LENGUAJES Y AUTOMATAS 1

INVESTIGACION DEL JAX EN JAVA

ALUMNA: DARCY ZUZET LONA CRUZ

N° CONTROL: 16480037

Lunes 12 de noviembre del 2018

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

Jax es un compilador java lex que genera escáneres a partir de expresiones regulares incrustadas en un archivo java esqueleto. Jax procesa estas expresiones regulares y genera un archivo java que puede compilarse para crear el escáner. Los escáneres generados por jax tienen buffers de entrada de tamaño arbitrario, y al menos son más convenientes de crear que los ajustes de StreamTokenizer. Jax maneja solo caracteres ASCII de 7 bits y no maneja Unicode . Además, jax no es [f] lex, y en particular, no permite definir macros o coincidencias de expresiones dependientes del contexto. También revisa * JavaLex , que es otro compilador de lex para Java con una sintaxis similar a la de lex.

Jax es un código alfa, y lo único que sé con certeza es que su escáner (escrito en Jax) se regenera de forma idéntica. Jax está disponible de forma gratuita con la fuente y puede usarse / modificarse.

METODOS 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.

GRAMATICA QUE JAX ENTIENDE

```
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) +
```

Esta gramática fue generada a partir de la especificación de jell para jax.

COSAS DIFERENTES DE F/LEX SOBRE LA SINTAXIS DEL JAX

- Barras utilizadas para delimitar expresiones regulares. Encuentro esto más fácil de leer que tener una combinación de espacios en blanco y citas como lo hace lex.
- El espacio en blanco no es significativo en las expresiones regulares. Me confundo incrustar múltiples espacios en blanco. Así que una secuencia de escape especial para espacios en blanco.
- Los puntos y coma terminan una declaración de expresión regular. Esto no debería ser necesario, pero lo es, y espero encontrar una buena excusa para ello más tarde.
- Jax insiste en llamar a la barra un PATRÓN cuando encuentra un error.
- Debe incrustar acciones en un par % {...%} .
- No es necesario iniciar una acción para una expresión regular en la misma línea que el patrón.
- No se proporcionan macros para expandir expresiones regulares.
- No se proporcionan expresiones regulares dependientes del contexto

ERRORES CONOCIDOS Y LIMITACIONES

- Jax no es lex. Ese fue casi mi primer acrónimo, pero extraño ... ese nombre tuvo un hechizo de mala suerte. Específicamente, las características clave que faltan es que jax no permite macros, tablas de traducción o expresiones regulares dependientes del contexto.
- Sólo se generan escáneres de siete bits. Creo que planeo dejarlo de esta manera, ya que este enfoque cubre una fracción significativa de las tareas de escaneo de manera eficiente. Tal vez use una diferente (como la perezosa NFA a DFA) para manejar Unicode.
- jax_cur_line devuelve la línea en la que termina el token, en lugar de donde comienza. Esto es una tontería, pero esa solución no se ha puesto en esta versión.

EJEMPLO:

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 y 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.

Siempre que una expresión regular coincida con la entrada, el escáner generado ejecutará cualquier acción asociada con la expresión regular exactamente como si se hubiera alcanzado una entrada de caso en una declaración de cambio. La acción para la expresión regular se convierte en el cuerpo de la entrada del caso. En el ejemplo, la acción actualiza una variable para realizar un seguimiento del recuento de palabras. La "caída" de una acción como la de este ejemplo hace que el lexer continúe el proceso de emparejamiento desde donde se quedó, por lo que la instrucción switch está incrustada dentro de un ciclo while.

Para generar el escáner, primero ejecute jax en el archivo y luego compile el archivo generado. El mismo ejemplo se proporciona en la distribución. Si está en la raíz de la distribución, así es como podría funcionar, jax no proporciona operadores ^ o $ o a / b para proporcionar una coincidencia sensible al contexto.

```
% java sbktech.tools.jax.driver -lexFile wc.java examples / wc.lex
% javac wc.java
% java wc <wc.java
677 palabras
% wc wc.java
     255 677 6723 wc.java
%
```

EJEMPLO DEL JAX UTILIZANDO UN JELL

En la variante de jax de este bucle, tiene una sentencia de cambio gigante para todos los estados, y donde parece útil, el cálculo de la siguiente función se implementa mediante un conjunto de sentencias if en lugar de una búsqueda de matriz. Por ejemplo, con una expresión regular que se parece a

