How to Start Writing a Compiler?

赖睿航

2020.3.28

Big Picture

- Parse
- Semantic Check
- Build IR
- Optimize…
- Code Generate…

Parse

- 拿到一段源代码之后，通过 Parser 将源代码转换为 Concrete Syntax Tree(CST)
- 你应该会用到 ANTLR（或者像 mgg 一样手写 Parser）

What is ANTLR?

- ANother Tool for Language Recognition
- “ANTLR 是一款强大的语法分析器生成工具。”

——《ANTLR 4 权威指南》

- 简单来说，就是个 Parser

How to Use ANTLR?

- 你只需要做一件事情：写一个 g4 文件
- 然后生成与这个 g4 相匹配的 ANTLR recognizer

What do you get when the code is parsed?

- 你得到了一棵 CST（Concrete Syntax Tree）
- 可以通过 IDEA 的 ANTLR 插件来看一看这棵 CST 的样子（这只需要用到 g4 文件）

Build AST

- AST 实际上是 CST 去掉了一些无用的信息
- 遍历 Parser 给你生成的 CST，在这个过程中建立起 AST
- 首先要想好你的 AST 里，有哪些类型的节点，每个节点包含哪些成员，把这些类设计出来
- 然后写一个 ASTBuilder，遍历 CST

Visitor and Listener

- ANTLR 的运行库提供了两种遍历树的机制：监听器(Listener)和访问器(Visitor)
- 如果你使用 Listener，你的 ASTBuilder 需要继承 BaseListener 类
- 如果你使用 Visitor，你的 ASTBuilder 需要继承 BaseVisitor 类

- Listener 提供 enter*** 和 exit*** 两种接口，遍历树的方式为 ANTLR 自动帮你遍历，你可以在 exit*** 的时候处理当前节点
- Visitor 提供 visit*** 接口，手动遍历子节点，可以按照自己制定的顺序

AST is done!

- 接下来要做什么呢？
- Semantic Check!
- 不过在这之前，我们需要实现一个 ASTVisitor，不然怎么做语义分析呢？
- ASTVisitor 是一个接口(interface)，相当于一个纯虚类
- 回想一下你是如何使用 ANTLR 的 Visitor 的，你需要支持一个类似的 visit 功能

Semantic Check

- 在你的 ASTVisitor 上跑，遍历这棵 AST
- 你需要做几件事情（by 青木峰郎《自制编译器》）：
  - 变量引用的消解
  - 类型名称的消解
  - 类型定义检查
  - 表达式的有效性检查
  - 静态类型检查
- 你可以把所有要做的合在一个类里，也可以用多个类来做这些事情

Variable Reference Resolve

- 变量引用的消解：确定某个变量具体指向哪个变量
- 比如变量 “i” 可能是全局变量 i，也可能是局部变量 i，你需要通过这个过程来消除这样的不确定性，确定 “i” 到底是哪个变量
- 这其实是一个作用域(Scope)的问题
- Mx 有一套 Scope 的规则
- 你可以写一个 Scope 类，每个 Scope 里存一个符号表(Symbol Table)，记录当前这个类里每个 identifier 对应的变量是什么
- 查找时，如果当前 Scope 内找不到，就到父 Scope 里去找

Type Name Resolve

- 类型名称的消解
- 实际上你在 Front End 阶段需要有两套 Type
- 第一套 Type，指你构建 AST 时直接从 CST 上观察到的类型名称，比如 “int”, “bool”, “AAA”, “BBB”
- 你知道 “int” 就是 int，但在构建 AST 时，你知道 “AAA” 是一个什么类吗？你知道这个类存不存在，或者是这个类有哪些成员吗？
- 第二套 Type 即更完整的类型信息，你知道一个类里有哪些成员，哪些方法
- 知道具体的类型信息是 semantic 阶段必不可少的

Type Definition Check

- 类型定义检查
- 检查声明变量时的类型是否为 void

Expression Check & Type Check

- 表达式的有效性检查 & 静态类型检查
- “1++” 显然是不合法的，你需要检查一些表达式是否合法
- “1 + true” 显然也是不合法的，你需要检查表达式的类型
- “int a = false” 显然也是不合法的……
- 这些你都需要检查

- 注意，Mx 里不存在任何的隐式类型转换

Semantic Check is Done!

- 你可能需要花一段时间来看看自己是否能拦住有语法、语义错误的代码
- 如果在 Parse 阶段就解体了，ANTLR 会自动帮你拦下来
- 如果在 Semantic 阶段解体，那就是你写的代码把它拦下来

Some Questions

- “我发现不同的人写的前端都不一样，我设计的 AST 要是与别人的不一样，会不会对后面造成很大影响啊？”
- “不看别人代码就写不出代码，怎么办啊？”

Intermediate Representation

- 你需要花很多时间来思考它，在你完全没有头绪的时候
- 不要着急写出代码

Intermediate Representation

- 选择写什么 IR 呢？
- CFG + 线性 IR（感觉班上很多同学选择了 LLVM IR）
- 你可以自己设计一套 IR，也可以照着 LLVM IR 写
- 生成完 IR 之后，按照 ASTVisitor 的做法，写一个 IRVisitor 的 interface
- IRPrinter…
- IRInterpreter…

Q&A

Thanks!