Sitecore Search Advanced Database Crawler and PDF File Content Searching

by Mark S on November 13, 2012

One of the projects we currently are participating in is a site redesign using Sitecore for a local client. This implementation is based on Sitecore 6.6 and their search involves use of Alex Shyba's Advanced Database Crawler (ADC) - http://sitecoreblog.alexshyba.com/2010/11/sitecore-searcher-and-advanced-database.html.

Sitecore's search APIs sit on top of Lucene.NET and the ADC essentially sits on top of that stack. In my opinion, using the ADC is the preferred way to retrieve Sitecore content. It's responsive, flexible, and can be controlled easily.

Our client had a need to be able to incorporate the ability to search the content of their PDF library and to incorporate those results in their particular ordering scheme. ADC while great in searching Sitecore, doesn't natively handle search file content. In order to meet this requirement of our client, decrease development costs and keep their ownership costs to a minimum, we ended up going with a model that consists of breaking up the search into three unique portions and then combining those results into the ADC SkinnyItems collection.

First the initial search index was broken down into the different separate indexes. The first index focused on their product offerings. These results were to be their highest rated results. The index in the configuration file was like the one below:

```
<index id="ProductRelated" type="Sitecore.Search.Index, Sitecore.Kernel">
<param desc="name">$(id)</param>
       <param desc="folder">ProductRelated</param>
       <Analyzer ref="search/analyzer" />
       <locations hint="list:AddCrawler">
               <products
type="scSearchContrib.Crawler.Crawlers.AdvancedDatabaseCrawler,scSearchContrib.Crawler">
                      <Database>web</Database>
                      <Root>/sitecore/productcatalog</Root>
                      <IndexAllFields>true</IndexAllFields>
                      <include hint="list:IncludeTemplate">
                             <product>{f5390db0-1f83-11e2-81c1-0800200c9a66}</product>
                      </include>
                      <include hint="list:IncludeTemplate">
                             <variant>{a34505d0-1f84-11e2-81c1-0800200c9a66}</variant>
                      </include>
                      <Boost>6.0</Boost>
                      <fieldCrawlers hint="raw:AddFieldCrawlers">
                             <fieldCrawler
```

```
type="scSearchContrib.Crawler.FieldCrawlers.LookupFieldCrawler,scSearchContrib.Crawler" fieldType="Droplink"
/>
                                <fieldCrawler
type="scSearchContrib.Crawler.FieldCrawlers.DateFieldCrawler,scSearchContrib.Crawler" fieldType="Datetime"
/>
{…}
                        </fieldCrawlers>
                        <fieldTypes hint="raw:AddFieldTypes">
                                <fieldType name="single-line text" storageType="NO" indexType="TOKENIZED"
vectorType="NO" boost="1f" />
                                <fieldType name="multi-line text" storageType="NO" indexType="TOKENIZED"
vectorType="NO" boost="1f" />
                                {…}
                        </fieldTypes>
                </products>
                {…}
        </locations>
</index>
```

The second index was related to the actual PDF repository. Notice the custom crawler needed for the files. We'll get into that next. This index will be processed directly through Sitecore's SearchManager.

```
<index id="Documents" type="Sitecore.Search.Index, Sitecore.Kernel">
        <param desc="name">$(id)</param>
        <param desc="folder">Documents</param>
        <Analyzer ref="search/analyzer" />
        <locations hint="list:AddCrawler">
                <filesystem type="Customer.Search.FileCrawler,Customer.Search">
                        <Root>/Documents</Root>
                        <Tags>documents</Tags>
                        <Boost>4.0</Boost>
                </filesystem>
        </locations>
</index>
```

The final index is much like the very first index, going after Sitecore content related to more corporate items like press releases and general page content. The file crawler is pretty straight forward in following the Sitecore document: Sitecore Search and Indexing. For obtaining the PDF content, iTextSharp was chosen to use versus the Adobe PDF IFilter or the PDFBox route. The code to retrieve the PDF content was quick to put together:

```
private string ReadPDFContent(FileInfo file)
{
var returnText = new StringBuilder();
PdfReader pdfReader = new PdfReader(file.FullName);

        //Loop through pages
        for (int page = 1; page <= pdfReader.NumberOfPages; page++)
        {
returnText.Append(PdfTextExtractor.GetTextFromPage(pdfReader, page));
                pdfReader.Close();
        }

        return returnText.ToString();
}
```

