

Faceted Searching for Apache Ofbiz E-Commerce

Anshuman S Manur
Senior Software Engineer
Stragure Software Technologies Pvt Ltd

E-COMMERCE NEEDS FACETED SEARCHING

Apache Ofbiz provides a great e-commerce solution on Java. It is stable and has a large active community for support. However, I believe that they omitted one fundamental necessity to the e-commerce package - faceted search.

The default Ofbiz install gives us an e-commerce package with keyword search. This proves to be quite inadequate for most implementations. Faceted searching is essential to an e-commerce website, and this has been effectively demonstrated by the e-commerce giant Amazon, whose faceted search makes it easier for users to navigate the huge number of items on sale at the site.

Faceting or faceted classification is the process of classifying an object in multiple ways, and thus enabling multiple hierarchies, instead of one pre-defined taxonomic order. A facet can be thought of as a set of characteristics that define an object. As an example, socks as an article of clothing, can be used in multiple ways - sports, dress, home etc., and thus can be listed under the respective categories. These categories are then grouped together by commonness of characteristics. Dress socks would be listed with tuxedos, while sports socks would be listed with gym clothing.

Apart from classification, faceting also allows very effective filtering. Numerical data about an object such as its cost or size can be used to apply filters.

Users who visit an e-store usually are of two kinds, those who know exactly what they want to buy, and those who don't. The first kind of users know how to find the products that they are interested in, and can effectively use keywords to describe them. Keyword search is perfect for them, as it is quicker of the two kinds of search. They key in a few words into a text box, and find what they're looking for in among the search results.

The second kind of users however, are either not sure if they want to buy anything at all and are just browsing, or have a faint idea of what they desire, and want to see what all the store has on offer. This group will effectively have a problem coming up with keywords to narrow down, as they don't grok their products yet and will be unfamiliar with its parlance. Search suggestions are of course one way to aid these kind of customers, but faceted search will be, I believe, more effective.

With faceting, a customer can effectively start from a high level in the hierarchy, an area he knows that he is interested in, and repeatedly drill down all the while narrowing down his list of products. And when he can combine that with filtering, one finds that he can very effectively find what he is looking for, or at the very least amuse himself by playing around with filters, spending more time at the site which increases his chances of considering buying an article that may catch his eye.

FACETED SEARCH VS. KEYWORD SEARCH

Faceted searching differs fundamentally from keyword searching, and a brief discussion on these aspects of difference are:

- Different ways of organising information

At the heart, the two differ in the way the engines index and organise the information on the site. Keyword search builds indexes based on pages, all the words on a particular page are indexed as belonging to that page. Thus the page is the basic unit of information organisation. The page can be HTML, PDF, JPEG, etc. and will be indexed so long as the engine can read that information.

Faceted search involves breaking down all searchable elements into individual units with properties (In Apache Solr, this individual unit is called a document). In an e-commerce site, this individual unit would be an item for sale, its properties being its size, manufacturer etc. This makes it easier to group items according to common properties, which is the fundamental idea behind faceting. This also focuses on relevance of information - typically you wouldn't want the irrelevant information, such as a disclaimer notice, to be grouped along with an item for sale. Yet this is exactly what happens in keyword search. Even assuming that, a keyword search engine gives enough fine grained control against this, the document definition process in faceted search, I believe, gives you the power to define exactly what appears in your search results.

- Search Results

Faceted search engines give out results in a format that can be used to indicate each individual document. Apache Solr, for example, gives out results in XML or JSON which is usually composed of a list of documents. Thus search results are independant of how one would want to display them to the user. Keyword search however usually gives out results as a list of URLs where the keyword occurs, and thus, can be displayed only in that manner.

In reality however, faceted search is not the panacea for an e-commerce website - one cannot do without keyword search. When a store has an unreasonably large number of items for sale (as does amazon.com), one cannot expect a customer to navigate through umpteen levels of hierarchy in order to find his product, and along with those who wouldn't prefer faceting, keyword search cannot be avoided. Thus a combination of both is ideal, giving the user complete freedom to choose his method of search. Apache Solr supports both keyword and facted search, and comes with a host of other useful features, and would make a good candidate for a integration.

