a.k.a the "Solr Cell" project!

Contents

Introduction

Solr1.4

A common need of users is the ability to ingest binary and/or structured documents such as Office, Word, PDF and other proprietary formats. The Apache Tika project provides a framework for wrapping many different file format parsers, such as PDFBox, POI and others.

Solr's ExtractingRequestHandler uses Tika to allow users to upload binary files to Solr and have Solr extract text from it and then index it.

Concepts

Before getting started, there are a few concepts that are helpful to understand.

- Tika will automatically attempt to determine the input document type (word, pdf, etc.) and extract the content appropriately. If you want, you can explicitly specify a MIME type for Tika wth the stream.type parameter
- Tika does everything by producing an XHTML stream that it feeds to a SAX

ContentHandler.

- Solr then reacts to Tika's SAX events and creates the fields to index.
- Tika produces Metadata information such as Title, Subject, and Author, according to specifications like DublinCore. See http://tika.apache.org/ site for the file types supported.
- All of the extracted text is added to the "content" field
- We can map Tika's metadata fields to Solr fields. We can boost these fields
- We can also pass in literals for field values.
- We can apply an XPath expression to the Tika XHTML to restrict the content that is produced.

Getting Started with the Solr Example

- Check out Solr trunk or get a 1.4 release or later.
- Note: if using a check out of the Solr source code instead of a binary release, running "ant example" will build the necessary jars.

Now start the solr example server:

```
cd example
java -jar start.jar
```

In a separate window go to the `docs/` directory (which contains some nice example docs), or the `site` directory if you built Solr from source, and send Solr a file via HTTP POST:

```
cd site/html
curl "http://localhost:8983/solr/update/extract?
literal.id=doc1&commit=true" -F "myfile=@tutorial.html"
```

- Note, the /site directory in the solr download contains some nice example docs to try
- hint: myfile=@tutorial.html needs a valid path (absolute or relative), e.g. "myfile=@../../site/tutorial.html" if you are still in exampledocs dir.
- the `literal.id=doc1` param provides the necessary unique id for the document being indexed
- the `commit=true` param causes Solr to do a commit after indexing the document, making it immediately searchable. For good performance when loading many documents, don't call commit until you are done.
- using "curl" or other command line tools to post documents to Solr is nice for testing, but not the recommended update method for best performance.

Now, you should be able to execute a query and find that document (open the following link in your browser): http://localhost:8983/solr/select?q=tutorial

You may notice that although you can search on any of the text in the sample document, you may not be able to see that text when the document is retrieved. This is simply because the "content" field generated by Tika is mapped to the Solr field called "text", which is indexed but not stored. This is done via the default map rule in the `/update/extract` handler in `solrconfig.xml` and can be easily changed or overridden. For example, to store and see all metadata and content, execute the following:

```
curl "http://localhost:8983/solr/update/extract?
literal.id=doc1&uprefix=attr_&fmap.content=attr_content&commit=true"
-F "myfile=@tutorial.html"
```

- The `uprefix=attr_` param causes all generated fields that aren't defined in the schema to be prefixed with attr_ (which is a dynamic field that is stored).
- The `fmap.content=attr_content` param overrides the default `fmap.content=text` causing the content to be added to the attr_content field instead.

And then query via http://localhost:8983/solr/select?q=attr_content:tutorial

Input Parameters

- fmap.<source_field>=<target_field> - Maps (moves) one field name to another. Example: `fmap.content=text` will cause the content field normally generated by Tika to be moved to the "text" field.
- boost.<fieldname>=<float> - Boost the specified field.
- literal.<fieldname>=<value> - Create a field with the specified value. May be multivalued if the Field is multivalued.
- uprefix=<prefix> - Prefix all fields that are not defined in the schema with the given prefix. This is very useful when combined with dynamic field definitions. Example: `uprefix=ignored_` would effectively ignore all unknown fields generated by Tika given the example schema contains `<dynamicField name="ignored_*" type="ignored"/>`
- defaultField=<Field Name> - If uprefix is not specified and a Field cannot be determined, the default field will be used.
- extractOnly=true|false - Default is false. If true, return the extracted content from Tika without indexing the document. This literally includes the extracted XHTML as a string in the response. When viewing manually, it may be useful to use a response format other than XML to aid in viewing the embedded XHTML tags. See TikaExtractOnlyExampleOutput.
- resource.name=<File Name> - The optional name of the file. Tika can use it as a hint for detecting mime type.
- capture=<Tika XHTML NAME> - Capture XHTML elements with the name separately for adding to the Solr document. This can be useful for grabbing chunks of the XHTML into a separate field. For instance, it could be used to grab paragraphs (<p>) and index them into a separate field. Note that content is also still captured into the overall "content" field.
- captureAttr=true|false - Index attributes of the Tika XHTML elements into separate fields, named after the element. For example, when extracting from HTML, Tika can return the href attributes in <a> tags as fields named "a". See the examples below.
- xpath=<XPath expression> - When extracting, only return Tika XHTML content that

