1. 解释器 和虚拟机

树和子树，方便表示连续的语法关系? 还是把不同分号语句分开，表示不同的语法?

编译成汇编是最直白的

链接器又如何实现？

可以在其他语言实现的虚拟机中直接编译出机器码加载到代码段，然后跳转执行

申请大量内存，把真正的代码加载进去，原程序只是个壳

语法的拆分方式 拆出基本单元 把嵌套层移出 回退到父节点

If(a+b>0){} else{} 需要拆出括号内的内容 (a+b)>0 拆a+b a b 对于中缀，后缀前缀的拆分

执行顺序

最终目标是机器码还是其他码

要自己设计出对应规则

编译过程

问题1:空间如何分配 ，答案：可以分段 全局使用数据段 局部使用堆栈

定义语句 直接分配空间

赋值语句 mov [] ,立即数或者是

If(){} 翻译成 cmp jb

编译器 转为机器语言

Java先编译再解释，执行时会把一部分转为机器语言。 JIT编译

开发语言处理器

用JAVA开发Stone 用jvm执行 用java是为了面向对象 ， haskell适合开发语言处理器 不用yacc

单词排列 抽象语法树 生成其 代码或者执行

1、Stone语法 动态类型 分号 无ret 最后一条语句是结果 if while没有括号 句尾分号可以省略 多句一行不能省 语句体必须用{}包

不能模棱两可，避免歧义 if if else

三、1、 源代码是长字符串 包括 空格和换行符 切割成多个子串

token对象 去除空白和注释

使用Token对象，保留字符串 ，类型，位置行号 每个单词是一个对象 子类 标识符

整型字面量 字符串字面量 对象拥有 isidentifier isString isNumber方法 获取行号

获取文本

静态：Token.EOF程序结束 Token.EOL换行符

2、分类 对子串类别 无歧义

标识符 变量 函数 类名 + -等 标点和保留也被分类到标识

使用正则表达式匹配 数字 标识符号 字符串（需要处理转义）

逐行读取 所有匹配项

Jflex工具来分析单词（本例不必）

四、抽象语法树 语法分析 分析单词关系 同一语句/表达式 左右配对() 语法错误

对象和树形结构 操作符为根

叶根

Bnf 上下文无关表达 铁路图

格式:

