package com.twitter.timelineranker.model

import com.twitter.servo.util.Gate

import com.twitter.timelines.model.candidate.CandidateTweetSourceId

import com.twitter.timelineranker.{thriftscala => thrift}

import com.twitter.timelines.common.model.\_

import com.twitter.timelines.earlybird.common.options.EarlybirdOptions

import com.twitter.timelines.earlybird.common.utils.SearchOperator

import com.twitter.timelines.configapi.{

DependencyProvider => ConfigApiDependencyProvider,

FutureDependencyProvider => ConfigApiFutureDependencyProvider,

\_

}

import com.twitter.timelines.model.TweetId

import com.twitter.timelines.model.UserId

import com.twitter.timelineservice.DeviceContext

object RecapQuery {

val EngagedTweetsSupportedTweetKindOption: TweetKindOption.ValueSet = TweetKindOption(

includeReplies = false,

includeRetweets = false,

includeExtendedReplies = false,

includeOriginalTweetsAndQuotes = true

)

val DefaultSearchOperator: SearchOperator.Value = SearchOperator.Exclude

def fromThrift(query: thrift.RecapQuery): RecapQuery = {

RecapQuery(

userId = query.userId,

maxCount = query.maxCount,

range = query.range.map(TimelineRange.fromThrift),

options = query.options

.map(options => TweetKindOption.fromThrift(options.to[Set]))

.getOrElse(TweetKindOption.None),

searchOperator = query.searchOperator

.map(SearchOperator.fromThrift)

.getOrElse(DefaultSearchOperator),

earlybirdOptions = query.earlybirdOptions.map(EarlybirdOptions.fromThrift),

deviceContext = query.deviceContext.map(DeviceContext.fromThrift),

authorIds = query.authorIds,

excludedTweetIds = query.excludedTweetIds,

searchClientSubId = query.searchClientSubId,

candidateTweetSourceId =

query.candidateTweetSourceId.flatMap(CandidateTweetSourceId.fromThrift),

hydratesContentFeatures = query.hydratesContentFeatures

)

}

def fromThrift(query: thrift.RecapHydrationQuery): RecapQuery = {

require(query.tweetIds.nonEmpty, "tweetIds must be non-empty")

RecapQuery(

userId = query.userId,

tweetIds = Some(query.tweetIds),

searchOperator = DefaultSearchOperator,

earlybirdOptions = query.earlybirdOptions.map(EarlybirdOptions.fromThrift),

deviceContext = query.deviceContext.map(DeviceContext.fromThrift),

candidateTweetSourceId =

query.candidateTweetSourceId.flatMap(CandidateTweetSourceId.fromThrift),

hydratesContentFeatures = query.hydratesContentFeatures

)

}

def fromThrift(query: thrift.EngagedTweetsQuery): RecapQuery = {

val options = query.tweetKindOptions

.map(tweetKindOptions => TweetKindOption.fromThrift(tweetKindOptions.to[Set]))

.getOrElse(TweetKindOption.None)

if (!(options.isEmpty ||

(options == EngagedTweetsSupportedTweetKindOption))) {

throw new IllegalArgumentException(s"Unsupported TweetKindOption value: $options")

}

RecapQuery(

userId = query.userId,

maxCount = query.maxCount,

range = query.range.map(TimelineRange.fromThrift),

options = options,

searchOperator = DefaultSearchOperator,

earlybirdOptions = query.earlybirdOptions.map(EarlybirdOptions.fromThrift),

deviceContext = query.deviceContext.map(DeviceContext.fromThrift),

authorIds = query.userIds,

excludedTweetIds = query.excludedTweetIds,

)

}

def fromThrift(query: thrift.EntityTweetsQuery): RecapQuery = {

require(

query.semanticCoreIds.isDefined,

"entities(semanticCoreIds) can't be None"

)

val options = query.tweetKindOptions

.map(tweetKindOptions => TweetKindOption.fromThrift(tweetKindOptions.to[Set]))

.getOrElse(TweetKindOption.None)

