Moteurs de recherche multimédias

Introduction & Requêtes

Moteur de recherche

- A quoi ça sert?
  - Trouver des informations
- De quoi a-t-on besoin?

Les données

- Quel genre de données?
  - Texte
  - Numérique
  - Date
  - URL
  - Images
  - Coordonnées géographiques
- Sous quel format?
  - Idéalement un format générique le plus transparent possible

Les requêtes

- Requêtes simples
  - document dont la date de création est le 28 oct. 2009
  - article dont le numéro d'identification est DK-2093

- Requêtes complexes
  - document créé le 24 octobre 2009 plus ou moins 2 semaines dont le corps principal contient le verbe «créer» et le mot «bonjour», en ne tenant pas compte de la casse.
  - article valant entre 200 et 300 CHF et stocké dans un rayon de 2km d'une coordonnée géographique donnée.

Alternatives

- Base de données relationnelle compatible SQL
  - Avantages: requêtes effectuées directement à la source, syntaxe connue et générique (SQL)
  - Désavantage: requêtes limitées, sans notions sémantiques et rapidement compliquées
- Base de données XML (requêtes XPath)
- Moteur de recherche
  - Avantages: spécialisé dans le traitement des requêtes complexes et utilisant des notions sémantiques
  - Désavantage: séparé de la source de données initiale

Solution proposée

- Données (besoin d'un format générique)
  - XML
- Requêtes (besoin d'un système flexible)
  - Lucene

Lucene

- C’est une librairie Java de recherche FullText
- Fonctionne avec des index composés de documents, stockés sur disque
- Un document est un ensemble de champs
- Les documents peuvent être cherchés par des requêtes faites sur les champs
- Propose de l’analyse de texte flexible (filtres, tokenizers, analyzers)

XML + Lucene = Solr

- Projet OpenSource
- Développé en Java
  => nécessite une machine virtuelle Java.
- Serveur de recherche basé sur Lucene
- Interfaces XML et HTTP
- Définition des types de données et des champs par un schéma flexible

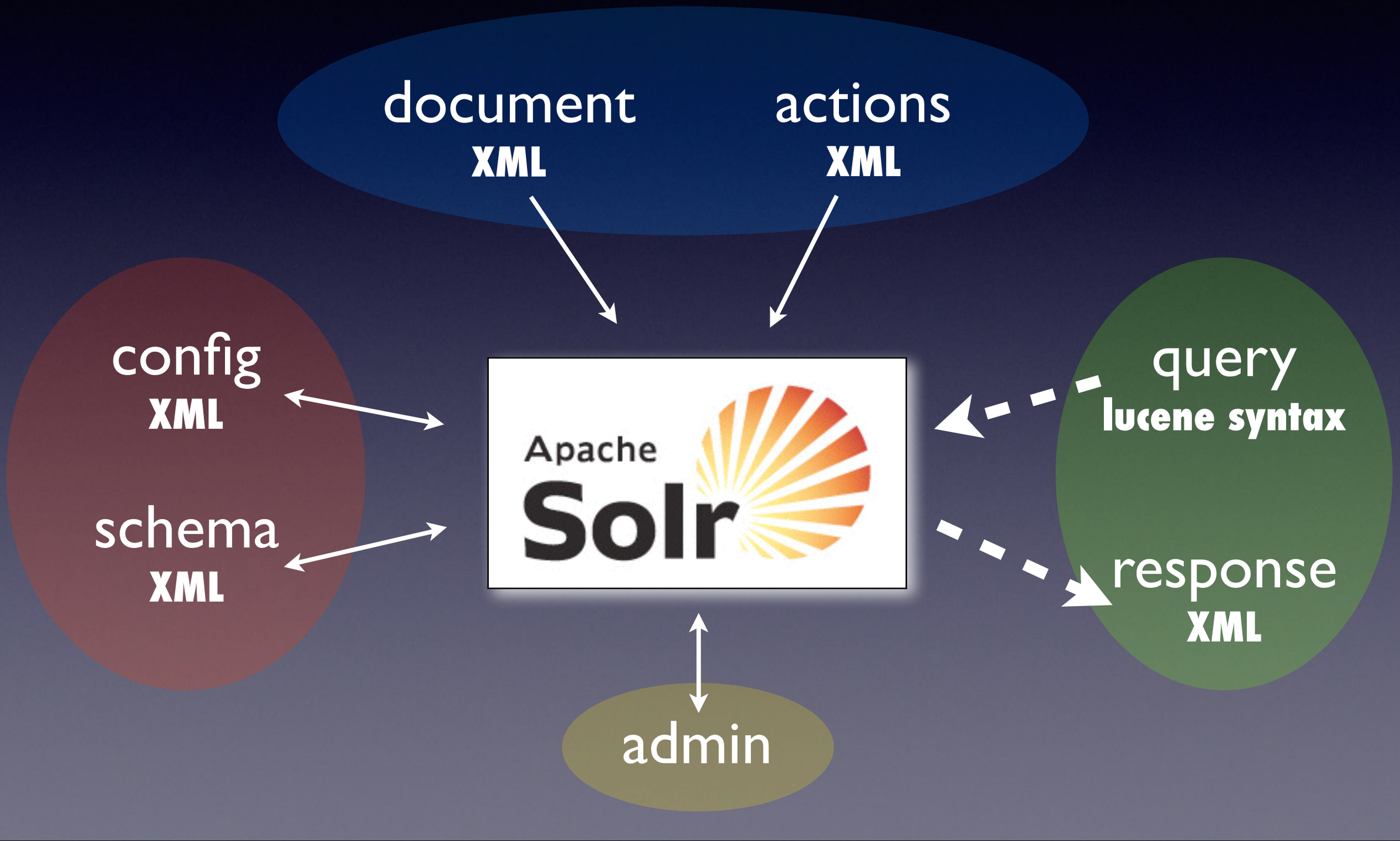
Fonctionnalités avancées de Solr

- Recherche par facettes
- Un riche ensemble d'analyseurs sémantiques
- Soulignement des résultats (Highlighter)
- Correcteur orthographique (SpellChecker)
- Proposition de documents similaires (MoreLikeThis)

