JEB ????

JEB 反编译软件使用

- python 插件__
    - 反混淆___

python ??

python □□□□□□

python □□

???

□□ mapping.txt□□

```
#coding=utf-8
#?description=Create JEB2DeobscureClass (Android Debug Bridge) wrappers and adb utility
objects
#?shortcut=
from gettext import find
from pickle import NONE
import re
import string
from unittest import result
from com.pnfsoftware.jeb.client.api import IScript
from com.pnfsoftware.jeb.core import RuntimeProjectUtil
from com.pnfsoftware.jeb.core.units.code import ICodeUnit, ICodeItem
from com.pnfsoftware.jeb.core.units.code.android import IDexUnit
from com.pnfsoftware.jeb.core.actions import Actions, ActionContext, ActionCommentData,
ActionRenameData,ActionMoveToData,ActionMoveToPackageData,ActionCreatePackageData
from java.lang import Runnable


from com.pnfsoftware.jeb.client.api import IScript
from com.pnfsoftware.jeb.client.api import IScript, IGraphicalClientContext
from com.pnfsoftware.jeb.core import RuntimeProjectUtil
from com.pnfsoftware.jeb.core.actions import Actions, ActionContext, ActionXrefsData
from com.pnfsoftware.jeb.core.events import JebEvent, J
from com.pnfsoftware.jeb.core.output import AbstractUnitRepresentation,
UnitRepresentationAdapter
from com.pnfsoftware.jeb.core.units.code import ICodeUnit, ICodeItem
from com.pnfsoftware.jeb.core.units.code.java import IJavaSourceUnit, IJavaStaticField,
IJavaNewArray, IJavaConstant, IJavaCall, IJavaField, IJavaMethod, IJavaClass
from com.pnfsoftware.jeb.core.actions import ActionTypeHierarchyData
from com.pnfsoftware.jeb.core.actions import ActionRenameData
from com.pnfsoftware.jeb.core.util import DecompilerHelper
from com.pnfsoftware.jeb.core.output.text import ITextDocument
from com.pnfsoftware.jeb.core.units.code.android import IDexUnit
```

```
from com.pnfsoftware.jeb.core.actions import ActionOverridesData
from com.pnfsoftware.jeb.core.units import UnitUtil
from com.pnfsoftware.jeb.core.units import UnitAddress



import threading

# from Reader import Reader, Resource
import os.path, time, sys



# from Reader import Reader, Resource, allClassEntry

curPosition = 0


textLine = []

allClassEntry = {}

rlock = threading.RLock()


"""
Sample script for JEB Decompiler.
This script shows how to create and use Android Debug Bridge (adb) wrappers and adb utility
objects.
Reference: AdbWrapperFactory, AdbWrapper, AndroidDeviceUtil
"""
class DeGurd(IScript):
    def __init__(self):
        path = os.path.dirname(os.path.realpath(__file__))
        self.readerFile(path + "/../mapping.txt")

    def readerFile(self, fileName):

        starttime = time.clock()
        #□□□
        threadNum = 4
```

```
        #□□
        res = Resource(fileName)
        threads = []
        #□□□□□
        for i in range(threadNum):
            rdr = Reader(res)
            threads.append(rdr)
        #□□□□
        for i in range(threadNum):
            threads[i].start()
        #□□□□
        # result = {};
        for i in range(threadNum):
            threads[i].join()
            # result[i] = threads[i].getResult();


        # file =open('D:/data.txt','w')

        # for key in allClassEntry :
        #     file.write("class: " + key + ", origalClassName" +
allClassEntry[key].getOrigalClassName() + "\n")
        #     methods = allClassEntry.get(key).getMethodEntry();
        #     members = allClassEntry.get(key).getMemberEntry();
        #     if NONE == methods:
        #         continue
        #     for method in methods:
        #         file.write("    confusionmethod: " + method + ", origalMethodName" +
methods[method].getOrigalMethodName() + "\n")

        #     if NONE == members:
        #         continue

        #     for member in members:
        #         file.write("    confusionmember: " + member.getConfusionMemberName() + ",
origalMemberName" + member.getOrigalMemberName() + "\n")

        # file.close()
        return

    def run(self, ctx):
```

```
        ctx.executeAsync("Running deobscure class ...", JEB2AutoRename(ctx))
