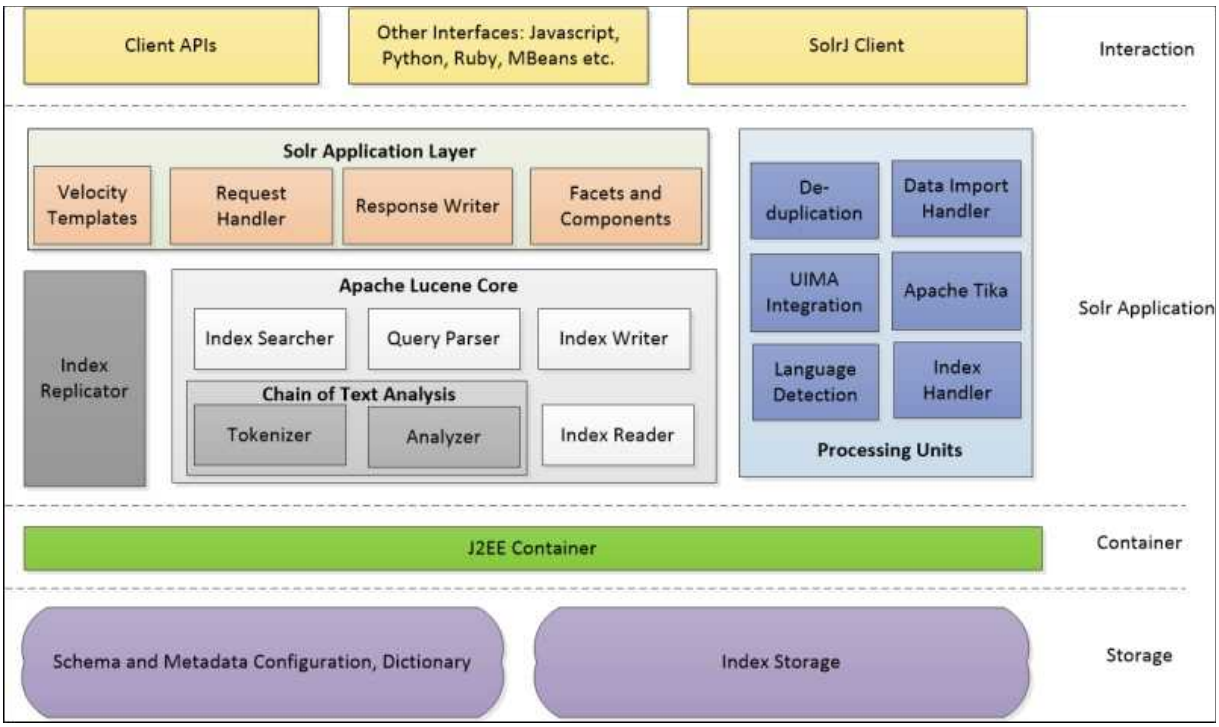
SolR Base de Données Documentaires et Distribuées

Introduction à SolR

SolR (prononcé "solar") est une plateforme logicielle de moteur de recherche s'appuyant sur la bibliothèque de recherche « Lucene », créée par la Fondation Apache et distribuée et conçue sous licence libre.
SolR utilise le langage Java et est développé en Java, il est exécuté par un conteneur de servlets, comme Tomcat, GlassFish, Jboss, Resin, Weblogic, WebSphere. Il peut être exécuté également en mode standAlone, sous Jetty qui est embarqué dans le package SolR. Il communique avec le client à l'aide d'une interface de programmation en XML et JSON, généralement via le protocole HTTP et se base sur les web services en mode REST.

SolR est très fiable, évolutive et tolérante aux pannes, en fournissant une indexation distribuée, la réplication et l'interrogation à charge équilibrée, le basculement automatique et la récupération automatique en cas de problème, la configuration centralisée et plus encore. SolR est utilisé pour ses propriétés techniques de recherche et de navigation par plusieurs sociétés et les plus grands sites Internet du monde.

Comme vous pouvez le constater sur le schéma de fonctionnement de SolR ci dessous, SolR est un ensemble d'éléments qui sont imbriqués en couche et qui ont chacun leur responsabilités et utilité dans l'outils.



La couche la plus profonde est la couche physique(écriture sur disque physique) qui se nomme « Storage », cette couche est celle qui contient les index, les schémas des données et les métadonnées, ainsi que ls dictionnaires des données, ce sont les fichiers physiques.

Une couche conteneur, J2EE est présente également, celle couche est le socle des api de SolR, elle permet notamment la mise en place des web services SolR.

La couche SolR application est la plus complexe, elle est le coeur même de SolR et est composé de dizaine de briques applicatives complexes.

La couche client est la plus haute, c'est elle qui permet aux Crawlers d'interagir avec SolR pour indexer vos documents, mais aussi à vos applications FrontEnd.
Vos Crawlers pourront être développés en Java et utilisez le client SolRJ et les API SolR Cell, utilisez les Web Services.
Vos FrontEnd pourront utilisez les langages classique de développement comme Java, Php, JavaScript, Python, Ruby, Perl et bien d'autres langages qui ont la capacité de communiquer par web services de type REST

La prémisse fondamentale de SolR est simple. Vous donnez beaucoup d'informations à SolR, puis plus tard, vous pouvez lui poser des questions et trouver la pièce d'information que vous voulez. La partie où vous alimentez toutes les informations est appelée indexation. Lorsque vous posez une question, cela s'appel une requête.
Une façon de comprendre comment fonctionne SolR est de penser à un livre à feuillets mobiles de recettes de cuisine. Chaque fois que vous ajoutez une recette dans le livre, vous mettez à jour l'index à l'arrière de celui ci. Vous énumérez chaque ingrédient et le numéro de page de la recette que vous venez d'ajouter. Supposons que vous ajoutez une centaine de recettes. En Utilisant l'index, vous pouvez trouver très rapidement toutes les recettes qui utilisent des pois chiches ou des artichauts, ou du café, comme ingrédient. En utilisant l'index c'est plus rapide que la recherche à travers chaque recette, une par une. Imaginez un livre de mille recettes, ou un million et le temps gagné !.

SolR vous permet de construire un index avec de nombreux champs et domaines différents, ou types d'entrées.

Le schéma est le lieu où vous dites à SolR comment il devrait créer les index à partir de documents d'entrée.

La documentation SolR se trouve sur un wiki à l'adresse suivante :
https://wiki.apache.org/SolR/

Installation

Il faut télécharger le package Zip de SolR sur le site officiel (https://lucene.apache.org/SolR/ ou directement dans le repository Apache pour la version 6.1.0 qui est ici le sujet de notre conversation http://www.apache.org/dyn/closer.lua/lucene/SolR/6.1.0), ce package est à dézipper dans un dossier destination qui ne devra plus changer (dossier des binaires).

Il faut également s'assurer qu'il y a un jdk version 8 ou plus ou un JRE version 8 ou plus installé également pour permettre l'exécution de SolR, car SolR est développé en Java.

Idéalement sur une machine 64bits, il faut utiliser la version 64bits de Java, cela fonctionne avec une version 32bits de Java, mais pose des problèmes avec le monitorage jmx (jconsole & jvisualVM), « si le monitorage est nécessaire ».

Où télécharger une version de Java ?
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Une version JRE est suffisante sauf si l'on veut développer avec les API SolRJ (dans ce cas je vous recommande d'installer directement le JDK qui permettra a terme de faire votre développement SolR en utilisant SolRJ et SolR Cell), dans la foulée n'hésitez pas à installer un IDE pour Java, je vous recommande l'excellent IDE NetBeans qui répond parfaitement aux besoins en développement Java et possède tous les outils de base pour développer de manière sereine en Java.

A noter :
SolR fonctionne parfaitement sur Linux / Microsoft / Mac (testé et validé) avec le jdk/jre 8 et openJdk.

