PacktBook2009

`https://www.packtpub.com/sites/default/files/imagecache/productview/bookimages/5883_MockupCover.jpg`

Book Info

Title	Solr 1.4 Enterprise Search Server
Publisher	Packt
Language	English
Format	Paperback
Release date	August 2009
ISBN 10	1847195881
ISBN 13	978-1-847195-88-3
Authors	David Smiley, Eric Pugh
Pages	336
Dimensions	191mm x 235mm

The first comprehensive resource guide on Solr, written and tested for the 1.4 version. Essential reading for developers, this book covers nearly every feature with a tutorial approach and realistic examples.

Available For Purchase...

- Directly from Packt (A portion of proceeds are donated to the ASF)
- From Amazon

Addendum

This book aimed to cover all the features in Solr 1.4 but some features were overlooked at the time of writing or were implemented after the book was published. This document is a listing of the missed content organized by the chapter it would most likely have been added to. There are some other known "features" in Solr that are not in the book and aren't here because they are either internal to Solr or have dubious purpose or value.

Chapter 2: Schema and Text Analysis

Trie based field types

The schema.xml used in the book examples has a schema version 1.1 instead of 1.2 which is Solr 1.4's new default. The distinction is fairly trivial. The bigger difference is that Solr 1.4 defines a set of "Trie" based field types which are used in preference to the "Sortable" based ones. For example, there is now a `TrieIntField` using a field type named `tint` which is to be used in preference to `SortableIntField` with a field type named `sint`. The trie field types have improved performance characteristics, particularly for range queries, and they are of course sortable. However, the "Sortable" field variants still do one thing that the trie based fields cannot do which is the ability to specify `sortMissingLast` and `sortMissingFirst`. There is further documentation about these field types in the Solr 1.4 example schema.xml file.

Text Analysis

- ReverseWildcardFilter

There is support for leading wildcards when using ReverseWildcardFilterFactory. See that link for some configuration options. For example, using this filter allows a query `*book` to match the text `cookbook`. It essentially works by reversing the words as indexed variants. Be aware that this can more than double the number of indexed terms for the field and thus increase the disk usage proportionally. For a configuration snippet using this feature, consider this sample field type definition excerpted from the unit tests:

```
<fieldtype name="srev" class="solr.TextField">
  <analyzer type="index">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.ReversedWildcardFilterFactory" withOriginal="true"
        maxPosAsterisk="3" maxPosQuestion="2" maxFractionAsterisk="0.33"/>
  </analyzer>

  <analyzer type="query">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>

  </analyzer>
</fieldtype>
```

An interesting under-the-hood detail is that this filter requires the Solr query parsing code to check for the presence of this filter to change its behavior – something not true for any other filter.

- ASCIIFoldingFilter

For mapping of non-ascii characters to reasonable ASCII equivalents use `ASCIIFoldingFilterFactory` which is best documented here.

- WordDelimiterFilter

There are a couple extra options for this filter not covered in the book. One is an option `stemEnglishPossessive` which is either 1 to enable (the default) or 0. When enabled it strips off trailing `'s` on words. For example "O'Neil's" becomes "O", "Neil". Another point is that this filter supports the same `protected` attribute that the stemmer filters do so that you can exclude certain input tokens listed in a configuration file from word delimiter processing.

Misc

- copyField maxChars

The copyField directive in the schema can contain an optional `maxChars` attribute which puts a cap on the number of characters copied. This is useful for copying potentially large text fields into a catch-all searched field.

- ExternalFileField

There is a field type you can use called `ExternalFileField` that only works when referenced in function queries. As its name suggests, its data is in an external file instead of in the index. It's suitability is only for manipulating boosts of a document without requiring re-indexing the document. There is some rudimentary javadocs but you'll want to search solr's mailing list for further info.

Chapter 3: Indexing Data

- Duplicate detection

In some Solr usage situations you may need to prevent duplication where documents that are the same could get added. This is called *deduplication*. This doesn't have to do with your unique key field, it is for when there is some other text field(s) that should be unique, perhaps from a crawled file. This feature is documented on Solr's wiki.

Automatically Committing

The book discusses how to explicitly commit added data. Solr can also be configured to automatically commit. This feature is particularly useful when updating the index with changed data as it occurs externally.

- autoCommit

In solrconfig.xml there is an <updateHandler> configuration element. Within it there is the following XML commented in the default configuration:

```
<autoCommit>
  <maxDocs>10000</maxDocs>
  <maxTime>1000</maxTime>
</autoCommit>
```

You can specify `maxDocs` and/or `maxTime` depending on your needs. `maxDocs` simply sets a threshold at which a commit happens if there are this many documents not yet committed. Most useful is `maxTime` (milliseconds) which essentially sets a count-down timer from the first document added after the previous commit for a commit to occur automatically. The only problem with using these is that it can't be disabled, which is something you might want to do for bulk index loads. Instead, consider `commitWithin` described below.