satisfies the XPath expression. See http://tika.apache.org/1.2/parser.html for details on the format of Tika XHTML. See also TikaExtractOnlyExampleOutput.

- lowernames=true|false - Map all field names to lowercase with underscores. For example, Content-Type would be mapped to content_type.
- literalsOverride=true|false - Solr4.0 When true, literal field values will override other values with same field name, such as metadata and content. If false, then literal field values will be appended to any extracted data from Tika, and the resulting field needs to be multi valued. Default: true
- resource.password=<password> - Solr4.0 The optional password for a password protected PDF or OOXML file. File format support depends on Tika.
- passwordsFile=<file name> - Solr4.0 The optional name of a file containing file name pattern to password mappings. See chapter "Encrypted Files" below

If extractOnly is true, additional input parameters:

- extractFormat=xml|text - Default is xml. Controls the serialization format of the extract content. xml format is actually XHTML, like passing the -x command to the tika command line application, while text is like the -t command.

Order of field operations

1. fields are generated by Tika or passed in as literals via `literal.fieldname=value`. Before Solr4.0 or if literalsOverride=false, then literals will be appended as multi-value to tika generated field.
2. if lowernames==true, fields are mapped to lower case
3. mapping rules `fmap.source=target` are applied
4. if `uprefix` is specified, any unknown field names are prefixed with that value, else if `defaultField` is specified, unknown fields are copied to that.

Configuration

The ExtractingRequestHandler is not incorporated into the solr war file, it is provided as a SolrPlugins, and you have to load it (and it's dependencies) explicitly.

Example configuration for loading plugin and dependencies:

```
  <lib dir="../../dist/" regex="apache-solr-cell-\d.*\.jar" />
  <lib dir="../../contrib/extraction/lib" regex=".*\.jar" />
```

Example configuration for the Handler:

```
<requestHandler name="/update/extract"
class="org.apache.solr.handler.extraction.ExtractingRequestHandler">
    <lst name="defaults">
      <str name="fmap.Last-Modified">last_modified</str>
      <str name="uprefix">ignored_</str>
    </lst>
    <!--Optional.  Specify a path to a tika configuration file.  See
the Tika docs for details.-->
    <str name="tika.config">/my/path/to/tika.config</str>
    <!-- Optional. Specify one or more date formats to parse.  See
DateUtil.DEFAULT_DATE_FORMATS for default date formats -->
    <lst name="date.formats">
      <str>yyyy-MM-dd</str>
    </lst>
  </requestHandler>
```

In the defaults section, we are mapping Tika's Last-Modified Metadata attribute to a field named last_modified. We are also telling it to ignore undeclared fields. These are all overridden parameters.

The tika.config entry points to a file containing a Tika configuration. You would only need this if you have customized your own Tika configuration. The Tika config contains info about parsers, mime types, etc.

