package com.twitter.tweetypie.core

import com.twitter.servo.data.Lens

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.thriftscala.FieldByPath

import com.twitter.tweetypie.thriftscala.HydrationType

/\*\*

\* Encapsulates a value and associated HydrationState. This class is intended to be used

\* with `ValueHydrator`, as the result type for hydrators that directly produce updated values,

\* in contrast with EditHydrator which uses `EditState` as a result type to produce update functions.

\*

\* @tparam A The type of the enclosed value, which is the result of hydration.

\*/

final case class ValueState[+A](value: A, state: HydrationState) {

/\*\*

\* Applies a function to the enclosed value and produces a new `ValueState` instance.

\*/

def map[B](f: A => B): ValueState[B] =

ValueState(f(value), state)

/\*\*

\* Produces a new `ValueState` that contains the value generated by `f`, but with state that is

\* the sum of the state from this `ValueState` and the one produced by `f`.

\*/

def flatMap[B](f: A => ValueState[B]): ValueState[B] = {

val ValueState(value2, state2) = f(value)

ValueState(value2, state ++ state2)

}

/\*\*

\* Applies a function to the enclosed state and produces a new `ValueState` instance.

\*/

def mapState[T](f: HydrationState => HydrationState): ValueState[A] =

ValueState(value, f(state))

/\*\*

\* Converts a `ValueState[A]` to an `EditState[B]`, using a lens. The resulting `EditState`

\* will overwrite the lensed field with the value from this `ValueState`.

\*/

def edit[B, A2 >: A](lens: Lens[B, A2]): EditState[B] =

EditState[B](b => ValueState(lens.set(b, value), state))

}

object ValueState {

val UnmodifiedNone: ValueState[None.type] = unmodified(None)

val StitchUnmodifiedNone: Stitch[ValueState[None.type]] = Stitch.value(UnmodifiedNone)

val UnmodifiedUnit: ValueState[Unit] = unmodified(())

val StitchUnmodifiedUnit: Stitch[ValueState[Unit]] = Stitch.value(UnmodifiedUnit)

val UnmodifiedNil: ValueState[Nil.type] = unmodified(Nil)

val StitchUnmodifiedNil: Stitch[ValueState[Nil.type]] = Stitch.value(UnmodifiedNil)

/\*\*

\* Produces a ValueState instance with the given value and an empty state HydrationState.

\*/

def unit[A](value: A): ValueState[A] =

ValueState[A](value, HydrationState.empty)

def unmodified[A](value: A): ValueState[A] =

ValueState(value, HydrationState.empty)

def modified[A](value: A): ValueState[A] =

ValueState(value, HydrationState.modified)

def modified[A](value: A, hydrationType: HydrationType): ValueState[A] =

ValueState(value, HydrationState.modified(hydrationType))

def success[A](value: A, modified: Boolean): ValueState[A] =

ValueState(value, HydrationState(modified))

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

ValueState(next, HydrationState.delta(prev, next))

def partial[A](value: A, field: FieldByPath): ValueState[A] =

ValueState(value, HydrationState.partial(field))

def partial[A](value: A, fields: Set[FieldByPath]): ValueState[A] =

ValueState(value, HydrationState.partial(fields))

/\*\*

\* Converts a `Seq` of `ValueState[A]` to a `ValueState` of `Seq[A]`.

