WritingPluginExample

Writing a Nutch Plugin

Introduction

This plugin example focuses on the urlmeta plugin which which is packaged with Nutch-1.3. It aims to provide a comprehensive introduction to plugin development for Apache Nutch.

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 crawling this site with Nutch and indexing it with Apache Solr. As you may have noticed, there are a number of pages that talk about plugins. If someone searches for the term "plugin", we want the first hit returned to be the Nutch PluginCentral page, however we also want to return all the normal hits in the expected ranking.

This is where we find a use for the urlmeta plugin. It is designed to enhance the original NUTCH-655 patch, by doing two things:

1. Meta Tags that are supplied with your Crawl URLs, during injection, will be propagated throughout the out-links of those Crawl URLs 2. When you index your URLs, the meta tags that you specified with your URLs will be indexed alongside those URLs--and can be directly queried, assuming you have done everything else correctly.

The first step here is to go through our 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 2 different extension points. Firstly we need to extend the IndexingFilter by creating an URLMetaIndexingFilter as we need to add any additional meta-tags to the index. Secondly we need to extend the ScoringFilter by creating an URLMetaScoringFilter. The idea here is that this will take the metatags we have listed in our "urlmeta.tags" property, and look for them inside the ! parseData object. This allows us to match recommended terms out of the meta tags. If there was no property within nutch-default.xml for us to specify these terms we would be required to add the new field to our Nutch schema.xml which would similarly add the ability to search against the new field in the index.

Setup

Start by downloading the Nutch-1.3 source code. Once you've got that make sure it compiles as is before you decide to 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 distributed with Nutch as a reference. They're in $NUTCH_HOME/src/plugin. In particular we focus on the urlmeta plugin within this example.

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 organisation you'd want to replace that with something like "org /my_organization/nutch". If you look at the structure of the urlmeta plugin you will see it follows this convention e.g. org.apache.nutch.indexer and org. apache.nutch.scoring

Required Files

This section covers the integral components required to develop and use a plugin. As you can see inside the $NUTCH_HOME/src/plugin directory, the plugin folder urlmeta contains the following:

- A plugin.xml file that tells Nutch about your plugin.
- A build.xml file that tells ant how to build your plugin.
- A ivy.xml file containing either the description of the dependencies of a module, its published artifacts and its configurations or else the location of another file which does specify this information.

- A /src directory containing the source code of our plugin with the directory structure shown in the hierarchical view below.

```
plugin.xml
build.xml
ivy.xml
src
  java
    org
      apache
        nutch
          indexer
            urlmeta <<< Source Folder
              package.html
              URLMetaIndexingFilter.java
          scoring
            urlmeta <<< Source Folder
              package.html
              URLMetaScoringFilter.java
```

plugin.xml

Your plugin.xml file should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
   id="urlmeta"
   name="URL Meta Indexing Filter"
   version="1.0.0"
   provider-name="sgonyea">


   <runtime>
      <library name="urlmeta.jar">
         <export name="*"/>
      </library>
   </runtime>

   <requires>
      <import plugin="nutch-extensionpoints"/>
   </requires>

   <extension      id="org.apache.nutch.indexer.urlmeta"
                   name="URL Meta Indexing Filter"
                   point="org.apache.nutch.indexer.IndexingFilter">
   <implementation id="indexer-urlmeta"
                   class="org.apache.nutch.indexer.urlmeta.URLMetaIndexingFilter"/>
   </extension>
   <extension      id="org.apache.nutch.scoring.urlmeta"
                   name="URL Meta Scoring Filter"
                   point="org.apache.nutch.scoring.ScoringFilter">
   <implementation id="scoring-urlmeta"
                   class="org.apache.nutch.scoring.urlmeta.URLMetaScoringFilter" />
   </extension>
</plugin>
```

build.xml

In its simplest form:

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

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

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

</project>
```

ivy.xml

This file is used to describe the dependencies of the plugin on other libraries.

```
<ivy-module version="1.0">
  <info organisation="org.apache.nutch" module="${ant.project.name}">
    <license name="Apache 2.0"/>
    <ivyauthor name="Apache Nutch Team" url="http://nutch.apache.org"/>
    <description>
        Apache Nutch
    </description>
  </info>

  <configurations>
    <include file="${nutch.root}/ivy/ivy-configurations.xml"/>
  </configurations>

  <publications>
    <!--get the artifact from our module name-->
    <artifact conf="master"/>
  </publications>

  <dependencies>
  </dependencies>

</ivy-module>
```

