package com.twitter.search.ingester.pipeline.twitter.userupdates;

import java.util.AbstractMap;

import java.util.Collection;

import java.util.Collections;

import java.util.EnumSet;

import java.util.List;

import java.util.Map;

import java.util.Objects;

import java.util.Set;

import java.util.function.Function;

import java.util.stream.Collectors;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Sets;

import org.apache.commons.text.CaseUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.collections.Pair;

import com.twitter.decider.Decider;

import com.twitter.finagle.util.DefaultTimer;

import com.twitter.gizmoduck.thriftjava.LifecycleChangeReason;

import com.twitter.gizmoduck.thriftjava.LookupContext;

import com.twitter.gizmoduck.thriftjava.QueryFields;

import com.twitter.gizmoduck.thriftjava.Safety;

import com.twitter.gizmoduck.thriftjava.UpdateDiffItem;

import com.twitter.gizmoduck.thriftjava.User;

import com.twitter.gizmoduck.thriftjava.UserModification;

import com.twitter.gizmoduck.thriftjava.UserService;

import com.twitter.gizmoduck.thriftjava.UserType;

import com.twitter.search.common.indexing.thriftjava.AntisocialUserUpdate;

import com.twitter.search.common.indexing.thriftjava.UserUpdateType;

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

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

import com.twitter.util.Duration;

import com.twitter.util.Future;

import com.twitter.util.TimeoutException;

/\*\*

\* This class ingests {@link UserModification} events and transforms them into a possibly empty list

\* of {@link AntisocialUserUpdate}s to be indexed by Earlybirds.

\*/

public class UserUpdateIngester {

private static final Logger LOG = LoggerFactory.getLogger(UserUpdateIngester.class);

private static final Duration RESULT\_TIMEOUT = Duration.fromSeconds(3);

private static final List<AntisocialUserUpdate> NO\_UPDATE = Collections.emptyList();

// Map from UserUpdateType to a set of Safety fields to examine.

private static final Map<UserUpdateType, Set<Safety.\_Fields>> SAFETY\_FIELDS\_MAP =

ImmutableMap.of(

UserUpdateType.ANTISOCIAL,

Sets.immutableEnumSet(

Safety.\_Fields.SUSPENDED, Safety.\_Fields.DEACTIVATED, Safety.\_Fields.OFFBOARDED),

UserUpdateType.NSFW,

Sets.immutableEnumSet(Safety.\_Fields.NSFW\_USER, Safety.\_Fields.NSFW\_ADMIN),

UserUpdateType.PROTECTED, Sets.immutableEnumSet(Safety.\_Fields.IS\_PROTECTED));

private static final Function<Safety.\_Fields, String> FIELD\_TO\_FIELD\_NAME\_FUNCTION =

field -> "safety." + CaseUtils.toCamelCase(field.name(), false, '\_');

private static final Map<String, UserUpdateType> FIELD\_NAME\_TO\_TYPE\_MAP =

SAFETY\_FIELDS\_MAP.entrySet().stream()

.flatMap(

entry -> entry.getValue().stream()

.map(field -> new AbstractMap.SimpleEntry<>(

FIELD\_TO\_FIELD\_NAME\_FUNCTION.apply(field),

entry.getKey())))

.collect(Collectors.toMap(

AbstractMap.SimpleEntry::getKey,

AbstractMap.SimpleEntry::getValue));

private static final Map<String, Safety.\_Fields> FIELD\_NAME\_TO\_FIELD\_MAP =

SAFETY\_FIELDS\_MAP.values().stream()

.flatMap(Collection::stream)

.collect(Collectors.toMap(

FIELD\_TO\_FIELD\_NAME\_FUNCTION,

Function.identity()));

private static final LookupContext LOOKUP\_CONTEXT = new LookupContext()

.setInclude\_deactivated(true)

.setInclude\_erased(true)

.setInclude\_suspended(true)

.setInclude\_offboarded(true)

