PyCon China 2020

PyConChina 2020
PYTHON 中国开发者大会 2020

Python For Good

用Python从0构建一个简单的脚本语言

二两

广州省省回头车 Python工程师/滞销书作者

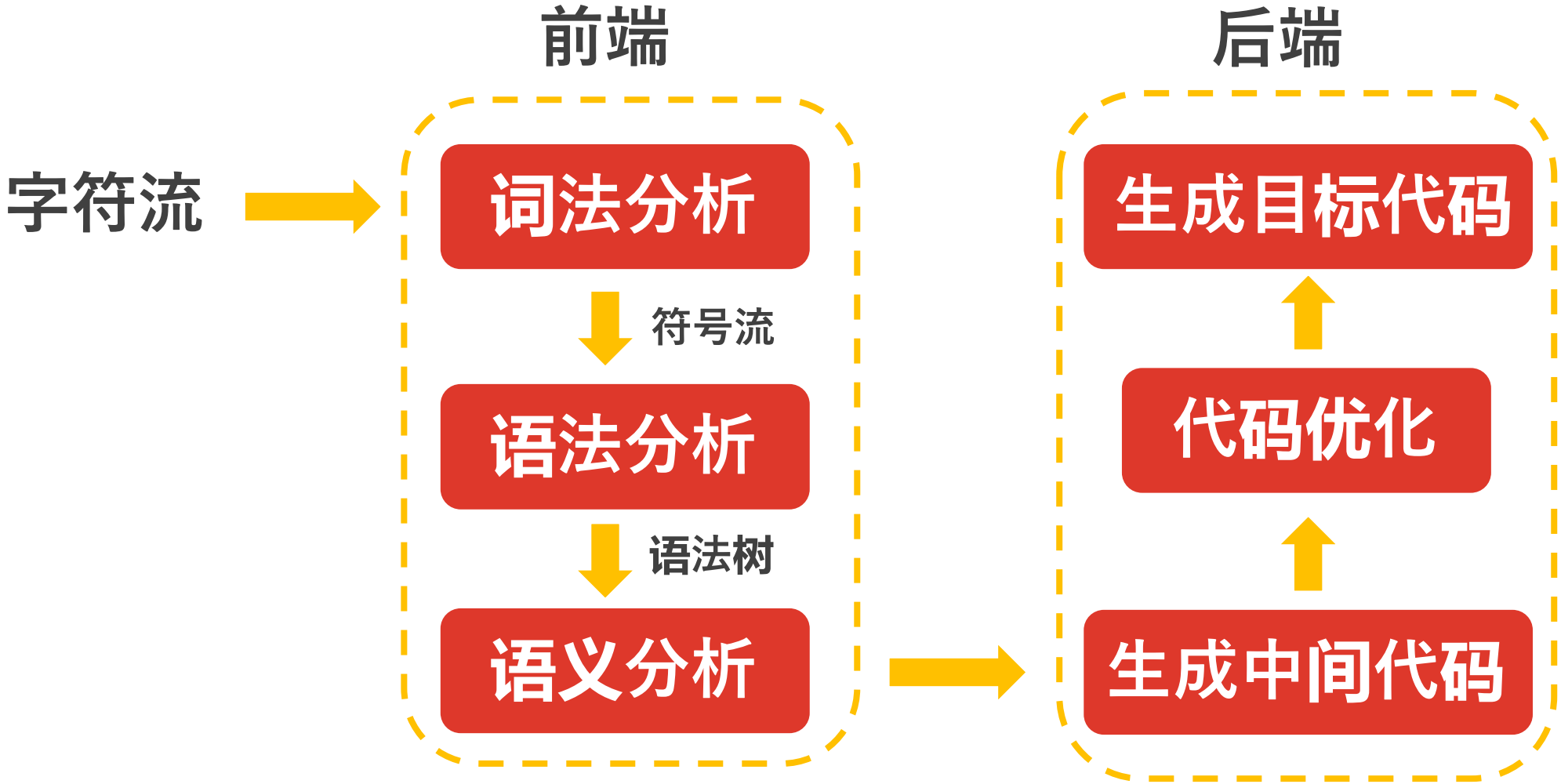
概览

保持思辨思想，大家一同讨论与学习。

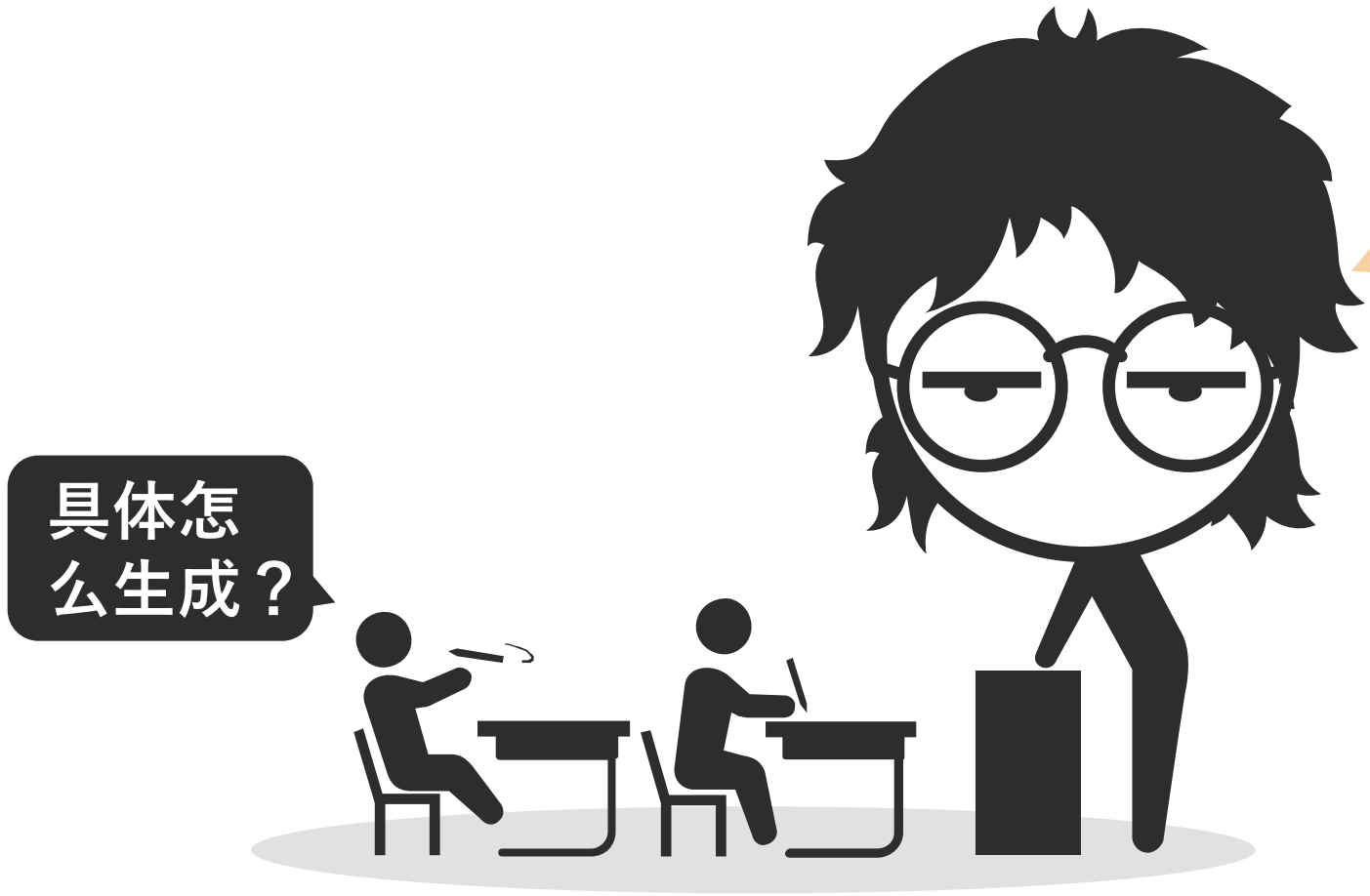
1.编译原理背景知识

2.ToyPL的实现细节

编译整体流程

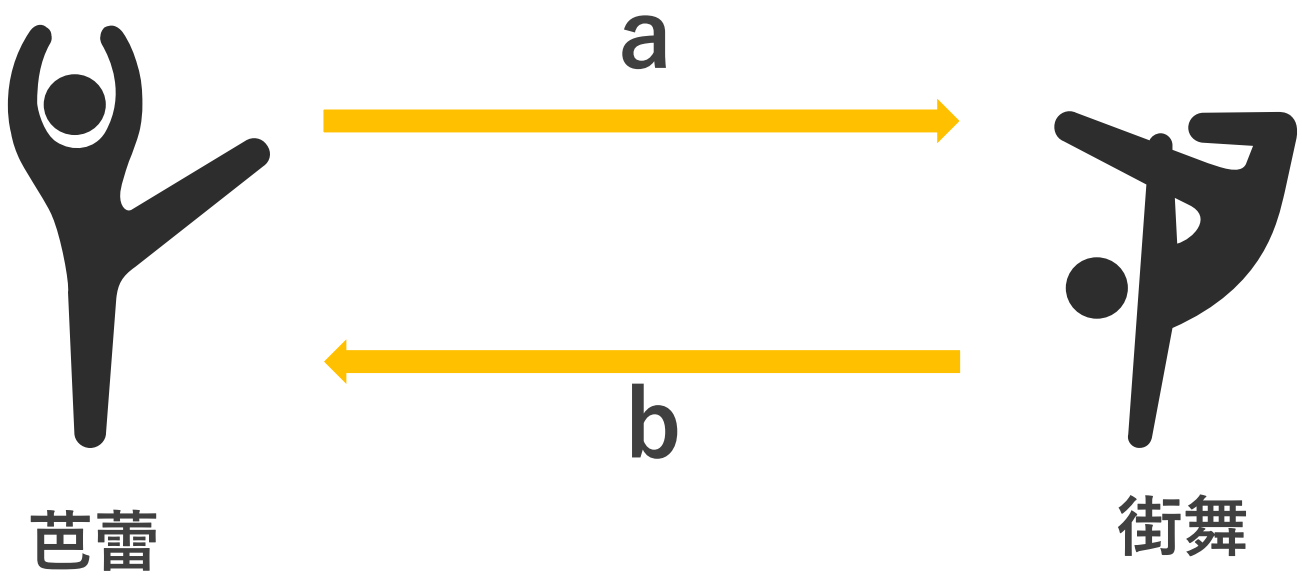