Vocabulaire

Une instance SolR, est une instance au sens processus qui permet l'administration de plusieurs Core ou Shards (en mode Cloud), une instance SolR permet la gestion de plusieurs « Base d'index ».

Un core est une structure qui va contenir des documents sous forme indexés, c'est « l'ensemlbe des fichiers de la base de donnée d'indexation des documents », il est possible d'avoir plusieurs core dans une instance SolR.

Le mode StandAlone, est un mode de fonctionnement de SolR qui permet de gerer des bases d'indexations via des core qui peuvent être réparti sur plusieurs unitées de disques, ce mode ne permet par contre que d'utiliser un seul serveur et plusieurs « Core », mode non distribué.

Le mode Cloud, est un mode spécifique dit distribué qui est pris en charge par SolR que l'on nomme SolRCloud, SolRCloud est conçu pour fournir un environnement à hautement disponible et très tolérant lors de la distribution de vos contenus et de vos requêtes indexées sur plusieurs serveurs. C'est un système dans lequel les données sont organisées en plusieurs morceaux, ou éclats(Shards), qui peuvent être hébergés sur plusieurs machines, avec des répliques assurant la redondance à la fois pour l'évolutivité et la tolérance aux pannes, et met à disposition un serveur ZooKeeper qui permet de gérer la structure globale de sorte que l'indexation et la recherche les demandes peuvent être acheminées correctement.

Un document est un ensemble de champs de différents types (c'est l'équivalent d'une table dans une base de données relationnelle ou de manière plus proche un enregistrement dans une base de type nosql).

Un champ est une valeur quelconque qui peut être de type booléen, entier, flottant, chaine de caractères, ou liste de chaines de caractères (multivalued).

Un champ multi-valué, est un champ qui peut contenir plusieurs valeurs de même type sous forme d'une liste, il n'est pas possible de trier sur ce type de champ (c'est à noter car cela prend de l'importance lors de la conception de nos outils d'indexation et de recherche).

L'indexation, est la fonction d'alimentation des documents en mettant en index certains champs sur lesquels on pourra réaliser nos recherche.

Crawler, est un programme qui permet de faire du (crawling) rechercher tous les éléments (documents, url, métadonnées) et de les insérer dans les champs de notre documents pour indexer ce document dans le core.

Un index inversé, est un index qui part d'un mot et contient tous les références des documents qui contiennent ce mot.

Configuration d'un serveur SolR en mode StandAlone

Une configuration SolR nécessite 2 ou 3 fichiers selon que l'on travaille en mode « SchemaLess » (pas de schèma/types obligatoires) ou en mode « Schema Mandatory » forcé.
Le répertoire central pour SolR est connu dans la variable « SolR.SolR.home » et à pour valeur par défaut « ./SolR/home » dans une configuration de conteneur de servlet.
Le fichier « SolR.xml » est le premier fichier de configuration que SolR essaie de trouver, et il recherche en premier dans cette variable SolR.SolR.home. Ce fichier de configuration,donne à SolR des informations globales de configuration et indique où trouver ses noyaux(cores). Le reste de la configuration provient de chaque noyau défini.

Dans chaque noyau(core), SolR va chercher un fichier « conf/SolRconfig.xml ». Le fichier SolRconfig.xml peut faire pointer SolR vers d'autres fichiers de configuration, comme conf/dih -config.xml pour le gestionnaire de DataImport. À moins que le nom de fichier config est modifié dans SolRconfig.xml, conf/managed-schema sera utilisé pour charger le schéma automatique (mode schemaless), dans le cas ou l'on va mettre en place un mode « schema mandatory ».

Démarrer le serveur SolR

on désignera comme dossier des fichiers d'installation de base de SolR la variable SolR_HOME dans la suite de notre discussion.

Pour lancer SolR, il faut lancer la commande :

```
$SolR_HOME/bin/SolR start [-f] [-c] [-h hostname] [-p port] [-d directory] [-z zkHost] [-m memory] [-e example] [-s SolR.SolR.home] [-a "additional-options"] [-V]
```

-f met en l'affichage en avant plan les logs de l'application , sans cette option les actions sont invisibles sur la fenêtre de commande, et tout est inscrit uniquement dans les fichiers de logs.

-c ou -cloud, permet de démarrer SolR en mode cloud (voir plus loin ce qu'est le mode cloud).

-h hostname, permet de désigner l'adresse ip ou le nom de machine sur laquelle le service va être lancé, « localhost » par défaut ou nom ip par défaut s'il n'y a qu'une carte réseau ou une adresse réseau désignée par défaut.

-p port, numéro de port à utiliser pour le service, par défaut c'est le port 8983.

-d directory, indique le répertoire qui spécifie le répertoire du server SolR , le dossier par défaut est $SolR_HOME/server

-z zookString Chaine de connexion Zookeeper ; a n'utilisez qu'en mode cloud (-c ou -cloud).
Pour charger une instance Zookeeper embarqué, ne passez pas ces paramètres.

-m memory affecte le minimum (-Xms) et maximum (-Xmx) heap size pour la JVM, comme par exemple: -m 4g (équivaut à : -Xms4g -Xmx4g; par défaut le script affecte 512m)

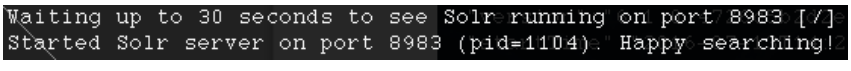
-e example, active des cores d'exemple (pour les tests).

-s SolR.SolR.home, affecte les propriétés de la variable système SolR.SolR.home; SolR va créer les cores sous ce répertoire. Ceci vous permet de lancer de multiples instance de SolR sur le même host tout en réutilisant le même répertoire du serveur en affectant l'utilisation du paramètre -d. Si cette option est utilisé, le dossier doit comporter un fichier SolR.xml, sauf si SolR.xml existe dans Zookeeper.
Ce paramètre est ignoré quand on exécute un exemple avec l'option -e, le dossier par défaut est $SolR_HOME/server/SolR.

Exemple de script de démarrage en background

```
/SolR start -h pc42.home -m 1g -s /Users/tondeurh/Public/SolR/ -Dcom.sun.management.jmxremote
```

NB : l'option *-Dcom.sun.management.jmxremote*, permet de mettre en place le monitorage JMX et contrôler le fonctionnement de SolR via jconsole ou jvisualVM



A remarquer et connaître : SolR dés son démarrage, lance une instance de Jetty accessible via le port 8983 (par défaut, il peut être modifié) et met à disposition un Web Serveur en mode REST (appel par URL GET ou POST), ainsi qu'une UI d'administration via l'URL http://serveur:8983.

Vérifier que SolR est bien démarré (vérification du status)

lancer la commande :
$SolR_HOME/bin/SolR ***status***

Cela doit vous répondre si SolR à une instance en cours ou si aucune instance existe.

```
pc42:bin tondeurh$ ./solr status

Found 1 Solr nodes:

Solr process 621 running on port 8983
{
  "solr_home":"/Users/tondeurh/Public/solr",
  "version":"6.1.0 4726c5b2d2efa9ba160b608d46a977d0a6b83f94 - jpountz - 2016-06-13 09:46:58",
  "startTime":"2016-07-13T14:21:37.039Z",
  "uptime":"0 days, 0 hours, 0 minutes, 14 seconds",
  "memory":"156.3 MB (%15.9) of 981.4 MB"}
```

Il est possible de vérifier le statut en passant une requête GET sur le web service de type REST cURL http://localhost:8983/SolR/admin/cores?action=STATUS

*nb : Sur les OS de type *NIX, il est possible d'utiliser l'application en ligne de commande dans une console cURL, elle est en générale installé par défaut sur la majorité des distributions Linux, Unix, Mac, sous Windows cette application existe également et peut être téléchargé et installé ou vous pouvez utiliser une instance CygWin pour faire une partie du job. Mais vous pouvez également utiliser votre browser Internet préféré également (pour les requête GET), pour ce qui est de passer des requêtes en mode POST la solution la plus simple reste l'utilisation de cURL.*

