ParserFactoryImprovementProposal

Parser Factory Improvement Proposal

Jerome Charron <jerome.charron@gmail.com>, Sébastien Le Callonnec <slc_ie@yahoo.ie>, Chris A. Mattmann <chris.mattmann@jpl.nasa.gov>

Wednesday, September 14th, 2005

DRAFT

Summary of Issue

Currently Nutch provides a plugin mechanism wherein plugins register certain metadata about themselves, including their id, classname, and so forth. In particular, the set of parsing plugins register which contentTypes and file suffixes they can support with a PluginRepository.

One “adopted practice” in current Nutch parsing plugins (committed in Subversion, e.g., see parse-pdf, parse-rss, etc.) has also been to verify that the content type passed to it during a fetch is indeed one of the contentTypes that it supports (be it application/xml, or application/pdf, etc.). This practice is cumbersome for a few reasons:

*Any updates to supported content types for a parsing plugin will require a recompilation of the plugin code
*Checking for “hard coded” content types within the parsing plugin is a duplication of information that already exists in the plugin’s descriptor file, plugin.xml
*By the time that content gets to a parsing plugin, (e.g., the parsing plugin is returned by the ParserFactory, and provided content during a fetch), the ParsingFactory should have already ensured that the appropriate plugin is getting called for a particular contentType.

In addition to this problem is the fact that several parsing plugins may all support many of the same content types. For instance, the parse-js plugin may be the only well suited parsing plugin for javascript, but perhaps it may also provided a good enough heuristic parser for plain text as well, and so it may support both types. However, there may be a parsing plugin for text (which there is!), parse-text, whose primary purpose is to parse plain text as well.

Suggested Remedy

To deal with ensuring the desired parsing plugin is called for the appropriate content type, and to in effect, “kill two birds with one stone”, we propose that there be a parsing plugin preference list for each content type that Nutch knows how to handle, i.e., each content type available via the mimeType system. Therefore, during a fetch, once the appropriate mimeType has been determined for content, and the ParserFactory is tasked with returning a parsing plugin, the ParserFactory should consult a preference list for that contentType, allowing it to determine which plugin has the highest preference for the contentType. That parsing plugin should be returned via the ParserFactory to the fetcher. If there is any problem using the initial returned parsing plugin for a particular contentType (i.e., if a ParseException is throw during the parser, or a null ParseStatus is returned), then the ParserFactory should be called again, this time asking for the “next highest ranked” plugin for that contentType. Such a process should repeat on and on until the parse is successful.

We propose that the “plugin preference list” should be a separate file that lives in $NUTCH_HOME/conf called “parse-plugins.xml”. The format of the file (full DTD to be developed during coding) should be something like:

```
<parse-plugins>

  <mimeType name="*">
      <plugin id=”parse-text” order=”1”/>
      <plugin id=”another-one-default-parser” order=”2”/>
     ....
  </mimeType>

  <mime-type name="application/vnd.ms-powerpoint">
   <!-- if no order is specified, then order is significant -->
    <plugin id="parse-mspowerpoint"/>
  </mime-type>

  <mime-type name="application/pdf">
    <plugin id="parse-pdf" order="1"/>
    <plugin id="parse-pdf-worse" order="2" />
  </mime-type>
  ....
</parse-plugins>
```

Activating Parse Plugins

If an activated parse plugin is not listed in the parse-plugins.xml, then it won’t get called for parsing. The purpose of the parse-plugins.xml file would be to map parsing-plugin to contentType. Therefore, if an activated plugin is not mapped to a content type, then it is “activated”, but won’t get called. This is very similar to Apache HTTPD. See below:

```
//httpd.conf example
//add handler for php

LoadModule php4_module        libexec/httpd/libphp4.so

// map handler to mimeType
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps

AddHandler php-script   php
AddHandler php-script   phps
```

There are two different levels in the above example. First, the plugin is “activated” in the LoadModule section. Then, the plugin is “mapped” to a content type in the AddHandler section. We believe that this is the way to go. Apache HTTPD is pervasive, and its model is well understood by many of the same folks who would want to use Nutch. Although we realize that this is a change from the way that Nutch currently works, and that people don’t like change, we believe that this change is entirely needful and represents something that Nutch should adopt.

Maintaining consistency between parse-plugins.xml and nutch-default.xml activated plugins

An interesting question arises in the following two examples:

1. No plugin defined in parse-plugins for a specified content-type, but many activated plugins that can deal with this content-type. 2. Many plugins defined in the parse-plugins for a specified content-type, but with the same priority

This unfortunately is something that as developers we cannot elegantly prevent in this case – erroneous input by the user. We propose a simple way to handle this is:

For example 1 above – if the user activates many parser plugins via the plugin.includes nutch conf property, but then fails to map those plugins to a particular content type in the parse-plugins.xml, then the plugins simply won't get called for parsing. They will be enabled, but will not be returned as viable parsers in the ordered list of parsing plugins for a content type.

