package com.twitter.servo.store

import com.twitter.servo.util.Gate

import com.twitter.util.Future

/\*\*

\* models a write-store of key/values

\*/

trait Store[K, V] {

def create(value: V): Future[V]

def update(value: V): Future[Unit]

def destroy(key: K): Future[Unit]

}

object Store {

/\*\*

\* Filter store operations based on either the key or the value. If the gate passes then forward

\* the operation to the underlying store, if not then forward the operation to a null store

\* (effectively a no-op)

\*/

def filtered[K, V](store: Store[K, V], filterKey: Gate[K], filterValue: Gate[V]) =

new GatedStore(store, new NullStore[K, V], filterKey, filterValue)

/\*\*

\* A store type that selects between one of two underlying stores based on the key/value of the

\* operation. If the key/value gate passes, forward the operation to the primary store, otherwise

\* forward the operation to the secondary store.

\*/

def gated[K, V](

primary: Store[K, V],

secondary: Store[K, V],

usePrimaryKey: Gate[K],

usePrimaryValue: Gate[V]

) = new GatedStore(primary, secondary, usePrimaryKey, usePrimaryValue)

/\*\*

\* A store type that selects between one of two underlying stores based on a predicative value,

\* which may change dynamically at runtime.

\*/

def deciderable[K, V](

primary: Store[K, V],

backup: Store[K, V],

primaryIsAvailable: => Boolean

) = new DeciderableStore(primary, backup, primaryIsAvailable)

}

trait StoreWrapper[K, V] extends Store[K, V] {

def underlyingStore: Store[K, V]

override def create(value: V) = underlyingStore.create(value)

override def update(value: V) = underlyingStore.update(value)

override def destroy(key: K) = underlyingStore.destroy(key)

}

class NullStore[K, V] extends Store[K, V] {

override def create(value: V) = Future.value(value)

override def update(value: V) = Future.Done

override def destroy(key: K) = Future.Done

}

/\*\*

\* A Store type that selects between one of two underlying stores based

\* on the key/value, which may change dynamically at runtime.

\*/

private[servo] class GatedStore[K, V](

primary: Store[K, V],

secondary: Store[K, V],

usePrimaryKey: Gate[K],

usePrimaryValue: Gate[V])

extends Store[K, V] {

private[this] def pick[T](item: T, gate: Gate[T]) = if (gate(item)) primary else secondary

override def create(value: V) = pick(value, usePrimaryValue).create(value)

override def update(value: V) = pick(value, usePrimaryValue).update(value)

override def destroy(key: K) = pick(key, usePrimaryKey).destroy(key)

}

/\*\*

\* A Store type that selects between one of two underlying stores based

\* on a predicative value, which may change dynamically at runtime.

\*/

class DeciderableStore[K, V](

primary: Store[K, V],

backup: Store[K, V],

primaryIsAvailable: => Boolean)

extends Store[K, V] {

private[this] def pick = if (primaryIsAvailable) primary else backup

override def create(value: V) = pick.create(value)

override def update(value: V) = pick.update(value)

override def destroy(key: K) = pick.destroy(key)

}