package com.twitter.tweetypie.core

import com.twitter.spam.rtf.thriftscala.FilteredReason

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

/\*\*

\* The data about a quoted tweet that needs to be carried forward to

\* Tweetypie clients.

\*/

sealed trait QuotedTweetResult {

def filteredReason: Option[FilteredReason]

def toOption: Option[TweetResult]

def map(f: TweetResult => TweetResult): QuotedTweetResult

}

object QuotedTweetResult {

case object NotFound extends QuotedTweetResult {

def filteredReason: None.type = None

def toOption: None.type = None

def map(f: TweetResult => TweetResult): NotFound.type = this

}

case class Filtered(state: FilteredState.Unavailable) extends QuotedTweetResult {

def filteredReason: Option[FilteredReason] =

state match {

case st: FilteredState.HasFilteredReason => Some(st.filteredReason)

case \_ => None

}

def toOption: None.type = None

def map(f: TweetResult => TweetResult): Filtered = this

}

case class Found(result: TweetResult) extends QuotedTweetResult {

def filteredReason: Option[FilteredReason] = result.value.suppress.map(\_.filteredReason)

def toOption: Option[TweetResult] = Some(result)

def map(f: TweetResult => TweetResult): QuotedTweetResult = Found(f(result))

}

def fromTry(tryResult: Try[TweetResult]): Try[QuotedTweetResult] =

tryResult match {

case Return(result) => Return(Found(result))

case Throw(state: FilteredState.Unavailable) => Return(Filtered(state))

case Throw(com.twitter.stitch.NotFound) => Return(NotFound)

case Throw(e) => Throw(e)

}

}