The Indexer Extension

This is the source code for the IndexingFilter extension. Meta Tags that are included in your Crawl URLs, during injection, will be propagated throughout the outlinks of those Crawl URLs. This means that when you index your URLs, the meta tags that you specified with your URLs will be indexed alongside those URLs--and can be directly queried.

```
package org.apache.nutch.indexer.urlmeta;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.Text;
import org.apache.nutch.crawl.CrawlDatum;
import org.apache.nutch.crawl.Inlinks;
import org.apache.nutch.indexer.IndexingException;
import org.apache.nutch.indexer.IndexingFilter;
import org.apache.nutch.indexer.NutchDocument;
import org.apache.nutch.parse.Parse;

public class URLMetaIndexingFilter implements IndexingFilter {

        private static final Log LOG = LogFactory
                        .getLog(URLMetaIndexingFilter.class);
        private static final String CONF_PROPERTY = "urlmeta.tags";
        private static String[] urlMetaTags;
        private Configuration conf;

        /**
         * This will take the metatags that you have listed in your "urlmeta.tags"
         * property, and looks for them inside the CrawlDatum object. If they exist,
         * this will add it as an attribute inside the NutchDocument.
         *
         * @see IndexingFilter#filter
         */
        public NutchDocument filter(NutchDocument doc, Parse parse, Text url,
                        CrawlDatum datum, Inlinks inlinks) throws IndexingException {
                if (conf != null)
                        this.setConf(conf);

                if (urlMetaTags == null || doc == null)
                        return doc;

                for (String metatag : urlMetaTags) {
                        Text metadata = (Text) datum.getMetaData().get(new Text(metatag));

                        if (metadata != null)
                                doc.add(metatag, metadata.toString());
                }

                return doc;
        }

        /** Boilerplate */
        public Configuration getConf() {
                return conf;
        }

        /**
         * handles conf assignment and pulls the value assignment from the
         * "urlmeta.tags" property
         */
        public void setConf(Configuration conf) {
                this.conf = conf;

                if (conf == null)
                        return;

                urlMetaTags = conf.getStrings(CONF_PROPERTY);
        }
}
```

