JavaDemoApplication

Integrating Nutch search functionality into a Java application

⚠️ :MESSAGE: ⚠️ This page is **extremely** out of date. It is not useful for modern versions of Nutch.

This example is the fruit of much searching of the nutch users mailing list in order to get a working application that used the Nutch APIs. I couldn't find all that was needed to provide a quick-start in one place, so this document was born...

Using Nutch within an application is actually very simple; the requirements are merely the existence of a previously created crawl index, a couple of settings in a configuration file, and a handful of jars in your classpath. Nothing else is needed from the Nutch release that you can download.

This example assumes that an index has been created in the directory /home/nutch-java-demo/crawl-dir and a copy of the 'plugins' folder from the nutch distribution is in the directory /home/nutch-java-demo/plugins. This directory tree is completely external to the deployment of the java application.

Configuration

For the search to work, some appropriate settings need to be in a file called nutch-site.xml. If you have read the first part of this document, this file will be familiar to you. While you could use the same version of that file as before, there is no need to do so, as only two properties are required within it:

1) plugin.folders must be a fully qualified path, such as:

```
<property>
  <name>plugin.folders</name>
  <value>/home/nutch-java-demo/plugins</value>
  <description />
</property>
```

This should point to a folder containing all the Nutch plugins. This can be placed anywhere within the filesystem and has no dependency on any other files distributed with Nutch.

2) searcher.dir must be a fully qualified path to the crawl directory you want to use

```
<property>
  <name>searcher.dir</name>
  <value>/home/nutch-java-demo/crawl_dir</value>
  <description />
</property>
```

Place this copy of nutch-site.xml and a copy of common-terms.utf8 (from the conf directory in the Nutch distribution) in the WEB-INF/classes directory of the web application that you're deploying. For a standalone application, the mentioned files have to be available in the CLASSPATH. In some cases, you might want to have extra-flexibility by using runtime configuration parameters. This can be achieved using variable substitution. For example, the nutch-site.xml might look like this:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
        <property>
                <name>plugin.folders</name>
                <value>${nutch.site.plugin.folders}
                </value>
                <description />
        </property>

        <property>
                <name>searcher.dir</name>
                <value>${nutch.site.searcher.dir}</value>
                <description />
        </property>
</configuration>
```

and run the java application using the appropriate parameters:

```
-Dnutch.site.plugin.folders="c:\tools\crawlers\apache-nutch-1.1-bin\plugins"
-Dnutch.site.searcher.dir="c:\tools\crawlers\apache-nutch-1.1-bin\crawl"
```

CLASSPATH Configuration

You also need to make sure that the following jars are placed in WEB-INF/lib (this assumes usage of Nutch 0.9):

```
commons-cli-2.0-SNAPSHOT.jar
hadoop-0.12.2-core.jar
lucene-core-2.2.0.jar
lucene-misc-2.2.0.jar
nutch-0.9.jar
```

For a standalone application, one might want to use Apache maven (this configuration assumes Nutch 1.1). At the moment of writing this note, Nutch does not publish its artifacts to maven. However we (members of community) hope that maven support will be added soon. In the meantime, just install the nutch-1.1.jar to your maven repository. Here is a snippet that will manage the dependencies that you need to run this example (note that the 1.1-XXX version of Nutch marks the fact that the artifact cannot be found in any public repository yet):

```
<dependency>
        <groupId>org.apache.nutch</groupId>
        <artifactId>nutch</artifactId>
        <version>1.1-XXX</version>
</dependency>

<dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-core</artifactId>
        <version>0.20.2</version>
</dependency>

<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-core</artifactId>
        <version>3.0.1</version>
        <scope>runtime</scope>
</dependency>

<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-misc</artifactId>
        <version>3.0.1</version>
        <scope>runtime</scope>
</dependency>

<dependency>
        <groupId>commons-lang</groupId>
        <artifactId>commons-lang</artifactId>
        <version>2.1</version>
        <scope>runtime</scope>
</dependency>
```

Sample code

With that, all is ready and we can now write some simple code to search. A quick example in Java to search the crawl index and return the number of hits found is:

```
// necessary imports
import org.apache.hadoop.conf.Configuration;
import org.apache.nutch.searcher.Hit;
import org.apache.nutch.searcher.HitDetails;
import org.apache.nutch.searcher.Hits;
import org.apache.nutch.searcher.NutchBean;
import org.apache.nutch.searcher.Query;
import org.apache.nutch.util.NutchConfiguration;
import java.util.Date;

public class Search {
        public static void main(String[] args) {
                try {
                        // define a keyword for the search
                        String nutchSearchString = "smart";

                        // configure nutch
                        Configuration nutchConf = NutchConfiguration.create();
                        NutchBean nutchBean = new NutchBean(nutchConf);
                        // build the query
                        Query nutchQuery = Query.parse(nutchSearchString, nutchConf);
                        // optionally specify the maximum number of hits (default is 10)
                        // nutchQuery.getParams().setNumHits(100);
                        // nutchQuery.getParams().setMaxHitsPerDup(100);
                        Hits nutchHits = nutchBean.search(nutchQuery);

                        // display the number of hits
                        System.out.println("Found " + nutchHits.getLength() + " hits.\n");

                        // get the details about each hit (includes title, URL, a summary
                        // and the date when this was fetched)
                        for (int i = 0; i < nutchHits.getLength(); i++) {
                                Hit hit = nutchHits.getHit(i);
                                HitDetails details = nutchBean.getDetails(hit);
                                String title = details.getValue("title");
                                String url = details.getValue("url");
                                String summary = nutchBean.getSummary(details, nutchQuery)
                                                .toString();
                                System.out.println("Title is: " + title);
                                System.out.println("(" + url + ")");
                                Date date = new Date(nutchBean.getFetchDate(details));
                                System.out.println("Date Fetched: " + date);
                                System.out.println(summary + "\n");
                                System.out.println("----------------------------------------");
                        }

                        // as usually, don't forget to close the resources
                        nutchBean.close();
                } catch (Throwable e) {
                        e.printStackTrace();
                }
        }
}
```

Extra information about developing a standalone application that does the search can be obtained by inspecting the main method in org.apache.nutch.searcher.NutchBean.

Authors

Chaz Hickman (Jan 2008)

Cristi Vulpe (Aug 2010)