Différences SGBD - Solr

	SGBD	Solr
structure	multi-niveaux (database - table - record - field)	un seul niveau (document - field)
commit	instantané	“lent”
update	par champ	par document
recherche	limitée	très rapide
texte	sous-chaînes	termes (mots)

Architecture



admin

admin

- le serveur Solr met à disposition du développeur une interface d'administration, qui s'avère très vite indispensable!
- elle permet d'obtenir des informations sur l'instance Solr ainsi que de la tester.
- http://localhost:8983/solr/admin/

admin

Solr admin page

http://localhost:8983/solr/admin/

Google

Solr admin page

Solr Admin (example)

172.16.43.1:8983
cwd=/Volumes/Autre/Datas/tsr/software/solr/apache-solr-1.4.0/example
SolrHome=solr/

Apache Solr

Solr	[SCHEMA] [CONFIG] [ANALYSIS] [SCHEMA BROWSER] [STATISTICS] [INFO] [DISTRIBUTION] [PING] [LOGGING]
App server:	[JAVA PROPERTIES] [THREAD DUMP]

Make a Query [FULL INTERFACE]

Query String: solr

Search

Assistance	[DOCUMENTATION] [ISSUE TRACKER] [SEND EMAIL] [SOLR QUERY SYNTAX]
	Current Time: Tue Jan 05 19:08:47 CET 2010
	Server Start At: Tue Jan 05 19:08:06 CET 2010

Ouvrir « http://localhost:8983/solr/admin/schema.jsp »

admin

- La zone du haut permet d'accéder, en consultation, aux différents fichiers de configuration et pages d'informations.
- La zone centrale permet d'envoyer des requêtes au serveur Solr.
- La zone du bas donne accès à des ressources documentaires

query
lucene syntax

response
XML

queries

Les requêtes sur Solr se font par l'intermédiaire de la syntaxe Lucene

Le but de la syntaxe lucene est de pouvoir exprimer une recherche sur des valeurs spécifiques de champs permettant de choisir et retourner les documents contenant ces champs.

queries

Une requête Lucene est une expression
se composant de termes et d'opérateurs.

Un terme peut être de type unique (un seul mot)
ou phrase (plusieurs mots entre guillemets).

Il peut débuter par un identifiant de champ,
suivi de ':' afin d'indiquer le champ à rechercher.
(sans cela, le champ par défaut est utilisé).

exemples: `titre:dernier` `texte:"le journal"`

queries

Les termes peuvent être organisés entre eux à l'aide de ces 3 opérateurs:

obligatoire	+	AND	&&
interdit	-	NOT	
optionnel		OR	||

Attention, les opérateurs AND / && et OR / || n'ont pas la même signification que d'habitude. Leur utilisation équivaut à définir les 2 termes les entourant comme obligatoires / optionnels

queries

La définition de termes obligatoires et optionnels permet de faire des requêtes très précises et d'influencer le score des éléments trouvés.

bonjour +comment +va

va retourner tous les documents contenant 'comment' et 'va' mais pas forcément 'bonjour'. Ceux contenant aussi 'bonjour' auront par contre un meilleur score que les autres

bonjour comment AND va

aura le même résultat que l'expression précédente car l'opérateur AND a pour effet d'ajouter un '+' aux deux termes qui l'entourent.

queries

Pour faire des expressions plus complexes
avec des groupes de termes optionnels par exemple,
il faut utiliser des sous-expressions.

(+voici +exemple) (+sous +expression) -ignoré

retourne les documents contenant soit 'voici' et 'exemple',
soit 'sous' et 'expression', soit les deux. Mais pas 'ignoré'.

(voici AND exemple) OR (sous AND expression) NOT ignoré

même signification que l'expression précédente.

voici AND exemple OR sous AND expression

retournera les documents contenant les 4 mots
car est équivalent à +voici +exemple +sous +expression.

queries

Wildcards

? – 1 caractère

* – 0-n caractères

(attention: un terme ne peut pas commencer par un de ces caractères)

Intervales

`[start TO end]` – valeur entre start et end - extrémités comprises

`{start TO end}` – valeur entre start et end - extrémités exclues

Boosting

Permet de mettre du poids sur certains termes, s'utilise en ajoutant le caractère '^' et une valeur après le terme à booster.

Exemple: `title:docteur^10 text:docteur` – met l'accent sur docteur dans le titre

queries

Fuzzy searches

permet de spécifier, pour un terme ‘unique’, le taux de similarité que doit avoir un champ pour être sélectionné
s’utilise en ajoutant le caractère ‘~’ après le terme.

exemple: `nusic~` — retournerait les champs contenant “music”

Proximity searches

permet de spécifier, pour un terme ‘phrase’, la distance maximale qui doit séparer les mots de la phrase pour qu’un champ soit sélectionné
s’utilise en ajoutant le caractère ‘~’ après la phrase.

exemple: `“iPhone 4S”~2` — retournerait les champs ayant le mot '4S' au maximum à une distance de 2 du mot 'iPhone'.

queries

Les valeurs de type 'date' doivent correspondre à un format particulier, selon la norme ISO-8601.

yyyy-mm-ddThh:mm:ssZ
exemple: 2010-01-11T10:45:55Z

Le mot-clef "NOW" peut aussi être utilisé.
Il correspond à l'instant présent.

queries

Des dates relatives peuvent aussi être calculées automatiquement grâce à une syntaxe mathématique.

Les unités suivantes sont reconnues:
MILLI - SECOND - MINUTE - HOUR - DAY - MONTH - YEAR

Il est possible de faire des arrondis avec le caractère '/'.

