package com.twitter.tweetypie

package federated

package warmups

import com.twitter.context.TwitterContext

import com.twitter.context.thriftscala.Viewer

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

import com.twitter.stitch.Stitch

import com.twitter.strato.access.Access

import com.twitter.strato.access.Access.AccessToken

import com.twitter.strato.access.Access.AuthenticatedTwitterUserId

import com.twitter.strato.access.Access.AuthenticatedTwitterUserNotSuspended

import com.twitter.strato.access.Access.TwitterUserId

import com.twitter.strato.access.Access.TwitterUserNotSuspended

import com.twitter.strato.catalog.Ops

import com.twitter.strato.client.StaticClient

import com.twitter.strato.context.StratoContext

import com.twitter.strato.opcontext.DarkRequest

import com.twitter.strato.opcontext.OpContext

import com.twitter.strato.test.config.bouncer.TestPrincipals

import com.twitter.strato.thrift.ScroogeConvImplicits.\_

import com.twitter.tweetypie.federated.columns.CreateRetweetColumn

import com.twitter.tweetypie.federated.columns.CreateTweetColumn

import com.twitter.tweetypie.federated.columns.DeleteTweetColumn

import com.twitter.tweetypie.federated.columns.UnretweetColumn

import com.twitter.tweetypie.service.WarmupQueriesSettings

import com.twitter.tweetypie.thriftscala.graphql.\_

import com.twitter.util.logging.Logger

import com.twitter.util.Future

import com.twitter.util.Stopwatch

object StratoCatalogWarmups {

private[this] val log = Logger(getClass)

// Performs warmup queries, failing after 30 seconds

def warmup(

warmupSettings: WarmupQueriesSettings,

catalog: PartialFunction[String, Ops]

): Future[Unit] = {

val elapsed = Stopwatch.start()

// note: we need to supply bouncer principals here, because the

// columns are gated by a bouncer policy

Access

.withPrincipals(WarmupPrincipals) {

StratoContext.withOpContext(WarmupOpContext) {

TwitterContext.let(viewer = WarmupViewer) {

warmupSettings.clientId.asCurrent {

Stitch.run(executeDarkly(catalog))

}

}

}

}

.onSuccess { \_ => log.info("warmup completed in %s".format(elapsed())) }

.onFailure { t => log.error("could not complete warmup queries before startup.", t) }

}

private val WarmupTwitterUserId = 0L

private val WarmupPrincipals = Set(

TestPrincipals.normalStratoBouncerAccessPrincipal,

AuthenticatedTwitterUserId(WarmupTwitterUserId),

TwitterUserId(WarmupTwitterUserId),

TwitterUserNotSuspended,

AuthenticatedTwitterUserNotSuspended,

AccessToken(isWritable = true)

)

private[this] val RwebClientId = 0L

private[this] val WarmupViewer = Viewer(

userId = Some(WarmupTwitterUserId),

authenticatedUserId = Some(WarmupTwitterUserId),

clientApplicationId = Some(RwebClientId),

)

private[this] val WarmupOpContext =

OpContext

.safetyLevel(SafetyLevel.TweetWritesApi.name)

.copy(darkRequest = Some(DarkRequest()))

.toThrift()

private[this] val EllenOscarSelfie = 440322224407314432L

private[this] val TwitterContext: TwitterContext =

com.twitter.context.TwitterContext(com.twitter.tweetypie.TwitterContextPermit)

private[this] def executeDarkly(catalog: PartialFunction[String, Ops]): Stitch[Unit] = {

val stratoClient = new StaticClient(catalog)

val tweetCreator =

stratoClient.executer[CreateTweetRequest, CreateTweetResponseWithSubqueryPrefetchItems](

CreateTweetColumn.Path)

val tweetDeletor =

stratoClient

.executer[DeleteTweetRequest, DeleteTweetResponseWithSubqueryPrefetchItems](

DeleteTweetColumn.Path)

val retweetCreator =

stratoClient

.executer[CreateRetweetRequest, CreateRetweetResponseWithSubqueryPrefetchItems](

CreateRetweetColumn.Path)

val unretweetor =

stratoClient

.executer[UnretweetRequest, UnretweetResponseWithSubqueryPrefetchItems](

UnretweetColumn.Path)

val stitchCreateTweet =

tweetCreator

.execute(CreateTweetRequest("getting warmer"))

.onSuccess(\_ => log.info(s"${CreateTweetColumn.Path} warmup success"))

.onFailure(e => log.info(s"${CreateTweetColumn.Path} warmup fail: $e"))

val stitchDeleteTweet =

tweetDeletor

.execute(DeleteTweetRequest(-1L))

.onSuccess(\_ => log.info(s"${DeleteTweetColumn.Path} warmup success"))

.onFailure(e => log.info(s"${DeleteTweetColumn.Path} warmup fail: $e"))

val stitchCreateRetweet =

retweetCreator

.execute(CreateRetweetRequest(EllenOscarSelfie))

.onSuccess(\_ => log.info(s"${CreateRetweetColumn.Path} warmup success"))

.onFailure(e => log.info(s"${CreateRetweetColumn.Path} warmup fail: $e"))

val stitchUnretweet =

unretweetor

.execute(UnretweetRequest(EllenOscarSelfie))

.onSuccess(\_ => log.info(s"${UnretweetColumn.Path} warmup success"))

.onFailure(e => log.info(s"${UnretweetColumn.Path} warmup fail: $e"))

Stitch

.join(

stitchCreateTweet,

stitchDeleteTweet,

stitchCreateRetweet,

stitchUnretweet,

).unit

}

}