/\*\*

\* Created by <a href="mailto:118808@qq.com">Rustic</a> @ 9/18/18 12:13 PM.

\*/

class CommitContractAction : AnAction() {

override fun actionPerformed(e: AnActionEvent) {

// Using the event, create and show a dialog

val currentProject = e.project ?: return

val file = e.getData(LangDataKeys.VIRTUAL\_FILE) ?: return

val psiDirectory = PsiManager.getInstance(currentProject).findDirectory(file) ?: return

val psiFiles = psiDirectory.files.filter { psiFile ->

psiFile.name.endsWith(".go", true)

&& !psiFile.name.contains("\_autogen\_")

&& !psiFile.name.endsWith("\_test.go")

}

if (psiFiles.isEmpty()) {

Messages.showMessageDialog(currentProject, " The directory does not contains \".go\" file ", "Error", Messages.getInformationIcon())

return

}

val checkObject = CheckContract()

if (!checkObject.checkFiles(psiFiles,file.path)) return

try {

val message = CommitContract().commit(psiDirectory, checkObject)

when (message) {

"" -> Messages.showMessageDialog(currentProject, "Commit Contract Success\n ✔", "Congratulations", Messages.getInformationIcon())

"cancel" -> println("--------cancel---------")

else -> {

Messages.showMessageDialog(currentProject, message, "ErrorInfo", Messages.getInformationIcon())

}

}

}catch(e: Exception) {

Messages.showMessageDialog(currentProject, e.toString(), "ErrorInfo", Messages.getInformationIcon())

}

}

override fun update(e: AnActionEvent) {

val project = e.project

e.presentation.isEnabledAndVisible = project != null

}

}

/\*\*

\* Created by [Rustic](mailto:118808@qq.com) @ 9/17/18 5:46 PM.

\*/

class CheckContractAction : AnAction() {

override fun actionPerformed(e: AnActionEvent) {

val file = e.getData(CommonDataKeys.VIRTUAL\_FILE) ?: return

val project = e.project ?: return

val psiDirectory = PsiManager.getInstance(project).findDirectory(file) ?: return

val psiFiles = psiDirectory.files.filter { psiFile ->

psiFile.name.endsWith(".go", true)

// && !psiFile.name.contains("\_autogen\_")

&& !psiFile.name.endsWith("\_test.go")

}

if (psiFiles.isEmpty()) {

Messages.showMessageDialog(project, " The directory does not contains \".go\" file ", "Error", Messages.getInformationIcon())

return

}

try {

CheckContract().checkFiles(psiFiles,file.path)

}catch (e: Exception) {

Messages.showMessageDialog(project, e.toString(), "Plugin Error", Messages.getInformationIcon())

}

}

override fun update(anActionEvent: AnActionEvent) {

// Set the availability based on whether a project is open

val project = anActionEvent.project

anActionEvent.presentation.isEnabledAndVisible = project != null

}

}

/\*\*

\* Created by <a href="mailto:118808@qq.com">Rustic</a> @ 9/18/18 1:25 PM.

\*/

class ContractMenuGroup : DefaultActionGroup() {

override fun update(e: AnActionEvent) {

super.update(e)

e.presentation.isEnabledAndVisible = getDirectory(e, e.project) != null

}

@Nullable

private fun getDirectory(@NotNull e: AnActionEvent?, @Nullable project: Project?): VirtualFile? {

if (e == null) {

return null

}

if (project == null) {

return null

} else {

val file = e.getData(CommonDataKeys.VIRTUAL\_FILE)

return if (file != null && file.isDirectory) {

val psiDirectory = PsiManager.getInstance(project).findDirectory(file)

if (psiDirectory == null) {

null

} else {

// TODO some additional check

file

}

} else {

null

}

}

}

}

/\*\*

\* Created by <a href="mailto:118808@qq.com">Rustic</a> @ 9/18/18 12:11 PM.

\*/

class ExecuteTestAction : AnAction() {

override fun actionPerformed(e: AnActionEvent) {

// Using the event, create and show a dialog

val currentProject = e.project

val dlgMsg = StringBuilder(e.presentation.text + " Selected!")

val dlgTitle = e.presentation.description

// If an element is selected in the editor, add info about it.

val nav = e.getData(CommonDataKeys.NAVIGATABLE)

if (nav != null) {

dlgMsg.append(String.format("\nSelected Element: %s", nav.toString()))

}

Messages.showMessageDialog(currentProject, dlgMsg.toString(), dlgTitle, Messages.getInformationIcon())

}

override fun update(e: AnActionEvent) {

val project = e.project

e.presentation.isEnabledAndVisible = project != null

}

}

/\*\*

\* Created by <a href="mailto:118808@qq.com">Rustic</a> @ 9/18/18 12:11 PM.

\*/

class GenCodeAction : AnAction() {

override fun actionPerformed(e: AnActionEvent) {

// Using the event, create and show a dialog

val currentProject = e.project ?: return

// val dlgMsg = StringBuilder(e.presentation.text + " Selected!")

// val dlgTitle = e.presentation.description

// // If an element is selected in the editor, add info about it.

// val nav = e.getData(CommonDataKeys.NAVIGATABLE)

// if (nav != null) {

// dlgMsg.append(String.format("\nSelected Element: %s", nav.toString()))

// }

// Messages.showMessageDialog(currentProject, dlgMsg.toString(), dlgTitle, Messages.getInformationIcon())

//

val file = e.getData(LangDataKeys.VIRTUAL\_FILE) ?: return

val psiDirectory = PsiManager.getInstance(currentProject).findDirectory(file) ?: return

val psiFiles = psiDirectory.files.filter { psiFile ->

psiFile.name.endsWith(".go", true)

&& !psiFile.name.contains("\_autogen\_")

&& !psiFile.name.endsWith("\_case\_test.go")

&& !psiFile.name.endsWith("\_wrap\_test.go")

}

if (psiFiles.isEmpty()) {

Messages.showMessageDialog(currentProject, " The directory does not contains \".go\" file ", "Error", Messages.getInformationIcon())

return

}

val checkObject = CheckContract()

if (!checkObject.checkFiles(psiFiles,file.path)) return

try {

val resultStr = GenerateCode().generateFile(psiDirectory, checkObject)

if (resultStr == "")

Messages.showMessageDialog(currentProject, "Generate Code Success\n ✔", "Congratulations", Messages.getInformationIcon())

else

Messages.showMessageDialog(currentProject, resultStr, "ErrorInfo:", Messages.getInformationIcon())

} catch (e: Exception) {

Messages.showMessageDialog(currentProject, e.toString(), "Plugin Error", Messages.getInformationIcon())

}

}

override fun update(e: AnActionEvent) {

val project = e.project

e.presentation.isEnabledAndVisible = project != null

}

}

class GenContractFrameWorkAction : AnAction() {

override fun actionPerformed(e: AnActionEvent) {

val currentProject = e.project ?: return

val file = e.getData(LangDataKeys.VIRTUAL\_FILE) ?: return

val psiDirectory = PsiManager.getInstance(currentProject).findDirectory(file) ?: return

val resultStr = GenContractFrameWork().showGenerateDialog(psiDirectory)

when (resultStr) {

"" -> Messages.showMessageDialog(currentProject, "New Smart Contract Success\n ✔", "Congratulations", Messages.getInformationIcon())

"cancel" -> println("--------cancel---------")

else -> {

Messages.showMessageDialog(currentProject, resultStr, "ErrorInfo", Messages.getInformationIcon())

}

}

}

override fun update(e: AnActionEvent) {

val project = e.project

e.presentation.isEnabledAndVisible = project != null

}

}

class ManagerPrivateKeyAction : AnAction() {

override fun actionPerformed(e: AnActionEvent) {

val currentProject = e.project ?: return

val file = e.getData(LangDataKeys.VIRTUAL\_FILE) ?: return

val psiDirectory = PsiManager.getInstance(currentProject).findDirectory(file) ?: return

try {

ManagePrivateKey().showGenerateDialog(psiDirectory)

} catch (e: Exception) {

Messages.showMessageDialog(currentProject, e.toString(), "Plugin Error", Messages.getInformationIcon())

}

}

override fun update(e: AnActionEvent) {

val project = e.project

e.presentation.isEnabledAndVisible = project != null

}

}

class SettingsAction : AnAction() {

override fun actionPerformed(e: AnActionEvent) {

val currentProject = e.project ?: return

val file = e.getData(LangDataKeys.VIRTUAL\_FILE) ?: return

val psiDirectory = PsiManager.getInstance(currentProject).findDirectory(file) ?: return

try {

val message = GenerateSettingsFile().showGenerateDialog(psiDirectory)

when (message) {

"" -> Messages.showMessageDialog(currentProject, "Set Settings Success\n ✔", "Congratulations", Messages.getInformationIcon())

"cancel" -> println("--------cancel---------")

else -> {

Messages.showMessageDialog(currentProject, message, "ErrorInfo", Messages.getInformationIcon())

}

}

} catch (e: Exception) {

Messages.showMessageDialog(currentProject, e.toString(), "Plugin Error", Messages.getInformationIcon())

}

}

override fun update(e: AnActionEvent) {

val project = e.project

e.presentation.isEnabledAndVisible = project != null

}

}

/\*\*

\* @program: contract-sdk

\* @description: 打包，签名，提交

\* @author: Mr.Wang

\* @create: 2018-12-07 14:49

\*\*/

@Suppress("DEPRECATION")

class CommitContract {

fun commit(psiDir: PsiDirectory, checkObj: CheckContract): String {

var resultStr = ""

// val project = psiDir.project

val name = "${checkObj.contractPackName}-${checkObj.version}"

val tarName = "$name.tar.gz"

val settingsProperties = getPropertyFile("${psiDir.project.basePath}/$settingPropertyName")

?: return "Please set settings first!"

val path = settingsProperties.getProperty(FILEOUTPUTPATH)

val file = psiDir.virtualFile

// /\*\*-----------------检查文件编码格式是否为UTF-8------------------\*/

// psiDir.files.forEach {

// if (it.originalFile.virtualFile.canonicalPath.toString().contains(".go")) {

// if (!checkTextCharset(it.virtualFile.canonicalPath.toString())) {

// return "The file: ${it.name} CharSet is not correct! \n"

// }

// if (!checkFloatType(it.virtualFile.canonicalPath.toString())) {

// return "The file: ${it.name} contract can not be used <float> Type \n"

// }

// if (!checkPanicType(it.virtualFile.canonicalPath.toString())) {

// return "The file: ${it.name} contract can not be used <panic> Type \n"

// }

// }

// }

// /\*\*-----------------检查包名不能是"std"------------------\*/

// var checkName = checkObj.contractPackName

// if (checkName.toLowerCase() == "std" && checkName.length == 3) {

// return "author:${checkObj.author} \n ->: 打包失败，包名不能是\"std\""

// }

//打包

val returnStr = genTarGzip(path, name, file.path)

if (returnStr != "") return returnStr

val pubToPriProperties = getPropertyFile(settingsProperties.getProperty(KEYSTOREPATH) + "/$pubToPriPropertyName")

?: return "Please manager private key first!"

val priKeyPath = pubToPriProperties.getProperty(checkObj.author)

if (priKeyPath == null || priKeyPath == "") {

return "author:${checkObj.author} \n ->: 对应的keystore文件不存在"

}

val form = CommitForm()

val rootPanel = form.`$$$getRootComponent$$$`()

val pw = form.textPW

val keyStoreName = form.labelName

val aliasToPubProperties = getPropertyFile(settingsProperties.getProperty(KEYSTOREPATH) + "/$aliasToPubPropertyName")

?: return "Please manager private key first!"

val keyValue = aliasToPubProperties.toMap()

keyValue.forEach { key, value ->

if (value == checkObj.author) {

keyStoreName.text = "$key"

