package com.twitter.servo.util

import com.twitter.finagle.util.DefaultTimer

import com.twitter.finagle.{Addr, Name, Namer}

import com.twitter.logging.Logger

import com.twitter.util.\_

import scala.collection.JavaConverters.\_

/\*\*

\* A simple utility class to wait for serverset names to be resolved at startup.

\*

\* See [[com.twitter.finagle.client.ClientRegistry.expAllRegisteredClientsResolved()]] for an

\* alternative way to wait for ServerSet resolution.

\*/

object WaitForServerSets {

val log = Logger.get("WaitForServerSets")

/\*\*

\* Convenient wrapper for single name in Java. Provides the default timer from Finagle.

\*/

def ready(name: Name, timeout: Duration): Future[Unit] =

ready(Seq(name), timeout, DefaultTimer)

/\*\*

\* Java Compatibility wrapper. Uses java.util.List instead of Seq.

\*/

def ready(names: java.util.List[Name], timeout: Duration, timer: Timer): Future[Unit] =

ready(names.asScala, timeout, timer)

/\*\*

\* Returns a Future that is satisfied when no more names resolve to Addr.Pending,

\* or the specified timeout expires.

\*

\* This ignores address resolution failures, so just because the Future is satisfied

\* doesn't necessarily imply that all names are resolved to something useful.

\*/

def ready(names: Seq[Name], timeout: Duration, timer: Timer): Future[Unit] = {

val vars: Var[Seq[(Name, Addr)]] = Var.collect(names.map {

case n @ Name.Path(v) => Namer.resolve(v).map((n, \_))

case n @ Name.Bound(v) => v.map((n, \_))

})

val pendings = vars.changes.map { names =>

names.filter { case (\_, addr) => addr == Addr.Pending }

}

pendings

.filter(\_.isEmpty)

.toFuture()

.unit

.within(

timer,

timeout,

new TimeoutException(

"Failed to resolve: " +

vars.map(\_.map { case (name, \_) => name }).sample()

)

)

}

}