### filehelper.py

import os,shutil,subprocess

class FileHelper:

def \_\_init\_\_(self):

pass

@staticmethod

def \_\_execute(cmd):

print('execute: [{}]'.format(cmd))

mytask = subprocess.Popen(cmd, shell=True)

while mytask.poll() is None:

pass

pass

if mytask.returncode == 0:

print('Command success')

else:

print('Command failed')

pass

@staticmethod

def copy\_tree(src, dst, ignore=None):

if not os.path.exists(src):

print('src path do not exists')

return

pass

names = os.listdir(src)

if not os.path.exists(dst):

os.makedirs(dst)

for name in names:

if ignore is not None and type(ignore) == type('a') and name.find(ignore) > 0:

continue

pass

srcname = os.path.join(src, name)

dstname = os.path.join(dst, name)

try:

if os.path.isdir(srcname):

FileHelper.copy\_tree(srcname, dstname)

else:

if (not os.path.exists(dstname) or ((os.path.exists(dstname)) and (os.path.getsize(dstname) != os.path.getsize(srcname)))):

print('Copy [%s]' % dstname)

shutil.copy2(srcname, dst)

pass

pass

except:

raise

pass

pass

@staticmethod

def delete\_file(file\_name):

try:

os.remove(file\_name)

print('File deleted: %s' % file\_name)

except:

print('Failed to delete file: %s' % file\_name)

pass

@staticmethod

def delete\_files(file\_name\_list):

for file\_name in file\_name\_list:

FileHelper.delete\_file(file\_name)

pass

@staticmethod

def delete\_folder(delete\_path):

del\_path = str(delete\_path).replace('/', '\\')

if os.path.exists(del\_path):

cmd\_str = 'rd /s /q %s' % del\_path

FileHelper.\_\_execute(cmd\_str)

pass

pass

@staticmethod

def delete\_file\_with\_suffix(delete\_path, suffix):

delete\_path = str(delete\_path).replace('/', '\\')

if os.path.isdir(delete\_path):

cmd\_str = 'del /a /f /q %s %s' % (delete\_path, suffix)

FileHelper.\_\_execute(cmd\_str)

Pass

### ftphelper.py

from ftplib import FTP

import os

class FtpHelper():

\_username=''

\_password=''

\_hostpath=''

\_remotepath=''

\_initialized=False

\_ftp=None

\_file\_list = []

def \_\_init\_\_(self, param):

try:

self.\_username = param['username']

self.\_password = param['password']

self.\_hostpath = param['hostpath']

self.\_remotepath = param['remotepath']

#print(param['username'])

#print(param['password'])

print('hostpath:: ' + param['hostpath'])

print('remotepath:: ' + param['remotepath'])

self.\_ftp = FTP()

except:

print('Error FtpHelper::Init Failed!')

exit(-1)

else:

self.\_initialized = True

pass

def \_\_connect(self):

print('Ftp Connecting')

if self.\_initialized is False:

print('FtpHelper must be initialized.')

return

pass

try:

self.\_ftp.connect(self.\_hostpath, 21)

self.\_ftp.login(self.\_username, self.\_password)

print('Ftp Connect!')

except:

print('Error login or connect failed!')

pass

try:

self.\_ftp.cwd(self.\_remotepath)

except:

print('Error change server dir failed')

pass

def \_\_is\_same\_size(self, local\_file, remote\_file):

try:

remotefile\_size = self.\_ftp.size(remote\_file)

except:

remotefile\_size = -1

pass

try:

localfile\_size = os.path.getsize(local\_file)

except:

localfile\_size = -1

pass

print ('lo:%d re:%d' % (localfile\_size, remotefile\_size))

return True if remotefile\_size == localfile\_size else False

def login(self):

self.\_\_connect()

pass

def quit(self):

self.\_ftp.close()

pass

def get\_file\_list(self, line):

file\_arr = self.get\_filename(line)

#print('file\_arr = %s' % line)

if file\_arr[1] not in ['.', '..']:

self.\_file\_list.append(file\_arr)

def get\_filename(self, line):

file\_type = 0 if line[0]=='d' else -1#line.find('<DIR>')

file\_name = line.split(" ")[-1]

file\_arr = [file\_type, file\_name]

return file\_arr

def download\_file(self, local\_file, remote\_path):

print('>>>>>>>>>>>>下载文件 %s ... ...' % local\_file)

file\_handler = open(local\_file, 'wb')

self.\_ftp.retrbinary('RETR %s' % remote\_path, file\_handler.write)

file\_handler.close()

pass

def download\_files(self, local\_path='./', remote\_path='./'):

print('>>>>>>>>>>>>下载文件夹 %s ... ...' % remote\_path)

try:

self.\_ftp.cwd(remote\_path)

except:

print('目录%s不存在' % remote\_path)

return

if os.path.exists(local\_path):

from filehelper import FileHelper

FileHelper.delete\_folder(local\_path)

pass

if not os.path.isdir(local\_path):

os.makedirs(local\_path)

pass

self.\_file\_list = []

self.\_ftp.dir(self.get\_file\_list)

remotenames = self.\_file\_list

for item in remotenames:

filetype = item[0]

filename = item[1]

local = os.path.join(local\_path, filename)

if filetype != -1:

self.download\_files(local, filename)

else:

self.download\_file(local, filename)

pass

pass

self.\_ftp.cwd('..')

pass

def upload\_file(self, local\_file, remote\_file, ingore=None):

if not os.path.isfile(local\_file):

return

pass

print('[========================upload\_file=========================]')

print('local\_file = [%s]' % local\_file)

print('remote\_file = [%s]' % remote\_file)

print('ingore = %s' % ingore)

if ingore is not None and (local\_file.find(ingore) != -1):

return

# if self.\_\_is\_same\_size(local\_file, remote\_file):

# print('Has same size file already exist: %s' % local\_file)

# return

# pass

pass

file\_handler = open(local\_file, 'rb')

self.\_ftp.storbinary('STOR %s' % remote\_file, file\_handler)

file\_handler.close()

pass

def upload\_files(self, local\_path='./', remote\_path='./', ingore=None):

if not os.path.isdir(local\_path):

return

self.\_ftp.cwd(remote\_path)

local\_names = os.listdir(local\_path)

for item in local\_names:

src = os.path.join(local\_path, item)

if os.path.isdir(src):

try:

self.\_ftp.mkd(item)

except:

print('folder has already exist: %s' % item)

pass

self.upload\_files(src, item, ingore)

else:

self.upload\_file(src, item, ingore)

pass

pass

self.\_ftp.cwd('..')

pass

# 创建检查ftp 是否存在结构，

def create\_dir(self, path):

try:

self.\_ftp.cwd(path)

except:

sub\_path\_list = path.split("/")

dest\_path = '/'

for sub\_path in sub\_path\_list:

dest\_path = '%s/%s' % (dest\_path, sub\_path)

try:

self.\_ftp.cwd(dest\_path)

except:

self.\_ftp.mkd(dest\_path)

pass

pass

self.\_ftp.cwd("/")

pass

def cwd\_front(self):

print('%s' % self.\_ftp.pwd())

self.\_ftp.cwd('..')

pass

def cwd(self, cmd):

self.\_ftp.cwd(cmd)

pass

def file\_exist(self, path):

bRet = False

try:

self.\_ftp.cwd(path)

bRet = True

except:

pass

return bRet

pass

### svnhelper.py

import os

from tera\_utility import TeraUtility

class SvnHelper:

initialized=False

username=None

password=None

svnpath=None

parentpath=None

def \_\_init\_\_(self, param):

try:

self.username = param['username']

self.password = param['password']

self.svnpath = param['svnpath']

self.parentpath = param['parentpath']

#print('username = %s' % param['username'])

#print('password = %s' % param['password'])

#print('svnpath = %s' % param['svnpath'])

#print('parentpath = %s' % param['parentpath'])

except:

print('Error SvnHelp::Init Failed!')

exit(-1)

else:

self.initialized = True

pass

def svn\_process(self, cmd):

if not self.initialized:

return

print('Not Coding finish! Can not use this function!')

pass

def show\_svn\_log(self, process):

if process is None:

return

while process.poll() is None:

line = process.stdout.readline()

line = line.strip()

if line:

print('Subprogram output: [{}]'.format(line))

pass

pass

if process.returncode == 0:

print('Subprogram success')

else:

print('Subprogram failed')

pass

pass

def commit(self, dest\_path, msg='python svnhelper commit'):

if not self.initialized:

return

print('svn commit')

strCmd = 'svn ci %s -m \'%s\' --username %s --password %s' % (dest\_path, msg, self.username, self.password)

TeraUtility.execute(strCmd)

pass

def checkout(self, revision=None):

if not self.initialized:

return

strCmd='svn co %s -r %s %s --username %s --password %s' % (self.svnpath, 'HEAD' if (revision is None or type(revision) != type('a')) else revision, self.parentpath, self.username, self.password)

TeraUtility.execute(strCmd)

pass

def update(self):

if not self.initialized:

return

print('svn update')

strCmd = 'svn up %s --username %s --password %s' % (self.parentpath ,self.username, self.password)

