package com.twitter.timelineranker.visibility

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.timelineranker.core.FollowGraphData

import com.twitter.timelineranker.core.FollowGraphDataFuture

import com.twitter.timelines.clients.socialgraph.ScopedSocialGraphClientFactory

import com.twitter.timelines.model.\_

import com.twitter.timelines.util.FailOpenHandler

import com.twitter.timelines.util.stats.\_

import com.twitter.timelines.visibility.\_

import com.twitter.util.Future

object SgsFollowGraphDataProvider {

val EmptyUserIdsSet: Set[UserId] = Set.empty[UserId]

val EmptyUserIdsSetFuture: Future[Set[UserId]] = Future.value(EmptyUserIdsSet)

val EmptyUserIdsSeq: Seq[UserId] = Seq.empty[UserId]

val EmptyUserIdsSeqFuture: Future[Seq[UserId]] = Future.value(EmptyUserIdsSeq)

val EmptyVisibilityProfiles: Map[UserId, VisibilityProfile] = Map.empty[UserId, VisibilityProfile]

val EmptyVisibilityProfilesFuture: Future[Map[UserId, VisibilityProfile]] =

Future.value(EmptyVisibilityProfiles)

}

object SgsFollowGraphDataFields extends Enumeration {

val FollowedUserIds: Value = Value

val MutuallyFollowingUserIds: Value = Value

val MutedUserIds: Value = Value

val RetweetsMutedUserIds: Value = Value

val None: ValueSet = SgsFollowGraphDataFields.ValueSet()

def throwIfInvalid(fields: SgsFollowGraphDataFields.ValueSet): Unit = {

if (fields.contains(MutuallyFollowingUserIds) && !fields.contains(FollowedUserIds)) {

throw new IllegalArgumentException(

"MutuallyFollowingUserIds field requires FollowedUserIds field to be defined."

)

}

}

}

/\*\*

\* Provides information on the follow graph of a given user.

\*/

class SgsFollowGraphDataProvider(

socialGraphClientFactory: ScopedSocialGraphClientFactory,

visibilityProfileHydratorFactory: VisibilityProfileHydratorFactory,

fieldsToFetch: SgsFollowGraphDataFields.ValueSet,

scope: RequestScope,

statsReceiver: StatsReceiver)

extends FollowGraphDataProvider

with RequestStats {

SgsFollowGraphDataFields.throwIfInvalid(fieldsToFetch)

private[this] val stats = scope.stats("followGraphDataProvider", statsReceiver)

private[this] val scopedStatsReceiver = stats.scopedStatsReceiver

private[this] val followingScope = scopedStatsReceiver.scope("following")

private[this] val followingLatencyStat = followingScope.stat(LatencyMs)

private[this] val followingSizeStat = followingScope.stat(Size)

private[this] val followingTruncatedCounter = followingScope.counter("numTruncated")

private[this] val mutuallyFollowingScope = scopedStatsReceiver.scope("mutuallyFollowing")

private[this] val mutuallyFollowingLatencyStat = mutuallyFollowingScope.stat(LatencyMs)

private[this] val mutuallyFollowingSizeStat = mutuallyFollowingScope.stat(Size)

private[this] val visibilityScope = scopedStatsReceiver.scope("visibility")

private[this] val visibilityLatencyStat = visibilityScope.stat(LatencyMs)

private[this] val mutedStat = visibilityScope.stat("muted")

private[this] val retweetsMutedStat = visibilityScope.stat("retweetsMuted")

private[this] val socialGraphClient = socialGraphClientFactory.scope(scope)

private[this] val visibilityProfileHydrator =

createVisibilityProfileHydrator(visibilityProfileHydratorFactory, scope, fieldsToFetch)

private[this] val failOpenScope = scopedStatsReceiver.scope("failOpen")

private[this] val mutuallyFollowingHandler =

new FailOpenHandler(failOpenScope, "mutuallyFollowing")

private[this] val obtainVisibilityProfiles = fieldsToFetch.contains(

SgsFollowGraphDataFields.MutedUserIds

) || fieldsToFetch.contains(SgsFollowGraphDataFields.RetweetsMutedUserIds)

/\*\*

\* Gets follow graph data for the given user.

\*

\* @param userId user whose follow graph details are to be obtained.

\* @param maxFollowingCount Maximum number of followed user IDs to fetch.

\* If the given user follows more than these many users,

\* then the most recent maxFollowingCount users are returned.

\*/

def get(

userId: UserId,

maxFollowingCount: Int

): Future[FollowGraphData] = {

getAsync(

userId,

maxFollowingCount

).get()

}

def getAsync(

userId: UserId,

maxFollowingCount: Int

): FollowGraphDataFuture = {

stats.statRequest()

val followedUserIdsFuture =

if (fieldsToFetch.contains(SgsFollowGraphDataFields.FollowedUserIds)) {

getFollowing(userId, maxFollowingCount)

} else {

SgsFollowGraphDataProvider.EmptyUserIdsSeqFuture

}

val mutuallyFollowingUserIdsFuture =

if (fieldsToFetch.contains(SgsFollowGraphDataFields.MutuallyFollowingUserIds)) {

followedUserIdsFuture.flatMap { followedUserIds =>

getMutuallyFollowingUserIds(userId, followedUserIds)

}

} else {

SgsFollowGraphDataProvider.EmptyUserIdsSetFuture

}

val visibilityProfilesFuture = if (obtainVisibilityProfiles) {

followedUserIdsFuture.flatMap { followedUserIds =>

getVisibilityProfiles(userId, followedUserIds)

