package com.twitter.product\_mixer.core.functional\_component.candidate\_source.strato

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSourceWithExtractedFeatures

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidatesWithSourceFeatures

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Fetcher

/\*\*

\* A [[CandidateSource]] for getting Candidates from Strato where the

\* Strato column's View is [[StratoView]] and the Value is a [[StratoValue]]

\*

\* A [[stratoResultTransformer]] must be defined to convert the

\* [[StratoValue]] into a Seq of [[Candidate]]

\*

\* [[extractFeaturesFromStratoResult]] must be defined to extract a

\* [[FeatureMap]] from the [[StratoValue]]. If you don't need to do that,

\* use a [[StratoKeyViewFetcherSource]] instead.

\*

\* @tparam StratoKey the column's Key type

\* @tparam StratoView the column's View type

\* @tparam StratoValue the column's Value type

\*/

trait StratoKeyViewFetcherWithSourceFeaturesSource[StratoKey, StratoView, StratoValue, Candidate]

extends CandidateSourceWithExtractedFeatures[StratoKeyView[StratoKey, StratoView], Candidate] {

val fetcher: Fetcher[StratoKey, StratoView, StratoValue]

/\*\*

\* Transforms the value type returned by Strato into a Seq[Candidate].

\*

\* This might be as simple as `Seq(stratoResult)` if you're always returning a single candidate.

\*

\* Often, it just extracts a Seq from within a larger wrapper object.

\*

\* If there is global metadata that you need to include, see [[extractFeaturesFromStratoResult]]

\* below to put that into a Feature.

\*/

protected def stratoResultTransformer(

stratoKey: StratoKey,

stratoResult: StratoValue

): Seq[Candidate]

/\*\*

\* Transforms the value type returned by Strato into a FeatureMap.

\*

\* Override this to extract global metadata like cursors and place the results

\* into a Feature.

\*

\* For example, a cursor.

\*/

protected def extractFeaturesFromStratoResult(

stratoKey: StratoKey,

stratoResult: StratoValue

): FeatureMap

override def apply(

request: StratoKeyView[StratoKey, StratoView]

): Stitch[CandidatesWithSourceFeatures[Candidate]] = {

fetcher

.fetch(request.key, request.view)

.map { result =>

val candidates = result.v

.map((stratoResult: StratoValue) => stratoResultTransformer(request.key, stratoResult))

.getOrElse(Seq.empty)

val features = result.v

.map((stratoResult: StratoValue) =>

extractFeaturesFromStratoResult(request.key, stratoResult))

.getOrElse(FeatureMap.empty)

CandidatesWithSourceFeatures(candidates, features)

}.rescue(StratoErrCategorizer.CategorizeStratoException)

}

}