TeraUtility.execute(strCmd)

pass

def cleanup(self):

if not self.initialized:

return

print('svn cleanup')

if not os.path.exists(self.parentpath):

print('dest path not exist!')

return

strCmd = 'svn cleanup %s --username %s --password %s' % (self.parentpath ,self.username, self.password)

TeraUtility.execute(strCmd)

pass

def revert(self):

if not self.initialized:

return

print('svn revert')

strCmd = 'svn revert -R %s --username %s --password %s' % (self.parentpath, self.username, self.password)

TeraUtility.execute(strCmd)

pass

def add(self, dest\_path):

if not self.initialized:

return

print('svn add')

strCmd = 'svn add %s --username %s --password %s --force' % (dest\_path, self.username, self.password)

TeraUtility.execute(strCmd)

pass

def delete(self, dest\_path, del\_folder=True):

if not self.initialized:

return

print('svn delete')

strCmd =''

if del\_folder:

strCmd = 'svn del %s --username %s --password %s --force' % (dest\_path, self.username, self.password)

else:

strCmd = 'svn del %s/\* --username %s --password %s --force' % (dest\_path, self.username, self.password)

pass

TeraUtility.execute(strCmd)

Pass

### tera\_config.py

################################################################

# ReadMe：

# 1.Python环境 Anaconda Python3.6

# 2.Unity环境 UnityEditorGroup 5.3.6p3

# 3.使用者机器参数 需要手动设置; 修改以下参数 即可完成打包工程配置

################################################################

## AutoBuild 工程路径

Content\_AutoBuild\_Python\_Workspace\_Path = 'X:\\Python\_Workspace'

Content\_AutoBuild\_Tera\_Workspace\_Path = 'X:\\\_' #'X:\\Tera\_Workspace'

## Unity 相关

Content\_Unity\_Group\_Path = 'X:\\UnityEditorGroup'

## Android SDK 相关

Content\_Android\_Platforms\_Path = 'X:\\AndroidEnv\\android-sdk\\platforms'

Content\_Android\_All\_Platforms\_Path = 'X:\\AndroidEnv\\all\_android\_api\_platforms'

## Patch 相关

Content\_Patch\_Username = 'Patch\_Username'

Content\_Patch\_Password = 'Patch\_Password'

Content\_Patch\_Host\_Url = '10.35.49.163'

Content\_Patch\_Remote\_Path = '/Patch\_Remote\_Path/Tera'

## Svn 相关

Content\_Svn\_Client\_Url = 'svn://10.35.49.171/M1Client'

Content\_Svn\_Art\_Url = 'svn://10.35.49.171/M1Res4Build'

Content\_Svn\_Username = 'Svn\_Username'

Content\_Svn\_Password = 'Svn\_Password'

## php完成信息 Post Url

Content\_Post\_Finish\_Url = 'http://10.35.51.30/buildcmd\_finish'

## ftp backup Assetbundle 相关

Content\_Ftp\_Backup\_Assetbundle\_Username = 'Assetbundle\_Username'

Content\_Ftp\_Backup\_Assetbundle\_Password = 'Assetbundle\_Password'

Content\_Ftp\_Backup\_Assetbundle\_Host\_Url = '10.35.51.37'

Content\_Ftp\_Backup\_Assetbundle\_Remote\_Path = 'M1Res4Build\_Export'

## ftp backup Package 相关

Content\_Ftp\_Backup\_Package\_Username = 'Package\_Username'

Content\_Ftp\_Backup\_Package\_Password = 'Package\_Password'

Content\_Ftp\_Backup\_Package\_Host\_Url = '10.35.51.37'

Content\_Ftp\_Backup\_Package\_Remote\_Path = 'Package\_Backup'

Content\_Ftp\_Backup\_Dsyms\_Remote\_Path = "dsyms"

## ftp backup Patch 相关

Content\_Ftp\_Backup\_Patch\_Username = 'Backup\_Patch\_Username'

Content\_Ftp\_Backup\_Patch\_Password = 'Backup\_Patch\_Password'

Content\_Ftp\_Backup\_Patch\_Host\_Url = '10.35.51.37'

Content\_Ftp\_Backup\_Patch\_Remote\_Path = 'Patch\_Backup'

## MacOS终端 ssh信息 IP:['用户名','密码','端口号','workspace','library\_path', 'unity\_path']

Content\_MacOS\_Data = {'10.35.51.41':['usrname',

'password',

'22',

'/Users/tera/Documents/Tera\_Workspace',

'/Users/tera/Library',

'/Applications/Unity'],

'10.35.51.40': ['usrname',

'password',

'22',

'/Users/lantu/Documents/Tera\_Workspace',

'/Users/lantu/Library',

'/Applications/Unity'],

}

Content\_Android\_Api = {'kakao':[28],

'dev':[26],

}

### tera\_globals.py

# 全局数据定义

param = None

def log(log\_str):

print('Log:[ %s ]' % log\_str)

param.file\_log\_handler.flush()

### tera\_param.py

from enum import Enum

import pandas as pd

from tera\_config import \*

# 需要配置盘符 及其路径一致,可直接使用环境备份包 安装

# Unity信息

class UnityParam:

def \_\_init\_\_(self):

self.\_unity\_windows\_path = '%s\\Unity\_Windows\\Editor\\Unity.exe' % Content\_Unity\_Group\_Path

self.\_unity\_ios\_path = '%s\\Unity\_iOS\\Editor\\Unity.exe' % Content\_Unity\_Group\_Path

self.\_unity\_android\_path = '%s\\Unity\_Android\\Editor\\Unity.exe' % Content\_Unity\_Group\_Path

self.\_real\_path = None

pass

@property

def real\_path(self):

return self.\_real\_path

'''

@property

def windows\_path(self):

return self.\_unity\_windows\_path

@property

def ios\_path(self):

return self.\_unity\_ios\_path

@property

def android\_path(self):

return self.\_unity\_android\_path

'''

# 修复平台UnityEditor路径

def fix\_unity\_path\_by\_platform(self, platform\_type):

if 'iOS' == platform\_type:

self.\_real\_path = self.\_unity\_ios\_path

elif 'Android' == platform\_type:

self.\_real\_path = self.\_unity\_android\_path

else:

self.\_real\_path = self.\_unity\_windows\_path

pass

pass

# Class UnityParam End

pass

# 更新工作路径 & 上传信息

class PatchParam:

def \_\_init\_\_(self):

self.\_workspace\_path = '%s\\Update\_Workspace' % Content\_AutoBuild\_Tera\_Workspace\_Path

self.\_username = Content\_Patch\_Username

self.\_password = Content\_Patch\_Password

self.\_host\_url = Content\_Patch\_Host\_Url

self.\_remote\_path = Content\_Patch\_Remote\_Path

pass

@property

def workspace\_path(self):

return self.\_workspace\_path

@property

def username(self):

return self.\_username

@property

def password(self):

return self.\_password

@property

def host\_url(self):

return self.\_host\_url

@property

def remote\_path(self):

return self.\_remote\_path

# Class PatchParam End

pass

# svn信息

class VersionParam:

def \_\_init\_\_(self):

self.\_base\_version = None # 4.项目基础版本号('1.0.0')

self.\_last\_version = None # 5.上个 [程序版本号]

self.\_current\_version = None # 6.当前 [程序版本号]

self.\_client\_revision = None # 7.客户端 当前 [svn版本号]

self.\_art\_revision = None # 8.美术资源 当前 [svn版本号]

self.\_client\_last\_revision = None # 9.客户端 上个 [svn版本号]

self.\_is\_smallpack = '0' # 10.是否为小包(空app,不包含数据 & 资源)

self.\_is\_build\_art = True # 11.是否需要编译美术资源(AssetBundles)

self.\_complatform = None # 12.渠道名称('cn-dev' / ..)

pass

# 项目基础版本号('1.0.0')

def \_\_get\_base\_version(self):

return self.\_base\_version

def \_\_set\_base\_version(self, base\_version):

assert (isinstance(base\_version, str) and len(base\_version) > 0)

self.\_base\_version = base\_version

pass

# 上个 [程序版本号]

def \_\_get\_last\_version(self):

return self.\_last\_version

def \_\_set\_last\_version(self, last\_version):

assert (isinstance(last\_version, str) and len(last\_version) > 0)

self.\_last\_version = last\_version

pass

# 当前 [程序版本号]

def \_\_get\_current\_version(self):

return self.\_current\_version

def \_\_set\_current\_version(self, current\_version):

assert (isinstance(current\_version, str) and len(current\_version) > 0)

self.\_current\_version = current\_version

pass

# 客户端 当前 [svn版本号]

def \_\_get\_client\_revision(self):

return self.\_client\_revision

def \_\_set\_client\_revision(self, client\_revision):

assert (isinstance(client\_revision, str) and len(client\_revision) > 0)

self.\_client\_revision = client\_revision

pass

# 客户端 上个 [svn版本号]

def \_\_get\_client\_last\_revision(self):

return self.\_client\_last\_revision

def \_\_set\_client\_last\_revision(self, client\_last\_revision):

assert (isinstance(client\_last\_revision, str) and len(client\_last\_revision) > 0)

self.\_client\_last\_revision = client\_last\_revision

pass

# 美术资源 当前 [svn版本号]

def \_\_get\_art\_revision(self):