You may also need to adjust the `multipartUploadLimitInKB` attribute as follows if you are submitting very large documents.

```
  <requestDispatcher handleSelect="true" >
    <requestParsers enableRemoteStreaming="{true|false}"
multipartUploadLimitInKB="2048000" />
    ....
```

For remote streaming, you must enable remote stream. See ContentStream for more info or just set enableRemoteStreaming=true in the snippet above. As an example of using remote streaming, you can do:

```
 curl "http://localhost:8983/solr/update/extract?
stream.file=/path/to/file/StatesLeftToVisit.doc&stream.contentType=a
pplication/msword&literal.id=states.doc"
```

Lastly, the date.formats allows you to specify various java.text.SimpleDateFormat date formats for working with transforming extracted input to a Date. Solr comes configured with the following date formats (see the DateUtil class in Solr)

```
yyyy-MM-dd'T'HH:mm:ss'Z'
yyyy-MM-dd'T'HH:mm:ss
yyyy-MM-dd
yyyy-MM-dd hh:mm:ss
yyyy-MM-dd HH:mm:ss
EEE MMM d hh:mm:ss z yyyy
EEE, dd MMM yyyy HH:mm:ss zzz
EEEE, dd-MMM-yy HH:mm:ss zzz
EEE MMM d HH:mm:ss yyyy
```

MultiCore config

- For multi-core, specify `sharedLib='lib'` in `<solr />` in example/solr/solr.xml in order for Solr to find the jars in example/solr/lib

- Lib resources in solrconfig.xml must point to the lib folder relative form where the actual used solrconfig.xml.
- For multi cores with common solrconfig and schema the can use the same instanceDir

Metadata

As has been implied up to now, Tika produces Metadata about the document. Metadata often contains things like the author of the file or the number of pages, etc. The Metadata produced depends on the type of document submitted. For instance, PDFs have different metadata from Word docs.

In addition to Tika's metadata, Solr adds the following metadata (defined in ExtractingMetadataConstants):

- "stream_name" - The name of the ContentStream as uploaded to Solr. Depending on how the file is uploaded, this may or may not be set.
- "stream_source_info" - Any source info about the stream. See ContentStream.
- "stream_size" - The size of the stream in bytes(?)
- "stream_content_type" - The content type of the stream, if available.

It is highly recommend that you try using the extract only option to see what values actually get set for these.

Encrypted files

Solr4.0 By supplying a password in either `resource.password` on the request, or in a `passwordsFile` file, you can have ExtractingRequestHandler decrypt encrypted files and index their content. In the case of `passwordsFile` the file supplied must be on the format: One line per rule, each rule contains a file name regular expression followed by "=" followed by the password in clear-text (thus this file should have strict access restrictions).

```
# This is a comment
myFileName = myPassword
.*\.docx$ = myWordPassword
.*\.pdf$ = myPdfPassword
```

Examples

Mapping and Capture

Capture <div> tags separate, and then map that field to a dynamic field named foo_txt.

```
 curl "http://localhost:8983/solr/update/extract?
literal.id=doc2&captureAttr=true&defaultField=text&fmap.div=foo_txt&
capture=div"  -F "tutorial=@tutorial.pdf"
```

Mapping, Capture and Boost

Capture <div> tags separate, and then map that field to a dynamic field named foo_txt. Boost foo_txt by 3.

```
curl "http://localhost:8983/solr/update/extract?
literal.id=doc3&captureAttr=true&defaultField=text&capture=div&fmap.
div=foo_txt&boost.foo_txt=3" -F "tutorial=@tutorial.pdf"
```

Literals

To add in your own metadata, pass in the literal parameter along with the file:

```
curl "http://localhost:8983/solr/update/extract?
literal.id=doc4&captureAttr=true&defaultField=text&capture=div&fmap.
div=foo_txt&boost.foo_txt=3&literal.blah_s=Bah"  -F
"tutorial=@tutorial.pdf"
```

XPath

Restrict down the XHTML returned by Tika by passing in an XPath expression

```
curl "http://localhost:8983/solr/update/extract?
literal.id=doc5&captureAttr=true&defaultField=text&capture=div&fmap.
div=foo_txt&boost.foo_txt=3&literal.id=id&xpath=/xhtml:html/xhtml:bo
dy/xhtml:div/descendant:node()"  -F "tutorial=@tutorial.pdf"
```

Extract Only

```
curl "http://localhost:8983/solr/update/extract?&extractOnly=true"
--data-binary @tutorial.html  -H 'Content-type:text/html'
```

A the output includes XML generated by Tika and is thus further escaped by Solr's XML format. Using a different output format like json or ruby enhances the readability:

```
curl "http://localhost:8983/solr/update/extract?
&extractOnly=true&wt=ruby&indent=true"  --data-binary @tutorial.html
-H 'Content-type:text/html'
```

See TikaExtractOnlyExampleOutput.