RecapQuery(

userId = query.userId,

maxCount = query.maxCount,

range = query.range.map(TimelineRange.fromThrift),

options = options,

searchOperator = DefaultSearchOperator,

earlybirdOptions = query.earlybirdOptions.map(EarlybirdOptions.fromThrift),

deviceContext = query.deviceContext.map(DeviceContext.fromThrift),

excludedTweetIds = query.excludedTweetIds,

semanticCoreIds = query.semanticCoreIds.map(\_.map(SemanticCoreAnnotation.fromThrift).toSet),

hashtags = query.hashtags.map(\_.toSet),

languages = query.languages.map(\_.map(Language.fromThrift).toSet),

candidateTweetSourceId =

query.candidateTweetSourceId.flatMap(CandidateTweetSourceId.fromThrift),

includeNullcastTweets = query.includeNullcastTweets,

includeTweetsFromArchiveIndex = query.includeTweetsFromArchiveIndex,

authorIds = query.authorIds,

hydratesContentFeatures = query.hydratesContentFeatures

)

}

def fromThrift(query: thrift.UtegLikedByTweetsQuery): RecapQuery = {

val options = query.tweetKindOptions

.map(tweetKindOptions => TweetKindOption.fromThrift(tweetKindOptions.to[Set]))

.getOrElse(TweetKindOption.None)

RecapQuery(

userId = query.userId,

maxCount = query.maxCount,

range = query.range.map(TimelineRange.fromThrift),

options = options,

earlybirdOptions = query.earlybirdOptions.map(EarlybirdOptions.fromThrift),

deviceContext = query.deviceContext.map(DeviceContext.fromThrift),

excludedTweetIds = query.excludedTweetIds,

utegLikedByTweetsOptions = for {

utegCount <- query.utegCount

weightedFollowings <- query.weightedFollowings.map(\_.toMap)

} yield {

UtegLikedByTweetsOptions(

utegCount = utegCount,

isInNetwork = query.isInNetwork,

weightedFollowings = weightedFollowings

)

},

candidateTweetSourceId =

query.candidateTweetSourceId.flatMap(CandidateTweetSourceId.fromThrift),

hydratesContentFeatures = query.hydratesContentFeatures

)

}

val paramGate: (Param[Boolean] => Gate[RecapQuery]) = HasParams.paramGate

type DependencyProvider[+T] = ConfigApiDependencyProvider[RecapQuery, T]

object DependencyProvider extends DependencyProviderFunctions[RecapQuery]

type FutureDependencyProvider[+T] = ConfigApiFutureDependencyProvider[RecapQuery, T]

object FutureDependencyProvider extends FutureDependencyProviderFunctions[RecapQuery]

}

/\*\*

\* Model object corresponding to RecapQuery thrift struct.

\*/

case class RecapQuery(

userId: UserId,

maxCount: Option[Int] = None,

range: Option[TimelineRange] = None,

options: TweetKindOption.ValueSet = TweetKindOption.None,

searchOperator: SearchOperator.Value = RecapQuery.DefaultSearchOperator,

earlybirdOptions: Option[EarlybirdOptions] = None,

deviceContext: Option[DeviceContext] = None,

authorIds: Option[Seq[UserId]] = None,

tweetIds: Option[Seq[TweetId]] = None,

semanticCoreIds: Option[Set[SemanticCoreAnnotation]] = None,

hashtags: Option[Set[String]] = None,

languages: Option[Set[Language]] = None,

excludedTweetIds: Option[Seq[TweetId]] = None,

// options used only for yml tweets

utegLikedByTweetsOptions: Option[UtegLikedByTweetsOptions] = None,

searchClientSubId: Option[String] = None,

override val params: Params = Params.Empty,

candidateTweetSourceId: Option[CandidateTweetSourceId.Value] = None,

includeNullcastTweets: Option[Boolean] = None,

includeTweetsFromArchiveIndex: Option[Boolean] = None,

hydratesContentFeatures: Option[Boolean] = None)

extends HasParams {

override def toString: String = {

s"RecapQuery(userId: $userId, maxCount: $maxCount, range: $range, options: $options, searchOperator: $searchOperator, " +

s"earlybirdOptions: $earlybirdOptions, deviceContext: $deviceContext, authorIds: $authorIds, " +

s"tweetIds: $tweetIds, semanticCoreIds: $semanticCoreIds, hashtags: $hashtags, languages: $languages, excludedTweetIds: $excludedTweetIds, " +

s"utegLikedByTweetsOptions: $utegLikedByTweetsOptions, searchClientSubId: $searchClientSubId, " +

s"params: $params, candidateTweetSourceId: $candidateTweetSourceId, includeNullcastTweets: $includeNullcastTweets, " +

s"includeTweetsFromArchiveIndex: $includeTweetsFromArchiveIndex), hydratesContentFeatures: $hydratesContentFeatures"