return self.\_art\_revision

def \_\_set\_art\_revision(self, art\_revision):

assert (isinstance(art\_revision, str) and len(art\_revision) > 0)

self.\_art\_revision = art\_revision

pass

# 是否为小包(空app,不包含数据 & 资源)

def \_\_get\_is\_smallpack(self):

return self.\_is\_smallpack

def \_\_set\_is\_smallpack(self, is\_smallpack):

assert (isinstance(self.\_is\_smallpack, str) and len(self.\_is\_smallpack) > 0)

self.\_is\_smallpack = is\_smallpack

pass

# 是否需要编译美术资源(AssetBundles)

def \_\_get\_is\_build\_art(self):

return self.\_is\_build\_art

def \_\_set\_is\_build\_art(self, is\_build\_art):

assert (isinstance(is\_build\_art, str) and len(is\_build\_art) > 0)

self.\_is\_build\_art = is\_build\_art

pass

# 渠道名称('cn-dev' / ..)

def \_\_get\_complatform(self):

return self.\_complatform

def \_\_set\_complatform(self, complatform):

assert (isinstance(complatform, str) and len(complatform) > 0)

self.\_complatform = complatform

pass

base\_version = property(\_\_get\_base\_version, \_\_set\_base\_version)

last\_version = property(\_\_get\_last\_version, \_\_set\_last\_version)

current\_version = property(\_\_get\_current\_version, \_\_set\_current\_version)

client\_revision = property(\_\_get\_client\_revision, \_\_set\_client\_revision)

client\_last\_revision = property(\_\_get\_client\_last\_revision, \_\_set\_client\_last\_revision)

art\_revision = property(\_\_get\_art\_revision, \_\_set\_art\_revision)

is\_smallpack = property(\_\_get\_is\_smallpack, \_\_set\_is\_smallpack)

is\_build\_art = property(\_\_get\_is\_build\_art, \_\_set\_is\_build\_art)

complatform = property(\_\_get\_complatform, \_\_set\_complatform)

# Class VersionParam End

pass

# 输入类型枚举

class EnumInputParam(Enum):

EProjet\_Name = 1

EPlatform\_Type = 2

EBranch\_Path = 3

EBase\_Version = 4

ELast\_Version = 5

ECurrent\_Version = 6

EClient\_Revision = 7

EArt\_Revision = 8

EClient\_Last\_Revision = 9

EIs\_Smallpack = 10

EIs\_Build\_Art = 11

EComplatform = 12

EBuild\_Id = 13

EArt\_Backup\_Versoin = 14

# TODO 添加参数请注意

# MacOS ip 规则必须在最后以为参数

EMacOS\_Ip = 15

pass

# 输入的参数

class InputParam:

"""

Param: EnumInputParam

# 1.项目名称

# 2.平台名称(Windows/Android/iOS)

# 3.项目分支路径('trunk' / 'branches/kakao-1.0.0')

# 4.项目基础版本号('1.0.0')

# 5.上个 [程序版本号]

# 6.当前 [程序版本号]

# 7.客户端 当前 [svn版本号]

# 8.美术资源 当前 [svn版本号]

# 9.客户端 上个 [svn版本号]

# 10.是否为小包(空app,不包含数据 & 资源)

# 11.是否需要编译美术资源(AssetBundles)

# 12.渠道名称('cn-dev' / ..)

# 13.AutoBuild开启的序号,用于对应 MySQL中数据对应

# 14.MacOS 终端 IP

"""

def \_\_init\_\_(self):

self.\_project\_name = None # 1.项目名称

self.\_platform\_type = None # 2.平台名称(Windows/Android/iOS)

self.\_branch\_path = None # 3.项目分支路径('trunk' / 'branches/kakao-1.0.0')

# 版本信息

self.\_version\_param = VersionParam()# 4-12

self.\_build\_Id = None # 13.AutoBuild开启的序号,用于对应 MySQL中数据对应

self.\_macos\_ip = None # 14.MacOS 终端 IP

self.\_art\_backup\_version = None # 15.上一次备份的资源,不需要从打的情况

pass

# 项目名称

def \_\_get\_project\_name(self):

return self.\_project\_name

def \_\_set\_project\_name(self, project\_name):

assert (isinstance(project\_name, str) and len(project\_name) > 0)

self.\_project\_name = project\_name

pass

# 平台名称(Windows/Android/iOS)

def \_\_get\_platform\_type(self):

return self.\_platform\_type

def \_\_set\_platform\_type(self, platform\_type):

assert (isinstance(platform\_type, str) and len(platform\_type) > 0)

self.\_platform\_type = platform\_type

pass

# 项目分支路径('trunk' / 'branches/kakao-1.0.0')

def \_\_get\_branch\_path(self):

return self.\_branch\_path

def \_\_set\_branch\_path(self, branch\_path):

assert (isinstance(branch\_path, str) and len(branch\_path) > 0)

self.\_branch\_path = branch\_path

pass

# AutoBuild开启的序号,用于对应 MySQL中数据对应

def \_\_get\_build\_Id(self):

return self.\_build\_Id

def \_\_set\_build\_Id(self, build\_Id):

assert (isinstance(build\_Id, str) and len(build\_Id) > 0)

self.\_build\_Id = build\_Id

pass

# 14.MacOS 终端 IP

def \_\_get\_macos\_ip(self):

return self.\_macos\_ip

def \_\_set\_macos\_ip(self, macos\_ip):

assert (isinstance(macos\_ip, str) and len(macos\_ip) > 0)

self.\_macos\_ip = macos\_ip

pass

# # 15.上一次备份的资源,不需要从打的情况

def \_\_get\_art\_backup\_version(self):

return self.\_art\_backup\_version

def \_\_set\_art\_backup\_version(self, version):

assert (isinstance(version, str) and len(version) > 0)

self.\_art\_backup\_version = version

pass

# 版本信息

@property

def version\_param(self):

return self.\_version\_param

project\_name = property(\_\_get\_project\_name, \_\_set\_project\_name)

platform\_type = property(\_\_get\_platform\_type, \_\_set\_platform\_type)

branch\_path = property(\_\_get\_branch\_path, \_\_set\_branch\_path)

build\_Id = property(\_\_get\_build\_Id, \_\_set\_build\_Id)

macos\_ip = property(\_\_get\_macos\_ip, \_\_set\_macos\_ip)

art\_backup\_version = property(\_\_get\_art\_backup\_version, \_\_set\_art\_backup\_version)

# Class InputParam End

pass

# 本地配置的参数

class SvnParam:

def \_\_init\_\_(self):

# M1Client svn

self.\_client\_svn\_url = Content\_Svn\_Client\_Url

# M1Res4Build svn

self.art\_svn\_url = Content\_Svn\_Art\_Url

# svn user & pwd

self.\_username = Content\_Svn\_Username

self.\_password = Content\_Svn\_Password

pass

@property

def client\_svn\_path(self):

return self.\_client\_svn\_url

@property

def art\_svn\_path(self):

return self.art\_svn\_url

pass

@property

def username(self):

return self.\_username

@property

def password(self):

return self.\_password

#Class SvnParam End

pass

# 本地Mac配置的参数

class MacOSParam:

def \_\_init\_\_(self):

# MacOS terminal info pandas

self.\_pd\_macos = pd.DataFrame(data=list(Content\_MacOS\_Data.values()),

index=list(Content\_MacOS\_Data.keys()),

columns=['username','password','port','workspace','library\_path','unity\_path'])

# svn user & pwd & ip & # port

self.\_username = None

self.\_password = None

self.\_ip = None

self.\_port = None

self.\_workspace = None

self.\_library\_path = None

self.\_unity\_path = None

pass

def select\_macos\_terminal\_by\_ip(self, ip):

b\_ret = True

try:

self.\_username = self.\_pd\_macos['username'][ip]

self.\_password = self.\_pd\_macos['password'][ip]

self.\_port = int(self.\_pd\_macos['port'][ip])

self.\_workspace = self.\_pd\_macos['workspace'][ip]

self.\_library\_path = self.\_pd\_macos['library\_path'][ip]

self.\_unity\_path = self.\_pd\_macos['unity\_path'][ip]

self.\_ip = ip

except:

b\_ret = False

finally:

return b\_ret

pass

@property

def ip(self):

return self.\_ip

@property

def port(self):

return self.\_port

@property

def username(self):

return self.\_username

@property

def password(self):

return self.\_password

@property

def workspace(self):

return self.\_workspace

@property

def library\_path(self):

return self.\_library\_path

@property

def unity\_path(self):

return self.\_unity\_path

#Class MacOSParam End

pass

# ftp assetbundle

class FfpBackupAssetbundleParam:

def \_\_init\_\_(self):

self.\_username = Content\_Ftp\_Backup\_Assetbundle\_Username

self.\_password = Content\_Ftp\_Backup\_Assetbundle\_Password

self.\_host\_url = Content\_Ftp\_Backup\_Assetbundle\_Host\_Url

self.\_remote\_path = Content\_Ftp\_Backup\_Assetbundle\_Remote\_Path

pass

@property

def username(self):

return self.\_username

@property

def password(self):

return self.\_password

@property

def host\_url(self):

return self.\_host\_url

@property

def remote\_path(self):

return self.\_remote\_path

# Class FfpBackupAssetbundleParam End

pass

# ftp package

class FfpBackupPackageParam:

def \_\_init\_\_(self):

self.\_username = Content\_Ftp\_Backup\_Package\_Username

self.\_password = Content\_Ftp\_Backup\_Package\_Password

self.\_host\_url = Content\_Ftp\_Backup\_Package\_Host\_Url

self.\_remote\_path = Content\_Ftp\_Backup\_Package\_Remote\_Path

pass

@property

def username(self):

return self.\_username

@property

def password(self):

return self.\_password

@property

def host\_url(self):

return self.\_host\_url

@property

def remote\_path(self):

return self.\_remote\_path

@property

def fullpath(self):

return '%s/%s' % (self.\_host\_url, self.\_remote\_path)