The Scoring Extension

The following is the code for the URLMetaScoringFilter 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.

```
package org.apache.nutch.scoring.urlmeta;

import java.util.Collection;
import java.util.Map.Entry;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.io.Text;
import org.apache.nutch.crawl.CrawlDatum;
import org.apache.nutch.crawl.Inlinks;
import org.apache.nutch.indexer.NutchDocument;
import org.apache.nutch.parse.Parse;
import org.apache.nutch.parse.ParseData;
import org.apache.nutch.protocol.Content;
import org.apache.nutch.scoring.ScoringFilter;
import org.apache.nutch.scoring.ScoringFilterException;

/**
 * For documentation:
 *
 * @see URLMetaIndexingFilter
 */
public class URLMetaScoringFilter extends Configured implements ScoringFilter {

  private static final Log LOG = LogFactory.getLog(URLMetaScoringFilter.class);
  private static final String CONF_PROPERTY = "urlmeta.tags";
  private static String[] urlMetaTags;
  private Configuration conf;

  /**
   * This will take the metatags that you have listed in your "urlmeta.tags"
   * property, and looks for them inside the parseData object. If they exist,
   * this will be propagated into your 'targets' Collection's ["outlinks"]
   * attributes.
   *
   * @see ScoringFilter#distributeScoreToOutlinks
   */
  public CrawlDatum distributeScoreToOutlinks(Text fromUrl,
      ParseData parseData, Collection<Entry<Text, CrawlDatum>> targets,
      CrawlDatum adjust, int allCount) throws ScoringFilterException {
    if (urlMetaTags == null || targets == null || parseData == null)
      return adjust;

    Iterator<Entry<Text, CrawlDatum>> targetIterator = targets.iterator();

    while (targetIterator.hasNext()) {
      Entry<Text, CrawlDatum> nextTarget = targetIterator.next();

      for (String metatag : urlMetaTags) {
        String metaFromParse = parseData.getMeta(metatag);

        if (metaFromParse == null)
          continue;

        nextTarget.getValue().getMetaData().put(new Text(metatag),
            new Text(metaFromParse));
      }
    }
    return adjust;
  }

  /**
   * Takes the metadata, specified in your "urlmeta.tags" property, from the
```

```
 * datum object and injects it into the content. This is transfered to the
 * parseData object.
 *
 * @see ScoringFilter#passScoreBeforeParsing
 * @see URLMetaScoringFilter#passScoreAfterParsing
 */
public void passScoreBeforeParsing(Text url, CrawlDatum datum, Content content) {
  if (urlMetaTags == null || content == null || datum == null)
    return;

  for (String metatag : urlMetaTags) {
    Text metaFromDatum = (Text) datum.getMetaData().get(new Text(metatag));

    if (metaFromDatum == null)
      continue;

    content.getMetadata().set(metatag, metaFromDatum.toString());
  }
}

/**
 * Takes the metadata, which was lumped inside the content, and replicates it
 * within your parse data.
 *
 * @see URLMetaScoringFilter#passScoreBeforeParsing
 * @see ScoringFilter#passScoreAfterParsing
 */
public void passScoreAfterParsing(Text url, Content content, Parse parse) {
  if (urlMetaTags == null || content == null || parse == null)
    return;

  for (String metatag : urlMetaTags) {
    String metaFromContent = content.getMetadata().get(metatag);

    if (metaFromContent == null)
      continue;

    parse.getData().getParseMeta().set(metatag, metaFromContent);
  }
}

/** Boilerplate */
public float generatorSortValue(Text url, CrawlDatum datum, float initSort)
    throws ScoringFilterException {
  return initSort;
}

/** Boilerplate */
public float indexerScore(Text url, NutchDocument doc, CrawlDatum dbDatum,
    CrawlDatum fetchDatum, Parse parse, Inlinks inlinks, float initScore)
    throws ScoringFilterException {
  return initScore;
}

/** Boilerplate */
public void initialScore(Text url, CrawlDatum datum)
    throws ScoringFilterException {
  return;
}

/** Boilerplate */
public void injectedScore(Text url, CrawlDatum datum)
    throws ScoringFilterException {
  return;
}

/** Boilerplate */
public void updateDbScore(Text url, CrawlDatum old, CrawlDatum datum,
    List inlinked) throws ScoringFilterException {
  return;
}
```

```
  /**
   * handles conf assignment and pulls the value assignment from the
   * "urlmeta.tags" property
   */
  public void setConf(Configuration conf) {
    super.setConf(conf);

    if (conf == null)
      return;

    urlMetaTags = conf.getStrings(CONF_PROPERTY);
  }

  /** Boilerplate */
  public Configuration getConf() {
    return conf;
  }
}
```

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>protocol-http|urlfilter-regex|parse-(html|tika)|index-(basic|anchor)|scoring-opic|urlnormalizer-
(pass|regex|basic)|urlmeta</value>
  <description>Regular expression naming plugin directory 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 name of the **urlmeta** plugin.

Getting Ant to Compile Your Plugin

In order for ant to compile and deploy your plugin 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="urlmeta" 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 both the scoring and indexing extension will be used which will enable us to search for meta tags within our Solr index.

"This was written for Nutch 1.3 official release. There are various other plugin examples available at the Old Plugin Central for older implementations of previously used Nutch Plugins.

<<< See also: HowToContribute

<<< PluginCentral