Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Procesadores de Lenguajes

Una breve introducción a JavaCC

Grado en Ingeniería Informática

Especialidad de Computación

JavaCC

- ***Java Compiler Compiler***
  - https://javacc.github.io/javacc
- Es un **metacompilador**: un generador de analizadores **libre** para Java
  - También llamado *parsers*
- Genera código Java como salida
- Es una de las herramientas para la generación de analizadores más utilizada en Java

JavaCC™

JavaCC

- JavaCC genera analizadores **descendentes** recursivos
  - YACC genera analizadores **ascendentes**
    - Permite el uso de gramáticas más generales
- Por defecto, JavaCC genera un analizador LL (1)
  - Se puede modificar el comportamiento *look-ahead(k)*
- Incluye **JJTree**, un preprocesador de construcción de árboles muy potente
- Incluye **JJDoc**, una herramienta que convierte archivos de gramática en archivos de documentación (y HTML)

Powered by JavaCC

JavaCC is used in many commercial applications and open source projects.

The following list highlights a few notable JavaCC projects that run interesting use cases in production, with links to the relevant grammar specifications.

User	Use Case	Grammar File(s)
Apache ActiveMQ	Parsing JMS selector statements	SelectorParser.jj, HyphenatedParser.jj
Apache Avro	Parsing higher-level languages into Avro Schema	idl.jj
Apache Calcite	Parsing SQL statements	Parser.jj
Apache Camel	Parsing stored SQL templates	sspt.jj
Apache Jena	Parsing queries written in SPARQL, ARQ, SSE, Turtle and JSON	sparql_10, sparql_11, arq.jj, sse.jj, turtle.jj, json.jj
Apache Lucene	Parsing search queries	QueryParser.jj
Apache Tomcat	Parsing Expression Language (EL) and JSON	ELParser.jjt, JSONParser.jj
Apache Zookeeper	Optimising serialisation/deserialisation of Hadoop I/O records	rcc.jj
Java Parser	Parsing Java language files	java.jj

Funcionamiento

- Analiza un fichero de entrada (.jj) con la descripción de un lenguaje y una gramática
  - Especificación léxica
  - Especificación sintáctica
  - Acciones semánticas
    - código java que incrusta en lo generado

- Genera automáticamente un conjunto de clases Java que **implementan el analizador léxico y sintáctico** de la gramática especificada
  - Incluyendo las rutinas semánticas

Especificación del lenguaje

- La especificación léxica se basa en **expresiones regulares**
- La especificación sintáctica está basada en **EBNF**
  - *Extended Backus-Naur Form*
  - Extensión de BNF
    - (A) *, (A) + etc., dentro de las especificaciones léxicas y gramaticales
    - Alivia la necesidad de la recursividad izquierda en cierta medida
    - De hecho, el BNF extendido es a menudo más fácil de leer
      - *A ::= y (x)** vs *A ::= Ax | y*
  - Muy común en lenguajes de programación

- JavaCC no entiende de aspectos semánticos del lenguaje

Página del proyecto

- Página principal:
  - https://javacc.github.io/javacc
- Aspectos importantes:
  - https://javacc.github.io/javacc/tutorials/
- FAQ:
  - https://javacc.github.io/javacc/faq.html
- Gramáticas de ejemplo:
  - En los propios fuentes de JavaCC
  - https://javacc.github.io/javacc/tutorials/examples.html

Componentes de JavaCC

- **javacc**: generador de analizadores léxicos y sintácticos en Java

- **jjdoc:** generador de la documentación de la gramática en formato HTML de manera automática

- **jjtree:** preprocesador de apoyo para tareas semánticas
  - Permite generar automáticamente árboles sintácticos

- La espec … e cuatro s…
  - Opcior…
  - Identif…
  - Lexicog…
  - Sintaxi…

```
Options
{
     ...
}
//----------------------------------------
PARSER_BEGIN(ejemplo)
public class ejemplo {
    public static void main(String[] args) {
     ...
     }
}
PARSER_END(ejemplo)
//----------------------------------------
TOKEN_MGR_DECLS :
{
     ...
}
//----------------------------------------
TOKEN :
{
     ...
}
//----------------------------------------
void S() :
{
     ...
}
{
     ...
}
```

Opciones

- Si se añade, **debe ser la primera sección**
- Permite añadir características avanzadas al generador de JavaCC
- Permite modificar aspectos que condicionan el funcionamiento de los analizadores generados
- Cada opción tiene un valor por defecto
- Si no necesitamos modificar las opciones por defecto, no merece la pena incluir la sección

Opciones

```
options {
    IGNORE_CASE = false;
    COMMON_TOKEN_ACTION = false;
    STATIC = true;
    …
}
```