}

}

val author = form.labelAuthor

author.text = checkObj.author

val dialog = DialogBuilder()

dialog.setTitle(" Password for signature ")

dialog.removeAllActions()

dialog.setCenterPanel(rootPanel)

dialog.setPreferredFocusComponent(pw)

form.buttonOk.addActionListener {

when {

pw.text == "" -> {

Messages.showMessageDialog(psiDir.project, " Failure to get private key \n : -> The password cannot be empty!", "ErrorInfo", Messages.getInformationIcon())

}

else -> {

try {

val priKeyByte = getPriKeyFromKeyStore(priKeyPath, pw.text)

if (priKeyByte.isEmpty()) {

// pubToPriProperties.remove(author)

resultStr = "author:${checkObj.author} \n ->: The corresponding private key does not exist"

return@addActionListener

}

val privateKeySpec = getPrivateKeySpecFromBytes(priKeyByte)

resultStr = genSignatureFile("$path/$name.signature", privateKeySpec, "$path/$tarName")

if (resultStr != "") {

return@addActionListener

}

//.tar.gz和sign文件上传到服务器

val map = mutableMapOf("$name.signature" to "$path/$name.signature", tarName to "$path/$tarName")

val pubKey = getPubKeyFromPrivateKey(privateKeySpec)

val pubKeyStr = Hex.toHexString(pubKey.abyte)

resultStr = upLoadFile(settingsProperties.getProperty(COMMITURL), map, pubKeyStr)

} catch (e: Exception) {

// pubToPriProperties.remove(author)

resultStr = if (e is BadPaddingException) {

"the password is error"

} else {

e.toString()

}

}

dialog.show()

}

}

}

form.buttonCancel.addActionListener {

resultStr = "cancel"

dialog.show()

}

dialog.show()

return resultStr

}

}

object DESCrypt {

//des加密

fun encryptDES(data: ByteArray, password: String): ByteArray {

//1.创建cipher对象 学习查看api

val cipher = Cipher.getInstance("DES")

//2.初始化cirpher（参数1：加密/解密模式）

val kf = SecretKeyFactory.getInstance("DES")

val digestSHA3 = SHA3.Digest256()

val pwb = digestSHA3.digest(password.toByteArray())

val keySpe = DESKeySpec(pwb)

val key = kf.generateSecret(keySpe)

//加密模式

cipher.init(Cipher.ENCRYPT\_MODE, key)

//3.加密/解密

return cipher.doFinal(data)

//通过Base64解决乱码问题

// return Base64.encode(encrypt)

}

object AESCrypt {

//aes加密

fun encryptAES(data: ByteArray, password: String): ByteArray {

//创建cipher对象

val cipher = Cipher.getInstance("AES")

val digestSHA3 = SHA3.Digest256()

val pwb = digestSHA3.digest(password.toByteArray())

//初始化cipher

//通过秘钥工厂生产秘钥

val keySpec = SecretKeySpec(pwb, "AES")

cipher.init(Cipher.ENCRYPT\_MODE, keySpec)

//加密、解密

return cipher.doFinal(data)

}

//aes解密

fun decryptAES(data: ByteArray, password: String): ByteArray {

//创建cipher对象

val cipher = Cipher.getInstance("AES")

val digestSHA3 = SHA3.Digest256()

val pwb = digestSHA3.digest(password.toByteArray())

//初始化cipher

//通过秘钥工厂生产秘钥

val keySpec = SecretKeySpec(pwb, "AES")

cipher.init(Cipher.DECRYPT\_MODE, keySpec)

//加密、解密

return cipher.doFinal(data)

}

}

fun genPrivateKey(): ByteArray {

val keyPairGenerator = KeyPairGenerator()

val keyPair = keyPairGenerator.generateKeyPair()

val priKey = keyPair.private as EdDSAPrivateKey

return priKey.abyte

}

fun getAddress(pubKey: EdDSAPublicKey, chainId: String): String {

val hasherSHA3256 = SHA3.Digest256()

val data = chainId.toByteArray().plus(pubKey.abyte)

val sha = hasherSHA3256.digest(data)

val hasherRIPEMD160 = RIPEMD160Digest()

hasherRIPEMD160.update(sha, 0, sha.size)

var rpd = ByteArray(hasherRIPEMD160.digestSize)

hasherRIPEMD160.doFinal(rpd, 0)

val hasher = RIPEMD160Digest()

hasher.update(rpd, 0, rpd.size)

var md = ByteArray(hasher.digestSize)

hasher.doFinal(md, 0)

val addr = rpd.plus(md.copyOfRange(0, 4))

return chainId + Base58.base58Encode(addr)

}

fun signFromByte(privKey: PrivateKey, data: ByteArray): ByteArray {

val sgr = EdDSAEngine(MessageDigest.getInstance("SHA-512"))

sgr.initSign(privKey)

sgr.update(data)

return sgr.sign()

}

fun verify(pubKey: PublicKey, message: String, data: ByteArray) {

val sgr = EdDSAEngine(MessageDigest.getInstance("SHA-512"))

sgr.initVerify(pubKey)

sgr.update(message.toByteArray())

val bool = sgr.verify(data)

println(bool)

}

fun getOrgID(orgName: String): String {

val hasherSHA3256 = SHA3.Digest256()

val data = orgName.toByteArray()

val sha = hasherSHA3256.digest(data)

val hasherRIPEMD160 = RIPEMD160Digest()

hasherRIPEMD160.update(sha, 0, sha.size)

var rpd = ByteArray(hasherRIPEMD160.digestSize)

hasherRIPEMD160.doFinal(rpd, 0)

val hasher = RIPEMD160Digest()

hasher.update(rpd, 0, rpd.size)

var md = ByteArray(hasher.digestSize)

hasher.doFinal(md, 0)

val addr = rpd.plus(md.copyOfRange(0, 4))

return "org" + Base58.base58Encode(addr)

}

/\*\*

\* @Description:

\* 如果导入的包是别名开头的，就把会名单中的对应的包名调用改成，别名调用

\* import f "fmt" -> (fmt.Print 改成 f.Print)

\* @Param:

\* @return:

\* @Author: Mr.Wang

\* @Date: 18-12-6

\*\*/

fun exchangePackage(byName: String, packageStr: String, greyListTemp: MutableList<String>) {

val packageNameList = packageStr.split("/")

val packageNameLast = packageNameList[packageNameList.lastIndex].trim()

for ((i, value) in greyListTemp.withIndex()) {

if (value.startsWith(packageNameLast)) {

val newValue = value.replace(packageNameLast, byName)

greyListTemp.removeAt(i)

greyListTemp.add(i, newValue)

}

}

}

/\*\*

\* @Description:

\* 如果导入的包是“.”开头的

\* 并且对应的包的调用存在与灰名单中，就提示错误（禁止使用 “.”)

\* @Param:

\*\*/

fun checkPackage(packageStr: String): Boolean {

val packageNameList = packageStr.split("/")

val packageNameLast = packageNameList[packageNameList.lastIndex].trim()

for (value in greyListFunction) {

if (value.startsWith(packageNameLast)) {

return false

}

}

return true

}

/\*\*

\* @Description: 获取字符转中的大写字母在转成小写返回

\* @Param:

\*\*/

fun getStrazFromAZ(inputStr: String): String {

val template = Regex("[^A-Z]")

val result = inputStr.replace(template, "")

return if (result != "") result.toLowerCase() else inputStr

}

/\*\*

\* @Description: 文件上传

\* @Param:

\*\*/

fun upLoadFile(uriStr: String, fileMap: Map<String, String>, author: String): String {

if (uriStr == "") return ""

var resultStr = ""

try {

val uriObj = URI.create(uriStr)

when (uriObj.scheme) {

"ftp" -> {

//弹出对话框，输入用户名密码

val form = FtpInfoForm()

val rootPanel = form.`$$$getRootComponent$$$`()

val pw = form.textPW

val name = form.textName

val dialog = DialogBuilder()

// dialog.setTitle(" input password to get private key ")

dialog.setCenterPanel(rootPanel)

dialog.setPreferredFocusComponent(name)

dialog.addOkAction()

dialog.addCancelAction()

dialog.setOkOperation {

dialog.show()

resultStr = ftpUpLoadFile(uriObj.host, uriObj.port, uriObj.path, name.text, pw.text, fileMap)

}

dialog.setCancelOperation {

dialog.show()

}

dialog.show()

}

"http" -> {

resultStr = httpUpLoadFile(uriObj.host, uriObj.port, uriObj.path, fileMap, author)

}

}

} catch (e: Exception) {

resultStr = if (e is IllegalArgumentException)

"uri is invalid"

else "commit error: $e"

}

return resultStr

}

/\*\*

\* @Description: 通过ftp上传文件到服务器

\* @Param:

\*\*/

fun ftpUpLoadFile(

host: String,

port: Int,

remotePath: String,

username: String?,

password: String?,

fileMap: Map<String, String>

): String {

var success = ""

val ftp = FTPClient()

try {

ftp.connect(host, port)//连接FTP服务器

//如果采用默认端口，可以使用ftp.connect(url)的方式直接连接FTP服务器

var name = username

var pw = password

if (name == "" || name == "anonymous") {

name = "anonymous"

pw = "gichain@ftp.com"

}

if (!ftp.login(name, pw)) return "Login Fail (username or password is error)"//登录

val reply = ftp.replyCode

if (!FTPReply.isPositiveCompletion(reply)) {

ftp.disconnect()

return "Ftp server is not positive"

}

fileMap.forEach { remoteFilename, localFilePath ->

val input = FileInputStream(File(localFilePath))

if (!ftp.changeWorkingDirectory(remotePath)) {

success = "The path of $remotePath is error"

return@forEach

}

if (!ftp.storeFile(remoteFilename, input)) {

success = "The $remoteFilename commit failed (maybe the files is exist)"

return@forEach

}

input.close()

}

ftp.logout()

} catch (e: IOException) {

success = e.toString()

}

return success

}

/\*\*

\* @Description: kotlin上传文件

\* @Param:

\*\*/

fun httpUpLoadFile(host: String, port: Int, path: String, fileMap: Map<String, String>, author: String): String =

runBlocking {

val client = HttpClient()

val response = client.submitFormWithBinaryData<HttpResponse>(

"http",

host,

port,

path,

formData {

append("author", author)

fileMap.forEach { (t, u) ->

appendInput(

"f",

headersOf(HttpHeaders.ContentDisposition, "filename=$t")

) { Files.newInputStream(Paths.get(u)).asInput() }

}

}

)

val b = response.content.readRemaining()

// println("status=" + response.status)

when (response.status.value) {

200 -> {

""

}

else -> {

b.readText()

}

}

}

/\*\*

\* @Description: 生成配置文件

\* @Param:

\*\*/

fun genPropertyFile(filePath: String, properties: Properties): String {

try {

properties.store(FileOutputStream(filePath), "key===value")

} catch (e: Exception) {

return e.toString()

}

return ""

}

/\*\*

\* @Description: 获取配置文件

\* @Param:

\*\*/

fun getPropertyFile(parentPath: String): Properties? {

val properties = Properties()//生成实例

return try {

properties.load(FileInputStream(parentPath))

properties

} catch (e: Exception) {

null

}

}

/\*\*

\* @Description: 从keystore文件中获取私钥字符串

\* @Param:

\*\*/

fun getPriKeyFromKeyStore(filePath: String, password: String): ByteArray {

val file = File(filePath)

return AESCrypt.decryptAES(file.readBytes(), password)

}

/\*\*

\* @Description: 生成文件

\* @Param:

\*\*/

fun genFileFromStr(savePath: String, text: String): String {

val file = File(savePath)

if (!file.exists() || file.isDirectory) {

val parent = file.parent

val parentFile = File(parent)

if (!parentFile.exists() || !parentFile.isDirectory) {

parentFile.mkdirs()

