

词法分析

(1. 词法分析器生成器 ANTLR4)

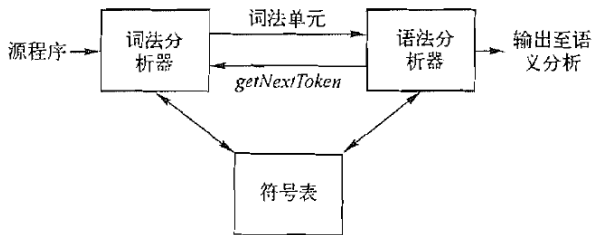
魏恒峰

hfwei@nju.edu.cn

2021 年 11 月 05 日 (周五)



输入: 程序文本/字符串 s & **词法单元 (token) 的规约**



输出: 词法单元流

```
CharStream input = CharStreams.fromStream(is);  
SimpleExprLexer lexer = new SimpleExprLexer(input);  
CommonTokenStream tokens = new CommonTokenStream(lexer);  
SimpleExprParser parser = new SimpleExprParser(tokens);  
ParseTree tree = parser.prog();
```

SimpleExpr.g4

```
int main(void)
{
    printf("hello, world\n");
}
```

```
int main(void)
{
    printf("hello, world\n");
}
```

int WS **main/id** **LP** void **RP** WS

LB WS

WS id LP literal RP SC WS

RB

```
int main(void)
{
    printf("hello, world\n");
}
```

int WS **main/id** **LP** void **RP** WS

LB WS

WS id LP literal RP SC WS

RB

本质上, 这就是一个**字符串 (匹配/识别) 算法**

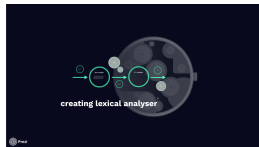
词法分析器的三种设计方法 (由易到难)



词法分析器生成器



手写词法分析器



自动化词法分析器

生产环境下的编译器 (如 gcc) 通常选择**手写词法分析器**



master

[gcc / gcc / c-family / c-lex.c](#)

ians Objective-C/C++ : Improve '@' keyword locations. ...

19 contributors

1435 lines (1278 sloc) | 38.8 KB

master

[gcc / libcpp / lex.c](#)

urnathan cpplib: EOF in pragmas ...

25 contributors

4364 lines (3825 sloc) | 119 KB



词法分析器生成器

输入: 词法单元的规约

antlr4 SimpleExpr.g4

输入: 词法单元的规约

antlr4 SimpleExpr.g4

输出: 词法分析器

- ▶ SimpleExprLexer.java
- ▶ SimpleExpr.tokens

输入: 词法单元的规约

```
antlr4 SimpleExpr.g4
```

输出: 词法分析器

▶ SimpleExprLexer.java

▶ SimpleExpr.tokens

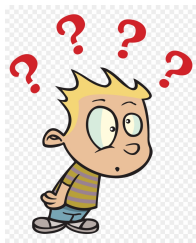
```
javac SimpleExpr*.java
```

```
grun simpleexpr.SimpleExpr prog -tokens
```

词法单元的规约

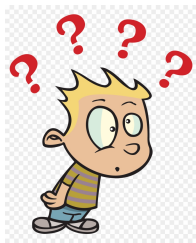
词法单元	非正式描述	词素示例
if	字符 i, f	if
else	字符 e, l, s, e	else
comparison	< 或 > 或 <= 或 >= 或 == 或 !=	<=, !=
id	字母开头的字母 / 数字串	pi, score, D2
number	任何数字常量	3.14159, 0, 6.02e23
literal	在两个 "之间, 除" 以外的任何字符	"core dumped"

词法单元的规约



词法单元	非正式描述	词素示例
if	字符 i, f	if
else	字符 e, l, s, e	else
comparison	< 或 > 或 <= 或 >= 或 == 或 !=	<=, !=
id	字母开头的字母 / 数字串	pi, score, D2
number	任何数字常量	3.14159, 0, 6.02e23
literal	在两个 "之间, 除" 以外的任何字符	"core dumped"

词法单元的规约



词法单元	非正式描述	词素示例
if	字符 i, f	if
else	字符 e, l, s, e	else
comparison	< 或 > 或 <= 或 >= 或 == 或 !=	<=, !=
id	字母开头的字母 / 数字串	pi, score, D2
number	任何数字常量	3.14159, 0, 6.02e23
literal	在两个 "之间, 除" 以外的任何字符	"core dumped"

我们需要词法单元的**形式化**规约

id: 字母开头的字母/数字串

id 定义了一个集合, 我们称之为**语言 (Language)**

它使用了字母与数字等符号集合, 我们称之为**字母表 (Alphabet)**

该语言中的每个元素 (即, 标识符) 称为**串 (String)**

Definition (字母表)