Ceci renvoi un format xml en réponse qui donne toutes les information sur le(s) core(s) en cours de fonctionnement.

```
▼<response>
 ▼<lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">1</int>
  </lst>
  <lst name="initFailures"/>
 ▼<lst name="status">
  ▼<lst name="BiblioHT">
     <str name="name">BiblioHT</str>
     <str name="instanceDir">/Users/tondeurh/Public/solr/BiblioHT</str>
     <str name="dataDir">/Users/tondeurh/Public/solr/BiblioHT/data/</str>
     <str name="config">solrconfig.xml</str>
     <str name="schema">managed-schema</str>
     <date name="startTime">2016-07-13T15:59:36.570Z</date>
     <long name="uptime">600564</long>
    ▼<lst name="index">
       <int name="numDocs">522</int>
       <int name="maxDoc">522</int>
       <int name="deletedDocs">0</int>
       <long name="indexHeapUsageBytes">-1</long>
       <long name="version">1584</long>
       <int name="segmentCount">1</int>
       <bool name="current">true</bool>
       <bool name="hasDeletions">false</bool>
      ▼<str name="directory">
         org.apache.lucene.store.NRTCachingDirectory:NRTCachingDirectory(MMapDirectory@/U
         lockFactory=org.apache.lucene.store.NativeFSLockFactory@4889e8a6; maxCacheMB=48.
       </str>
       <str name="segmentsFile">segments_a6</str>
       <long name="segmentsFileSizeInBytes">167</long>
      ▼<lst name="userData">
         <str name="commitTimeMSec">1468359854473</str>
       </lst>
       <date name="lastModified">2016-07-12T21:44:14.473Z</date>
       <long name="sizeInBytes">324774736</long>
       <str name="size">309.73 MB</str>
     </lst>
   </lst>
  </lst>
</response>
```


Arrêter SolR

il est toujours possible quand on est en mode foreground (option -f) de faire un ctrl+c pour arrêter SolR, ce n'est pas la meilleure manière, idéalement il faut lancer la commande d'arrêt prévu pour SolR.

```
$SolR_HOME/SolR stop [-k key] [-p port] [-V]
```

-p port correspond au port d'arrêt de SolR

-all permet de demander à SolR de rechercher tous les ports en activité pour SolR et d'arrêter toutes les instances de SolR (c'est la commande la plus simple et la plus efficace).

-k key, permet de passer la clé d'arret de SolR si une clé à été défini (sécurité à mettre en production par défaut la clé est SolRrocks) voir activation protection par clé.

```
pc42:bin tondeurh$ ./solr stop -all
Sending stop command to Solr running on port 8983 ... waiting 5 seconds to allow Jetty process 621 to stop gracefully
```

Vous pouvez ouvrir une nouvelle console et lancer cette commande avec l'option *-all*, c'est une manière plus propre d'arreter SolR quand il tourne en mode ForeGround, si SolR tourne en lode BackGround, la méthodologie ne change pas.

Démarrer le serveur SolR comme un service (Linux/Windows)

Créer un core lors du premier démarrage

Si l'on veut créer notre **core**(voir définition plus haut) sur un dossier ou disque particulier, il faut dans un premier temp créer le dossier nécessaire et y copier le fichier qui se trouvent dans le dossier $SolR_HOME/server/SolR.xml vers ce dossier de destination.

Exemple :
```
cp $SolR_HOME/server/SolR/SolR.xml //nts129/Demat/SolR/SolR.xml
```

Lancer l'instance SolR (voir plus haut) en passant l'option -s qui pointera vers ce dossier d'instance SolR.

Exemple :
```
$SolR_HOME/bin/SolR start -s //nts129/Demat/SolR
```
Cette commande permet de lancer une instance de SolR et de faire pointer le core vers ce dossier, si le core à déjà été créer alors il est utilisé instant&nnément, sinon SolR attend une instruction de création d'un core.

Lancer la commande de création d'un core :
```
$SolR_HOME/bin/SolR create_core [-c core] [-d confdir] [-p port]
```

-c core, prendra pour nom le nom du core que vous allez créer
-d confdir, permet de préciser le type de configuration à utiliser (ne pas préciser permet de garder la configuration par défaut qui est parfaite nommée « data_driven_schema_configs »).
-p port ou est démarrer l'instance sur laquelle on veut créer le core.

Exemple de commande :
```
$SolR_HOME/bin/SolR create_core -c arnum
```

permet de créer le core qui va se nommer `arnum` sur la destination défini par défaut avec l'option -s de notre instance en cours.

Nb : en mode cloud on utilisera l'option `create_collection` à la place de `create_core` (voir les options et la gestion spécifique du mode cloud).

Supprimer un core

utiliser la commande :
```
$SolR_HOME/bin/SolR delete [-c name] [-deleteConfig true|false] [-p port]
```

-c name, est le nom du core à supprimer.
-deleteConfig true|false, permet de préciser si l'on supprime également la configuration.
-p port, le numéro de port de l'instance de SolR.

Exemple :
```
$SolR_HOME/bin/SolR delete -c arnum -deleteConfig true -p 8983
```
Permet de supprimer le core arnum et sa configuration complète qui est actif sur le port 8983 de l'instance SolR en cours.

Les Web Services REST

REST (Representational State Transfer) est l**'un de ces acronymes qui représente une non technologie comme peuvent l'être Ajax, DHTML, Web 2.0 et autres.**
REST est un style d'architecture qui repose sur le protocole HTTP : On accède à une ressource (par son URI unique) pour procéder à diverses opérations (GET lecture / POST écriture / PUT modification / DELETE suppression), opérations supportées nativement par HTTP.

Principes d'une architecture REST
Supposons que nous voulons réaliser un serveur REST pour gérer les livres d'une bibliothèque. Nous devons pouvoir ajouter (POST), modifier (PUT), Lire (GET) et Supprimer (DELETE) ces livres (la ressource à manipuler).

Les formats d'échange
REST n'impose ni ne revendique un format d'échange entre client et serveur.
Vous êtes libre de représenter vos données en **XML**, en **JSON**, en PHP **sérialisé**, en **MessagePack** ou dans tout autre dialecte de votre propre cru (sans oublier que le but est souvent d'exposer des services vers l'extérieur).
Il n'est pas rare que les services REST permettent au client d'indiquer le format dans lequel ils souhaitent dialoguer, sous la forme par exemple d'un paramètre supplémentaire dans l'URL, ou plus simplement grâce aux en tête HTTP en spécifiant le content-type.

Tester un service REST
Pour tester un service REST, nous pouvons utiliser votre navigateur WEB (au moins pour les requêtes de lecture) ou cURL pour tester un service REST (pour la lecture, écriture, suppression).
Par exemple, pour tester les **API REST de SolR**, l'exemple suivant est parfaitement opérationnel :
- **Avec un format d'échange JSON**

Au delà de la théorie
Si REST met en avant le protocole HTTP en proposant la manipulation de ressources interrogées via le bon verbe (GET/POST/PUT/DELETE), on continue de parler d'architectures REST dès lors qu'il s'agit de fournir des services sous la forme de méthodes accessibles via une URI.
Ainsi, il n'est pas rare de voir des architectures REST présentant des URI de type :
```
http://monserveur/rest/?
method=nom_de_methode¶metre=parametre_de_methode
```

Au lieu du « standard théorique »
```
http://monserveur/rest/ressource/id_ressource
```

De même, il n'est pas rare de voir des API simplifiées qui n'utiliseront que GET pour les opérations de lecture et POST pour toutes les autres opérations (création, modification et supression).

```
//Exemples accessible en POST et non nécessairement en PUT /DELETE
http://monserveur/rest/?methode=supprime_commentaire&id=1
http://monserveur/rest/?methode=modifier_commentaire&id=2
```

Quelle est la différence entre REST et RESTful ?
Régulièrement vous pourrez lire REST ou RESTful lorsque la technologie est abordée. En bref, il n'y a aucune différence, RESTful est simplement l'adjectif qui qualifie une architecture de type REST.

