**Web Mining - Laboratoire 1**

Karim Azzab, Lukas Bitter et Danick Panchard

**Introduction**

Ce rapport porte sur la réalisation d’un petit programme codé en Java avec pour but les vocations suivantes :

* Utilisation d’un crawler
* Découverte d’une librairie permettant de *parser* des documents
* Utilisation d’un serveur Solr
* Utilisation d’une librairie permettant de faire le lien entre notre crawler et le serveur Solr

Les librairies utilisées dans le cadre de ce petit projet sont les suivantes :

1. Librairie *crawler4j*
2. Librairie jsoup
3. Librairie *solrj*
4. Serveur Solr configuré pour une utilisation avec 2 *core*

Le contexte dans lequel s’inscrit notre projet consiste à indexer dans *Solr* du contenu provenant d’une crypto-monnaie qui se nomme *E-currency coin*. Notre projet final pour le cours de *Web Mining* portera sur des crypto-monnaies que nous allons probablement mettre en lien avec une plateforme de *news* et/ou des réseaux sociaux.

Ce petit projet consiste donc à avoir une première idée du “comment s’y prendre?“. Le but recherché est de pouvoir parser les données intéressantes concernant cette crypto-monnaie. Les données liées à ces crypto-monnaies sont donc :

1. Les *blocs* créés pour une crypto-monnaie donnée
2. Les transactions se rapportant aux *blocs*.

Le site web que nous avons choisi de *crawler* peut-être consulté à l’adresse “<https://cryptobe.com>” qui propose la consulation des *blocks* et des transactions pour plusieurs crypto-monnaies. Nous avons donc choisi un peu au hasard de *parser* une partie du contenu en lien avec la monnaie *E-currency coin.*

**Note**

Au départ, nous avions choisi une autre crypto-monnaie mais nous avons rencontré des problèmes car certains éléments de nos pages étaient chargés *via* du code *Javascript*. Les parties chargées à l’aide de ce langage n’étant pas disponible lors de la consultation du code source de la page, le *crawler* (en tout cas *crawler4j* il semble que d’autres outils peuvent gérer cela. Du moins, en partie...) ne pouvait tout simplement pas voir cette partie de la page web.

Finalement, et après avoir déjà perdu un peu de temps, l’autre site ne voulait pas se laisser *crawler* à cause de son fichier “robot.txt”. Nous savons que nous sommes censés respecter les volontés de ce fichier mais pour des questions de temps, nous avons décidé d’outrepasser cette restriction en désactivant ce fichier dans notre code. Cela nous a permis de pouvoir faire ce laboratoire mais nous prendrons en compte ce paramètre pour le projet final.

**Architecture et fonctionnement du projet**

Pour répondre au mieux à nos besoins, nous avons choisi de diviser le projet en 5 classes réparties de la sorte :

1. Block
2. Transaction
3. EcurrencyTransactionsCrawler
4. SolrjPopulator
5. Controller

Les classes **Block** et **Transaction** sont de simples classes qui nous servent uniquement à stocker notre contenu dans un objet (POO oblige) avant qu’il soit traité par le reste du programme. Elles ont donc uniquement des attributs ainsi que les *getters* et *setters* associés à ces mêmes attributs.

La clase ***SolrjPopulator*** a la responsabilité d’implémenter la librairie Java *solrj* servant à indexer les document sur *Solr* mais également de pouvoir effectuer des requêtes vers le serveur *Solr* à partir de notre petit programme. La classe implémente par conséquent les prototypes de méthodes suivantes :

* public static SolrjPopulator getInstance();
* public void addTransaction(Transaction trans);
* public void addBlock(Block block);
* public void commit();
* public SolrDocumentList query(String query);

Les méthode *addXXX()* permettent de créer les champs directement utiles à notre application, à savoir les transactions et les blocs, la méthode *commit()* sert à envoyer les documents sur solr car les méthodes *addXXX()* ne font que transmettre les documents au client. La méthode *query()* nous permet d’interroger le serveur *Solr* depuis notre programme Java.