        print('Done')


class JEB2AutoRename(Runnable):
    def __init__(self, ctx):
        print ("__file__=%s" % __file__)
        self.ctx = ctx

         # 逻辑开始
        self.debug = 0    #0=False, 1=True
        self.ctx = ctx
        self.errMsg1 = u'请移动鼠标，将光标输入点放到源码文件中的函数名称处~然后使用F2快捷键运行此脚本(JEB 中 F2快捷键的功能是运行最近使用的脚本)'
        self.errMsg2 = u'未输入新函数名称，脚本已退出'
        # 获取所有重载了此方法的函数清单，然后逐个重命名
        self.nTotal = 0
        self.nSucc = 0
        self.nFail = 0

        # for key in allClassEntry:
        #     print("key: %s" % key)

    def run(self):
        ctx = self.ctx
        # print(allClassEntry);
        engctx = ctx.getEnginesContext()
        if not engctx:
            print('Back-end engines not initialized')
            return

        projects = engctx.getProjects()
        if not projects:
            print('There is no opened project')
            return

        prj = projects[0]

        units = RuntimeProjectUtil.findUnitsByType(prj, IDexUnit, False)
        try:
            for unit in units:
                classes = unit.getClasses()
```

```
            if not classes:
                continue
            self.parseClass(classes, unit)

    except Exception , e:
            print ( e)

# □□□
def parseClass(self, classes, unit):
     for clazz in classes:
        sourceIndex = clazz.getSourceStringIndex()
        clazzAddress = clazz.getAddress()

        # if sourceIndex == -1 or '$' in clazzAddress:# Do not rename inner class
        #     # print('without have source field', clazz.getName(True))
        #     continue

        sourceStr = str(unit.getString(sourceIndex))
        if '.java' in sourceStr:
            sourceStr = sourceStr[:-5]

        className = clazz.getName(True)

        # if  className!= sourceStr:

        package= clazz.getAddress(True)
        # print("package: %s, className: %s" % (package, className))
        package = package[1:len(package) - 1]
        package = package.replace("/", ".")

        # print("package: %s" % package)

        # completeClassName = package + "." +  className
        # if "a.b" != package:
        #     continue
        classEntry = allClassEntry.get(package)



        # print("package: %s" % package);

        if not classEntry :
```

```
                continue

            originalName =  classEntry.getOrigalClassName()
            # print(" classEntry.getOrigalClassName(): %s" %  classEntry.getOrigalClassName())

            # print("method: " % method)
            self.comment_class(unit, clazz, originalName)  # Backup origin clazz name to
comment
            self.rename_class(unit, clazz, originalName, True)  # Rename to source name

            methodsEntry = classEntry.getMethodEntry();

            self.renameMethodName(methodsEntry, clazz, unit)

            memberEntry = classEntry.getMemberEntry()

            self.renameMemberName(classEntry, clazz, unit)

            self.moveToPackage(unit, clazz, classEntry)




    # □□□□□□□
    # classEntry mapping.txt □□□□□□□□
    def renameMemberName(self, classEntry, clazz, unit):

        fieldEntry = classEntry.getMemberEntry()
        if not fieldEntry:
            print ("□□□□")
            return

        originName = None
        pos = 0
        print("\n")
        for field in clazz.getFields():


            fieldName = field.getName(True)

            # print("fieldName: %s" % fieldName)
```

```
            confusionEntry = fieldEntry.get(fieldName)

            # print(confusionEntry)

            pos = fieldName.find("$")

            if None != confusionEntry:
                originName =  confusionEntry.getOrigalMemberName();
                self.realRename(field, originName, unit)
            # elif -1 != pos:
                # 匿名内部类
                # anonymousInnerClassName = field.getAddress();
                # print("address: %s" % anonymousInnerClassName)

                # anonymousInnerClassName =
anonymousInnerClassName[1:anonymousInnerClassName.find(";")]