}

try {

file.createNewFile()

LocalFileSystem.getInstance().refresh(true)

} catch (e: Exception) {

return e.toString()

}

}

file.writeText(text)

return ""

}

/\*\*

\* @Description: 生成签名文件

\* @Param:

\*\*/

fun genSignatureFile(savePath: String, privateKeySpec: EdDSAPrivateKeySpec, sourcePath: String): String {

val bean = SignBean()

val pubKey = getPubKeyFromPrivateKey(privateKeySpec)

bean.pubkey = Hex.toHexString(pubKey.abyte)

bean.giAddr = getAddress(pubKey, "gi")

bean.giTestAddr = getAddress(pubKey, "gitest")

bean.devTestAddr = getAddress(pubKey, "devtest")

val privateKey = EdDSAPrivateKey(privateKeySpec)

val file = File(sourcePath)

val digestSHA3 = SHA3.Digest256()

val hashCode = digestSHA3.digest(file.readBytes())

bean.signature = Hex.toHexString(signFromByte(privateKey, hashCode))

val gson = Gson()

val jsonStr = gson.toJson(bean)

return genFileFromStr(savePath, jsonStr)

}

/\*\*

\* @Description: 打包生成 .tar.gz文件

\* @Param:

\*\*/

fun genTarGzip(savePath: String, archiveName: String, sourcePath: String): String {

return try {

val destination = File(savePath)

if (!destination.exists() || destination.isDirectory) {

val parent = destination.parent

val parentFile = File(parent)

if (!parentFile.exists() || !parentFile.isDirectory) {

parentFile.mkdirs()

}

}

val source = File(sourcePath)

val list = source.listFiles().filter { !it.name.contains("\_autogen\_") }

val arr = list.toTypedArray()

val archiver = ArchiverFactory.createArchiver(ArchiveFormat.TAR, CompressionType.GZIP)

archiver.create(archiveName, destination, \*arr)

""

} catch (e: Exception) {

"Packaging failure：$e"

}

}

/\*\*

\* @Description: 删除file

\* @Param:

\*\*/

fun removeFile(path: String): Boolean {

return try {

val file = File(path)

file.delete()

} catch (e: Exception) {

false

}

}

/\*\*

\* @Description: 统计一个字符串中，特定字符出现的个数

\* @Param:

\*\*/

fun getCharCountFromStr(str: String, char: Char): Int {

return str.toCharArray().filter {

it == char

}.size

}

/\*\*

\* @Description: 检查文本编码格式类型必需为UTF-8

\* @Param:

\*\*/

@Throws(IOException::class)

fun checkTextCharset(string: String): Boolean {

var fc: FileChannel? = null

try {

fc = FileInputStream(string).channel

val buf = fc!!.map(FileChannel.MapMode.READ\_ONLY, 0, fc.size())

//BOM 检查

if (buf[0].toInt() == 256 && buf[1].toInt() == 255) {

return false

}

if (buf[0].toInt() == 255 && buf[1].toInt() == 256) {

return false

}

val utf8 = Charset.forName("UTF-8")

val decoder = utf8.newDecoder()

val cbuf = CharBuffer.allocate((buf.limit() \* decoder.averageCharsPerByte()).toInt())

val result = decoder.decode(buf, cbuf, true)

return !result.isError

} finally {

fc?.close()

val end = System.nanoTime()

}

}

/\*\*

\* @Description: 检查文件里是否包含panic类型

\* @Param:

\*\*/

fun checkPanicType(path: String): Boolean {

val file = File(path)

val texts = file.readText()

val regex = Regex("""panic""")

val result = regex.find(texts)

if (result != null) {

return false

}

return true

}

/\*\*

\* @Description: 检查文件里是否包含panic类型

\* @Param:

\*\*/

fun smcCheck(contractPath: String): String {

val pluginPath = PathManager.getPluginsPath()

var smcCheckName = "smccheck.linux"

val os = System.getProperty("os.name")

if (os.toLowerCase().startsWith("win")) {

smcCheckName = "smccheck.exe"

} else if (os.toLowerCase().contains("mac os")) {

smcCheckName = "smccheck.mac"

} else if (os.toLowerCase().contains("linux")) {

smcCheckName = "smccheck.linux"

}

var runPath = pluginPath + "/gichain\_goland\_plugin/bin/"

val cmd = arrayOf(runPath + smcCheckName, "-s", contractPath)

var process: Process? = null

try {

val runtime = Runtime.getRuntime()

process = runtime.exec(cmd)

val br = BufferedInputStream(process.getInputStream())

var ch = 0

val text = StringBuffer("")

while ((ch) != -1) {

ch = br.read()

if (ch != -1) {

text.append(ch.toChar())

}

}

process.waitFor()

return text.toString()

} catch (e: IOException) {

return "$runPath can not find $smcCheckName"

} finally {

process?.destroy()

}

}

@Suppress("DEPRECATION")

class GenContractFrameWork {

fun showGenerateDialog(psiDirectory: PsiDirectory): String {

var resultStr = ""

val project = psiDirectory.project

val organizationPath = psiDirectory.project.basePath + "/" + organizationInfo

val form = SmartContractForm()

val rootPanel = form.`$$$getRootComponent$$$`()

val version = form.textVersion

val name = form.textName

val type = form.textType

name.addCaretListener {

val tempText = name.text

val template = Regex("[^A-Za-z0-9]")

val result = tempText.replace(template, "").capitalize()

type.text = result

type.invalidate()

// /\*\*-----------------检查包名不能是"std"------------------\*/

// if (tempText.toLowerCase() == "std" && tempText.length == 3){

// name.text = ""

// Messages.showMessageDialog(project, " Failure to generate contract framework \n : -> contract name cannot be \"std\"", "ErrorInfo", Messages.getInformationIcon())

// }

}

var tmpMap = OrganizationInfo()

val file = File(organizationPath)

val gson = Gson()

if (!file.exists() || file.readText() == "") {

tmpMap.orgMap = mapOf()

} else {

tmpMap = gson.fromJson(file.readText(), OrganizationInfo().javaClass)

}

val orgName = form.comboBoxOrgName

val organization = form.textOrgID

var temp = orgName.getEditor().getEditorComponent() as JTextField

orgName.maximumRowCount = 10

organizationList.clear()

tmpMap.orgMap.forEach {

orgName.addItem(it.key)

organization.text = it.value

organizationMapList[it.key] = it.value

organizationList.add(it.value)

}

var organizationStr = ""

if (tmpMap.orgMap.isNotEmpty()) {

organization.text = organizationList.first()

organizationStr = organization.text

orgName.getItemAt(0).toString()

}

var tempOrgName = ""

var orgID = ""

temp.addCaretListener {

tempOrgName = temp.text

orgID = ""

if (tempOrgName != "") {

orgID = getOrgID(tempOrgName)

// if (!organizationMapList.containsKey(tempOrgName)) {

// organizationList.add(orgID)

// }

organizationMapList[tempOrgName] = orgID

}

organization.text = orgID

organizationStr = organization.text.toString()

organization.invalidate()

}

organization.addCaretListener {

organizationStr = organization.text.toString()

}

val settingsProperties = getPropertyFile("${psiDirectory.project.basePath}/$settingPropertyName")

?: return "Please set settings first!"

val aliasToPubProperties = getPropertyFile(settingsProperties.getProperty(KEYSTOREPATH) + "/$aliasToPubPropertyName")

?: return "Please manager private key first!"

val mapPublic = aliasToPubProperties.toMap()

if (mapPublic.isEmpty()) return "Please manager private key first!"

val author = form.comboBoxAuthor

author.maximumRowCount = 10

mapPublic.forEach {

author.addItem(it.key)

}

var authorStr = mapPublic[author.getItemAt(0).toString()].toString()

author.addItemListener {

authorStr = mapPublic[it.item.toString()].toString()

}

val dialog = DialogBuilder()

dialog.setTitle(" generate contract framework ")

dialog.setCenterPanel(rootPanel)

dialog.setPreferredFocusComponent(name)

dialog.addOkAction()

dialog.addCancelAction()

dialog.setOkOperation {

when {

!checkStrIsValid(contractTemplate, name.text) || (name.text.toLowerCase() == "std") -> {

name.text = ""

type.text = ""

name.invalidate()

type.invalidate()

Messages.showMessageDialog(project, " Failure to generate contract framework \n : -> contract name is invalid or cannot be \"std\"", "ErrorInfo", Messages.getInformationIcon())

}

!checkStrIsValid(contractStructName, type.text) -> {

type.text = ""

type.invalidate()

Messages.showMessageDialog(project, " Failure to generate contract framework \n : -> contract type is invalid", "ErrorInfo", Messages.getInformationIcon())

}

!checkStrIsValid(contractVersionTemplate, version.text) -> {

version.text = ""

version.invalidate()

Messages.showMessageDialog(project, " Failure to generate contract framework \n : -> contract version is invalid", "ErrorInfo", Messages.getInformationIcon())

}

else -> {

val application: Application = ApplicationManager.getApplication()

application.runWriteAction(

Runnable {

resultStr = generate(psiDirectory, name.text, type.text, "v${version.text}", organizationStr, authorStr)

return@Runnable

})

dialog.show()

//save orgInfo

tmpMap.orgMap = tmpMap.orgMap.plus(Pair(tempOrgName, orgID))

val jsonStr = gson.toJson(tmpMap)

genFileFromStr(organizationPath, jsonStr)

}

}

}

dialog.setCancelOperation {

resultStr = "cancel"

dialog.show()

}

dialog.show()

return resultStr

}

private fun generate(psiDirectory: PsiDirectory, contractName: String, structName: String, version: String, contractOrg: String, contractAuthor: String): String {

val project = psiDirectory.project

val psiFileFactory = PsiFileFactory.getInstance(project)

val childDirectory = psiDirectory.subdirectories

var contractFirstDirectory: PsiDirectory? = null

childDirectory.forEach {

if (it.name == contractName) {

contractFirstDirectory = it

return@forEach

}

}

if (contractFirstDirectory == null) {

contractFirstDirectory = psiDirectory.createSubdirectory(contractName)

} else {

contractFirstDirectory!!.subdirectories.forEach {

val pointCount = it.name.filter { name -> name == '.' }

val versionPointCount = version.filter { name -> name == '.' }

if (pointCount != versionPointCount) {

return "The version is not Unified"

}

if (it.name == version) {

return "The $version is exist!"

}

}

}

val versionDirectory = contractFirstDirectory!!.createSubdirectory(version)

val contractSecondDirectory = versionDirectory.createSubdirectory(contractName)

val fileName = "$contractName.go"

// val structName = contractType.capitalize()

val fileText = """package ${structName.toLowerCase()}

import (

"$packageSDKPath"

)

//$structName This is struct of contract

//@:contract:$contractName

//@:version:${version.trim('v')}

//@:organization:$contractOrg

//@:author:$contractAuthor

type $structName struct {

sdk sdk.ISmartContract

//This is a sample field which is to store in db

//@:public:store

sampleStore string

}

//InitChain Constructor of this $structName

//@:constructor

func (${getStrazFromAZ(structName)} \*$structName) InitChain() {

//TODO

//This method is automatically selected when the block height reaches the contract effective block height.

}

//UpdateChain Constructor of this $structName

//@:constructor

func (${getStrazFromAZ(structName)} \*$structName) UpdateChain() {

//TODO

//This method is automatically selected when the block height reaches the new version contract effective block height.

