Character Mapping Component

Ivan Provalov
iprovalov@netflix.com

Description

One of the challenges in search is recall of an item with a common typing variant. These cases can be as simple as lower/upper case in most languages, accented characters, or more complex morphological phenomena like prefix omitting, or constructing a character with some combining mark. This component addresses the cases, which are not covered by ASCII folding component, or more complex to design with other tools. The idea is that a linguist could provide the mappings in a tab-delimited file, which then can be directly used by Solr.

Some Use Cases

Case	Language	From	Source Description	To	Target Description
Voiced vowels	Japanese	ゴ	katakana letter ko 12467 and combining voiced sound mark 12468	コ	katakana letter ko 12467
Common Typing Substitutions	Korean	ㅅ	hangul jongseong sios 4538	ㅅ	hangul letter sios 12613
	Russian	ë	cyrillic small letter io 1105	e	cyrillic small letter ie 1077
	Polish	ś	latin small letter s with acute 347	s	latin small letter s 115
Transliteration	Polish	dż	[dʑ]	j	[dʑ]
	Arabic	غ	arabic letter ghain 1594	ج	arabic letter jeem 1580
		ج	arabic letter jeem 1580	ك	arabic letter kaf 1603
		ك	arabic letter kaf 1603	ق	arabic letter qaf 1602
		خ	arabic letter khah 1582	ق	arabic letter qaf 1602
Prefix Removal	Arabic	ال	alef lam		
Suffix folding	Japanese	ァ	katakana letter small a 12449	ア	katakana letter a 12450

Character Sequence Mappings

The mappings are maintained in the tab-delimited file, which could be just a copy paste from Excel spreadsheet. This gives the linguists the opportunity to create the mappings, then for the developer to include them in Solr configuration. There are a few cases, when the mappings

grow complex, where some additional debugging may be required. The mappings can contain any sequence of characters to any other sequence of characters.

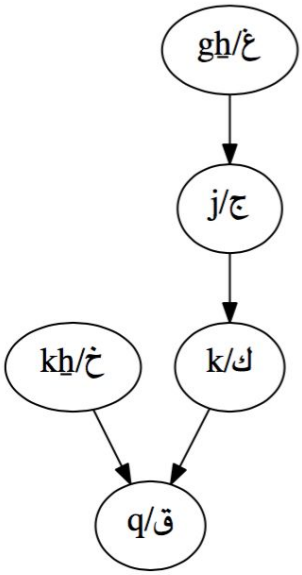
Graph Collapsing

In some cases, the mappings can grow significantly, and it is hard to recognize the that the mappings could be simplified (if A->B->C, then A->C). In the Arabic transliteration example above, all four letters map to arabic letter ‘qaf’, by applying the collapsing during the addition to the data structure. These mappings can be easily visualized during the development time, by producing the .dot file and running a graphviz tool.

Above example will produce a graph .dot file (label attribute includes the original character sequence as well as a transliterated label) like this:

```
digraph {
	gẖ [label="غ/gẖ"];
	j [label="ج/j"];
		gẖ->j
	j [label="ج/j"];
	k [label="ك/k"];
		j->k
	k [label="ك/k"];
	q [label="ق/q"];
		k->q
	kẖ [label="خ/kẖ"];
	q [label="ق/q"];
		kẖ->q
}
```

Which visualizes like this:



There is no need to organize the mappings to only map every of the four characters to the parent - "ق/q", the mapping component does this for you.

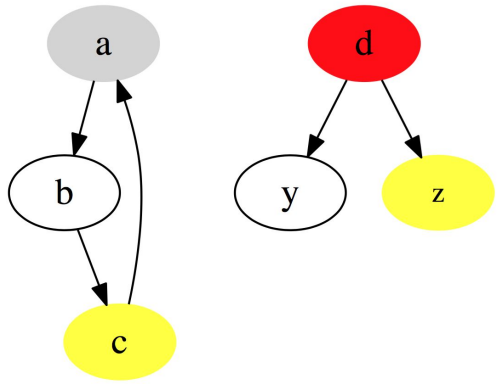
Circular Reference and Repeated Keys

In large mapping files, sometimes it is easy to add the same key twice, or create the circular dependency. These will not be added to the structure, and an exception will be logged. Also, the bad nodes will be added to the graph for debugging purposes.