NOW/DAY -7DAYS
NOW/MONTH -12MONTHS
NOW+18HOURS-5SECONDS
2010-01-01T00:00:00Z + 10YEARS

queries

Pour faire une recherche sur tous les documents il faut utiliser la syntaxe:

:

(car '*' seul n'est pas autorisé par Lucene)

Pour rechercher tous les valeurs possibles d'un champ, il faut utiliser la syntaxe:

champ:[* TO *]

Solr étant très flexible, il est possible de définir des types de valeurs spécifiques et donc aussi des manières spécifiques de les rechercher (ex: géolocalisation)

queries

Exemples de requêtes:

- content-type: video AND title:reportage
- publication-date:
  [2009-01-01T00:00:00Z TO 2009-12-31T00:00:00Z]
- duration: [30 TO 90] AND location:(lausanne OR genève)
- type: (maison NOT jumelée)
- creation-date: {* TO NOW-2MONTHS}
- birthday: [NOW/YEAR -10YEARS TO *]
- name: [* TO *]
- *:*

response

Solr utilise un format XML pour répondre aux requêtes.

Ce format est composé de plusieurs éléments,
permettant de déduire les types de données:

<lst/> - liste nommée d'éléments
<arr/> - tableau d'éléments
<str/> - valeur de type String
<int/> <long/> <float/> <double/> - valeurs numériques
<bool/> - valeur booléenne
<date/> - valeur de type Date (format ISO-8601)

response


```
- <response>
  - <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">31</int>
    - <lst name="params">
        <str name="indent">on</str>
        <str name="start">0</str>
        <str name="q">rosé AND fribourg</str>
        <str name="version">2.2</str>
        <str name="rows">10</str>
      </lst>
    </lst>
  - <result name="response" numFound="1" start="0">
    - <doc>
        <int name="city.code">1754</int>
        <str name="city.name">Rosé</str>
        <str name="city.region.code">FR</str>
        <str name="city.region.name">Fribourg</str>
        <str name="id">city.1754-4</str>
        <str name="role">city</str>
        <date name="timestamp">2010-01-10T19:06:20.538Z</date>
      </doc>
    </result>
</response>
```

paramètres

q – la requête
fq – le filtre de requête
start – le numéro du premier document à retourner
rows – le nombre de documents à retourner
sort – paramètre de tri (ex: city.code asc, city.name desc)
fl – la liste des champs à afficher (ex: score)

wt – le format de sortie (xml, xslt, json, php, etc)
tr – si wt=xslt, indique le fichier xslt à utiliser

debugQuery – si 'true' ajoute des infos de débuggage

paramètres

http://localhost:8983/solr/select?
q=city.code:[2000 TO 2999]&
sort=city.code desc&start=100&rows=5&
fl=city.code,city.name,city.region.name

va trier les résultats par ordre décroissant des numéros postaux, depuis le résultat numéro 100 par page de 5 éléments. Et n'afficher que les champs numéro postal, nom de localité et acronyme de canton.

paramètres

```
– <response>
  – <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">0</int>
    – <lst name="params">
        <str name="sort">city.code desc</str>
        <str name="fl">city.code,city.name,city.region.name</str>
        <str name="start">100</str>
        <str name="q">city.code:[2000 TO 2999]</str>
        <str name="rows">5</str>
      </lst>
    </lst>
– <result name="response" numFound="335" start="100">
  – <doc>
      <int name="city.code">2738</int>
      <str name="city.name">Court</str>
      <str name="city.region.name">Berne</str>
    </doc>
  – <doc>
      <int name="city.code">2736</int>
      <str name="city.name">Sorvilier</str>
      <str name="city.region.name">Berne</str>
    </doc>
```

paramètres

Il faut toujours faire bien attention à ce que les URLs soient encodées correctement lors des requêtes Solr.

En effet certains caractères, comme le "+", l'espace, le slash, sont traduits par d'autres dans le navigateur.

Le caractère "+" doit par exemple être inséré dans son format URL-Encodé: %2B

paramètres

Certains paramètres peuvent être définis via le formulaire de recherche avancée de Solr (lien "FULL INTERFACE"). Par exemple, q (requête), fl (liste des champs), start (début des résultats), rows (nombre de résultats par page).

MAIS sort, le paramètre pour le tri n'est pas accessible ainsi. Pour l'utiliser, il faut donc l'ajouter "à la main", dans l'URL du navigateur.

facettes

La recherche par facettes permet, à partir d'une requête, d'affiner la recherche en se basant sur les valeurs effectives des résultats trouvés.

Par exemple, en faisant une recherche sur toutes les localités contenant 'sur', on va demander à Solr de nous retourner la liste des cantons trouvés.

Il y a 3 types de facettes:
les facettes par champs, par dates et par requêtes.

facettes

Pour activer la recherche par facette, il faut ajouter un paramètre 'facet' avec la valeur 'true'.

Pour utiliser les facettes par champ, il faut indiquer à Solr sur quels champs on souhaite calculer les facettes, par le paramètre 'facet.field'. Solr va ensuite faire un total pour chaque valeur possible du champ spécifié.

http://localhost:8983/solr/select?q=sur&facet=true&facet.field=city.region.name

facettes

```
– <response>
  – <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">1</int>
    – <lst name="params">
        <str name="facet">true</str>
        <str name="facet.field">city.region.name</str>
        <str name="q">sur</str>
        <str name="facet.limit">5</str>
      </lst>
    </lst>
  + <result name="response" numFound="46" start="0"></result>
  – <lst name="facet_counts">
      <lst name="facet_queries"/>
    – <lst name="facet_fields">
      – <lst name="city.region.name">
          <int name="Vaud">35</int>
          <int name="Fribourg">7</int>
          <int name="Grisons">1</int>
          <int name="Jura">1</int>
          <int name="Neuchâtel">1</int>
        </lst>
      </lst>
      <lst name="facet_dates"/>
    </lst>
</response>
```

facettes

Pour utiliser les facettes par requête, il faut indiquer à Solr des requêtes qu'il doit exécuter pour calculer les différents totaux de facettes.