}

//SampleMethod This is a sample method

//@:public:method:gas[500]

func (${getStrazFromAZ(structName)} \*$structName) SampleMethod() {

}

"""

val testFile = psiFileFactory.createFileFromText(fileName, fileText)

CodeStyleManager.getInstance(project).reformat(testFile)

contractSecondDirectory.add(testFile)

return ""

}

}

@Suppress("DEPRECATION", "LABEL\_NAME\_CLASH", "NAME\_SHADOWING")

class GenerateCode {

fun generateFile(psiDirectory: PsiDirectory, checkFile: CheckContract): String {

return try {

var tempStr = ""

val receipt = generateReceipt(psiDirectory, checkFile)

if (receipt != "") tempStr += receipt + "\n"

tempStr += generateWrap(psiDirectory, checkFile)

if (tempStr != "") tempStr += "\n"

val case = generateCase(psiDirectory, checkFile)

if (case != "") tempStr += case + "\n"

/\*\*--------------------------------------------------------\*/

val sdk = generateSdk(psiDirectory, checkFile)

if (sdk != "") tempStr += sdk + "\n"

val store = generateStore(psiDirectory, checkFile)

if (store != "") tempStr += store + "\n"

// val receipt = generateReceipt(psiDirectory, checkFile)

// if (receipt != "") tempStr += receipt + "\n"

/\*\*---------------------------------------------------------\*/

val importContract = genAllImportContractFile(psiDirectory, checkFile)

if (importContract != "") tempStr += importContract + "\n"

tempStr

} catch (e: Exception) {

e.toString()

}

}

//NewTestObject This is a function

func NewTestObject(sender sdk.IAccount) \*TestObject {

return &TestObject{obj: &${checkFile.labelMarkedStruct}{sdk: utest.UTP.ISmartContract}}

}

//transfer This is a method of TestObject

func (t \*TestObject) transfer(args ...interface{}) \*TestObject {

utest.Assert(utest.GetFlag())

contract := t.obj.sdk.Message().Contract()

t.obj.sdk.Message().Sender().(\*object.Account).SetSMC(t.obj.sdk)

if utest.TransferEx(t.obj.sdk.Message().Sender(), contract.Account().Address(), args...).ErrorCode != types.CodeOK {

return nil

}

t.resetMsg(t.obj.sdk.Message().Origins(), t.obj.sdk.Message().(\*object.Message).OutputReceipts())

return t

}

//setSender This is a method of TestObject

func (t \*TestObject) setSender(sender sdk.IAccount) \*TestObject {

t.obj.sdk = utest.SetSender(sender.Address())

return t

}

// run This is a method of TestObject

func (t \*TestObject) run(errCode uint32, f func(t \*TestObject) types.Error) {

utest.SetFlag(true)

msg := t.obj.sdk.Message()

smc := t.obj.sdk

// new message, empty input

sdkhelper.OriginNewMessage(smc, smc.Message().Contract(), smc.Message().MethodID(), nil)

t.resetMsg(t.obj.sdk.Message().Origins(), nil)

err := f(t)

utest.AssertError(err, errCode)

if err.ErrorCode == types.CodeOK {

utest.Commit()

} else {

utest.Rollback()

}

utest.SetFlag(false)

t.obj.sdk.(\*sdkimpl.SmartContract).SetMessage(msg)

newll := llstate.NewLowLevelSDB(t.obj.sdk, 0, 0)

t.obj.sdk.(\*sdkimpl.SmartContract).SetLlState(newll)

}

// runf This is a method of TestObject

func (t \*TestObject) resetMsg(origins []types.Address, receipts []types.KVPair) {

smc := t.obj.sdk

inR := smc.Message().InputReceipts()

if receipts != nil {

inR = append(inR, receipts...)

}

smc.(\*sdkimpl.SmartContract).SetMessage(object.NewMessage(smc,

smc.Message().Contract(),

smc.Message().MethodID(),

smc.Message().Items(),

smc.Message().Sender().Address(),

smc.Message().Payer().Address(),

origins,

inR))

}

// addOrigins This is a method of TestObject

func (t \*TestObject) addOrigins(newOrigins []string) {

smc := t.obj.sdk

oldO := smc.Message().Origins()

oldO = append(oldO, newOrigins...)

t.resetMsg(oldO, smc.Message().InputReceipts())

}

// emitReceipt This is a method of TestObject

func (t \*TestObject) emitReceipt(receipt interface{}) {

t.obj.sdk.Helper().ReceiptHelper().Emit(receipt)

}

func (t \*TestObject) assertReceipt(index int, value interface{}) {

outReceipts := t.obj.sdk.Message().(\*object.Message).InputReceipts()

utest.Assert(index < len(outReceipts) && index >= 0)

receipt := outReceipts[index]

name := receiptName(value)

utest.Assert(strings.HasSuffix(string(receipt.Key), name))

var r std.Receipt

err := jsoniter.Unmarshal(receipt.Value, &r)

utest.Assert(err == nil)

res, err := jsoniter.Marshal(value)

utest.Assert(err == nil)

utest.Assert(bytes.Equal(res, r.Bytes))

}

func (t \*TestObject) assertReceiptNil() {

utest.Assert(len(t.obj.sdk.Message().InputReceipts()) == 0)

}

func receiptName(receipt interface{}) string {

typeOfInterface := reflect.TypeOf(receipt).String()

if strings.HasPrefix(typeOfInterface, "std.") {

prefixLen := len("std.")

return "std::" + strings.ToLower(typeOfInterface[prefixLen:prefixLen+1]) + typeOfInterface[prefixLen+1:]

}

return typeOfInterface

}

//Set blockInfo

func (t \*TestObject) SetNextBlock(block std.Block) {

utest.NextBlockEx(1,

block.Height,

block.Time,

block.LastFee,

block.BlockHash,

block.DataHash,

block.LastBlockHash,

block.LastCommitHash,

block.LastAppHash,

block.ProposerAddress,

block.RewardAddress,

block.RandomNumber,

)

t.isSetBlock = true

return

}

$initStr

$allMethodStr"""

var testFile: PsiFile? = psiDirectory.findFile(fileName)

val application: Application = ApplicationManager.getApplication()

if (testFile != null) {

application.runWriteAction(

Runnable {

testFile!!.delete()

return@Runnable

})

}

application.runWriteAction(

Runnable {

testFile = psiFileFactory.createFileFromText(fileName, textWrap)

CodeStyleManager.getInstance(project).reformat(testFile!!)

psiDirectory.add(testFile!!)

return@Runnable

})

return ""

}

private fun getWrapAllMethodStr(checkFile: CheckContract): String {

var methodStr = ""

checkFile.contractMethodList.forEach {

methodStr += getWrapMethodStr(it) + "\n\n"

}

// checkFile.contractMethodList.forEach {

// methodStr += getTestMethodStr(it) + "\n\n"

// }

return methodStr

}

private fun getWrapMethodStr(methodBean: ContractMethodBean): String {

var resultTypeStr = ""

//判断有没有返回值

val middleCode = when (methodBean.resultTypeList.lastIndex >= 0) {

false -> {

"t.obj.${methodBean.parameterNameStr}"

}

else -> {

var returnStr = ""

for ((index, v) in methodBean.resultTypeList.withIndex()) {

returnStr += "result$index,"

resultTypeStr += "result$index ${v.text},"

}

returnStr = returnStr.trimEnd(',')

"$returnStr = t.obj.${methodBean.parameterNameStr}"

}

}

val methodParamStr = methodBean.methodDecStr.split(")")[0]

return """

//${methodBean.methodName} This is a method of TestObject

func (t \*TestObject) $methodParamStr) ($resultTypeStr err types.Error) {

err.ErrorCode = types.CodeOK

defer FuncRecover(&err)

utest.UTP.ISmartContract = t.obj.sdk

if !t.isSetBlock {

utest.NextBlock(1)

}

$middleCode

t.resetMsg(t.obj.sdk.Message().Origins(),

t.obj.sdk.Message().(\*object.Message).OutputReceipts())

t.isSetBlock = false

return

}

"""

}

private fun getTestMethodStr(methodBean: ContractMethodBean): String {

var resultTypeStr = ""

//判断有没有返回值

val middleCode = when (methodBean.resultTypeList.lastIndex >= 0) {

false -> {

"t.obj.${methodBean.parameterNameStr}"

}

else -> {

var returnStr = ""

for ((index, v) in methodBean.resultTypeList.withIndex()) {

returnStr += "result$index,"

resultTypeStr += "result$index ${v.text},"

}

returnStr = returnStr.trimEnd(',')

"$returnStr = t.obj.${methodBean.parameterNameStr}"

}

}

val methodParamStr = methodBean.methodDecStr.split(")")[0]

val newMethodParamStr = methodParamStr.split("(")[0]

return """

//${methodBean.methodName} This is a method of MySuite

func (mysuit \*MySuite) TestDemo\_${newMethodParamStr} (c \*check.C) {

utest.Init(orgID)

contractOwner := utest.DeployContract(c, contractName, orgID, contractMethods, contractInterfaces)

gls.Mgr.SetValues(gls.Values{gls.SDKKey: utest.UTP.ISmartContract}, func() {

test := NewTestObject(contractOwner)

test.setSender(contractOwner).InitChain()

mysuit.test\_${newMethodParamStr}(contractOwner, test)

})

}

"""

}

/\*\*

\* @Description: 生成case\_test

\* @Param:

\*\*/

private fun generateCase(psiDirectory: PsiDirectory, checkFile: CheckContract): String {

val project = psiDirectory.project

val psiFileFactory = PsiFileFactory.getInstance(project)

val MethodList = ArrayList<String>()

val MethodIndex = ArrayList<String>()

getCaseMethodDecList(project, checkFile, listOf()).forEach {

val methodParamStr = it.name.toString().replace("test\_", "")

val fileName = checkFile.labelMarkedStruct.toLowerCase() + "\_case\_${methodParamStr}\_test.go"

MethodList.add(fileName)

MethodIndex.add(methodParamStr)

}

if (checkFile.labelMarkedStruct == "") return "合约类不存在"

val application: Application = ApplicationManager.getApplication()

var index = 0

MethodList.forEach {

var testFile: PsiFile? = psiDirectory.findFile(it)

if (testFile == null) {

application.runWriteAction(

Runnable {

val textWrap = """package ${checkFile.filePackageName}

import (

"$packageSDKPath"

"$packagecheckPath"

"$packageUtestPath"

"$packageglsPath"

)

//${MethodIndex[index]} This is a method of MySuite

func (mysuit \*MySuite) TestDemo\_${MethodIndex[index]} (c \*check.C) {

utest.Init(orgID)

contractOwner := utest.DeployContract(c, contractName, orgID, contractMethods, contractInterfaces)

gls.Mgr.SetValues(gls.Values{gls.SDKKey: utest.UTP.ISmartContract}, func() {

test := NewTestObject(contractOwner)

test.setSender(contractOwner).InitChain()

mysuit.test\_${MethodIndex[index]}(contractOwner, test)

})

}

func (mysuit \*MySuite) test\_${MethodIndex[index]}(owner sdk.IAccount, test \*TestObject) {

//TODO

}

"""

val newFileName =

checkFile.labelMarkedStruct.toLowerCase() + "\_case\_${MethodIndex[index]}\_test.go"

testFile = psiFileFactory.createFileFromText(newFileName, textWrap)

CodeStyleManager.getInstance(project).reformat(testFile!!)

psiDirectory.add(testFile!!)

return@Runnable

})

}

index++

}

return ""

}

private fun getCaseMethodDecList(

project: Project,

checkFile: CheckContract,

existMethodNameList: List<String>

): List<GoMethodDeclaration> {

val methodList = mutableListOf<GoMethodDeclaration>()

var index = 0

checkFile.methodNameList.forEach {

val initStr = if (!checkFile.labelList.contains("constructor")) ""

else "test.setSender(contractOwner).InitChain()"

val name = "test\_$it"

if (name !in existMethodNameList) {

val method = getCaseMethodDec(project, name, initStr, checkFile.contractMethodList[index].methodDecStr)

if (method != null) {

methodList.add(method)

