1. C++语言语法

项目中j-ast-node全都是采用继承和多态机制定义的，理解很困难，各种类型的转换相对复杂，在写的过程中也出现了很多类型不匹配的错误。

1. 代码中定义方便的工具函数可以极大降低工作量

比如可以对parser部分的mustbe和have进行扩充，不仅对单一的expected进行判别，而是对一组expected符号进行判别

static bool have(vector<TokenKind>expected)

{

return find(expected.cbegin(), expected.cend(), token.token)!=expected.cend();

}

static Token mustbe(vector<TokenKind>expected)

{

if(!have(expected))

exit(0);

Token old = token;

next();

return old; //返回原token

}

使用方便：

string type = mustbe({jk\_int, jk\_boolean, jk\_identifier, jk\_char}).spelling;

1. 翻译过程中的函数调用是一个难点
2. func\_name(arg1, arg2 ...)
3. variable.func\_name(arg1, arg2, ...)
4. class\_name.func\_name(arg1, arg2, ...)

如何区分这三种情况，正确处理函数调用就是parser最难的部分。

4.有了前面的经验，codegen采用了新方案，避免大量的switch和if-else条件语句：

map<ast\_kind, void(\*)(jn\_statement)>\_smap = {

{k\_jn\_return, t\_return}, {k\_jn\_return\_expr, t\_return}, {k\_jn\_let, t\_let}, {k\_jn\_let\_array, t\_let\_array},{k\_jn\_do, t\_do}, {k\_jn\_if, t\_if}, {k\_jn\_if\_else, t\_if}, {k\_jn\_while, t\_while}

};

其中map的key是statement的类型，map的value是处理该类型的handler函数，这样可以直接这样调用：

smap[ast\_kind\_of(statement)](statement);

针对expresion的翻译也是类似的。