Quelles sont les différences entre SOAP et REST ?
REST et SOAP sont tous les deux des architectures utilisées pour fournir des services web. Contrairement à ce que l'acronyme SOAP laisse entendre (Simple Object Access Protocol), REST est souvent utilisé lorsque la simplicité de mise en oeuvre est recherchée.
REST est lisible (pas d'enveloppe XML superflue) et facile à tester (un navigateur suffit) tout en étant facile de mise en oeuvre (un script PHP classique peut souvent être considéré comme RESTful).
SOAP reste toutefois intégré dans de nombreux outils de développements (possibilité d'export de classes en webservices, possibilité de génération automatique de clients à partir des WSDL) et permet des contrôles forts sur les types de données attendus.

Administration des CORES

bibliographie : http://wiki.apache.org/SolR/CoreAdmin

Utilisation des web-services REST pour communiquer avec SolR

Il est possible de communiquer avec SolR par web service REST, pour cela on peu utiliser soit un browser internet et lui passer des requêtes GET, ou utiliser cURL sous linux/Unix ou Mac pour passer ces requêtes GET, il est possible de passer également des requêtes de type POST avec du json en paramètre POST.

Vider un core par requête REST

Il est possible tout en conservant un core et son paramétrage, vider ce core clomplet de toutes les données, cette opération est l'équivalent d'un Truncate en SGBD relationnelle.

```
//réalise le truncate des données
curl http://localhost:8983/SolR/arnum/update?
stream.body=<delete><query>*:*</query></delete>

//réalise ensuite un commit de ce truncate
curl http://localhost:8983/SolR/arnum/update?stream.body=<commit/>
```

Optimize(hard commit) le core

Cette opération d'optimisation peut être longue, voire très longue, il faut donc la réaliser durant des heures de non ou basse production, cette optimisation est nécessaire lorsque l'on à terminé l'alimentation du ou des core avec nos documents.

```
curl http://localhost:8983/SolR/BiblioHT/update?optimize=true
```

Recharger un core par web service

Cette action permet de recharger un core avec les changements réalisés sur celui ci sans provoquer d'arrêts.

Utiliser la requête GET suivante :
`http://localhost:8983/SolR/admin/cores?action=RELOAD&core=core0`

ou core0 est le nom du core à recharger.

Renommer un core

Change le nom d'accès à un core. L'exemple ci dessous change le nom du core0 en core5.
http://localhost:8983/SolR/admin/cores?action=RENAME&core=core0&other=core5

Administration SolR avec la Web Interface

pour une administration en local, vous pouvez appeler l'url suivante :
http://localhost:8983

cette adresse permet d'appeler la web interface qui est mise à disposition par SolR en mode StandAlone via l'utilisation du serveur Web Jetty.

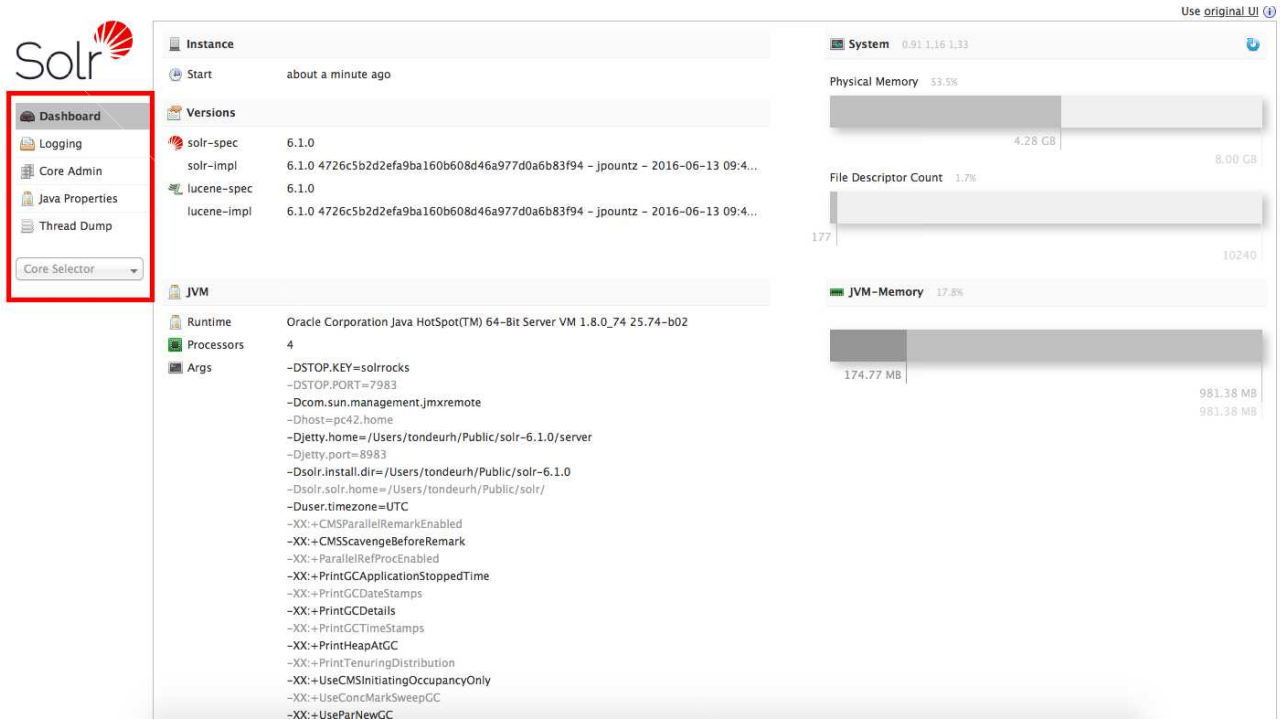


Figure 1

La Web UI possède sur la partie latérale gauche un menu qui propose les options suivantes :

- DashBoard, donne un aperçu rapide de l'état de SolR, durée de l'instance, paramètres de la JVM, version, état de la mémoire et des descripteurs de fichiers(figure 1).

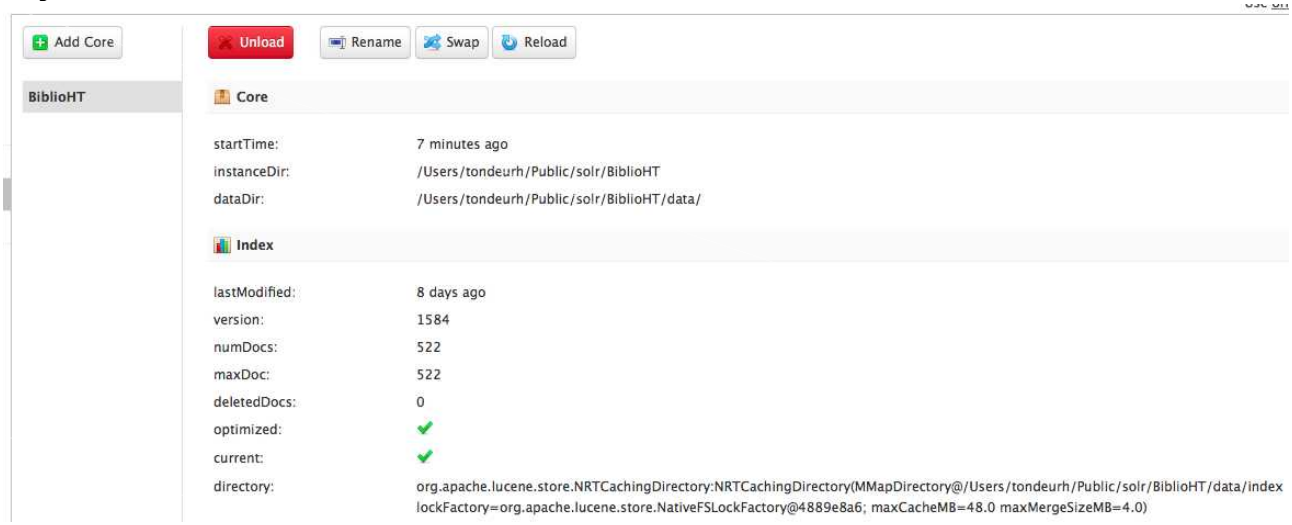
- Logging, permet d'afficher la liste des derniers logs en cours, SolR utilise log4j par défaut, il est toujours possible d'utiliser d'autres API pour les logs, les fichiers logs se trouvent dans le dossier $SolR_HOME/server/logs
Ce dossier est à surveiller car il peut grossir très rapidement et occuper beaucoup de volume, attention également sur les systèmes 32 bits il n'est pas possible de dépasser la taille de 2^32-1 octets pour un fichiers, et ils n'est pas rare d'arriver rapidement sur des instances fortement solicité à cette limite de capacité...
Il y a trois type de fichiers log qui sont généré lors du démarrage d'une instance :