In our client's scenario, there was additional metadata for each PDF file also stored in Sitecore. Because of this, we incorporated some of that information into the Lucene document construction as additional content to be searched, and also stored the Sitecore Item ID in the Lucene document to refer back to that additional metadata that we did not include in the Lucene documents. This will later allow us to turn the search result hit for the file back into a SkinnyItem to include in the ADC results.

```
private void AddPDFContent(Document document, FileInfo file)
{
        var item = sitecoreDatabaseContext.SelectSingleItem("<insert your Sitecore fast query here>");

var description = string.Empty;
        var itemID = string.Empty;

        if (item != null)
        {
                description = item["Description"];
                itemID = item.ID.ToString();
        }

        document.Add(this.CreateDataField(BuiltinFields.Tags, itemID));
        document.Add(CreateTextField(BuiltinFields.Content, description));
        document.Add(CreateTextField(BuiltinFields.Content, ReadPDFContent(file)));
}
```

The file crawler will then execute when the appropriate Sitecore Search index is rebuilt and the PDF contents will be added to the index. Now, doing the search three times isn't the first solution that popped into our heads, but it seemed to fit the need for the time. Accomplishing this really meant to run our search criteria through both ADC and the SearchManager. Keeping this simple we will assume our search consists of a full text keyword search versus the field, multifield, and range search you can do with ADC. Our ADC searches look similar to this:

```
// Product search
using (var runner = new QueryRunner(ProductSearchIndex))
{
// Product result
        List<SkinnyItem> productResults = runner.GetItems(AssembleSearchParameters(criteria));
}
```

First the product catalogs are searched and the resulting SkinnyItems representing the Sitecore Items are returned. We then go ahead and search the documents index. If you have looked into Alex's code, this should look familiar:

```
// Document search
// Go after the index directly outside of ADC
var luceneIndex = SearchManager.GetIndex(DocumentSearchIndex);
using (var context = new IndexSearchContext(luceneIndex))
{
var analyzer = new StandardAnalyzer();
        var parser = new QueryParser("_content", analyzer);
        var query = parser.Parse(criteria.FullTextQuery.Trim());

        SearchHits searchhits;
        searchhits = context.Search(query);

        var end = 0;
        var start = 0;

        if (searchhits != null)
        {
                if (end == 0 || end > searchhits.Length) end = searchhits.Length;
                var resultCollection = searchhits.FetchResults(start, end);
                GetItemsFromSearchResult(resultCollection, documentResults, false);
        }
}
```

We created our own ‘GetItemsFromSearchResult’ in our own search logic. We needed to do this in order to turn our PDF search results into SkinnyItems by taking that Sitecore ID to eventually create that appropriate SkinnyItem.

```
// retrieve sitecore item for filename
var guid = result.Document.GetField(BuiltinFields.Tags).StringValue();

if (!string.IsNullOrEmpty(guid))
{
var itemId = new ID(guid);
        var db = Sitecore.Context.Database;
        var item = db.GetItem(itemId);

        // load to skinny item
        var itemInfo = new SkinnyItem(item.Uri);

        foreach (Field field in result.Document.GetFields())
        {
                itemInfo.Fields[field.Name()] = field.StringValue();
        }

        items.Add(itemInfo);
}
```

Once all three indexes are search, we are simply adding each collection to one large collection of SkinnyItems maintaining the desired ordering for the search results.

This was our approach to incorporate PDF content search with the Advanced Database Crawler.

Posted in

Development

Tags:

development .net

Powered by WeBlog

Topics

Business analysis
Communication
Development
IA and Usability
Our team
Project management
Quality assurance
The Latest

Authors

Aaron D
Anna J
Chris A
Dale M
Hank L
Jason R
Jenn H
Katie G
Mark S
Melissa K
Michael G
Tyler H
Wyatt W

Archive

- 2013
- 2012
  - July (4)
  - August (5)
  - September (3)
  - October (4)
  - November (4)

Syndication

Blog Entries

Tagcloud

testing development .net news software requirements business analysis web design ia usability ios agile cloud data culture software careers

Industries Served

Environmental Engineering

Field Service

Healthcare

Insurance

Manufacturing

Professional Services

Utilities

REDarrow
software architects

About / Our Work /

Careers / Contact

Phone: 414.289.7960

twitter

Solutions Provided

Billing and Point of Sale

Business Intelligence

ERP

Sales and CRM