THE SOLUTION

Apache Solr is an open source enterprise search platform, built on top of the Lucene Java search library. It was originally developed as an in-house CNET project to satisfy their search needs, but was later donated to the Apache Software Foundation where it accumulated a vast community of users. It has a large set of features including but not limited to:

- Advanced full set search
- Faceted search and filtering
- HTML administrative interfaces
- XML based configuration
- Extensible plugin architecture
- Highly scalable distributed search capabilities
- Rich document parsing (PDF, MS Word)
- Server statistics over JMX for monitoring

Apache Solr is written in Java and runs on a servlet container such as Apache Tomcat. It communicates using well-known standards such as HTTP, XML, and JSON, thus making integration easy. Any external entity, regardless of what technology it is built on, needs only HTTP to make calls to the Solr search server. Search results can be sought in XML or JSON, and there is also a Java wrapper over the results set available which is extremely convenient. Indexing using Solr involves XML based configuration, and the engine is capable of building its index from CVS, XML, binary or from a relational database.

Among Solr's more interesting features (apart from faceting) are:

- Caching

Search results are cached, so as to make subsequent searches faster.

- Autowarming

Solr will make dummy search queries before actual ones in order to warm the server up, and populate the cache.

- Autosuggestion

Powerful feature that can be used to provide suggestions to a user as he types a search keyword.

- Search results scoring

Search result scores are numeric values that rate the relevancy of a result item to the

searched for term, thus deciding the ordering of search results based on this value. Solr enables us to define complex function in order to influence these scores according to our needs, overruling the default score of a result in certain cases.

- Search results highlighting

Highlighting the searched-for keywords in the search results, to inform the user where exactly his search terms have been found.

APPLYING THE SOLUTION

Since Ofbiz is itself a J2EE application that runs on a Servlet container, Solr integration is as simple as adding another webapp to the many that come with Ofbiz, noting the URL and making HTTP calls to use it.

Configuring the Solr engine to suit one's needs is an intensive process, due to the sheer number of options available. XML based configuration makes specifying options uniform and error-free. Sample configuration files, the wiki, the mailing list and other documentation all help immensely.

Interpreting search results is rather easy, since Ofbiz runs on the Java platform. Solrj is a java client that exposes an API to make calls to the server, and interpret search results, saving developers the trials of URL construction and XML parsing. Configuration for using this involves dropping a JAR in the classpath and instructing the Solr engine to use the API to return results.

Here's a step-by-step instruction guide to the integration:

Step 1: Setting up a new Ofbiz component

- Create a new folder (lets call it 'solr') under <OFBIZ_HOME>/hot-deploy/
Apache Ofbiz is an ERP engine, but it is also a framework where one can write one's own components. What we are doing here is deploying an external component to run on the Ofbiz framework. The hot-deploy folder is reserved for user's components. All the components under hot-deploy will be automatically loaded onto the Ofbiz framework, but only after all the other Ofbiz components.

- Create a ofbiz-component.xml file

In order for Ofbiz to load a component under hot-deploy, one needs to include an XML file describing the component under the component's root. This file, called 'ofbiz-component.xml' details the paths where the component stores its resources and JARs, where its entity and service definitions are to be found, how many webapps the component exposes and so on. It is meant to contain all the meta-data about a component that the framework will need to know.

A sample component configuration file is give in the text box below.

```

<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="solr"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org/dtds/ofbiz-
z-component.xsd">

  <resource-loader name="main" type="component"/>
  <classpath type="jar" location="webapp/solr/WEB-INF/lib/*"/>

  <webapp name="solr"
    title="Solr"
    server="default-server"
    location="webapp/solr"
    mount-point="/solr"
    app-bar-display="false"/>
</ofbiz-component>

```

Step 2: Adding the webapp

- Create the folders <OFBIZ_HOME>/hot-deploy/solr/webapp and <OFBIZ_HOME>/hot-deploy/solr/webapp/solr

Our Apache Solr component will expose just one web interface called 'solr'.