五、语法分析器 语法规则匹配

内部dsl 流畅界面库

预处理器处理# 宏展开

C语言生成汇编语言 汇编器 生成目标语言 链接器 动态加载 垃圾收集

Jit编译器程序转机器语言

不转换语言直接运行就叫解释器（或者虚拟机，虚拟机是其他语言写的程序，而且也可以动态加载机器码

）

静态类型检查 运行中动态类型检查（由于会修改类型，所以要时刻检查，解释时进行的工作会增多）

语法树 x=y+z 抽象语法树AST

中序 以符号为中心 (要求有一个值) 理解函数式编程? 代码其实是数的传递 执行顺序

AST生成中间代码 方便不同语言使用同样 的编译器 gcc用的是rtl

我们也可以让自己的语言生成c目标语言 , java字节码 然后借用他们的解释器

代码生成器 优化

比较难的可以先不做 浮点运算 预处理器

1. 词法分析 扫描 分割单词，删除空白和注释

单词token分类 数值 字符串 函数名，变量名（标识符） 保留字 int char ret

符号

1. 生成语法树 解析器生成器 LR LALR LL 现在的语法基本都属于LALR yacc

解析器

Javacc 语法描述文件

正则表达式 保留字 标识 数

解析器 识别多个token构成的语法

非终端符 组合的单位 终端符token

Bnf: 终端 非终端 连接 重复 选择（多选1 ） 可省略

语法，语义 语法是组合规律 语义是语句含义，记号和记号对象的关系

字母表 符号 符号串 符号串是有序的 长度 连接 相等 幂运算

闭包 集合闭包

文法 语言只含有穷句子

文法表示 (VN，VT P S) 非终结符（可被取代） 终结符 产生式规则集合 开始符号

2 型文法 下推自动机 左部是非终结符

::= 定义为 |或者 <非终结符>

<有符号整数>::=<无符号整数>|<正负号><无符号

[]0或1

{}多次重复

ebnf

递归下降分析程序:需要消除右边递归的情形

状态转换图 结点是状态 开始状态 接收状态

1. 词法分析

继承多态实现

Token类（接口）

子类Token.EOL(endof line)换行 是字符串 Token.EOF

移除空白和注释

运行时异常 Runtimeexcption stoneexception

Token EOF EOL lineNumber行号 getLineNumber getText getNumber

isString isNumber

isIdentifier()是否标识符？

利用正则 .匹配任何字符 ...

字符串字面量 整型[0-9]+ 标识符（包括运算符括号等）[A-Z\_a-z][ A-Z\_a-z0-9]\*|==|<=

第一个字符不能是数字

转义\n \” \\

一次性匹配所有

//.\*匹配 注释 \s\*匹配任意长空格 根据匹配状况可以知道类型

ASTree

Stone中所有符号当标识符

词法分析器

Lexer

正则表达式 regexPattern read()每次读一个 peek(i) 返回read后面的单词 读完返回EOF

构造时需要回溯 这里用预读后面的方法peek peek读的单词保存在queue

Read的单词从queue删除

LineNumberReader reader

hasMore

pattern对象=Pattern.compile()

matcher.region matcher.lookingAt mather.end matcher.group(1)

addToken

fillQueue queue.remove() queue.get() queue.size() qunue.add()

line=reader.readLine() reader.getLineNumber()

new IdToken(line, matcher)

NumberToken value getText {toString} getNumber

IdToken text

ParseException

二元 binaryExpress left operator right

整型字面量 NumberLiteral value

变量名Name name

不含括号

ASTree child(i) children()遍历器 numChildren location()所在位置 iterator()

ASTleaf ASTlist （抽象类） 上面的整型，表达式是实现 叶节点孩子为0 空集

Chilren 列表数组 叶没有chilren 有token

组合模式

利用封装，不直接访问数据 **这样设计好吗？left right指针明显更方便，然后就是用数组保存子元素 好吗？**

ASTLeaf empty(数组列表) token ASTLeaf(t){token = t } child() numChildren()

Children() 返回empty toString() location( token.getLineNumber())

ASTList(list) {chilren = list} child(i) children toString()(所有子单词都加入)

Location() (子元素第一个非空)

NumberLiteral(token) value() { token.getNumber()}

Name(t) name() {token.getText}

BinaryExpr(list) left() {children(0)} operator() right

Ebnf {} [] | ()

Factor 因子 number| （表达式）

number

Expression表达式 term { +|- term } N个项的加减

Term 项 factor { (\*|/) factor } n个项乘除

不匹配就是语法错误 铁道图：圆是终结 方是非终结 分支或者循环

**存在下层递归上层的可能 factor里有expression**

**语法分析是找和ebnf匹配的单词组 分组后的单词组 继续向后匹配，得到更大的单词组**

**子树就是单词组**

终结符 Number indentifier string op(运算符) eol

**这次不用语法规则来规定优先级了**

Primary（基本组成元素） (exper) |number|string

Factor -primary|primary

Expr factor {op factor}

Block { [statement]{ ; | eol [] } } 可以没语句 换行必须; 或eol

Statement 包括空语句

假如 if expr block [else block ] | while expr block | simple

Simple expr

Program

Yacc

Parse类 rule() rule(class)创建语法规则对象parser

parse() number() number(class)

indentifer（） string() token( pattern) sep(pattern)

ast(parser)

option ()

maybe(parser) or() repeat() expression(parser,op)

reset()清空 class是语法分析树节点类

Operator类

BasicParser 语法规则 reserved operators

Parse 类型 exr0 primary factor expr statement0 block simple

Statement

Sep(“(”) 是指 一些特殊的符号 比如括号 ( )

Rule.sep(“(”).ast

Exp0=rule 空对象