package com.twitter.product\_mixer.core.feature.featuremap.featurestorev1

import com.twitter.ml.api.DataRecord

import com.twitter.ml.featurestore.lib.EntityId

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

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

import com.twitter.product\_mixer.core.feature.featurestorev1.FeatureStoreV1CandidateFeature

import com.twitter.product\_mixer.core.feature.featurestorev1.FeatureStoreV1CandidateFeatureGroup

import com.twitter.product\_mixer.core.feature.featurestorev1.FeatureStoreV1QueryFeature

import com.twitter.product\_mixer.core.feature.featurestorev1.FeatureStoreV1QueryFeatureGroup

import com.twitter.product\_mixer.core.feature.featurestorev1.featurevalue.FeatureStoreV1ResponseFeature

import com.twitter.product\_mixer.core.model.common.UniversalNoun

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.util.Try

object FeatureStoreV1FeatureMap {

/\*\*

\* Implicitly add convenience accessors for FeatureStoreV1 features in [[FeatureMap]]. Note that

\* we cannot add these methods directly to [[FeatureMap]] because it would introduce a circular

\* dependency ([[PipelineQuery]] depends on [[FeatureMap]], and the methods below depend on

\* [[PipelineQuery]])

\*

\* @param featureMap the featureMap we are wrapping

\* @note The FeatureStoreV1Feature::defaultValue set on the BoundFeature is only used and set

\* during PredictionRecord to DataRecord conversion. Therefore, the default will not be set

\* on the PredictionRecord value if reading from it directly, and as such for convenience

\* the defaultValue is manually returned during retrieval from PredictionRecord.

\* @note the Value generic type on the methods below cannot be passed to

\* FeatureStoreV1QueryFeature's Value generic type. While this is actually the same type,

\* (note the explicit type cast back to Value), we must instead use an existential on

\* FeatureStoreV1QueryFeature since it is constructed with an existential for the Value

\* generic (see [[FeatureStoreV1QueryFeature]] and [[FeatureStoreV1CandidateFeature]])

\*/

implicit class FeatureStoreV1FeatureMapAccessors(private val featureMap: FeatureMap) {

def getFeatureStoreV1QueryFeature[Query <: PipelineQuery, Value](

feature: FeatureStoreV1QueryFeature[Query, \_ <: EntityId, Value]

): Value =

getOrElseFeatureStoreV1QueryFeature(

feature,

feature.defaultValue.getOrElse {

throw MissingFeatureException(feature)

})

def getFeatureStoreV1QueryFeatureTry[Query <: PipelineQuery, Value](

feature: FeatureStoreV1QueryFeature[Query, \_ <: EntityId, Value]

): Try[Value] =

Try(getFeatureStoreV1QueryFeature(feature))

def getOrElseFeatureStoreV1QueryFeature[Query <: PipelineQuery, Value](

feature: FeatureStoreV1QueryFeature[Query, \_ <: EntityId, Value],

default: => Value

): Value = {

/\*\*

\* FeatureStoreV1ResponseFeature should never be missing from the FeatureMap as FSv1 is

\* guaranteed to return a prediction record per feature store request. However, this may be

\* called on candidates that never hydrated FSv1 features. For example by

\* [[com.twitter.product\_mixer.component\_library.selector.sorter.featurestorev1.FeatureStoreV1FeatureValueSorter]]

\*/

val featureStoreV1FeatureValueOpt = featureMap.getTry(FeatureStoreV1ResponseFeature).toOption

val dataRecordValue: Option[Value] = featureStoreV1FeatureValueOpt.flatMap {

featureStoreV1FeatureValue =>

featureStoreV1FeatureValue.richDataRecord.getFeatureValueOpt(

feature.boundFeature.mlApiFeature)(feature.fromDataRecordValue)

}

dataRecordValue.getOrElse(default)

}

def getFeatureStoreV1CandidateFeature[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

Value

](

feature: FeatureStoreV1CandidateFeature[Query, Candidate, \_ <: EntityId, Value]

): Value =

getOrElseFeatureStoreV1CandidateFeature(

feature,

feature.defaultValue.getOrElse {

throw MissingFeatureException(feature)

})

def getFeatureStoreV1CandidateFeatureTry[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

