JavaCC

Dominik Oepen

John-von-Neumann-Institut HU-Berlin

22.06.2006

Inhalt

Geschichte

- Java Compiler Compiler
- gegründet durch Sreeni Viswandha und Siriam Sanker bei Sun Microsystems
- gehörte zwischenzeitlich zu Metamata, Webgain
- wird jetzt wieder von Sunlabs betreut

- implementiert in JavaCC
- seit Juni 2003 Open Source (BSD Lizenz)
- aktuell Version 4.0

Eigenschaften

- LL(k) Parser
- Scanner und Parser in einer Datei
- EBNF
- Zielsprache: Java

Eigenschaften

- unterstützt Unicode
- generierter Quellcode sehr gut lesbar
- gute Fehlerausgabe/Debugging
- 100% pure Java

Tools

- Tool zur AST Erzeugung: JJtree
- Tool zur Dokumentation: JJDoc

JavaCC

JavaCC

Dateiaufbau

- Datei: `*.jj`
- JavaCC erzeugt `*.java` Dateien:
  - `MyParser.java`
  - `MyParserTokenManager.java`
  - `MyParserConstants.java`
  - vorgefertigte Klassen, die bei jedem Parser gleich sind

Dateiaufbau

Aufbau in EBNF:

```
options { ... }

PARSER_BEGIN( <IDENTIFIER> )
    java_compilation_unit
PARSER_END( <IDENTIFIER> )

( production )*
<EOF>
```

Scanner

4 Tokenklassen:

- `TOKEN`: Token wird normal an Parser weitergegeben
- `SKIP`: Token wird verworfen
- `MORE`: kein Token wird generiert, gematchter String Präfix des nächsten gematchten Strings
- `SPECIAL_TOKEN`: Token wird nicht explizit generiert, aber Zugriff in semantischen Aktionen

Scanner

- zusätzlich definierbare Zustände
- lexikalische Aktionen am Ende regulärer Ausdrücke
- `TOKEN_MGR_DECLS`

Lexikalische Regeln

```
<Zustand> Tokenklasse : "{"

regexpr ( "|" regexpr)*
"{" lexical_action "}"
[ ":" next_state ]

"}"
```

Parser

Produktionen:

```
void metasymbol():
{ javacode }
{ Produktionen + semantische Aktionen }
```

Lookahead

- fester Lookahead
- wird in Optionen gesetzt, Standard ist 1
- in den Produktionen lokaler Lookahead möglich, durch `LOOKAHEAD(x)`

Lookahead

Beispiel:

```
void Produktion()
{}
{ LOOKAHEAD(3) "R" "o" "c" "k"
  | "R" "o" "l" "l"
}
```

JJtree

JJtree

Eigenschaften

- Präprozessor für JavaCC
- Input: `*.jjt` Datei
- Output `*.jj` Datei + Klassendateien für Knoten

Funktionsweise

- AST wird bottom-up generiert
- Knoten werden auf Stack abgelegt

Funktionsweise

Zwei Modi: Simple und Multi
Simple:

- alle Knoten Instanzen von `SimpleNode.java`
- für jedes Nichtterminal wird ein Knoten erstellt

Funktionsweise

Multi:

- Option: `MULTI = true`
- für jedes Nichtterminal eigene Klasse abgeleitet von SimpleNode
- Präfix für Knotenklassen durch Option: `NODE_PREFIX`
- Benutzer legt fest welche Knoten in AST übernommen werden durch `'#'` nach Produktion
- umfangreiche Möglichkeiten den Stack zu manipulieren

Vorteile

- Sehr gut dokumentiert
- leichter Einstieg
- extrem flexibel

Nachteile

- nur LL(k)
- JJtree im Multi Mode unübersichtlich

Quellen

- offizielle Homepage: `https://javacc.dev.java.net/`
- Dokumentation:
  `https://javacc.dev.java.net/doc/docindex.html`
- FAQ: `http://www.engr.mun.ca/ theo/JavaCC-FAQ/`