@property

def dsym\_fullpath(self):

return '%s/%s/%s' % (self.\_host\_url, self.\_remote\_path, Content\_Ftp\_Backup\_Dsyms\_Remote\_Path)

# Class FfpBackupPackageParam End

pass

# ftp patch

class FfpBackupPatchParam:

def \_\_init\_\_(self):

self.\_username = Content\_Ftp\_Backup\_Patch\_Username

self.\_password = Content\_Ftp\_Backup\_Patch\_Password

self.\_host\_url = Content\_Ftp\_Backup\_Patch\_Host\_Url

self.\_remote\_path = Content\_Ftp\_Backup\_Patch\_Remote\_Path

pass

@property

def username(self):

return self.\_username

@property

def password(self):

return self.\_password

@property

def host\_url(self):

return self.\_host\_url

@property

def remote\_path(self):

return self.\_remote\_path

# Class FfpBackupPackageParam End

# Autobuild \_G.param 全局参数

class TeraParam:

def \_\_init\_\_(self):

# log handler

self.\_file\_log\_handler = None

# 项目最后输出的名称

self.\_project\_all\_name = ''

# 输入的参数

self.\_input\_param = InputParam()

# Unity参数

self.\_unity\_param = UnityParam()

# svn参数

self.\_svn\_param = SvnParam()

# patch参数

self.\_patch\_param = PatchParam()

# assetbundle backup参数

self.\_ftp\_backup\_ab\_param = FfpBackupAssetbundleParam()

# package backup参数

self.\_ftp\_backup\_package\_param = FfpBackupPackageParam()

# patch backup参数

self.\_ftp\_backup\_patch\_param = FfpBackupPatchParam()

# macos ssh终端信息

self.\_macos\_param = MacOSParam()

# python workspace path

self.\_python\_workspace\_path = Content\_AutoBuild\_Python\_Workspace\_Path

# Tera workspace path

self.\_tera\_workspace\_path = Content\_AutoBuild\_Tera\_Workspace\_Path

# m1client\_path

self.\_m1client\_path = 'M1Client'

# m1res4build\_path

self.\_m1res4build\_path = 'M1Res4Build'

# php完成信息 Post Url

self.\_post\_finish\_url = Content\_Post\_Finish\_Url

pass

pass

@property

def input\_param(self):

return self.\_input\_param

@property

def unity\_param(self):

return self.\_unity\_param

@property

def svn\_param(self):

return self.\_svn\_param

@property

def patch\_param(self):

return self.\_patch\_param

@property

def ftp\_backup\_ab\_param(self):

return self.\_ftp\_backup\_ab\_param

@property

def ftp\_backup\_package\_param(self):

return self.\_ftp\_backup\_package\_param

@property

def ftp\_backup\_patch\_param(self):

return self.\_ftp\_backup\_patch\_param

@property

def macos\_param(self):

return self.\_macos\_param

@property

def python\_workspace\_path(self):

return self.\_python\_workspace\_path

@property

def tera\_workspace\_path(self):

return self.\_tera\_workspace\_path

def get\_project\_all\_name(self):

return self.\_project\_all\_name

def set\_project\_all\_name(self, project\_all\_name):

assert (isinstance(project\_all\_name, str) and len(project\_all\_name) > 0)

self.\_project\_all\_name = project\_all\_name

def get\_file\_log\_handler(self):

return self.\_file\_log\_handler

def set\_file\_log\_handler(self, file\_handler):

self.\_file\_log\_handler = file\_handler

project\_all\_name = property(get\_project\_all\_name, set\_project\_all\_name)

file\_log\_handler = property(get\_file\_log\_handler,set\_file\_log\_handler)

@property

def m1client\_path(self):

return self.\_m1client\_path

@property

def m1res4build\_path(self):

return self.\_m1res4build\_path

@property

def post\_finish\_url(self):

return self.\_post\_finish\_url

# 修复平台 M1Client & M1Res4Build 路径

def fix\_working\_path(self):

platfrom\_type = self.\_input\_param.platform\_type

self.\_unity\_param.fix\_unity\_path\_by\_platform(platfrom\_type)

self.\_m1client\_path = '%s\\%s\_%s' % (self.\_tera\_workspace\_path,

self.\_m1client\_path,

platfrom\_type)

self.\_m1res4build\_path = '%s\\%s\_%s' % (self.\_tera\_workspace\_path,

self.\_m1res4build\_path,

platfrom\_type)

pass

# 修复平台 最后生成包文件 添加对应后缀

def fix\_project\_all\_name(self):

platfrom\_type = self.\_input\_param.platform\_type

if "Windows" == platfrom\_type:

self.project\_all\_name = '%s.zip' % self.project\_all\_name

elif "iOS" == platfrom\_type:

self.project\_all\_name = '%s.ipa' % self.project\_all\_name

elif "Android" == platfrom\_type:

self.project\_all\_name = '%s.apk' % self.project\_all\_name

pass

pass

# 查找对应IP的 MacOS SSH配置信息

def fix\_macos\_info(self):

b\_ret = True

if self.input\_param.platform\_type == 'iOS':

b\_ret = self.\_macos\_param.select\_macos\_terminal\_by\_ip(self.input\_param.macos\_ip)

pass

return b\_ret

# 获取 当前版本 art svn用的结构

def get\_cur\_art\_svn\_param(self):

svn\_path = '%s/%s' % (self.svn\_param.art\_svn\_path, self.input\_param.branch\_path)

local\_dest\_path = '%s\\%s' % (self.\_m1res4build\_path, self.input\_param.branch\_path)

cur\_revision = self.input\_param.version\_param.art\_revision

usr = self.svn\_param.username

pwd = self.svn\_param.password

art\_svn\_param = {'username': usr, 'password': pwd, 'svnpath': svn\_path, 'parentpath': local\_dest\_path}

return art\_svn\_param, cur\_revision

# 获取 当前版本 client svn用的结构

def get\_cur\_client\_svn\_param(self):

svn\_path = '%s/%s' % (self.svn\_param.client\_svn\_path, self.input\_param.branch\_path)

local\_dest\_path = '%s\\%s' % (self.\_m1client\_path, self.input\_param.branch\_path)

cur\_revision = self.input\_param.version\_param.client\_revision

usr = self.svn\_param.username

pwd = self.svn\_param.password

art\_svn\_param = {'username': usr, 'password': pwd, 'svnpath': svn\_path, 'parentpath': local\_dest\_path}

return art\_svn\_param, cur\_revision

# 获取 Assetbundle Ftp URL 备份路径

def get\_ab\_download\_path(self):

ab\_down\_path = '%s/%s/%s/%s' % (self.ftp\_backup\_ab\_param.remote\_path,

self.input\_param.branch\_path,

self.input\_param.platform\_type,

self.input\_param.version\_param.current\_version)

return ab\_down\_path

# # 获取 Assetbundle Ftp URL 备份路径

# def get\_build\_ios\_ssh\_cmd\_str(self):

# cmd\_str = '%s/tera\_autobuild.sh %s %s %s %s %s %s %s %s %s' % (self.macos\_param.workspace,

# self.input\_param.project\_name,

# self.input\_param.branch\_path,

# self.input\_param.version\_param.base\_version,

# self.input\_param.version\_param.current\_version,

# self.get\_ab\_download\_path(),

# self.input\_param.version\_param.client\_revision,

# self.project\_all\_name,

# self.input\_param.version\_param.complatform,

# self.input\_param.version\_param.is\_smallpack)

#

# return cmd\_str

# 获取 Assetbundle Ftp URL 备份路径

def get\_build\_ios\_ssh\_cmd\_str(self):

cmd\_str = '%s/tera\_autobuild.sh %s %s %s %s %s %s %s %s %s %s %s' % (self.macos\_param.workspace,

self.input\_param.project\_name,

self.input\_param.branch\_path,

self.input\_param.version\_param.base\_version,

self.input\_param.version\_param.current\_version,

self.get\_ab\_download\_path(),

self.input\_param.version\_param.client\_revision,

self.project\_all\_name,

self.input\_param.version\_param.complatform,

self.input\_param.version\_param.is\_smallpack,

self.macos\_param.workspace,

self.macos\_param.unity\_path)

return cmd\_str

# 获取平台宏定义

def get\_macro\_definition(self):

macro\_definition = 'IN\_GAME'

definition\_list=[]

definition\_list.append(macro\_definition)

if self.input\_param.version\_param.complatform.find('kakao-deploy') != -1:

definition\_list.append('USING\_FABRIC')

definition\_list.append('PLATFORM\_KAKAO')

elif self.input\_param.version\_param.complatform.find("kakao-inhouse-deploy") != -1:

definition\_list.append('USING\_FABRIC')

definition\_list.append('PLATFORM\_KAKAO')

else:

pass

str = ';'

if len(definition\_list) > 1:

macro\_definition = str.join(definition\_list)

else:

macro\_definition = macro\_definition + str;

pass

return macro\_definition

def \_\_del\_\_(self):

del self.\_input\_param

del self.\_unity\_param

del self.\_svn\_param

del self.\_patch\_param

del self.\_ftp\_backup\_ab\_param

del self.\_ftp\_backup\_package\_param

self.file\_log\_handler.close()

pass

# Class TeraParam End

Pass

### tera\_post\_log.py

import requests,os,sys

def post\_log\_file(url, log\_file):

ret = requests.post(url, data=None, files=log\_file)

print('post ret = %s ' % ret)

pass

def get\_log\_path():

return '%s/out.log' % get\_root\_path()

def get\_root\_path():

return os.path.dirname(os.path.abspath(\_\_file\_\_))