- Todas las opciones tienen un valor por defecto

Identificación

- La sección se delimita con las etiquetas:

```
PARSER_BEGIN(especification)
PARSER_END(especification)
```

- El nombre de la especificación se utilizará para nombrar los ficheros autogenerados
- Debe existir una clase Java con el nombre de la especificación

Identificación

- Normalmente, añadiremos un método `main`
  - Aunque esto es opcional

```
PARSER_BEGIN(ejemplo)
public class ejemplo {
   public static void main(String[] args) {
      ejemplo parser = new ejemplo(System.in);
      ...
   }
}
PARSER_END(ejemplo)
```

Identificación

- El analizador léxico y sintáctico se suele invocar desde el método principal:

```
PARSER_BEGIN(ejemplo)
public class ejemplo {
  public static void main(String[] args) {
    ejemplo trad = new ejemplo(System.in);
     trad.S();
    …
  }
}
PARSER_END(ejemplo)
```

Identificación

- Y se puede añadir/debe añadir el control de excepciones:

```
PARSER_BEGIN(ejemplo)
public class ejemplo {
  public static void main(String[] args) {
    ejemplo trad = new ejemplo(System.in);
    try {
      trad.S();
    } catch TokenMgrError(e) {
      System.err.println("ERROR en análisis léxico");
    }
…
PARSER_END(ejemplo)
```

Lexicografía

- En la sección de lexicografía:
  - Se definen los *token* del lenguaje mediante **expresiones regulares** y **literales**
    - Similares al resto de expresiones regulares utilizadas en otros lenguajes de programación
    - *Ligeras variaciones*
  - Las definiciones se pueden hacer en cualquier orden
  - **Se recomienda agrupar los tokens para que el analizador léxico sea lo más claro posible**

Expresiones regulares

- No se admiten algunos metacaracteres:

```
\d \D \s \W \w \b \B
```

- Los caracteres y los dígitos debe ir entre comillas:

```
[0-9] → ["0"-"9"]
[a-z] → ["a"-"z"]

[0-9]* → (["0"-"9"])*

[azf] → ["a","z","f"]
[a-zA-Z] → ["a"-"z","A"-"Z"]

[^a] → ~["a"]
[^\n\r] → ~["\n","\r"]
```

```
\t | \n → "\t" | "\n"
hola → "hola"
a? → ("a")?,"\r"]
```

Lexicografía

- Todos los tokens son palabras del lenguaje
- Los caracteres separadores se pueden evitar con un bloque SKIP
  - Espacios, saltos de línea, etc.
  - Comentarios
- Cuatro tipos de descripciones:
  - **TOKEN**
  - **SKIP**
  - **MORE**
  - **SPECIAL_TOKEN**

Lexicografía

- Las declaraciones las añadiremos justo después del PARSER
- Se pueden agrupar por categorías

```
SKIP :
{
   " "
|  "\n"
|  "\t"
|  "\r"
}
```

```
TOKEN :
{
  <tPAL: ["a"-"z","A"-"Z"]
         (~[" ","\t","\n","\r","0"-"9"])* >
}
```

Lexicografía

- Se pueden definir **tokens internos** utilizando #
  - Sólo se pueden utilizar en la especificación de la lexicografía
  - Sirven para mejorar el mantenimiento y la comprensión del lenguaje

```
TOKEN :
{
   < #tDIGITO: ["0"-"9"] >
|  < tNUM: (< tDIGITO >)+ >
|  < tPAL: ["a"-"z","A"-"Z"]
            (~[" ","\t","\n","\r","0"-"9"])*>
}
```

Lexicografía

- Se pueden ejecutar acciones al reconocer un patrón:

```
TOKEN :
{
   < #tDIGITO: ["0"-"9"] >
|  < tNUM: (< tDIGITO >)+ >
      {
         valNums += Integer.parseInt(matchedToken.image);
      }
|  < tPAL: < tPAL: ["a"-"z","A"-"Z"]
            (~[" ","\t","\n","\r","0"-"9"])* >
      {
         nPal++;
      }
}
```

Lexicografía

- Las variable para el analizado léxico se declaran en una sección especial:

```
TOKEN_MGR_DECLS :
{
   //lo declarado aquí estará en "ejemploTokenManager"
   //también estará "lengthOfMatch", "image", ...
   static int nPal = 0;
   static int valNums = 0;
}
```

- Para conocer lo que puedo manejar:
  - la clase “ejemploTokenManager.java”
  - la clase “TokenManager.java”

Lexicografía

- Cuidado con las ambigüedades
  - ¿"**integer**" es palabra reservada o un identificador?
  - "**donald**" es un "**do**" seguido del identificador "**nald**" o solo un identificador ?
- ¿Cómo indico que lo que quiero es que
  - "**integer**" sea palabra reservada
  - "**donald**" sea un identificador ?