http://localhost:8983/solr/select?q=*:*&facet=true&facet.query=city.code:[* TO 2999]&facet.query=city.code:[3000 TO 5999]&facet.query=city.code:[6000 TO *]

facettes

```
- <response>
  - <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">14</int>
    - <lst name="params">
     + <arr name="facet.query"></arr>
       <str name="q">*:*</str>
       <str name="facet">true</str>
      </lst>
    </lst>
 + <result name="response" numFound="5014" start="0"></result>
 - <lst name="facet_counts">
    - <lst name="facet_queries">
        <int name="city.code:[* TO 2999]">1420</int>
        <int name="city.code:[3000 TO 5999]">1411</int>
        <int name="city.code:[6000 TO *]">2183</int>
      </lst>
      <lst name="facet_fields"/>
      <lst name="facet_dates"/>
    </lst>
</response>
```

facettes

Pour utiliser les facettes par date, il faut indiquer à Solr le champ de type 'date' à utiliser pour les facettes, via le paramètre 'facet.date', ainsi que les intervales désirés.

Les intervales se définissent à l'aide des paramètres 'facet.date.start', 'facet.date.end' et 'facet.date.gap'.

http://localhost:8983/solr/select?q=*:*&facet=true&facet.date=timestamp&facet.date.start=NOW/YEAR&facet.date.end=NOW&facet.date.gap=%2B1MONTH

facettes

D'autre paramètres des facettes permettent d'influencer la manière dont elles s'affichent.

facet.limit - permet de définir le nombre maximal de facettes affichées (par défaut 100).

facet.offset - permet de définir le numéro de la première facette à afficher (par défaut 0), équivalent à 'start' pour les requêtes.

facet.mincount - permet de définir le nombre de résultats minimum requis pour qu'une facette soit affichée (par défaut 0).

spellchecker

Le correcteur orthographique fonctionne en établissant un dictionnaire lors de l'indexation des documents, avec les valeurs des champs.

Il retourne ensuite une liste de termes semblables.

Pour l'activer, il faut ajouter un paramètre 'spellcheck' avec la valeur 'true'.

http://localhost:8983/solr/select?q=rosé&spellcheck=true

spellchecker

```
– <response>
  – <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">10</int>
    – <lst name="params">
        <str name="spellcheck">true</str>
        <str name="q">rose</str>
      </lst>
    </lst>
  + <result name="response" numFound="2" start="0"></result>
  – <lst name="spellcheck">
    – <lst name="suggestions">
      – <lst name="rose">
          <int name="numFound">3</int>
          <int name="startOffset">0</int>
          <int name="endOffset">4</int>
        – <arr name="suggestion">
            <str>Rose</str>
            <str>Arosa</str>
            <str>Gross</str>
          </arr>
        </lst>
      </lst>
    </lst>
</response>
```

highlighting

La fonction de soulignement automatique permet de demander à Solr de nous indiquer où le mot-clef a été trouvé dans les champs retournés.

Cette fonction s'active en ajoutant un paramètre 'hl' avec la valeur 'true', puis en spécifiant quels champs doivent être soulignés via le paramètre 'hl.fl' (spécifier * pour souligner tous les champs).

http://localhost:8983/solr/select?q=le&hl=true&hl.fl=*

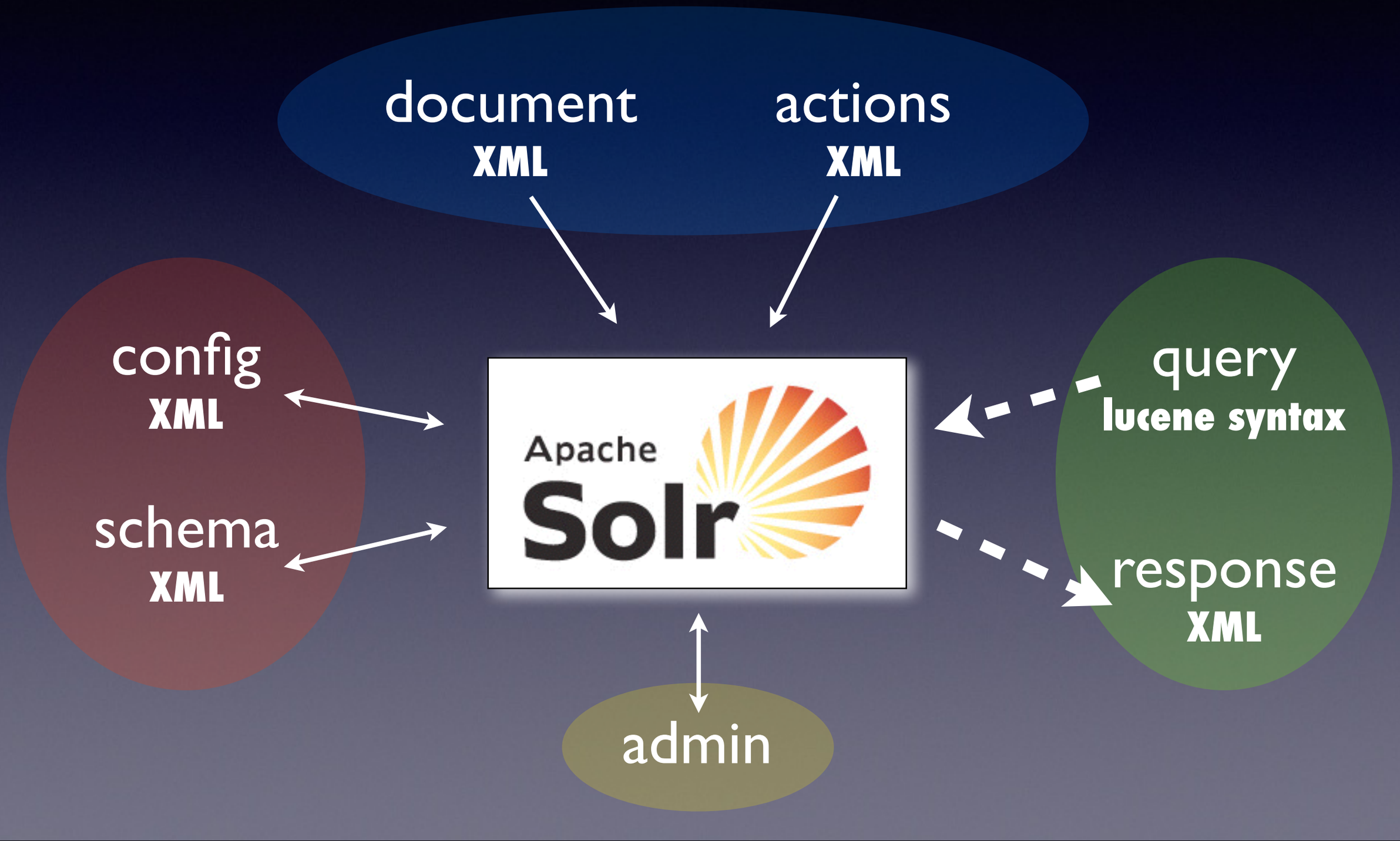
highlighting

```
– <response>
  – <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">2</int>
    – <lst name="params">
        <str name="hl.fl">*</str>
        <str name="hl">true</str>
        <str name="q">le</str>
        <str name="rows">2</str>
      </lst>
    </lst>
+ <result name="response" numFound="132" start="0"></result>
– <lst name="highlighting">
  – <lst name="city.1040-18">
    – <arr name="city.name">
        <str>Villars-<em>le</em>-Terroir</str>
      </arr>
    </lst>
  – <lst name="city.1041-15">
    – <arr name="city.name">
        <str>Poliez-<em>le</em>-Grand</str>
      </arr>
    </lst>
  </lst>
</response>
```

