package com.twitter.home\_mixer.util.earlybird

import com.twitter.conversions.DurationOps.\_

import com.twitter.search.common.query.thriftjava.{thriftscala => scq}

import com.twitter.search.common.ranking.{thriftscala => scr}

import com.twitter.search.earlybird.{thriftscala => eb}

import com.twitter.timelines.clients.relevance\_search.SearchClient.TweetFeatures

import com.twitter.timelines.clients.relevance\_search.SearchClient.TweetTypes

import com.twitter.timelines.clients.relevance\_search.SearchQueryBuilder

import com.twitter.timelines.clients.relevance\_search.SearchQueryBuilder.QueryWithNamedDisjunctions

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

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

import com.twitter.util.Duration

object EarlybirdRequestUtil {

val DefaultMaxHitsToProcess = 1000

val DefaultSearchProcessingTimeout: Duration = 200.milliseconds

val DefaultHydrationMaxNumResultsPerShard = 1000

val DefaultQueryMaxNumResultsPerShard = 300

val DefaultHydrationCollectorParams = mkCollectorParams(DefaultHydrationMaxNumResultsPerShard)

private val queryBuilder = new SearchQueryBuilder

object EarlybirdScoringModels {

val UnifiedEngagementProd: Seq[EarlybirdScoringModelConfig] = Seq(

EarlybirdScoringModelConfig("timelines\_unified\_engagement\_prod.schema\_based", 1.0)

)

val UnifiedEngagementRectweet: Seq[EarlybirdScoringModelConfig] = Seq(

EarlybirdScoringModelConfig("timelines\_unified\_engagement\_rectweet.schema\_based", 1.0)

)

}

private[earlybird] def mkCollectorParams(numResultsToReturn: Int): scq.CollectorParams = {

scq.CollectorParams(

// numResultsToReturn defines how many results each EB shard will return to search root

numResultsToReturn = numResultsToReturn,

// terminationParams.maxHitsToProcess is used for early terminating per shard results fetching.

terminationParams = Some(

scq.CollectorTerminationParams(

maxHitsToProcess = Some(DefaultMaxHitsToProcess),

timeoutMs = DefaultSearchProcessingTimeout.inMilliseconds.toInt

))

)

}

private def getRankingParams(

authorScoreMap: Option[Map[Long, Double]],

tensorflowModel: Option[String],

ebModels: Seq[EarlybirdScoringModelConfig]

): Option[scr.ThriftRankingParams] = {

if (tensorflowModel.nonEmpty) {

Some(

scr.ThriftRankingParams(

`type` = Some(scr.ThriftScoringFunctionType.TensorflowBased),

selectedTensorflowModel = tensorflowModel,

minScore = -1.0e100,

applyBoosts = false,

authorSpecificScoreAdjustments = authorScoreMap

)

)

} else if (ebModels.nonEmpty) {

Some(

scr.ThriftRankingParams(

`type` = Some(scr.ThriftScoringFunctionType.ModelBased),

selectedModels = Some(ebModels.map(m => m.name -> m.weight).toMap),

applyBoosts = false,

minScore = -1.0e100,

authorSpecificScoreAdjustments = authorScoreMap

)

)

} else None

}

def getTweetsRequest(

userId: Option[Long],

clientId: Option[String],

skipVeryRecentTweets: Boolean,

followedUserIds: Set[Long],

retweetsMutedUserIds: Set[Long],

beforeTweetIdExclusive: Option[Long],

afterTweetIdExclusive: Option[Long],

excludedTweetIds: Option[Set[Long]] = None,

maxCount: Int,

tweetTypes: TweetTypes.ValueSet,

authorScoreMap: Option[Map[Long, Double]] = None,

tensorflowModel: Option[String] = None,

ebModels: Seq[EarlybirdScoringModelConfig] = Seq.empty,

queryMaxNumResultsPerShard: Int = DefaultQueryMaxNumResultsPerShard

): eb.EarlybirdRequest = {

val QueryWithNamedDisjunctions(query, namedDisjunctionMap) = queryBuilder.create(

followedUserIds,

retweetsMutedUserIds,

beforeTweetIdExclusive,

afterTweetIdExclusive,

semanticCoreIds = None,

languages = None,

tweetTypes = tweetTypes,

searchOperator = SearchOperator.Exclude,

tweetFeatures = TweetFeatures.All,

excludedTweetIds = excludedTweetIds.getOrElse(Set.empty),

enableExcludeSourceTweetIdsQuery = false

)

val ebRankingParams = getRankingParams(authorScoreMap, tensorflowModel, ebModels)

val relOptions = RelevanceSearchUtil.RelevanceOptions.copy(

rankingParams = ebRankingParams

)

val followedUserIdsSeq = followedUserIds.toSeq

val namedDisjunctionMapOpt =

if (namedDisjunctionMap.isEmpty) None

else Some(namedDisjunctionMap.mapValues(\_.toSeq))

val thriftQuery = eb.ThriftSearchQuery(

serializedQuery = Some(query.serialize),

fromUserIDFilter64 = Some(followedUserIdsSeq),

numResults = maxCount,

collectConversationId = true,

rankingMode = eb.ThriftSearchRankingMode.Relevance,

relevanceOptions = Some(relOptions),

collectorParams = Some(mkCollectorParams(queryMaxNumResultsPerShard)),

facetFieldNames = Some(RelevanceSearchUtil.FacetsToFetch),

resultMetadataOptions = Some(RelevanceSearchUtil.MetadataOptions),

searcherId = userId,

searchStatusIds = None,

namedDisjunctionMap = namedDisjunctionMapOpt

)

eb.EarlybirdRequest(

searchQuery = thriftQuery,

clientId = clientId,

getOlderResults = Some(false),

followedUserIds = Some(followedUserIdsSeq),

getProtectedTweetsOnly = Some(false),

timeoutMs = DefaultSearchProcessingTimeout.inMilliseconds.toInt,

skipVeryRecentTweets = skipVeryRecentTweets,

numResultsToReturnAtRoot = Some(maxCount)

)

}

def getTweetsFeaturesRequest(

userId: Option[Long],

tweetIds: Option[Seq[Long]],

clientId: Option[String],

getOnlyProtectedTweets: Boolean = false,

authorScoreMap: Option[Map[Long, Double]] = None,

tensorflowModel: Option[String] = None,

ebModels: Seq[EarlybirdScoringModelConfig] = Seq.empty

): eb.EarlybirdRequest = {

val candidateSize = tweetIds.getOrElse(Seq.empty).size

val ebRankingParams = getRankingParams(authorScoreMap, tensorflowModel, ebModels)

val relOptions = RelevanceSearchUtil.RelevanceOptions.copy(

rankingParams = ebRankingParams

)

val thriftQuery = eb.ThriftSearchQuery(

numResults = candidateSize,

collectConversationId = true,

rankingMode = eb.ThriftSearchRankingMode.Relevance,

relevanceOptions = Some(relOptions),

collectorParams = Some(DefaultHydrationCollectorParams),

facetFieldNames = Some(RelevanceSearchUtil.FacetsToFetch),

resultMetadataOptions = Some(RelevanceSearchUtil.MetadataOptions),

searcherId = userId,

searchStatusIds = tweetIds.map(\_.toSet),

)

eb.EarlybirdRequest(

searchQuery = thriftQuery,

clientId = clientId,

getOlderResults = Some(false),

getProtectedTweetsOnly = Some(getOnlyProtectedTweets),

timeoutMs = DefaultSearchProcessingTimeout.inMilliseconds.toInt,

skipVeryRecentTweets = true,

// This param decides # of tweets to return from search superRoot and realtime/protected/Archive roots.

// It takes higher precedence than ThriftSearchQuery.numResults

numResultsToReturnAtRoot = Some(candidateSize)

)

}

}