字母表 Σ 是一个有限的符号集合。



Definition (串)

字母表 Σ 上的**串** (s) 是由 Σ 中符号构成的一个**有穷**序列。

ϵ

空串 : $|\epsilon| = 0$

Definition (串上的“连接”运算)

$$x = \text{dog}, y = \text{house} \quad xy = \text{doghouse}$$

$$s\epsilon = \epsilon s = s$$

Definition (串上的“连接”运算)

$$x = \text{dog}, y = \text{house} \quad xy = \text{doghouse}$$

$$s\epsilon = \epsilon s = s$$

Definition (串上的“指数”运算)

$$s^0 \triangleq \epsilon$$

$$s^i \triangleq ss^{i-1}, i > 0$$

Definition (语言)

语言是给定字母表 Σ 上一个任意的**可数的**串集合。

$$\emptyset$$
$$\{\epsilon\}$$

Definition (语言)

语言是给定字母表 Σ 上一个任意的**可数的**串集合。

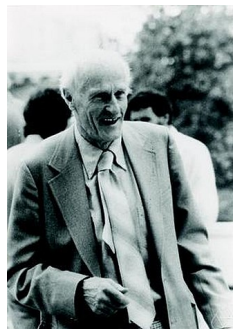
$$\emptyset$$
$$\{\epsilon\}$$
$$\text{id} : \{a, b, c, a1, a2, \dots\}$$
$$\text{ws} : \{\text{blank}, \text{tab}, \text{newline}\}$$
$$\text{if} : \{\text{if}\}$$

语言是串的集合

因此, 我们可以通过集合操作**构造**新的语言。

运算	定义和表示
L 和 M 的并	$L \cup M = \{s \mid s \text{ 属于 } L \text{ 或者 } s \text{ 属于 } M\}$
L 和 M 的连接	$LM = \{st \mid s \text{ 属于 } L \text{ 且 } t \text{ 属于 } M\}$
L 的 Kleene 闭包	$L^* = \bigcup_{i=0}^{\infty} L^i$
L 的正闭包	$L^+ = \bigcup_{i=1}^{\infty} L^i$

L^* (L^+) 允许我们构造**无穷**集合



Stephen Kleene
(1909 ~ 1994)

$$L = \{A, B, \dots, Z, a, b, \dots, z\}$$

$$D = \{0, 1, \dots, 9\}$$

运算	定义和表示
L 和 M 的并	$L \cup M \doteq \{s \mid s \text{ 属于 } L \text{ 或者 } s \text{ 属于 } M\}$
L 和 M 的连接	$LM = \{st \mid s \text{ 属于 } L \text{ 且 } t \text{ 属于 } M\}$
L 的 Kleene 闭包	$L^* = \bigcup_{i=0}^{\infty} L^i$
L 的正闭包	$L^+ = \bigcup_{i=1}^{\infty} L^i$

$$L = \{A, B, \dots, Z, a, b, \dots, z\}$$

$$D = \{0, 1, \dots, 9\}$$

运算	定义和表示
L 和 M 的并	$L \cup M \doteq \{s \mid s \text{ 属于 } L \text{ 或者 } s \text{ 属于 } M\}$
L 和 M 的连接	$LM = \{st \mid s \text{ 属于 } L \text{ 且 } t \text{ 属于 } M\}$
L 的 Kleene 闭包	$L^* = \bigcup_{i=0}^{\infty} L^i$
L 的正闭包	$L^+ = \bigcup_{i=1}^{\infty} L^i$

$$L \cup D \quad LD \quad L^4 \quad L^* \quad D^+$$

$$L(L \cup D)^*$$

$$L = \{A, B, \dots, Z, a, b, \dots, z\}$$

$$D = \{0, 1, \dots, 9\}$$

运算	定义和表示
L 和 M 的并	$L \cup M \doteq \{s \mid s \text{ 属于 } L \text{ 或者 } s \text{ 属于 } M\}$
L 和 M 的连接	$LM = \{st \mid s \text{ 属于 } L \text{ 且 } t \text{ 属于 } M\}$
L 的 Kleene 闭包	$L^* = \bigcup_{i=0}^{\infty} L^i$
L 的正闭包	$L^+ = \bigcup_{i=1}^{\infty} L^i$

$$L \cup D \quad LD \quad L^4 \quad L^* \quad D^+$$

$$L(L \cup D)^* \quad \text{标识符}$$

id : $L(L \cup D)^*$

该如何告诉 antlr4: 这个集合就是 **id** 呢?

$\text{id} : L(L \cup D)^*$

该如何告诉 antlr4: 这个集合就是 **id** 呢?



