package com.twitter.tweetypie

package hydrator

import com.twitter.stitch.NotFound

import com.twitter.stitch.Stitch

import com.twitter.tco\_util.DisplayUrl

import com.twitter.tco\_util.InvalidUrlException

import com.twitter.tco\_util.TcoSlug

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.repository.\_

import com.twitter.tweetypie.thriftscala.\_

import scala.util.control.NonFatal

object UrlEntitiesHydrator {

type Type = ValueHydrator[Seq[UrlEntity], TweetCtx]

def once(h: ValueHydrator[UrlEntity, TweetCtx]): Type =

TweetHydration.completeOnlyOnce(

queryFilter = queryFilter,

hydrationType = HydrationType.Urls,

hydrator = h.liftSeq

)

def queryFilter(opts: TweetQuery.Options): Boolean =

opts.include.tweetFields.contains(Tweet.UrlsField.id)

}

/\*\*

\* Hydrates UrlEntities. If there is a failure to hydrate an entity, the entity is left

\* unhydrated, so that we can try again later. The PartialEntityCleaner will remove

\* the partial entity before returning to clients.

\*/

object UrlEntityHydrator {

/\*\*

\* a function type that takes a shorten-url and an expanded-url, and generates a

\* "display url" (which isn't really a url). this may fail if the expanded-url

\* can't be parsed as a valid url, in which case None is returned.

\*/

type Truncator = (String, String) => Option[String]

val hydratedField: FieldByPath = fieldByPath(Tweet.UrlsField)

val log: Logger = Logger(getClass)

def apply(repo: UrlRepository.Type, stats: StatsReceiver): ValueHydrator[UrlEntity, TweetCtx] = {

val toDisplayUrl = truncator(stats)

ValueHydrator[UrlEntity, TweetCtx] { (curr, \_) =>

val slug = getTcoSlug(curr)

val result: Stitch[Option[Try[ExpandedUrl]]] = Stitch.collect(slug.map(repo(\_).liftToTry))

result.map {

case Some(Return(expandedUrl)) =>

ValueState.modified(update(curr, expandedUrl, toDisplayUrl))

case None =>

ValueState.unmodified(curr)

case Some(Throw(NotFound)) =>

// If the UrlEntity contains an invalid t.co slug that can't be resolved,

// leave the entity unhydrated, to be removed later by the PartialEntityCleaner.

// We don't consider this a partial because the input is invalid and is not

// expected to succeed.

ValueState.unmodified(curr)

case Some(Throw(\_)) =>

// On failure, use the t.co link as the expanded url so that it is still clickable,

// but also still flag the failure

ValueState.partial(

update(curr, ExpandedUrl(curr.url), toDisplayUrl),

hydratedField

)

}

}.onlyIf((curr, ctx) => !ctx.isRetweet && isUnhydrated(curr))

}

/\*\*

\* a UrlEntity needs hydration if the expanded url is either unset or set to the

\* shortened url .

\*/

def isUnhydrated(entity: UrlEntity): Boolean =

entity.expanded.isEmpty || hydrationFailed(entity)

/\*\*

\* Did the hydration of this URL entity fail?

\*/

def hydrationFailed(entity: UrlEntity): Boolean =

entity.expanded.contains(entity.url)

def update(entity: UrlEntity, expandedUrl: ExpandedUrl, toDisplayUrl: Truncator): UrlEntity =

entity.copy(

expanded = Some(expandedUrl.text),

display = toDisplayUrl(entity.url, expandedUrl.text)

)

def getTcoSlug(entity: UrlEntity): Option[UrlSlug] =

TcoSlug.unapply(entity.url).map(UrlSlug(\_))

def truncator(stats: StatsReceiver): Truncator = {

val truncationStats = stats.scope("truncations")

val truncationsCounter = truncationStats.counter("count")

val truncationExceptionsCounter = truncationStats.counter("exceptions")

(shortUrl, expandedUrl) =>

try {

truncationsCounter.incr()

Some(DisplayUrl(shortUrl, Some(expandedUrl), true))

} catch {

case NonFatal(ex) =>

truncationExceptionsCounter.incr()

truncationStats.counter(ex.getClass.getName).incr()

ex match {

case InvalidUrlException(\_) =>

log.warn(s"failed to truncate: `$shortUrl` / `$expandedUrl`")

case \_ =>

log.warn(s"failed to truncate: `$shortUrl` / `$expandedUrl`", ex)

}

None

}

}

}