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

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

import scala.collection.mutable

/\*\*

\* [[FeatureMapBuilder]] is a typesafe way (it checks types vs the [[Feature]]s on `.add`) to build a [[FeatureMap]].

\*

\* Throws a [[DuplicateFeatureException]] if you try to add the same [[Feature]] more than once.

\*

\* These builders are \_\_not\_\_ reusable.

\*/

class FeatureMapBuilder {

private val underlying = Map.newBuilder[Feature[\_, \_], Try[Any]]

private val keys = mutable.HashSet.empty[Feature[\_, \_]]

private var built = false

/\*\*

\* Add a [[Try]] of a [[Feature]] `value` to the map,

\* handling both the [[Return]] and [[Throw]] cases.

\*

\* Throws a [[DuplicateFeatureException]] if it's already present.

\*

\* @note If you have a [[Feature]] with a non-optional value type `Feature[\_, V]`

\* but have an `Option[V]` you can use [[Try.orThrow]] to convert the [[Option]]

\* to a [[Try]], which will store the successful or failed [[Feature]] in the map.

\*/

def add[V](feature: Feature[\_, V], value: Try[V]): FeatureMapBuilder = addTry(feature, value)

/\*\*

\* Add a successful [[Feature]] `value` to the map

\*

\* Throws a [[DuplicateFeatureException]] if it's already present.

\*

\* @note If you have a [[Feature]] with a non-optional value type `Feature[\_, V]`

\* but have an `Option[V]` you can use [[Option.get]] or [[Option.getOrElse]]

\* to convert the [[Option]] to extract the underlying value,

\* which will throw immediately if it's [[None]] or add the successful [[Feature]] in the map.

\*/

def add[V](feature: Feature[\_, V], value: V): FeatureMapBuilder =

addTry(feature, Return(value))

/\*\*

\* Add a failed [[Feature]] `value` to the map

\*

\* Throws a [[DuplicateFeatureException]] if it's already present.

\*/

def addFailure(feature: Feature[\_, \_], throwable: Throwable): FeatureMapBuilder =

addTry(feature, Throw(throwable))

/\*\*

\* [[add]] but for when the [[Feature]] types aren't known

\*

\* Add a [[Try]] of a [[Feature]] `value` to the map,

\* handling both the [[Return]] and [[Throw]] cases.

\*

\* Throws a [[DuplicateFeatureException]] if it's already present.

\*

\* @note If you have a [[Feature]] with a non-optional value type `Feature[\_, V]`

\* but have an `Option[V]` you can use [[Try.orThrow]] to convert the [[Option]]

\* to a [[Try]], which will store the successful or failed [[Feature]] in the map.

\*/

def addTry(feature: Feature[\_, \_], value: Try[\_]): FeatureMapBuilder = {

if (keys.contains(feature)) {

throw new DuplicateFeatureException(feature)

}

addWithoutValidation(feature, value)

}

/\*\*

\* [[addTry]] but without a [[DuplicateFeatureException]] check

\*

\* @note Only for use internally within [[FeatureMap.merge]]

\*/

private[featuremap] def addWithoutValidation(

feature: Feature[\_, \_],

value: Try[\_]

): FeatureMapBuilder = {

keys += feature

underlying += ((feature, value))

this

}

/\*\* Builds the FeatureMap \*/

def build(): FeatureMap = {

if (built) {

throw ReusedFeatureMapBuilderException

}

built = true

new FeatureMap(underlying.result())

}

}

object FeatureMapBuilder {

/\*\* Returns a new [[FeatureMapBuilder]] for making [[FeatureMap]]s \*/

def apply(): FeatureMapBuilder = new FeatureMapBuilder

}

class DuplicateFeatureException(feature: Feature[\_, \_])

extends UnsupportedOperationException(s"Feature $feature already exists in FeatureMap")

object ReusedFeatureMapBuilderException

extends UnsupportedOperationException(

"build() cannot be called more than once since FeatureMapBuilders are not reusable")