COOL 语法解析器（Parser）实验报告

Compiler Principle Assignment

姓名: **谭润**
学号: **20238131027**
班级: **大数据二班**
项目代码：Github 链接

2025 年 11 月 27 日

摘要

本文基于简化版 COOL 语法完成了一个功能齐全的解析器示例，配合手写的 Flex 词法分析器用于测试语法功能。实验中实现了类和继承、属性、方法、表达式（函数调用、分支、循环、let、case）、常量和基础运算符的解析功能，并输出简单的树状结构以验证解析正确性。实验使用 Bison（Yacc）和 Flex 构建解析管道，并提供 Makefile 便于快速构建和运行测试用例。

1 任务与设计目标

本次实验目标是实现 COOL 语言的一个基础解析器（Parser），能够将词法分析器（Lexer）生成的 token 流解析为树状结构（AST）。主要目标包括：

- 实现类/继承、方法/属性解析
- 处理表达式（if/while/let/dispatch、操作符）
- 能够解析并报告语法错误，并在测试文件（good.cl / bad.cl / stack.cl / complex.cl）上演示

2 实现说明

2.1 代码结构

```
My_work/
|-- cool.y
|-- lexer.l
|-- parser.y
|-- Makefile
|-- README.md
|-- report_final.tex
|-- run_tests.ps1
```

2.2 核心实现要点

- Lexer 使用 Flex 编写，实现关键词大小写不敏感识别，支持 ‘TYPEID‘/‘OBJECTID‘ 区分，字符串转义解析、嵌套注释与错误检测。
- Parser 使用 Bison 编写，在语义动作中调用提供的 ‘cool-tree.h‘ 的构造函数以生成 AST，并使用 ‘SET_NODELOC‘/‘@$‘ 来记录位置。
- 为便于演示，实现了一个简短的 AST 打印器”print_node”，可以在解析成功时打印解析树。
- 本项目参考了 ‘cool.y‘，实现中保留了要求的语法规则和 AST 构建约定，并在语义动作中使用 ‘SET_NODELOC‘ 为 AST 设置行号，便于后续语义分析和错误报告。

2.3 主要实现

- feature_list: 支持空列表与带分号的 feature 列表，解决部分边界导致解析失败的问题。
- let 表达式: 实现了多变量 let 的嵌套解析，将多个绑定展开成嵌套的 let 调用，保持语义正确。
- Node 位置定位: 在每个语义动作中使用 ‘@$‘ 和 ‘SET_NODELOC(@1)‘ 显式设置位置，使后续错误报告更准确。

- 错误恢复：在一些规则中使用 ‘error‘ 产生式并创建默认 AST 节点，以便解析器在错误时继续处理后续输入。

3 构建与运行

3.1 依赖

- flex (>= 2.6.x)
- bison (>= 3.x)
- g++ (支持 C++11 或更高)
- Make（建议用于快速构建）
- 推荐在 WSL 或 Linux 环境中执行（Windows 环境下需安装 MSYS/Cygwin 或 WSL 来运行 ‘bison‘/‘flex‘/‘make‘）

3.2 开发环境

- 操作系统：Ubuntu 22.04.1 LTS
- 内核版本：5.15.0-60-generic
- Flex 版本：2.6.4
- G++ 版本：g++ (Ubuntu 11.3.0) 11.3.0
- Make 版本：GNU Make 4.3
- SPIM 版本：SPIM Version 8.0

3.3 构建命令

```
cd d:\\VS_code\\compile\\shiyan3\My_work
make
```

或者手动：

```
bison -d -o cool-parse.cc cool_ai.y
flex -o lex.yy.c lexer.l
g++ -std=c++11 -Wall -g cool-parse.cc lex.yy.c
-lfl -o parser
```

3.4 运行示例

运行示例（使用 demo parser/lexer）：

```
./parser ..\good.cl
./parser ..\bad.cl
./parser ..\stack.cl
```

