Mémo : Usage des expressions régulières dans Openrefine

Auteur : Mathieu Saby

Licence CC-BY

Historique

|  |  |  |
| --- | --- | --- |
| V1.1 | 21/03/2019 | Ajouts |
| V1 | 10/11/2018 | Création, à compléter |

La version d’Openrefine décrite est la 3.1.

Table des matières

[1. Utilisation des expressions régulières dans Openrefine 2](#_Toc4102455)

[2. Syntaxe des expressions régulières 2](#_Toc4102456)

[3. Astuces et cas d’usage 3](#_Toc4102457)

# Utilisation des expressions régulières dans Openrefine

Les expressions régulières (ou rationnelles) permettent de **repérer**, **d'extraire** et de **remplacer** des motifs dans un texte.

Elles sont utilisables dans Openrefine:

* dans l'interface graphique :
	+ filtres
	+ facettes
	+ menu Remplacer
	+ menu Diviser les cellules multi-valuées
	+ menu Diviser en plusieurs colonnes
* dans certaines fonctions GREL :
	+ find ()
	+ match ()
	+ partition ()
	+ rpartition ()
	+ replace ()
	+ split ()

# Syntaxe des expressions régulières

Nombreux documents en ligne et outils pour créer des expressions régulières. Cf. par exemple <http://cyberzoide.developpez.com/tutoriels/java/regex/> et [regex101.com](http://regex101.com)

Pour les détails de l’implémentation en Java, et donc en GREL, cf. la documentation officielle <https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>

Pour l’implémentation en Jython, cf <http://www.jython.org/docs/library/re.html>

Site pour construire et tester des expression régulières : <https://regex101.com/>

Site sur les expressions régulières en général <https://www.regular-expressions.info/>

**Particularités par rapport à d’autres implémentations :**

Il est possible d’utiliser les « références arrières » sous la forme \n pour faire référence à un groupe capturé par des parenthèses (voir <https://www.regular-expressions.info/backref.html>).

Ex en GREL :

"ABA".match(/(.)B\1/) renverra ["A"]

"ABC".match(/(.)B\1/) renverra null

"ABA".find(/(.)B\1/) renverra ["ABA"]

"ABC".find(/(.)B\1/) renverra null

Il est possible d’utiliser les « lookbehind » positifs et négatifs pour trouver un motif précédé ou non précédé d’un autre.

Ex en GREL :

"BA".find(/(?<=B)A/) renverra ["A"]

"BA".find(/(?<!B)A/) renverra null

En GREL comme en Jython, les quantifieurs (\*, +, ?, {x,y}) sont « gourmands » (*greedy*) par défaut (les motifs sont répété un nombre maximum de fois) : il faut de les faire suivre de ? pour passer en mode « non gourmand», ce qui est en général le comportement souhaité.

Ex. en GREL :

"xx1xx2xx".match(/.\*(\d).\*/) renverra ["2"]

"xx1xx2xx".match(/.\*?(\d).\*/) renverra ["1"]

Ex. en Jython :

import re

return re.findall(".\*(\d).\*","xx1xx2xx") renverra ["2"]

return re.findall(".\*?(\d).\*","xx1xx2xx") renverra ["1"]

En GREL dans la formule *match(),* l’expression régulière doit correspondre à **l’ensemble de la chaîne** testée, sans troncature automatique à droite ni à gauche.

Ex :

"123abc456".match (/(a.c)/) renvoie null

"123abc456".match (/.\*?(a.c)/) renvoie null

"123abc456".match (/.\*?(a.c).\*/) renvoie ["abc"]

La formule *match()* permet de « capturer » un ou plusieurs motifs en utilisant un ou plusieurs couples de parenthèses, mais elle ne permet pas de capturer un motif répété un nombre indéfini de fois. Pour cela il faut recourir à *find()* ou à Jython.

Ex. en GREL :

find("123abc456adc",/a.c/) renvoie ["abc","adc"]

Ex. en Jython :

import re

return re.findall("a.c","123abc456adc") renvoie ["abc","adc"]

# Astuces et cas d’usage

A compléter…

Pour rechercher tout caractère Unicode y compris avec des diacritiques (mais pas les chiffres, la ponctuation ni les symboles) utiliser \p{L} au lieu de \w

Pour rechercher les débuts ou fin de mots ou d’expressions, utiliser \b