- SolR-8983-console.log, conserve toutes les sorties de la console vers ce fichier (redirection stdout & stderr vers ce fichier).
- SolR.log contient les logs de l'instance serveur SolR.
- SolR_gc.log, contient les logs du garbage collector de la jvm sur les classes SolR.

- Core Admin, permet dde lister et sélectionner un core pour obtenir des informations sur celui ci, donne des informations sur le temps de démarrage du core, les dossiers de localisation du core, le nombre de documents indexés pour le core sélectionné et son état, il est possible avec cette page.
Ce dialogue permet également de créer de nouveau Core, de renommer un core, d'échanger un core et recharger un core.
Attention vous pouvez également supprimer un core (ici seul le fichier core.properties est supprimé).

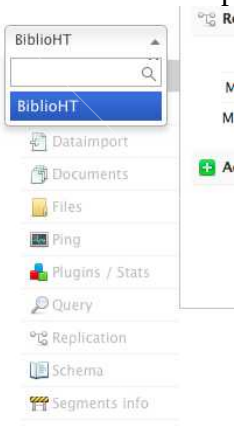
- Thread dump, permet d'obtenir des informations sur les threads SolR en cours(temp cpu/temp utilisateur), mais cette partie dépasse le cadre de notre étude..

Figure 2



Manager et interroger un « core » via la Web Interface

Il faut que votre core soit démarré et sélectionné pour pouvoir manager et interroger ce « core », vous devez sélectionner votre core dans la liste déroulante prévu à cet effet.



Exemple ci dessus on ouvre le core qui se nomme « BiblioHT », on obtient immédiatement un menu complémentaire sous l'application qui permet un certains nombres d'action sur ce core.

Ce menu nous permet d'avoir :
- Overview, ce menu nous donne un apperçu du contenu du core, notamment le nombre de documents et la place qu'occupe la base des index sur le dossier destination.
- ping, qui nous permet de voir le temps de réponse de l'instance SolR.
- DataImport, permet d'utiliser un DIH (lecture import a partir d'une base de données relationnelle) si vous en avez paramètre un.
- Documents, permet de réaliser l'indexation de documents, xml, json, ou fichiers à plat uniquement, les autres types de fichiers ne sont pas pris en charge par cette interface.
- Files, qui permet de consulter les fichiers de paramètre de ce core(lecture seule).
- Schéma, qui permet d'explorer le schéma des données de votre core, cette fonction est interessante pour le développement de votre application cliente.
- Query, permet de requête dans le core les documents en fonction de paramètres spécifiques, appel nosql. (Vous section sur les requête nosql que l'on peut produire sur SolR et les formats de réponses).

Notion sur les documents (table des champs)

L'unité de base SolR de l'information est un document, qui est un ensemble de données qui décrit quelque chose. Un document d'une recette contiendra les ingrédients, les instructions, le temps de préparation, le temps de cuisson, les outils nécessaires, et ainsi de suite. Un document sur une personne, par exemple, peut contenir le nom, la biographie de la personne, la couleur préférée, et la taille des chaussures. Un document d'un livre pourrait contenir le titre, auteur, année de publication, nombre de pages, et ainsi de suite.

Dans l'univers SolR, les documents sont composés de champs, qui sont des pièces plus spécifiques d'information. Pointure pourrait être un champ. Prénom et nom peuvent être des champs.
Les champs peuvent contenir différents types de données. Un champ de nom, par exemple, est un texte (données de caractères). Un champ de taille de la chaussure pourrait être un nombre à virgule flottante afin qu'il puisse contenir des valeurs comme 36 et 50. De toute évidence, la définition des champs est flexible (vous pouvez définir un champ de taille de la chaussure comme un champ de texte plutôt que d'un nombre à virgule flottante, par exemple), mais si vous définissez vos champs correctement, SolR sera en mesure de les interpréter correctement et vos utilisateurs vont obtenir de meilleurs résultats quand ils effectuera une requête.
Vous pouvez indiquer à SolR le type de données contenu par un champ en spécifiant son type de champ. Le type de champ indique à SolR comment interpréter le champ et comment il peut être interrogé.
Lorsque vous ajoutez un document, SolR prend les informations dans les champs du document et ajoute les informations dans un index. Lorsque vous effectuez une requête, SolR peut consulter rapidement l'index et retourner les documents correspondants.

Le mode Schema Mandatory
C'est un mode qui impose un schéma (une structure) de l'index des documents, on impose donc les champs de l'index et également le type des champs, la clé unique, les champs dynamique, et de recopies, les champs qui servirons à la recherche et ceux ne seront pas affichage mais utilisable pour la recherche.
En un mot ce mode est un mode qui permet de définir et maitriser de manière stricte, ce que l'on pourra insérer dans les champs et le modèle qui servira à la recherche et les données affichages ou

non. Ce mode interdit la création de champs à la volée et l'ajout dynamique de champs par le Crawler.

C'est le mode le plus intéressant, mais complexe à mettre en place et demande à ce que l'on ai bien réfléchi à notre schéma d'indexation.

Le schéma est le lieu où vous dites à SolR comment il devrait créer les index à partir de documents d'entrée.

Les types des champs sous SolR

Il existe un certains nombres de type de champs prédéfini sous SolR, ces type sont définie par le tag <typeField> dans le fichier managed-schema ou schema.xml.
Les types standard sont les suivant :

String	solr.StrField
	solr.TextField
Boolean	solr.BoolField
Number	solr.TrieIntField
	solr.TrieLongField
Float	solr.TrieFloatField
	solr.TrieDoubleField
Date	solr.TrieDateField
	solr.DateRangeField
Binary data	solr.BinaryField
Spatial data	solr.LatLonType
	solr.PointType
	solr.SpatialRecursivePrefixTreeFieldType
Closed set of values	solr.EnumField
Random unique ID	solr.UUIDField
Value from external source	solr.ExternalFileField

Ces types sont assez nombreux et suffisants pour décrire l'ensemble des besoins que l'on peut avoir sous SolR, bien entendu, il est possible de redéfinir un type de données sous SolR en utilisant le tag <typeField> qui à pour syntaxe celle ci :

Un champ de type typeField, peu inclure quatre types d'informations au minimum :
- Le nom du type (obligatoire)
- Une implementation d'une class pour ce type (obligatoire)
- Si le type de champs est « TextField », une description de l'analyse du champ pour ce type.
- Des propriétés pour ce type – Cela dépend de l'implémentation de la classe du type, certaines propriétés sont obligatoires.

