# mini-Python3解释器文档



# 抽象语法树部分

## 概述

本解释器使用的抽象语法树是由c++标准库构建的，其中为了函数调用的方便，使用了c++17标准的一些新特性（即std::invoke)，此外没有依赖。

由于程序的解析能力和抽象语法树的实现密切相关，也就是说本解释器所能运行的代码只能在抽象语法树范围内。而本项目的语法树只实现了一个面向过程语言的功能，即抛弃了python中面向对象、异步等高级功能。

从这一方面来说，这些数据结构可以被用在任何一个编程语言上。不过对于python3，语法树预置了一些内置函数，从而更适配这门语言。

## 实现的功能

这颗抽象语法树能进行如下操作：

- 定义变量
- 运算，包括算术、移位、逻辑运算、比较运算
- 定义函数
- 删除变量（delete关键字）
- 分支语句（if-else，elif被展开而不是单独实现）
- 基本的两种容器：tuple和list
- 循环（for和while）

此外有一下特性：

- 垃圾回收
- 初步类型错误检测

内置函数只有两个：

- print
- range

他们是用c++的特性实现的

## API

对使用者而言，只需要使用ast的两种数据结构：表达式和语句。这两个类具备运行时多态的能力。

由于在定义节点时需要提供子节点，因此，语法分析器生成抽象语法树的过程应该是自底向上的。

表达式分为(类名)：

- UnaryOperation 单元运算
- BinaryOperation 双元运算
- CompareOperation 比较
- BooleanOperation 逻辑运算
- FunctionCall 函数调用
- Slice 切片
- Name 自定义标识符
- Number 整数或浮点数字面量
- String 字符串字面量
- NameConstant 包括：True False None () [] 后两个是空元组和空列表
- Formatted_String 格式化字符串

语句分为（类名）：

- Pass_Statement 空语句
- Delete_Statement 删除语句
- Assign_Statement 赋值语句
- AugAssign_Statement 带有运算符的赋值语句，比如a+=1
- FunctionDefinition_Statement 函数定义
- Expression_Statement 如果表达式被计算，但其结果没有被使用，则属于此表达式语句
- Suite 语句块
- If_Statement 分支语句
- While_Statement 循环语句
- For_Statement 带有容器范围的循环语句
- Break_Statement 产生break信号
- Continue_Statement 产生continue信号
- Return_Statement 产生return信号，并返回指定的返回值

只使用表达式和语句两类结构就可以完成ast的构建。ast运行过程中使用的额外结构是隐藏的。

要创建这些结构，使用在AST.h中定义的宏CREATE。这个宏实际上是make_shared的别名。ast内部的各个节点，必须使用智能指针相连，只也是其效率为何如此低下的原因。需要注意，不能在任何时候使用原生指针，否则可能会出现意想不到的内存泄漏，或者双重delete。

为创建函数调用时参数列表的方便，宏ArgList同时被定义。

要让节点附着到ast上，需要调用factory.addStatement( )。其中factory是对astfactory单例类静态方法调用的宏。

## 实现细节

ast的数据结构部分并不复杂，实际上上述列表已经指出了ast的大概结构，通过继承实现。这有点类似于模板方法，在数量不多的时候勉强可用。

ast在运行的时候会需要一些额外的结构的辅助，其一是函数类Function，它是内部类，由函数定义类创建、函数调用类调用，不被外部所见。函数类由函数名称、一系列参数和函数体组成，它的实现比较粗暴：每一次调用函数，就创建一个上下文（见下文），在其中手动把参数赋值为传入的值，这也意味着这里的传参数都是按值传参数，不能按引用传参数。在函数结束的时候，把本地上下文删除。

这些过程都是由函数调用类自动完成的（压栈等）。其中使用的第二个重要单例类：astfactory，虽然叫它factory，但这只是历史上的原因（名称已经固定），实际上它是ast本身的抽象，有点像是pool。它存储所有将要被运行的节点，其方法run会实际上运行这个程序。在此之前的操作是由语法分析器完成的，在此之后则是语法树本身的完成。它会动态地创建删除变量、定义调用函数。

其内部维护了函数的列表，上下文的列表，变量表等。每一个都对应相应的语句。上下文是很重要的，它是一个stl栈，存储了函数调用关系。每一个上下文对应一张变量表，变量表是变量名到其值（见下文）的stl map。由上所说，函数调用会创建一个临时的上下文。

需要说明的是，ast在其内部抽象了变量表，其对外的变量创建、赋值、删除方法是和上下文无关的。规则是：如果当前上下文的变量表不支持操作，则会转向global变量表，如果也不支持，则报错。这样做的好处是简化了赋值、创建、删除的语句实现。

第三个重要的类是returnvalue，这是一个所有类都用到的工具类，它是一系列值的集合，代表了返回值（或者说任何一个可能的值），它的类型是动态的，可以是数、字符串、容器、布尔值等，取决于调用者调用的构造函数。虽然这么做很浪费资源，但确保了python的动态类型。returnvalue作为返回值被嵌入ast基类astnode的虚函数exec（）中，当ast运行的时候，exec（）总会返回returnvalue，从而在树的内部保证动态。