

Lexer mit ANTLR generieren

Carsten Gips (HSBI)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

Hello World

```
grammar Hello;

start          : 'hello' GREETING ;

GREETING       : [a-zA-Z]+ ;
WHITESPACE     : [ \t\n]+ -> skip ;
```

Konsole: Hello (Classpath, Aliase, grun, Main, Dateien, Ausgabe)

Verhalten des Lexers: 1. Längster Match

Primäres Ziel: Erkennen der längsten Zeichenkette

```
CHARS   : [a-z]+ ;  
DIGITS  : [0-9]+ ;  
FOO     : [a-z]+ [0-9]+ ;
```

Verhalten des Lexers: 2. Reihenfolge

Reihenfolge in Grammatik definiert Priorität

```
F00      : 'f' .*? 'r' ;  
BAR      : 'foo' .*? 'bar' ;
```

Verhalten des Lexers: 3. Non-greedy Regeln

Non-greedy Regeln versuchen *so wenig* Zeichen wie möglich zu matchen

```
FOO      : 'foo' .*? 'bar' ;  
BAR      : 'bar' ;
```

Verhalten des Lexers: 3. Non-greedy Regeln

Non-greedy Regeln versuchen *so wenig* Zeichen wie möglich zu matchen

```
FOO      : 'foo' .*? 'bar' ;  
BAR      : 'bar' ;
```

Achtung: Nach einer non-greedy Sub-Regel gilt “*first match wins*”

```
.*? ('4' | '42')
```

=> '42' ist “toter Code” (wegen der non-greedy Sub-Regel `.*?`)!

Attribute und Aktionen

```
grammar Demo;

@header {
import java.util.*;
}

@members {
String s = "";
}

start      : TYPE ID '=' INT ';' ;

TYPE       : ('int' | 'float') {s = getText();} ;
INT        : [0-9]+             {System.out.println(s+" "+Integer.valueOf(getText()));};
ID         : [a-z]+             {setText(String.valueOf(getText().charAt(0)));} ;
WS         : [ \t\n]+ -> skip ;
```

Hilfsregeln mit Fragmenten

```
NUM      : DIGIT+ ;
```

```
fragment
```

```
DIGIT    : [0-9] ;
```

=> Keine Token (für den Parser)!

Lexer Kommandos (Auswahl)

TokenName : Alternative -> command-name

- skip
- more
- mode
- channel

Modes und Insel-Grammatiken

```
lexer grammar ModeLexer;  
  
LCOMMENT      : '/*' -> more, mode(CMNT) ;  
WS            : [ \t\n]+ -> skip ;  
  
mode CMNT;  
COMMENT       : '*/' -> mode(DEFAULT_MODE) ;  
CHAR          : . -> more ;
```

Channels

```
channels { WHITESPACE, COMMENTS }
```

```
BLOCK_COMMENT : '/*' .*? '*/' -> channel(COMMENTS) ;
```

```
LINE_COMMENT  : '//' ~[\n]* -> channel(COMMENTS) ;
```

```
WS            : [ \t\n]+ -> channel(WHITESPACE) ;
```

Lexer mit ANTLR generieren: Lexer-Regeln werden mit **Großbuchstaben** geschrieben

- Längster Match gewinnt, Gleichstand: zuerst definierte Regel
- *non greedy*-Regeln: versuche so *wenig* Zeichen zu matchen wie möglich
- Aktionen beim Matchen
- Hilfsregeln mit “Fragments”
- Lexer Kommandos: `skip`, `more`, ...
- Modes für Insel-Grammatiken
- Channels als parallele Tokenstreams (Vorsortieren)
- Teilgrammatiken importieren

LICENSE



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.