                # anonymousInnerClassName = anonymousInnerClassName.replace("/", ".")

                # anonymousInnerEntry = classEntry.getAnonymousInnerClassEntry();

                # anonymousInnerClass = anonymousInnerEntry.get(anonymousInnerClassName);

                # newName = anonymousInnerClass.getOrigalAnonymousInnerClassName()

                # self.realRename(field, newName, unit)


                # print ("anonymousInnerClassName: %s" % anonymousInnerClassName)


    # classEntry 从 mapping.txt 中提取的信息
    # clazz 当前字节码信息
    def renameMethodName(self, methodsEntry, clazz, unit):

        # className = classEntry.getOrigalClassName()

        if not methodsEntry:
            print ("没有方法")
            return
```

```
        # for key in methodsEntry:
        #     print ("key: %s, original: %s" % (key,
methodsEntry.get(key).getOrigalMethodName()))
        # print ("className: %s" % className)
        # □□□□□□
        for method in clazz.getMethods():
            # print("value: " % value)

            # print("method: " % method)
            confusionName = method.getName()
            # print("confusionName: %s" % (confusionName))

            current = methodsEntry.get(confusionName);
            # print (current)
            if not current:
                continue

            originName =  current.getOrigalMethodName();
            # print ("originName: %s" % originName)
            self.realRename(method, originName, unit)
            # method = clazz.getMethod(clazz.getAddress, True, True);


    def rename_class(self, unit, originClazz, sourceName, isBackup):
        actCtx = ActionContext(unit, Actions.RENAME, originClazz.getItemId(),
originClazz.getAddress())
        actData = ActionRenameData()

        # dir(actData)

        # print("sourceNameFront: %s" % sourceName)

        length = len(sourceName);

        lastPos = sourceName.rfind(".", 0, length)

        posDoller = sourceName.find("$")

        if -1 != posDoller:
            sourceName = sourceName[posDoller+1: length]
```

```
        elif -1 != lastPos:
            sourceName = sourceName[lastPos + 1: length]

        # print("sourceNameAfter: %s" % sourceName)

        actData.setNewName(sourceName)

        # print('sourceName: %s !' % sourceName)

        if unit.prepareExecution(actCtx, actData):
            try:
                result = unit.executeAction(actCtx, actData)
                # if result:
                #     print('rename to %s success!' % sourceName)
                # else:
                #     print('rename to %s failed!' % sourceName)
            except Exception, e:
                print (Exception, e)

    # Actions,
    #  ActionContext,
    #  ActionCommentData, ActionRenameData,ActionMoveToPackageData,ActionCreatePackageData
    # □□□□
    def moveToPackage(self, unit, originClazz, classEntry):

        actCntx = ActionContext(unit, Actions.CREATE_PACKAGE, originClazz.getItemId(),
originClazz.getAddress())
        actData = ActionCreatePackageData()

        packageName = classEntry.getOrigalClassName();

        packageName = packageName.replace(" ", "");

        # print("packageName: %s\n" % packageName)

        actData.setFqname   (packageName[0: packageName.rfind(".")])

        if(unit.prepareExecution(actCntx, actData)):
            try:
                bRlt = unit.executeAction(actCntx, actData)
                if(not bRlt):
```

```
                    print('executeAction fail!')
            except Exception, e:
                print (e)

        actCntx = ActionContext(unit, Actions.MOVE_TO_PACKAGE, originClazz.getItemId(),
originClazz.getAddress())
        actData = ActionMoveToPackageData ()

        confusionMemberName = classEntry.getConfusionClassName();

        confusionMemberNameTxt = confusionMemberName[0:confusionMemberName.rfind(".")]

        specifical = packageName[0: packageName.rfind(".")];

        actData.setCurrentPackageFqname("L" + confusionMemberNameTxt+"/")

        actData.setDstPackageFqname ("L" + specifical+"/")

        # print("specifical: %s, confusionMemberNameTxt: %s\n" % (specifical,
confusionMemberNameTxt))

        if(unit.prepareExecution(actCntx, actData)):
            try:
                bRlt = unit.executeAction(actCntx, actData)
            except Exception, e:
                print (e)

        print('total:%d succ:%d fail:%d' %(self.nTotal, self.nSucc, self.nFail))
        print('Done.')



        return

    def parseMethod(self,  unit, originClazz, newMethodName):
        # □□□□□□□□□□□□□□□
        # newMethodName = self.ctx.displayQuestionBox('input new function name', 'input new
function name\n\n', '')
        # if newMethodName == None:
        #     print(self.errMsg2)
        #     return
```

```
        actCntx = ActionContext(unit, Actions.QUERY_OVERRIDES, originClazz.getItemId(),
originClazz.getAddress())
        actData = ActionOverridesData()



        if(self.focusUnit.prepareExecution(actCntx, actData)):
            try:
                bRlt = self.focusUnit.executeAction(actCntx, actData)
                if(not bRlt):
                    print('executeAction fail!')
                else:
                    overrideAddrList = actData.getAddresses()
                    self.nTotal = len(overrideAddrList)
                    for addr in overrideAddrList:
                        #print('renaming %s' % addr)
                        if(self.realRename(addr, newMethodName)):
                            self.nSucc += 1
                        else:
                            self.nFail += 1
                        if(self.debug):
                            break;
            except Exception, e:
                print (e)

        print('total:%d succ:%d fail:%d' %(self.nTotal, self.nSucc, self.nFail))
        print('Done.')




