XsltResponseWriter

Overview

The XSLT Response Writer captures the output of the XML Response Writer and applies an XSLT transform to it.

This can be used to generate any text-based format from the Solr query results, as the Content-Type of the response is supplied by the XSLT transform, as shown below.

Example

With Solr's example configuration, this URL uses the XSLT Response Writer with the supplied example.xsl transform:

```
http://localhost:8983/solr/select/?q=apache&wt=xslt&tr=example.xsl
```

Parameters

tr parameter

Selects the XSLT transformation to use, which must be found in Solr's conf/xslt directory.

The Content-Type of the response is set according to the <xsl:output> statement in the XSLT transform, for example:

```
<xsl:output media-type="text/html"/>
```

Note that on Windows, the encoding defaults to 8859-1. To ensure that UTF-8 is used for encoding, use this:

```
<xsl:output media-type="text/xml;charset=utf-8"/>
```

Configuration

The XSLT Response Writer is configured as in this example (from the default solrconfig.xml):

```
<!--
  Changes to XSLT transforms are taken into account
  every xsltCacheLifetimeSeconds at most.
-->
<queryResponseWriter name="xslt" class="org.apache.solr.response.XSLTResponseWriter">
  <int name="xsltCacheLifetimeSeconds">5</int>
</queryResponseWriter>
```

A value of 5 for xsltCacheLifetimeSeconds is good for development, to see XSLT changes quickly. For production you probably want a much higher value.

Performance

The last XSLT transform used is cached, and reread and recompiled every xsltCacheLifetimeSeconds, even if it has not changed. This simplistic caching mechanism is good enough in many cases, but heavy use of XSLT transforms might require implementing a better caching mechanism (or ~~shameless plug~~ put Cocoon in front of Solr for more serious XSLT-based stuff).

When first used, the XSLT Response Writer outputs the following warning to the Solr log, to make sure you're aware of this limitation:

```
ATTENTION: The TransformerProvider's simplistic XSLT caching mechanism is
not appropriate for high load scenarios, unless a single XSLT transform is used
and xsltCacheLifetimeSeconds is set to a sufficiently high value.
```

Also, the output of the XML Response Writer is captured in memory for the XSLT transform. Big results set could cause high RAM usage.

The performance of XSLT transforms can vary dramatically depending on how well they're written - you've been warned.

Using Saxon for XSLT 2.0 Transforms

The XsltResponseWriter can easily be changed to use other transformers. The Saxon-B open source XSL transformer supports XSL 2.0 features, such as URI encoding, time and date functions, and field grouping. To change the default transformer to use Saxon, simply copy the Saxon .jar files into your classpath and set the system property javax.xml.TransformerFactory to "net.sf.saxon.TransfomerFactoryImpl". No changes to the Solr source are needed.

For Jetty (in the Solr example), make sure all the saxon-*.jars are in Jetty's ext/ folder (or lib/ext folder in current version of Jetty), and start Jetty like:

```
java -Djavax.xml.transform.TransformerFactory=net.sf.saxon.TransformerFactoryImpl -jar start.jar
```

For Resin, put this in your resin.conf:

```
<system-property javax.xml.transform.TransformerFactory="net.sf.saxon.TransformerFactoryImpl"/>
```

Grouping example

Now that you've got Saxon-B running, make sure your XSL files state the right version:

```
<xsl:stylesheet version='2.0'...
```

and you can use the best new feature for Solr users of XSL, for-each-grouping. Here I'll group by the Solr field 'username' and show the "comments" field per document:

```
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet version='2.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:output media-type="text/html; charset=UTF-8" encoding="UTF-8" indent="yes" omit-xml-declaration="yes" />
  <xsl:template match='/'>
    <xsl:for-each-group select="response/result/doc" group-by="str[@name='username']">
      <P>Documents for:
        <xsl:value-of select="current-grouping-key()"/></P>
        <ul>
          <xsl:for-each select="current-group()">
            <li><xsl:value-of select="str[@name='comments']"/></li>
          </xsl:for-each>
        </ul>
      </xsl:for-each-group>
  </xsl:template>
</xsl:stylesheet>
```

Date example

With Saxon installed you can create valid RSS using the date functionality. (http://www.w3.org/TR/xslt20/#function-format-date)

```
<!-- Convert from ISO 8601 dateformat to rfc 822 dateformat (XSLT 2.0) -->
  <xsl:template name="ISO8601-to-rfc822">
          <!-- Input:  2009-01-25T23:20:30.45+01:00 -->
          <!-- Output: Sun, 25 Jan 2009 20:30:45 +0100 GMT -->
          <xsl:param name="date"/>
          <xsl:value-of select="format-dateTime($date, '[F, 3-3], [D, 2-2] [MNn, 3-3] [Y] [h01]:[m01]:[s01]
[z]')"/>
  </xsl:template>
```

Call this with something like:

```
...
<pubDate><xsl:call-template name="ISO8601-to-rfc822"><xsl:with-param name="date" select="date[@name='changed']"
/></xsl:call-template></pubDate>
...
```

CategoryQueryResponseWriter