Nous tenons à préciser que la méthode *getInstance()* sert à retourner notre instance de *SolrjPopulator* en utilisant un pattern de type singleton.

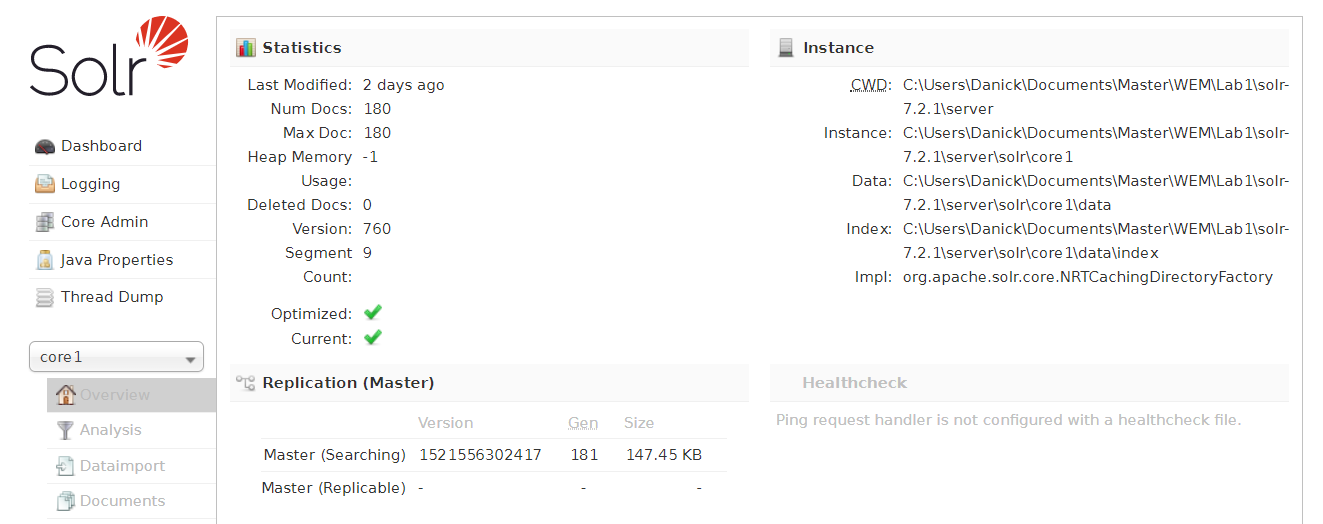
La classe **EcurrencyTransactionsCrawler** est la classe responsable de crawler les pages contenant nos blocset nos transactions. Pour résumer, la classe cible une *URL* de départ puis elle va parcourir les liens se trouvant sur la page pour atteindre les blocset transactions suivantes de la *blockchain* en limitant l’accès aux *url’s* qui nous intéressent (les blocset transactions se rapportant à la monnaie *E-currency coin*). À chaque fois qu’une page est visitée, certains éléments étant sur cette page (comme le numéro du block, la date, etc...) sont mis en forme en vue d’être indexé. On peut noter que puisque nous utilisons solrj pour faire le lien entre le crawler et solr, nous indexons directement nos données depuis notre code java, et envoyons par conséquent les pages indexées sur solr. Pour le *parsing* des pages, c’est à ce moment que la librairie *jsoup* entre en action. En effet, elle permet de sélectionner le contenu ayant une importance pour nos besoins. Cette librairie propose de pouvoir atteindre les différents éléments d’une page web *HTML* à l’aide de sélecteurs *CSS*. Ces sélecteurs correspondent en tout point à ce que l’on a l’habitude de trouver dans une librairie tel que *jQuery*. Les données sont donc récupérées *via jsoup* pour être ensuite passé à la classe *SolrjPopulator* qui aura à charge d’envoyer les données sur le serveur *Solr*.

