Solr tutorial

Table of contents

1 Overview..2

2 Requirements.. 2

3 Getting Started... 2

4 Indexing Data..3

5 Updating Data... 4

5.1 Deleting Data...4

6 Querying Data...5

6.1 Sorting.. 5

7 Text Analysis.. 6

7.1 Analysis Debugging.. 6

1. Overview

This document covers the basics of running Solr using an example schema, and some sample data.

2. Requirements

To follow along with this tutorial, you will need...

1. Java 1.5 or greater. Some places you can get it are from Sun, IBM, or BEA.
2. A Solr release.
3. On Win32, cygwin, for shell support. (If you plan to use Subversion on Win32, be sure to select the subversion package when you install, in the "Devel" category.) This tutorial will assume that "`sh`" is in your PATH, and that you have "curl" installed from the "Web" category.
4. FireFox or Mozilla is the preferred browser to view the admin pages... the current stylesheet doesn't currently look good on IE.

3. Getting Started

Begin by unziping the Solar release and changing your working directory to be the "`example`" directory

```
chrish@asimov:~/tmp/solr$ ls
solr-1.0.zip
chrish@asimov:~/tmp/solr$ unzip -q solr-1.0.zip
chrish@asimov:~/tmp/solr$ cd solr-1.0/example/
```

Solr can run in any Java Servlet Container of your choice, but to simplify this tutorial, the example index includes a small installation of Jetty.

To launch Jetty with the Solr WAR, and the example configs, just run the `start.jar` ...

```
chrish@asimov:~/tmp/solr/solr-1.0/example$ java -jar start.jar
1 [main] INFO org.mortbay.log - Logging to
org.slf4j.impl.SimpleLogger@1f436f5 via org.mortbay.log.Slf4jLog
334 [main] INFO org.mortbay.log - Extract
jar:file:/home/chrish/tmp/solr/solr-1.0/example/webapps/solr.war!/ to
/tmp/Jetty__solr/webapp
Feb 24, 2006 5:54:52 PM org.apache.solr.servlet.SolrServlet init
INFO: user.dir=/home/chrish/tmp/solr/solr-1.0/example
Feb 24, 2006 5:54:52 PM org.apache.solr.core.SolrConfig <clinit>
INFO: Loaded Config solrconfig.xml

...
```

```
1656 [main] INFO org.mortbay.log - Started SelectChannelConnector @
0.0.0.0:8983
```

This will start up the Jetty application server on port 8983, and use your terminal to display the logging information from Solr.

You can see that the Solr is running by loading http://localhost:8983/solr/admin/ in your web browser. This is the main starting point for Administering Solr.

4. Indexing Data

Your Solr port is up and running, but it doesn't contain any data. You can modify a Solr index by POSTing XML Documents containing instructions to add (or update) documents, delete documents, commit pending adds and deletes, and optimize your index. The `exampledocs` directory contains samples of the types of instructions Solr expects, as well as a Shell script for posting them using the command line utility "`curl`".

Open a new Terminal window, enter the exampledocs directory, and run the "`post.sh`" script on some of the XML files in that directory...

```
chrish@asimov:~/tmp/solr/solr-1.0/example/exampledocs$ sh post.sh solr.xml
Posting file solr.xml to http://localhost:8983/solr/update
<result status="0"></result>
<result status="0"></result>
```

You have now indexed one document about Solr, and committed that change. You can now search for "solr" using the "Make a Query" interface on the Admin screen, and you should get one result. Clicking the "Search" button should take you to the following URL...

http://localhost:8983/solr/select/?stylesheet=&q=solr&version=2.1&start=0&rows=10&indent=on

You can index all of the sample data, using the following command...

```
chrish@asimov:~/tmp/solr/solr-1.0/example/exampledocs$ sh post.sh *.xml
Posting file hd.xml to http://localhost:8983/solr/update
<result status="0"></result><result status="0"></result>
Posting file ipod_other.xml to http://localhost:8983/solr/update
<result status="0"></result><result status="0"></result>
Posting file ipod_video.xml to http://localhost:8983/solr/update
<result status="0"></result>
Posting file mem.xml to http://localhost:8983/solr/update
<result status="0"></result><result status="0"></result><result
status="0"></result>
Posting file monitor.xml to http://localhost:8983/solr/update
<result status="0"></result>
Posting file monitor2.xml to http://localhost:8983/solr/update
<result status="0"></result>
Posting file mp500.xml to http://localhost:8983/solr/update
<result status="0"></result>
```

```
Posting file sd500.xml to http://localhost:8983/solr/update
<result status="0"></result>
Posting file solr.xml to http://localhost:8983/solr/update
<result status="0"></result>
Posting file vidcard.xml to http://localhost:8983/solr/update
<result status="0"></result><result status="0"></result>
<result status="0"></result>
```

...and now you can search for all sorts of things using the default Lucene QueryParser syntax...

