Parser mit ANTLR generieren

Carsten Gips (HSBI)

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

Hello World

```
grammar Hello;

start : stmt* ;

stmt  : ID '=' expr ';' | expr ';' ;

expr  : term ('+' term)* ;
term  : atom ('*' atom)* ;

atom  : ID | NUM ;

ID    : [a-z][a-zA-Z]* ;
NUM   : [0-9]+ ;
WS    : [ \t\n]+ -> skip ;
```

Konsole: Hello (grun, Parse-Tree)

Expressions und Vorrang (Operatoren)

```
expr  : term ('+' term)* ;
term  : atom ('*' atom)* ;
atom  : ID ;
```

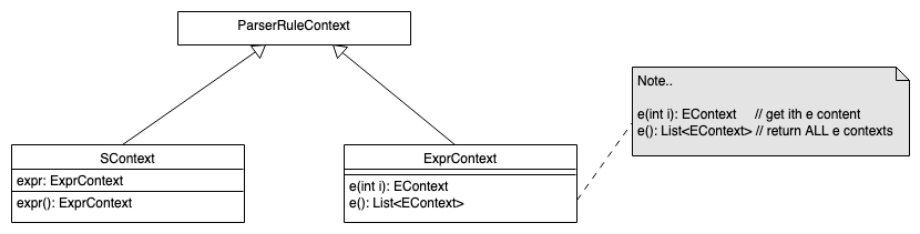
Expressions und Vorrang (Operatoren)

```
expr  : term ('+' term)* ;
term  : atom ('*' atom)* ;
atom  : ID ;
```

```
expr  : expr '*' expr
      | expr '+' expr
      | ID
      ;
```

Kontext-Objekte für Parser-Regeln

```
s    : expr          {List<EContext> x = $expr.ctx.e();}
     ;
expr : e '*' e ;
```



Benannte Regel-Elemente oder Alternativen

```
stat  : 'return' value=e ';'    # Return
      | 'break' ';'             # Break
      ;
```

```
public static class StatContext extends ParserRuleContext { ... }
public static class ReturnContext extends StatContext {
    public EContext value;
    public EContext e() { ... }
}
public static class BreakContext extends StatContext { ... }
```

Arbeiten mit ANTLR-Listeners

```
expr : e1=expr '*' e2=expr      # MULT
     | e1=expr '+' e2=expr      # ADD
     | DIGIT                    # ZAHL
     ;
```

```
public static class MyListener extends calcBaseListener {
    public void exitMULT(calcParser.MULTContext ctx) {
        ...
    }
    public void exitADD(calcParser.ADDContext ctx) {
        ...
    }
    public void exitZAHL(calcParser.ZAHLContext ctx) {
        ...
    }
}
```

Arbeiten mit dem Visitor-Pattern

```
expr : e1=expr '*' e2=expr      # MULT
     | e1=expr '+' e2=expr      # ADD
     | DIGIT                    # ZAHL
     ;
```

```
public static class MyVisitor extends calcBaseVisitor<Integer> {
    public Integer visitMULT(calcParser.MULTContext ctx) {
        return ...
    }
    public Integer visitADD(calcParser.ADDContext ctx) {
        return ...
    }
    public Integer visitZAHL(calcParser.ZAHLContext ctx) {
        return ...
    }
}
```

Wrap-Up

Parser mit ANTLR generieren: Parser-Regeln werden mit **Kleinbuchstaben** geschrieben

- Regeln können Lexer- und Parser-Regeln "aufrufen"
- Regeln können Alternativen haben
- Bei Mehrdeutigkeit: Vorrang für erste Alternative
- ANTLR erlaubt direkte Links-Rekursion
- ANTLR erzeugt Parse-Tree
- Benannte Alternativen und Regel-Elemente
- Traversierung des Parse-Tree: Listener oder Visitoren, Zugriff auf Kontextobjekte

LICENSE

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