.setInclude\_protected(true);

private final UserService.ServiceToClient userService;

private final Decider decider;

private final SearchLongGauge userModificationLatency;

private final SearchCounter unsuccessfulUserModificationCount;

private final SearchCounter byInactiveAccountDeactivationUserModificationCount;

private final SearchCounter irrelevantUserModificationCount;

private final SearchCounter notNormalUserCount;

private final SearchCounter missingSafetyCount;

private final SearchCounter userServiceRequests;

private final SearchCounter userServiceSuccesses;

private final SearchCounter userServiceNoResults;

private final SearchCounter userServiceFailures;

private final SearchCounter userServiceTimeouts;

private final Map<Pair<UserUpdateType, Boolean>, SearchCounter> counterMap;

public UserUpdateIngester(

String statPrefix,

UserService.ServiceToClient userService,

Decider decider

) {

this.userService = userService;

this.decider = decider;

userModificationLatency =

SearchLongGauge.export(statPrefix + "\_user\_modification\_latency\_ms");

unsuccessfulUserModificationCount =

SearchCounter.export(statPrefix + "\_unsuccessful\_user\_modification\_count");

byInactiveAccountDeactivationUserModificationCount =

SearchCounter.export(statPrefix

+ "\_by\_inactive\_account\_deactivation\_user\_modification\_count");

irrelevantUserModificationCount =

SearchCounter.export(statPrefix + "\_irrelevant\_user\_modification\_count");

notNormalUserCount =

SearchCounter.export(statPrefix + "\_not\_normal\_user\_count");

missingSafetyCount =

SearchCounter.export(statPrefix + "\_missing\_safety\_count");

userServiceRequests =

SearchCounter.export(statPrefix + "\_user\_service\_requests");

userServiceSuccesses =

SearchCounter.export(statPrefix + "\_user\_service\_successes");

userServiceNoResults =

SearchCounter.export(statPrefix + "\_user\_service\_no\_results");

userServiceFailures =

SearchCounter.export(statPrefix + "\_user\_service\_failures");

userServiceTimeouts =

SearchCounter.export(statPrefix + "\_user\_service\_timeouts");

counterMap = ImmutableMap.<Pair<UserUpdateType, Boolean>, SearchCounter>builder()

.put(Pair.of(UserUpdateType.ANTISOCIAL, true),

SearchCounter.export(statPrefix + "\_antisocial\_set\_count"))

.put(Pair.of(UserUpdateType.ANTISOCIAL, false),

SearchCounter.export(statPrefix + "\_antisocial\_unset\_count"))

.put(Pair.of(UserUpdateType.NSFW, true),

SearchCounter.export(statPrefix + "\_nsfw\_set\_count"))

.put(Pair.of(UserUpdateType.NSFW, false),

SearchCounter.export(statPrefix + "\_nsfw\_unset\_count"))

.put(Pair.of(UserUpdateType.PROTECTED, true),

SearchCounter.export(statPrefix + "\_protected\_set\_count"))

.put(Pair.of(UserUpdateType.PROTECTED, false),

SearchCounter.export(statPrefix + "\_protected\_unset\_count"))

.build();

}

/\*\*

\* Convert a UserModification event into a (possibly empty) list of antisocial updates for

\* Earlybird.

\*/

public Future<List<AntisocialUserUpdate>> transform(UserModification userModification) {

userModificationLatency.set(System.currentTimeMillis() - userModification.getUpdated\_at\_msec());

if (!userModification.isSuccess()) {

unsuccessfulUserModificationCount.increment();

return Future.value(NO\_UPDATE);

}

// To avoid UserTable gets overflowed, we exclude traffic from ByInactiveAccountDeactivation

if (userModification.getUser\_audit\_data() != null

&& userModification.getUser\_audit\_data().getReason() != null

&& userModification.getUser\_audit\_data().getReason()

== LifecycleChangeReason.BY\_INACTIVE\_ACCOUNT\_DEACTIVATION) {

byInactiveAccountDeactivationUserModificationCount.increment();

return Future.value(NO\_UPDATE);

}

long userId = userModification.getUser\_id();

Set<UserUpdateType> userUpdateTypes = getUserUpdateTypes(userModification);

if (userUpdateTypes.isEmpty()) {

irrelevantUserModificationCount.increment();

return Future.value(NO\_UPDATE);

