package com.twitter.tweetypie

package hydrator

import com.twitter.tweetypie.repository.TweetQuery

/\*\*

\* An instance of `TweetQueryOptionsExpander.Type` can be used to take a `TweetQuery.Options`

\* instance provided by a user, and expand the set of options included to take into account

\* dependencies between fields and options.

\*/

object TweetQueryOptionsExpander {

import TweetQuery.\_

/\*\*

\* Used by AdditionalFieldsHydrator, this function type can filter out or inject fieldIds to

\* request from Manhattan per tweet.

\*/

type Type = Options => Options

/\*\*

\* The identity TweetQueryOptionsExpander, which passes through fieldIds unchanged.

\*/

val unit: TweetQueryOptionsExpander.Type = identity

case class Selector(f: Include => Boolean) {

def apply(i: Include): Boolean = f(i)

def ||(other: Selector) = Selector(i => this(i) || other(i))

}

private def selectTweetField(fieldId: FieldId): Selector =

Selector(\_.tweetFields.contains(fieldId))

private val firstOrderDependencies: Seq[(Selector, Include)] =

Seq(

selectTweetField(Tweet.MediaField.id) ->

Include(tweetFields = Set(Tweet.UrlsField.id, Tweet.MediaKeysField.id)),

selectTweetField(Tweet.QuotedTweetField.id) ->

Include(tweetFields = Set(Tweet.UrlsField.id)),

selectTweetField(Tweet.MediaRefsField.id) ->

Include(tweetFields = Set(Tweet.UrlsField.id, Tweet.MediaKeysField.id)),

selectTweetField(Tweet.CardsField.id) ->

Include(tweetFields = Set(Tweet.UrlsField.id)),

selectTweetField(Tweet.Card2Field.id) ->

Include(tweetFields = Set(Tweet.UrlsField.id, Tweet.CardReferenceField.id)),

selectTweetField(Tweet.CoreDataField.id) ->

Include(tweetFields = Set(Tweet.DirectedAtUserMetadataField.id)),

selectTweetField(Tweet.SelfThreadInfoField.id) ->

Include(tweetFields = Set(Tweet.CoreDataField.id)),

(selectTweetField(Tweet.TakedownCountryCodesField.id) ||

selectTweetField(Tweet.TakedownReasonsField.id)) ->

Include(

tweetFields = Set(

Tweet.TweetypieOnlyTakedownCountryCodesField.id,

Tweet.TweetypieOnlyTakedownReasonsField.id

)

),

selectTweetField(Tweet.EditPerspectiveField.id) ->

Include(tweetFields = Set(Tweet.PerspectiveField.id)),

Selector(\_.quotedTweet) ->

Include(tweetFields = Set(Tweet.QuotedTweetField.id)),

// asking for any count implies getting the Tweet.counts field

Selector(\_.countsFields.nonEmpty) ->

Include(tweetFields = Set(Tweet.CountsField.id)),

// asking for any media field implies getting the Tweet.media field

Selector(\_.mediaFields.nonEmpty) ->

Include(tweetFields = Set(Tweet.MediaField.id)),

selectTweetField(Tweet.UnmentionDataField.id) ->

Include(tweetFields = Set(Tweet.MentionsField.id)),

)

private val allDependencies =

firstOrderDependencies.map {

case (sel, inc) => sel -> transitiveExpand(inc)

}

private def transitiveExpand(inc: Include): Include =

firstOrderDependencies.foldLeft(inc) {

case (z, (selector, include)) =>

if (!selector(z)) z

else z ++ include ++ transitiveExpand(include)

}

/\*\*

\* Sequentially composes multiple TweetQueryOptionsExpander into a new TweetQueryOptionsExpander

\*/

def sequentially(updaters: TweetQueryOptionsExpander.Type\*): TweetQueryOptionsExpander.Type =

options =>

updaters.foldLeft(options) {

case (options, updater) => updater(options)

}

/\*\*

\* For requested fields that depend on other fields being present for correct hydration,

\* returns an updated `TweetQuery.Options` with those dependee fields included.

\*/

def expandDependencies: TweetQueryOptionsExpander.Type =

options =>

options.copy(

include = allDependencies.foldLeft(options.include) {

case (z, (selector, include)) =>

if (!selector(options.include)) z

else z ++ include

}

)

/\*\*

\* If the gate is true, add 'fields' to the list of tweetFields to load.

\*/

def gatedTweetFieldUpdater(

gate: Gate[Unit],

fields: Seq[FieldId]

): TweetQueryOptionsExpander.Type =

options =>

if (gate()) {

options.copy(

include = options.include.also(tweetFields = fields)

)

} else {

options

}

/\*\*

\* Uses a `ThreadLocal` to remember the last expansion performed, and to reuse the

\* previous result if the input value is the same. This is useful to avoid repeatedly

\* computing the expansion of the same input when multiple tweets are queried together

\* with the same options.

\*/

def threadLocalMemoize(expander: Type): Type = {

val memo: ThreadLocal[Option[(Options, Options)]] =

new ThreadLocal[Option[(Options, Options)]] {

override def initialValue(): None.type = None

}

options =>

memo.get() match {

case Some((`options`, res)) => res

case \_ =>

val res = expander(options)

memo.set(Some((options, res)))

res

}

}

}