    # □□□□□□□□□□□
    def realRename(self, method, newMethodName, unit):
        # m = self.GetMethodByAddress(methodAddr)
        m = method
        if(not m):
            print(u'□□ %s' % m)
            return False

        actCntx = ActionContext(unit, Actions.RENAME, m.getItemId(), m.getAddress())
        actData = ActionRenameData()
        actData.setNewName(newMethodName)
```

```
    if(unit.prepareExecution(actCntx, actData)):
    # □□□□□□□□
        try:
            bRlt = unit.executeAction(actCntx, actData)
            if(not bRlt):
                (u'□□ %s' % method.getName())
            else:
                print('%s => %s' %(method.getName(), newMethodName))
                return True
        except Exception,e:
            print (e)
    return False
## end of realRename



# def GetUnitByAddress(self, addr):
#     decomp = DecompilerHelper.getDecompiler(self.focusUnit2)
#     if not decomp:
#         print('There is no decompiler available for code unit %s' % self.focusUnit2)
#         return

#     tmpUnit = decomp.decompile(addr)
#     self.ctx.openView(tmpUnit)
#     if(self.ctx.getFocusedView().getActiveFragment().setActiveAddress(methodAddr)):
#         print('setActiveAddress succ')
#     return tmpUnit
## end of GetUnitByAddress


def GetMethodByAddress(self, addr):
    found = 0
    self.codeUnit = RuntimeProjectUtil.findUnitsByType(self.prj, ICodeUnit, False)
    if(not self.codeUnit):
        return None
    for unit in self.codeUnit:
        classes = unit.getClasses()
        if(not classes):
            continue
        for c in classes:
```

```
                cAddr = c.getAddress()
                if(not cAddr):
                    continue
                if(addr.find(cAddr) == 0):
                    mlist = c.getMethods()
                    if(not mlist):
                        continue
                    for m in mlist:
                        mAddr = m.getAddress()
                        if(addr == mAddr):
                            #print(mAddr)
                            return m
        return None


    def comment_class(self, unit, originClazz, commentStr):
        actCtx = ActionContext(unit, Actions.COMMENT, originClazz.getItemId(),
originClazz.getAddress())
        actData = ActionCommentData()
        actData.setNewComment(commentStr)

        if unit.prepareExecution(actCtx, actData):
            try:
                result = unit.executeAction(actCtx, actData)
                # if result:
                #     print('comment to %s success!' % commentStr)
                # else:
                #     print('comment to %s failed!' % commentStr)
            except Exception, e:
                print (Exception, e)


class MethodEntry:
    def __init__(self, confusionMethodName, origalMethodName):
        self.confusionMethodName = confusionMethodName
        self.origalMethodName = origalMethodName


    def getConfusionMethodName(self):
        return self.confusionMethodName
```

```
    def getOrigalMethodName(self):
        return self.origalMethodName

    def __getattribute__(self, name):

        return self[name]

# □□□□□ □□
class AnonymousInnerClass:
    def __init__(self, confusionAnonymousInnerName, origalAnonymousInnerClassName):
        self.confusionAnonymousInnerName = confusionAnonymousInnerName;
        self.origalAnonymousInnerClassName = origalAnonymousInnerClassName;

    def getConfusionAnonymousInnerName(self):
        return self.confusionAnonymousInnerName

    def getOrigalAnonymousInnerClassName(self):
        return self.origalAnonymousInnerClassName

    def setConfusionAnonymousInnerName(self, confusionAnonymousInnerName):
        self.confusionAnonymousInnerName = confusionAnonymousInnerName;

    def getOrigalAnonymousInnerClassName(self):
        return self.origalAnonymousInnerClassName


class MemberEntry:
    def __init__(self, confusionMemberName, origalMemberName):
        self.confusionMemberName = confusionMemberName
        self.origalMemberName = origalMemberName


    def getConfusionMemberName(self):
        return self.confusionMemberName

    def getOrigalMemberName(self):
        return self.origalMemberName

    def __getattribute__(self, name):
```

```
        return self[name]

class ClassEntry:
    """
        confusionClassName □□□□□□
        origalClassName □□□□□
        methodEntry:MethodEntry
    """
    def __init__(self, confusionClassName, origalClassName):

        self.confusionClassName = confusionClassName
        self.origalClassName = origalClassName
        self.methodEntry = {}
        self.memberEntry = {}
        self.anonymousInnerClass = {}

    def getConfusionClassName(self):
        return self.confusionClassName

    def setMethodEntry(self, methodEntry):
        self.methodEntry[methodEntry.getConfusionMethodName()] = (methodEntry)

