编译器文档

语言选择

选择Java，虽然Java8缺少很多特性，有些代码显得比较冗长，相对而言C++11更加现代，性能也更好，但是之后代码生成和优化部分可能需要维护比较复杂的数据结构（图），Java自动内存管理可以降低编码难度。

词法分析

主要参考了《crafting interpreter》的词法分析器实现，总结了自己之前实现过的词法分析器的实现，sloth-lang: 出于学习目的编写的基于字节码的解释器。

整个词法分析器是一个while{switch{...}}结构，对于关键字和标识符，采用直接识别而不是真的构建DFA，这样比较便于维护，对于不可能冲突的单字符，如+，-，直接识别，对于可能冲突的单双字符进行特判，如/，/*，//。此外需要处理数字串、注释和格式字符串等。

语法分析

递归下降法配合适当的Look forward，因为输出语法树形式要求的限定，没有使用可以简化表达式识别的Pratt Parser（自顶向下算符优先级，这大概会把表达式树展平，不太好处理）。

最终的设计中，对赋值语句采用了回溯的方式（和表达式语句冲突），对表达式采用了改写文法然后对语法树进行变形的方式。

错误处理相关

对于可能（因为错误）不存在的Token，如`)`，`]`，`;`，不能作为判断语法成分的条件。因为错误处理需要建立符号表，所以是和中间代码生成一起写的（词法分析和语法分析也处理一些错误）。