For example 2 above - if the user specifies multiple parser plugin ids for a content type in parse-plugins.xml with the same priority, then LOG.severe(), and exit. This isn’t anything outside of what other systems do with bogus user input. For instance, in Apache HTTPD, if a user specifies that .cgi files should be handled by a text-handler, *and* by a perl-handler, Apache HTTPD will come back, and log an error message, and exit, which we believe is the correct thing to do in that case. The parse-plugins.xml file will need to be examined by the users of the Nutch system, and they will need to ensure that they don’t’ set the priorities for 2 different parse plugins to be the same for a particular mimeType. We propose to note this in a comment in the parse-plugins.xml file, and then also note it as a major change in the Nutch installation process.

Path Suffix Attribute in plugin.xml files and erroneous mime types returned by web servers for files

Another one of the main impacts of having a file like parse-plugins.xml is that no longer should the pathSuffix="" be part of the plugin.xml descriptor. We propose to move that out of plugin.xml and into the mime-types.xml file. Additionally, we can also "kill two birds with one stone" here and handle an oft-occuring problem users are experiencing with Nutch in terms of erroneous mime types returned by web servers for particular files. Specifically we propose to add an MimeType Alias mapper to the mime-types.xml file that will allow us to map the standard IANA mime types to other web server returned mime types that are non-standard. These two proposed changes to mime-types.xml would look like the following:

```
<!-- mime-types.xml file -->
  <mime-type name="application/vnd.ms-powerpoint">
    <!-- pathSuffix lives here now -->
      <ext>ppt</ext>
      <magic offset="....." type="..." value="..."/>

    <!-- here are other mime types that are not the default IANA mime types, but still returned by servers -->
      <mapped-type name="application/powerpoint"/>
      <mapped-type name="application/mspowerpoint"/>
   </mime-type>
```

To handle this mapping, two new methods should be added to the mime types class. In particular, we propose a `public static MimeType map (MimeType);` method and a `public static MimeType map(String);` method to be added to the MimeType java class to handle the mapping in the mime-types.xml file.

Proposal Task Summary

To summarize, our proposal to improve the parser factory consists of the following tasks:

1. Provide a mime-type mapper (based on IANA) in the util.mime package. Implementation to be refined: Uses and extension of the existing mime-type.dtd 2. Define a schema for the parse-plugin.xml file 3. Deprecate the pathSuffix from plugin.xml file 4. ParserFactory must check the content-type used in the parse-plugin.xml file and the content-type(s) specified in the plugin.xml; If it matches, all is ok, if not the plugin is not used.

Architectural Impact

Components

*Fetcher
*PluginSystem
*ParserFactory
*MimeTypeSystem

Impact on current releases of Nutch

Incompatibilities

By moving the pathSuffix out of the plugin.xml file, and into the mime-types.xml file, this would create an updated version of the plugin.xml descriptor schema for each plugin, along with an updated mime-types.xml descriptor schema. Additionally, storing the mime type aliases in the mime-types.xml file will also require an addition to the mime-types.xml schema. To lessen the effect on previous and near-term releases of Nutch the pathSuffix attribute could be left as an option in the plugin.xml schema, but marked as “deprecated” to let people know that this functionality isn’t part of the parse plugin identification process anymore, but it is left in the schema so as not to create incompatibilities with the plugin.xml files that people have already written. However, ultimately in future releases of Nutch, we propose that the pathSuffix attribute should be removed from the plugin.xml schema.

The proposed capability addition will simply control the order in which parsing plugins get called during fetching activities. It won’t directly impact the segments stored, or the webapp. It will only affect the fetcher component, and the mime types component.

Issues

The proposed new capabilities should be first tested on local systems, and if successful, uploaded to JIRA, and verified against the latest SVNs. Unit tests should be written to verify appropriate plugin parsing order. Users will need to be notified in the Nutch tutorial and instruction lists about how to set up the parsing plugin preferences prior to performing a fetch.

Personnel

*Jerome Charron
*Sébastien Le Callonnec
*Chris A. Mattmann

Timeframe

- Nutch Release 0.8
- (Merge in 0.7.1 ?)

Affected files

*ParserFactory.java
*plugin.xml descriptor files
*mime-types.xml file
*addition of parse-plugins.xml file
*files in package `org.apache.nutch.util.mime`