}

}

index += 1

}

return methodList

}

private fun getCaseMethodDec(

project: Project,

methodName: String,

initStr: String,

method: String

): GoMethodDeclaration? {

val receiver = "mysuit \*MySuite"

val listParam = listOf("owner sdk.IAccount, test \*TestObject")

val listBody = listOf(" //TODO")

val text = prepareMethodDeclarationText(receiver, methodName, listParam, listOf(), false, listBody)

return GoElementFactory.createElement<GoMethodDeclaration>(

project,

"package a; $text",

GoMethodDeclaration::class.java

)

}

/\*\*---sdk,store Code-----------------------------------------------------------------------------------------------\*/

/\*\*

\* @Description: 生成autogen\_skd

\* @Param:

\*\*/

private fun generateSdk(psiDirectory: PsiDirectory, checkFile: CheckContract): String {

val project = psiDirectory.project

val psiFileFactory = PsiFileFactory.getInstance(project)

if (checkFile.labelMarkedStruct == "") return "合约类不存在"

val fileName = checkFile.labelMarkedStruct.toLowerCase() + "\_autogen\_sdk.go"

val sdkText = """package ${checkFile.filePackageName}

import (

"$packageSDKPath"

)

//SetSdk This is a method of ${checkFile.labelMarkedStruct}

func (${getStrazFromAZ(checkFile.labelMarkedStruct)} \*${checkFile.labelMarkedStruct}) SetSdk(sdk sdk.ISmartContract) {

${getStrazFromAZ(checkFile.labelMarkedStruct)}.sdk = sdk

}

//GetSdk This is a method of ${checkFile.labelMarkedStruct}

func (${getStrazFromAZ(checkFile.labelMarkedStruct)} \*${checkFile.labelMarkedStruct}) GetSdk() sdk.ISmartContract {

return ${getStrazFromAZ(checkFile.labelMarkedStruct)}.sdk

}"""

var testFile: PsiFile? = psiDirectory.findFile(fileName)

val application: Application = ApplicationManager.getApplication()

if (testFile != null) {

application.runWriteAction(

Runnable {

testFile!!.delete()

return@Runnable

})

}

application.runWriteAction(

Runnable {

testFile = psiFileFactory.createFileFromText(fileName, sdkText)

CodeStyleManager.getInstance(project).reformat(testFile!!)

psiDirectory.add(testFile!!)

return@Runnable

})

return ""

}

/\*\*

\* @Description: 生成autogen\_store

\* @Param:

\*\*/

private fun generateStore(psiDirectory: PsiDirectory, checkFile: CheckContract): String {

val project = psiDirectory.project

val psiFileFactory = PsiFileFactory.getInstance(project)

if (checkFile.labelMarkedStruct == "") return "合约类不存在!!!"

val fileName = checkFile.labelMarkedStruct.toLowerCase() + "\_autogen\_store.go"

var testFile: PsiFile? = psiDirectory.findFile(fileName)

val application: Application = ApplicationManager.getApplication()

if (testFile != null) {

application.runWriteAction(

Runnable {

testFile!!.delete()

return@Runnable

})

}

//如果没有store或者store:cache就不生成文件

if (checkFile.storeFieldDecList.isEmpty()) return ""

val importList = mutableListOf<String?>()

var importFmt = ""

val storeTypeList = mutableListOf<String>()

checkFile.storeFieldDecList.forEach {

val type = it.goField?.type

checkType(storeTypeList, type)

if (type is GoMapType) {

importFmt = "\"fmt\""

}

}

getImportList(storeTypeList, importList)

var importFirst = ""

var importEnd = ""

val importStr = if (!importList.isEmpty() || importFmt != "") {

importFirst = "import ("

importEnd = ")"

importList.joinToString("\n")

} else {

""

}

val sdkText = """package ${checkFile.filePackageName}

$importFirst

$importFmt

$importStr

$importEnd

${getStoreAllMethodStr(checkFile)}"""

application.runWriteAction(

Runnable {

testFile = psiFileFactory.createFileFromText(fileName, sdkText)

val codeStyleManager = CodeStyleManager.getInstance(project)

codeStyleManager.reformat(testFile!!)

psiDirectory.add(testFile!!)

return@Runnable

})

return ""

}

private fun getStoreAllMethodStr(checkFile: CheckContract): String {

var methodStr = ""

checkFile.storeFieldDecList.forEach {

it.goField!!.fieldDefinitionList.forEach { child ->

methodStr += getStoreMethodStr(

checkFile.labelMarkedStruct,

child.name!!,

it.goField!!.type!!.text!!,

it.label,

it.goField!!

)

}

}

return methodStr

}

private fun getStoreMethodStr(

structName: String,

methodName: String,

type: String,

comment: String,

fieldMap: GoFieldDeclaration

): String {

var typeName = type

var isMapOther = "\"/$methodName\""

var isMapSet = ""

//如果是map的话区别对待，多了K

val tempType = fieldMap.type

val isMap = when (tempType) {

is GoMapType -> {

typeName = tempType.valueType!!.text.trim().trim('\n')

isMapOther = "fmt.Sprintf(\"/$methodName/%v\", k)"

val keyOneName = tempType.keyType!!.text.trim().trim('\n')

isMapSet = "k $keyOneName, "

var tempStr = "k $keyOneName"

val typeValue = tempType.valueType

if (typeValue is GoMapType) {

typeName = typeValue.valueType!!.text.trim().trim('\n')

isMapOther = "fmt.Sprintf(\"/$methodName/%v/%v\", k1, k2)"

val keyTwoName = typeValue.keyType!!.text.trim().trim('\n')

isMapSet = "k1 $keyOneName, k2 $keyTwoName, "

tempStr = "k1 $keyOneName, k2 $keyTwoName"

}

tempStr

}

else -> {

""

}

}

val typeNameNoPoint = typeName.trim().trim('\*').trim()

var typeIsNumberTemp = ""

val typeIsNumber = if (typeNameNoPoint.split(".").last() == "Number") {

typeIsNumberTemp = "temp := bn.N(0)"

"&temp"

} else "new($typeNameNoPoint)"

var isNotPoint = ""//返回值是不是指针的处理

val setValue = when {

type.contains("\*") -> {

"v"

}

else -> {

isNotPoint = "\*"

"&v"

}

}

var methodStartClr = ""

var methodMiddleClr = ""

var methodStartMcChk = ""

var methodMiddleMcChk = ""

var methodStartMcDel = ""

var methodMiddleMcDel = ""

val isHaveMC = when {

comment.contains("cache") -> {

methodStartClr = """

//\_clr${methodName.capitalize()} This is a method of $structName

func (${getStrazFromAZ(structName)} \*$structName) \_clr${methodName.capitalize()}($isMap){

"""

methodMiddleClr = "${getStrazFromAZ(structName)}.sdk.Helper().StateHelper().McClear($isMapOther)\n}\n"

methodStartMcChk = """

//\_McChk${methodName.capitalize()} This is a method of $structName

func (${getStrazFromAZ(structName)} \*$structName) \_McChk${methodName.capitalize()}($isMap) bool {

"""

methodMiddleMcChk =

"return ${getStrazFromAZ(structName)}.sdk.Helper().StateHelper().McCheck($isMapOther)\n}\n"

methodStartMcDel = """

//\_McDel${methodName.capitalize()} This is a method of $structName

func (${getStrazFromAZ(structName)} \*$structName) \_McDel${methodName.capitalize()}($isMap) {

"""

methodMiddleMcDel =

"${getStrazFromAZ(structName)}.sdk.Helper().StateHelper().McDelete($isMapOther)\n}\n"

"Mc"

}

else -> ""

}

val methodStartSet = """

//\_set${methodName.capitalize()} This is a method of $structName

func (${getStrazFromAZ(structName)} \*$structName) \_set${methodName.capitalize()}($isMapSet v $typeName) {

"""

val methodMiddleSet =

"${getStrazFromAZ(structName)}.sdk.Helper().StateHelper().${isHaveMC}Set($isMapOther, $setValue)\n}\n"

val methodStartGet = """

//\_$methodName This is a method of $structName

func (${getStrazFromAZ(structName)} \*$structName) \_$methodName($isMap) $typeName {

$typeIsNumberTemp

"""

val methodMiddleGet =

"return $isNotPoint${getStrazFromAZ(structName)}.sdk.Helper().StateHelper().${isHaveMC}GetEx($isMapOther, $typeIsNumber).(\*$typeNameNoPoint)\n}\n"

val methodStartCheck = """

//\_chk${methodName.capitalize()} This is a method of $structName

func (${getStrazFromAZ(structName)} \*$structName) \_chk${methodName.capitalize()}($isMap) bool {

"""

val methodMiddleCheck =

"return ${getStrazFromAZ(structName)}.sdk.Helper().StateHelper().Check($isMapOther)\n}\n"

val methodStartDelete = """

//\_del${methodName.capitalize()} This is a method of $structName

func (${getStrazFromAZ(structName)} \*$structName) \_del${methodName.capitalize()}($isMap) {

"""

val methodMiddleDelete = "${getStrazFromAZ(structName)}.sdk.Helper().StateHelper().Delete($isMapOther)\n}\n"

val methodSetStr = methodStartSet + methodMiddleSet

val methodGetStr = methodStartGet + methodMiddleGet

val methodClrStr = methodStartClr + methodMiddleClr

val methodCheckStr = methodStartCheck + methodMiddleCheck

val methodMcCheckStr = methodStartMcChk + methodMiddleMcChk

val methodDeleteStr = methodStartDelete + methodMiddleDelete

val methodMcDeleteStr = methodStartMcDel + methodMiddleMcDel

return methodGetStr + methodCheckStr + methodSetStr + methodClrStr + methodDeleteStr + "\n"

}

private fun checkType(list: MutableList<String>, type: GoType?) {

when (type) {

is GoMapType -> {

checkType(list, type.keyType)

checkType(list, type.valueType)

}

//最高只有2维的 数组或者切片

is GoArrayOrSliceType -> {

checkType(list, type.type)

}

else -> {

val typeStr = type?.text?.trim()?.trim('\*')?.trim()

list.add(typeStr ?: "")

}

}

}

/\*\*

\* @Description: 生成autogen\_receipt

\* @Param:

\*\*/

private fun generateReceipt(psiDirectory: PsiDirectory, checkFile: CheckContract): String {

val project = psiDirectory.project

val psiFileFactory = PsiFileFactory.getInstance(project)

if (checkFile.labelMarkedStruct == "") return "合约类不存在"

val fileName = checkFile.labelMarkedStruct.toLowerCase() + "\_autogen\_receipt.go"

var testFile: PsiFile? = psiDirectory.findFile(fileName)

val application: Application = ApplicationManager.getApplication()

if (testFile != null) {

application.runWriteAction(

Runnable {

testFile!!.delete()

return@Runnable

})

}

//如果没有receipt 标签就不生成文件

if ("receipt" !in checkFile.labelList) return ""

val importList = mutableListOf<String?>()

val typeList = mutableListOf<String>()

val receiptStr = getAllReceiptStr(checkFile, typeList)

getImportList(typeList, importList)

var importFirst = ""

var importEnd = ""

val importStr = if (!importList.isEmpty()) {

importFirst = "import ("

importEnd = ")"

importList.joinToString("\n")

} else {

""

}

val receiptText = """package ${checkFile.filePackageName}

$importFirst

$importStr

$importEnd

var \_ receipt = (\*${checkFile.labelMarkedStruct})(nil)

$receiptStr

"""

application.runWriteAction(

Runnable {

testFile = psiFileFactory.createFileFromText(fileName, receiptText)

CodeStyleManager.getInstance(project).reformat(testFile!!)

psiDirectory.add(testFile!!)

return@Runnable

})

return ""

}

private fun getReceiptStr(contractName: String, goMethodSpec: GoMethodSpec, typeList: MutableList<String>): String {

var fieldStr = ""

var fieldInitStr = ""

val signature = goMethodSpec.signature!!

