package com.twitter.tweetypie.core

import com.twitter.tweetypie.thriftscala.FieldByPath

import com.twitter.tweetypie.thriftscala.HydrationType

/\*\*

\* HydrationState is used to record whether a particular piece of data was modified as a result

\* of hydration, and/or if there was a failure to hydrate the data.

\*/

sealed trait HydrationState {

def isEmpty: Boolean

def modified: Boolean

def completedHydrations: Set[HydrationType] = Set.empty

def failedFields: Set[FieldByPath] = Set.empty

def cacheErrorEncountered: Boolean = false

def ++(that: HydrationState): HydrationState

}

object HydrationState {

/\*\*

\* Base `HydrationState`. It acts as an identity value when combined with any other

\* `HydrationState`.

\*/

case object Empty extends HydrationState {

def isEmpty = true

def modified = false

def ++(that: HydrationState): HydrationState = that

}

/\*\*

\* A `HydrationState` with metadata indicating a non-fatal hydration operation.

\*/

case class Success(

override val modified: Boolean = false,

override val completedHydrations: Set[HydrationType] = Set.empty,

override val failedFields: Set[FieldByPath] = Set.empty,

override val cacheErrorEncountered: Boolean = false)

extends HydrationState {

def isEmpty: Boolean = !modified && failedFields.isEmpty && !cacheErrorEncountered

def ++(that: HydrationState): HydrationState =

that match {

case Empty => this

case that: Success =>

HydrationState(

modified || that.modified,

completedHydrations ++ that.completedHydrations,

failedFields ++ that.failedFields,

cacheErrorEncountered || that.cacheErrorEncountered

)

}

/\*\*

\* An implementation of `copy` that avoids unnecessary allocations, by

\* using the constant `HydrationState.unmodified` and `HydrationState.modified`

\* values when possible.

\*/

def copy(

modified: Boolean = this.modified,

completedHydrations: Set[HydrationType] = this.completedHydrations,

failedFields: Set[FieldByPath] = this.failedFields,

cacheErrorEncountered: Boolean = this.cacheErrorEncountered

): HydrationState =

HydrationState(modified, completedHydrations, failedFields, cacheErrorEncountered)

}

val empty: HydrationState = Empty

val modified: HydrationState = Success(true)

def modified(completedHydration: HydrationType): HydrationState =

modified(Set(completedHydration))

def modified(completedHydrations: Set[HydrationType]): HydrationState =

Success(modified = true, completedHydrations = completedHydrations)

def partial(failedField: FieldByPath): HydrationState =

partial(Set(failedField))

def partial(failedFields: Set[FieldByPath]): HydrationState =

Success(modified = false, failedFields = failedFields)

def apply(

modified: Boolean,

completedHydrations: Set[HydrationType] = Set.empty,

failedFields: Set[FieldByPath] = Set.empty,

cacheErrorEncountered: Boolean = false

): HydrationState =

if (completedHydrations.nonEmpty || failedFields.nonEmpty || cacheErrorEncountered) {

Success(modified, completedHydrations, failedFields, cacheErrorEncountered)

} else if (modified) {

HydrationState.modified

} else {

HydrationState.empty

}

/\*\*

\* Creates a new HydrationState with modified set to true if `next` and `prev` are different,

\* or false if they are the same.

\*/

def delta[A](prev: A, next: A): HydrationState =

if (next != prev) modified else empty

/\*\*

\* Join a list of HydrationStates into a single HydrationState.

\*

\* Note: this could just be a reduce over the HydrationStates but that would allocate

\* \_N\_ HydrationStates. This approach also allows for shortcircuiting over the boolean

\* fields.

\*/

def join(states: HydrationState\*): HydrationState = {

val statesSet = states.toSet

HydrationState(

modified = states.exists(\_.modified),

completedHydrations = statesSet.flatMap(\_.completedHydrations),

failedFields = statesSet.flatMap(\_.failedFields),

cacheErrorEncountered = states.exists(\_.cacheErrorEncountered)

)

}

}