package com.twitter.tweetypie

package federated

package prefetcheddata

import com.twitter.consumer\_privacy.mention\_controls.thriftscala.UnmentionInfo

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.gizmoduck.thriftscala.LookupContext

import com.twitter.gizmoduck.thriftscala.QueryFields

import com.twitter.gizmoduck.thriftscala.UserResult

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.stitch.compat.LegacySeqGroup

import com.twitter.stitch.SeqGroup

import com.twitter.stitch.Stitch

import com.twitter.strato.graphql.thriftscala.CacheMissStrategy

import com.twitter.strato.graphql.thriftscala.PrefetchedData

import com.twitter.strato.graphql.thriftscala.TweetResult

import com.twitter.tweetypie.backends.Gizmoduck

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.util.Throwables

import com.twitter.vibes.thriftscala.VibeV2

import com.twitter.weaverbird.common.GetRequestContext

import com.twitter.weaverbird.common.PerTOOAppCallerStats

import com.twitter.weaverbird.common.RequestContext

import com.twitter.weaverbird.converters.tweet.WeaverbirdEntitySetMutations

import com.twitter.weaverbird.converters.tweet.WeaverbirdTweetMutations

import com.twitter.weaverbird.hydrators.\_

import com.twitter.weaverbird.mappers.ApiTweetPrefetchedMapper

import com.twitter.weaverbird.repositories.UserRepository

import com.twitter.weaverbird.converters.common.EntityRenderingOptions

private[federated] final case class PrefetchedDataRequest(

tweet: Tweet,

sourceTweet: Option[Tweet],

quotedTweet: Option[Tweet],

unmentionInfo: Option[UnmentionInfo] = None,

vibe: Option[VibeV2] = None,

safetyLevel: SafetyLevel,

requestContext: RequestContext)

private[federated] final case class PrefetchedDataResponse(value: PrefetchedData)

private[federated] object PrefetchedDataResponse {

// For NotFound, there is no subsequent result or quoted\_tweet\_results field, so both

// settings are false here. These deciders will be removed post migration.

private[this] val prefetchedMapper = new ApiTweetPrefetchedMapper(

skipTweetResultPrefetchItem = () => false

)

def notFound(tweetId: Long): PrefetchedDataResponse =

PrefetchedDataResponse(

value = prefetchedMapper.getPrefetchedData(

tweetId = tweetId,

apiTweet = None,

tweetResult = None

)

)

}

private[federated] object PrefetchedDataRepository {

def apply(

thriftTweetToApiTweet: ThriftTweetToApiTweet,

prefetchedMapper: ApiTweetPrefetchedMapper,

statsReceiver: StatsReceiver,

): PrefetchedDataRequest => Stitch[PrefetchedDataResponse] =

(request: PrefetchedDataRequest) => {

val thriftTweetToApiTweetRequest = ThriftTweetToApiTweetRequest(

tweet = request.tweet,

sourceTweet = request.sourceTweet,

quotedTweet = request.quotedTweet,

// For Tweet writes, filteredReason will always be None.

filteredReason = None,

safetyLevel = request.safetyLevel,

requestContext = request.requestContext,

entityRenderingOptions = EntityRenderingOptions()

)

val successCounter = statsReceiver.counter("success")

val failuresCounter = statsReceiver.counter("failures")

val failuresScope = statsReceiver.scope("failures")

thriftTweetToApiTweet

.arrow(thriftTweetToApiTweetRequest)

.onSuccess(\_ => successCounter.incr())

.onFailure { t =>

failuresCounter.incr()

failuresScope.counter(Throwables.mkString(t): \_\*).incr()

}

.map((resp: ThriftTweetToApiTweetResponse) => {

val prefetchedData: PrefetchedData = prefetchedMapper.getPrefetchedData(

tweetId = request.tweet.id,

apiTweet = Some(resp.apiTweet),

// since ApiTweet was hydrate, we can fabricate a TweetResult.Tweet

tweetResult = Some(TweetResult.Tweet(request.tweet.id)),

unmentionInfo = request.unmentionInfo,

editControl = request.tweet.editControl,

previousCounts = request.tweet.previousCounts,

vibe = request.vibe,

editPerspective = request.tweet.editPerspective,

noteTweet = request.tweet.noteTweet

)

// Notify GraphQL API to not attempt hydration for missing

// ApiTweet/TweetResult fields. This is only needed on the

// Tweet write path since the newly created Tweet may not

// be fully persisted yet in tbird Manhattan.

val shortCircuitedPrefetchedData = prefetchedData.copy(

onCacheMiss = CacheMissStrategy.ShortCircuitExisting

)

PrefetchedDataResponse(shortCircuitedPrefetchedData)

})

}

}

private[federated] object PrefetchedDataRepositoryBuilder {

def apply(

getUserResultsById: Gizmoduck.GetById,

statsReceiver: StatsReceiver

): PrefetchedDataRequest => Stitch[PrefetchedDataResponse] = {

val repoStats = statsReceiver.scope("repositories")

case class GetUserResultById(

queryFields: Set[QueryFields],

lookupContext: LookupContext,

) extends SeqGroup[UserId, UserResult] {

override def run(keys: Seq[UserId]): Future[Seq[Try[UserResult]]] =

LegacySeqGroup.liftToSeqTry(getUserResultsById((lookupContext, keys, queryFields)))

override def maxSize: Int = 100

}

val stitchGetUserResultById: UserRepository.GetUserResultById =

(userId: UserId, queryFields: Set[QueryFields], lookupContext: LookupContext) =>

Stitch.call(userId, GetUserResultById(queryFields, lookupContext))

val userRepository = new UserRepository(stitchGetUserResultById, repoStats)

// Note, this is weaverbird.common.GetRequestContext

val getRequestContext = new GetRequestContext()

// TwiggyUserHydrator is needed to hydrate TwiggyUsers for CWC and misc. logic

val twiggyUserHydrator = new TwiggyUserHydrator(userRepository, getRequestContext)

val weaverbirdMutations = new WeaverbirdTweetMutations(

new WeaverbirdEntitySetMutations(

new PerTOOAppCallerStats(statsReceiver, getRequestContext)

)

)

val prefetchedMapper = new ApiTweetPrefetchedMapper(

// do not skip this in mutation path as we depends on it

skipTweetResultPrefetchItem = () => false

)

val thriftTweetToApiTweet: ThriftTweetToApiTweet =

new FoundThriftTweetToApiTweet(

statsReceiver,

twiggyUserHydrator,

weaverbirdMutations

)

PrefetchedDataRepository(

thriftTweetToApiTweet,

prefetchedMapper,

repoStats.scope("prefetched\_data\_repo")

)

}

}