signature.parameters.parameterDeclarationList.forEach { parameter ->

parameter.paramDefinitionList.forEach {

if (parameter.type.text !in typeList) {

typeList.add(parameter.type.text)

}

fieldStr += """${it.text.capitalize()} ${parameter.type.text} `json:"${it.text}"`

"""

fieldInitStr += """${it.text.capitalize()}: ${it.text},

"""

}

}

val methodName = goMethodSpec.text.split("(")[0]

val structName = methodName.split("emit").last().decapitalize().trim().trim('\n')

emitStruct[structName] = fieldStr

return """

//$methodName This is a method of $contractName

func (${getStrazFromAZ(contractName)} \*$contractName) ${goMethodSpec.text} {

type $structName struct {

$fieldStr

}

${getStrazFromAZ(contractName)}.sdk.Helper().ReceiptHelper().Emit(

$structName{

$fieldInitStr

},

)

}

"""

}

/\*\*----------------------------------------------------------------------------------------------------------------\*/

private fun getImportList(typeList: MutableList<String>, importList: MutableList<String?>) {

typeList.forEach {

val typeList = it.split(".")

val packPathStr = typeList.last()

val typeFirst = typeList[0].split("]").last().trim()

val alisName = if (typeFirst != packPathStr && typeFirst !in sdkTypeList) typeFirst else ""

if (packPathStr in sdkPackagePathMap) {

if (sdkPackagePathMap[packPathStr] !in importList) {

val importStr = "$alisName \"${sdkPackagePathMap[packPathStr]}\""

if (importStr !in importList)

importList.add(importStr)

}

}

}

}

private fun genImportContractFile(

psiDirectory: PsiDirectory,

checkFile: CheckContract,

importContractMethodList: MutableList<GoMethodSpec>,

importContract: String

): String {

val project = psiDirectory.project

val psiFileFactory = PsiFileFactory.getInstance(project)

if (checkFile.labelMarkedStruct == "") return "合约类不存在"

val fileName = checkFile.labelMarkedStruct.toLowerCase() + "\_autogen\_import\_${importContract.toLowerCase()}.go"

var testFile: PsiFile? = psiDirectory.findFile(fileName)

val application: Application = ApplicationManager.getApplication()

if (testFile != null) {

application.runWriteAction(

Runnable {

testFile!!.delete()

return@Runnable

})

}

//如果没有跨合约调用就不生成文件

if (importContractMethodList.isEmpty()) return ""

/\*\*---自动导包-------------------------------------------------------------------------------------------------\*/

val importList = mutableListOf<String?>()

checkFile.icParamAndResultTypeMap[importContract]?.forEach {

val typeList = it.text.split(".")

val packPathStr = typeList.last()

val typeFirst = typeList[0].split("]").last().trim().trim('\*').trim()

val alisName = if (typeFirst != packPathStr && typeFirst !in sdkTypeList) typeFirst else ""

if (packPathStr in sdkPackagePathMap) {

if (sdkPackagePathMap[packPathStr] !in importList) {

val importStr = "$alisName\"${sdkPackagePathMap[packPathStr]}\""

if (importStr !in importList)

importList.add(importStr)

}

}

}

var importFirst = ""

var importEnd = ""

val importStr = if (!importList.isEmpty()) {

importFirst = "import ("

importEnd = ")"

importList.joinToString("\n")

} else {

""

}

val structName = "Interface${importContract}Stub"

val tmp = getStrazFromAZ(structName)

val sdkText = """package ${checkFile.filePackageName}

$importFirst

$importStr

"$packageSDKPath"

"$packageObjectPath"

types2 "$packagesStuTypesPath"

types3 "$packageResponsePath"

$importEnd

type Func func()

//$structName This is a interface stub of ${checkFile.importContract}

type $structName struct {

stub types2.IContractIntfcStub

receipts []types.KVPair

}

const importContractName = "${checkFile.importContract}"

//${checkFile.importContract.toLowerCase()}Stub This is method of ${checkFile.labelMarkedStruct}

func (${getStrazFromAZ(checkFile.labelMarkedStruct)} \*${checkFile.labelMarkedStruct}) ${checkFile.importContract}() \*$structName {

return &$structName{

stub: NewInterfaceStub(${getStrazFromAZ(checkFile.labelMarkedStruct)}.GetSdk()),

receipts: make([]types.KVPair, 0),

}

}

type Intfc${checkFile.importContract}Stub struct {

smc sdk.ISmartContract

}

// NewInterfaceStub new interface stub

func NewInterfaceStub(smc sdk.ISmartContract) types2.IContractIntfcStub {

return NewIntfcStub(smc)

}

func NewIntfcStub(smc sdk.ISmartContract) types2.IContractIntfcStub {

return &Intfc${checkFile.importContract}Stub{smc: smc}

}

//GetSdk get sdk

func (intfc \*Intfc${checkFile.importContract}Stub) GetSdk() sdk.ISmartContract {

return intfc.smc

}

//SetSdk set sdk

func (intfc \*Intfc${checkFile.importContract}Stub) SetSdk(smc sdk.ISmartContract) {

intfc.smc = smc

}

//Invoke invoke function

func (intfc \*Intfc${checkFile.importContract}Stub) Invoke(methodID string, p interface{}) (response types3.Response) {

return

}

func ($tmp \*$structName) run(f Func) \*$structName {

// step 1. save old all receipts

oldReceipts := $tmp.stub.GetSdk().Message().(\*object.Message).OutputReceipts()

// step 2. run function

f()

// step3. save new all receipts

newReceipts := $tmp.stub.GetSdk().Message().(\*object.Message).OutputReceipts()

// step4. sub new all receipts off old all receipts

if len(newReceipts) > len(oldReceipts) {

$tmp.receipts = newReceipts[len(oldReceipts):]

}

return $tmp

}

func ($tmp \*$structName) contract() sdk.IContract {

return $tmp.stub.GetSdk().Helper().ContractHelper().ContractOfName(importContractName)

}

${getImportPlayerBookAllMethodStr(importContractMethodList, structName)}"""

application.runWriteAction(

Runnable {

testFile = psiFileFactory.createFileFromText(fileName, sdkText)

val codeStyleManager = CodeStyleManager.getInstance(project)

codeStyleManager.reformat(testFile!!)

psiDirectory.add(testFile!!)

return@Runnable

})

return ""

}

private fun getImportPlayerBookAllMethodStr(

importContractMethodList: MutableList<GoMethodSpec>,

receiver: String

): String {

var methodStr = ""

importContractMethodList.forEach {

methodStr += getImportPlayerBookMethodStr(receiver, it)

}

return methodStr

}

private fun getImportPlayerBookMethodStr(receiver: String, methodSpec: GoMethodSpec): String {

val signature = methodSpec.signature

val name = methodSpec.name!!.trim().trim('\n')

var paramStr = "()"

var resultStr = ""

if (signature != null) {

paramStr = signature.parameters.text.trim().trim('\n')

val resultList = mutableListOf<GoType>()

val results = signature.result?.children

results?.forEach { result ->

when (result) {

is GoParameters -> {

val params = result.parameterDeclarationList

params.forEach { parameter ->

parameter.paramDefinitionList.forEach {

resultList.add(parameter.type)

}

}

}

is GoTypeList -> {

resultList.addAll(result.typeList)

}

else -> {

if (result is GoType) {

resultList.add(result)

}

}

}

}

for ((index, value) in resultList.withIndex()) {

resultStr += "result$index ${value.text}, "

}

resultStr = resultStr.trim().trimEnd(',')

if (resultStr != "") {

resultStr = "(${resultStr.trim().trim('\n')})"

}

}

return """

//$name This is a method of $receiver

func (${getStrazFromAZ(receiver)} \*$receiver)$name$paramStr $resultStr {

return

}

"""

}

companion object {

private fun prepareMethodDeclarationText(

receiver: String,

name: String,

params: List<String>,

returnParams: List<String>,

forceWrapReturnParamsInParens: Boolean,

body: List<String>

): String {

val paramsList = "(" + StringUtil.join(params, ",") + ")"

var returnParamsList = StringUtil.join(returnParams, ",")

if (returnParams.size > 1 || forceWrapReturnParamsInParens) {

returnParamsList = "($returnParamsList)"

}

if (returnParams.isEmpty()) {

returnParamsList = ""

}

val bodyBlock = if (body.isEmpty()) "{}" else "{\n" + StringUtil.join(body, "\n") + "\n}"

return String.format("func(%s)%s%s%s%s", receiver, name, paramsList, returnParamsList, bodyBlock)

}

}

}

@Suppress("NAME\_SHADOWING", "DEPRECATION")

class GenerateKeyStoreFile {

fun showGenerateDialog(psiDir: PsiDirectory): String {

val settingsPropertyPath = "${psiDir.project.basePath}/$settingPropertyName"

var resultStr = ""

// val file = psiDir.virtualFile

val form = PrivateKeyForm()

val rootPanel = form.`$$$getRootComponent$$$`()

val textName = form.textName

val pw = form.textPW

val pwTwo = form.textPWTwo

val dialog = DialogBuilder()

dialog.setTitle(" generate keystore file ")

dialog.setCenterPanel(rootPanel)

dialog.addOkAction()

dialog.addCancelAction()

dialog.setOkOperation {

when {

textName.text == "" -> {

Messages.showMessageDialog(psiDir.project, " Failure to generate keystore file \n : -> The file name is invalid!", "ErrorInfo", Messages.getInformationIcon())

}

pw.text == "" -> {

Messages.showMessageDialog(psiDir.project, " Failure to generate keystore file \n : -> The password is invalid!", "ErrorInfo", Messages.getInformationIcon())

}

pwTwo.text != pw.text -> {

Messages.showMessageDialog(psiDir.project, " Failure to generate keystore file \n : -> The password is not equal!", "ErrorInfo", Messages.getInformationIcon())

pw.text = ""

pwTwo.text = ""

}

else -> {

val properties = getPropertyFile(settingsPropertyPath)

if (properties == null) {

resultStr = "Please set settings first!"

dialog.show()

return@setOkOperation

}

val keyStorePath = properties.getProperty(KEYSTOREPATH)

val file = File(keyStorePath)

if (!file.exists()) {

resultStr = "Please modify the keyStorePath of settings.properties!"

dialog.show()

return@setOkOperation

}

val destinationPath = keyStorePath + "/${textName.text}.ks"

val privateKeyByte = genPrivateKey()

val result = genEncryptedFile(destinationPath, privateKeyByte, "${pw.text}")

if (result != "") {

Messages.showMessageDialog(psiDir.project, " Failure to generate keystore file \n : -> $result", "ErrorInfo", Messages.getInformationIcon())

} else {

//生成公钥串和私钥文件路径的对应关系

val priKeySpec = getPrivateKeySpecFromBytes(privateKeyByte)

val pubKey = getPubKeyFromPrivateKey(priKeySpec)

val pubToPriPath = "$keyStorePath/$pubToPriPropertyName"

var pubToPri = getPropertyFile(pubToPriPath)

if (pubToPri == null) pubToPri = Properties()

val pubKeyStr = Hex.toHexString(pubKey.abyte)

pubToPri.setProperty(pubKeyStr, destinationPath)

resultStr = genPropertyFile(pubToPriPath, pubToPri)

if (resultStr != "") {

removeFile(destinationPath)

} else {

//生成公钥串和别名的对应关系

val aliasToPubPath = "$keyStorePath/$aliasToPubPropertyName"

var aliasToPub = getPropertyFile(aliasToPubPath)

if (aliasToPub == null) aliasToPub = Properties()

aliasToPub.setProperty(textName.text, Hex.toHexString(pubKey.abyte))

resultStr = genPropertyFile(aliasToPubPath, aliasToPub)

if (resultStr != "") {

removeFile(destinationPath)