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) == 2:

url = sys.argv[1]

file = open('%s/out.log' % get\_root\_path(), 'rb')

post\_log\_file(url, {'log\_file': file})

file.close()

pass

pass

### tera\_utility.py

import os,sys

import subprocess

import requests

import tera\_globals as \_G

import datetime

from ftphelper import FtpHelper

import socket

import ctypes

import os

import platform

import sys

class TeraUtility:

\_file\_log = Noneb

def \_\_int\_\_(self):

pass

@staticmethod

def get\_log\_path():

return '%s/out.log' % TeraUtility.get\_root\_path()

@staticmethod

def get\_start\_log\_path():

return '%s/Autobuild.log' % TeraUtility.get\_root\_path()

@staticmethod

def get\_root\_path():

return os.path.dirname(os.path.abspath(\_\_file\_\_))

@staticmethod

def get\_lock\_path():

return '%s/%s' % (TeraUtility.get\_root\_path(), 'tera\_autobuilding.lock')

@staticmethod

def is\_running():

return os.path.exists(TeraUtility.get\_lock\_path())

@staticmethod

def try\_start\_build():

if TeraUtility.is\_running():

\_G.log('Is Running Now!')

TeraUtility.failed\_exit(True)

else:

file = open(TeraUtility.get\_lock\_path(), 'w')

file.flush()

file.close()

pass

pass

@staticmethod

def padding(list):

result = ''

if list is None or len(list) == 0:

pass

else:

index = 0

for name in list:

if index == 0:

result = name

else:

result = str('%s\_%s' % (result, name))

pass

index += 1

pass

pass

return result

@staticmethod

def execute(cmd, desc=None):

if desc is None:

print('execute: [{}]'.format(cmd))

else:

print('execute: [{}]'.format(desc))

pass

\_G.param.file\_log\_handler.flush()

mytask = subprocess.Popen(cmd,

shell=True,

stdin=subprocess.PIPE,

stdout=\_G.param.file\_log\_handler.fileno(),

stderr=\_G.param.file\_log\_handler.fileno())

while mytask.poll() is None:

pass

print('Execute::[%s]:RetCode = [%s]' % (cmd, mytask.returncode))

if mytask.returncode == 0:

print('Command success')

\_G.param.file\_log\_handler.flush()

else:

print('Command failed!!!: [{}]'.format(cmd))

TeraUtility.failed\_exit()

pass

pass

@staticmethod

def ssh\_execute(ssh\_client, cmd):

print('ssh\_execute: [{}]'.format(cmd))

stdin, stdout, stderr = ssh\_client.exec\_command(cmd)

print(stdout.read())

pass

@staticmethod

def zip\_file(file\_name, src\_path, dest\_path):

from filehelper import FileHelper

import zipfile

os.chdir(src\_path)

if os.path.exists(dest\_path):

file\_helper = FileHelper()

file\_helper.delete\_folder(dest\_path)

del file\_helper

pass

if not os.path.exists(dest\_path):

os.makedirs(dest\_path)

pass

dest\_name = '%s\\%s' % (dest\_path, file\_name)

zip = zipfile.ZipFile(dest\_name, 'w', zipfile.ZIP\_DEFLATED, allowZip64=True)

for dirpath, dirnames, filenames in os.walk('.\\'):

for filename in filenames:

full\_path = os.path.join(dirpath, filename)

print('Zip:[ %s ]' % full\_path)

zip.write(full\_path)

pass

pass

zip.close()

pass

@staticmethod

def ftp\_upload(upload\_path, param=None, ingore=None):

if param is None:

print('ftp upload param is None')

return

ftp = FtpHelper(param)

ftp.login()

print('upload\_path = %s' % upload\_path)

print('remotepath = %s' % param['remotepath'])

ftp.create\_dir(param['remotepath'])

ftp.upload\_files(upload\_path, param['remotepath'], ingore)

ftp.quit()

pass

@staticmethod

def ftp\_download(local\_dest\_path, param=None):

if param is None:

print('ftp upload param is None')

return

ftp = FtpHelper(param)

ftp.login()

print('remotepath = %s' % param['remotepath'])

real\_remote\_path = '/%s' % param['remotepath']

ftp.download\_files(local\_dest\_path, real\_remote\_path)

ftp.quit()

pass

@staticmethod

def ftp\_file\_exist(param=None):

if param is None:

print('ftp upload param is None')

return False

ftp = FtpHelper(param)

ftp.login()

print('remotepath = %s' % param['remotepath'])

real\_remote\_path = '/%s' % param['remotepath']

bRet = ftp.file\_exist(real\_remote\_path)

ftp.quit()

return bRet

@staticmethod

def success\_exit():

TeraUtility.post\_succeed(True)

TeraUtility.\_\_stop\_build\_release\_lock(True)

pass

@staticmethod

def failed\_exit(is\_running = False):

TeraUtility.post\_succeed(False)

if not is\_running:

TeraUtility.\_\_stop\_build\_release\_lock(False)

pass

pass

@staticmethod

def post\_succeed(succeed=False):

file = open('%s/out.log' % TeraUtility.get\_root\_path(), 'rb')

build\_id = \_G.param.input\_param.build\_Id

if succeed:

ftp\_package\_url = \_G.param.ftp\_backup\_package\_param.fullpath

app\_http\_path = 'ftp://%s/%s/%s' % (ftp\_package\_url,

\_G.param.input\_param.platform\_type,

\_G.param.project\_all\_name)

dsym\_http\_path = "#"

if \_G.param.input\_param.platform\_type == 'iOS':

ftp\_dsym\_url = \_G.param.ftp\_backup\_package\_param.dsym\_fullpath

dsym\_http\_path = 'ftp://%s/%s.tar' % (ftp\_dsym\_url, \_G.param.project\_all\_name)

pass

TeraUtility.post\_http(build\_id, {'log\_file': file}, "1", app\_http\_path, dsym\_http\_path)

else:

TeraUtility.post\_http(build\_id, {'log\_file': file}, "0")

pass

@staticmethod

def post\_http(build\_id, log\_file, succeed, app\_http\_path="#", dsym\_path="#"):

postdata = dict(build\_id=build\_id,

app\_http\_path=app\_http\_path,

dsym\_path=dsym\_path,

succeed=succeed)

\_G.log('post\_http app\_http\_path = %s' % app\_http\_path)

ret = requests.post(\_G.param.post\_finish\_url, data=postdata, files=log\_file)

\_G.log('post ret = %s ' % ret)

pass

@staticmethod

def \_\_stop\_build\_release\_lock(succeed=False):

if TeraUtility.is\_running():

print('\_\_stop\_build\_release\_lock')

os.remove(TeraUtility.get\_lock\_path())

pass

if succeed:

print('All Job Success')

else:

print('Error Stop')

exit(-1)

pass

pass

@staticmethod

def get\_now():

return (datetime.datetime.now()).strftime('%Y-%m-%d %H:%M:%S')

@staticmethod

def get\_timestamp():

return (datetime.datetime.now()).strftime('%y-%m-%d-%I-%M-%S-%p')

@staticmethod

def get\_build\_target():

cur\_platform\_type = \_G.param.input\_param.platform\_type

if cur\_platform\_type == 'iOS':

return 'iOS'

elif cur\_platform\_type == 'Android':

return 'Android'

else:

return 'Win64'

pass

@staticmethod

def modify\_patch\_permission\_ssh(param):

import paramiko

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect(param['hostpath'], 22, param['username'], param['password'])

transport = ssh.get\_transport()

transport.set\_keepalive(999999)

cmd\_str = 'chmod -R 755 /home/lantu/www/meteorite%s' % param['remotepath']

TeraUtility.ssh\_execute(ssh, cmd\_str)

ssh.close()

pass

@staticmethod

def get\_ip():

# 获取本机计算机名称

hostname = socket.gethostname()

