WritingPluginExample-1.2

Most of the text and original code from this page are originally from WritingPluginExample. It's been updated to work with the trunk as of revision 506842, and to add unit testing.

The Example

Consider this as a plugin example: We want to be able to recommend specific web pages for given search terms. For this example we'll assume we're indexing this site. As you may have noticed, there are a number of pages that talk about plugins. What we want to do is have it so that if someone searches for the term "plugins" we recommend that they start at the PluginCentral page, but we also want to return all the normal hits in the expected ranking. We'll seperate the search results page into a section of recommendations and then a section with the normal search results.

You go through your site and add meta-tags to pages that list what terms they should be recommended for. The tags look something like this:

```
<meta name="recommended" content="plugins" />
```

In order to do this we need to write a plugin that extends 3 different extension points. We need to extend the HTMLParser in order to get the recommended terms out of the meta tags. The IndexingFilter will need to be extended to add a recommended field to the index. The QueryFilter needs to be extended to add the ability to search againsed the new field in the index.

Setup

Start by downloading the Nutch source code. Once you've got that make sure it compiles as is before you make any changes. You should be able to get it to compile by running ant from the directory you downloaded the source to. If you have trouble you can write to one of the Mailing Lists.

Use the source code for the plugins distrubuted with Nutch as a reference. They're in [!YourCheckoutDir]/src/plugin.

For the example we're going to assume that this plugin is something we want to contribute back to the Nutch community, so we're going to use the directory /package structure of "org/apache/nutch". If you're writing a plugin solely for the use of your organization you'd want to replace that with something like "org /my_organization/nutch".

Required Files

You're going to need to create a directory inside of the plugin directory with the name of your plugin ('recommended' in this case) and inside that directory you need the following:

- A plugin.xml file that tells nutch about your plugin.
- A build.xml file that tells ant how to build your plugin.

- The source code of your plugin in the directory structure recommended/src/java/org/apache/nutch/parse/recommended/[Source_Here].

Plugin.xml

Your plugin.xml file should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
   id="recommended"
   name="Recommended Parser/Filter"
   version="0.0.1"
   provider-name="nutch.org">

   <runtime>
      <!-- As defined in build.xml this plugin will end up bundled as recommended.jar -->
      <library name="recommended.jar">
         <export name="*"/>
      </library>
   </runtime>

   <!-- The RecommendedParser extends the HtmlParseFilter to grab the contents of
        any recommended meta tags -->
   <extension id="org.apache.nutch.parse.recommended.recommendedfilter"
              name="Recommended Parser"
              point="org.apache.nutch.parse.HtmlParseFilter">
      <implementation id="RecommendedParser"
                      class="org.apache.nutch.parse.recommended.RecommendedParser"/>
   </extension>

   <!-- TheRecommendedIndexer extends the IndexingFilter in order to add the contents
        of the recommended meta tags (as found by the RecommendedParser) to the lucene
        index. -->
   <extension id="org.apache.nutch.parse.recommended.recommendedindexer"
              name="Recommended identifier filter"
              point="org.apache.nutch.indexer.IndexingFilter">
      <implementation id="RecommendedIndexer"
                      class="org.apache.nutch.parse.recommended.RecommendedIndexer"/>
   </extension>

   <!-- The RecommendedQueryFilter gets called when you perform a search. It runs a
        search for the user's query against the recommended fields.  In order to get
        add this to the list of filters that gets run by default, you have to use
        "fields=DEFAULT". -->
   <extension id="org.apache.nutch.parse.recommended.recommendedSearcher"
              name="Recommended Search Query Filter"
              point="org.apache.nutch.searcher.QueryFilter">
      <implementation id="RecommendedQueryFilter"
                      class="org.apache.nutch.parse.recommended.RecommendedQueryFilter">
        <parameter name="fields" value="recommended"/>
        </implementation>
   </extension>

</plugin>
```

Build.xml

In its simplest form:

```
<?xml version="1.0"?>

<project name="recommended" default="jar">

  <import file="../build-plugin.xml"/>

</project>
```

For Nutch-1.0 write the following:

```
<?xml version="1.0"?>

<project name="recommended" default="jar-core">

  <import file="../build-plugin.xml"/>

 <!-- Build compilation dependencies -->
 <target name="deps-jar">
   <ant target="jar" inheritall="false" dir="../lib-xml"/>
 </target>

  <!-- Add compilation dependencies to classpath -->
 <path id="plugin.deps">
   <fileset dir="${nutch.root}/build">
     <include name="**/lib-xml/*.jar" />
   </fileset>
 </path>

  <!-- Deploy Unit test dependencies -->
 <target name="deps-test">
   <ant target="deploy" inheritall="false" dir="../lib-xml"/>
   <ant target="deploy" inheritall="false" dir="../nutch-extensionpoints"/>
   <ant target="deploy" inheritall="false" dir="../protocol-file"/>
 </target>


  <!-- for junit test -->
  <mkdir dir="${build.test}/data"/>
  <copy file="data/recommended.html" todir="${build.test}/data"/>
</project>
```

Save this file in directory [!YourCheckoutDir]/src/plugin/recommended

The HTML Parser Extension

NOTE: Nutch-1.0 users make sure that you save all your java files in this directory C:\nutch-1.0 \src\plugin\recommended\src\java\org\apache\nutch\parse\recommended

This is the source code for the HTML Parser extension. It tries to grab the contents of the recommended meta tag and add them to the document being parsed. On the directory , create a file called RecommendedParser.java and add this as the contents:

```
package org.apache.nutch.parse.recommended;

// JDK imports
import java.util.Enumeration;
import java.util.Properties;
import java.util.logging.Logger;

// Nutch imports
import org.apache.hadoop.conf.Configuration;
import org.apache.nutch.parse.HTMLMetaTags;
import org.apache.nutch.parse.Parse;
import org.apache.nutch.parse.HtmlParseFilter;
import org.apache.nutch.parse.ParseResult;
import org.apache.nutch.protocol.Content;

// Commons imports
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

// W3C imports
import org.w3c.dom.DocumentFragment;

public class RecommendedParser implements HtmlParseFilter {

  private static final Log LOG = LogFactory.getLog(RecommendedParser.class.getName());

  private Configuration conf;

  /** The Recommended meta data attribute name */
  public static final String META_RECOMMENDED_NAME="Recommended";

  /**
   * Scan the HTML document looking for a recommended meta tag.
   */
  public ParseResult filter(Content content, ParseResult parseResult,
      HTMLMetaTags metaTags, DocumentFragment doc) {

    String recommendation = null;

    Properties generalMetaTags = metaTags.getGeneralTags();

    for (Enumeration tagNames = generalMetaTags.propertyNames(); tagNames.hasMoreElements(); ) {
        if (tagNames.nextElement().equals("recommended")) {
           recommendation = generalMetaTags.getProperty("recommended");
           LOG.info("Found a Recommendation for " + recommendation);
        }
    }

    Parse parse = parseResult.get(content.getUrl());

    if (recommendation == null) {
        LOG.info("No Recommendation");
    } else {
        LOG.info("Adding Recommendation for " + recommendation);
        parse.getData().getContentMeta().set(META_RECOMMENDED_NAME, recommendation);
    }

    return parseResult;
  }


  public void setConf(Configuration conf) {
    this.conf = conf;
  }

  public Configuration getConf() {
    return this.conf;
  }

}
```

The Indexer Extension

The following is the code for the Indexing Filter extension. If the document being indexed had a recommended meta tag this extension adds a lucene text field to the index called "recommended" with the content of that meta tag. Create a file called RecommendedIndexer.java in the source code directory:

```
package org.apache.nutch.parse.recommended;

// JDK import
import java.util.logging.Logger;

// Commons imports
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;


// Nutch imports
import org.apache.nutch.util.LogUtil;
import org.apache.nutch.fetcher.FetcherOutput;
import org.apache.nutch.indexer.IndexingFilter;
import org.apache.nutch.indexer.IndexingException;
import org.apache.nutch.indexer.NutchDocument;
import org.apache.nutch.parse.Parse;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.Text;
import org.apache.nutch.crawl.CrawlDatum;
import org.apache.nutch.crawl.Inlinks;

