package com.twitter.product\_mixer.core.service.debug\_query

import com.fasterxml.jackson.databind.SerializationFeature

import com.twitter.finagle.Service

import com.twitter.finagle.context.Contexts

import com.twitter.finagle.tracing.Trace.traceLocal

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.finagle.transport.Transport

import com.twitter.product\_mixer.core.functional\_component.configapi.ParamsBuilder

import com.twitter.product\_mixer.core.model.common.identifier.ComponentIdentifierStack

import com.twitter.product\_mixer.core.model.marshalling.request.Product

import com.twitter.product\_mixer.core.model.marshalling.request.Request

import com.twitter.product\_mixer.core.pipeline.product.ProductPipeline

import com.twitter.product\_mixer.core.pipeline.product.ProductPipelineRequest

import com.twitter.product\_mixer.core.product.registry.ProductPipelineRegistry

import com.twitter.product\_mixer.core.{thriftscala => t}

import com.twitter.scrooge.ThriftStruct

import com.twitter.scrooge.{Request => ScroogeRequest}

import com.twitter.scrooge.{Response => ScroogeResponse}

import com.twitter.stitch.Stitch

import com.twitter.turntable.context.TurntableRequestContextKey

import com.twitter.util.jackson.ScalaObjectMapper

import javax.inject.Inject

import javax.inject.Singleton

import scala.reflect.runtime.universe.TypeTag

/\*\*

\* Returns the complete execution log for a pipeline query. These endpoints are intended for

\* debugging (primarily through Turntable).

\*/

@Singleton

class DebugQueryService @Inject() (

productPipelineRegistry: ProductPipelineRegistry,

paramsBuilder: ParamsBuilder,

authorizationService: AuthorizationService) {

private val mapper =

ScalaObjectMapper.builder

.withAdditionalJacksonModules(Seq(ParamsSerializerModule))

.withSerializationConfig(

Map(

// These are copied from the default serialization config.

SerializationFeature.WRITE\_DATES\_AS\_TIMESTAMPS -> false,

SerializationFeature.WRITE\_ENUMS\_USING\_TO\_STRING -> true,

// Generally we want to be defensive when serializing since we don't control everything that's

// serialized. This issue also came up when trying to serialize Unit as part of sync side effects.

SerializationFeature.FAIL\_ON\_EMPTY\_BEANS -> false,

))

// The default implementation represents numbers as JSON Numbers (i.e. Double with 53 bit precision

// which leads to Snowflake IDs being cropped in the case of tweets.

.withNumbersAsStrings(true)

.objectMapper

def apply[

ThriftRequest <: ThriftStruct with Product1[MixerServiceRequest],

MixerServiceRequest <: ThriftStruct,

MixerRequest <: Request

](

unmarshaller: MixerServiceRequest => MixerRequest

)(

implicit requestTypeTag: TypeTag[MixerRequest]

): Service[ScroogeRequest[ThriftRequest], ScroogeResponse[t.PipelineExecutionResult]] = {

(thriftRequest: ScroogeRequest[ThriftRequest]) =>

{

val request = unmarshaller(thriftRequest.args.\_1)

val params = paramsBuilder.build(

clientContext = request.clientContext,

product = request.product,

featureOverrides = request.debugParams.flatMap(\_.featureOverrides).getOrElse(Map.empty)

)

val productPipeline = productPipelineRegistry

.getProductPipeline[MixerRequest, Any](request.product)

verifyRequestAuthorization(request.product, productPipeline)

Contexts.broadcast.letClear(TurntableRequestContextKey) {

Stitch

.run(productPipeline

.arrow(ProductPipelineRequest(request, params)).map { detailedResult =>

// Serialization can be slow so a trace is useful both for optimization by the Promix

// team and to give visibility to customers.

val serializedJSON =

traceLocal("serialize\_debug\_response")(mapper.writeValueAsString(detailedResult))

t.PipelineExecutionResult(serializedJSON)

})

.map(ScroogeResponse(\_))

}

}

}

private def verifyRequestAuthorization(

product: Product,

productPipeline: ProductPipeline[\_, \_]

): Unit = {

val serviceIdentifier = ServiceIdentifier.fromCertificate(Transport.peerCertificate)

val requestContext = Contexts.broadcast

.get(TurntableRequestContextKey).getOrElse(throw MissingTurntableRequestContextException)

val componentStack = ComponentIdentifierStack(productPipeline.identifier, product.identifier)

authorizationService.verifyRequestAuthorization(

componentStack,

serviceIdentifier,

productPipeline.debugAccessPolicies,

requestContext)

}

}

object MissingTurntableRequestContextException

extends Exception("Request is missing turntable request context")