/ ba (na) + /
El código generado para las transiciones de estado (ligeramente editado para facilitar la lectura) se parece a

```
switch (jax_cur_state) {
  caso 0:
    if (jax_next_char == 'n') {jax_cur_state = 4;}
    más jax_cur_state = -1;
    descanso;
  caso 1:
    if (jax_next_char == 'b') {jax_cur_state = 2;}
    más jax_cur_state = -1;
    descanso;
  caso 2:
    if (jax_next_char == 'a') {jax_cur_state = 3;}
    más jax_cur_state = -1;
    descanso;
  caso 3:
    if (jax_next_char == 'n') {jax_cur_state = 4;}
    más jax_cur_state = -1;
    descanso;
  caso 4:
    if (jax_next_char == 'a') {jax_cur_state = 0;}
    más jax_cur_state = -1;
    descanso;
  defecto:
    jax_cur_state = -1;
    descanso;
}
```

La máquina de estados para este ejemplo comienza en el estado 1 incidentalmente. Este método evita generar tablas de estado para las filas que tienen solo unas pocas transiciones diferentes. Además, en el caso especial, un estado se asigna a sí mismo la mayor parte del tiempo, la instrucción de cambio se omite y se genera un bucle while. Por ejemplo, la expresión regular de jax para eliminar los comentarios de una sola línea que comienzan con un #
/#.*/
mapas en el siguiente código (ligeramente editado)

```
caso 4:
booleano jax_bool_4 = falso;
while ((jax_next_char! = 0) && (jax_next_char! = '\ n'))
  {
    jax_bool_4 = verdadero;
    jax_next_char = jax_advance_char ();
  }
```

En los escáneres "típicos", esto permite que cosas como cadenas y comentarios se asimilen rápidamente.

El segundo truco consiste en almacenar las tablas de transición de estado para una fila como una cadena en lugar de una matriz estática, y luego llamar a getChars () en la cadena estática en tiempo de ejecución para convertirla en una matriz. Esto reduce tanto el tamaño de la clase como el tiempo necesario para inicializar la matriz. El compilador java crea matrices estáticas al generar código para inicializar esto en tiempo de ejecución, mientras que las cadenas se almacenan tal como están en el archivo de clase. El método getChars () utiliza una operación de copia codificada nativa rápida (System.arraycopy) para realizar la copia.

RENDIMIENTO

El rendimiento de Jax se ve afectado en gran medida por la longitud del patrón que coincide. Jax se desempeña mejor cuando combina secuencias grandes que secuencias más cortas. La razón es que jax tiene que guardar una cierta cantidad de contexto cada vez que realiza una acción, y también a medida que jax acumula datos cada vez más grandes en sus buffers de entrada en expansión, obtiene bloques de datos cada vez más grandes, lo que parece mejorar la entrada general. velocidad de lectura En menor medida, la coincidencia de patrones "simples" como cadenas y palabras clave genera algunas optimizaciones adicionales.

En cuatro programas utilizados están en la distribución, bajo el directorio bnchmark . El sistema que utilicé es un sparc 10 que ejecuta SunOS 5.4 en condiciones de poca carga. Todos los programas se compilaron con javac -O y la salida es el tiempo devuelto para la tercera ejecución sucesiva de cada programa.

```
_____________________________________________________________________ |
lurch
|
| ____________________ | sorber
|
| ___________________________________ | jaxTokenizer
|
| ___________________________________ | buffer
```

lurch.lex

Este es un archivo jax correspondiente a /.|\n/ que coincide con cada carácter y ejecuta (un vacío) la acción del usuario.
8.50u 0.57s 0: 09.28 97.7%

slurp.lex

Este es un lexer igualmente inútil, pero es un lexer rápido e inútil, salvando esos momentos preciosos esperando que no haga nada. Corresponde a /(.|\n)*/ que lee todo el archivo en su búfer antes de regresar.
2.37u 0.57s 0: 03.13 93.9%

jaxTokenizer.lex

Este es un lexer que identifica palabras / [a-zA-Z0-9] + /
4.83u 0.58s 0: 05.52 98.0%

buffer.java

Esta es una prueba que simplemente lee un byte a la vez desde un BufferedInputStream .
5.16u 0.64s 0: 05.98 96.9%

Primero, las llamadas a métodos son bastante caras en Java. La lectura de un byte de un BufferedInputStream se comporta mejor si se leen fragmentos de matriz de bytes grandes y luego escanea la matriz de uno en uno. En segundo lugar, ninguna de las pruebas incluye el costo de hacer un jax_text () , lo que requiere un tiempo adicional para convertir los caracteres coincidentes en una cadena. En tercer lugar, todos los patrones son lo suficientemente simples, por lo que permite que se activen un par de optimizaciones adicionales.

FUENTES

http://linux4u.jinr.ru/usoft/WWW/www_blackdown.org/kbs/jax.html