package com.twitter.search.common.relevance;

import java.util.Collections;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.cache.CacheBuilder;

import com.google.common.collect.ImmutableList;

import com.twitter.common\_internal.text.version.PenguinVersion;

import com.twitter.penguin.search.filter.StringMatchFilter;

import com.twitter.util.Duration;

/\*\*

\* the Cache for Trends

\*/

public class NGramCache {

private static final int DEFAULT\_MAX\_CACHE\_SIZE = 5000;

private static final long DEFAULT\_CACHE\_ITEM\_TTL\_SEC = 24 \* 3600; // 1 day

private final PenguinVersion penguinVersion;

// Keys are trends. Values are empty strings.

private final Map<String, String> trendsCache;

private volatile StringMatchFilter trendsMatcher = null;

/\*\*

\* Extract Trends from a list of normalized tokens

\*/

public List<String> extractTrendsFromNormalized(List<String> tokens) {

if (trendsMatcher == null) {

return Collections.emptyList();

}

ImmutableList.Builder<String> trends = ImmutableList.builder();

for (String trend : trendsMatcher.extractNormalized(tokens)) {

if (trendsCache.containsKey(trend)) {

trends.add(trend);

}

}

return trends.build();

}

/\*\*

\* Extract Trends from a list of tokens

\*/

public List<String> extractTrendsFrom(List<String> tokens, Locale language) {

if (trendsMatcher == null) {

return Collections.emptyList();

}

return trendsMatcher.extract(language, tokens);

}

/\*\*

\* Extract Trends from a given CharSequence

\*/

public List<String> extractTrendsFrom(CharSequence text, Locale language) {

if (trendsMatcher == null) {

return Collections.emptyList();

}

ImmutableList.Builder<String> trends = ImmutableList.builder();

for (String trend : trendsMatcher.extract(language, text)) {

if (trendsCache.containsKey(trend)) {

trends.add(trend);

}

}

return trends.build();

}

public long numTrendingTerms() {

return trendsCache.size();

}

public Set<String> getTrends() {

return trendsCache.keySet();

}

public void clear() {

trendsCache.clear();

trendsMatcher = null;

}

/\*\* Adds all trends to this NGramCache. \*/

public void addAll(Iterable<String> trends) {

for (String trend : trends) {

trendsCache.put(trend, "");

}

trendsMatcher = new StringMatchFilter(trendsCache.keySet(), penguinVersion);

}

public static Builder builder() {

return new Builder();

}

public static class Builder {

private int maxCacheSize = DEFAULT\_MAX\_CACHE\_SIZE;

private long cacheItemTTLSecs = DEFAULT\_CACHE\_ITEM\_TTL\_SEC; // 1 day

private PenguinVersion penguinVersion = PenguinVersion.PENGUIN\_4;

public Builder maxCacheSize(int cacheSize) {

this.maxCacheSize = cacheSize;

return this;

}

public Builder cacheItemTTL(long cacheItemTTL) {

this.cacheItemTTLSecs = cacheItemTTL;

return this;

}

public Builder penguinVersion(PenguinVersion newPenguinVersion) {

this.penguinVersion = Preconditions.checkNotNull(newPenguinVersion);

return this;

}

/\*\* Builds an NGramCache instance. \*/

public NGramCache build() {

return new NGramCache(

maxCacheSize,

Duration.apply(cacheItemTTLSecs, TimeUnit.SECONDS),

penguinVersion);

}

}

// Should be used only in tests that want to mock out this class.

@VisibleForTesting

public NGramCache() {

this(DEFAULT\_MAX\_CACHE\_SIZE,

Duration.apply(DEFAULT\_CACHE\_ITEM\_TTL\_SEC, TimeUnit.SECONDS),

PenguinVersion.PENGUIN\_4);

}

private NGramCache(int maxCacheSize, Duration cacheItemTTL, PenguinVersion penguinVersion) {

// we only have 1 refresher thread that writes to the cache

this.trendsCache = CacheBuilder.newBuilder()

.concurrencyLevel(1)

.expireAfterWrite(cacheItemTTL.inSeconds(), TimeUnit.SECONDS)

.maximumSize(maxCacheSize)

.<String, String>build()

.asMap();

this.penguinVersion = penguinVersion;

}

}