Exemple :

```
<fieldType name="text_general" class="solr.TextField"
positionIncrementGap="100">

<fieldType name="date" class="solr.TrieDateField"
sortMissingLast="true" omitNorms="true"/>
```

Propriété des types de champs et les cas d'utilisations, le tableau ci dessous donne une liste de cas d'utilisation et les propriétés qui sont à utiliser en fonction des cas d'utilisations, vous remarquerez surtout la propriété « indexed » qui permet d'indexer un champ utilisable pour la recherche, la propriété « stored » qui permet d'indiquer s'il faut stocker ou non le texte du champs, ceci est

surtout quand on veut utiliser le mode « highlighting », et la propriété « multiValued » qui permet d'indiquer si le champ peut contenir plusieurs valeurs (liste de valeurs).

Use Case	indexed	stored	multiValued	omitNorms	termVectors	termPositions	docValues
search within field	true						
retrieve contents		true					
use as unique key	true		false				
sort on field	true[7]		false	true [1]			true[7]
use field boosts [5]				false			
document boosts affect searches within field				false			
highlighting	true [4]	true			true[2]	true [3]	

Je vous invite à aller consulter la documentation SolR sur les différentes propriétés que la définition des types de champs peu supporter, il y en à beaucoup, certains seront obligatoires selon le cas, la documentation à consulter se trouve dans le PDF nommé « appache-solr-reference-guide-6.1.pdf » dans la rubrique « solr Field Type ».

Définir un champ

La définition d'un champ se fait dans les fichiers managed-schema ou schema.xml. Un champ se défini en utilisant l'un dés type prédéfini de SolR ou un des types que vous aurez défini.
Définir un champ est simple et intuitif, il faut comme pour la définition des types, utiliser un tag spécifique qui se nomme <field>

```
<field name="price" type="float" default="0.0" indexed="true" stored="true"/>
```

L'exemple ci dessus, permet de définir un champ qui portera le nom de « price » de type « float » dont la valeur par défaut vaut « 0.0 » qui sera indexé donc utilisable pour la recherche et stocké en tant que tel, donc affichable lors de notre recherche.

Les trois propriétés obligatoires et minimales sont, « name », « type », « default », il existe d'autres propriétés qui pourront venir compléter la définition d'un champ dans une table SolR, ces propriétés ne sont pas obligatoires.

Ici également je vous invite à consulter la documentation sur ces différentes propriétés qui sont également nombreuses. On remarquera tout de même les propriétés suivantes :
indexed
stored
sortMissingLast/First
required

Définir un champ de copie

Vous désirez peut être interpréter certains champs de documents de plusieurs manières, SolR possède pour cela un mécanisme qui permet de faire des copies de champs, il vous est donc possible d'appliquer des types de champs différents pour une information unique.
Le nom du champ que vous voulez copier est la source, et le nom du champ de la copies et la destination, dans le fichier schema.xml il est très simple de réaliser une copie de champs comme l'exemple ci dessous :

```
<copyField source="cat" dest="text" maxChars="30000" />
```

Dans l'exemple , on demande à SolR de copier le champ « cat » vers le champs de nom destination « text », le champ sera copié avant son analyse, ce qui signifie que vous pouvez avoir deux champs avec un contenu identique, mais des significations différentes et indexé de manière différente.

Dans cet exemple, si le champ « text » de destination possède des données, le contenu de « cat » sera ajouté au contenu du champ « text ».
N'oublier pas de configurer votre champ destination comme multivalued="true", s'il doit absolument prendre plusieurs valeurs.
Un usage classique de cette fonctionnalité est de créer un champ de recherche unique qui servira pour la recherche par défaut, exemple titre, auteur, mots clés et corps du texte d'un document qui doivent être utilisé pour la recherche par défaut pourront tous être copié dans ce champ unique que l'on pourra nommer « tout ».
Plus tard vous pourrez configurer SolR pour effectuer une recherche par défaut dans ce champ, le probléme avec ce type d'actions est que votre index vas grossir rapidement.

Le paramètre « maxChars », est un paramètre de type entier, et pose une limite supérieure sur le nombre de caractéres qui doivent être copiés de la source vers la destination.

La source et la destination d'un tag «copyField » peuvent contenir des astérisques (*) caractère jocker qui remplace plusieurs caractères) exemple, la ligne suivante copie le contenu de tout les champs qui ont pour nom « *_t » ou * peut être remplacé par n'importe quels caractères :

```
<copyField source="*_t" dest="text" maxChars="25000" />
```

Le champ « dest » peut contenir une astérisque (*) si et seulement si la source en utilise une également.

Définir les champs dynamiques

Les champs dynamiques permettent à SolR d'indéxer des champs que vous n'avez pas défini explicitement dans le fichier schema.xml de SolR. Ceci est utile si vous découvrez que vous avez oublié de définir un ou plusieurs champs. Les champs dynamiques permettent de rendre votre application moins fragile en fournissant de la flexibilité dans les documents qe vous pouvez ajouter à SolR.
Un champ dynamique est comme un champ régulier, sauf que l'on utilise un nom avec un joker. Lorsque vous indexez des documents, un champ qui ne correspond pas à l'un des champs explicitement définis peut être associé à un champ dynamique.
Par exemple, supposons que votre schéma comprend un champ dynamique avec un nom de * _i . Si vous essayez d' indexer un document avec un champ de nom « cost_i », mais aucun champ de nom « cost_i » explicite est défini dans le schéma, le champ « cost_i » aura le type de champ et d'analyse défini selon « * _i ».

Comme les champs réguliers , les champs dynamiques ont un nom, un type de champ, et des options.

```
<dynamicField name="*_i" type="int" indexed="true" stored="true"/>
```

Il est recommandé d'inclure les mappages de base des champs dynamique (comme celui représenté ci-dessus) dans votre schema.xml.
Ce type de mappings peuvent être très utiles.

Clé unique
L'élément "UniqueKey", indique que le champs en question est de type unique et peut servir de clé pour cette table.
Il faut savoir qu'une clé unique dans SolR n'est pas requise, car l'unicité des lignes est garantie par votre application via le jeu des clés/valeurs
Par exemple vous pouvez utilisez une clé unique s'il vous est nécessaire de mettre à jour un index de documents.
Vous pouvez défini un champ de clé unique en le nommant :

```
<uniqueKey>id</uniqueKey>
```

Les schémas par défaut et les champs de copies ne peuvent pas être utilisé comme champ de clé unique. Vous ne pouvez pas non plus utiliser le type UUID pdateProcessorFactory) pour générer automatiquement une clé unique.
Une clé unique ne peut pas être un champ « multivalued ».

Le mode SchemaLess
Ce mode, est un mode très ouvert de création d'index, il permet effectivement de ne pas concevoir le schéma d'indexation à l'avance et de créer des champs à la volée, c'est notre Crawler qui pourra par exemple créer ces champs.
C'est le mode par défaut lors de l'installation de SolR.

La recherche sous SolR
SolR offre un outil de recherche riche et flexible
Quand vous voulez lancer une recherche dans SolR, il faut utiliser une ligne de requête composé de mots clés spécifiques et qui seront transmis via un Web-service REST.

Une requête pourra permettre de rechercher des documents qui comportent un ou plusieurs mots associés ou réaliser des opérations plus évolués comme le Faceting, ou l'HighLighting par exemple.

Pour répondre à une requête, SolR utilise un "parser de requêtes" qui interprète les termes et les valeurs de la requête. Il existe différents parser de requêtes qui supportent différentes syntaxes.
le parser par défaut de SolR est connu sous le nom de "parser Standard" ou plus communément sous le nom de "parser lucene".
SolR inclus également le "parser DisMaxquery", et le parser "Extended DisMax (eDisMax)".

