Lenguajes y Autómatas I

Juan Pablo Rosas Baldazo

JAX

Alfonso Guerrero Contreras

de control16480211

07/10/2018

Tabla de contenido

Introducción

Un analizador léxico también es conocido como escáner; pues su funcionalidad es la de analizar el lexema de las palabras o cadenas de caracteres sobre un patrón definido.

En este documento, veremos el analizador léxico de java “JAX”.

Analizador léxico

Un analizador léxico también es conocido como escáner; pues su funcionalidad es la de analizar el lexema de las palabras o cadenas de caracteres sobre un patrón definido.

Es decir; El proceso de análisis léxico se refiere al trabajo que realiza el scanner con relación al proceso de compilación.

Un analizador de léxico tiene como función principal el tomar secuencias de caracteres o símbolos del alfabeto del lenguaje y ubicarlas dentro de categorías, conocidas como unidades de léxico.

Las unidades de léxico son empleadas por el analizador gramatical para determinar si lo escrito en el programa fuente es correcto o no gramaticalmente.

JAX

Jax es un compilador léxico creado en lenguaje Java, que genera un escáner a partir de expresiones regulares que existen por defecto en un archivo de java.

Como funciona JAX

Jax procesa estas expresiones regulares y genera un ficher Java que pueda ser compilado por Java y así crear el escáner.

Los escáneres generados por Jax tienen entradas de búfer de tamaño arbitrario, y es al menos más conveniente para crear las tablas de tokens, Jax utiliza solo 7 bits de caracteres ASCII, y no permite código Unario.

Uno de los aspectos fundamentales que tienen los lenguajes Lex y Yacc es que son parte importante de un compilador. Pues la unión de estos dos genera un compilador, claro cada uno de ellos aportando su propio diseño y su forma de ejecutar sus procesos.

Tanto el analizador léxico como el sintáctico pueden ser escritos en cualquier lenguaje de programación. A pesar de la habilidad de tales lenguajes de propósito general como C, lex y yacc son más flexibles y mucho menos complejos de usar.

Métodos y variables definidas por jax

.

Estos son métodos que están incrustados en el archivo java después de que jax procesa la especificación jax.

int jax_next_token () lanza IOException

- Esto se utiliza para iniciar el proceso de coincidencia. Devolverá un valor entero devuelto por una acción. Esta función devolverá -1 cuando se alcance EOF en el flujo de entrada. Si llamas a break; desde dentro de una acción, esta función continuará escaneando desde donde se detuvo en lugar de regresar.

void init (InputStream inp) lanza IOException

- Esto se usa para cebar el lexer, y debe llamarse antes de llamar a jax_next_token () o sucederán cosas extrañas.

Cadena jax_text ()

- Se puede usar para devolver una cadena que contiene la sección coincidente de la expresión regular.

void jax_switch_state ()

- Esto se usa para cambiar el escáner a un nuevo estado si hay algún % de directivas de estado en el archivo.

int jax_cur_line

- Esta es una variable que realiza un seguimiento de la línea actual si utiliza la directiva % line .

int jax_cur_char

- Esta es una variable que apunta a la posición del carácter actual en el archivo de entrada si usa la directiva % char .

Además, hay algunos métodos internos que están destinados a no ser utilizados, excepto por el lexer.

Sintaxix formal

```
st :: =
  (VERBATIMA)? ((((lexStatement | stateStatement) | LINE_DIRECTIVE) | CHAR_DIRECTIVE)) + (VERBATIM)?
lexStatement :: =
  PATTERN or_expr PATTERN (VERBATIM)? SEMI
or_expr :: =
  cat_expr (O cat_expr) *
cat_expr :: =
  singleton (singleton) *
singleton :: =
  (((DOT | CHAR) | fullccl) | PAREN_OPEN or_expr PAREN_CLOSE) (((STAR | PLUS) | QMARK))?
fullccl :: =
  SQUARE_OPEN (CARET)? ccl SQUARE_CLOSE
ccl :: =
  (((CHAR DASH CHAR | CHAR) | DOT)) *
StateStatement :: =
  ESTADO (NOMBRE) +
```

Ejemplo jax

Aquí hay una forma de generar un escáner que cuenta la cantidad de palabras en un archivo. Supongamos que una palabra es cualquier secuencia de caracteres que no sea un espacio en blanco, una nueva línea o una pestaña.

```
# Contar el número de palabras en un archivo
#
# Sección de encabezado
% {

import java.io. *;

clase pública wc
{
  int word_count = 0;

  public static void main (String argv []) lanza IOException
  {
    wc myLexer = new wc ();

    myLexer.init (System.in);
    myLexer.jax_next_token ();
    System.out.println (myLexer.word_count + "palabras");
  }

%}
```

```
# Se asume que una palabra es cualquier secuencia de caracteres
# que no es un espacio en blanco, tabulador o nueva línea.


/ [^ \ _ \ n \ t] + / # La expresión regular para hacer coincidir una palabra

  % {word_count ++; %} # Y su acción asociada.
  ; # No olvides el punto y coma final.

# Sección de seguimiento
% {

}

%}
```

Un archivo de especificación jax tiene tres partes.

- El encabezado
- Las expresiones regulares que deben coincidir
- El avance.

El encabezado y el final están incluidos dentro de % {..%} y se reproducen en el archivo de salida. Cualquier acción asociada con una expresión regular también se especifica de la misma manera. Jax procesa este archivo y genera un archivo java con una función jax_next_token () que se utiliza para iniciar el proceso de coincidencia. Sin embargo, primero debe cebar el lexer, y lo hace llamando al método init () con un flujo de entrada.

Conclusión

En conclusión, Jax es un JAX sirve para agrupar y categorizar una cadena de texto, poniéndola en un objeto “TOKEN”.

Fuentes

http://biblioteca.uns.edu.pe/saladocentes/archivoz/curzoz/sesion_5.pdf

http://linux4u.jinr.ru/usoft/WWW/www_blackdown.org/kbs/jax-ref.html#jaxmethods