Memoria

# ApiCreatorXML V0.1

## Clase ApiCreatorXML V0.1

Se utiliza para comprobar el funcionamiento de la api. Crea el escritor y escribe la parte del traductor. Añade los nodos y escribe la parte del árbol. Añade una serie de pasos y escribe la parte de contenido. Por último, crea el XML.

## Clase Node V0.1

Representa los nodos de la parte de árbol del XML

**Atributos:**

**private Integer id:** id del nodo, se crea automáticamente según vas añadiendo nodos.

**private String element:** elemento del nodo.

**private Boolean terminal:** true si es terminal false si no lo es.

**private Integer nivel:** niveldel árbol en el que se encuentra.

**Constructor:**

**public Node(Integer id, String element, Boolean terminal, Integer nivel):**

Recoge los datos pertenecientes a todos los atributos y los asigna.

## Clase Paso V0.1

**private Integer id:** representa el id del paso, el cual se asigna automáticamente cuando se añaden los pasos.

**private String tipo:** describe el tipo de paso(“DespDes” para desplazamientos en descendentes, “derivacion” para derivaciones, “desplazamiento” para desplazamientos en ascendentes y “reduccion” para reducciones ).

**private String leído:** parte de la cadena leída.

**private String pendiente:** parte dela cadena pendiente.

**private String elemento:** elemento que se está procesando.

**private String valor:** valor del elemento que se está procesando.

**private String regla:** regla correspondiente al paso (dicha regla solo se añade si es el primer elemento de la regla )

**private Integer widthRule**: ancho de la regla (pendiente de eliminar)

**private Integer relNodo:** nodo que produce este paso. (paso padre).

**private ArrayList<Integer> changedNodes:** nodos que cambian después de este paso.

**private ArrayList<String> changes:** valor de los pasos que van a cambiar después de este paso.

**Constructor:**

**public Paso(Integer id, String tipo, String leido, String pendiente, String elemento, String valor,Integer relNodo, String regla, Integer widthRule)**

Recoge los valores de los atributos e inicializa las listas **changedNodes** y **changes.**

## Clase Writer V0.1

**private HashMap<String, ArrayList<String>> grammarWithActions:** corresponde a la gramática , es un mapa que contiene las producciones por cada antecedente.

**private String path:** ruta de la gramatica

**private String pathResult:** ruta donde guardar el xml

**private String entryChain:** cadena de entrada

**private Element espec:** corresponde al elemento a partir del que se va a escribir las partes del traductor , la cadena , el árbol y el contenido.

**private Document doc:** representa el documento XML

**private Integer ruleCount:** contador de reglas.

**private ArrayList<String> antecedentes:** lista con los antecedentes de la gramática.

**private Integer numNodos:** contador de nodos

**private ArrayList<Node> nodes:** lista de nodos

**private ArrayList<Paso> steps:** lista de pasos

**private HashMap<String, String> ruleId:** mapa con los ids de cada regla

**private Integer pasoCount:** contador de pasos

**Constructor:**

**public Writer( String path, String pathResult, String entryChain):**

Recoge la ruta de la gramática, la ruta de destino y la cadena de entrada.

Inicializa los mapas de la gramática y de los ids por regla y las listas de antecedentes, pasos y nodos. Lee la gramática y la guarda en los mapas. Inicializa el documento XML.

**Métodos:**

**public Node addNode(String element, Boolean terminal, Integer nivel):**

Crea un nodo, lo añade a la lista de nodos y aumenta el contador de nodos. Devuelve el nodo creado.

**public Paso addPaso(String tipo,String leido, String pendiente,String element, String valor, String regla,Integer relNodo)**

Crea un paso, lo añade a la lista de pasos y aumenta el contador de pasos. Devuelve el paso creado.

**public Boolean writeXML():**

Crea el archivo XML.

**public String removeActions(String production):**

Dada una producción elimina las acciones y los símbolos recursivos.

Se separa la producción y por cada símbolo si no es una acción, si pertenece a los antecedentes se añade al resultado , si no pertenece se añade al resultado la parte del símbolo que termina cuando empiezan los dígitos.

**public String removeOnlyActions(String production):**

Dada una producción elimina las acciones.

Se separa la producción y por cada símbolo si no es una acción se añade al resultado.

**private Integer getNumberIndex(String symbol):**

Devuelve el índice donde empiezan los dígitos en un símbolo recursivo.

Por cada letra se suma uno hasta que se detecta un digito.

**private void readFile():**

Lee y guarda la gramática desde un archivo de texto.