词法分析（lexical analysis）



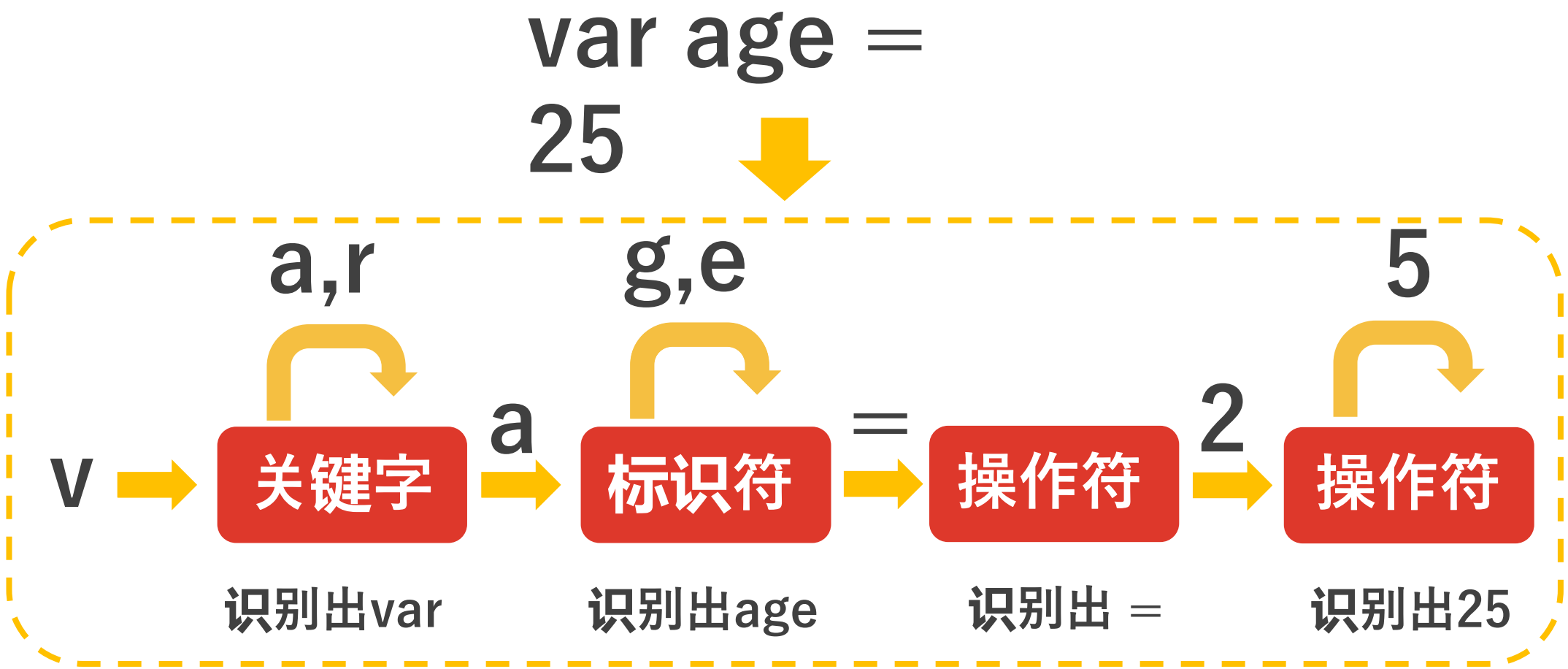
词法分析：读入组成源程序的字符流，将它们组成有意义的词法单元(tokens)，该tokens指的是一种二元组，具体形式为< token-name, attribute-value >。

有限自动机（finite automaton）



- a被称为输入
- 图中有芭蕾和街舞这两种状态
- 从芭蕾状态接受了输入a转移为了街舞状态，这种转移称为转移规则

词法分析（lexical analysis）



词法分析（lexical analysis）