4 测试说明与结果

4.1 基础测试

使用提供的 ‘good.cl‘进行测试，构建并运行解析器得到解析成功，并打印树结构用于验证。

示例输出（节选，运行 ‘./parser .. good.cl‘）：

```
Parse succeeded
Classes
    Class
        type: A
        Features
            Method
                name: ana
                // feature: ana
```

4.2 错误测试

对 ‘bad.cl‘ 进行测试，解析器会检测语法问题（例如继承关键字拼写错误、缺少大括号等），输出示例（运行 ‘./parser .. bad.cl‘）：

```
Parse error at line 8: syntax error at or near TYPEID
Parse failed
```

5 遇到的问题与解决方案

在实验过程中，主要遇到以下挑战：

- 字符串与注释边界处理：由于字符串需支持转义字符并且不能跨行，采用独占状态（‘%x‘）并在字符串状态内解析转义字符，遇到换行或 EOF 则返回适当的 ERROR Token；注释实现支持多行和嵌套，采用计数器和注释状态管理。
- 工具环境限制（Windows）：在默认 Windows PowerShell 中通常缺少 ‘bison‘/‘flex‘/‘make‘，建议在 WSL/Ubuntu 环境下运行编译测试；在 Windows 环境下也可使用 MSYS2/Cygwin 或安装相关工具后再运行。
- 与课程 AST 集成：将 Bison 的语义动作映射到 ‘cool-tree.h‘ 中的构造函数需要仔细对齐类型签名和节点位置语义（‘SET_NODELOC‘），否则编译或链接阶段会报错；我在 ‘cool.y‘ 中遵循标准接口以保证兼容性。

- let 等复杂语法：多变量 let 的嵌套需要在语义动作中通过嵌套 ‘let‘ 调用来还原 AST 结构，若实现不当会导致语法和语义不一致。

6 总结

本实验实现了一个简化但功能完整的 COOL 语法解析器，支持核心语法结构（如类、继承、属性、方法、表达式、分发调用、let、case 等）的解析。实现的 ‘cool_ai.y‘（位于 ‘My_work/‘）已与课程 AST 构造函数对接，并做了错误恢复处理；‘lexer.l‘ 与 ‘parser.y‘ 提供了用于快速测试的演示实现。使用 Flex（词法）和 Bison（语法）提供方便的解析开发与测试流程。该实现为课程作业提交提供了完整的 ‘cool.y‘ 参考实现，并可在必要时进一步对齐课程评分工具链。

7 实现范围与局限

本次实现覆盖了课程要求的主要解析规则与语义动作，详细实现如下：

- 已实现：类、继承、方法、属性、常见表达式（算术、比较、赋值、if、while、case、let、dispatch 等）、常量、字符串与注释处理、错误检测与恢复。
- 局限：完全的类型检查、语义分析、代码生成（Semant/Cgen）未在本项目中实现；另外对字符串极端边界（极长字符串）额外校验策略可进一步加强。