# 获取本机ip

ip = socket.gethostbyname(hostname)

return ip

@staticmethod

def get\_free\_space(folder):

result = 0.0

if platform.system() == 'Windows':

free\_bytes = ctypes.c\_ulonglong(0)

ctypes.windll.kernel32.GetDiskFreeSpaceExW(ctypes.c\_wchar\_p(folder), None, None, ctypes.pointer(free\_bytes))

result = free\_bytes.value / 1024 / 1024 / 1024

else:

st = os.statvfs(folder)

result = st.f\_bavail \* st.f\_frsize / 1024 / 1024

return result

# Class TeraUtility End

Pass

### tera\_autobuild\_art.py

import os

from svnhelper import SvnHelper

import tera\_globals as \_G

from tera\_utility import TeraUtility

from filehelper import FileHelper

###########################################################

# 美术资源打包模块

###########################################################

class AutobuildArt:

def \_\_init\_\_(self):

self.\_export\_path = '%s\\%s\\TERAMobile\\Export\\AssetBundles\\%s' % (\_G.param.m1res4build\_path,

\_G.param.input\_param.branch\_path,

\_G.param.input\_param.platform\_type)

pass

@property

def export\_path(self):

return self.\_export\_path

@staticmethod

def start():

\_G.log('Autobuild\_M1Res4Build::start')

AutobuildArt.checkout\_m1res4build()

AutobuildArt.build\_m1res4build()

pass

# 更新 M1Res4Build资源 工程

@staticmethod

def checkout\_m1res4build():

local\_dest\_path = '%s\\%s' % (\_G.param.m1res4build\_path, \_G.param.input\_param.branch\_path)

cur\_art\_param, cur\_revision = \_G.param.get\_cur\_art\_svn\_param()

svn = SvnHelper(cur\_art\_param)

if os.path.exists(local\_dest\_path):

svn.cleanup()

svn.revert()

pass

\_G.log('cur\_revision = %s' % cur\_revision)

svn.checkout(None if cur\_revision == '0' else cur\_revision)

svn.revert()

del svn

pass

@staticmethod

def build\_m1res4build():

platform\_type = \_G.param.input\_param.platform\_type

base\_version = \_G.param.input\_param.version\_param.base\_version

current\_version = \_G.param.input\_param.version\_param.current\_version

unity\_path = \_G.param.unity\_param.real\_path

m1res4build\_path = \_G.param.m1res4build\_path

branch\_path = \_G.param.input\_param.branch\_path

build\_target = TeraUtility.get\_build\_target()

check\_autobuild\_timestamp = '%s/\_\_\_\_timestamp\_%s' % (\_G.param.tera\_workspace\_path, TeraUtility.get\_timestamp())

# \_G.log('check\_autobuild\_timestamp = %s' % check\_autobuild\_timestamp)

untiy\_run = '%s -quit -nographics -projectPath %s\\%s\\TERAMobile' % (unity\_path,

m1res4build\_path,

branch\_path)

# 基础 / 增量

if base\_version == current\_version:

method\_base = '%s -executeMethod BuildScript.AutoBuildBasicAssetBundle4%s' % (untiy\_run, platform\_type)

build\_ab\_cmd\_str = '%s -buildTarget %s -batchmode %s' % (method\_base,

build\_target,

check\_autobuild\_timestamp)

else:

export\_path = '%s\\%s\\TERAMobile\\Export\\AssetBundles\\%s' % (\_G.param.m1res4build\_path,

\_G.param.input\_param.branch\_path,

\_G.param.input\_param.platform\_type)

filehelper = FileHelper()

if os.path.exists(export\_path):

filehelper.delete\_folder(export\_path)

pass

del filehelper

os.makedirs(export\_path)

platform\_type = \_G.param.input\_param.platform\_type

branch\_path = \_G.param.input\_param.branch\_path

base\_version = \_G.param.input\_param.version\_param.base\_version

ftp\_ab\_usr = \_G.param.ftp\_backup\_ab\_param.username

ftp\_ab\_pwd = \_G.param.ftp\_backup\_ab\_param.password

ftp\_ab\_host\_url = \_G.param.ftp\_backup\_ab\_param.host\_url

ftp\_ab\_remote\_path = \_G.param.ftp\_backup\_ab\_param.remote\_path

ftp\_ab\_full\_path = '%s/%s/%s/%s/Base' % (ftp\_ab\_remote\_path, branch\_path, platform\_type, base\_version)

ab\_ftp\_download\_param = {'username': ftp\_ab\_usr,

'password': ftp\_ab\_pwd,

'hostpath': ftp\_ab\_host\_url,

'remotepath': ftp\_ab\_full\_path}

TeraUtility.ftp\_download(export\_path, ab\_ftp\_download\_param)

method\_update = '%s -executeMethod BuildScript.AutoBuildUpdateBasicAssetBundle4%s' % (

untiy\_run, platform\_type)

build\_ab\_cmd\_str = '%s -buildTarget %s -batchmode %s' % (method\_update,

build\_target,

check\_autobuild\_timestamp)

pass

TeraUtility.execute(build\_ab\_cmd\_str)

if os.path.exists(check\_autobuild\_timestamp):

\_G.log('Build AssetBundles %s success' % platform\_type)

os.remove(check\_autobuild\_timestamp)

else:

\_G.log('Build AssetBundles, Unity may Crash!!!')

TeraUtility.failed\_exit()

pass

pass

### tera\_autobuild\_client.py

import os

from svnhelper import SvnHelper

from filehelper import FileHelper

import tera\_globals as \_G

from tera\_utility import TeraUtility

import pandas as pd

from tera\_config import \*

###########################################################

# 客户端打包模块

###########################################################

class AutobuildClient:

def \_\_init\_\_(self):

pass

@staticmethod

def start():

\_G.log('Autobuild\_M1Client::Start At %s' % TeraUtility.get\_now())

platform\_type = \_G.param.input\_param.platform\_type

if 'iOS' == platform\_type:

# iOS Call ssh MacOS builder

AutobuildClient.build\_m1client\_ios()

pass

else:

# Windows & Android

AutobuildClient.checkout\_m1client()

# Set Base Version

AutobuildClient.set\_client\_base\_version()

# Set Complatform

AutobuildClient.set\_complatform()

# Set Macro Definition

AutobuildClient.set\_macro\_definition()

# Prebuild

AutobuildClient.prebuild()

# Build Client

if platform\_type == 'Android':

AutobuildClient.select\_android\_api()

AutobuildClient.build\_m1client\_android()

pass

else:

AutobuildClient.build\_m1client\_windows()

pass

pass

pass

# 更新 M1Client客户端 工程

@staticmethod

def checkout\_m1client():

local\_dest\_path = '%s\\%s' % (\_G.param.m1client\_path, \_G.param.input\_param.branch\_path)

cur\_client\_param, cur\_revision = \_G.param.get\_cur\_client\_svn\_param()

svn = SvnHelper(cur\_client\_param)

\_G.log('local path = %s' % local\_dest\_path)

if os.path.exists(local\_dest\_path):

svn.cleanup()

svn.revert()

pass

else:

os.makedirs(local\_dest\_path)

pass

svn.checkout(None if cur\_revision == '0' else cur\_revision)

svn.revert()

del svn

pass

# 设置基础版本号 更新基础版办号

@staticmethod

def set\_client\_base\_version():

\_G.log('Set Client Base Version')

base\_version = \_G.param.input\_param.version\_param.base\_version

unity\_path = \_G.param.unity\_param.real\_path

m1client\_path = \_G.param.m1client\_path

branch\_path = \_G.param.input\_param.branch\_path

build\_target = TeraUtility.get\_build\_target()

untiy\_run = '%s -quit -nographics -projectPath %s\\%s\\UnityProject -project-Tera' % (unity\_path,

m1client\_path,

branch\_path)

build\_target\_cmd\_str = '%s -buildTarget %s -batchmode' % (untiy\_run, build\_target)

set\_base\_ver\_cmd\_str = '%s -executeMethod BuildTools.SetClientBaseVersion %s' % (build\_target\_cmd\_str, base\_version)

TeraUtility.execute(set\_base\_ver\_cmd\_str)

\_G.log('Set Client Base Version success')

pass

@staticmethod

def set\_complatform():

\_G.log('Set Complatform type')

complatform = \_G.param.input\_param.version\_param.complatform

unity\_path = \_G.param.unity\_param.real\_path

m1client\_path = \_G.param.m1client\_path

branch\_path = \_G.param.input\_param.branch\_path

build\_target = TeraUtility.get\_build\_target()

untiy\_run = '%s -quit -nographics -projectPath %s\\%s\\UnityProject -project-Tera' % (unity\_path,

m1client\_path,

branch\_path)

build\_target\_cmd\_str = '%s -buildTarget %s -batchmode' % (untiy\_run, build\_target)

set\_complatform\_cmd\_str = '%s -executeMethod BuildTools.SetComplatform %s' % (build\_target\_cmd\_str, complatform)

TeraUtility.execute(set\_complatform\_cmd\_str)

\_G.log('Set Complatform success')

pass

@staticmethod

def set\_macro\_definition():

