package com.twitter.servo.json

import com.fasterxml.jackson.core.JsonParser

import com.fasterxml.jackson.databind.JsonNode

import com.fasterxml.jackson.databind.ObjectMapper

import com.twitter.scrooge.ThriftStruct

import com.twitter.scrooge.ThriftStructCodec

import com.twitter.scrooge.ThriftStructSerializer

import difflib.DiffUtils

import java.io.StringWriter

import org.apache.thrift.protocol.TField

import org.apache.thrift.protocol.TProtocol

import org.apache.thrift.protocol.TProtocolFactory

import org.apache.thrift.protocol.TSimpleJSONProtocol

import org.apache.thrift.transport.TTransport

import scala.collection.JavaConverters.\_

import scala.language.experimental.macros

import scala.reflect.macros.blackbox.Context

object ThriftJsonInspector {

private val mapper = new ObjectMapper()

mapper.configure(JsonParser.Feature.ALLOW\_UNQUOTED\_FIELD\_NAMES, true)

private val factory = mapper.getFactory()

private def mkSerializer[T <: ThriftStruct](\_codec: ThriftStructCodec[T]) =

new ThriftStructSerializer[T] {

def codec = \_codec

def protocolFactory =

// Identical to TSimpleJSONProtocol.Factory except the TProtocol

// returned serializes Thrift pass-through fields with the name

// "(TField.id)" instead of empty string.

new TProtocolFactory {

def getProtocol(trans: TTransport): TProtocol =

new TSimpleJSONProtocol(trans) {

override def writeFieldBegin(field: TField): Unit =

writeString(if (field.name.isEmpty) s"(${field.id})" else field.name)

}

}

}

def apply[T <: ThriftStruct](codec: ThriftStructCodec[T]) = new ThriftJsonInspector(codec)

}

/\*\*

\* Helper for human inspection of Thrift objects.

\*/

class ThriftJsonInspector[T <: ThriftStruct](codec: ThriftStructCodec[T]) {

import ThriftJsonInspector.\_

private[this] val serializer = mkSerializer(codec)

/\*\*

\* Convert the Thrift object to a JSON representation based on this

\* object's codec, in the manner of TSimpleJSONProtocol. The resulting

\* JSON will have human-readable field names that match the field

\* names that were used in the Thrift definition that the codec was

\* created from, but the conversion is lossy, and the JSON

\* representation cannot be converted back.

\*/

def toSimpleJson(t: T): JsonNode =

mapper.readTree(factory.createParser(serializer.toBytes(t)))

/\*\*

\* Selects requested fields (matching against the JSON fields) from a

\* Thrift-generated class.

\*

\* Paths are specified as slash-separated strings (e.g.,

\* "key1/key2/key3"). If the path specifies an array or object, it is

\* included in the output in JSON format, otherwise the simple value is

\* converted to a string.

\*/

def select(item: T, paths: Seq[String]): Seq[String] = {

val jsonNode = toSimpleJson(item)

paths.map {

\_.split("/").foldLeft(jsonNode)(\_.findPath(\_)) match {

case node if node.isMissingNode => "[invalid-path]"

case node if node.isContainerNode => node.toString

case node => node.asText

}

}

}

/\*\*

\* Convert the given Thrift struct to a human-readable pretty-printed

\* JSON representation. This JSON cannot be converted back into a

\* struct. This output is intended for debug logging or interactive

\* inspection of Thrift objects.

\*/

def prettyPrint(t: T): String = print(t, true)

def print(t: T, pretty: Boolean = false): String = {

val writer = new StringWriter()

val generator = factory.createGenerator(writer)

if (pretty)

generator.useDefaultPrettyPrinter()

generator.writeTree(toSimpleJson(t))

writer.toString

}

/\*\*

\* Produce a human-readable unified diff of the json pretty-printed

\* representations of `a` and `b`. If the inputs have the same JSON

\* representation, the result will be the empty string.

\*/

def diff(a: T, b: T, contextLines: Int = 1): String = {

val linesA = prettyPrint(a).linesIterator.toList.asJava

val linesB = prettyPrint(b).linesIterator.toList.asJava

val patch = DiffUtils.diff(linesA, linesB)

DiffUtils.generateUnifiedDiff("a", "b", linesA, patch, contextLines).asScala.mkString("\n")

}

}

object syntax {

private[this] object CompanionObjectLoader {

def load[T](c: Context)(implicit t: c.universe.WeakTypeTag[T]) = {

val tSym = t.tpe.typeSymbol

val companion = tSym.asClass.companion

if (companion == c.universe.NoSymbol) {

c.abort(c.enclosingPosition, s"${tSym} has no companion object")

} else {

c.universe.Ident(companion)

}

}

}

/\*\*

\* Load the companion object of the named type parameter and require

\* it to be a ThriftStructCodec. Compilation will fail if the

\* companion object is not a ThriftStructCodec.

\*/

implicit def thriftStructCodec[T <: ThriftStruct]: ThriftStructCodec[T] =

macro CompanionObjectLoader.load[T]

implicit class ThriftJsonSyntax[T <: ThriftStruct](t: T)(implicit codec: ThriftStructCodec[T]) {

private[this] def inspector = ThriftJsonInspector(codec)

def toSimpleJson: JsonNode = inspector.toSimpleJson(t)

def prettyPrint: String = inspector.prettyPrint(t)

def diff(other: T, contextLines: Int = 1): String =

inspector.diff(t, other, contextLines)

}

}