Le "parser Standard" admet une plus grande ^récision de recherche et de paramètres, et le « parser Dismax » admet plus de tolérance sur les erreurs. Le « parser DisMax » et construit pour donner la même expérience que les moteurs de recherche de type Google par exemple, qui sont très tolérant sur les erreurs de syntaxes sur les requêtes et affichent que trés rarement des erreurs.

Le "parser Extended Dismax" et une version du "parser DisMax" amélioré dans le sens ou il accepte la même syntaxe que le "parser Standard" tout en ayant une bonne tolérance aux erreurs comme le "parser DisMax", et inclus de plus plusieurs outils complémentaires.

En résumé, il y à des paramètres commun à tout les parsers.

Les entrées sur une requêtes peuvent inclures :

Une chaîne de recherche, c'est un terme ou une liste de termes recherché dans les documents auquel on pourra également appliquer des opérations Booléen qui permettront soit d'inclure ou d'exclure certains termes dans la recherche, spécifier l'ordre de recherche et beaucoup de possibilités que l'on détaillera plus bas.

Les paramètres de recherches peuvent aussi préciser des filtres de recherche, filtre sur le nom du champ sur lequel on va effectuer la recherche, trie, nombre de documents qui seront retournés.

Une recherche pourra mettre aussi en lumière (surligner) dans le documents les mots recherchés (ce que l'on nomme l'HighLighting). Les moteurs de recherche comme Google ou Yahoo réalisent ce genre d'opération, certains paramètres de la requêtes permettrons de gérer la manière dont on mettra cette recherche en valeur.

Solr support deux manière de grouper les résultats des recherches, le faceting et le clustering.

Le Faceting est le fait d'arranger les recherches en catégories. Pour chaque catégorie, SolR reporte le nombre d'éléments qui correspondent, ce que l'on nomme une contrainte de facet.

Syntaxe des requêtes

Lancer une requête solr n'est pas plus difficile que de lancer un appel d'un web service en mode REST, oit simplement passer une requête de type http avec un ensemble de paramètres en mode GET.

Exemple :
http://monserveur:8983/core/select?var1=val1&var2=val2&var3=val3 …..

Facteurs de scoring
• **tf** (term frequency) : Plus un terme apparaît de fois dans le document, plus le score est haut

• **idf** (inverse document frequency) – nombre de documents total / nombre de documents contenant le terme

• **lengthNorm** – privilégie les champs contenant le plus petit nombre de termes

• **index-time boost et query-clause boost** => score = tf * idf * lengthNorm * boosts

• **Scoring** : ordre des résultats des requêtes, par défaut

• **Choix explicite de l'ordre** : *select?q=powers:agility**;name asc;***

Faceting

Le "Faceting" est l'un des fonctionnalité les plus puissante de Solr, particulièrement si vous comparer avec d'autres technologies comme les base de données ou les bases NoSQL.
"La recherche par Facette" ou "Facet Search" permet aux utilisateurs qui effectuent une recherche de visualiser un classement haut de leurs éléments basé sur un ou plusieurs aspect de leur recherche dans les documents.
Ceci permet aux utilisateurs d'utiliser des filtres pour effectuer leurs recherches.
Exemple : Vous rechercher un boulot sur un site spécialisé, vous vous attendez a trouver des filtres qui vous permettrons de filtrer vos recherches par ville, régions, type d'emploi, nom de société, et vous espéré également retrouver le nombre d'emplois disponible dans chacune de ces catégories, comme le moteur de recherche ne peut pas afficher tous les documents, il trie les documents pour chacune des facettes et également les facette elle même par ordre de nombre de documents disponibles par catégories,c'est ce que vous permet le « Faceting » sous SolR.

Exemple de requête avec Faceting :

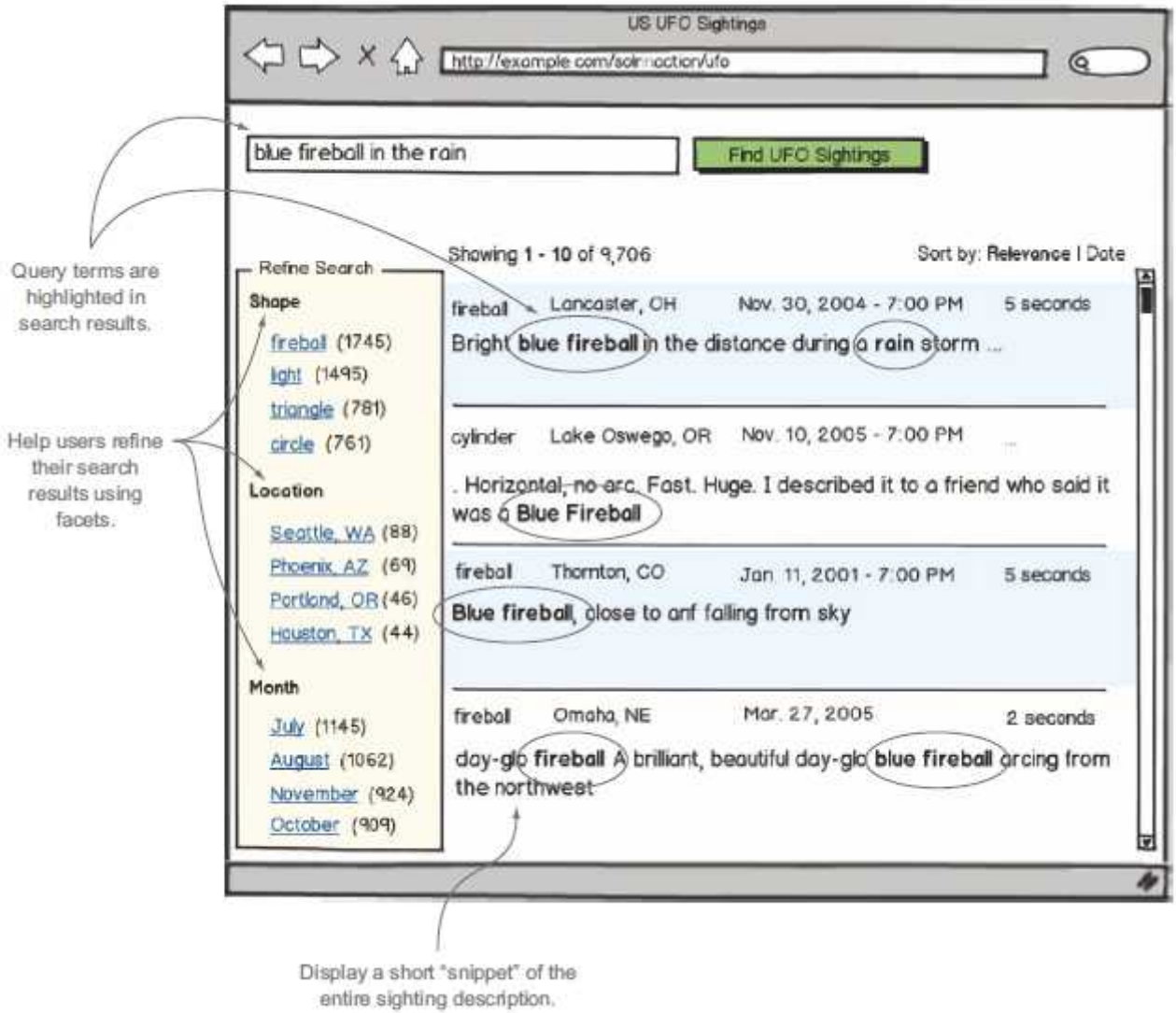
http://localhost:8983/solr/restaurants/select?q=*:*&fq=price:[5 TO 25]&facet=true&
facet.query=state:("Paris" OR "Versailles" OR "Schelle")&
facet.query=state:("Valenciennes" OR "Lille")&
facet.query=state:("Perpignan")

Cette requête ci dessus permet de rechercher tous les restaurant dont le prix est compris entre 5 et 25€ dans les villes de Paris ou Versailles ou Schelle de présenter ces éléments dans une première facette, et de présenter une seconde Facette permettant d'afficher tous les éléments qui se trouvent à Valenciennes ou Lille et une troisième Facette qui se trouve sur Perpignan.