\_G.log('Set Macro Definition')

unity\_path = \_G.param.unity\_param.real\_path

m1client\_path = \_G.param.m1client\_path

branch\_path = \_G.param.input\_param.branch\_path

build\_target = TeraUtility.get\_build\_target()

macro\_definition = \_G.param.get\_macro\_definition()

untiy\_run = '%s -quit -nographics -projectPath %s\\%s\\UnityProject -project-Tera' % (unity\_path,

m1client\_path,

branch\_path)

build\_target\_cmd\_str = '%s -buildTarget %s -batchmode' % (untiy\_run, build\_target)

set\_macro\_definition\_cmd\_str = '%s -executeMethod BuildTools.SetScriptingDefineSymbols %s' % (build\_target\_cmd\_str, macro\_definition)

TeraUtility.execute(set\_macro\_definition\_cmd\_str)

\_G.log('Set Macro Definition success')

pass

@staticmethod

def prebuild():

\_G.log('Prebuild Client')

unity\_path = \_G.param.unity\_param.real\_path

m1client\_path = \_G.param.m1client\_path

branch\_path = \_G.param.input\_param.branch\_path

build\_target = TeraUtility.get\_build\_target()

untiy\_run = '%s -quit -nographics -projectPath %s\\%s\\UnityProject -project-Tera' % (unity\_path,

m1client\_path,

branch\_path)

build\_target\_cmd\_str = '%s -buildTarget %s -batchmode' % (untiy\_run, build\_target)

prebuild\_cmd\_str = '%s -executeMethod BuildTools.PrebuildClient' % build\_target\_cmd\_str

TeraUtility.execute(prebuild\_cmd\_str)

\_G.log('Prebuild Client success')

pass

# @staticmethod

# def prebuild():

# \_G.log('Prebuild Client')

# platform\_type = \_G.param.input\_param.platform\_type

# complatform = \_G.param.input\_param.version\_param.complatform

# if platform\_type == 'Android' and complatform.find('cn') != 1:

# m1client\_path = '%s\\%s' % (\_G.param.m1client\_path, \_G.param.input\_param.branch\_path)

#

# file\_help = FileHelper()

# src\_path = '%s\\SDK\\Fabric\\3rd\\Fabric' % m1client\_path

# dest\_path = '%s\\UnityProject\\Assets\\3rd\\Fabric' % m1client\_path

# file\_help.copy\_tree(src\_path,dest\_path)

#

# src\_path = '%s\\SDK\\Fabric\\Plugins\\Android' % m1client\_path

# dest\_path = '%s\\UnityProject\\Assets\\Plugins\\Android' % m1client\_path

# file\_help.copy\_tree(src\_path, dest\_path)

# del file\_help

# pass

#

# \_G.log('Prebuild Client success')

# pass

###############################################################################################

@staticmethod

def build\_m1client\_windows():

\_G.log('Build Client Windows')

platform\_type = \_G.param.input\_param.platform\_type

unity\_path = \_G.param.unity\_param.real\_path

m1client\_path = \_G.param.m1client\_path

branch\_path = \_G.param.input\_param.branch\_path

build\_target = TeraUtility.get\_build\_target()

untiy\_run = '%s -quit -nographics -projectPath %s\\%s\\UnityProject -project-Tera' % (unity\_path,

m1client\_path,

branch\_path)

method\_base = '%s -executeMethod BuildTools.BuildForWindows -project-Tera' % untiy\_run

build\_client\_cmd\_str = '%s -buildTarget %s -batchmode' % (method\_base, build\_target)

# Clean old Package

copy\_from\_path = '%s\\%s\\GameRes' % (m1client\_path, branch\_path)

package\_path = '%s\\%s\\Package' % (m1client\_path, branch\_path)

file\_help = FileHelper()

file\_help.delete\_folder(package\_path)

build\_client\_cmd\_str = '%s -buildTarget %s -batchmode %s' % (method\_base, build\_target, package\_path)

TeraUtility.execute(build\_client\_cmd\_str)

\_G.log('Build Client Windows success')

\_G.log('Copy Client need')

# Copy Client need

copy\_dest\_path = package\_path + '\\GameRes'

copy\_list = ['\\BehaviacData','\\Configs','\\Data','\\Maps','\\Audio\\GeneratedSoundBanks\\Windows','\\Video']

for name in copy\_list:

file\_help.copy\_tree(copy\_from\_path + name, copy\_dest\_path + name, '.meta')

pass

# lua compile

lua\_compile\_path = '%s\\%s\\Tools\\lua\_compiler' % (m1client\_path, branch\_path)

lua\_compile\_cmd\_str = '%s\\cmd\_compile.bat' % lua\_compile\_path

TeraUtility.execute(lua\_compile\_cmd\_str)

src\_path = '%s\\Output\\lua' % lua\_compile\_path

dest\_path = '%s\\Lua' % copy\_dest\_path

file\_help.copy\_tree(src\_path, dest\_path)

# assetbundles

#src\_path = '%s\\AssetBundles\\%s' % (copy\_from\_path, platform\_type)

dest\_path = '%s\\AssetBundles\\%s' % (copy\_dest\_path, platform\_type)

# Download Assetbundle

ftp\_ab\_usr = \_G.param.ftp\_backup\_ab\_param.username

ftp\_ab\_pwd = \_G.param.ftp\_backup\_ab\_param.password

host\_url = \_G.param.ftp\_backup\_ab\_param.host\_url

remote\_path = \_G.param.ftp\_backup\_ab\_param.remote\_path

base\_version = \_G.param.input\_param.version\_param.base\_version

ab\_ftp\_backup\_full\_path = '%s/%s/%s/%s/Base' % (remote\_path, branch\_path, platform\_type, base\_version)

ab\_ftp\_download\_param = {'username': ftp\_ab\_usr,

'password': ftp\_ab\_pwd,

'hostpath': host\_url,

'remotepath': ab\_ftp\_backup\_full\_path}

TeraUtility.ftp\_download(dest\_path, ab\_ftp\_download\_param)

del file\_help

pass

###############################################################################################

@staticmethod

def build\_m1client\_ios():

\_G.log('build\_m1client\_ios')

import paramiko

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ip = \_G.param.macos\_param.ip

port = \_G.param.macos\_param.port

usr = \_G.param.macos\_param.username

pwd = \_G.param.macos\_param.password

workspace = \_G.param.macos\_param.workspace

library\_path = \_G.param.macos\_param.library\_path

\_G.log('ip = %s' % ip)

\_G.log('port = %s' % port)

\_G.log('usr = %s' % usr)

\_G.log('pwd = %s' % pwd)

\_G.log('workspace = %s' % workspace)

\_G.log('library\_path = %s ' % library\_path)

# Connect

ssh.connect(ip,port,usr,pwd)

transport = ssh.get\_transport()

transport.set\_keepalive(999999)

# kill running sh

TeraUtility.ssh\_execute(ssh, 'killall tera\_autobuild.sh')

TeraUtility.ssh\_execute(ssh, 'killall Unity')

# Chmod Permission

cmd\_str = 'chmod -R 777 %s' % workspace

TeraUtility.ssh\_execute(ssh, cmd\_str)

cmd\_str = 'security unlock -p %s %s/Keychains/login.keychain' % (pwd,library\_path)

TeraUtility.ssh\_execute(ssh, cmd\_str)

# Build iOS

cmd\_str = \_G.param.get\_build\_ios\_ssh\_cmd\_str()

\_G.log('Build iOS cmd\_str = %s' % cmd\_str)

TeraUtility.ssh\_execute(ssh, cmd\_str)

ssh.close()

pass

# Class AutobuildClient End

pass

###############################################################################################

@staticmethod

def select\_android\_api():

\_G.log('Select Android Api')

key = 'dev'

complatform = \_G.param.input\_param.version\_param.complatform

if complatform.find('kakao') != -1:

key = 'kakao'

pass

elif complatform.find('longtu') != -1:

key = 'longtu'

pass

else:

pass

pd\_android\_api = pd.DataFrame(data=list(Content\_Android\_Api.values()),

index=list(Content\_Android\_Api.keys()),

columns=['level'])

lv = pd\_android\_api['level'][key]

\_G.log('Android Api [Lv = %d] for %s' % (lv, complatform))

api\_src = '%s\\android-%d' % (Content\_Android\_All\_Platforms\_Path, lv)

api\_dst = '%s\\android-%d' % (Content\_Android\_Platforms\_Path, lv)

\_G.log("api\_src path = " + api\_src)

\_G.log('api\_dst\_path = ' + api\_dst)

file\_helper = FileHelper()

file\_helper.delete\_folder(Content\_Android\_Platforms\_Path)

if os.path.exists(Content\_Android\_Platforms\_Path) is False:

os.makedirs(Content\_Android\_Platforms\_Path)

pass

file\_helper.copy\_tree(api\_src, api\_dst)

del file\_helper

pass

@staticmethod

def build\_m1client\_android():