}

def throwIfInvalid(): Unit = {

def noDuplicates[T <: Traversable[\_]](elements: T) = {

elements.toSet.size == elements.size

}

maxCount.foreach { max => require(max > 0, "maxCount must be a positive integer") }

range.foreach(\_.throwIfInvalid())

earlybirdOptions.foreach(\_.throwIfInvalid())

tweetIds.foreach { ids => require(ids.nonEmpty, "tweetIds must be nonEmpty if present") }

semanticCoreIds.foreach(\_.foreach(\_.throwIfInvalid()))

languages.foreach(\_.foreach(\_.throwIfInvalid()))

languages.foreach { langs =>

require(langs.nonEmpty, "languages must be nonEmpty if present")

require(noDuplicates(langs.map(\_.language)), "languages must be unique")

}

}

throwIfInvalid()

def toThriftRecapQuery: thrift.RecapQuery = {

val thriftOptions = Some(TweetKindOption.toThrift(options))

thrift.RecapQuery(

userId,

maxCount,

range.map(\_.toTimelineRangeThrift),

deprecatedMinCount = None,

thriftOptions,

earlybirdOptions.map(\_.toThrift),

deviceContext.map(\_.toThrift),

authorIds,

excludedTweetIds,

Some(SearchOperator.toThrift(searchOperator)),

searchClientSubId,

candidateTweetSourceId.flatMap(CandidateTweetSourceId.toThrift)

)

}

def toThriftRecapHydrationQuery: thrift.RecapHydrationQuery = {

require(tweetIds.isDefined && tweetIds.get.nonEmpty, "tweetIds must be present")

thrift.RecapHydrationQuery(

userId,

tweetIds.get,

earlybirdOptions.map(\_.toThrift),

deviceContext.map(\_.toThrift),

candidateTweetSourceId.flatMap(CandidateTweetSourceId.toThrift)

)

}

def toThriftEntityTweetsQuery: thrift.EntityTweetsQuery = {

val thriftTweetKindOptions = Some(TweetKindOption.toThrift(options))

thrift.EntityTweetsQuery(

userId = userId,

maxCount = maxCount,

range = range.map(\_.toTimelineRangeThrift),

tweetKindOptions = thriftTweetKindOptions,

earlybirdOptions = earlybirdOptions.map(\_.toThrift),

deviceContext = deviceContext.map(\_.toThrift),

excludedTweetIds = excludedTweetIds,

semanticCoreIds = semanticCoreIds.map(\_.map(\_.toThrift)),

hashtags = hashtags,

languages = languages.map(\_.map(\_.toThrift)),

candidateTweetSourceId.flatMap(CandidateTweetSourceId.toThrift),

includeNullcastTweets = includeNullcastTweets,

includeTweetsFromArchiveIndex = includeTweetsFromArchiveIndex,

authorIds = authorIds

)

}

def toThriftUtegLikedByTweetsQuery: thrift.UtegLikedByTweetsQuery = {

val thriftTweetKindOptions = Some(TweetKindOption.toThrift(options))

thrift.UtegLikedByTweetsQuery(

userId = userId,

maxCount = maxCount,

utegCount = utegLikedByTweetsOptions.map(\_.utegCount),

range = range.map(\_.toTimelineRangeThrift),

tweetKindOptions = thriftTweetKindOptions,

earlybirdOptions = earlybirdOptions.map(\_.toThrift),

deviceContext = deviceContext.map(\_.toThrift),

excludedTweetIds = excludedTweetIds,

isInNetwork = utegLikedByTweetsOptions.map(\_.isInNetwork).get,

weightedFollowings = utegLikedByTweetsOptions.map(\_.weightedFollowings),

candidateTweetSourceId = candidateTweetSourceId.flatMap(CandidateTweetSourceId.toThrift)

)

}

}