Moteurs de recherche multimédias

Schéma et Indexation

Architecture



configuration

configuration

La configuration de Solr passe par deux fichiers principaux:

- solrconfig.xml
- schema.xml

Ils se trouvent dans le répertoire "solr/conf"

Ils peuvent être consultés directement via l'interface d'administration de Solr.

configuration

solrconfig.xml

solrconfig

Ce fichier contient tous les paramètres de configuration de l'instance Solr.

Solr étant un projet hautement modulable, car basé sur des interfaces Java très génériques, une grande quantité de modules peuvent être activés pour modifier son comportement.

C'est précisément dans ce fichier de configuration que les modules peuvent être activés et paramètrés.

solrconfig

Le principe général de configuration est que Solr est construit autours de composants (SearchComponent).

Ces composants sont des classes Java qui sont configurées dans le fichier solrconfig.xml

Chaque composant peut ensuite être utilisé dans un gestionnaire de requête (RequestHandler) qui est aussi configuré dans ce fichier.

Les pages 'admin' et 'select' sont 2 RequestHandler.

schema.xml

schema

Ce fichier contient la définition des champs et types de données utilisés dans l'index Solr.

Il est composé de trois parties:

- définition des types
- définition des champs
- paramètres généraux

types

Solr met à disposition plusieurs types de données, par l'intermédiaire de classes java.

La plupart des types sont des types simples, ne nécessitant pas une logique de traitement particulière:

int, float, long, double, boolean, string, binary, date, random

Ces types simples sont indexés tels-quels.

type text

En plus des types simple, Solr propose des champs de type 'text'. Ce type est assez complexe à gérer car il peut être configuré de manière très précise.

C'est grâce à ce type qu'il est possible de faire de la rechercher fulltext avec Solr.

Sa configuration se base sur un principe de filtres.

type text

Lors de l'indexation, Solr aura pour objectif de générer une liste de termes distincts à partir d'un texte.

Pour générer ces termes distincts, il va faire passer le texte par une chaîne de filtres.

Il stockera ensuite dans l'index les termes résultant de l'application de cette chaîne de filtres.

type text

Cours CM393: étude de Solr à Comem, Yverdon-les-Bains!

WhitespaceTokenizer

Cours	CM393:	étude	de	Solr	à	Comem,	Yverdon-les-Bains!