\*/

def sequence[A](seq: Seq[ValueState[A]]): ValueState[Seq[A]] = {

ValueState(

value = seq.map(\_.value),

state = HydrationState.join(seq.map(\_.state): \_\*)

)

}

def join[A, B](va: ValueState[A], vb: ValueState[B]): ValueState[(A, B)] = {

val state =

HydrationState.join(

va.state,

vb.state

)

val value = (

va.value,

vb.value

)

ValueState(value, state)

}

def join[A, B, C](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C]

): ValueState[(A, B, C)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state

)

val value = (

va.value,

vb.value,

vc.value

)

ValueState(value, state)

}

def join[A, B, C, D](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D]

): ValueState[(A, B, C, D)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value

)

ValueState(value, state)

}

def join[A, B, C, D, E](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D],

ve: ValueState[E]

): ValueState[(A, B, C, D, E)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state,

ve.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value,

ve.value

)

ValueState(value, state)

}

def join[A, B, C, D, E, F](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D],

ve: ValueState[E],

vf: ValueState[F]

): ValueState[(A, B, C, D, E, F)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state,

ve.state,

vf.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value,

ve.value,

vf.value

)

ValueState(value, state)

}

def join[A, B, C, D, E, F, G](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D],

ve: ValueState[E],

vf: ValueState[F],

vg: ValueState[G]

): ValueState[(A, B, C, D, E, F, G)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state,

ve.state,

vf.state,

vg.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value,

ve.value,

vf.value,

vg.value

)

ValueState(value, state)

}

def join[A, B, C, D, E, F, G, H](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D],

ve: ValueState[E],

vf: ValueState[F],

vg: ValueState[G],

vh: ValueState[H]

): ValueState[(A, B, C, D, E, F, G, H)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state,

ve.state,

vf.state,

vg.state,

vh.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value,

ve.value,

vf.value,

vg.value,

vh.value

)

ValueState(value, state)

}

def join[A, B, C, D, E, F, G, H, I](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D],

ve: ValueState[E],

vf: ValueState[F],

vg: ValueState[G],

vh: ValueState[H],

vi: ValueState[I]

): ValueState[(A, B, C, D, E, F, G, H, I)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state,

ve.state,

vf.state,

vg.state,

vh.state,

vi.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value,

ve.value,

vf.value,

vg.value,

vh.value,

vi.value

)

ValueState(value, state)

}

def join[A, B, C, D, E, F, G, H, I, J](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D],

ve: ValueState[E],

vf: ValueState[F],

vg: ValueState[G],

vh: ValueState[H],

vi: ValueState[I],

vj: ValueState[J]

): ValueState[(A, B, C, D, E, F, G, H, I, J)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state,

ve.state,

vf.state,

vg.state,

vh.state,

vi.state,

vj.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value,

ve.value,

vf.value,

vg.value,

vh.value,

vi.value,

vj.value

)

ValueState(value, state)

}

def join[A, B, C, D, E, F, G, H, I, J, K](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D],

ve: ValueState[E],

vf: ValueState[F],

vg: ValueState[G],

vh: ValueState[H],

vi: ValueState[I],

vj: ValueState[J],

vk: ValueState[K]

): ValueState[(A, B, C, D, E, F, G, H, I, J, K)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state,

ve.state,

vf.state,

vg.state,

vh.state,

vi.state,

vj.state,

vk.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value,

ve.value,

vf.value,

vg.value,

vh.value,

vi.value,

vj.value,

vk.value

)

ValueState(value, state)

}

def join[A, B, C, D, E, F, G, H, I, J, K, L](

va: ValueState[A],

vb: ValueState[B],

vc: ValueState[C],

vd: ValueState[D],

ve: ValueState[E],

vf: ValueState[F],

vg: ValueState[G],

vh: ValueState[H],

vi: ValueState[I],

vj: ValueState[J],

vk: ValueState[K],

vl: ValueState[L]

): ValueState[(A, B, C, D, E, F, G, H, I, J, K, L)] = {

val state =

HydrationState.join(

va.state,

vb.state,

vc.state,

vd.state,

ve.state,

vf.state,

vg.state,

vh.state,

vi.state,

vj.state,

vk.state,

vl.state

)

val value = (

va.value,

vb.value,

vc.value,

vd.value,

ve.value,

vf.value,

vg.value,

vh.value,

vi.value,

vj.value,

vk.value,

vl.value

)

ValueState(value, state)

}

}