package com.twitter.servo

import com.twitter.util.Future

package object repository {

/\*\*

\* Base repository type. Maps a Query to a future Result

\*/

type Repository[-Q, +R] = Q => Future[R]

/\*\*

\* RepositoryFilters can be chained onto Repositories to asynchronously apply transformations to

\* Repository results.

\*/

type RepositoryFilter[-Q, -R, +S] = (Q, Future[R]) => Future[S]

type KeyValueResult[K, V] = keyvalue.KeyValueResult[K, V]

val KeyValueResult = keyvalue.KeyValueResult

/\*\*

\* A KeyValueRepository is a type of repository that handles bulk gets of data. The query

\* defines the values to fetch, and is usually made of up of a Seq[K], possibly with other

\* contextual information needed to perform the query. The result is a KeyValueResult,

\* which contains a break-out of found, notFound, and failed key lookups. The set of

\* keys may or may-not be computable locally from the query. This top-level type does not

\* require that the keys are computable from the query, but certain instances, such as

\* CachingKeyValueRepository, do require key-computability.

\*/

type KeyValueRepository[Q, K, V] = Repository[Q, KeyValueResult[K, V]]

type CounterKeyValueRepository[K] = KeyValueRepository[Seq[K], K, Long]

/\*\*

\* For KeyValueRepository scenarios where the query is a sequence of keys, a SubqueryBuilder

\* defines how to convert a sub-set of the keys from the query into a query.

\*/

type SubqueryBuilder[Q <: Seq[K], K] = (Seq[K], Q) => Q

/\*\*

\* A SubqueryBuilder where the query type is nothing more than a sequence of keys.

\*/

@deprecated("use keysAsQuery", "1.1.0")

def KeysAsQuery[K]: SubqueryBuilder[Seq[K], K] = keysAsQuery[K]

/\*\*

\* A SubqueryBuilder where the query type is nothing more than a sequence of keys.

\*/

def keysAsQuery[K]: SubqueryBuilder[Seq[K], K] = (keys, parentQuery) => keys

}