SGML Linter avec JFlex et CUP

Maxime Lovino

15 janvier 2017

1 Introduction

Pour ce travail, j'ai donc réalisé un Linter pour des fichiers de type SGML, ainsi que SGML étendu en utilisant JFlex et CUP. Le linter va vérifier la validité du fichier SGML, c'est-à-dire par exemple l'ordre d'ouverture et de fermeture des balises. Dans le cas d'un fichier SGML étendu, le programme va également afficher la version simplifiée après évaluation des conditions du fichier.

2 Détection des différents symboles avec JFlex

Le programme JFlex va détecter les différents éléments composants le langage SGML et envoyer les symboles à CUP, il s'agit des

- Balises ouvrantes (avec ou sans attributs) ⇒ `OPENTAG`
- Balises inline (avec ou sans attributs) ⇒ `ORPHAN`
- Balises fermantes ⇒ `CLOSINGTAG`
- Contenu entre deux balises ⇒ `CONTENT`

Ensuite, pour gérer les fichiers étendus, on ajoute la détection de

- Setter pour set une variable ⇒ `SET`
- If ouvrant ⇒ `OPENIF`
- If fermant ⇒ `CLOSEIF`

A la fin du fichier, nous retournons le symbole `EOF` à CUP également. Avec les différentes balises et le contenu, on retourne à CUP le texte entier de ce que nous détectons (car nous devons pouvoir réecrire le fichier de façon simplifiée) alors que pour le If ouvrant et le Setter, on ne renvoie que la valeur de l'id, grâce à une fonction en java qui retourne cette valeur à partir du tag entier (utilisation de `String.split` sur le caractère `"`)

3 Analyse syntaxique avec CUP

Dans le programme CUP, l'analyse syntaxique du langage est réalisée, c'est-à-dire que c'est dans cette partie que nous allons vérifier l'ordre d'ouverture et de fermeture des balises, ainsi que produire la version simplifiée d'un fichier SGML étendu. On utilise les symboles non terminaux suivants :

- file, symbolisant notre fichier SGML
- content, symbolisant le contenu du fichier (voir plus bas pour la raison de la distinction)
- openif
- setter
- endif
- opentag
- closedtag
- orphantag
- text

Ensuite on définit les symboles terminaux suivants également, qui sont ceux renvoyés par JFlex :

- OPENTAG
- CLOSINGTAG
- ORPHAN
- OPENIF
- CLOSEIF
- SET
- CONTENT

Ici, en fait nous distinguons `file` et `content` pour pouvoir lancer l'affichage de notre fichier simplifié en fin de parsing du fichier. C'est-à-dire que CUP va détecter un `file` qui comprend un `content` uniquement

puis va sauter aux règles pour `content`, ensuite à la fin de ses règles il va revenir au code de la règle pour `file` et afficher le fichier simplifié dans le cas où il s'agit d'un fichier SGML étendu.

3.1 Logique pour l'affichage d'un fichier simplifié à partir d'un fichier étendu

Pour la détection et la création d'un fichier simplfié à partir d'un fichier SGML étendu, j'utilise un `boolean` nommé `expanded` qui va passer à `true` dès la détection d'un tag `<#set...>`, ce booléen va faire en sorte qu'un fichier simplifié soit affiché ou pas à la fin de l'execution du programme. En effet, le fichier simplifié est de toute façon créé, sauf que sans balise `set`, tout le code sera affiché, car il fait entièrement parti du fichier simplifié.

L'id utilisé par le fichier et servant ensuite à décider de ce qui est interprété du fichier est stocké dans la `String fileID`
Au niveau de la détection de ce qui sera contenu dans le fichier simplifié, le `boolean` nommé `toOutput` est de base à `true` et va changer à la detection d'un tag `if` ouvrant en fonction de la validation de la condition, puis repasser à `true` lors de la fermeture du `if`. En effet, ce qui n'est pas entre dans une condition est affiché de toute façon. Lors de la détection de tout autre contenu (balises, texte), ce contenu est ajouté à la `String simplifiedFileContent` avec l'indendation correcte que l'on calcule en partant à 0 et en l'incrémentant d'un niveau en cas de balise ouvrante, et en la décrementant de 1 en cas de balise fermante.

3.2 Logique pour la détection de l'ordre des balises

CUP recçoit chaque balise dans son integralité à partir de flex, ceci est fait pour que l'on puisse ajouter toute la balise à notre fichier simplifié lors de sa création. De ce fait, si nous voulons avoir la valeur de la balise, c'est-à-dire le nom de la balise, par exemple dans le cas d'une balise `<A t="1.3">` il s'agit de `A`, j'ai écrit deux fonctions qui prennnent en paramètre la balise entière et retournent son nom. Ces deux fonctions sont :

- `String getNameFromOpenTag(String tag)`
- `String getNameFromClosingTag(String tag)`

Dans ces deux fonctions, on utilise `String.charAt(int index)` pour récupérer la lettre du nom du tag. Celle-ci est retournée plus haut dans CUP en l'assignant à la variable `RESULT`. Ensuite durant les règles de grammaire sur la structure du fichier, on vérifie que lorsque nous avons une combinaison de balises ouvrantes puis fermantes leurs noms soient identiques, sinon nous levons une exception, ce qui fera quitter l'analyse syntaxique et qui sera géré dans `Test.java` en affichant "(error)". On peut voir un exemple de ce type de règles ici :

```
content  ::= opentag :n1 closedtag :n2 { :
if(!n1.equals(n2)) throw new Exception ("order of tag is incorrect");:};
```

4 Conclusion

En conclusion, ce TP m'a beaucoup servi à me familiariser avec la syntaxe de JFlex et CUP, ainsi que la logique de leur utilisation. Au début, j'avais pu tout faire dans JFlex en standalone, mais finalement il est beaucoup plus simple de traiter des questions syntaxiques (ordre etc) avec CUP car il permet réellement de définir une grammaire. Ce travail m'a aussi permis de me perfectionner avec les RegExp, pour ce qui est de l'analyse lexicale avec JFlex.