UnicodeCollation

Unicode Collation

⚠️ Solr3.1

Overview

Unicode Collation is a method to sort text in a language-sensitive way. It is primarily intended for sorting, but can also be used for advanced search purposes.

Unicode Collation in Solr is fast, all the work is done at index time. The way it works is that instead of just using a KeywordTokenizerFactory to create a sort field, you use KeywordTokenizerFactory followed by CollationKeyFilterFactory. At index time this indexes special "sort keys" into the sort field, so that at search you just sort on the sort field, and it comes back in collated order.

For more information, see the Javadocs.

Sorting text for a specific language

In the example below, text will be sorted according to the default German rules provided by Java. The rules for sorting German in Java are defined in a package called a Java Locale.

Locales are typically defined as a combination of language and country, but you can specify just the language if you want. For example, if you specify "de" as the language, you will get sorting that works well for German language. If you specify "de" as the language and "CH" as the country, you will get German sorting specifically tailored for Switzerland.

You can see a list of supported Locales here.

```
<!-- define a field type for German collation -->
<fieldType name="collatedGERMAN" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.KeywordTokenizerFactory"/>
    <filter class="solr.CollationKeyFilterFactory"
        language="de"
        strength="primary"
    />
  </analyzer>
</fieldType>
...
<!-- define a field to store the German collated manufacturer names -->
<field name="manuGERMAN" type="collatedGERMAN" indexed="true" stored="false" />
...
<!-- copy the text to this field. we could create French, English, Spanish versions too, and sort differently
for different users! -->
<copyField source="manu" dest="manuGERMAN"/>
```

In the example above, you will notice we defined the strength as "primary". The strength of the collation determines how "picky" the sort order will be, but depends upon the language. For example in English, "primary" strength ignores differences in case and accents.

For more information, see the Collator javadocs.

Sorting text for multiple languages

There are two approaches to supporting multiple languages:

- If there is a small list of languages you wish to support, consider defining collated fields for each language and using copyField.
- However, adding a large number of sort fields can increase disk and indexing costs. An alternative approach is to use the Unicode "default" collator.

The Unicode default, or "ROOT" Locale, has rules that are designed to work well in general for most languages. To use it, simply define the language as the empty string.

This Unicode default sort is still significantly more advanced than the standard Solr sort.

```
<fieldType name="collatedROOT" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.KeywordTokenizerFactory"/>
    <filter class="solr.CollationKeyFilterFactory"
        language=""
        strength="primary"
    />
  </analyzer>
</fieldType>
```

Sorting text with custom rules

For advanced usage, you can define your own set of rules that determine how the sorting takes place. Its easiest not to start from scratch, but instead to take existing rules that are close to what you want, and "tailor" or customize them.

In the example below, we create a custom ruleset for German known as DIN 5007-2. This ruleset treats umlauts in German differently, for example it treats ö as equivalent to oe.

For more information, see the RuleBasedCollator javadocs.

The example code below shows how to create a custom ruleset and dump it to a file.

```
    // get the default rules for Germany
    // these are called DIN 5007-1 sorting
    RuleBasedCollator baseCollator = (RuleBasedCollator) Collator.getInstance(new Locale("de", "DE"));

    // define some tailorings, to make it DIN 5007-2 sorting.
    // For example, this makes ö equivalent to oe
    String DIN5007_2_tailorings =
      "& ae , a\u0308 & AE , A\u0308"+
      "& oe , o\u0308 & OE , O\u0308"+
      "& ue , u\u0308 & UE , u\u0308";

    // concatenate the default rules to the tailorings, and dump it to a String
    RuleBasedCollator tailoredCollator = new RuleBasedCollator(baseCollator.getRules() + DIN5007_2_tailorings);
    String tailoredRules = tailoredCollator.getRules();
    // write these to a file, be sure to use UTF-8 encoding!!!
    IOUtils.write(tailoredRules, new FileOutputStream("/solr_home/conf/customRules.dat"), "UTF-8");
```

This file of rules can now be used for custom collation in Solr.

```
<fieldType name="collatedCUSTOM" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.KeywordTokenizerFactory"/>
    <filter class="solr.CollationKeyFilterFactory"
        custom="customRules.dat"
        strength="primary"
    />
  </analyzer>
</fieldType>
```

Searching

For advanced use cases, Collation can be used for search as well, on a tokenized field.

In the example below, we use the same custom German rules defined above on a tokenized field. Just like when using a stemmer, although the output tokens are nonsense, they are the same values and will match for search purposes.

```
<fieldType name="collatedCUSTOM" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.CollationKeyFilterFactory"
        custom="customRules.dat"
        strength="primary"
    />
  </analyzer>
</fieldType>
```

Below is an example of what this would look like for two words that should match with this collator: Töne and toene.

org.apache.solr.analysis.StandardTokenizerFactory

term position	1	2
term text	Töne	toene
term type	<ALPHANUM>	<ALPHANUM>
source start, end	0,4	5,10
payload		

org.apache.solr.analysis.CollationKeyFilterFactory {strength=primary, custom=customRules.dat}

term position	1	2
term text	3#6;#0;#0;#0;	3#6;#0;#0;#0;
term type	<ALPHANUM>	<ALPHANUM>
source start, end	0,4	5,10
payload		

Please note that the strange output you see from the filter is really a binary collation key encoded in a special form. What is important is that it is the same value for equivalent tokens as defined by that collator.

ICU Collation

For better performance, less memory usage, and support for more locales, you can add the analysis-extras contrib and use ICUCollationKeyFilterFactory instead. See the javadocs for more information.

In general, the principles are the same, you just specify an RFC3066 language identifier with the locale parameter instead of specifying language+country+variant.

For example, to get German phonebook sort order:

```
<fieldType name="collatedICU" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.KeywordTokenizerFactory"/>
    <filter class="solr.ICUCollationKeyFilterFactory"
        locale="de@collation=phonebook"
        strength="primary"
    />
  </analyzer>
</fieldType>
```

To use this filter, see solr/contrib/analysis-extras/README.txt for instructions on which jars you need to add to your SOLR_HOME/lib