Notre dernière classe est celle qui se nomme **Controller** et qui sert de point d’entrée pour notre programme (la méthode *main()*). Elle sert à définir quelle page va servir de point de départ pour le *crawl* et à définir quelle classe doit être utilisée pour réaliser l’opération de *crawling*. La fin de cette classe contient pour finir terminer quelques lignes de code qui permettent d’afficher le contenu retourné suite à une requête envoyée sur le serveur *Solr.*

**Le fonctionnement en images**

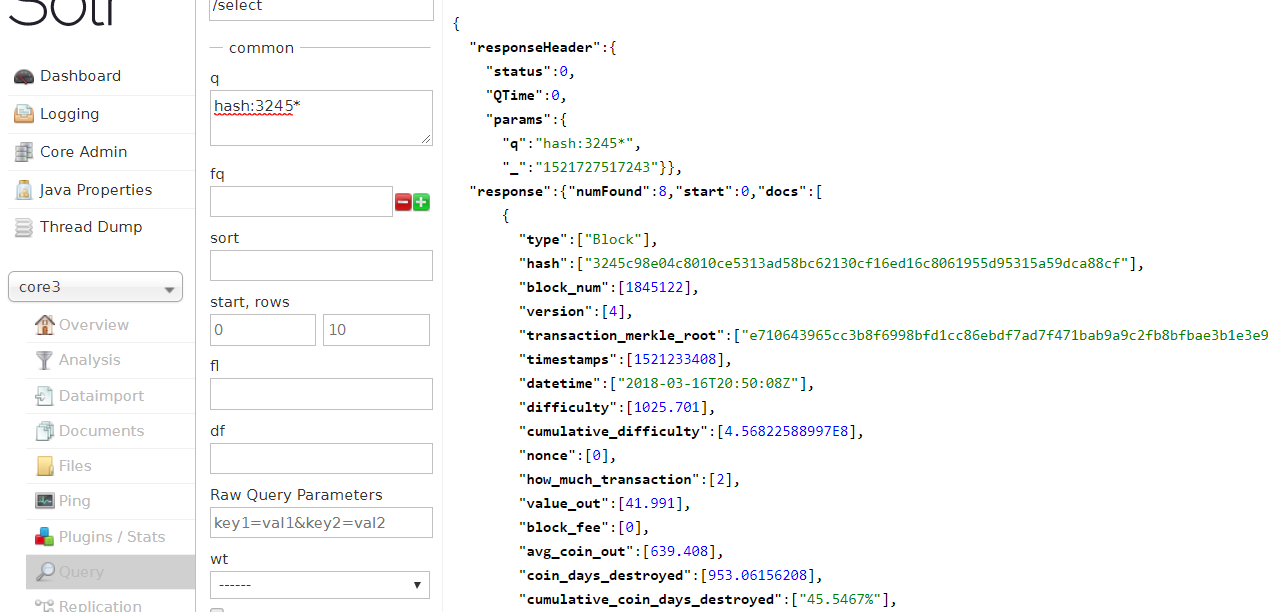
Console Solr :

On peut voir sur cette image que le serveur Solr fonctionne, et que notre cœur (core 1) contient des données (180 documents)