Por cada línea la separa en antecedente y producciones y se separa las producciones. Se añade el antecedente a la lista si no está y se guarda en el mapa de la gramática con acciones una lista nueva con el antecedente como clave. Se añaden cada producción a la lista anteriormente mencionada.

**public void writeTraductor():**

Escribe la parte de <traductor> y <cadena> del XML.

Se crea la etiqueta “traductor” y se añade a la etiqueta espec. Se crea la etiqueta “tipo” y se añade a traductor, después se añade el texto “Descendente” a tipo. Por cada antecedente , por cada producción se añade la regla. Cuando se han añadido todas las reglas se crea la etiqueta “cadena” y se añade a espec, por ultimo se añade el texto de la cadena de entrada a cadena.

**private void addRule(String antecedent ,String production,Element traductor):**

Dado un antecedente , una producción y la etiqueta a la que añadir la regla , añade una regla al xml.

Forma el id de la regla añadiendo a “R” el contador de las reglas. Añade al mapa de ruleIds la regla con el id de la regla como clave. Crea la etiqueta “regla” y la añade a la etiqueta traductor, después le añade un atributo “id” con el id y recoge las acciones semánticas de la regla. Por cada acción semántica:

crea una etiqueta “accionSemantica”, la añade a la etiqueta de regla, obtiene la posición de la acción semántica a la regla para crear un atributo ”pos” con el valor obtenido. Añade como texto la acción semántica y si la acción esta en el medio de la regla se añade una etiqueta “intermedio” con el texto sí.

Añade los símbolos de la producción. Y suma uno al contador de reglas.

**private ArrayList<String> actions(String production):**

Dada una producción devuelve todas las acciones en ella.

Separa la producción en símbolos y por cada símbolo si es una acción la añade a la lista que va a devolver.

**private Integer getPos(String action, String production):**

Dada una acción y una producción devuelve la posición detrás de la que se encuentra la acción.

Divide la producción en símbolos y mientras no sean igual a la acción si el símbolo no es una acción se suma uno a la posición y se pasa al siguiente símbolo.

**private void addSymbols(String production, Element regla,String antecedente):**

Dada una producción, un antecedente y la etiqueta regla escribe los símbolos de la producción en el xml.

Separa la producción en símbolos, por cada símbolo:

Si es el primero crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Si es la primera producción de la regla le añade a valor el antecedente si no añade “|”. Por último, crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false”.

Si el símbolo no es una acción crea la etiqueta “simbolo” , la añade a regla , crea la etiqueta “valor” y la añade a símbolo. Le añade a valor el símbolo. Crea la etiqueta “terminal”, la añade a símbolo y le añade el texto “false” si la primera letra es mayúscula y “true” si no.

**private void writeArbol(Integer altura):**

Escribe la parte de <arbol> del xml.

Crea la etiqueta árbol y la añade al documento. Crea la etiqueta numNodos y la añade la etiqueta arbol con el número de nodos como texto. Crea la etiqueta altura y la añade la etiqueta arbol con la altura del árbol como texto. Por cada nodo en la lista de nodos:

Creo la etiqueta nodo y la añado a la etiqueta arbol con un atributo con el id del nodo. Creo la etiqueta elemento y la añado a la etiqueta nodo con el elemento del nodo como texto, creo la etiqueta nivel y la añado a la etiqueta nodo con el nivel del nodo como texto, creo la etiqueta terminal y la añado a la etiqueta nodo con true si es un terminal y false si no como texto.

**private void writeContenido():**

Escribe la parte de <contenido> del xml.

Crea la etiqueta contenido y la añade al documento. Por cada paso de la lista pasos:

Crea la etiqueta paso y la añade a contenido con un atributo id. Crea la etiqueta tipo y la añade a contenido con el texto del tipo de paso, Si el paso tiene regla:

Crea la etiqueta nuevaRegla y la añade a paso con un atributo refRegla con el id de la regla y la regla como texto. Crea la etiqueta widthRegla y la añade a paso con el ancho de la regla como texto.

Crea la etiqueta cadena y la añade a paso. Crea la etiqueta leido y la añade a cadena con la cadena leído como texto. Crea la etiqueta pendiente y la añade a cadena con la cadena pendiente como texto. Crea la etiqueta elemento y la añade a paso con el elemento del paso como texto. Crea la etiqueta valor y la añade a paso con el valor del paso como texto. Si tiene nodos que cambien de valor por él:

Crea la etiqueta accionSemanticaEjecutada y la añade a paso. Por cada nodo que cambia de valor:

Crea la etiqueta nodo y la añade a accionSemanticaEjecutada. Crea la etiqueta refNodo y la añade a nodo con el id del node que cambia. Crea la etiqueta atributos y la añade a nodo con el valor al que cambia el nodo.