Value

](

feature: FeatureStoreV1CandidateFeature[Query, Candidate, \_ <: EntityId, Value]

): Try[Value] =

Try(getFeatureStoreV1CandidateFeature(feature))

def getOrElseFeatureStoreV1CandidateFeature[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

Value

](

feature: FeatureStoreV1CandidateFeature[Query, Candidate, \_ <: EntityId, Value],

default: => Value

): Value = {

/\*\*

\* FeatureStoreV1ResponseFeature should never be missing from the FeatureMap as FSv1 is

\* guaranteed to return a prediction record per feature store request. However, this may be

\* called on candidates that never hydrated FSv1 features. For example by

\* [[com.twitter.product\_mixer.component\_library.selector.sorter.featurestorev1.FeatureStoreV1FeatureValueSorter]]

\*/

val featureStoreV1FeatureValueOpt = featureMap.getTry(FeatureStoreV1ResponseFeature).toOption

val dataRecordValue: Option[Value] = featureStoreV1FeatureValueOpt.flatMap {

featureStoreV1FeatureValue =>

featureStoreV1FeatureValue.richDataRecord.getFeatureValueOpt(

feature.boundFeature.mlApiFeature)(feature.fromDataRecordValue)

}

dataRecordValue.getOrElse(default)

}

/\*\*

\* Get queryFeatureGroup, which is store in the featureMap as a DataRecordInAFeature

\* It doesn't have the mlApiFeature as other regular FeatureStoreV1 features

\* Please refer to [[com.twitter.product\_mixer.core.feature.datarecord.DataRecordInAFeature]] scaladoc for more details

\*/

def getFeatureStoreV1QueryFeatureGroup[Query <: PipelineQuery](

featureGroup: FeatureStoreV1QueryFeatureGroup[Query, \_ <: EntityId]

): DataRecord =

getOrElseFeatureStoreV1QueryFeatureGroup(

featureGroup,

throw MissingFeatureException(featureGroup)

)

def getFeatureStoreV1CandidateFeatureGroupTry[Query <: PipelineQuery](

featureGroup: FeatureStoreV1QueryFeatureGroup[Query, \_ <: EntityId]

): Try[DataRecord] =

Try(getFeatureStoreV1QueryFeatureGroup(featureGroup))

def getOrElseFeatureStoreV1QueryFeatureGroup[Query <: PipelineQuery](

featureGroup: FeatureStoreV1QueryFeatureGroup[Query, \_ <: EntityId],

default: => DataRecord

): DataRecord = {

featureMap.getTry(featureGroup).toOption.getOrElse(default)

}

/\*\*

\* Get candidateFeatureGroup, which is store in the featureMap as a DataRecordInAFeature

\* It doesn't have the mlApiFeature as other regular FeatureStoreV1 features

\* Please refer to [[com.twitter.product\_mixer.core.feature.datarecord.DataRecordInAFeature]] scaladoc for more details

\*/

def getFeatureStoreV1CandidateFeatureGroup[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any]

](

featureGroup: FeatureStoreV1CandidateFeatureGroup[Query, Candidate, \_ <: EntityId]

): DataRecord =

getOrElseFeatureStoreV1CandidateFeatureGroup(

featureGroup,

throw MissingFeatureException(featureGroup)

)

def getFeatureStoreV1CandidateFeatureGroupTry[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any]

](

featureGroup: FeatureStoreV1CandidateFeatureGroup[Query, Candidate, \_ <: EntityId]

): Try[DataRecord] =

Try(getFeatureStoreV1CandidateFeatureGroup(featureGroup))

def getOrElseFeatureStoreV1CandidateFeatureGroup[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any]

](

featureGroup: FeatureStoreV1CandidateFeatureGroup[Query, Candidate, \_ <: EntityId],

default: => DataRecord

): DataRecord = {

featureMap.getTry(featureGroup).toOption.getOrElse(default)

}

def getOrElseFeatureStoreV1FeatureDataRecord(

default: => DataRecord

) = {

val featureStoreV1FeatureValueOpt = featureMap.getTry(FeatureStoreV1ResponseFeature).toOption

featureStoreV1FeatureValueOpt

.map { featureStoreV1FeatureValue =>

featureStoreV1FeatureValue.richDataRecord.getRecord

}.getOrElse(default)

}

}

}