package com.twitter.tweetypie

package handler

import com.twitter.tweetypie.thriftscala.Reply

import com.twitter.tweetypie.thriftscala.SelfThreadMetadata

import org.apache.thrift.protocol.TField

trait SelfThreadBuilder {

def requiredReplySourceFields: Set[TField] =

Set(

Tweet.CoreDataField, // for Reply and ConversationId

Tweet.SelfThreadMetadataField // for continuing existing self-threads

)

def build(authorUserId: UserId, replySourceTweet: Tweet): Option[SelfThreadMetadata]

}

/\*\*

\* SelfThreadBuilder is used to build metadata for self-threads (tweetstorms).

\*

\* This builder is invoked from ReplyBuilder on tweets that pass in a inReplyToStatusId and create

\* a Reply. The invocation is done inside ReplyBuilder as ReplyBuilder has already loaded the

\* "reply source tweet" which has all the information needed to determine the self-thread metadata.

\*

\* Note that Tweet.SelfThreadMetadata schema supports representing two types of self-threads:

\* 1. root self-thread : self-thread that begins alone and does not start with replying to another

\* tweet. This self-thread has a self-thread ID equal to the conversation ID.

\* 2. reply self-thread : self-thread that begins as a reply to another user's tweet.

\* This self-thread has a self-thread ID equal to the first tweet in the

\* current self-reply chain which will not equal the conversation ID.

\*

\* Currently only type #1 "root self-thread" is handled.

\*/

object SelfThreadBuilder {

def apply(stats: StatsReceiver): SelfThreadBuilder = {

// We want to keep open the possibility for differentiation between root

// self-threads (current functionality) and reply self-threads (possible

// future functionality).

val rootThreadStats = stats.scope("root\_thread")

// A tweet becomes a root of a self-thread only after the first self-reply

// is created. root\_thread/start is incr()d during the write-path of the

// self-reply tweet, when it is known that the first/root tweet has not

// yet been assigned a SelfThreadMetadata. The write-path of the second

// tweet does not add the SelfThreadMetadata to the first tweet - that

// happens asynchronously by the SelfThreadDaemon.

val rootThreadStartCounter = rootThreadStats.counter("start")

// root\_thread/continue provides visibility into the frequency of

// continuation tweets off leaf tweets in a tweet storm. Also incr()d in

// the special case of a reply to the root tweet, which does not yet have a

// SelfThreadMetadata(isLeaf=true).

val rootThreadContinueCounter = rootThreadStats.counter("continue")

// root\_thread/branch provides visibility into how frequently self-threads

// get branched - that is, when the author self-replies to a non-leaf tweet

// in an existing thread. Knowing the frequency of branching will help us

// determine the priority of accounting for branching in various

// tweet-delete use cases. Currently we do not fix up the root tweet's

// SelfThreadMetadata when its reply tweets are deleted.

val rootThreadBranchCounter = rootThreadStats.counter("branch")

def observeSelfThreadMetrics(replySourceSTM: Option[SelfThreadMetadata]): Unit = {

replySourceSTM match {

case Some(SelfThreadMetadata(\_, isLeaf)) =>

if (isLeaf) rootThreadContinueCounter.incr()

else rootThreadBranchCounter.incr()

case None =>

rootThreadStartCounter.incr()

}

}

new SelfThreadBuilder {

override def build(

authorUserId: UserId,

replySourceTweet: Tweet

): Option[SelfThreadMetadata] = {

// the "reply source tweet"'s author must match the current author

if (getUserId(replySourceTweet) == authorUserId) {

val replySourceSTM = getSelfThreadMetadata(replySourceTweet)

observeSelfThreadMetrics(replySourceSTM)

// determine if replySourceTweet stands alone (non-reply)

getReply(replySourceTweet) match {

case None | Some(Reply(None, \_, \_)) =>

// 'replySourceTweet' started a new self-thread that stands alone

// which happens when there's no Reply or the Reply does not have

// inReplyToStatusId (directed-at user)

// requiredReplySourceFields requires coreData and conversationId

// is required so this would have previously thrown an exception

// in ReplyBuilder if the read was partial

val convoId = replySourceTweet.coreData.get.conversationId.get

Some(SelfThreadMetadata(id = convoId, isLeaf = true))

case \_ =>

// 'replySourceTweet' was also a reply-to-tweet, so continue any

// self-thread by inheriting any SelfThreadMetadata it has

// (though always setting isLeaf to true)

replySourceSTM.map(\_.copy(isLeaf = true))

}

} else {

// Replying to a different user currently never creates a self-thread

// as all self-threads must start at the root (and match conversation

// ID).

//

// In the future replying to a different user \*might\* be part of a

// self-thread but we wouldn't mark it as such until the \*next\* tweet

// is created (at which time the self\_thread daemon goes back and

// marks the first tweet as in the self-thread.

None

}

}

}

}

}