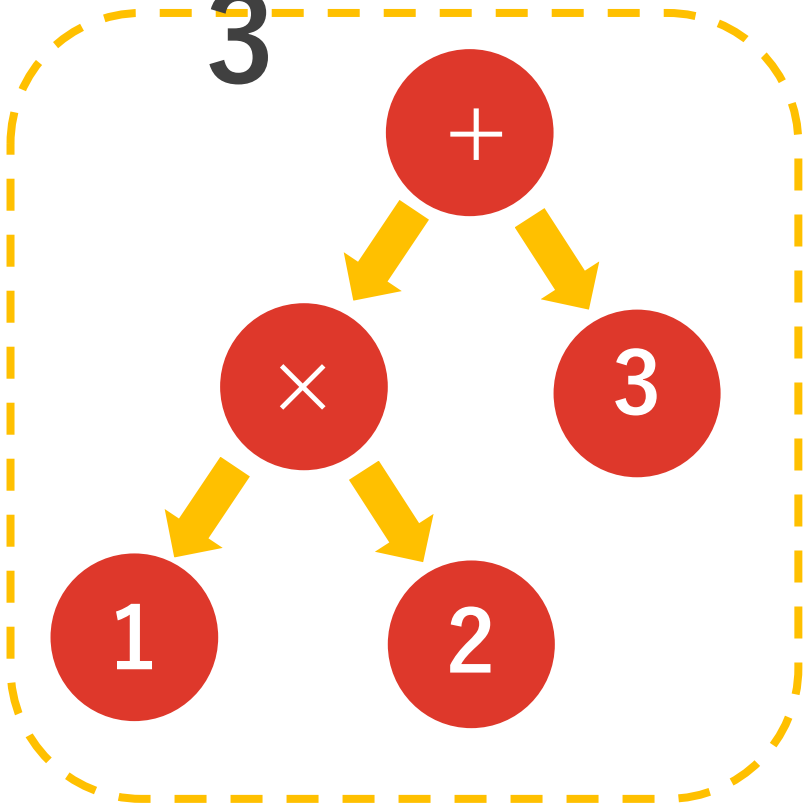
词法分析的原理就是依据构建好的有限自动机，在不同状态中迁移，从而解析出token。
具体的实现方式：通过while循环，一直读取字符流，然后通过if判断对不同的内容生成不同的tokens便可完成词法分析。

```
# 词法分析
def make_tokens():
  while current_chat != None:
    if current_chat == "=":
      ...
    elif current_chat == "<":
      ...
    elif ...
```

语法分析（Parsing）

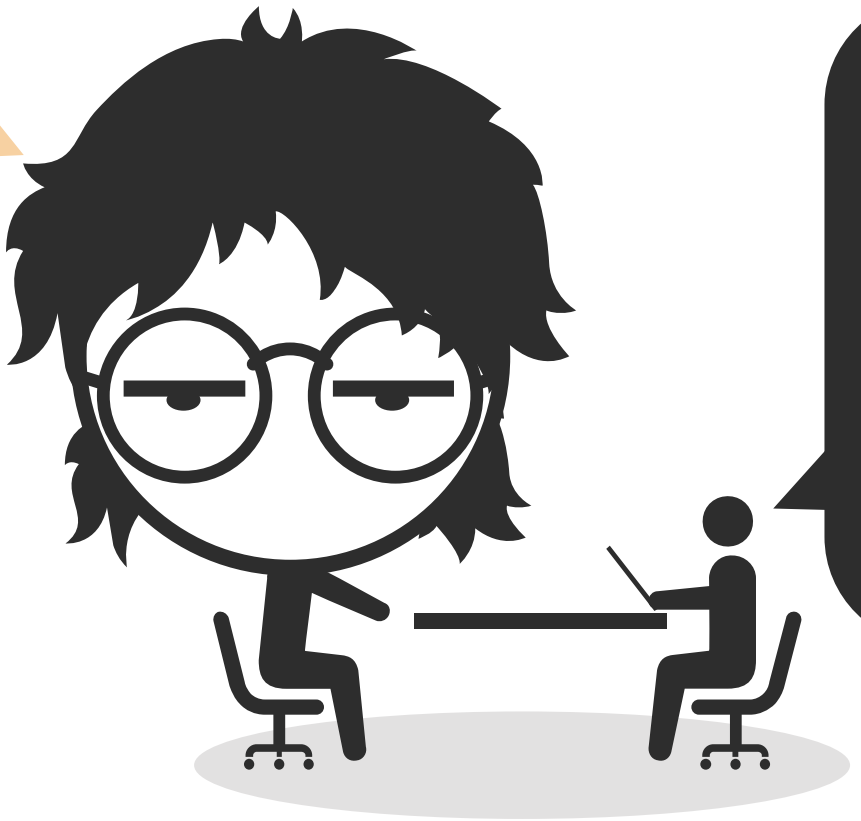
语法分析：使用词法分析生成的tokens序列构造抽象语法树。

1×2+3



语法分析（Parsing）

这几个问题很关键呀！



我有几个疑惑：
1.抽象语法树按什么规则生成？
2.通过什么算法生成抽象语法树？
3.如何使用生成的抽象语法树？

上下文无关文法（context-free grammar）


```
add  -> add + mul | mul
mul  -> mul * item | item
item -> id | number | add
```

凡是编程语言都有具体的语法规则，我们使用文法来描述这种规则，所谓文法，就是一种描述规则的规则，我们常用上下文无关文法来描述语言的语法规则。

扩展巴科斯范式（EBNF）


```
# BNF
add ::= add + mul | mul

# EBNF,*表示重复0次到多次
add -> mul (+mul)*
```

上下文无关文法是一种概念，我们通常使用巴科斯范式(BNF)这种约定俗成的写法将其表示出来，对应一些复杂的规则，我们可以使用扩展巴科斯范式(EBNF)，它相对于BNF多了类似正则表达式的写法。

产生式与推导

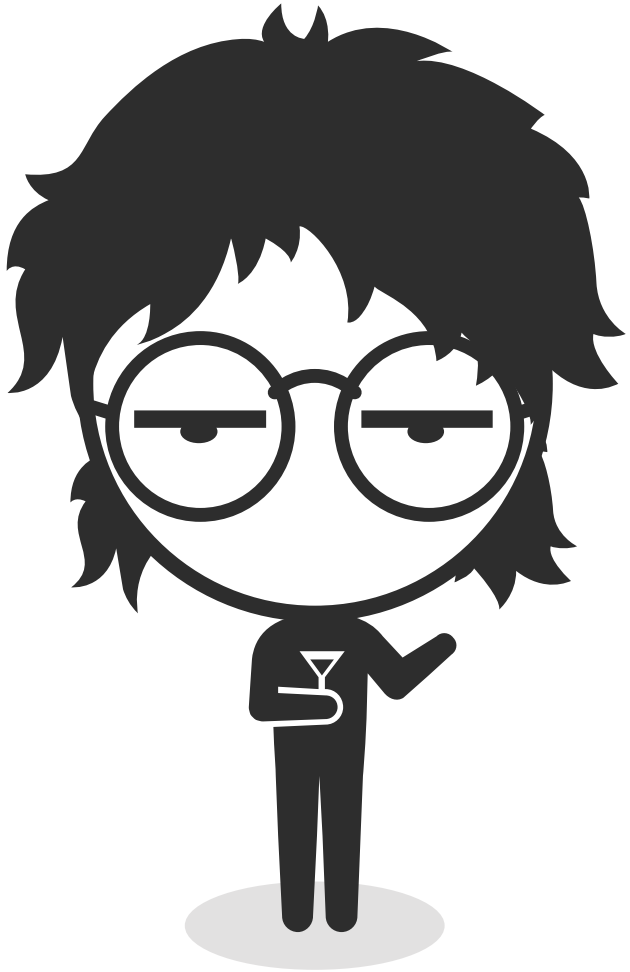
产成式

```
Sent -> S V O

S -> 人 | 天

V -> 吃 | 下

O -> 雨 | 雪 | 饭 | 肉
```



编程语言的语法规则由上下文无关文法表示，而上下文无关文法由一组产生式(替换规则)组成，这些产生式可以由EBNF书写。

产生式与推导


```
Sent -> S V O

S -> 人 | 天

V -> 吃 | 下

O -> 雨 | 雪 | 饭 | 肉
```

人吃饭，天下雨，人吃肉，天下雪，……

1. 图中4条语句就是4个产生式，也就是4种替换规则
2. Sent、S、V、O都是非终结符，而汉字都是终结符
3. 把非终结符不断替换，最终然结果只有终结符的过程称为文法推导

抽象语法树按什么规则生成？

抽象语法树按你自己定义好的上下文无关文法生成。

语法分析（Parsing）

大多数语法分析方法可以归为两类：自顶向下、自底向上

1. 自顶向下语法分析器：构造过程从根节点开始，逐步向叶子节点方向进行。

2. 自底向上语法分析器：构造过程从叶子节点开始，逐步构造出根节点。

递归下降算法（Recursive Descent Parsing）

递归下降算法是一种自顶向下的算法，自顶向下的算法可以比较容易手写出高效的语法分析器，而自底向上的塑封可以处理更多的文法，很多文法生成语法分析器工具更常使用自底向上的方法。

递归下降算法（Recursive Descent Parsing）


```
S -> N V N
N -> s
   | t
   | g
   | w
V -> e
   | d
```

文法

```
parse_S()
    parse_N()
    parse_V()
    parse_N()

parse_N()
    token = tokens[i++]
    if(token == s || token == t || token == g || token == w)
        return;
    error("...")

parse_V()
    token = tokens[i++]
    ... //
```

伪代码

通过什么算法生成抽象语法树？

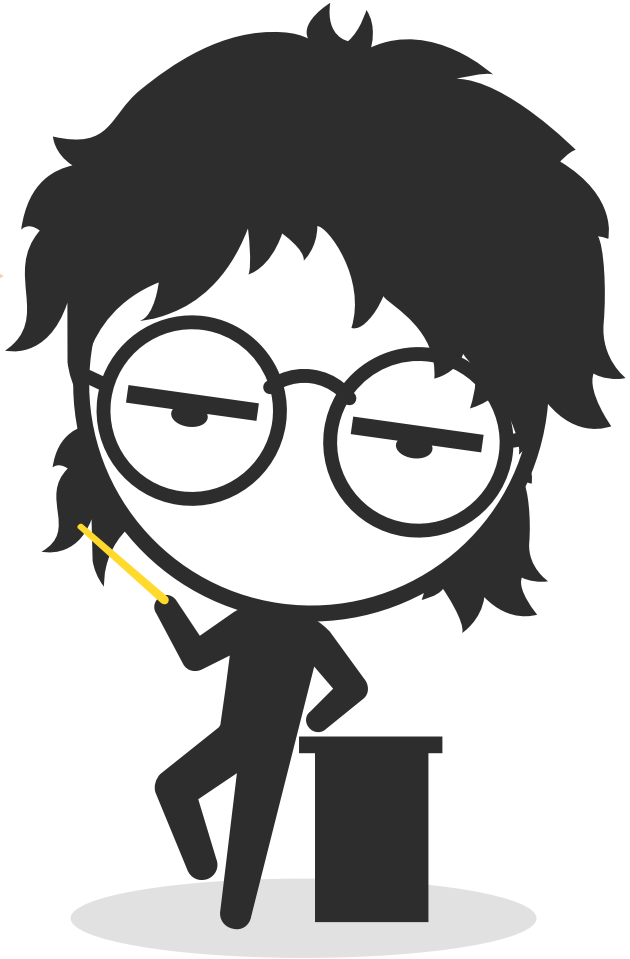
使用递归下降算法来生成抽象语法树。

如何使用生成的抽象语法树？

使用深度优先算法对抽象语法树进行前序遍历。

语义分析（Semantic Analysis）

语义分析会使用语法树和符号表中的信息来检查源程序是否和语言定义的语义一致，简单而言，就是消除具有二义性的部分，让计算机明白我们的真实意图。

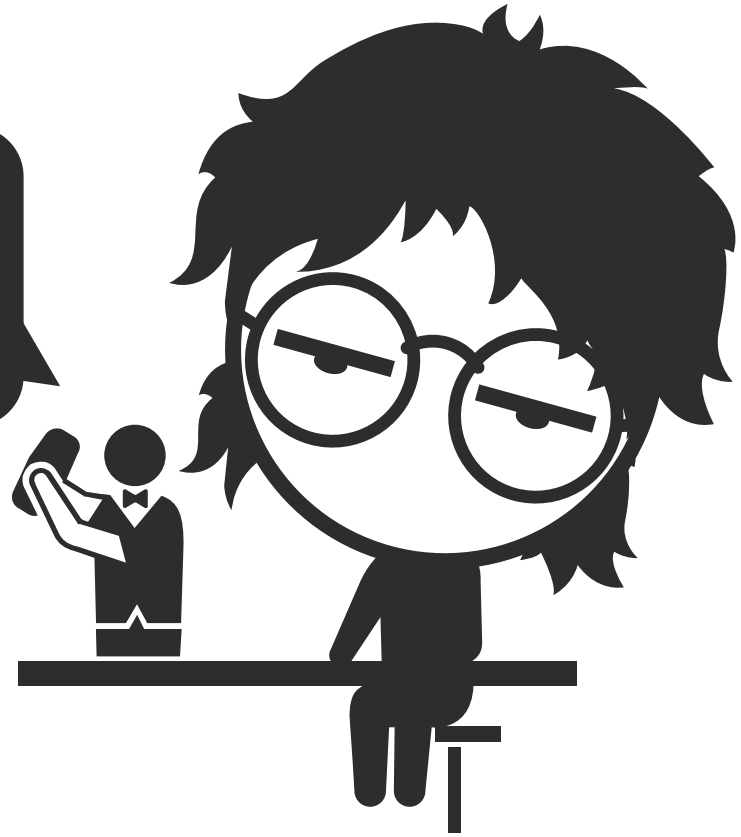


语义分析（Semantic Analysis）

1. 类型检查，比如很多编译语言都要求数组下标是整数
2. 同名变量判断，比如在不同作用域下，使用哪个变量
3. 类型自动转换，比如浮点型和整数一同运算是将该整数自动转为浮点型

符号表（symbol table）

那函数作用域这些要怎么实现？



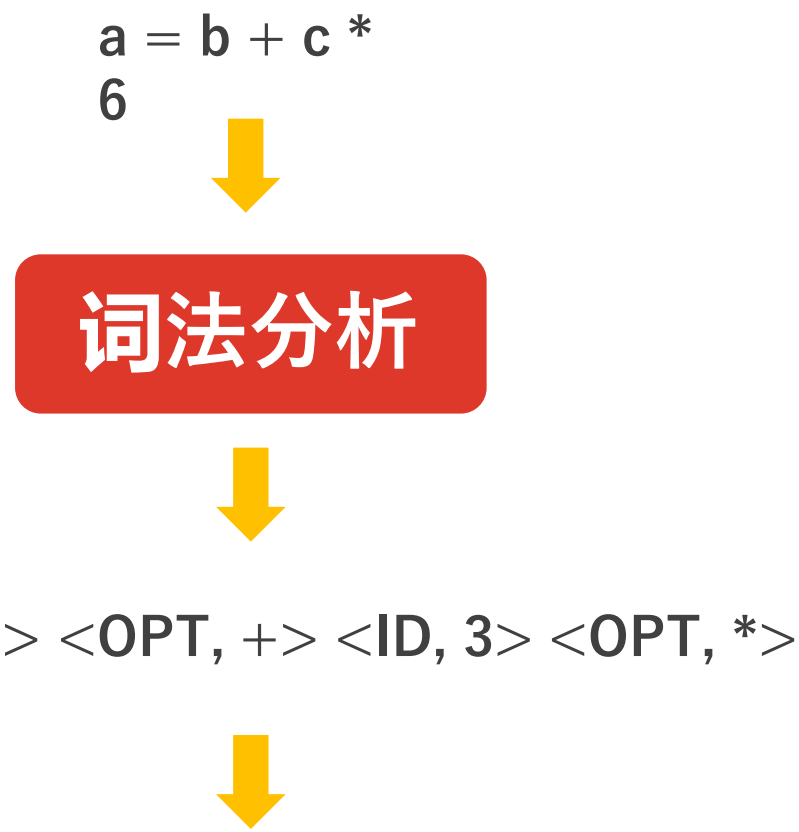
使用符号表便可实现函数的作用域，我们可以为每个作用域设置一个符号表，符号表之间关联起来实现作用域的嵌套关系。

符号表（symbol table）

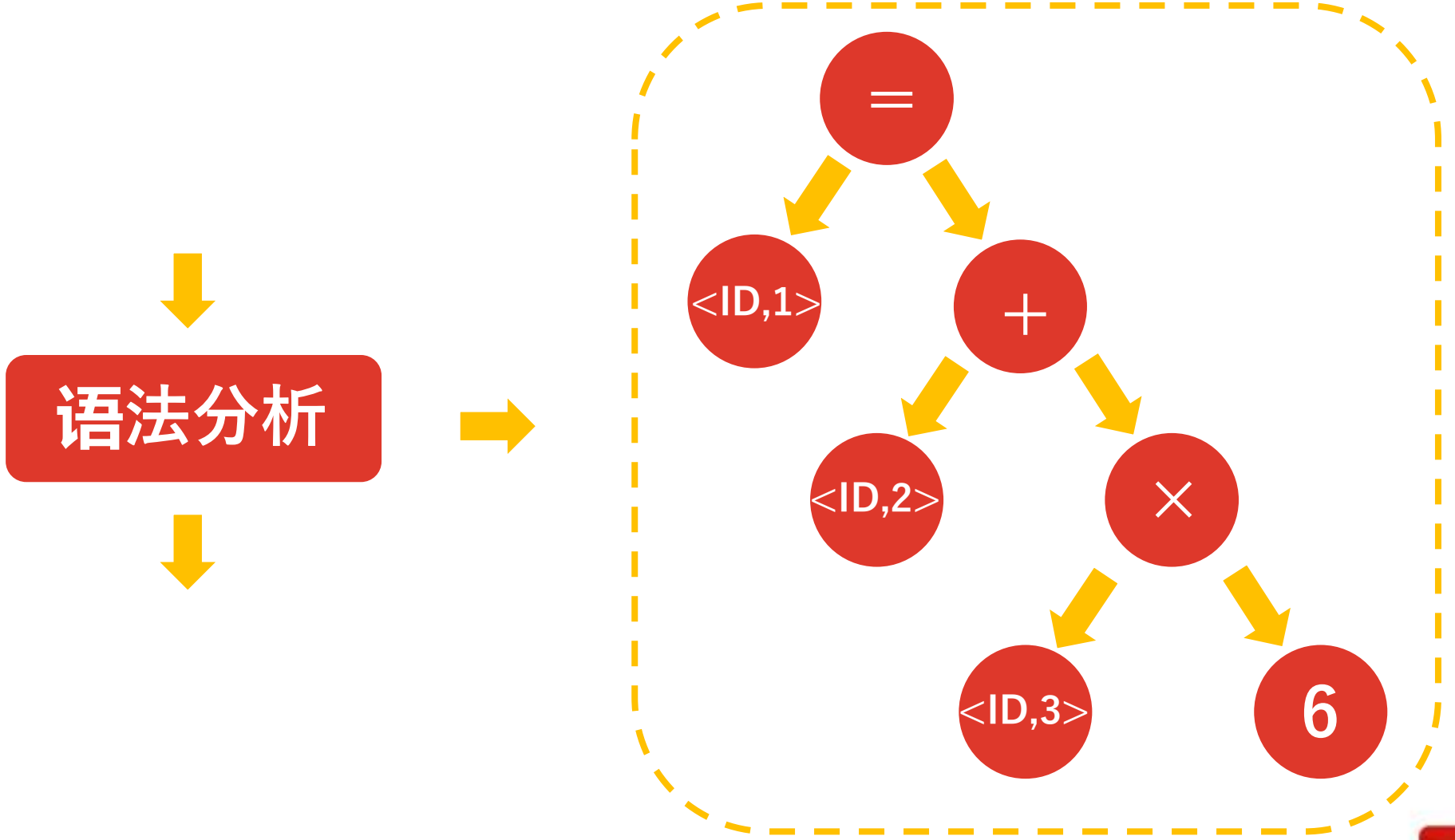
符号表是一种供编译器保存有关源程序各种信息的数据结构。符号表的每个条目都包含一个标识符相关的信息，如它的类型、它的存储位置等等。

词法分析（lexical analysis）

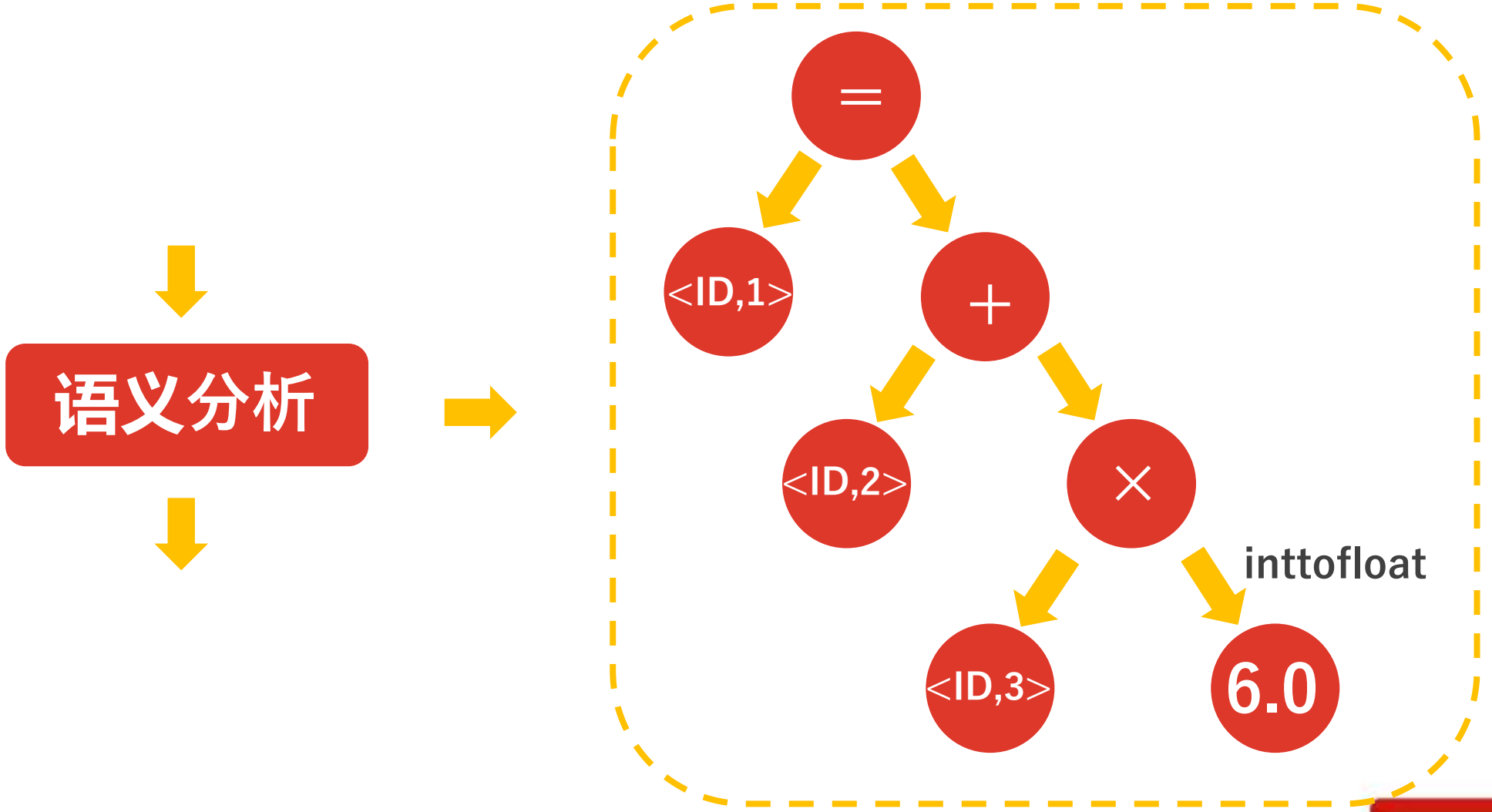
id	name	value
1	a	10.0
2	b	20.0
3	c	30.0



语法分析（Parsing）

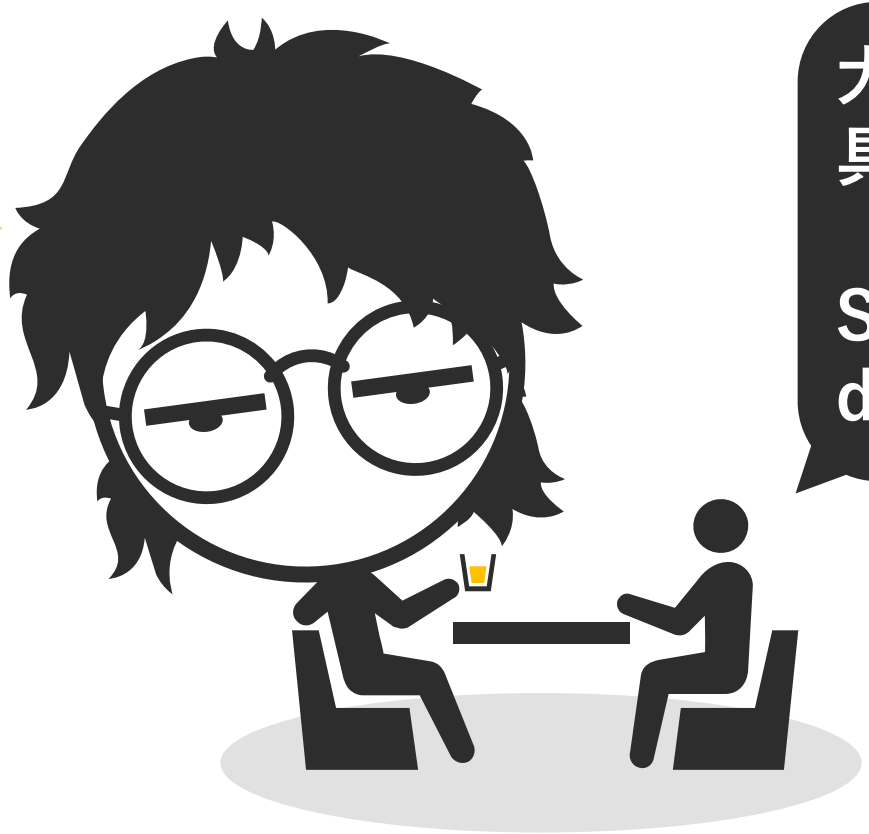


语义分析（Semantic Analysis）



Show me the code!

下面我们就直接来看ToyPL项目代码来直接感受一下吧



大致的原理我懂了，但具体要怎么实现呢？

Show me the code, don't BB to much.

Thank You

个人微信

工作微信