下面向大家隆重介绍简洁、优雅、强大的**正则表达式**

每个正则表达式 r 对应一个正则语言 $L(r)$



正则表达式是**语法**, 正则语言是**语义**

Definition (正则表达式)

给定字母表 Σ , Σ 上的正则表达式由且仅由以下规则定义:

- (1) ϵ 是正则表达式;
- (2) $\forall a \in \Sigma$, a 是正则表达式;
- (3) 如果 r 是正则表达式, 则 (r) 是正则表达式;
- (4) 如果 r 与 s 是正则表达式, 则 $r|s$, rs , r^* 也是正则表达式。

运算优先级: $() \succ * \succ \text{连接} \succ |$

$$(a)|((b)^*(c)) \equiv a|b^*c$$

每个正则表达式 r 对应一个正则语言 $L(r)$

Definition (正则表达式对应的正则语言)

$$L(\epsilon) = \{\epsilon\} \quad (1)$$

$$L(a) = \{a\}, \forall a \in \Sigma \quad (2)$$

$$L((r)) = L(r) \quad (3)$$

$$L(r|s) = L(r) \cup L(s) \quad L(rs) = L(r)L(s) \quad L(r^*) = (L(r))^* \quad (4)$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$L((a|b)^*)$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$L((a|b)^*)$$

$$L(a|a^*b)$$

So
EASY

表达式	匹配	例子
c	单个非运算符字符 c	a
$\backslash c$	字符 c 的字面值	$\backslash *$
$"s"$	串 s 的字面值	$"**"$
$.$	除换行符以外的任何字符	$a.*b$
$^$	一行的开始	abc
$\$$	行的结尾	$abc\$$
$[s]$	字符串 s 中的任何一个字符	$[abc]$
$[^s]$	不在串 s 中的任何一个字符	$[^abc]$
r^*	和 r 匹配的零个或多个串连接成的串	a^*
r^+	和 r 匹配的一个或多个串连接成的串	a^+
$r^?$	零个或一个 r	$a^?$
$r\{m,n\}$	最少 m 个, 最多 n 个 r 的重复出现	$a\{1,5\}$
r_1r_2	r_1 后加上 r_2	ab
$r_1 r_2$	r_1 或 r_2	$a b$
(r)	与 r 相同	$(a b)$
r_1/r_2	后面跟有 r_2 时的 r_1	$abc/123$

表达式	匹配	例子
c	单个非运算符字符 c	a
$\backslash c$	字符 c 的字面值	$\backslash *$
$"s"$	串 s 的字面值	$"**"$
$.$	除换行符以外的任何字符	$a.*b$
$^$	一行的开始	abc
$\$$	行的结尾	$abc\$$
$[s]$	字符串 s 中的任何一个字符	$[abc]$
$[^s]$	不在串 s 中的任何一个字符	$[^abc]$
r^*	和 r 匹配的零个或多个串连接成的串	a^*
r^+	和 r 匹配的一个或多个串连接成的串	a^+
$r^?$	零个或一个 r	$a^?$
$r\{m, n\}$	最少 m 个, 最多 n 个 r 的重复出现	$a\{1, 5\}$
r_1r_2	r_1 后加上 r_2	ab
$r_1 r_2$	r_1 或 r_2	$a b$
(r)	与 r 相同	$(a b)$
r_1/r_2	后面跟有 r_2 时的 r_1	$abc/123$