WordDelimiterFilter

Cours	CM	393	étude	de	Solr	à	Comem	Yverdon	les	Bains

LowerCaseFilter

cours	cm	393	étude	de	solr	à	comem	yverdon	les	bains

ISOLatin1AccentFilte

cours	cm	393	etude	de	solr	a	comem	yverdon	les	bains

StopFilter

cours	cm	393	etude	solr	comem	yverdon	bains

SnowballPorterFilter

cour	cm	393	etud	solr	comem	yverdon	bain

type text

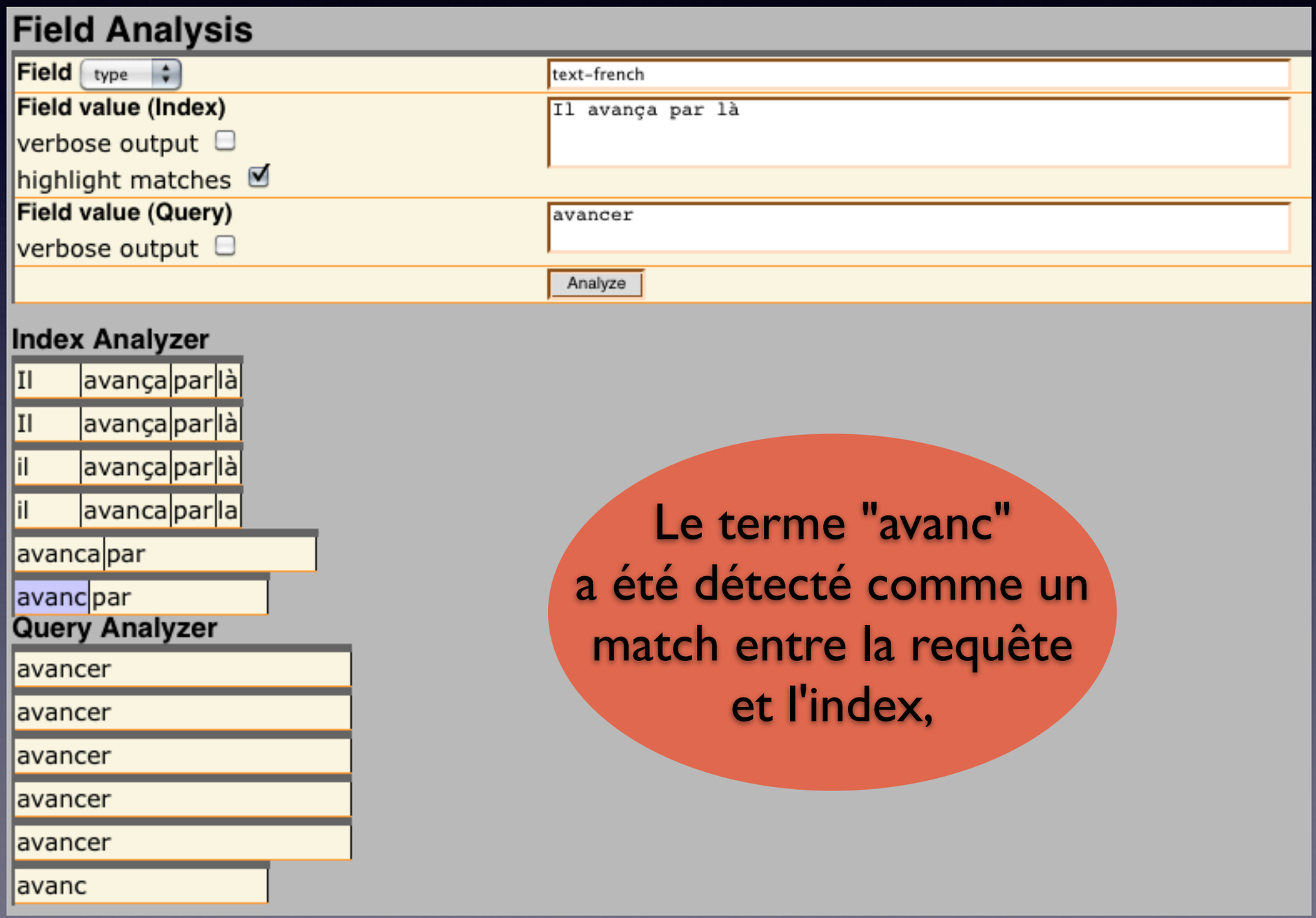
Ensuite, lors de la recherche, Solr fera passer la requête de l'utilisateur par les mêmes filtres que ceux utilisés lors de l'indexation.

Tous les termes finaux similaires entre la requête et les termes indexés permettront à Solr de savoir quels documents correspondent à la requête.

L'enchaînement des filtres et leur configuration sont intégralement définis dans le schema.

type text

L'outil "Analysis" permet de tester les chaînes de filtres



Le terme "avanc"
a été détecté comme un
match entre la requête
et l'index,

type text

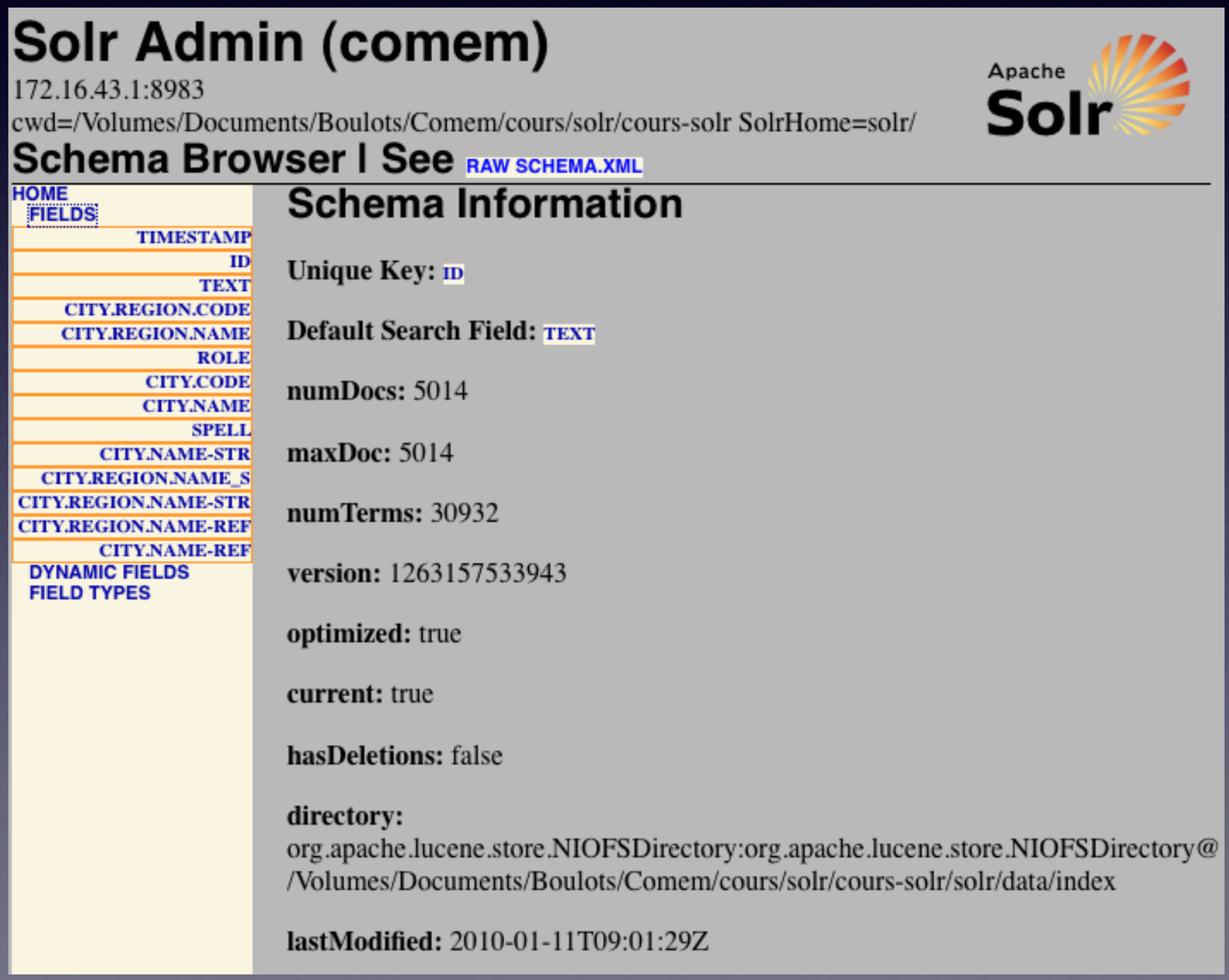
La plupart du temps, plusieurs types de champs textes vont être définis dans le schéma. Chacun d'eux permettant des recherches tenant compte de critères différents.

