Jmm语言编译器设计与实现

第三次迭代报告

沈斯杰 5130379036

朱维之 5130379043

刘 洋 5130379049

本次迭代实现了我们语音的后端，包括语义分析以及中间代码的生成。

1. 自动化工具

我们的编译器的词法分析和语法分析所借助的自动化工具是javacc，所有的词法、语法规则都写在了.jjt的文件中。.jjt文件属于.jj的脚本，它能够支持更多辅助命令。通过.jjt文件生成.jj文件，然后就能够生成我们所需要的词法、语法分析器以及所有节点的类。

我们所有这次所有的脚本都写在jmmCompiler/lexical/Scanner.jjt中，这个文件内容分为若干部分。第一部分，options选项，做一些设置，基本上用的都是默认设置。第二部分，在PARSER\_BEGIN和PARSER\_END之间，这是我们生成的分析器类的一些附加的函数，这个在之后会详细描述。第三部分，词法规则的描述，分为SKIP和TOKEN。第四部分，是语法描述，如何构建抽象语法树。

编译器的中间代码，选择的是产生LLVM使用的中间代码，由于在JAVA上，LLVM不提供中间代码开发的接口，所以只能通过走AST进行硬编码，一行一行输出中间代码。

1. 编译器前端的实现

主要结合.jjt文件中的第二、第三、第四部分描述。

在Scanner类中，我们需要两个函数，一个是词法分析函数scan，它的输入是一个输入流，输出是一个字符串。输出格式是，所有的常量、运算符等符号、数字，都是直接用引号引起来，如“123”、“int”、“{”、“；”……所有的标示符，都用<IDENTIFIER, \*\*\*>表示，其中\*\*\*表示该标示符，如<IDENTIFIER, abc>。

另一个是语法分析函数parse，它的输入依然是一个输入流，输出是AST的根节点Node。Node类其实是一个接口，它有许多实现类。针对不同的节点。这里的语法规则与之前设计文档中的描述基本一致，有一些语法有略微改动，但是整体不影响语言特性。

在这两个函数中，首先判断Scanner（静态类）是否已经被初始化过，如果未初始化，我们就用new来初始化，否则，就用Reinit方法重新初始化。这两个函数在分析出错时还能够报出错误信息。

第二部分的词法规则中，分成SKIP(空白符)、LITERAL、KEYWORDS、OPERATORS、IDENTIFIER这五大类，分别与之前的设计文档所对应。

第三部分的语法规则中，按照之前的设计，逐个节点进行分析，其中LOOKAHEAD将会向前看1个或者若干个TOKEN。在所有的节点中，有些节点是不必产生的，用#void表示，有些节点是按照子节点的情况，有条件地产生的。

1. 编译器后端的实现
2. **语义分析**
   1. 类型检查

1.1.1 判定每一个表达式的声明类型（生成符号表）

* + 1. 判定每一个字段、形式参数、变量声明的类型（生成函数的符号表）
    2. 判断每一次赋值、传参数时，是否存在合法的隐式类型转换

支持的隐式转换：int/double/char之间转化，子类、父类之间的转化

1.1.4 判断一元和二元运算符左右两侧的类型是否合法

* 1. 与类有关的检查

1.2.1类中的成员的访问权限

* + 1. 函数的继承和覆盖
    2. 主类中主函数的检查

1. **中间代码生成**

从AST到LLVM中间代码生成

1. **外部支持**

需要LLVM提供的printf函数

1. 图形界面