A 附录：cool.y 核心源码片段

```
%{
#include "cool-tree.h"
#include "stringtab.h"
#include "utilities.h"
#include <iostream>
using namespace std;

extern char *curr_filename;
extern int curr_lineno;

// 位置处理
#define YYLTYPE int
extern int node_lineno;
#define SET_NODELOC(Loc) node_lineno = Loc;

void yyerror(const char *s);
extern int cool_yylex();
static int yylex_wrapper() {
    extern YYLTYPE cool_yylloc;
    int tok = cool_yylex();
    cool_yylloc = curr_lineno;
    return tok;
}
#define yylex yylex_wrapper

Program ast_root;
Classes parse_results;
int omerrs = 0;
%}

// 语法单元类型定义
%union {
  Boolean boolean;
  Symbol symbol;
  Program program;
  Class_ class_;
  Classes classes;
  Feature feature;
  Features features;
  Formal formal;
  Formals formals;
  Case case_;
  Cases cases;
  Expression expression;
  Expressions expressions;
  char *error_msg;
}

// Token定义
%token CLASS ELSE FI IF IN
%token INHERITS LET LOOP POOL THEN WHILE
%token CASE ESAC OF DARROW NEW ISVOID NOT
%token <symbol> STR_CONST INT_CONST BOOL_CONST TYPEID
    OBJECTID
%token ASSIGN LE ERROR

// 运算符优先级与结合性
%nonassoc IN
%right ASSIGN NOT
%nonassoc LE '<' '='
%left '+' '-'
%left '*' '/'
%left ISVOID
%left '~' '@' '.'

// 非终结符类型声明
%type <program> program
%type <classes> class_list
%type <class_> class
%type <features> feature_list
%type <feature> feature
%type <formals> formal_list
```

```
%type <formal> formal
%type <cases> case_list
%type <case_> case_branch
%type <expressions> expr_list expr_block_list
%type <expression> expr

%start program
%%

// 程序入口
program:
  class_list { @$ = @1; ast_root = program($1); }
  ;

// 类列表
class_list:
  class { @$ = @1; $$ = single_Classes($1); }
  | class_list class { @$ = @1; $$ = append_Classes($1, single_Classes($2)); }
  ;

// 类定义（含继承）
class:
  CLASS TYPEID '{' feature_list '}' ';' {
      @$ = @1;
      $$ = class_($2, idtable.add_string("Object"), $4, stringtable.add_string(curr_filename));
  }
  | CLASS TYPEID INHERITS TYPEID '{' feature_list '}' ';' {
      @$ = @1;
      $$ = class_($2, $4, $6, stringtable.add_string(curr_filename));
  }
  | error ';' {
      @$ = @1;
      $$ = class_(idtable.add_string("Error"), idtable.add_string("Object"), nil_Features(), stringtable.add_string(curr_filename));
  }
  ;

// 特征列表（方法/属性）
feature_list:
  /* 空列表 */ { $$ = nil_Features(); }
  | feature_list feature ';' { @$ = @2; $$ = append_Features($1, single_Features($2)); }
  ;

// 特征（方法/属性）
feature:
  // 方法定义
  OBJECTID '(' formal_list ')' ':' TYPEID '{' expr '}' {
      @$ = @1; SET_NODELOC(@1); $$ = method($1, $3, $6, $8);
  }
  // 无初始值的属性
  | OBJECTID ':' TYPEID {
      @$ = @1; node_lineno = 0; Expression ne = no_expr(); SET_NODELOC(@1); $$ = attr($1, $3, ne);
  }
  // 有初始值的属性
  | OBJECTID ':' TYPEID ASSIGN expr {
      @$ = @1; SET_NODELOC(@1); $$ = attr($1, $3, $5);
  }
  ;

// 形参列表
formal_list:
  /* 空列表 */ { $$ = nil_Formals(); }
  | formal { $$ = single_Formals($1); }
  | formal_list ',' formal { @$ = @1; $$ = append_Formals($1, single_Formals($3)); }
  ;

// 单个形参
formal:
  OBJECTID ':' TYPEID { @$ = @1; $$ = formal($1, $3); }
  ;

// 表达式块（分号分隔）
expr_block_list:
  expr ';' { $$ = single_Expressions($1); }
  | expr_block_list expr ';' { @$ = @1; $$ = append_Expressions($1, single_Expressions($2)); }
  ;