}

} else {

SgsFollowGraphDataProvider.EmptyVisibilityProfilesFuture

}

val mutedUserIdsFuture = if (fieldsToFetch.contains(SgsFollowGraphDataFields.MutedUserIds)) {

getMutedUsers(visibilityProfilesFuture).map { mutedUserIds =>

mutedStat.add(mutedUserIds.size)

mutedUserIds

}

} else {

SgsFollowGraphDataProvider.EmptyUserIdsSetFuture

}

val retweetsMutedUserIdsFuture =

if (fieldsToFetch.contains(SgsFollowGraphDataFields.RetweetsMutedUserIds)) {

getRetweetsMutedUsers(visibilityProfilesFuture).map { retweetsMutedUserIds =>

retweetsMutedStat.add(retweetsMutedUserIds.size)

retweetsMutedUserIds

}

} else {

SgsFollowGraphDataProvider.EmptyUserIdsSetFuture

}

FollowGraphDataFuture(

userId,

followedUserIdsFuture,

mutuallyFollowingUserIdsFuture,

mutedUserIdsFuture,

retweetsMutedUserIdsFuture

)

}

private[this] def getVisibilityProfiles(

userId: UserId,

followingIds: Seq[UserId]

): Future[Map[UserId, VisibilityProfile]] = {

Stat.timeFuture(visibilityLatencyStat) {

visibilityProfileHydrator(Some(userId), Future.value(followingIds.toSeq))

}

}

def getFollowing(userId: UserId, maxFollowingCount: Int): Future[Seq[UserId]] = {

Stat.timeFuture(followingLatencyStat) {

// We fetch 1 more than the limit so that we can decide if we ended up

// truncating the followings.

val followingIdsFuture = socialGraphClient.getFollowing(userId, Some(maxFollowingCount + 1))

followingIdsFuture.map { followingIds =>

followingSizeStat.add(followingIds.length)

if (followingIds.length > maxFollowingCount) {

followingTruncatedCounter.incr()

followingIds.take(maxFollowingCount)

} else {

followingIds

}

}

}

}

def getMutuallyFollowingUserIds(

userId: UserId,

followingIds: Seq[UserId]

): Future[Set[UserId]] = {

Stat.timeFuture(mutuallyFollowingLatencyStat) {

mutuallyFollowingHandler {

val mutuallyFollowingIdsFuture =

socialGraphClient.getFollowOverlap(followingIds.toSeq, userId)

mutuallyFollowingIdsFuture.map { mutuallyFollowingIds =>

mutuallyFollowingSizeStat.add(mutuallyFollowingIds.size)

}

mutuallyFollowingIdsFuture

} { e: Throwable => SgsFollowGraphDataProvider.EmptyUserIdsSetFuture }

}

}

private[this] def getRetweetsMutedUsers(

visibilityProfilesFuture: Future[Map[UserId, VisibilityProfile]]

): Future[Set[UserId]] = {

// If the hydrator is not able to fetch retweets-muted status, we default to true.

getUsersMatchingVisibilityPredicate(

visibilityProfilesFuture,

(visibilityProfile: VisibilityProfile) => visibilityProfile.areRetweetsMuted.getOrElse(true)

)

}

private[this] def getMutedUsers(

visibilityProfilesFuture: Future[Map[UserId, VisibilityProfile]]

): Future[Set[UserId]] = {

// If the hydrator is not able to fetch muted status, we default to true.

getUsersMatchingVisibilityPredicate(

visibilityProfilesFuture,

(visibilityProfile: VisibilityProfile) => visibilityProfile.isMuted.getOrElse(true)

)

}

private[this] def getUsersMatchingVisibilityPredicate(

visibilityProfilesFuture: Future[Map[UserId, VisibilityProfile]],

predicate: (VisibilityProfile => Boolean)

): Future[Set[UserId]] = {

visibilityProfilesFuture.map { visibilityProfiles =>

visibilityProfiles

.filter {

case (\_, visibilityProfile) =>

predicate(visibilityProfile)

}

.collect { case (userId, \_) => userId }

.toSet

}

}

private[this] def createVisibilityProfileHydrator(

factory: VisibilityProfileHydratorFactory,

scope: RequestScope,

fieldsToFetch: SgsFollowGraphDataFields.ValueSet

): VisibilityProfileHydrator = {

val hydrationProfileRequest = HydrationProfileRequest(

getMuted = fieldsToFetch.contains(SgsFollowGraphDataFields.MutedUserIds),

getRetweetsMuted = fieldsToFetch.contains(SgsFollowGraphDataFields.RetweetsMutedUserIds)

)

factory(hydrationProfileRequest, scope)

}

}

class ScopedSgsFollowGraphDataProviderFactory(

socialGraphClientFactory: ScopedSocialGraphClientFactory,

visibilityProfileHydratorFactory: VisibilityProfileHydratorFactory,

fieldsToFetch: SgsFollowGraphDataFields.ValueSet,

statsReceiver: StatsReceiver)

extends ScopedFactory[SgsFollowGraphDataProvider] {

override def scope(scope: RequestScope): SgsFollowGraphDataProvider = {

new SgsFollowGraphDataProvider(

socialGraphClientFactory,

visibilityProfileHydratorFactory,

fieldsToFetch,

scope,

statsReceiver

)

}

}