package com.twitter.usersignalservice.signals

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.simclusters\_v2.thriftscala.InternalId

import com.twitter.strato.client.Client

import com.twitter.strato.data.Conv

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.twistly.thriftscala.RecentProfileClickImpressEvents

import com.twitter.twistly.thriftscala.ProfileClickImpressEvent

import com.twitter.usersignalservice.base.Query

import com.twitter.usersignalservice.base.StratoSignalFetcher

import com.twitter.usersignalservice.thriftscala.Signal

import com.twitter.usersignalservice.thriftscala.SignalType

import com.twitter.util.Future

import com.twitter.util.Timer

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class ProfileClickFetcher @Inject() (

stratoClient: Client,

timer: Timer,

stats: StatsReceiver)

extends StratoSignalFetcher[(UserId, Long), Unit, RecentProfileClickImpressEvents] {

import ProfileClickFetcher.\_

override type RawSignalType = ProfileClickImpressEvent

override val name: String = this.getClass.getCanonicalName

override val statsReceiver: StatsReceiver = stats.scope(name)

override val stratoColumnPath: String = stratoPath

override val stratoView: Unit = None

override protected val keyConv: Conv[(UserId, Long)] = Conv.ofType

override protected val viewConv: Conv[Unit] = Conv.ofType

override protected val valueConv: Conv[RecentProfileClickImpressEvents] =

ScroogeConv.fromStruct[RecentProfileClickImpressEvents]

override protected def toStratoKey(userId: UserId): (UserId, Long) = (userId, defaultVersion)

override protected def toRawSignals(

stratoValue: RecentProfileClickImpressEvents

): Seq[ProfileClickImpressEvent] = {

stratoValue.events

}

override def process(

query: Query,

rawSignals: Future[Option[Seq[ProfileClickImpressEvent]]]

): Future[Option[Seq[Signal]]] = {

rawSignals.map { events =>

events

.map { clicks =>

clicks

.filter(dwelltimeFilter(\_, query.signalType))

.map(signalFromProfileClick(\_, query.signalType))

.sortBy(-\_.timestamp)

.take(query.maxResults.getOrElse(Int.MaxValue))

}

}

}

}

object ProfileClickFetcher {

val stratoPath = "recommendations/twistly/userRecentProfileClickImpress"

private val defaultVersion = 0L

private val sec2millis: Int => Long = i => i \* 1000L

private val minDwellTimeMap: Map[SignalType, Long] = Map(

SignalType.GoodProfileClick -> sec2millis(10),

SignalType.GoodProfileClick20s -> sec2millis(20),

SignalType.GoodProfileClick30s -> sec2millis(30),

SignalType.GoodProfileClickFiltered -> sec2millis(10),

SignalType.GoodProfileClick20sFiltered -> sec2millis(20),

SignalType.GoodProfileClick30sFiltered -> sec2millis(30),

)

def signalFromProfileClick(

profileClickImpressEvent: ProfileClickImpressEvent,

signalType: SignalType

): Signal = {

Signal(

signalType,

profileClickImpressEvent.engagedAt,

Some(InternalId.UserId(profileClickImpressEvent.entityId))

)

}

def dwelltimeFilter(

profileClickImpressEvent: ProfileClickImpressEvent,

signalType: SignalType

): Boolean = {

val goodClickDwellTime = minDwellTimeMap(signalType)

profileClickImpressEvent.clickImpressEventMetadata.totalDwellTime >= goodClickDwellTime

}

}