- Create the folder <OFBIZ_HOME>/hot-deploy/solr/webapp/solr/WEB-INF and place the 'web.xml' file

The WEB-INF folder and the web.xml file are part of the J2EE specifications for describing webapps written using Java Servlet Technology. More specifically, the web.xml file is called a Deployment Descriptor (DD), a configuration file that describes an artifact to be deployed on a container. Here, we use it to describe Apache Solr's filters and servlets, and define various parameters for their operation.

For a quick setup, using the web.xml from the packaged Solr WAR file is recommended.

- Copy the JSP, HTML and associated files

In the WAR file that is packaged within the Apache Solr download, copy the folder named "admin" from the root, and the file named "index.jsp" and place them both in <OFBIZ_HOME>/hot-deploy/solr/webapp/solr/

Step 3: Adding the necessary JARs

- Create the folder <OFBIZ_HOME>/hot-deploy/solr/webapp/solr/WEB-INF/lib and populate it with all necessary JARs

The JARs that need to be included for a working Solr instance can be found in the WAR file that is included along with the standard Apache Solr 1.4 download. On unzipping the WAR to a location <SOLR_WAR>, all the JARs can be found in the <SOLR_WAR>/WEB-INF/lib directory. Copy all of these JARs to our Ofbiz

directory mentioned above.

- Add an entry into the ofbiz-component.xml file to pick up these JARs

As mentioned earlier, the ofbiz-component.xml completely describes a component, and for the Ofbiz framework to pick up all these JARs and put them on the classpath so that our Solr webapp can use them, we need to include a line in there.

```
<classpath type="jar" location="webapp/solr/WEB-INF/lib/*"/>
```

Step 4: Creating the Solr home

- Create folders 'bin', 'conf', 'data', 'lib'

Apache Solr needs to be told where it can find its configuration files, plugins and libraries, and where it can store its index. Apache Solr calls this place its home. A single instance of Apache Solr can have only one home. Solr's home is made up of 4 folders:

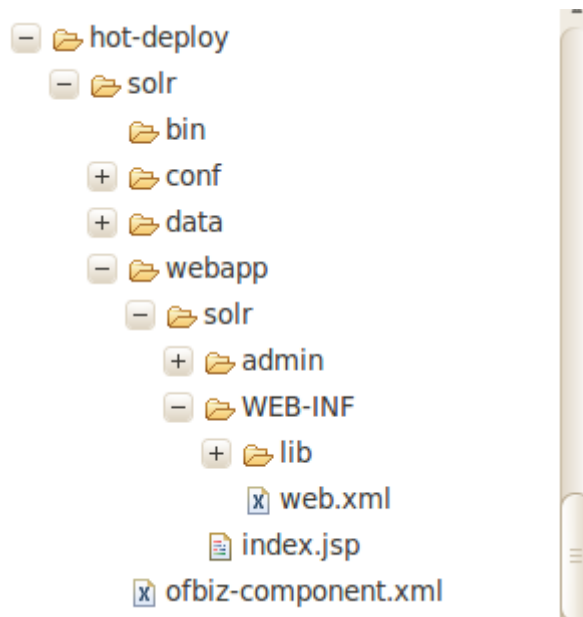
bin - for plugins

conf - configuration files

data - index

lib - additional JARs

For a quick setup, it is easy to copy the Solr home that is included as part of the examples along with the default Solr download, including all the configuration files, as is. Further refinement can be done as required. After all of this setup, the hot-deploy directory structure should look like this:



- Include the solrconfig.xml file

The solrconfig.xml is the file that contains most of the parameters for configuring Solr.

- Inform Tomcat of Solr's home

Finally, Tomcat has to be informed where Solr's home is so that Apache Solr itself can look up this information and find it. This can be done in multiple ways. A simple way to do it is to pass it as a system property to the JVM when running Ofbiz using the -D option. While running ofbiz.jar on the JRE, use the following command:

```
java -Dsolr.solr.home=<PATH_TO_SOLR_HOME> -jar ofbiz.jar
```

Another way to do this would be to set it as a JNDI lookup value in the web.xml DD.

Step 5: Testing the integration

- Startup Ofbiz

Do not forget to set Solr Home during startup.

- Hit the URL: `http://<host>:<port>/solr`

If Solr is successfully integrated, you should see the following default welcome page.



Contact the author at anshuman_manur@stragure.com