- commitWithin

When submitting documents to Solr, you can include a "commitWithin" attribute placed on the `<add/>` XML element. When >= 0, this tells Solr to perform a commit no later than this number of milliseconds relative to the time Solr finishes processing the data. Essentially it acts as an override to solrconfig.xml / updateHandler / autoCommit / maxTime.

Misc

I'd like to simply re-emphasize that the book covered the `DataImportHandler` fairly lightly. For the latest documentation, go to Solr's wiki.

- ContentStreamDataSource

One unique way to use the `DataImportHandler` is using the `ContentStreamDataSource`. It is like the `URLDataSource` except that instead of the DIH going out to fetch the XML, XML can be POST'ed to the DIH from some other system (i.e. pull vs push). Coupled together with the DIH's XSLT support, this is fairly powerful. The following is a snippet of `solrconfig.xml` and then an entire DIH configuration file, referencing this `DataSource` type and using XSL.

```
  <requestHandler name="/update/musicbrainz" class="org.apache.solr.handler.dataimport.DataImportHandler">
    <lst name="defaults">
      <str name="config">dih-musicbrainz-post-config.xml</str>
      <str name="optimize">false</str>
      <str name="clean">false</str>
      <str name="command">full-import</str>
    </lst>
  </requestHandler>
```

```
 <dataConfig>
  <dataSource type="ContentStreamDataSource" />
  <document>
    <entity name="mentity"
            xsl="xslt/musicbrains2solr.xsl"
            useSolrAddSchema="true"
            processor="XPathEntityProcessor">
    </entity>
  </document>
</dataConfig>
```

Chapter 5: Enhanced Searching

- QParserPlugin and LocalParams syntax and subqueries Another modification Solr has to Lucene's query syntax which is the use of {`{!qparser name=value name2=value2} yourquery`}. That is, at the very beginning of a query you can use this syntax to indicate a different query parser (optionally) and specify some so-called "local params" name-value pairs too (again, optionally), used for certain advanced cases. Solr's wiki has a bit more information on this. And in addition, there is a *query* pseudo field hack in the query syntax to support subqueries which is useful when used with the aforementioned QParserPlugin syntax to change the query type. Aside from Solr's wiki, you will also find this blog post by Yonik enlightening.

Function queries

The main reference for function queries is here at Solr's wiki. The following are the ones not covered in the book:

- sub(x,y) Subtracts: x - y
- query(subquery,default) This one is a bit tough to understand. It yields the *score* for this document as found from the given sub-query, defaulting to the 2nd argument if not found in that query. There are some interesting examples on the wiki.
- ms(), ms✖, ms(x,y) The `ms` function deals with times in milliseconds since the common 1970 epoch. Arguments either refer to a date field or it is a literal (ex: 2000-01-01T00:00:00Z). Without arguments it returns the current time. One argument will return the time referenced, probably a field reference. When there are two, it returns the difference `x-y`. This function is useful when boosting more recent documents sooner. There is excellent information on this subject at the wiki.
- Function Range Queries Functions Queries can also be used for filtering searches. Using the `frange` QParserPlugin, you specify a numeric range applied on the given function query. This advanced technique is best described at Yonik's blog post at Lucid Imationation.

Chapter 6: Search Components

Clustering Component

This is a Solr "contrib" module and was incorporated in the Solr 1.4 distribution near the end of the book's release. This component will "cluster" the search results based on statistical similarity of terms. It uses the Carrot2 open-source project as the implementation of the underlying algorithm. Clustering is useful for large text-heavy indexes, especially when there is little/no structural information for faceting.

More details: ClusteringComponent

Chapter 7: Deployment

- XInclude The `solrconfig.xml` file can be broken up into pieces and then included using the XInclude spec. An example of this is the following line:

  ```
  <xi:include href="solr/conf/solrconfig_master.xml" xmlns:xi="http://www.w3.org/2001/XInclude"/>
  ```

  This is particularly useful when there are multiple Solr cores that require only slightly different configurations. The common parts could be put into a file that is included into each config. There is more information about this at Solr's wiki.

Chapter 8: Integrating Solr

- VelocityResponseWriter

Solr incorporates a contrib module called Apache Velocity] (AKA Solritas). By using a special request handler, you can rapidly construct user web front-ends using the [http://velocity.apache.org/ templating system. It isn't expected that you would build sites with this, just proof-of-concepts.

- AJAX-Solr forks from SolrJs AJAX Solr is another option for browser JavaScript integration with Solr. Unlike SolrJs (from which it derives), AJAX-Solr is not tied to JQuery or any other JavaScript framework for that matter.

- Native PHP support PHP5 now has a client API for interacting with Solr.

Chapter 9: Scaling Solr

- partial optimize If the index is so large that optimizes are taking longer than desired or using more disk space during optimization than you can spare, consider adding the `maxSegments` parameter to the optimize command. In the XML message, this would be an attribute; the URL form and SolrJ have the corresponding option too. By default this parameter is 1 since an optimize results in a single Lucene "segment". By setting it larger than 1 but less than the `mergeFactor`, you permit partial optimization to no more than this many segments. Of course the index won't be fully optimized and therefore searches will be slower.