**技术文档**

**撰写者：罗溥晗 2019/3/1**

**1.编译部分**

1.1概述

编译部分将会按行读取源代码，逐行解析代码并且最后返回给执行部分一个语句列表。这个语句数组里的每个元素代表了一行代码执行时所必需的信息以及执行完的跳转信息。

1.2语句列表

该列表的定义是ArrayList<Statement\_AST>，列表中元素的类型是Statement\_AST，将在下方定义。列表的第一项置为空，这样可以使列表的下标和代码的行数直接对应。

1.3语句AST

语句AST是一个类，类名为Statement\_AST。属性和方法如下方代码所示。Pythy中有四种语句，分别是赋值语句（y=x+1），分支语句（if），循环语句（while）和空语句。并不是所有语句都用到了所有属性。其中type、line和next是共有属性。type表明该语句的类型，line指示了当前语句位于代码的哪一行，next指示了下一行的行号。空语句只有共有属性。赋值语句有left和right两个属性，分别代表赋值语句的左值和右值，其中语句的左边一定是一个变量名，右边则是一个表达式（将在1.4定义）。分支和循环语句有in和out两个属性，这是因为这两条语句会打乱程序的顺序执行，开启一段特殊的程序段。in属性表示该程序段的开始行号（通常就是line+1，也就是next），out属性表示如果分支或循环条件不成立，程序应当跳转到的行号（无法在解析这条语句时直接确定out的值，将在1.5节讨论）。语句AST还有一个copy方法，表示将自己复制一遍并返回。

class Statement\_AST

{

String type;//= if while null......

String left;

Expression\_AST right;

int line;

int next;

int in;

int out;

Statement\_AST copy()

{

Statement\_AST y = new Statement\_AST();

y.type = this.type;

y.left = this.left;

y.right = this.right;

y.line = this.line;

y.next = this.next;

y.in = this.in;

y.out = this.out;

return y;

}

}

1.4表达式AST

表达式AST是一个类，类名为Expression\_AST。属性和方法如下方代码所示。Pythy中有八种表达式，用公共属性type区分。四种运算表达式——加减乘除，和四种常变量表达式（我们认为一个常变量也是一种特殊的表达式）——变量（name），整型（int），浮点型（float）和布尔型（boolean）。和语句AST一样，并不是所有表达式都用到了所有属性。除了公共属性之外，运算表达式只用到了left和right两个属性，分别代表运算的左值和右值（采用最左推导，即始终将最左单项展开为常变量）。常变量表达式用到了type属性以及一个和type对应的value属性。表达式AST的copy方法同1.3。

class Expression\_AST

{

String type;//name int float boolean + - \* /......

Expression\_AST left;

Expression\_AST right;

int value\_int;

float value\_float;

boolean value\_boolean;

String value\_name;

Expression\_AST copy()

{

Expression\_AST y = new Expression\_AST()

y.type = this.type;

y.left = this.left;

y.right = this.right;

y.value\_int = this.value\_int;

y.value\_float = this.value\_float;

y.value\_boolean = this.value\_boolean;

y.value\_name = this.value\_name;

return y;

}

}

1.5处理缩进

不同于C、JAVA等语言使用大括号区分层级，Pythy采用的是tab缩进来区分层级。我们认为该行首有几个tab缩进，该行就处于第几级（设为level方便后续讨论）。在处理缩进时，我们首先要引入一些全局变量。current表示当前行的level，last表示上一行的level。在1.3节时我们提到，语句的out值无法在解析当前语句时确定，这是因为我们不知道if或while结束在什么地方。因此我们还需要引入一个栈暂时存放尚未填入out值的语句。如果发现了if和where语句，则压栈，并且此时我们还能估计出下一句level的上界是当前level+1，如果实际下一句level比上界还要大了，则报错。如果发现了level下降，则出栈并将出栈语句的out设置为当前行号。需要注意的是，level可能不止一次减1，也可能下降很多，因此我们规定下降几级就退栈几次（退栈次数=last-current）。同时，如果读到文件结束符，则认为该处level=1，全部出栈。