// Lucene imports
import org.apache.nutch.indexer.lucene.LuceneWriter;
import org.apache.nutch.indexer.lucene.LuceneWriter.INDEX;
import org.apache.nutch.indexer.lucene.LuceneWriter.STORE;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.Document;

public class RecommendedIndexer implements IndexingFilter {

  public static final Log LOG = LogFactory.getLog(RecommendedIndexer.class.getName());

  private Configuration conf;

  public RecommendedIndexer() {
  }

  public NutchDocument filter(NutchDocument doc, Parse parse, Text url,
      CrawlDatum datum, Inlinks inlinks) throws IndexingException {

    String recommendation = parse.getData().getMeta("Recommended");

    if (recommendation != null) {
        //Field recommendedField =
        //   new Field("recommended", recommendation,
        //        Field.Store.YES, Field.Index.UN_TOKENIZED);
        //recommendedField.setBoost(5.0f);
        doc.add("recommended", recommendation);
        LOG.info("Added " + recommendation + " to the recommended Field");
    }

    return doc;

  }

  public void setConf(Configuration conf) {
    this.conf = conf;
  }

  public Configuration getConf() {
    return this.conf;
  }
```

```
  public void addIndexBackendOptions(Configuration conf)
  {
    LuceneWriter.addFieldOptions(
        "recommended", STORE.YES, INDEX.UNTOKENIZED, conf);
  }

}
```

Note that the field is UN_TOKENIZED because we don't want the recommended tag to be cut up by a tokenizer. Change to TOKENIZED if you want to be able to search on parts of the tag, for example to put multiple recommended terms in one tag.

The QueryFilter

The QueryFilter gets called when the user does a search. We're bumping up the boost for the recommended field in order to increase its influence on the search results.

```
package org.apache.nutch.parse.recommended;

import org.apache.nutch.searcher.FieldQueryFilter;

import java.util.logging.Logger;

// Commons imports
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;


public class RecommendedQueryFilter extends FieldQueryFilter {
    private static final Log LOG = LogFactory.getLog(RecommendedParser.class.getName());

    public RecommendedQueryFilter() {
        super("recommended", 5f);
        LOG.info("Added a recommended query");
    }

}
```

Compiling the plugin

For ant installation in Windows, refer this - ant

In order to build the plugin - or Nutch itself - you'll need ant. If you're using MacOs you can easily get it via fink. Let's get junit while we're at it.

```
fink install ant ant-junit junit
```

In order to build it, change to your plugin's directory where you saved the build.xml file (probably [!YourCheckoutDir]/src/plugin/recommended), and simply type

```
ant
```

Hopefully you'll get a long string of text, followed by a message telling you of a successful build.

Getting Ant to Compile Your Plugin

In order for ant to compile and deploy your plugin on the global build you need to edit the src/plugin/build.xml file (NOT the build.xml in the root of your checkout directory). You'll see a number of lines that look like

```
  <ant dir="[plugin-name]" target="deploy" />
```

Edit this block to add a line for your plugin before the </target> tag.

```
  <ant dir="recommended" target="deploy" />
```

Running 'ant' in the root of your checkout directory should get everything compiled and jared up. The next time you run a crawl your parser and index filter should get used.

You'll need to run 'ant war' to compile a new ROOT.war file. Once you've deployed that, your query filter should get used when searches are performed.

Unit testing

We'll need to create two files for unit testing: a page we'll do the testing against, and a class to do the testing with. Again, let's assume your plugin directory is [!YourCheckoutDir]/src/plugin and that your test plugin is under that directory. Create directory recommended/data, and under it make a new file called recommended.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">

<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>recommended</title>
    <meta name="generator" content="TextMate http://macromates.com/">
    <meta name="author" content="Ricardo J. Méndez">
    <meta name="recommended" content="recommended-content"/>
    <!-- Date: 2007-02-12 -->
</head>
<body>
    Recommended meta tag test.
</body>
</html>
```

This file contains the meta tag we're currently parsing for, with the value **recommended-content**. After that gratuitous bit of free publicity for my current favorite editor, let's move on to the testing class.

