package com.twitter.ann.service.query\_server.common

import com.twitter.ann.common.\_

import com.twitter.ann.common.EmbeddingType.\_

import com.twitter.ann.common.thriftscala.AnnQueryService.Query

import com.twitter.ann.common.thriftscala.AnnQueryService

import com.twitter.ann.common.thriftscala.NearestNeighbor

import com.twitter.ann.common.thriftscala.NearestNeighborResult

import com.twitter.ann.common.thriftscala.{Distance => ServiceDistance}

import com.twitter.ann.common.thriftscala.{RuntimeParams => ServiceRuntimeParams}

import com.twitter.bijection.Injection

import com.twitter.finagle.Service

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finatra.thrift.Controller

import com.twitter.mediaservices.commons.{ThriftServer => TServer}

import java.nio.ByteBuffer

import javax.inject.Inject

class QueryIndexThriftController[T, P <: RuntimeParams, D <: Distance[D]] @Inject() (

statsReceiver: StatsReceiver,

queryable: Queryable[T, P, D],

runtimeParamInjection: Injection[P, ServiceRuntimeParams],

distanceInjection: Injection[D, ServiceDistance],

idInjection: Injection[T, Array[Byte]])

extends Controller(AnnQueryService) {

private[this] val thriftServer = new TServer(statsReceiver, Some(RuntimeExceptionTransform))

val trackingStatName = "ann\_query"

private[this] val stats = statsReceiver.scope(trackingStatName)

private[this] val numOfNeighboursRequested = stats.stat("num\_of\_neighbours\_requested")

private[this] val numOfNeighboursResponse = stats.stat("num\_of\_neighbours\_response")

private[this] val queryKeyNotFound = stats.stat("query\_key\_not\_found")

/\*\*

\* Implements AnnQueryService.query, returns nearest neighbours for a given query

\*/

val query: Service[Query.Args, Query.SuccessType] = { args: Query.Args =>

thriftServer.track(trackingStatName) {

val query = args.query

val key = query.key

val embedding = embeddingSerDe.fromThrift(query.embedding)

val numOfNeighbours = query.numberOfNeighbors

val withDistance = query.withDistance

val runtimeParams = runtimeParamInjection.invert(query.runtimeParams).get

numOfNeighboursRequested.add(numOfNeighbours)

val result = if (withDistance) {

val nearestNeighbors = if (queryable.isInstanceOf[QueryableGrouped[T, P, D]]) {

queryable

.asInstanceOf[QueryableGrouped[T, P, D]]

.queryWithDistance(embedding, numOfNeighbours, runtimeParams, key)

} else {

queryable

.queryWithDistance(embedding, numOfNeighbours, runtimeParams)

}

nearestNeighbors.map { list =>

list.map { nn =>

NearestNeighbor(

ByteBuffer.wrap(idInjection.apply(nn.neighbor)),

Some(distanceInjection.apply(nn.distance))

)

}

}

} else {

val nearestNeighbors = if (queryable.isInstanceOf[QueryableGrouped[T, P, D]]) {

queryable

.asInstanceOf[QueryableGrouped[T, P, D]]

.query(embedding, numOfNeighbours, runtimeParams, key)

} else {

queryable

.query(embedding, numOfNeighbours, runtimeParams)

}

nearestNeighbors

.map { list =>

list.map { nn =>

NearestNeighbor(ByteBuffer.wrap(idInjection.apply(nn)))

}

}

}

result.map(NearestNeighborResult(\_)).onSuccess { r =>

numOfNeighboursResponse.add(r.nearestNeighbors.size)

}

}

}

handle(Query) { query }

}