    def setMemberEntry(self, memberEntry):
        self.memberEntry[memberEntry.getConfusionMemberName()] = memberEntry;

    def setAnonymousInnerClassEntry(self, anonymousInnerClass):
        self.anonymousInnerClass[anonymousInnerClass.getConfusionAnonymousInnerName()] =
anonymousInnerClass

    def getAnonymousInnerClassEntry(self):
        return self.anonymousInnerClass

    def getMemberEntry(self):
        return self.memberEntry

    def getMethodEntry(self):
        return self.methodEntry



    def getOrigalClassName(self):
        return self.origalClassName
```

```
    def __getattribute__(self, name):

        return self[name]

class Reader(threading.Thread):
    def __init__(self, res):
        self.res = res
        super(Reader, self).__init__()

        self.result = {}

        self.spaceReal = "    "

        self.spaceCount = 4

    def getResult(self):

        return self.result;
    """
    m.i$a -> tree.Handle
        m.i$a a -> d
        m.i$a b -> c
        m.i$a c -> a
        m.i$a d -> b
        m.i$a e -> RIGHT
        m.i$a[] f -> $VALUES
    """
    # spaceCount: int 方法及属性空格数量
    # line 当前行
    # classEntry 当前要组装的类
    def parseMethodAndMember(self, spaceCount, line, classEntry):

        line = line[spaceCount: len(line)]

        pos = line.find("->");

        left = line[0: pos - 1]

        firstSpacePos = left.find(" ")
```

```
        bracketsPos = left.find("(");

        if -1 != pos:
            if -1 != bracketsPos:
                # left = left.replace(" ", "")
                classEntry.setMethodEntry(MethodEntry(left[firstSpacePos+1: bracketsPos],
line[pos+3: len(line) - 1]))
            else:
                # 寻找第一个空格的位置 方法
                # print("left : %s" % i.getConfusionMemberName())
                # firstSpacePos = left.find(" ")

                right = line[pos + 3: len(line) - 1]

                # right = right.replace("\n", "")

                classEntry.setMemberEntry(MemberEntry(
                    line[firstSpacePos + 1: pos - 1], right
                ))

    #
    def parseFileLBlock(self, pos, endPosition, fstream):
        global allClassEntry
        classEntry = None
        while pos < endPosition:
            line = fstream.readline()
            space = line[0: self.spaceCount]

            if space == self.spaceReal:
                self.parseMethodAndMember(self.spaceCount, line, classEntry)

            # 说明当前是包类名
            else:
                pos = line.find("->")
                if -1 == pos:
                    continue


                  # 判断是否是匿名内部类
                # posDoller = line.find("$")
                # if -1 != posDoller:
```

```
                #     masterName = line[0: posDoller];
                #     length = len(line) - 1
                #     classEntry = allClassEntry.get(masterName)
                #     anonymousInnerClass = AnonymousInnerClass(line[0: pos-1], line[pos+3:
length])



                #     if not classEntry:
                #         classEntry = ClassEntry(masterName, line[pos+3: length])
                #         classEntry.setAnonymousInnerClassEntry(anonymousInnerClass)
                #     else:

                #         classEntry.setAnonymousInnerClassEntry(anonymousInnerClass)
                # else:
                classNam = line[0:pos-1]
                classEntry = ClassEntry(classNam, line[pos+2: len(line) - 1])
            allClassEntry[classEntry.getConfusionClassName()] = (classEntry)
            # textLine.append(line);
            #□□line
            # print(line.strip())
            pos = fstream.tell()

    #
    def run(self):
        global curPosition

        fstream = open(self.res.fileName, 'r')
        while True:
            #□□□□□□
            rlock.acquire()
            startPosition = curPosition
            curPosition = endPosition = (startPosition + self.res.blockSize) if (startPosition
+ self.res.blockSize) < self.res.fileSize else self.res.fileSize
            #□□□□□□
            rlock.release()
            if startPosition == self.res.fileSize:
                break
            elif startPosition != 0:
                fstream.seek(startPosition)
                fstream.readline()
            pos = fstream.tell()
```

```
            self.parseFileLBlock(pos, endPosition, fstream)

        fstream.close()
        # self.result = allClassEntry;
        # print(allClassEntry[classEntry.getConfusionClassName()])
        # return allClassEntry

class Resource(object):
    def __init__(self, fileName):
        self.fileName = fileName
        #□□□□
        self.blockSize = 100000000
        self.getFileSize()
    #□□□□□□
    def getFileSize(self):
        fstream = open(self.fileName, 'r')
        fstream.seek(0, os.SEEK_END)
        self.fileSize = fstream.tell()
        fstream.close()
```