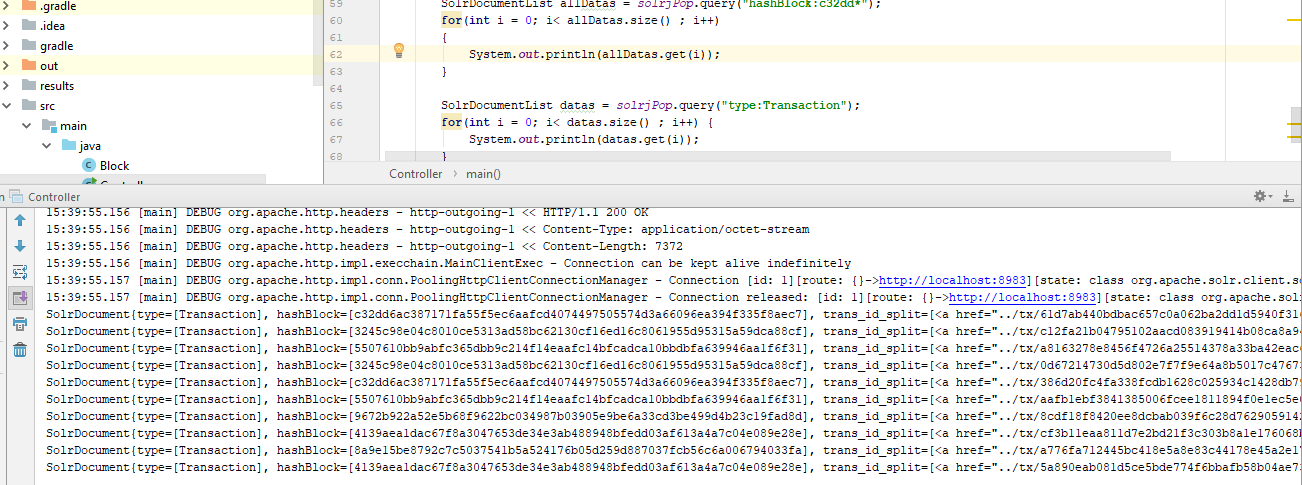


Requêtes depuis la console :

Requête pour le champ hash qui commence par 3245, on peut voir ici que les données sont indexées correctement.

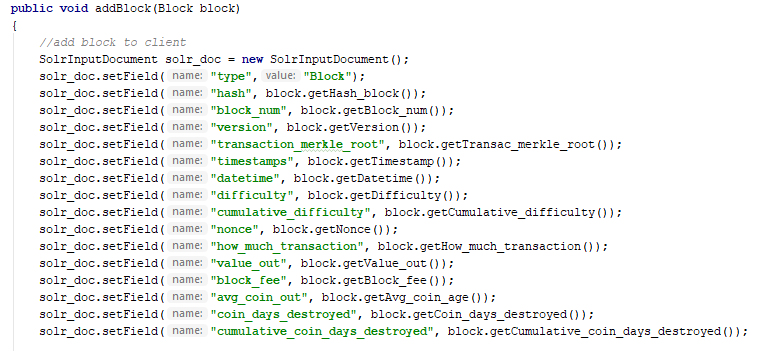


Requêtes depuis le code Java :



Exemple d’indexation :

Cette image présente l’indexation d’un bloc à l’aide de la librairie Solrj



**4.Questions théoriques**

4.1. Veuillez expliquer quelle stratégie il faut adopter pour indexer des pages dans plusieurs langues (chaque page est composée d’une seule langue, mais le corpus comporte des pages dans plusieurs langues). A quoi faut-il faire particulièrement attention ? Veuillez expliquer la démarche que vous proposez.

WARNING: A COMPLÉTER

4.2. Solr permet par défaut de faire de la recherche floue (fuzzy search). Veuillez expliquer de quoi il s’agit et comment Solr l’a implémenté. Certains prénoms peuvent avoir beaucoup de variation orthographiques (par exemple Caitlin : Caitilin, Caitlen, Caitlinn, Caitlyn, Caitlyne, Caitlynn, Cateline, Catelinn, Catelyn, Catelynn, Catlain, Catlin, Catline, Catlyn, Catlynn, Kaitlin, Kaitlinn, Kaitlyn, Kaitlynn, Katelin, Katelyn, Katelynn, etc). Est-il possible d’utiliser, tout en gardant une bonne performance, la recherche floue mise à disposition par Solr pour faire une recherche prenant en compte de telles variations ? Sinon quelle(s) alternative(s) voyez-vous,

veuillez justifier votre réponse.

WARNING: A COMPLÉTER