- video
- name:video
- +video +price:[* TO 400]

5. Updating Data

You may have noticed that even though the file `solr.xml` has now been POSTed to the server twice, you still only get 1 result when searching for "solr". This is because the example schema.xml specifies a "uniqueKey" field called "`id`". Whenever you POST instructions to Solr to add a document with the same value for the uniqueKey as an existing document, it automaticaly replaces it for you. You can see that that has happened by looking at the values for `numDocs` and `maxDoc` in the "CORE" section of the statistics page...

http://localhost:8983/solr/admin/stats.jsp

numDoc should be 15, but maxDoc may be larger (the maxDoc count includes logically deleted documents that have not yet been removed from the index). You can re-post the sample XML files over and over again as much as you want and numDocs will never increase, because the new documents will constantly be replacing the old.

Go ahead and edit the existing XML files to change some of the data, and re-run the post.sh command, you'll see your changes reflected in subsequent searches.

5.1. Deleting Data

You can delete data by POSTing a delete command to the update URL and specifying the value of the document's unique key field, or a query that matches multiple documents. Since these commands are smaller, we will specify them right on the command line rather than reference an XML file.

Execute the following command to delete a document

```
curl http://localhost:8983/solr/update --data-binary
'<delete><id>SP2514N</id></delete>'
```

Now if you go to the statistics page and scroll down to the UPDATE_HANDLERS section and verify that "`deletesPending : 1`"

If you search for id:SP2514N it will still be found, because index changes are not visible until changes are flushed to disk, and a new searcher is opened. To cause this to happen, send the following commit command to Solr:

```
curl http://localhost:8983/solr/update --data-binary '<commit/>'
```

Now re-execute the previous search and verify that no matching documents are found. Also revisit the statistics page and observe the changes in both the UPDATE_HANDLERS section and the CORE section.

Here is an example of using delete-by-query to delete anything with DDR in the name:

```
curl http://localhost:8983/solr/update --data-binary
'<delete><query>name:DDR</query></delete>'
curl http://localhost:8983/solr/update --data-binary '<commit/>'
```

Commit can be a very expensive operation so it's best to make many changes to an index in a batch and then send the commit command at the end. There is also an optimize command that does the same thing as commit, in addition to merging all index segments into a single segment, making it faster to search and causing any deleted documents to be removed. All of the update commands are documented here.

To continue with the tutorial, re-add any documents you may have deleted by going to the `exampledocs` directory and executing

```
sh post.sh *.xml
```

6. Querying Data

Searches are done via HTTP GET on the select URL with the query string in the q parameter. You can pass a number of optional request parameters to the request handler to control what information is returned. For example, you can use the "fl" parameter to control what stored fields are returned, and if the relevancy score is returned...

- q=video&fl=name,id (return only name and id fields)
- q=video&fl=name,id,score (return relevancy score as well)
- q=video&fl=*,score (return all stored fields, as well as relevancy score)
- q=video;price desc&fl=name,id (add sort specification: sort by price descending)

Solr provides a query form within the web admin interface that allows setting the various request parameters and is useful when trying out or debugging queries.

6.1. Sorting

Solr provides a simple extension to the Lucene QueryParser syntax for specifying sort options. After your search, add a semi-colon followed by a list of "field direction" pairs...

- video; price desc
- video; price asc
- video; inStock asc, price desc

"score" can also be used as a field name when specifying a sort...

- video; score desc
- video; inStock asc, score desc

If no sort is specified, the default is `score desc`, the same as in the Lucene search APIs.

7. Text Analysis

Text fields are typically indexed by breaking the field into words and applying various transformations such as lowercasing, removing plurals, or stemming to increase relevancy. The same text transformations are normally applied to any queries in order to match what is indexed.

Example queries demonstrating relevancy improving transformations:

- A search for power-shot matches `PowerShot`, and adata matches `A-DATA` due to the use of WordDelimiterFilter and LowerCaseFilter.
- A search for name:printers matches `Printer`, and features:recharging matches `Rechargeable` due to stemming with the EnglishPorterFilter.
- A search for "1 gigabyte" matches things with `GB`, and pixima matches `Pixma` due to use of a SynonymFilter.

The schema defines the fields in the index and what type of analysis is applied to them. The current schema your server is using may be accessed via the `[SCHEMA]` link on the admin page.

A full description of the analysis components, Analyzers, Tokenizers, and TokenFilters available for use is here.

7.1. Analysis Debugging

There is a handy analysis debugging page where you can see how a text value is broken down into words, and shows the resulting tokens after they pass through each filter in the chain.

This shows how "`Canon PowerShot SD500`" would be indexed as a value in the name field. Each row of the table shows the resulting tokens after having passed through the next

TokenFilter in the Analyzer for the `name` field. Notice how both `powershot` and `power`, `shot` are indexed. Tokens generated at the same position are shown in the same column, in this case `shot` and `powershot`.

Selecting verbose output will show more details, such as the name of each analyzer component in the chain, token positions, and the start and end positions of the token in the original text.

Selecting highlight matches when both index and query values are provided will take the resulting terms from the query value and highlight all matches in the index value analysis.

Here is an example of stemming and stop-words at work.