Par exemple, un type texte pour les contenus en français, un autre pour les contenus en anglais, etc.

Cela signifie qu'un même contenu texte se retrouvera indexé de plusieurs manières dans des champs différents. Cela permet une grande flexibilité.

schema

Solr offre un outil très pratique, le "Schema browser", pour appréhender de manière intuitive la structure d'un index.



schema

Chaque champ du schéma peut, grâce à cet outil, être analysé.

Field: city.region.code

Field Type: REFSTRING

Properties: Indexed, Tokenized, Stored, Omit Norms, Sort Missing Last

Schema: Indexed, Tokenized, Stored, Omit Norms, Sort Missing Last

Index: Indexed, Tokenized, Stored, Omit Norms

Index Analyzer: org.apache.solr.analysis.TokenizerChain DETAILS

Query Analyzer: org.apache.solr.analysis.TokenizerChain DETAILS

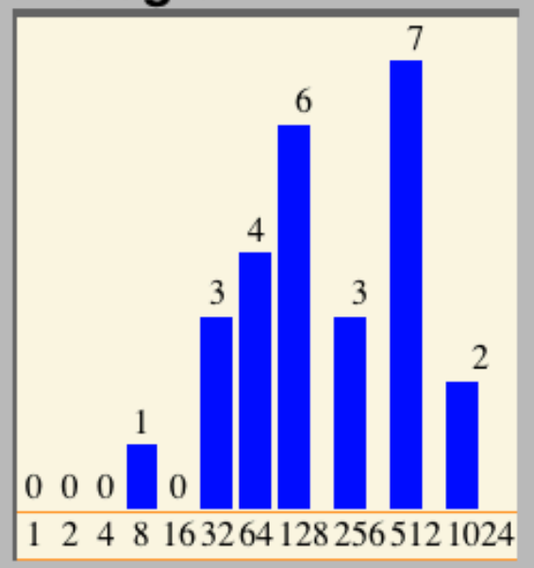
Docs: 5014

Distinct: 26

Top 10 Terms

term	frequency
be	689
vd	530
zh	465
ti	362
gr	339
vs	335
ag	326
fr	288
sg	261
tg	219

Histogram



schema

Pour définir effectivement le schema d'une instance Solr, il faut éditer le fichier "schema.xml" qui se trouve dans le répertoire "config".

En plus de la définition des types et des champs, il contient aussi des paramètres généraux:

- uniqueKey - champ utilisé comme identifiant unique
- defaultSearchField - champ de recherche par défaut
- defaultOperator - l'opérateur par défaut (OR ou AND)
- copyField - permet de copier un champ dans un autre

schema

La définition d'un champ dans le schema de Solr peut avoir plusieurs paramètres:

name (obligatoire) - le nom du champ
type (obligatoire) - le nom du type du champ
indexed - true si le champ doit être indexé (recherchable et triable)
stored - true si la valeur du champ doit être stockée
compressed - true si le champ est compressé (ralenti les traitements)
multiValued - true si le champ peut être multivalué
default - une valeur par défaut

indexation

indexation

L'indexation se fait par l'intermédiaire de fichiers XML qui sont envoyés à Solr, via une requête HTTP POST sur l'url http://localhost.8983/solr/update

Le fichier XML contient des actions qui sont ensuite interprétées par Solr:

- <add>
- <commit> <rollback> <optimize>
- <delete>

indexation

Pour effectuer la requête HTTP Post, n'importe quel logiciel peut être utilisé (par exemple, curl, wget, ...)

Un petit programme Java pratique est fourni avec Solr pour cela. Il s'agit du programme "post.jar".

```
java -jar post.jar fichier-a-indexer.xml
```

Aura pour effet d'envoyer le fichier à Solr et d'effectuer un commit automatique.

add

Permet d'aiouter un document dans

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<add>
  <doc>
    <field name="role">city</field>
    <field name="id">city.1000-8</field>
    <field name="city.code">1000</field>
    <field name="city.name">Lausanne</field>
    <field name="city.region.code">VD</field>
    <field name="city.region.name">Vaud</field>
  </doc>
</add>
```

actions

L'action 'commit' permet de dire à Solr que les documents ajoutés peuvent être effectivement indexés.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<commit/>
```

L'action 'rollback' permet d'annuler les dernières additions.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rollback/>
```

L'action 'optimize' permet d'optimiser l'index.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<optimize/>
```

Moteurs de recherche multimédias

Interfaçage

Formats de sortie

Formats de sortie

Plusieurs formats différents peuvent être définis à l'aide du paramètre 'wt'

Exemples: xml, xslt, csv, json, ruby, python php, phps, ...