La réponse Json à cette requête pourra être de ce type :

```
...
"response":{"numFound":11,"start":0,"docs":[]},
"facet_counts":{
"facet_queries":{
"state:(\"Paris\" OR \"Versailles\" OR \"Schelle\")":5,
"state:(\"Valenciennes\" OR \"Lille\")":3,
"state:(\"Perpignan\")":3},
...
}
```

Highlighting
L'Highlighting ou la mise ne lumière sous SolR est une technique qui permet lors de chacune de vos requête, d'afficher également une partie de texte qui contient le ou les mots de votre rechercher



Un parfait exemple en est ce MockUp qui montre ce qu'est l'highlighting dans un page Web.

Il est impératif pour faire de l'Highlighting, qu'il y ai un champ qui contient tout le texte de vos documents.

Exemple ici on a un champs description qui contient ce texte :

{"sighted_at": "19951009", "reported_at": "19951009", "location": " Iowa City, IA", "shape": "", "duration": "", "description": "Man repts. witnessing "flash, followed by a classic UFO, w/ a tailfin at back." Red color on top half of tailfin. Became triangular."}

{"sighted_at": "19951010", "reported_at": "19951011", "location": " Milwaukee, WI", "shape": "", "duration": "2 min.", "description": "Man on Hwy 43 SW of Milwaukee sees large, bright blue light streak by his car, descend, turn, cross road ahead, strobe. Bizarre!"}

Pagination des résultats
//TODO

Commit / autocommit / SoftCommit

La notion de transaction est présente également sous SolR, pour gérer ces transaction, SolR propose les notions de commit, d'optimize(hard commit), de softcommit et l'autocommit.
Et bien évidemment la notion de rollback.

Crawling with a Crawler – alimenter les documents avec un robot d'indexation

Un crawler est un robot d'indexation, c'est un logiciel qui explore automatiquement les documents.
Il est généralement conçu pour collecter les ressources (pages Web, images, vidéos, documents Word, PDF ou PostScript, etc.), afin de permettre à un moteur de recherche de les indexer, pour nous ce sera SolR.
En français, crawler est remplaçable par le mot collecteur.
Il existe aussi des collecteurs analysant finement les contenus afin de ne ramener qu'une partie de leur information.
On utilisera soit les applications fournis par le package SolR pour explorer ces documents, ce n'est pas la solution idéale, car cette application bien que très efficace, ne permet qu'une indexation sans métadonnées surajouté par notre crawler personnel.

Utilisation du crawler SolR - PostTool

SolR contient un outil simple en ligne de commande pour l'alimentation de contenu à un serveur SolR. L'outil se nomme $HOME_SOLR/bin/post. L'outil bin/post est un script shell Unix.

Pour l'exécuter, ouvrir une fenêtre et entrez :

Cette ligne de commande prendra contact avec le serveur sur l'ip localhost :8983. On va spécifier le nom de la collection/core qui est obligatoire et le fichier qui va servir à alimenter l'index.
L'option ' -help ' (ou simplement « -h ») affichera l'information sur son utilisation.

```
Usage: post -c <collection> [OPTIONS] <files|directories|urls|-d ["...",...]>
    or post -help

   collection name defaults to DEFAULT_SOLR_COLLECTION if not specified

OPTIONS
=======
  Solr options:
    -url <base Solr update URL> (overrides collection, host, and port)
    -host <host> (default: localhost)
    -p or -port <port> (default: 8983)
    -commit yes|no (default: yes)

  Web crawl options:
    -recursive <depth> (default: 1)
    -delay <seconds> (default: 10)

  Directory crawl options:
    -delay <seconds> (default: 0)

  stdin/args options:
    -type <content/type> (default: application/xml)

  Other options:
```

```
    -filetypes <type>[,<type>,...] (default:
xml,json,jsonl,csv,pdf,doc,docx,ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,html,tx
t,log)
    -params "<key>=<value>[&<key>=<value>...]" (values must be URL-encoded; these pass
through to Solr update request)
    -out yes|no (default: no; yes outputs Solr response to console)
    -format solr (sends application/json content as Solr commands to /update instead
of /update/json/docs)


Examples:

* JSON file: ./post -c wizbang events.json
* XML files: ./post -c records article*.xml
* CSV file: ./post -c signals LATEST-signals.csv
* Directory of files: ./post -c myfiles ~/Documents
* Web crawl: ./post -c gettingstarted http://lucene.apache.org/solr -recursive 1 -delay 1
* Standard input (stdin): echo '{commit: {}}' | ./post -c my_collection -type
application/json -out yes -d
* Data as string: ./post -c signals -type text/csv -out yes -d $'id,value\n1,0.47'
```

Cet application vous permet comme vous pouvez le constater ci dessus il est possible d'alimenter l'index dde SolR avec des contenu de type Json, XML, CSV, ou des fichiers de type pdf, xls, doc, etc.

Cette application est pratique, mais n'est pas assez efficace pour servir de crawler en production, il faudra donc développer le sien ou utiliser les outils dédié comme ManyFoldCF par exemple.
Cette application n'utilise pas le client SolRJ et SolR Cell, mais utilise des appels Web services REST pour insérer les données dans l'index.

Tika, petite introduction

La boîte à outils Apache Tika ™ détecte et extrait les métadonnées et le texte de plus d'un millier différents types de fichiers (tels que PPT , XLS et PDF) . Tous ces types de fichiers peuvent être analysées par le biais d'une seule interface , ce qui rend Tika utiles pour l'indexation des moteurs de recherche , l'analyse de contenu, la traduction, et bien plus encore.
Tika est développé en Java et fourni également des API qui peuvent être utilisé dans vos applications Java pour lire et convertir ces différents formats.

Les principaux formats supporté par la dernière version sont :

HyperText Markup Language (htm, html)
XML et formats derivé (xml, xsd, xsld, etc...)
Microsoft Office document (doc, xls, ppt, docx, xlsx, pptx, etc...)
OpenDocument Format (ODF, et tous les format openOffice et LibreOffice)
iWorks document formats (keynote, pages, numbers pour Mac)
Portable Document Format (pdf)
Electronic Publication Format (epub)
Rich Text Format (rtf)
Compression and packaging formats (tar, ar, cpio, Zip, 7Zip, Gzip, BZip2, XZ et Pack200)
Text formats (csv, txt, etc...)
Feed and Syndication formats (rss, atom, iptc anpa)
Help formats (chm)
Audio formats (wav, mp3, ogg, etc...)
Image formats (png, gif, bmp, jpg, tiff, svg, bpg, etc...)
Video formats (mpg, avi, flv, mp4, etc...)
Java class files and archives (class, jar, war, ear)
Source code (java, c, cpp, groovy)
Mail formats (mbox, pst, rfc822, etc...)
CAD formats (dwg)
Font formats (ttf)
Scientific formats (geoparser, dif, matlab, etc...)
Executable programs and libraries (exe, elf, dll, so)
Crypto formats (pkcs7)
Database formats (sqlite, Acess)

La liste complète des formats pris en charge par la version 1.13 peut être consultée à cette adresse :
http://tika.apache.org/1.13/formats.html#Full_list_of_Supported_Formats

Téléchargement et bibliographie sur Tika :
http://tika.apache.org

(REX) Retour d’expérience, indexation et exploitation d’une base documentaires de comptes rendus médicaux.

Le projet
//TODO

Collecte des données
//TODO

Paramétrage de SolR
//TODO

Utilisation des API SolRJ et SolR Cell pour créer notre Crawler.
//TODO

Developper une application FrontEnd pour notre projet
//TODO