package com.twitter.timelineranker.server

import com.twitter.concurrent.AsyncSemaphore

import com.twitter.finagle.Filter

import com.twitter.finagle.ServiceFactory

import com.twitter.finagle.thrift.filter.ThriftForwardingWarmUpFilter

import com.twitter.finagle.thrift.ClientIdRequiredFilter

import com.twitter.timelineranker.config.RuntimeConfiguration

import com.twitter.timelineranker.config.TimelineRankerConstants

import com.twitter.timelineranker.decider.DeciderKey

import com.twitter.timelineranker.entity\_tweets.EntityTweetsRepositoryBuilder

import com.twitter.timelineranker.observe.DebugObserverBuilder

import com.twitter.timelineranker.parameters.ConfigBuilder

import com.twitter.timelineranker.parameters.util.RecapQueryParamInitializer

import com.twitter.timelineranker.recap\_author.RecapAuthorRepositoryBuilder

import com.twitter.timelineranker.recap\_hydration.RecapHydrationRepositoryBuilder

import com.twitter.timelineranker.in\_network\_tweets.InNetworkTweetRepositoryBuilder

import com.twitter.timelineranker.repository.\_

import com.twitter.timelineranker.thriftscala.TimelineRanker$FinagleService

import com.twitter.timelineranker.uteg\_liked\_by\_tweets.UtegLikedByTweetsRepositoryBuilder

import com.twitter.timelines.filter.DarkTrafficFilterBuilder

import com.twitter.timelines.observe.ServiceObserver

import com.twitter.timelines.util.DeciderableRequestSemaphoreFilter

import org.apache.thrift.protocol.TBinaryProtocol

import org.apache.thrift.protocol.TCompactProtocol

import org.apache.thrift.protocol.TProtocolFactory

class TimelineRankerBuilder(config: RuntimeConfiguration) {

private[this] val underlyingClients = config.underlyingClients

private[this] val configBuilder =

new ConfigBuilder(config.deciderGateBuilder, config.statsReceiver)

private[this] val debugObserverBuilder = new DebugObserverBuilder(config.whitelist)

private[this] val serviceObserver = new ServiceObserver(config.statsReceiver)

private[this] val routingRepository = RoutingTimelineRepositoryBuilder(config, configBuilder)

private[this] val inNetworkTweetRepository =

new InNetworkTweetRepositoryBuilder(config, configBuilder).apply()

private[this] val recapHydrationRepository =

new RecapHydrationRepositoryBuilder(config, configBuilder).apply()

private[this] val recapAuthorRepository = new RecapAuthorRepositoryBuilder(config).apply()

private[this] val entityTweetsRepository =

new EntityTweetsRepositoryBuilder(config, configBuilder).apply()

private[this] val utegLikedByTweetsRepository =

new UtegLikedByTweetsRepositoryBuilder(config, configBuilder).apply()

private[this] val queryParamInitializer = new RecapQueryParamInitializer(

config = configBuilder.rootConfig,

runtimeConfig = config

)

val timelineRanker: TimelineRanker = new TimelineRanker(

routingRepository = routingRepository,

inNetworkTweetRepository = inNetworkTweetRepository,

recapHydrationRepository = recapHydrationRepository,

recapAuthorRepository = recapAuthorRepository,

entityTweetsRepository = entityTweetsRepository,

utegLikedByTweetsRepository = utegLikedByTweetsRepository,

serviceObserver = serviceObserver,

abdecider = Some(config.abdecider),

clientRequestAuthorizer = config.clientRequestAuthorizer,

debugObserver = debugObserverBuilder.observer,

queryParamInitializer = queryParamInitializer,

statsReceiver = config.statsReceiver

)

private[this] def mkServiceFactory(

protocolFactory: TProtocolFactory

): ServiceFactory[Array[Byte], Array[Byte]] = {

val clientIdFilter = new ClientIdRequiredFilter[Array[Byte], Array[Byte]](

config.statsReceiver.scope("service").scope("filter")

)

// Limits the total number of concurrent requests handled by the TimelineRanker

val maxConcurrencyFilter = {

val asyncSemaphore = new AsyncSemaphore(

initialPermits = config.maxConcurrency,

maxWaiters = 0

)

val enableLimiting = config.deciderGateBuilder.linearGate(

DeciderKey.EnableMaxConcurrencyLimiting

)

new DeciderableRequestSemaphoreFilter(

enableFilter = enableLimiting,

semaphore = asyncSemaphore,

statsReceiver = config.statsReceiver

)

}

// Forwards a percentage of traffic via the DarkTrafficFilter to the TimelineRanker proxy, which in turn can be

// used to forward dark traffic to staged instances

val darkTrafficFilter = DarkTrafficFilterBuilder(

config.deciderGateBuilder,

DeciderKey.EnableRoutingToRankerDevProxy,

TimelineRankerConstants.ClientPrefix,

underlyingClients.darkTrafficProxy,

config.statsReceiver

)

val warmupForwardingFilter = if (config.isProd) {

new ThriftForwardingWarmUpFilter(

Warmup.WarmupForwardingTime,

underlyingClients.timelineRankerForwardingClient.service,

config.statsReceiver.scope("warmupForwardingFilter"),

isBypassClient = { \_.name.startsWith("timelineranker.") }

)

} else Filter.identity[Array[Byte], Array[Byte]]

val serviceFilterChain = clientIdFilter

.andThen(maxConcurrencyFilter)

.andThen(warmupForwardingFilter)

.andThen(darkTrafficFilter)

.andThen(serviceObserver.thriftExceptionFilter)

val finagleService =

new TimelineRanker$FinagleService(timelineRanker, protocolFactory)

ServiceFactory.const(serviceFilterChain andThen finagleService)

}

val serviceFactory: ServiceFactory[Array[Byte], Array[Byte]] =

mkServiceFactory(new TBinaryProtocol.Factory())

val compactProtocolServiceFactory: ServiceFactory[Array[Byte], Array[Byte]] =

mkServiceFactory(new TCompactProtocol.Factory())

}