}

}

dialog.show()

}

}

}

}

dialog.setCancelOperation {

resultStr = "cancel"

dialog.show()

}

dialog.show()

return resultStr

}

}

class GenerateSettingsFile {

fun showGenerateDialog(psiDir: PsiDirectory): String {

var resultStr = ""

// val file = psiDir.virtualFile

val settingsPath = psiDir.project.basePath + "/" + settingPropertyName

val form = SettingsForm()

val rootPanel = form.`$$$getRootComponent$$$`()

val keyStorePath = form.keyStorePath

val chooser = FileChooserDescriptor(false, true, false, false, false, false)

keyStorePath.addBrowseFolderListener(object : TextBrowseFolderListener(chooser) {

})

val outputPath = form.textOutput

outputPath.addBrowseFolderListener(object : TextBrowseFolderListener(chooser) {

})

val commitUrl = form.commitUrl

val properties = getPropertyFile(settingsPath)

keyStorePath.textField.text = properties?.getProperty(KEYSTOREPATH)

outputPath.textField.text = properties?.getProperty(FILEOUTPUTPATH)

commitUrl.text = properties?.getProperty(COMMITURL)

val dialog = DialogBuilder()

dialog.setTitle(" generate settings file ")

dialog.setCenterPanel(rootPanel)

dialog.addOkAction()

dialog.addCancelAction()

dialog.setOkOperation {

when {

keyStorePath.textField.text == "" -> {

Messages.showMessageDialog(psiDir.project, " Failure to generate settings file \n : -> The keystore path cannot be empty!", "ErrorInfo", Messages.getInformationIcon())

}

outputPath.textField.text == "" -> {

Messages.showMessageDialog(psiDir.project, " Failure to generate settings file \n : -> The file output path cannot be empty!", "ErrorInfo", Messages.getInformationIcon())

}

else -> {

val keyStoreSavePath = File(keyStorePath.textField.text)

val fileOutPutPath = File(outputPath.textField.text)

when {

!keyStoreSavePath.exists() || !keyStoreSavePath.isDirectory -> {

Messages.showMessageDialog(psiDir.project, " Failure to generate settings file \n : -> The keystore path is not exist!", "ErrorInfo", Messages.getInformationIcon())

}

!fileOutPutPath.exists() || !fileOutPutPath.isDirectory -> {

Messages.showMessageDialog(psiDir.project, " Failure to generate settings file \n : -> The file output path is not exist!", "ErrorInfo", Messages.getInformationIcon())

}

else -> {

val map = mutableMapOf(KEYSTOREPATH to keyStorePath.textField.text,

FILEOUTPUTPATH to outputPath.textField.text,

COMMITURL to commitUrl.text)

val props = Properties()

for ((k, v) in map) {

props.setProperty(k, v)

}

resultStr = genPropertyFile(settingsPath, props)

dialog.show()

}

}

}

}

}

dialog.setCancelOperation {

resultStr = "cancel"

dialog.show()

}

dialog.show()

return resultStr

}

}

@Suppress("NAME\_SHADOWING", "DEPRECATION", "LABEL\_NAME\_CLASH")

class ManagePrivateKey {

var width = 0

fun showGenerateDialog(psiDir: PsiDirectory) {

val settingsProperties = getPropertyFile("${psiDir.project.basePath}/$settingPropertyName")

if (settingsProperties == null) {

Messages.showMessageDialog(psiDir.project, "Please set settings first!", "ErrorInfo", Messages.getInformationIcon())

return

}

val keyStorePath = settingsProperties.getProperty(KEYSTOREPATH)

var resultStr: String

// val form = ManageKeyFormNew()

val form = ManageKeyForm()

val rootPanel = form.`$$$getRootComponent$$$`()

val screenWidth = Toolkit.getDefaultToolkit().screenSize.width

val screenHeight = Toolkit.getDefaultToolkit().screenSize.height

val ge = GraphicsEnvironment.getLocalGraphicsEnvironment()

val gs = ge.screenDevices

var minWidth = screenWidth

var minHeight = screenHeight

for (curGs in gs) {

val gc = curGs.configurations

for (curGc in gc) {

val bounds = curGc.bounds

if (minWidth > bounds.getWidth()) {

minWidth = bounds.getWidth().toInt()

}

if (minHeight > bounds.getHeight()) {

minHeight = bounds.getHeight().toInt()

}

}

}

width = minWidth

val dim = Dimension(minWidth / 10 \* 7, minHeight / 2)

rootPanel.preferredSize = dim

// val list = form.list

// list.fixedCellHeight = 30

// list.selectionMode = ListSelectionModel.SINGLE\_SELECTION

// refreshList(list, keyStorePath)

//

// list.addListSelectionListener {

// when (list.selectedIndex) {

//

// }

// }

//

// // 设置默认选中项

// list.selectedIndex = 0

// form.buttonPanel.layout = FlowLayout(FlowLayout.RIGHT, 1, 1)

val table = form.table

table.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION)

refreshTable(table, keyStorePath)

form.buttonPanel.layout = FlowLayout(FlowLayout.RIGHT, 1, 1)

rootPanel.invalidate()

val generate = form.buttonGenerate

generate.addActionListener {

resultStr = ""

val formGen = GeneratePriKeyForm()

val keyBean = getKeyBean()

var privateKeyByte = keyBean.privateKeyByte

val labelPubKey = formGen.labelPubKey

labelPubKey.text = keyBean.pubKeyStr

val labelAddr = formGen.labelAddr

labelAddr.text = keyBean.address

val textName = formGen.textName

val textPW = formGen.textPW

val dialogGen = DialogBuilder()

dialogGen.setTitle("Generate Private Key ")

dialogGen.setCenterPanel(formGen.`$$$getRootComponent$$$`())

dialogGen.setPreferredFocusComponent(textName)

dialogGen.removeAllActions()

val buttonCancel = formGen.buttonCancel

buttonCancel.addActionListener {

dialogGen.show()

}

val buttonRefresh = formGen.buttonRefresh

buttonRefresh.addActionListener {

val keyBean = getKeyBean()

privateKeyByte = keyBean.privateKeyByte

labelPubKey.text = keyBean.pubKeyStr

labelAddr.text = keyBean.address

}

val buttonOk = formGen.buttonOK

buttonOk.addActionListener {

when {

textName.text == "" -> {

Messages.showMessageDialog(psiDir.project, "The file name is invalid", "ErrorInfo", Messages.getInformationIcon())

return@addActionListener

}

textPW.text == "" -> {

Messages.showMessageDialog(psiDir.project, "The password is invalid", "ErrorInfo", Messages.getInformationIcon())

return@addActionListener

}

}

val formPW = PasswordForm()

val textPWC = formPW.textPW

val dialogPW = DialogBuilder()

dialogPW.removeAllActions()

dialogPW.setTitle("Password Confirmation ")

dialogPW.setCenterPanel(formPW.`$$$getRootComponent$$$`())

dialogGen.setPreferredFocusComponent(textPWC)

formPW.buttonOk.addActionListener {

if (textPW.text == textPWC.text) {

//

resultStr = genKeyStoreFile(keyStorePath, textName.text, textPWC.text, privateKeyByte)

if (resultStr == "") {

refreshTable(table, keyStorePath)

dialogPW.show()

dialogGen.show()

// Messages.showMessageDialog(psiDir.project, "Success\n ✔", "Congratulations", Messages.getInformationIcon())

} else {

Messages.showMessageDialog(psiDir.project, resultStr, "ErrorInfo", Messages.getInformationIcon())

}

} else {

textPWC.text = ""

Messages.showMessageDialog(psiDir.project, "The password is error", "ErrorInfo", Messages.getInformationIcon())

}

}

formPW.buttonCancel.addActionListener {

dialogPW.show()

}

dialogPW.show()

}

dialogGen.show()

}

val import = form.buttonImport

import.addActionListener {

importPrivateKey(psiDir, table, keyStorePath)

}

val export = form.buttonExport

export.addActionListener {

if (table.rowCount == 0) {

Messages.showMessageDialog(psiDir.project, "The keystore file is not exist ", "ErrorInfo", Messages.getInformationIcon())

return@addActionListener

}

// val alias = table.selectedValue.toString().split("(")[0].trim()

val alias = table.getValueAt(table.selectedRow, 0).toString().trim()

exportKeyStoreFile(psiDir.project, keyStorePath, alias)

}

val delete = form.buttonDelete

delete.addActionListener {

if (table.rowCount == 0) {

Messages.showMessageDialog(psiDir.project, "The keystore file is not exist ", "ErrorInfo", Messages.getInformationIcon())

return@addActionListener

}

// val alias = table.selectedValue.toString().split("(")[0].trim()

val alias = table.getValueAt(table.selectedRow, 0).toString().trim()

deleteKeyStoreFile(psiDir.project, table, keyStorePath, alias)

}

val dialog = DialogBuilder()

dialog.setTitle("Manage Private Key ")

dialog.setCenterPanel(rootPanel)

dialog.removeAllActions()

val cancel = form.buttonCancel

cancel.addActionListener {

dialog.show()

}

dialog.show()

}

private fun refreshList(list: JList<Any>, keyStorePath: String) {

val listStr = mutableListOf<String>()

val aliasToPubProperties = getPropertyFile("$keyStorePath/$aliasToPubPropertyName")

val mapPublic = aliasToPubProperties?.toMap()

mapPublic?.forEach { key, value ->

val pubKey = getPubKeyFromBytes(Hex.decode(value.toString()))

val address = getAddress(pubKey, "gi")

val valueStr = value.toString()

val pubKeyStr = valueStr.substring(0, 4) + "..." + valueStr.substring(valueStr.length - 5, valueStr.length - 1)

val itemStr = "$key ($address) ($pubKeyStr)"

listStr.add(itemStr)

}

list.setListData(listStr.toTypedArray())

// 设置默认选中项

list.selectedIndex = 0

}

private fun refreshTable(table: JTable, keyStorePath: String) {

val columnNames = arrayOf("Name", "Address", "PubKey")

val rowList = arrayListOf<Array<String?>>()

val aliasToPubProperties = getPropertyFile("$keyStorePath/$aliasToPubPropertyName")

val mapPublic = aliasToPubProperties?.toMap()

mapPublic?.forEach { key, value ->

val pubKey = getPubKeyFromBytes(Hex.decode(value.toString()))

val address = getAddress(pubKey, "gi")

val pubKeyStr = value.toString()

val columnVale = arrayOfNulls<String>(3)

columnVale[0] = key.toString()

columnVale[1] = address

columnVale[2] = pubKeyStr

rowList.add(columnVale)

}

val rowData = rowList.toTypedArray()

val dtm = DefaultTableModel(rowData, columnNames)

table.model = dtm

val tempWidth = width / 9

table.columnModel.getColumn(0).preferredWidth = tempWidth

table.columnModel.getColumn(1).preferredWidth = tempWidth \* 3

table.columnModel.getColumn(2).preferredWidth = tempWidth \* 5

table.tableHeader.font = Font(null, Font.BOLD, 14)

// 设置默认选中项

if (table.rowCount > 0)

table.setRowSelectionInterval(0, 0)

}

private fun genKeyStoreFile(keyStorePath: String, fileName: String, password: String, privateKeyByte: ByteArray): String {

val file = File(keyStorePath)

if (!file.exists()) {

return "Please modify the keyStorePath of settings.properties!"

