COOL中，类由声明和类的作用域（下文称为“类体”）组成，如代码16：

|  |
| --- |
| **代码16**类示例 |
|  |
|  |
|  |

其中，“system”是声明类的关键字；“OperationLaw”是类的名字。

## 继承

在定义一个类时，可以使其继承其他的类以使用其成员函数和变量：

|  |
| --- |
| **代码17**类继承示例 |
| *,* |
|  |
|  |

其中，“<<”表示继承，当一个类继承多个类时，多个被继承的类的名称之间用逗号隔开。若当前类中存在与父类同名的变量或声明相同的函数，默认优先使用当前类的成员变量或函数；若多个父类存在相同的成员，则优先使用声明时位于左侧的类的成员（例如，若类与类有相同的成员，则优先使用的成员）；如果指定了类的名称则采用指定类的成员函数。

## 类实例的初始化

类的实例也属于变量，其声明方式如代码18：

|  |
| --- |
| **代码18**类实例声明示例 |
|  |

其初始化类似函数调用，在进入类的作用域后创建活动记录并依次执行类作用域内的代码，但是在离开类的作用域时不会销毁活动记录，而是将活动记录作为对应变量的值。

## 成员访问

通过“.”运算符访问成员变量或调用成员函数，如代码19：

|  |
| --- |
| **代码19**成员访问示例 |
|  |
|  |

## 完整代码示例2

代码20展示了COOL的类的综合使用。

首先定义了两个类，类包含了一些常用运算的变换规则，类包含了求解一元二次方程的一个公式，然后令主程序类继承先前定义的两个类，并在程序中进行定义两个函数求解一元二次方程和对成员变量进行修改：

|  |
| --- |
| **代码20**完整代码示例2 |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| *定义解二次方程的类* |
|  |
|  |
|  |
|  |
|  |
| *“<<”表示继承，主类继承运算律类和一元二次方程求解公式类，其可以同时访问两种类中的成员* |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |