package com.twitter.unified\_user\_actions.adapter.client\_event

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.clientapp.thriftscala.{Item => LogEventItem}

import com.twitter.search.common.constants.thriftscala.ThriftQuerySource

import com.twitter.search.common.constants.thriftscala.TweetResultSource

import com.twitter.search.common.constants.thriftscala.UserResultSource

import com.twitter.suggests.controller\_data.search\_response.item\_types.thriftscala.ItemTypesControllerData

import com.twitter.suggests.controller\_data.search\_response.item\_types.thriftscala.ItemTypesControllerData.TweetTypesControllerData

import com.twitter.suggests.controller\_data.search\_response.item\_types.thriftscala.ItemTypesControllerData.UserTypesControllerData

import com.twitter.suggests.controller\_data.search\_response.request.thriftscala.RequestControllerData

import com.twitter.suggests.controller\_data.search\_response.thriftscala.SearchResponseControllerData.V1

import com.twitter.suggests.controller\_data.search\_response.thriftscala.SearchResponseControllerDataAliases.V1Alias

import com.twitter.suggests.controller\_data.thriftscala.ControllerData.V2

import com.twitter.suggests.controller\_data.v2.thriftscala.ControllerData.SearchResponse

import com.twitter.unified\_user\_actions.thriftscala.SearchQueryFilterType

import com.twitter.unified\_user\_actions.thriftscala.SearchQueryFilterType.\_

class SearchInfoUtils(item: LogEventItem) {

private val searchControllerDataOpt: Option[V1Alias] = item.suggestionDetails.flatMap { sd =>

sd.decodedControllerData.flatMap { decodedControllerData =>

decodedControllerData match {

case V2(v2ControllerData) =>

v2ControllerData match {

case SearchResponse(searchResponseControllerData) =>

searchResponseControllerData match {

case V1(searchResponseControllerDataV1) =>

Some(searchResponseControllerDataV1)

case \_ => None

}

case \_ =>

None

}

case \_ => None

}

}

}

private val requestControllerDataOptFromItem: Option[RequestControllerData] =

searchControllerDataOpt.flatMap { searchControllerData =>

searchControllerData.requestControllerData

}

private val itemTypesControllerDataOptFromItem: Option[ItemTypesControllerData] =

searchControllerDataOpt.flatMap { searchControllerData =>

searchControllerData.itemTypesControllerData

}

def checkBit(bitmap: Long, idx: Int): Boolean = {

(bitmap / Math.pow(2, idx)).toInt % 2 == 1

}

def getQueryOptFromSearchDetails(logEvent: LogEvent): Option[String] = {

logEvent.searchDetails.flatMap { sd => sd.query }

}

def getQueryOptFromControllerDataFromItem: Option[String] = {

requestControllerDataOptFromItem.flatMap { rd => rd.rawQuery }

}

def getQueryOptFromItem(logEvent: LogEvent): Option[String] = {

// First we try to get the query from controller data, and if that's not available, we fall

// back to query in search details. If both are None, queryOpt is None.

getQueryOptFromControllerDataFromItem.orElse(getQueryOptFromSearchDetails(logEvent))

}

def getTweetTypesOptFromControllerDataFromItem: Option[TweetTypesControllerData] = {

itemTypesControllerDataOptFromItem.flatMap { itemTypes =>

itemTypes match {

case TweetTypesControllerData(tweetTypesControllerData) =>

Some(TweetTypesControllerData(tweetTypesControllerData))

case \_ => None

}

}

}

def getUserTypesOptFromControllerDataFromItem: Option[UserTypesControllerData] = {

itemTypesControllerDataOptFromItem.flatMap { itemTypes =>

itemTypes match {

case UserTypesControllerData(userTypesControllerData) =>

Some(UserTypesControllerData(userTypesControllerData))

case \_ => None

}

}

}

def getQuerySourceOptFromControllerDataFromItem: Option[ThriftQuerySource] = {

requestControllerDataOptFromItem

.flatMap { rd => rd.querySource }

.flatMap { querySourceVal => ThriftQuerySource.get(querySourceVal) }

}

def getTweetResultSources: Option[Set[TweetResultSource]] = {

getTweetTypesOptFromControllerDataFromItem

.flatMap { cd => cd.tweetTypesControllerData.tweetTypesBitmap }

.map { tweetTypesBitmap =>

TweetResultSource.list.filter { t => checkBit(tweetTypesBitmap, t.value) }.toSet

}

}

def getUserResultSources: Option[Set[UserResultSource]] = {

getUserTypesOptFromControllerDataFromItem

.flatMap { cd => cd.userTypesControllerData.userTypesBitmap }

.map { userTypesBitmap =>

UserResultSource.list.filter { t => checkBit(userTypesBitmap, t.value) }.toSet

}

}

def getQueryFilterType(logEvent: LogEvent): Option[SearchQueryFilterType] = {

val searchTab = logEvent.eventNamespace.map(\_.client).flatMap {

case Some("m5") | Some("android") => logEvent.eventNamespace.flatMap(\_.element)

case \_ => logEvent.eventNamespace.flatMap(\_.section)

}

searchTab.flatMap {

case "search\_filter\_top" => Some(Top)

case "search\_filter\_live" => Some(Latest)

// android uses search\_filter\_tweets instead of search\_filter\_live

case "search\_filter\_tweets" => Some(Latest)

case "search\_filter\_user" => Some(People)

case "search\_filter\_image" => Some(Photos)

case "search\_filter\_video" => Some(Videos)

case \_ => None

}

}

def getRequestJoinId: Option[Long] = requestControllerDataOptFromItem.flatMap(\_.requestJoinId)

def getTraceId: Option[Long] = requestControllerDataOptFromItem.flatMap(\_.traceId)

}