// 核心表达式（含赋值、调用、分支、循环等）
expr:
  // 变量赋值
  OBJECTID ASSIGN expr { @$ = @1; SET_NODELOC(@1); $$ = assign($1, $3); }
  // 静态方法调用
  | expr '@' TYPEID '.' OBJECTID '(' expr_list ')' { @$ = @1; SET_NODELOC(@1); $$ = static_dispatch($1, $3, $5, $7); }
  // 动态方法调用
  | expr '.' OBJECTID '(' expr_list ')' { @$ = @1; SET_NODELOC(@1); $$ = dispatch($1, $3, $5); }
  // 无接收者的方法调用（默认self）
  | OBJECTID '(' expr_list ')' { @$ = @1; SET_NODELOC(@1); $$ = dispatch(object(self_sym), $1, $3); }
  // 条件分支
  | IF expr THEN expr ELSE expr FI { @$ = @1; SET_NODELOC(@1); $$ = cond($2, $4, $6); }
  // 循环
  | WHILE expr LOOP expr POOL { @$ = @1; SET_NODELOC(@1); $$ = loop($2, $4); }
  // 模式匹配
  | CASE expr OF case_list ESAC { @$ = @1; SET_NODELOC(@1); $$ = typcase($2, $4); }
  // 代码块
  | '{' expr_block_list '}' { @$ = @1; SET_NODELOC(@1); $$ = block($2); }
  // 单变量let表达式
  | LET OBJECTID ':' TYPEID IN expr { @$ = @1; SET_NODELOC(@1); node_lineno = 0; Expression ne = no_
```

```
      expr(); SET_NODELOC(@1); $$ = let($2, $4, ne, $6);
      }
  // 带初始值的单变量let
  | LET OBJECTID ':' TYPEID ASSIGN expr IN expr { @$ =
      @1; SET_NODELOC(@1); $$ = let($2, $4, $6, $8); }
  // 多变量嵌套let
  | LET OBJECTID ':' TYPEID ',' OBJECTID ':' TYPEID IN
      expr { @$ = @1; SET_NODELOC(@1); node_lineno = 0;
      Expression e1 = no_expr(); SET_NODELOC(@1); node_
      lineno = 0; Expression e2 = no_expr(); $$ = let($2,
       $4, e1, let($6, $8, e2, $10)); }
  // 算术运算
  | expr '+' expr { @$ = @1; SET_NODELOC(@1); $$ = plus(
      $1, $3); }
  | expr '-' expr { @$ = @1; SET_NODELOC(@1); $$ = sub(
      $1, $3); }
  | expr '*' expr { @$ = @1; SET_NODELOC(@1); $$ = mul(
      $1, $3); }
  | expr '/' expr { @$ = @1; SET_NODELOC(@1); $$ =
      divide($1, $3); }
  // 一元运算
  | '~' expr { @$ = @1; SET_NODELOC(@1); $$ = neg($2); }
  | NOT expr { @$ = @1; SET_NODELOC(@1); $$ = comp($2);
      }
  | ISVOID expr { @$ = @1; SET_NODELOC(@1); $$ = isvoid(
      $2); }
  // 比较运算
  | expr '<' expr { @$ = @1; SET_NODELOC(@1); $$ = lt($1
      , $3); }
  | expr '=' expr { @$ = @1; SET_NODELOC(@1); $$ = eq($1
      , $3); }
  | expr LE expr { @$ = @1; SET_NODELOC(@1); $$ = leq($1
      , $3); }
  // 新建对象
  | NEW TYPEID { @$ = @1; SET_NODELOC(@1); $$ = new_($2)
      ; }
  // 常量
  | INT_CONST { @$ = @1; SET_NODELOC(@1); $$ = int_const
      ($1); }
  | BOOL_CONST { @$ = @1; SET_NODELOC(@1); $$ = bool_
      const($1); }
  | STR_CONST { @$ = @1; SET_NODELOC(@1); $$ = string_
      const($1); }
  // 变量引用
  | OBJECTID { @$ = @1; SET_NODELOC(@1); $$ = object($1)
      ; }
  // 括号表达式
  | '(' expr ')' { @$ = @1; $$ = $2; }
  ;

// case分支列表
case_list:
  case_branch { $$ = single_Cases($1); }
  | case_list case_branch { @$ = @1; $$ = append_Cases(
      $1, single_Cases($2)); }
  ;

// 单个case分支
case_branch:
  OBJECTID ':' TYPEID DARROW expr ';' { @$ = @1; SET_
      NODELOC(@1); $$ = branch($1, $3, $5); }
  ;

%%

// 全局变量初始化
int curr_lineno = 1;
Symbol self_sym = idtable.add_string("self");

// 语法错误处理
void yyerror(const char *s) {
  extern char *curr_filename;
  cerr << '"' << curr_filename << "", line " << curr_
     lineno << ": " << s << " at or near ";
  print_cool_token(yychar);
  cerr << endl;
  omerrs++;
  if (omerrs > 50) {
    fprintf(stdout, "More than 50 errors\n");
    exit(1);
  }
}
```