package com.twitter.product\_mixer.core.module.stringcenter

import com.google.inject.Provides

import com.twitter.abdecider.LoggingABDecider

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.util.jackson.ScalaObjectMapper

import com.twitter.inject.TwitterModule

import com.twitter.inject.annotations.Flag

import com.twitter.product\_mixer.core.product.guice.scope.ProductScoped

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

import com.twitter.stringcenter.client.ExternalStringRegistry

import com.twitter.stringcenter.client.StringCenter

import com.twitter.stringcenter.client.StringCenterClientConfig

import com.twitter.stringcenter.client.sources.RefreshingStringSource

import com.twitter.stringcenter.client.sources.RefreshingStringSourceConfig

import com.twitter.stringcenter.client.sources.StringSource

import com.twitter.translation.Languages

import javax.inject.Singleton

import scala.collection.concurrent

/\*

\* Fun trivia - this has to be a Class not an Object, otherwise when you ./bazel test blah/...

\* and glob multiple feature tests together it'll reuse the concurrentMaps below across

\* executions / different server objects.

\*/

class ProductScopeStringCenterModule extends TwitterModule {

private val loadNothing =

flag[Boolean](name = "stringcenter.dontload", default = false, help = "Avoid loading any files")

flag[Boolean](

name = "stringcenter.handle.language.fallback",

default = true,

help = "Handle language fallback for services that don't already handle it")

flag[String](

name = "stringcenter.default\_bundle\_path",

default = "stringcenter",

help = "The path on disk to the default bundle available at startup time")

private val refreshingInterval = flag[Int](

name = "stringcenter.refresh\_interval\_minutes",

default = 3,

help = "How often to poll the refreshing bundle path to check for new bundles")

/\* The Guice injector is single threaded, but out of a preponderance of caution we use a concurrent Map.

\*

\* We need to ensure that we only build one StringSource, StringCenter client, and External String

\* Registry for each String Center Project. @ProductScoped doesn't ensure this on it's own as

\* two products can have the same String Center Project set.

\*/

val stringSources: concurrent.Map[String, StringSource] = concurrent.TrieMap.empty

val stringCenterClients: concurrent.Map[String, StringCenter] = concurrent.TrieMap.empty

val externalStringRegistries: concurrent.Map[String, ExternalStringRegistry] =

concurrent.TrieMap.empty

@ProductScoped

@Provides

def providesStringCenterClients(

abDecider: LoggingABDecider,

stringSource: StringSource,

languages: Languages,

statsReceiver: StatsReceiver,

clientConfig: StringCenterClientConfig,

product: Product

): StringCenter = {

stringCenterClients.getOrElseUpdate(

stringCenterForProduct(product), {

new StringCenter(

abDecider,

stringSource,

languages,

statsReceiver,

clientConfig

)

})

}

@ProductScoped

@Provides

def providesExternalStringRegistries(

product: Product

): ExternalStringRegistry = {

externalStringRegistries.getOrElseUpdate(

stringCenterForProduct(product), {

new ExternalStringRegistry()

})

}

@ProductScoped

@Provides

def providesStringCenterSources(

mapper: ScalaObjectMapper,

statsReceiver: StatsReceiver,

product: Product,

@Flag("stringcenter.default\_bundle\_path") defaultBundlePath: String

): StringSource = {

if (loadNothing()) {

StringSource.Empty

} else {

val stringCenterProduct = stringCenterForProduct(product)

stringSources.getOrElseUpdate(

stringCenterProduct, {

val config = RefreshingStringSourceConfig(

stringCenterProduct,

defaultBundlePath,

"stringcenter/downloaded/current/stringcenter",

refreshingInterval().minutes

)

new RefreshingStringSource(

config,

mapper,

statsReceiver

.scope("StringCenter", "refreshing", "project", stringCenterProduct))

}

)

}

}

private def stringCenterForProduct(product: Product): String =

product.stringCenterProject.getOrElse {

throw new UnsupportedOperationException(

s"No StringCenter project defined for Product ${product.identifier}")

}

@Singleton

@Provides

def providesStringCenterClientConfig(

@Flag("stringcenter.handle.language.fallback") handleLanguageFallback: Boolean

): StringCenterClientConfig = {

StringCenterClientConfig(handleLanguageFallback = handleLanguageFallback)

}

}