package com.twitter.visibility.rules.utils

import com.twitter.visibility.features.Feature

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.models.ContentId

import com.twitter.visibility.models.SafetyLevel

import com.twitter.visibility.rules.Filtered

import com.twitter.visibility.rules.Rule

import com.twitter.visibility.rules.RuleBase

import com.twitter.visibility.rules.RuleBase.RuleMap

import com.twitter.visibility.rules.providers.ProvidedEvaluationContext

import com.twitter.visibility.rules.providers.PolicyProvider

object ShimUtils {

def preFilterFeatureMap(

featureMap: FeatureMap,

safetyLevel: SafetyLevel,

contentId: ContentId,

evaluationContext: ProvidedEvaluationContext,

policyProviderOpt: Option[PolicyProvider] = None,

): FeatureMap = {

val safetyLevelRules: Seq[Rule] = policyProviderOpt match {

case Some(policyProvider) =>

policyProvider

.policyForSurface(safetyLevel)

.forContentId(contentId)

case \_ => RuleMap(safetyLevel).forContentId(contentId)

}

val afterDisabledRules =

safetyLevelRules.filter(evaluationContext.ruleEnabledInContext)

val afterMissingFeatureRules =

afterDisabledRules.filter(rule => {

val missingFeatures: Set[Feature[\_]] = rule.featureDependencies.collect {

case feature: Feature[\_] if !featureMap.contains(feature) => feature

}

if (missingFeatures.isEmpty) {

true

} else {

false

}

})

val afterPreFilterRules = afterMissingFeatureRules.filter(rule => {

rule.preFilter(evaluationContext, featureMap.constantMap, null) match {

case Filtered =>

false

case \_ =>

true

}

})

val filteredFeatureMap =

RuleBase.removeUnusedFeaturesFromFeatureMap(featureMap, afterPreFilterRules)

filteredFeatureMap

}

}