Solr tutorial

Table of contents

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 1.6 or greater. Some places you can get it are from Oracle, Open JDK, IBM, or Running `java -version` at the command line should indicate a version number starting with 1.6. Gnu's GCJ is not supported and does not work with Solr.
2. A Solr release.

3 Getting Started

Please run the browser showing this tutorial and the Solr server on the same machine so tutorial links will correctly point to your Solr server.

Begin by unziping the Solr release and changing your working directory to be the "`example`" directory. (Note that the base directory name may vary with the version of Solr downloaded.) For example, with a shell in UNIX, Cygwin, or MacOS:

```
user:~solr$ ls
solr-nightly.zip
user:~solr$ unzip -q solr-nightly.zip
user:~solr$ cd solr-nightly/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` ...

```
user:~/solr/example$ java -jar start.jar
2009-10-23 16:42:53.816::INFO:  Logging to STDERR via org.mortbay.log.StdErrLog
2009-10-23 16:42:53.907::INFO:  jetty-6.1.26

...

Oct 23, 2009 4:41:56 PM org.apache.solr.core.SolrCore registerSearcher
INFO: [] Registered new searcher Searcher@7c3885 main
```

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 server 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 java utility for posting them from the command line (a `post.sh` shell script is also available, but for this tutorial we'll use the cross-platform Java client).

To try this, open a new terminal window, enter the exampledocs directory, and run "`java -jar post.jar`" on some of the XML files in that directory, indicating the URL of the Solr server:

```
user:~/solr/example/exampledocs$ java -jar post.jar solr.xml monitor.xml
SimplePostTool: version 1.2
SimplePostTool: WARNING: Make sure your XML documents are encoded in UTF-8, other encodings
 are not currently supported
SimplePostTool: POSTing files to http://localhost:8983/solr/update..
SimplePostTool: POSTing file solr.xml
SimplePostTool: POSTing file monitor.xml
SimplePostTool: COMMITting Solr index changes..
```

You have now indexed two documents in Solr, and committed these changes. 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/?q=solr&start=0&rows=10&indent=on

You can index all of the sample data, using the following command (assuming your command line shell supports the *.xml notation):

```
  user:~/solr/example/exampledocs$ java -jar post.jar *.xml
SimplePostTool: version 1.2
SimplePostTool: WARNING: Make sure your XML documents are encoded in UTF-8, other encodings
 are not currently supported
SimplePostTool: POSTing files to http://localhost:8983/solr/update..
SimplePostTool: POSTing file hd.xml
SimplePostTool: POSTing file ipod_other.xml
SimplePostTool: POSTing file ipod_video.xml
SimplePostTool: POSTing file mem.xml
SimplePostTool: POSTing file monitor.xml
SimplePostTool: POSTing file monitor2.xml
SimplePostTool: POSTing file mp500.xml
SimplePostTool: POSTing file sd500.xml
SimplePostTool: POSTing file solr.xml
SimplePostTool: POSTing file spellchecker.xml
```

```
SimplePostTool: POSTing file utf8-example.xml
SimplePostTool: POSTing file vidcard.xml
SimplePostTool: COMMITting Solr index changes..
```

...and now you can search for all sorts of things using the default Solr Query Syntax (a superset of the Lucene query syntax)...

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

There are many other different ways to import your data into Solr... one can

- Import records from a database using the Data Import Handler (DIH).
- Load a CSV file (comma separated values), including those exported by Excel or MySQL.
- POST JSON documents
- Index binary documents such as Word and PDF with Solr Cell (ExtractingRequestHandler).
- Use SolrJ for Java or other Solr clients to programatically create documents to send to Solr.

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 automatically replaces it for you. You can see that that has happened by looking at the values for `numDocs` and `maxDoc` in the "CORE"/searcher section of the statistics page...

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

`numDocs` represents the number of searchable documents in the index (and will be larger than the number of XML files since some files contained more than one `<doc>`). **`maxDoc`** may be larger as 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 `java -jar post.jar` 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 (be careful with that one!). 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

```
java -Ddata=args -Dcommit=no -jar post.jar "<delete><id>SP2514N</id></delete>"
```

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

If you search for id:SP2514N it will still be found, because index changes are not visible until changes are committed and a new searcher is opened. To cause this to happen, send a commit command to Solr (post.jar does this for you by default):