Utilisation en XSLT

- Le format d'export XSLT permet de traiter le résultat XML avec une feuille de style.
- Cette feuille de style peut être programmée pour afficher une interface de recherche.
- Cette méthode implique un traitement côté serveur et ne permet pas beaucoup d'interactivité.

Utilisation en javascript

- En utilisant javascript, on peut offrir une plus grande interaction.
- Le format le plus pratique pour une utilisation avec Javascript est le format 'json'.
- JSON est un format d'échange de données structurées entre différentes languages.
- Il permet de transférer d'un language à l'autre un tableau associatif.

Syntaxe JSON

- JSON = JavaScript Object Notation
- Un "objet" JSON peut contenir différents éléments:
  - nombre: 5
  - texte: "bonjour"
  - tableau: [item1, item2, item3]
  - objet: {"key1": value1, "key2": value2}
- Les éléments structurés peuvent être imbriqués.

Exemple JSON

http://localhost:8983/solr/select/?q=role:audio&rows=1&indent=on&wt=json


```
{
  "responseHeader":{
     "status":0,
     "QTime":4,
     "params":{
       "q":"role:audio",
       "start":"0",
       "rows":"1",
       "wt":"json",
       "ident":"on"}},
 "response":{"numFound":29,"start":0,"docs":[
    {
      "id":"audio-nicolas-Allumer le feu",
      "role":"audio",
      "date_creation":"2010-12-31T00:00:00Z",
      "date_modification":"2010-12-31T00:00:00Z",
      "url":"/comem/nicolas/audio/Allumer le feu",
      "album":"Ce que je sais",
      "auteur":"Hallyday Johnny",
      "taille":8.0,
      "titre":"Allumer le feu",
      "type":"mp3",
      "timestamp":"2011-12-05T13:29:55.045Z",
      "genre":["Rock"]}]
 }}
```

Utilisation de JSON

- Avec javascript, la construction de la page html va se faire côté client, à partir des donnée obtenues de Solr.
- Pour simplifier la programmation, on va utiliser la librairie JQuery.
- Cette librairie nous offre la méthode 'getJSON' qui permet d'obtenir un objet JS à partir d'une requête retournant un JSON.

Exemple avec JQuery

```
var request = {};
request['q'] = "*:*";
request['rows'] = 10;
request['start'] = 0;
request['wt'] = "json";
var url="http://localhost:8983/solr/select";
$.getJSON(url, request, function(result, status, data) {
      var documents = result.response.docs;
      for ( var i = 0; i < documents.length; i++) {
      displayDocument(documents[i]);
      }
   });
});
```

delete

Permet de supprimer un document de l'index.
Soit par query, soit par id.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<delete>
  <id>city-1000-11</id>
</delete>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<delete>
  <query>city.postcode:1000</query>
</delete>
```

références

- Solr 1.4 Enterprise Search Server by David Smiley and Eric Pugh - Packt Publishing Ltd. ISBN 978-1-847195-88-3

  (disponible à la bibliothèque HEIG-VD)

- http://lucene.apache.org/solr/tutorial.html
- http://people.apache.org/~yonik/presentations/Solr.pdf

Moteurs de recherche multimédias

Interfaçage

Formats de sortie

Formats de sortie

Plusieurs formats différents peuvent être définis à l'aide du paramètre 'wt'

Exemples: xml, xslt, csv, json, ruby, python php, phps, ...

Utilisation en XSLT

- Le format d'export XSLT permet de traiter le résultat XML avec une feuille de style.
- Cette feuille de style peut être programmée pour afficher une interface de recherche.
- Cette méthode implique un traitement côté serveur et ne permet pas beaucoup d'interactivité.

Utilisation en javascript

- En utilisant javascript, on peut offrir une plus grande interaction.
- Le format le plus pratique pour une utilisation avec Javascript est le format 'json'.
- JSON est un format d'échange de données structurées entre différentes languages.
- Il permet de transférer d'un language à l'autre un tableau associatif.

Syntaxe JSON

- JSON = JavaScript Object Notation
- Un "objet" JSON peut contenir différents éléments:
  - nombre: 5
  - texte: "bonjour"
  - tableau: [item1, item2, item3]
  - objet: {"key1": value1, "key2": value2}
- Les éléments structurés peuvent être imbriqués.

Exemple JSON

http://localhost:8983/solr/select/?q=role:audio&rows=1&indent=on&wt=json


```
{
  "responseHeader":{
     "status":0,
     "QTime":4,
     "params":{
       "q":"role:audio",
       "start":"0",
       "rows":"1",
       "wt":"json",
       "ident":"on"}},
 "response":{"numFound":29,"start":0,"docs":[
    {
      "id":"audio-nicolas-Allumer le feu",
      "role":"audio",
      "date_creation":"2010-12-31T00:00:00Z",
      "date_modification":"2010-12-31T00:00:00Z",
      "url":"/comem/nicolas/audio/Allumer le feu",
      "album":"Ce que je sais",
      "auteur":"Hallyday Johnny",
      "taille":8.0,
      "titre":"Allumer le feu",
      "type":"mp3",
      "timestamp":"2011-12-05T13:29:55.045Z",
      "genre":["Rock"]}]
 }}
```

Utilisation de JSON

- Avec javascript, la construction de la page html va se faire côté client, à partir des donnée obtenues de Solr.
- Pour simplifier la programmation, on va utiliser la librairie JQuery.
- Cette librairie nous offre la méthode 'getJSON' qui permet d'obtenir un objet JS à partir d'une requête retournant un JSON.

Exemple avec JQuery

```
var request = {};
request['q'] = "*:*";
request['rows'] = 10;
request['start'] = 0;
request['wt'] = "json";
var url="http://localhost:8983/solr/select";
$.getJSON(url, request, function(result, status, data) {
      var documents = result.response.docs;
      for ( var i = 0; i < documents.length; i++) {
      displayDocument(documents[i]);
      }
   });
});
```