package com.twitter.product\_mixer.core.controllers

import com.twitter.finagle.http.Request

import com.twitter.scrooge.BinaryThriftStructSerializer

import com.twitter.scrooge.ThriftMethod

import com.twitter.scrooge.schema.ThriftDefinitions

import com.twitter.scrooge.schema.scrooge.scala.CompiledScroogeDefBuilder

import com.twitter.scrooge.schema.serialization.thrift.ReferenceResolver

import com.twitter.scrooge.schema.serialization.thrift.ThriftDefinitionsSerializer

import com.twitter.scrooge.schema.{thriftscala => THRIFT}

/\*\*

\* Endpoint to expose a Mixer's expected query configuration, including the request schema.

\*

\* @param debugEndpoint the debug Thrift endpoint. Passing [[None]] disables the query debugging

\* feature.

\* @tparam ServiceIface a thrift service containing the [[debugEndpoint]]

\*/

case class GetDebugConfigurationHandler[ServiceIface](

thriftMethod: ThriftMethod

)(

implicit val serviceIFace: Manifest[ServiceIface]) {

// We need to binary encode the service def because the underlying Thrift isn't sufficiently

// annotated to be serialized/deserialized by Jackson

private val serviceDef = {

val fullServiceDefinition: ThriftDefinitions.ServiceDef = CompiledScroogeDefBuilder

.build(serviceIFace).asInstanceOf[ThriftDefinitions.ServiceDef]

val endpointDefinition: ThriftDefinitions.ServiceEndpointDef =

fullServiceDefinition.endpointsByName(thriftMethod.name)

// Create a service definition which just contains the debug endpoint. At a bare minimum, we need

// to give callers a way to identify the debug endpoint. Sending back all the endpoints is

// redundant.

val serviceDefinition: ThriftDefinitions.ServiceDef =

fullServiceDefinition.copy(endpoints = Seq(endpointDefinition))

val thriftDefinitionsSerializer = {

// We don't make use of references but a reference resolver is required by the Scrooge API

val noopReferenceResolver: ReferenceResolver =

(\_: THRIFT.ReferenceDef) => throw new Exception("no references")

new ThriftDefinitionsSerializer(noopReferenceResolver, enableReferences = false)

}

val thriftBinarySerializer = BinaryThriftStructSerializer.apply(THRIFT.Definition)

val serializedServiceDef = thriftDefinitionsSerializer(serviceDefinition)

thriftBinarySerializer.toBytes(serializedServiceDef)

}

def apply(request: Request): DebugConfigurationResponse =

DebugConfigurationResponse(thriftMethod.name, serviceDef)

}

case class DebugConfigurationResponse(

debugEndpointName: String,

serviceDefinition: Array[Byte])