}

val destinationPath = "$keyStorePath/$fileName.ks"

val result = genEncryptedFile(destinationPath, privateKeyByte, password)

if (result != "") {

return " Failure to generate keystore file \n : -> $result"

} else {

//生成公钥串和私钥文件路径的对应关系

val priKeySpec = getPrivateKeySpecFromBytes(privateKeyByte)

val pubKey = getPubKeyFromPrivateKey(priKeySpec)

val pubToPriPath = "$keyStorePath/$pubToPriPropertyName"

var pubToPri = getPropertyFile(pubToPriPath)

if (pubToPri == null) pubToPri = Properties()

val pubKeyStr = Hex.toHexString(pubKey.abyte)

pubToPri.setProperty(pubKeyStr, destinationPath)

var resultStr = genPropertyFile(pubToPriPath, pubToPri)

if (resultStr != "") {

removeFile(destinationPath)

} else {

//生成公钥串和别名的对应关系

val aliasToPubPath = "$keyStorePath/$aliasToPubPropertyName"

var aliasToPub = getPropertyFile(aliasToPubPath)

if (aliasToPub == null) aliasToPub = Properties()

aliasToPub.setProperty(fileName, Hex.toHexString(pubKey.abyte))

resultStr = genPropertyFile(aliasToPubPath, aliasToPub)

if (resultStr != "") {

removeFile(destinationPath)

}

}

}

return ""

}

private fun importPrivateKey(psiDir: PsiDirectory, table: JTable, keyStorePath: String) {

var resultStr: String

val formImport = ImportPriKeyForm()

val rootPanel = formImport.`$$$getRootComponent$$$`()

val textPW = formImport.textPW

val priKey = formImport.textPriKey

val name = formImport.textName

val dialog = DialogBuilder()

dialog.removeAllActions()

dialog.setTitle("Import Private Key ")

dialog.setCenterPanel(rootPanel)

dialog.setPreferredFocusComponent(priKey)

formImport.buttonOk.addActionListener{

when {

priKey.text == "" -> {

Messages.showMessageDialog(psiDir.project, "The private key cannot be empty!", "ErrorInfo", Messages.getInformationIcon())

}

name.text == "" -> {

Messages.showMessageDialog(psiDir.project, "The file name cannot be empty!", "ErrorInfo", Messages.getInformationIcon())

}

textPW.text == "" -> {

Messages.showMessageDialog(psiDir.project, "The password cannot be empty!", "ErrorInfo", Messages.getInformationIcon())

}

else -> {

try {

val priKeyData = Hex.decode(formImport.textPriKey.text)

val priKeySpec = getPrivateKeySpecFromBytes(priKeyData)

val savePath = "$keyStorePath/${name.text}.ks"

val pubKey = getPubKeyFromPrivateKey(priKeySpec)

val pubKeyStr = Hex.toHexString(pubKey.abyte)

val pubToPriPath = "$keyStorePath/$pubToPriPropertyName"

var pubToPri = getPropertyFile(pubToPriPath)

if (pubToPri == null) pubToPri = Properties()

if (pubToPri.getProperty(pubKeyStr) != null) {

Messages.showMessageDialog(psiDir.project, "The private key is exist!", "ErrorInfo", Messages.getInformationIcon())

priKey.text = ""

return@addActionListener

}

val formPW = PasswordForm()

val textPWC = formPW.textPW

val dialogPW = DialogBuilder()

dialogPW.removeAllActions()

dialogPW.setTitle(" Password Confirmation ")

dialogPW.setCenterPanel(formPW.`$$$getRootComponent$$$`())

dialogPW.setPreferredFocusComponent(textPWC)

formPW.buttonOk.addActionListener {

if (textPW.text == textPWC.text) {

//生成keystore文件

resultStr = genEncryptedFile(savePath, priKeyData, textPW.text)

if (resultStr == "") {

//修改aliasToPub.properties

pubToPri.setProperty(pubKeyStr, savePath)

var resultStr = genPropertyFile(pubToPriPath, pubToPri)

if (resultStr != "") {

removeFile(savePath)

Messages.showMessageDialog(psiDir.project, resultStr, "ErrorInfo", Messages.getInformationIcon())

} else {

//生成公钥串和别名的对应关系

val aliasToPubPath = "$keyStorePath/$aliasToPubPropertyName"

var aliasToPub = getPropertyFile(aliasToPubPath)

if (aliasToPub == null) aliasToPub = Properties()

aliasToPub.setProperty(name.text, pubKeyStr)

resultStr = genPropertyFile(aliasToPubPath, aliasToPub)

if (resultStr != "") {

removeFile(savePath)

Messages.showMessageDialog(psiDir.project, resultStr, "ErrorInfo", Messages.getInformationIcon())

} else {

dialogPW.show()

dialog.show()

refreshTable(table, keyStorePath)

// Messages.showMessageDialog(psiDir.project, "Success\n ✔", "Congratulations", Messages.getInformationIcon())

}

}

} else {

Messages.showMessageDialog(psiDir.project, resultStr, "ErrorInfo", Messages.getInformationIcon())

}

} else {

textPWC.text = ""

Messages.showMessageDialog(psiDir.project, "The password is error", "ErrorInfo", Messages.getInformationIcon())

}

}

formPW.buttonCancel.addActionListener {

dialogPW.show()

}

dialogPW.show()

} catch (e: Exception) {

resultStr = when (e) {

is IllegalArgumentException -> {

priKey.text = ""

"The length of private key is wrong"

}

is org.bouncycastle.util.encoders.DecoderException -> {

priKey.text = ""

"The private key must be hex"

}

else -> {

e.toString()

}

}

Messages.showMessageDialog(psiDir.project, resultStr, "ErrorInfo", Messages.getInformationIcon())

}

}

}

}

formImport.buttonCancel.addActionListener {

dialog.show()

}

dialog.show()

}

private fun exportKeyStoreFile(project: Project, keyStorePath: String, alias: String) {

val aliasToPubProperties = getPropertyFile("$keyStorePath/$aliasToPubPropertyName")

if (aliasToPubProperties == null) {

Messages.showMessageDialog(project, "Can not find the aliasToPub.properties file ", "ErrorInfo", Messages.getInformationIcon())

return

}

val pubKeyStr = aliasToPubProperties.getProperty(alias)

//修改aliasToPub.properties

val pubToPriPath = "$keyStorePath/$pubToPriPropertyName"

val pubToPri = getPropertyFile(pubToPriPath)

if (pubToPri == null) {

Messages.showMessageDialog(project, "Can not find the pubToPri.properties file ", "ErrorInfo", Messages.getInformationIcon())

return

}

val filePath = pubToPri.getProperty(pubKeyStr)

val file = File(filePath)

if (!file.exists()) {

Messages.showMessageDialog(project, "The keystore file is not exist ", "ErrorInfo", Messages.getInformationIcon())

return

}

val formPW = PasswordForm()

val textPW = formPW.textPW

val dialogPW = DialogBuilder()

dialogPW.removeAllActions()

dialogPW.setTitle("Password For Private Key")

dialogPW.setCenterPanel(formPW.`$$$getRootComponent$$$`())

dialogPW.setPreferredFocusComponent(textPW)

formPW.buttonOk.addActionListener {

if (textPW.text == "") {

Messages.showMessageDialog(project, "The password can not be empty", "ErrorInfo", Messages.getInformationIcon())

} else {

dialogPW.show()

try {

val priKeyData = getPriKeyFromKeyStore(filePath, textPW.text)

val formExport = ExportKSForm()

val labelPriKey = formExport.labelPriKey

labelPriKey.text = Hex.toHexString(priKeyData)

val dialog = DialogBuilder()

dialog.removeAllActions()

val buttonCTC = formExport.buttonCTC

buttonCTC.addActionListener {

dialog.show()

val text = StringSelection(labelPriKey.text)

val cli: Clipboard = Toolkit.getDefaultToolkit().systemClipboard

cli.setContents(text, null)

}

val buttonCancel = formExport.buttonCancel

buttonCancel.addActionListener {

dialog.show()

}

dialog.setTitle("Private Key")

dialog.setCenterPanel(formExport.`$$$getRootComponent$$$`())

dialog.show()

} catch (e: Exception) {

Messages.showMessageDialog(project, "The password is wrong", "ErrorInfo", Messages.getInformationIcon())

}

}

}

formPW.buttonCancel.addActionListener {

dialogPW.show()

}

dialogPW.show()

}

private fun deleteKeyStoreFile(project: Project, table: JTable, keyStorePath: String, aliasPubKeyDel: String) {

val aliasToPubProperties = getPropertyFile("$keyStorePath/$aliasToPubPropertyName")

if (aliasToPubProperties == null) {

Messages.showMessageDialog(project, "Can not find the aliasToPub.properties file ", "ErrorInfo", Messages.getInformationIcon())

return

}

val pubKeyStr = aliasToPubProperties.getProperty(aliasPubKeyDel)

val pubKey = getPubKeyFromBytes(Hex.decode(pubKeyStr))

val address = getAddress(pubKey, "gi")

//修改aliasToPub.properties

val pubToPriPath = "$keyStorePath/$pubToPriPropertyName"

val pubToPri = getPropertyFile(pubToPriPath)

if (pubToPri == null) {

Messages.showMessageDialog(project, "Can not find the pubToPri.properties file ", "ErrorInfo", Messages.getInformationIcon())

return

}

val filePath = pubToPri.getProperty(pubKeyStr)

val file = File(filePath)

if (!file.exists()) {

Messages.showMessageDialog(project, "The keystore file is not exist", "ErrorInfo", Messages.getInformationIcon())

return

}

val form = DeletePriKeyForm()

form.labelName.text = aliasPubKeyDel

form.labelAddr.text = address

form.labelPubKey.text = pubKeyStr

val dialog = DialogBuilder()

dialog.removeAllActions()

dialog.setTitle("Delete Private Key")

dialog.setCenterPanel(form.`$$$getRootComponent$$$`())

form.buttonOk.addActionListener {

val formPW = PasswordForm()

val textPW = formPW.textPW

val dialogPW = DialogBuilder()

dialogPW.removeAllActions()

dialogPW.setTitle("Password For Private Key")

dialogPW.setCenterPanel(formPW.`$$$getRootComponent$$$`())

dialogPW.setPreferredFocusComponent(textPW)

formPW.buttonOk.addActionListener {

if (textPW.text == "") {

Messages.showMessageDialog(project, "The password can not be empty", "ErrorInfo", Messages.getInformationIcon())

} else {

try {

if (!file.delete()) {

Messages.showMessageDialog(project, "Delete keystore file fail", "ErrorInfo", Messages.getInformationIcon())

return@addActionListener

}

pubToPri.remove(pubKeyStr)

LocalFileSystem.getInstance().refresh(true)

genPropertyFile(pubToPriPath, pubToPri)

aliasToPubProperties.remove(aliasPubKeyDel)

genPropertyFile("$keyStorePath/$aliasToPubPropertyName", aliasToPubProperties)

dialogPW.show()

dialog.show()

refreshTable(table, keyStorePath)

// Messages.showMessageDialog(project, "Success\n ✔", "Congratulations", Messages.getInformationIcon())

} catch (e: Exception) {

Messages.showMessageDialog(project, "The password is wrong", "ErrorInfo", Messages.getInformationIcon())

}

}

}

formPW.buttonCancel.addActionListener {

dialogPW.show()

}

dialogPW.show()

}

form.buttonCancel.addActionListener {

dialog.show()

}

dialog.show()

}

private fun getKeyBean(): KeyBean {

val key = KeyBean()

key.privateKeyByte = genPrivateKey()

key.privateKeySpec = getPrivateKeySpecFromBytes(key.privateKeyByte)

key.pubKey = getPubKeyFromPrivateKey(key.privateKeySpec!!)

key.pubKeyStr = Hex.toHexString(key.pubKey!!.abyte)

key.address = getAddress(key.pubKey!!, "gi")

return key

}

class KeyBean {

var privateKeyByte = byteArrayOf()

var privateKeySpec: EdDSAPrivateKeySpec? = null

var pubKey: EdDSAPublicKey? = null

var pubKeyStr = ""

var address = ""

}

}