ANTLR: **AN**other **T**ool for **L**anguage **R**ecognition

https://github.com/az82/antlr-demo

Andreas Zitzelsberger, 2017

Teile aus https://github.com/nielsutrecht/antlr-pres von Niels Dommerholt unter der MIT License übernommen.

xmlStr.find(userRegex)

```
if(nextToken != null && nextToken.equals(„/>“) {
        ...
} else if (...) {
        ...
```

Was kann ANTLR?

ANTLR ist ein LL(*)-Parser

- **L**eft-to-right
- **Leftmost** Derivation
- *-Lookahead

Bildquelle:
Pattern perception and computational complexity: introduction to the special issue
W. Tecumseh Fitch, Angela D. Friederici, Peter Hagoort
Philosophical Transactions of the Royal Society, Volume 367, issue 1598, 19 July 2012

Wie nutze ich ANTLR?

Nicht mehr ab ANTLR 4

Wie baue ich einen Parser in ANTLR?

ANTLR-Grammatiken

- Eine Grammatik ist eine Menge von Textdateien die eine formale Beschreibung einer formalen bzw. strukturierten Sprache enthalten.
- Grammatiken werden üblicherweise in EBNF (Extended Backus-Naur Form) ausgedrückt.
- ANTLR 4 hat seinen eigenen EBNF-Dialekt.

Eine einfache Beispiel-Grammatik

```
grammar Hello;

greeting: HELLO name;
name:     STRING;

HELLO:    'hello';
STRING:   CHAR+;
fragment
CHAR: [0-9a-zA-Z];
```

Parser-Regeln

Lexer-Regeln

Parsing mit der Beispiel-Grammatik

Der Taschenrechner

```
grammar Calc;

calc: expr EOF;

expr
:BR_OPEN expr BR_CLOSE
|expr TIMES   expr
|expr DIV     expr
|expr PLUS    expr
|expr MINUS   expr
|number
;

number: NUMBER;
```

```
PLUS:  '+';
MINUS: '-';
TIMES: '*';
DIV:   '/';

NUMBER: '-'? [0-9]+;
BR_OPEN: '(';
BR_CLOSE: ')';

WS: [ \t\r\n]+ -> skip;
```

Parsing mit der Taschenrechner-Grammatik

```
calc: expr EOF;

expr
:BR_OPEN expr BR_CLOSE
|expr TIMES   expr
|expr DIV     expr
|expr PLUS    expr
|expr MINUS   expr
|number
;

number: NUMBER;
```

“(1 + 2) * 5 - 10 / 2”

Visitor

```
@Override
public Integer visitExpr(CalcParser.ExprContext ctx) {
    if (ctx.PLUS() != null) {
        return lhs(ctx) + rhs(ctx);
    } else if (ctx.MINUS() != null) {
        return lhs(ctx) - rhs(ctx);
    } else if (ctx.TIMES() != null) {
        return lhs(ctx) * rhs(ctx);
    } else if (ctx.DIV() != null) {
        return lhs(ctx) / rhs(ctx);
    } else {
        return super.visitExpr(ctx);
    }
}

@Override
public Integer visitNumber(CalcParser.NumberContext ctx) {
    return Integer.parseInt(ctx.getText());
}

@Override
protected Integer aggregateResult(Integer aggregate, Integer nextResult) {
    return aggregate != null ? aggregate : nextResult;
}
```

Listener

```
@Override
public void exitExpr(CalcParser.ExprContext ctx) {
    if (ctx.PLUS() != null) {
        operands.push(operands.pop() + operands.pop());
    } else if (ctx.MINUS() != null) {
        operands.push(operands.pop() - operands.pop());
    } else if (ctx.TIMES() != null) {
        operands.push(operands.pop() * operands.pop());
    } else if (ctx.DIV() != null) {
        operands.push(operands.pop() / operands.pop());
    }
}

@Override
public void enterNumber(CalcParser.NumberContext ctx) {
    operands.push(Integer.valueOf(ctx.getText()));
}
```

Parse Tree Walking

```
private static Integer eval(CalcParser.ExprContext ctx) {
    if (ctx.PLUS() != null) {
        return evalLeft(ctx) + evalRight(ctx);
    } else if (ctx.MINUS() != null) {
        return evalLeft(ctx) - evalRight(ctx);
    } else if (ctx.TIMES() != null) {
        return evalLeft(ctx) * evalRight(ctx);
    } else if (ctx.DIV() != null) {
        return evalLeft(ctx) / evalRight(ctx);
    } else if (ctx.number() != null) {
        return Integer.valueOf(ctx.getChild(0).getText());
    } else if (ctx.BR_OPEN() != null) {
        return eval((CalcParser.ExprContext) ctx.getChild(1));
    } else {
        throw new IllegalStateException(format("Unexpected expression variant in \"%s\"", ctx.getText()));
    }
}

private static Integer evalLeft(CalcParser.ExprContext ctx){
    return eval((CalcParser.ExprContext) ctx.getChild(0));
}

private static Integer evalRight(CalcParser.ExprContext ctx){
    return eval((CalcParser.ExprContext) ctx.getChild(2));
}
```

XPath

```
return XPath.findAll(parser.calc(), "//number", parser).stream()
        .map(ParseTree::getText)
        .collect(toList());
```

Code

https://github.com/az82/antlr-demo/tree/hands-on

Vorteile und Pitfalls

Vorteile	Pitfalls

▪ Vergleichsweise einfach zu nutzen
▪ Klare Fehlermeldungen
▪ Plugins für IntelliJ und Eclipse
▪ Plugins für Gradle und Maven
▪ Kürzere Implementierungszeit als einen eigenen Parser zu schreiben
▪ Garantierte Korrektheit in Bezug auf die Grammatik
	▪ Lexing und Parsing findet in unterschiedlichen Schritten statt
	  ▪ Parser arbeitet auf dem Token stream vom Lexer
	  ▪ Lexer sieht nur Bytemuster und arbeitet greedy
	▪ Die Reihenfolge der Regeln ist wichtig
	  ▪ Der Parser verarbeitet Regeln in der Reihenfolge ihres Auftretens
	  ▪ ANTLR kann üblicherweise linksrekursive Regeln auflösen, aber nicht immer (expr: expr op expr)

Alternativen zu ANTLR

- YACC / GNU Bison
- lex / flex
- JFlex
- JavaCC

Mehr Informationen

- ANTLR Doku: https://github.com/antlr/antlr4/blob/master/doc/index.md
- The Definitive ANTLR 4 Reference: https://pragprog.com/book/tpantlr2/the-definitive-antlr-4-reference
- Plugins für IntelliJ und Eclipse
- Maven und Gradle-Plugins

Danke!

Fragen?