package com.twitter.cr\_mixer.source\_signal

import com.twitter.core\_workflows.user\_model.thriftscala.UserState

import com.twitter.cr\_mixer.config.TimeoutConfig

import com.twitter.cr\_mixer.source\_signal.SourceFetcher.FetcherQuery

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

import com.twitter.timelines.configapi.Params

import com.twitter.cr\_mixer.thriftscala.{Product => TProduct}

import com.twitter.finagle.GlobalRequestTimeoutException

import com.twitter.finagle.mux.ClientDiscardedRequestException

import com.twitter.finagle.mux.ServerApplicationError

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

import com.twitter.util.TimeoutException

import org.apache.thrift.TApplicationException

import com.twitter.util.logging.Logging

/\*\*

\* A SourceFetcher is a trait which, given a [[FetcherQuery]], returns [[ResultType]]

\* The main purposes of a SourceFetcher is to provide a consistent interface for source fetch

\* logic, and provides default functions, including:

\* - Identification

\* - Observability

\* - Timeout settings

\* - Exception Handling

\*/

trait SourceFetcher[ResultType] extends ReadableStore[FetcherQuery, ResultType] with Logging {

protected final val timer = com.twitter.finagle.util.DefaultTimer

protected final def identifier: String = this.getClass.getSimpleName

protected def stats: StatsReceiver

protected def timeoutConfig: TimeoutConfig

/\*\*\*

\* Use FeatureSwitch to decide if a specific source is enabled.

\*/

def isEnabled(query: FetcherQuery): Boolean

/\*\*\*

\* This function fetches the raw sources and process them.

\* Custom stats tracking can be added depending on the type of ResultType

\*/

def fetchAndProcess(

query: FetcherQuery,

): Future[Option[ResultType]]

/\*\*\*

\* Side-effect function to track stats for signal fetching and processing.

\*/

def trackStats(

query: FetcherQuery

)(

func: => Future[Option[ResultType]]

): Future[Option[ResultType]]

/\*\*\*

\* This function is called by the top level class to fetch sources. It executes the pipeline to

\* fetch raw data, process and transform the sources. Exceptions, Stats, and timeout control are

\* handled here.

\*/

override def get(

query: FetcherQuery

): Future[Option[ResultType]] = {

val scopedStats = stats.scope(query.product.originalName)

if (isEnabled(query)) {

scopedStats.counter("gate\_enabled").incr()

trackStats(query)(fetchAndProcess(query))

.raiseWithin(timeoutConfig.signalFetchTimeout)(timer)

.onFailure { e =>

scopedStats.scope("exceptions").counter(e.getClass.getSimpleName).incr()

}

.rescue {

case \_: TimeoutException | \_: GlobalRequestTimeoutException | \_: TApplicationException |

\_: ClientDiscardedRequestException |

\_: ServerApplicationError // TApplicationException inside

=>

Future.None

case e =>

logger.info(e)

Future.None

}

} else {

scopedStats.counter("gate\_disabled").incr()

Future.None

}

}

}

object SourceFetcher {

/\*\*\*

\* Every SourceFetcher all share the same input: FetcherQuery

\*/

case class FetcherQuery(

userId: UserId,

product: TProduct,

userState: UserState,

params: Params)

}