Password protected

```
curl "http://localhost:8983/solr/collection1/update/extract?
commit=true&literal.id=123&resource.password=mypassword" \
     -H "Content-Type: application/pdf" --data-binary @my-encrypted-
file.pdf
```

Sending documents to Solr

The ExtractingRequestHandler can process any document sent as a ContentStream ...

- Raw POST
- Multi-part file upload (each file is processed as a distinct document)
- "stream.body", "stream.url" and "stream.file" request params.

Example...

```
curl "http://localhost:8983/solr/update/extract?
literal.id=doc5&defaultField=text"  --data-binary @tutorial.html  -H
'Content-type:text/html'
```

NOTE, this literally streams the file as the body of the POST, which does not, then, provide info to Solr about the name of the file.

SimplePostTool (post.jar)

The simple post tool post.jar which ships with Solr in the `example/exampledocs` folder can post a file to ExtractingRequestHandler:

```
java -Durl=http://localhost:8983/solr/update/extract -
Dparams=literal.id=doc5 -Dtype=text/html -jar post.jar tutorial.html
```

Since Solr4.0 post.jar also has an `auto` mode which guesses content-type for you, and also sets a default ID and filename when sending to Solr. Also, a `recursive` option lets you automatically post a whole directory tree:

```
java -Dauto -jar post.jar tutorial.html
java -Dauto -Drecursive -jar post.jar .
```

NOTE: The post.jar utility is not meant for production use, but as a convenience tool for experimenting with Solr. It is made as a single .java file (see SVN) without dependencies, so it does on purpose not use SolrJ.

SolrJ

Use the ContentStreamUpdateRequest (see ContentStreamUpdateRequestExample for a full example):

Toggle line numbers

```
ContentStreamUpdateRequest up = new
ContentStreamUpdateRequest("/update/extract");
up.addFile(new File("mailing_lists.pdf"));
up.setParam("literal.id", "mailing_lists.pdf");
up.setAction(AbstractUpdateRequest.ACTION.COMMIT, true,
true);
```

```
result = server.request(up);
assertNotNull("Couldn't upload mailing_lists.pdf", result);
rsp = server.query( new SolrQuery( "*:*") );
Assert.assertEquals( 1, rsp.getResults().getNumFound() );
```

If you want to set a **multiValued** field, use the *ModifiableSolrParams* class like this:

Toggle line numbers

```
ModifiableSolrParams p = new ModifiableSolrParams();
for(String value : values) {
    p.add(ExtractingParams.LITERALS_PREFIX + "field", value);
}
up.setParams(p);
```

You could also set all of the other literals and parameters in this class, and then use the *setParams* method to apply the changes to your content stream.

Extending the ExtractingRequestHandler

If you wish to supply your own ContentHandler for Solr to use, you can extend the ExtractingRequestHandler and override the createFactory() method. This factory is responsible for constructing the SolrContentHandler that interacts with Tika.

Committer Notes

Upgrading Tika

- Run `ant clean clean-jars` from project root or `/solr`
- Update `solr/CHANGES.txt` and `solr/NOTICE.txt` as well as relevant LICENSE/NOTICE files in `solr/licenses/`
- Edit `solr/contrib/extraction/ivy.xml` according to instructions inline the file
- `ant jar-checksums` - to generate new sha1 files for the jars
- `ant test` - see that Ivy downloads the new dependencies
- Commit

Additional Resources

- Lucid Imagination article
- Supported document formats via Tika (1.2)

What's in a Name

Grant was writing the javadocs for the code and needed an entry for the <title> tag and wrote out "Solr Content Extraction Library", since the contrib directory is named "extraction". This then lead to an "acronym": Solr CEL which then gets mashed to: Solr Cell. Hence, the project name is "Solr Cell". It's also appropriate because a Solar Cell's job is to convert the raw energy of the Sun to electricity, and this contrib's module is responsible for converting the "raw" content of a document to something usable by Solr. http://en.wikipedia.org/wiki/Solar_cell

ExtractingRequestHandler (last edited 2013-05-08 19:10:53 by YonikSeeley)