package com.twitter.search.ingester.pipeline.twitter;

import java.util.Collection;

import java.util.List;

import java.util.Map;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Maps;

import com.twitter.pink\_floyd.thrift.ClientIdentifier;

import com.twitter.pink\_floyd.thrift.Mask;

import com.twitter.pink\_floyd.thrift.Storer;

import com.twitter.pink\_floyd.thrift.UrlData;

import com.twitter.pink\_floyd.thrift.UrlReadRequest;

import com.twitter.util.Function;

import com.twitter.util.Future;

/\*\*

\* Resolve compressed URL via Pink

\*/

public class AsyncPinkUrlsResolver {

private final Storer.ServiceIface storerClient;

private final ClientIdentifier pinkClientId;

private final Mask requestMask;

// Use ServerSet to construct a metadata store client

public AsyncPinkUrlsResolver(Storer.ServiceIface storerClient, String pinkClientId) {

this.storerClient = storerClient;

this.pinkClientId = ClientIdentifier.valueOf(pinkClientId);

requestMask = new Mask();

requestMask.setResolution(true);

requestMask.setHtmlBasics(true);

requestMask.setUrlDirectInfo(true);

}

/\*\*

\* resolve urls calling pink asynchronously

\* @param urls urls to resolve

\* @return Future map of resolved urls

\*/

public Future<Map<String, ResolveCompressedUrlsUtils.UrlInfo>> resolveUrls(

Collection<String> urls) {

if (urls == null || urls.size() == 0) {

Future.value(Maps.newHashMap());

}

List<String> urlsList = ImmutableList.copyOf(urls);

UrlReadRequest request = new UrlReadRequest();

request.setUrls(urlsList);

request.setClientId(pinkClientId);

request.setMask(requestMask);

return storerClient.read(request).map(Function.func(

response -> {

Map<String, ResolveCompressedUrlsUtils.UrlInfo> resultMap = Maps.newHashMap();

for (UrlData urlData : response.getData()) {

if (ResolveCompressedUrlsUtils.isResolved(urlData)) {

resultMap.put(urlData.url, ResolveCompressedUrlsUtils.getUrlInfo(urlData));

}

}

return resultMap;

}

));

}

}