$[0 - 9] \quad [a - zA - Z]$

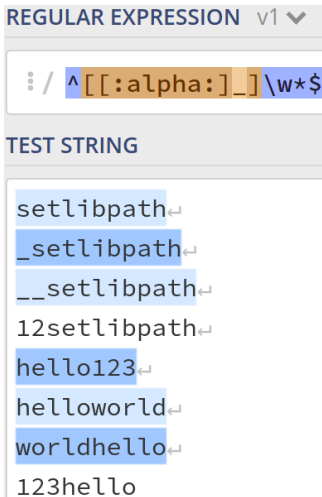
$^ \quad \$$

正则定义与简记法

Vim ↕	Java ↕	ASCII ↕	Description ↕
	\p{ASCII}	[\x00-\x7F]	ASCII characters
	\p{Alnum}	[A-Za-z0-9]	Alphanumeric characters
\w	\w	[A-Za-z0-9_]	Alphanumeric characters plus "_"
\W	\W	[^A-Za-z0-9_]	Non-word characters
\a	\p{Alpha}	[A-Za-z]	Alphabetic characters
\s	\p{Blank}	[\t]	Space and tab
\< \>	\b	(?<=\W) (?=\w) (?<=\w) (?=\W)	Word boundaries
	\B	(?<=\W) (?=\W) (?<=\w) (?=\w)	Non-word boundaries
	\p{Cntrl}	[\x00-\x1F\x7F]	Control characters
\d	\p{Digit} or \d	[0-9]	Digits
\D	\D	[^0-9]	Non-digits
	\p{Graph}	[\x21-\x7E]	Visible characters
\l	\p{Lower}	[a-z]	Lowercase letters
\p	\p{Print}	[\x20-\x7E]	Visible characters and the space character
	\p{Punct}	[] [!"#%&'()*+,-./:;<=>?@^_`{ }~ -]	Punctuation characters
\s	\p{Space} or \s	[\t\r\n\v\f]	Whitespace characters
\S	\S	[^ \t\r\n\v\f]	Non-whitespace characters
\u	\p{Upper}	[A-Z]	Uppercase letters
\x	\p{XDigit}	[A-Fa-f0-9]	Hexadecimal digits

C 语言中的标识符?

C 语言中的标识符?



<https://regex101.com/r/lVoghd/1>

C 语言中**单行注释**对应的正则表达式?

C 语言中**单行注释**对应的正则表达式?

REGULAR EXPRESSION v1 ▾

`// \/\\/.*`

TEST STRING

```
// this is a comment
int main() { // this is a comment
//
} // this is a comment
```

<https://regex101.com/r/ED4qgC/2>

如何满足 L1 关于多行注释的要求?

一种是使用 “/*” 以及 “*/” 进行多行注释，在这种情况下，在 “/*” 与之后最先遇到的 “*/” 之间的所有字符都被视作注释内容。需要注意的是，“/*” 与 “*/” 是不允许嵌套的：即在任意一对 “/*” 和 “*/” 之间不能再包含成对的 “/*” 和 “*/”，否则编译器需要进行报错。

如何满足 L1 关于多行注释的要求?

一种是使用 “/*” 以及 “*/” 进行多行注释，在这种情况下，在 “/*” 与之后最先遇到的 “*/” 之间的所有字符都被视作注释内容。需要注意的是，“/*” 与 “*/” 是不允许嵌套的：即在任意一对 “/*” 和 “*/” 之间不能再包含成对的 “/*” 和 “*/”，否则编译器需要进行报错。

```
LCOMMENT : '//' . * ? '\n' -> skip;    // non-greedy
MLCOMMENT : '/*' . * ? '*/' -> skip;    // non-greedy
```

如何满足 L1 关于多行注释的要求?

一种是使用 “/*” 以及 “*/” 进行多行注释, 在这种情况下, 在 “/*” 与之后最先遇到的 “*/” 之间的所有字符都被视作注释内容。需要注意的是, “/*” 与 “*/” 是不允许嵌套的: 即在任意一对 “/*” 和 “*/” 之间不能再包含成对的 “/*” 和 “*/”, 否则编译器需要进行报错。

```
LCOMMENT : '//' ..*? '\n' -> skip;    // non-greedy
MLCOMMENT : '/*' ..*? '*/' -> skip;    // non-greedy
```

在 antlr4 中, '.' 可匹配任意字符 (包括换行符)

$$\left(0|(1(01^*0)^*1)\right)^*$$



<https://regex101.com/r/ED4qgC/1>

REGULAR EXPRESSION v1 ▾

```
/^(0|1(01*0)*1)*$/
```

TEST STRING

0

1

10

11

100

101

110

111

1000

1001

1010

1011

1100

1101

1110

1111

10000

10001

10010

10011

10100

10101

10110

REGULAR EXPRESSION v1 ▾

/^(0|(1(01*0)*1))*\$/

TEST STRING

0

1

10

11

100

101

110

111

1000

1001

1010

1011

1100

1101

1110

1111

10000

10001

10010

10011

10100

10101

10110

3 的倍数 (二进制表示)

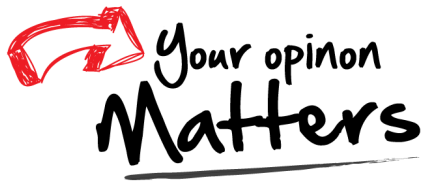
```
IF : 'if' ;  
ID : (LETTER | '_' ) WORD* ;  
INT : DIGIT | (NONZERODIGIT DIGIT+) ;  
WS : [ \t\r\n ]+ -> skip ;  
  
ASSIGN : '=' ;  
LBRACE : '(' ;  
RBRACE : ')' ;  
ADD : '+' ;  
SUB : '-' ;  
MUL : '*' ;  
DIV : '/' ;
```

Antlr4 中两大冲突解决规则

最前优先匹配: 关键字 *vs.* 标识符

最长优先匹配: \geq , ifhappy, thenext, 1.23

Thank
You!



Office 926

hfwei@nju.edu.cn