Given a graph like so:

```
Digraph{
        a->b
        b->c
        c->a
        d->y
        d->z
}
```

Where ‘a’->’b’->’c’->’a’ is a circular reference and ‘d’ is a duplicate key, ambiguously mapping to ‘y’ and ‘z’, the following graph will be generated:



Component Configuration

Parameters

mappingFile - e.g. "lang/mapping-ja.txt"
protectedWordsFile - e.g. "lang/protected-ja.txt"

Morphological Features

minPrefix - check for potential substitution character(s) starts with minPrefix+1 position (e.g. for suffix folding, one may only consider tokens starting in position two)
minLength - checks the min length of the token before applying the transformation
keepOriginalToken - e.g. in suffix folding (autocomplete case) it is useful to keep both tokens if ngram indexing is used
matchFromEnd - for prefix vs suffix replacement (suffix folding case)
isPrefixMatch - whether a match can start in any position, or only prefix match (Arabic prefix removal)
isFirstTokenOnly - only apply the substitutions to the first token in the field (Korean last names removal)

Debugging

transliteratorForDebug - ICU ID for debugging the keys/values, e.g. “Hiragana-Latin”. You would need to add an implementation of the ICU String Transformer and replace it in your CharacterMappingCollapser for this feature to work (see Appendix B)

dotFile - e.g. "logs/mapping-ja.dot" to produce a .dot file (one could convert the produced dot file into pdf using the graphviz: `dot -Tpdf logs/ar-map.dot -o logs/ar-map.pdf`)

Design Notes

Character Mapper is implemented as trie-tree structure with a load-time graph structure to collapse the keys. The graph implementation allows to add the root as a value for every key (as in the Arabic transliteration example).

Other Applications

One could create a light weight stemmer with different configuration params and externalized morphemes (see Appendix A). Another potential application is tokenization for the Chinese.

Lucene JIRA

Opened a Lucene JIRA for this patch - LUCENE-7321

Appendix A - Russian Lightweight Stemmer

This is just an example of how a light weight stemmer could be implemented using this Character Mapping component:

schema.xml

```
…
<filter class="CharacterMappingFilterFactory" mappingFile="lang/stemming-ru-6.txt" minLength="6" matchFromEnd=”true”/>
<filter class="CharacterMappingFilterFactory" mappingFile="lang/stemming-ru-5.txt" minLength="5" matchFromEnd=”true”/>
<filter class="CharacterMappingFilterFactory" mappingFile="lang/stemming-ru-4.txt" minLength="4" matchFromEnd=”true”/>
<filter class="CharacterMappingFilterFactory" mappingFile="lang/stemming-ru-3.txt" minLength="3" matchFromEnd=”true”/>
<filter class="CharacterMappingFilterFactory" mappingFile="lang/stemming-ru-3-norm.txt" minLength="3" matchFromEnd=”true”/>
…
```

stemming-ru-3-norm.txt

```
ь
и
нн	н
```

stemming-ru-3.txt

```
а
е
и
о
у
й
ы
```

я
ь

stemming-ru-4.txt
ая
яя
ях
ою
ах
ею
их
ия
ию
ьв
ою
ую
ям
ых
ея
ам
ем
ей
ём
ев
ий
им
ое
ой
ом
ов
ые
ый
ым
ми

stemming-ru-5.txt
иям
иях
оях
ями
оям
оьв

ами
его
ему
ери
ими
ого
ому
ыми
оев

stemming-ru-6.txt
иями
оями

Appendix B: Debugging with ICU transliterated values - CharacterMappingIcuStringTransformer

```
import com.ibm.icu.text.Transliterator;
public class CharacterMappingIcuStringTransformer implements CharacterMappingStringTransformer
{
private final Transliterator transliterator;
public CharacterMappingIcuStringTransformer(String transliteratorForDebug) {
this.transliterator = Transliterator.getInstance(transliteratorForDebug,
Transliterator.FORWARD);
}
public String getTransformedString(String string) {
return transliterator.transform(string).replaceAll("\\~", "a0").replaceAll("\\'",
"a1").replaceAll("node", "node_").replaceAll("-", "_");
}
}
```