Create a new tree structure, this time for the test code, for example recommended/src/test/org/apache/nutch/parse/recommended/[Test_Source_Here]. There you'll create a file called [TestRecommendedParser].java.

```
package org.apache.nutch.parse.recommended;

import org.apache.nutch.metadata.Metadata;
import org.apache.nutch.parse.Parse;
import org.apache.nutch.parse.ParseResult;
import org.apache.nutch.parse.ParseUtil;
import org.apache.nutch.protocol.Content;
import org.apache.hadoop.conf.Configuration;
import org.apache.nutch.util.NutchConfiguration;

import java.util.Properties;
import java.io.*;
import java.net.URL;

import junit.framework.TestCase;

/*
 * Loads test page recommended.html and verifies that the recommended
 * meta tag has recommended-content as its value.
 *
 */
public class TestRecommendedParser extends TestCase {

  private static final File testDir =
    //new File(System.getProperty("test.data"));
    new File("/work/nutch-1.2/src/plugin/recommended/data");

  public void testPages() throws Exception {
    pageTest(new File(testDir, "recommended.html"), "http://foo.com/",
             "recommended-content");

  }


  public void pageTest(File file, String url, String recommendation)
    throws Exception {

    String contentType = "text/html";
    InputStream in = new FileInputStream(file);
    ByteArrayOutputStream out = new ByteArrayOutputStream((int)file.length());
    byte[] buffer = new byte[1024];
    int i;
    while ((i = in.read(buffer)) != -1) {
      out.write(buffer, 0, i);
    }
    in.close();
    byte[] bytes = out.toByteArray();
    Configuration conf = NutchConfiguration.create();

    Content content =
      new Content(url, url, bytes, contentType, new Metadata(), conf);
    ParseResult parseResult = new ParseUtil(conf).parseByExtensionId("parse-html",content);
    Metadata metadata = parseResult.get(url).getData().getContentMeta();
    assertEquals(recommendation, metadata.get("Recommended"));
    assertTrue("somesillycontent" != metadata.get("Recommended"));
  }
}
```

As you can see, this code first parses the document, looks for the **Recommended** item in the object contentMeta - which we saved on RecommendedParser - and verifies that it's set to value **recommended-content**.

Now add some lines to the build.xml file located in [!YourCheckoutDir]/src/plugin/recommended directory, so that at a minimum its contents are:

```
<?xml version="1.0"?>

<project name="recommended" default="jar">

  <import file="../build-plugin.xml"/>

  <!-- for junit test -->
  <mkdir dir="${build.test}/data"/>
  <copy file="data/recommended.html" todir="${build.test}/data"/>

</project>
```

These lines will copy the test data to the proper directory for testing.

To run the test case, simply move back to your src plugin's root directory and execute.

```
ant test
```

To debug this code on eclipse it's essential that you make a recommended directory under the main plugins of your nutch installation and put there the plugin.xml file (and for run the .jar too). Because plugins may use Nutch core classes these must have been compiled before unit tests are run.

Getting Nutch to Use Your Plugin

In order to get Nutch to use your plugin, you need to edit your conf/nutch-site.xml file and add in a block like this:

```
<property>
  <name>plugin.includes</name>
  <value>nutch-extensionpoints|protocol-http|urlfilter-regex|parse-(text|html)|index-basic|query-
(basic|site|url)</value>
  <description>Regular expression naming plugin id names to
  include.  Any plugin not matching this expression is excluded.
  In any case you need at least include the nutch-extensionpoints plugin. By
  default Nutch includes crawling just HTML and plain text via HTTP,
  and basic indexing and search plugins.
  </description>
</property>
```

You'll want to edit the regular expression so that it includes the id of your plugin.

```
  <value>recommended|protocol-http|urlfilter-regex|parse-(text|html|js)|index-basic|query-(basic|site|url)
|summary-basic|scoring-opic|urlnormalizer-(pass|regex|basic)</value>
```

<<< See also: HowToContribute

<<< PluginCentral