}

Future<User> userFuture = userModification.isSetCreate()

? Future.value(userModification.getCreate())

: getUser(userId);

return userFuture

.map(user -> {

if (user == null) {

return NO\_UPDATE;

} else if (user.getUser\_type() != UserType.NORMAL) {

LOG.info("User with id={} is not a normal user.", userId);

notNormalUserCount.increment();

return NO\_UPDATE;

} else if (!user.isSetSafety()) {

LOG.info("Safety for User with id={} is missing.", userId);

missingSafetyCount.increment();

return NO\_UPDATE;

}

if (userModification.isSetUpdate()) {

// Apply relevant updates from UserModification as User returned from Gizmoduck may not

// have reflected them yet.

applyUpdates(user, userModification);

}

return userUpdateTypes.stream()

.map(userUpdateType ->

convertToAntiSocialUserUpdate(

user, userUpdateType, userModification.getUpdated\_at\_msec()))

.peek(update ->

counterMap.get(Pair.of(update.getType(), update.isValue())).increment())

.collect(Collectors.toList());

})

.onFailure(com.twitter.util.Function.cons(exception -> {

if (exception instanceof UserNotFoundException) {

userServiceNoResults.increment();

} else if (exception instanceof TimeoutException) {

userServiceTimeouts.increment();

LOG.error("UserService.get timed out for user id=" + userId, exception);

} else {

userServiceFailures.increment();

LOG.error("UserService.get failed for user id=" + userId, exception);

}

}));

}

private static Set<UserUpdateType> getUserUpdateTypes(UserModification userModification) {

Set<UserUpdateType> types = EnumSet.noneOf(UserUpdateType.class);

if (userModification.isSetUpdate()) {

userModification.getUpdate().stream()

.map(UpdateDiffItem::getField\_name)

.map(FIELD\_NAME\_TO\_TYPE\_MAP::get)

.filter(Objects::nonNull)

.collect(Collectors.toCollection(() -> types));

} else if (userModification.isSetCreate() && userModification.getCreate().isSetSafety()) {

Safety safety = userModification.getCreate().getSafety();

if (safety.isSuspended()) {

types.add(UserUpdateType.ANTISOCIAL);

}

if (safety.isNsfw\_admin() || safety.isNsfw\_user()) {

types.add(UserUpdateType.NSFW);

}

if (safety.isIs\_protected()) {

types.add(UserUpdateType.PROTECTED);

}

}

return types;

}

private Future<User> getUser(long userId) {

userServiceRequests.increment();

return userService.get(

LOOKUP\_CONTEXT,

Collections.singletonList(userId),

Collections.singleton(QueryFields.SAFETY))

.within(DefaultTimer.getInstance(), RESULT\_TIMEOUT)

.flatMap(userResults -> {

if (userResults.size() != 1 || !userResults.get(0).isSetUser()) {

return Future.exception(new UserNotFoundException(userId));

}

userServiceSuccesses.increment();

return Future.value(userResults.get(0).getUser());

});

}

private static void applyUpdates(User user, UserModification userModification) {

userModification.getUpdate().stream()

.filter(update -> FIELD\_NAME\_TO\_FIELD\_MAP.containsKey(update.getField\_name()))

.filter(UpdateDiffItem::isSetAfter)

.forEach(update ->

user.getSafety().setFieldValue(

FIELD\_NAME\_TO\_FIELD\_MAP.get(update.getField\_name()),

Boolean.valueOf(update.getAfter()))

);

}

private AntisocialUserUpdate convertToAntiSocialUserUpdate(

User user,

UserUpdateType userUpdateType,

long updatedAt) {

boolean value = SAFETY\_FIELDS\_MAP.get(userUpdateType).stream()

.anyMatch(safetyField -> (boolean) user.getSafety().getFieldValue(safetyField));

return new AntisocialUserUpdate(user.getId(), userUpdateType, value, updatedAt);

}

class UserNotFoundException extends Exception {

UserNotFoundException(long userId) {

super("User " + userId + " not found.");

}

}

}