package com.twitter.search.earlybird\_root.filters;

import java.util.Collections;

import java.util.Map;

import javax.inject.Inject;

import com.google.common.collect.Maps;

import com.twitter.finagle.Service;

import com.twitter.finagle.SimpleFilter;

import com.twitter.search.common.decider.SearchDecider;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.schema.earlybird.EarlybirdCluster;

import com.twitter.search.earlybird.thrift.EarlybirdResponse;

import com.twitter.search.earlybird.thrift.EarlybirdResponseCode;

import com.twitter.search.earlybird\_root.common.EarlybirdRequestContext;

import com.twitter.search.earlybird\_root.common.EarlybirdRequestType;

import com.twitter.util.Future;

/\*\*

\* A Finagle filter that determines if a certain cluster is available to the SuperRoot.

\*

\* Normally, all clusters should be available. However, if there's a problem with our systems, and

\* our search clusters are causing issues for other services (time outs, for example), then we might

\* want to be disable them, and return errors to our clients.

\*/

public class EarlybirdClusterAvailableFilter

extends SimpleFilter<EarlybirdRequestContext, EarlybirdResponse> {

private final SearchDecider decider;

private final EarlybirdCluster cluster;

private final String allRequestsDeciderKey;

private final Map<EarlybirdRequestType, String> requestTypeDeciderKeys;

private final Map<EarlybirdRequestType, SearchCounter> disabledRequests;

/\*\*

\* Creates a new EarlybirdClusterAvailableFilter instance.

\*

\* @param decider The decider to use to determine if this cluster is available.

\* @param cluster The cluster.

\*/

@Inject

public EarlybirdClusterAvailableFilter(SearchDecider decider, EarlybirdCluster cluster) {

this.decider = decider;

this.cluster = cluster;

String clusterName = cluster.getNameForStats();

this.allRequestsDeciderKey = "superroot\_" + clusterName + "\_cluster\_available\_for\_all\_requests";

Map<EarlybirdRequestType, String> tempDeciderKeys = Maps.newEnumMap(EarlybirdRequestType.class);

Map<EarlybirdRequestType, SearchCounter> tempCounters =

Maps.newEnumMap(EarlybirdRequestType.class);

for (EarlybirdRequestType requestType : EarlybirdRequestType.values()) {

String requestTypeName = requestType.getNormalizedName();

tempDeciderKeys.put(requestType, "superroot\_" + clusterName + "\_cluster\_available\_for\_"

+ requestTypeName + "\_requests");

tempCounters.put(requestType, SearchCounter.export(

"cluster\_available\_filter\_" + clusterName + "\_"

+ requestTypeName + "\_disabled\_requests"));

}

requestTypeDeciderKeys = Collections.unmodifiableMap(tempDeciderKeys);

disabledRequests = Collections.unmodifiableMap(tempCounters);

}

@Override

public Future<EarlybirdResponse> apply(

EarlybirdRequestContext requestContext,

Service<EarlybirdRequestContext, EarlybirdResponse> service) {

EarlybirdRequestType requestType = requestContext.getEarlybirdRequestType();

if (!decider.isAvailable(allRequestsDeciderKey)

|| !decider.isAvailable(requestTypeDeciderKeys.get(requestType))) {

disabledRequests.get(requestType).increment();

return Future.value(

errorResponse("The " + cluster.getNameForStats() + " cluster is not available for "

+ requestType.getNormalizedName() + " requests."));

}

return service.apply(requestContext);

}

private EarlybirdResponse errorResponse(String debugMessage) {

return new EarlybirdResponse(EarlybirdResponseCode.PERSISTENT\_ERROR, 0)

.setDebugString(debugMessage);

}

}