```
java -jar post.jar
```

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:

```
java -Ddata=args -jar post.jar "<delete><query>name:DDR</query></delete>"
```

Commit can be an 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

```
java -jar post.jar *.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&sort=price desc&fl=name,id,price (add sort specification: sort by price descending)
- q=video&wt=json (return response in JSON format)

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

6.1 Sorting

Solr provides a simple method to sort on one or more indexed fields. Use the "`sort`' parameter to specify "field direction" pairs, separated by commas if there's more than one sort field:

- q=video&sort=price desc
- q=video&sort=price asc
- q=video&sort=inStock asc, price desc

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

- q=video&sort=score desc
- q=video&sort=inStock asc, score desc

Complex functions may also be used to sort results:

- q=video&sort=div(popularity,add(price,1)) desc

If no sort is specified, the default is `score desc` to return the matches having the highest relevancy.

7 Highlighting

Hit highlighting returns relevent snippets of each returned document, and highlights terms from the query within those context snippets.

The following example searches for `video card` and requests highlighting on the fields `name,features`. This causes a `highlighting` section to be added to the response with the words to highlight surrounded with `<em>` (for emphasis) tags.

...&q=video card&fl=name,id&hl=true&hl.fl=name,features

More request parameters related to controlling highlighting may be found here.

8 Faceted Search

Faceted search takes the documents matched by a query and generates counts for various properties or categories. Links are usually provided that allows users to "drill down" or refine their search results based on the returned categories.

The following example searches for all documents (`*:*`) and requests counts by the category field `cat`.

...&q=*:*&facet=true&facet.field=cat

Notice that although only the first 10 documents are returned in the results list, the facet counts generated are for the complete set of documents that match the query.

We can facet multiple ways at the same time. The following example adds a facet on the boolean `inStock` field:

...&q=*:*&facet=true&facet.field=cat&facet.field=inStock

Solr can also generate counts for arbitrary queries. The following example queries for `ipod` and shows prices below and above 100 by using range queries on the price field.

...&q=ipod&facet=true&facet.query=price:[0 TO 100]&facet.query=price:[100 TO *]

One can even facet by date ranges. This example requests counts for the manufacture date (`manufacturedate_dt` field) for each year between 2004 and 2010.

...&q=*:*&facet=true&facet.date=manufacturedate_dt&facet.date.start=2004-01-01T00:00:00Z&facet.date.e
+1YEAR

More information on faceted search may be found on the faceting overview and faceting parameters pages.

9 Search UI

Solr includes an example search interface built with velocity templating that demonstrates many features, including searching, faceting, highlighting, autocomplete, and geospatial searching.

Try it out at http://localhost:8983/solr/browse

10 Text Analysis

Text fields are typically indexed by breaking the text 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.

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.

The best analysis components (tokenization and filtering) for your textual content depends heavily on language. As you can see in the above `[SCHEMA]` link, the fields in the example schema are using a `fieldType` named `text_general`, which has defaults appropriate for all languages.

If you know your textual content is English, as is the case for the example documents in this tutorial, and you'd like to apply English-specific stemming and stop word removal, as well as split compound words, you can use the `text_en_splitting` fieldType instead. Go ahead and edit the `schema.xml` under the `solr/example/solr/conf` directory, and change the `type` for fields `text` and `features` from `text_general` to `text_en_splitting`. Restart the server and then re-post all of the documents, and then these queries will show the English-specific transformations:

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

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

10.1 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 Power-Shot 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.

11 Conclusion

Congratulations! You successfully ran a small Solr instance, added some documents, and made changes to the index and schema. You learned about queries, text analysis, and the Solr admin interface. You're ready to start using Solr on your own project! Continue on with the following steps:

- Subscribe to the Solr mailing lists!
- Make a copy of the Solr `example` directory as a template for your project.
- Customize the schema and other config in `solr/conf/` to meet your needs.

Solr has a ton of other features that we haven't touched on here, including distributed search to handle huge document collections, function queries, numeric field statistics, and search results clustering. Explore the Solr Wiki to find more details about Solr's many features.

Have Fun, and we'll see you on the Solr mailing lists!