\_G.log('Build Client Android')

unity\_path = \_G.param.unity\_param.real\_path

m1client\_path = \_G.param.m1client\_path

branch\_path = \_G.param.input\_param.branch\_path

build\_target = TeraUtility.get\_build\_target()

project\_all\_name = \_G.param.project\_all\_name

package\_path = '%s\\%s\\Package' % (m1client\_path, branch\_path)

file\_help = FileHelper()

file\_help.delete\_folder(package\_path)

os.makedirs(package\_path)

res\_base\_path = '%s\\%s\\UnityProject\\Assets\\StreamingAssets\\res\_base' % (m1client\_path, branch\_path)

file\_help.delete\_folder(res\_base\_path)

os.makedirs(res\_base\_path)

fileListGenerator\_path = "%s\\%s\\FileListGenerator.exe" % (m1client\_path, branch\_path)

generator\_filelist\_cmd\_str = '%s %s' % (fileListGenerator\_path, res\_base\_path)

TeraUtility.execute(generator\_filelist\_cmd\_str)

\_G.log('FileListGenerator success')

del\_root\_path = '%s\\%s\\UnityProject\Assets' % (m1client\_path, branch\_path)

del\_list = ['\\Plugins\\iOS',

'\\Plugins\\x86',

'\\Plugins\\x86\_64',

'\\Wwise\Deployment\\Plugins\\Windows',

'\\Wwise\Deployment\\Plugins\\Mac',

'\\Wwise\Deployment\\Plugins\\iOS',]

for name in del\_list:

file\_help.delete\_folder(del\_root\_path + name)

pass

untiy\_run = '%s -quit -nographics -projectPath %s\\%s\\UnityProject -project-Tera' % (unity\_path,

m1client\_path,

branch\_path)

build\_target\_cmd\_str = '%s -buildTarget %s -batchmode' % (untiy\_run, build\_target)

build\_client\_cmd\_str = '%s -executeMethod BuildTools.BuildForAndroid %s %s' % (build\_target\_cmd\_str, project\_all\_name, package\_path)

#prebuild\_cmd\_str = '%s -executeMethod BuildTools.FixGraphicSetting\_iOS' % build\_target\_cmd\_str

#TeraUtility.execute(prebuild\_cmd\_str)

#\_G.log('Build Client Android success - FixGraphicSetting\_iOS')

TeraUtility.execute(build\_client\_cmd\_str)

\_G.log('Build Client Android success')

del file\_help

pass

### tera\_autobuild\_main.py

from tera\_param import TeraParam, EnumInputParam

import tera\_globals as \_G

from tera\_utility import \*

from filehelper import FileHelper

from tera\_autobuild\_art import AutobuildArt

from tera\_autobuild\_client import AutobuildClient

from tera\_autobuild\_patch import AutobuildPatch

import sys

import datetime

\_G.param = None

###################################################################################

# Tera Auto Build Entrance (M1Res4Build & M1Client & Patch)

#

# 1. Set setting path first please!

# 2. php call

# 3. param:[]

#

#

# Date:2017.11.30

###################################################################################

def initialize():

if len(sys.argv) == (EnumInputParam.\_\_len\_\_() + 1):

\_G.param.input\_param.project\_name = sys.argv[EnumInputParam.EProjet\_Name.value]

\_G.param.input\_param.platform\_type = sys.argv[EnumInputParam.EPlatform\_Type.value]

\_G.param.input\_param.branch\_path = sys.argv[EnumInputParam.EBranch\_Path.value]

\_G.param.input\_param.version\_param.base\_version = sys.argv[EnumInputParam.EBase\_Version.value]

\_G.param.input\_param.version\_param.last\_version = sys.argv[EnumInputParam.ELast\_Version.value]

\_G.param.input\_param.version\_param.current\_version = sys.argv[EnumInputParam.ECurrent\_Version.value]

\_G.param.input\_param.version\_param.client\_revision = sys.argv[EnumInputParam.EClient\_Revision.value]

\_G.param.input\_param.version\_param.art\_revision = sys.argv[EnumInputParam.EArt\_Revision.value]

\_G.param.input\_param.version\_param.client\_last\_revision = sys.argv[EnumInputParam.EClient\_Last\_Revision.value]

# Windows 为整包体

\_G.param.input\_param.version\_param.is\_smallpack = '1' if ((sys.argv[EnumInputParam.EIs\_Smallpack.value]=='1') and

\_G.param.input\_param.platform\_type != 'Windows') else '0'

\_G.param.input\_param.version\_param.is\_build\_art = sys.argv[EnumInputParam.EIs\_Build\_Art.value]

\_G.param.input\_param.version\_param.complatform = sys.argv[EnumInputParam.EComplatform.value]

\_G.param.input\_param.build\_Id = sys.argv[EnumInputParam.EBuild\_Id.value]

\_G.param.input\_param.macos\_ip = sys.argv[EnumInputParam.EMacOS\_Ip.value]

\_G.param.input\_param.art\_backup\_version = sys.argv[EnumInputParam.EArt\_Backup\_Versoin.value]

# 通过平台修复路径

\_G.param.fix\_working\_path()

# 选择MacOS终端信息

if \_G.param.fix\_macos\_info() is False:

TeraUtility.failed\_exit()

pass

else:

print('Error param count.', len(sys.argv))

TeraUtility.failed\_exit()

pass

print('Input Param============================================Begin')

print('Disk Free Space : %0.2f GB' % TeraUtility.get\_free\_space('X:\\'))

print('IP %s' % TeraUtility.get\_ip())

print('project\_name = %s' % \_G.param.input\_param.project\_name)

print('platform\_type = %s' % \_G.param.input\_param.platform\_type)

print('branch\_path = %s' % \_G.param.input\_param.branch\_path)

print('base\_version = %s' % \_G.param.input\_param.version\_param.base\_version)

print('last\_version = %s' % \_G.param.input\_param.version\_param.last\_version)

print('current\_version = %s' % \_G.param.input\_param.version\_param.current\_version)

print('client\_revision = %s' % \_G.param.input\_param.version\_param.client\_revision)

print('art\_revision = %s' % \_G.param.input\_param.version\_param.art\_revision)

print('client\_last\_revision = %s' % \_G.param.input\_param.version\_param.client\_last\_revision)

print('is\_smallpack = %s' % \_G.param.input\_param.version\_param.is\_smallpack)

print('is\_build\_art = %s' % \_G.param.input\_param.version\_param.is\_build\_art)

print('complatform = %s' % \_G.param.input\_param.version\_param.complatform)

print('build\_Id = %s' % \_G.param.input\_param.build\_Id)

print('macos\_ip = %s' % \_G.param.input\_param.macos\_ip)

print('MacOS Info IP = %s' % \_G.param.macos\_param.ip)

#print('MacOS Info usr = %s' % \_G.param.macos\_param.username)

#print('MacOS Info pwd = %s' % \_G.param.macos\_param.password)

print('MacOS Info port = %s' % \_G.param.macos\_param.port)

print('MacOS Info workspace = %s' % \_G.param.macos\_param.workspace)

print('MacOS Info unity\_path = %s' % \_G.param.macos\_param.unity\_path)

print('Art backup version = %s' % \_G.param.input\_param.art\_backup\_version)

print('Input Param============================================End\n')

\_G.param.file\_log\_handler.flush()

print('[Start Time : %s]' % datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

# set current project name

\_G.param.project\_all\_name = TeraUtility.padding([\_G.param.input\_param.project\_name,

\_G.param.input\_param.version\_param.complatform,

datetime.date.today(),

\_G.param.input\_param.platform\_type,

\_G.param.input\_param.version\_param.base\_version])

\_G.param.fix\_project\_all\_name()

print('[Project Name : %s]' % \_G.param.project\_all\_name)

# create workspace first time

tera\_workspace\_path = \_G.param.tera\_workspace\_path

if not os.path.exists(tera\_workspace\_path):

os.makedirs(tera\_workspace\_path)

pass

os.chdir(tera\_workspace\_path)

pass

def start():

# Build Assetbundle & Client Logic

b\_is\_build\_art = (\_G.param.input\_param.version\_param.is\_build\_art == '1')

platform\_type = \_G.param.input\_param.platform\_type

project\_all\_name = \_G.param.project\_all\_name

m1client\_path = \_G.param.m1client\_path

branch\_path = \_G.param.input\_param.branch\_path

base\_version = \_G.param.input\_param.version\_param.base\_version

current\_version = \_G.param.input\_param.version\_param.current\_version

last\_version = \_G.param.input\_param.version\_param.last\_version

complatform = \_G.param.input\_param.version\_param.complatform

ftp\_ab\_usr = \_G.param.ftp\_backup\_ab\_param.username

ftp\_ab\_pwd = \_G.param.ftp\_backup\_ab\_param.password

ftp\_ab\_host\_url = \_G.param.ftp\_backup\_ab\_param.host\_url

ftp\_ab\_remote\_path = \_G.param.ftp\_backup\_ab\_param.remote\_path

art\_backup\_version = \_G.param.input\_param.art\_backup\_version

b\_is\_build\_art = (b\_is\_build\_art and art\_backup\_version == '0')

print('[Need Build Art = < %s >)] ' % b\_is\_build\_art)

if b\_is\_build\_art:

# TODO Build Assetbundle

autobuild\_art = AutobuildArt()

AutobuildArt.start()

# TODO Backup Assetbundle

if base\_version == current\_version:

# Base version

print('Backup Base version')

ftp\_ab\_full\_path = '%s/%s/%s/%s/Base' % (ftp\_ab\_remote\_path, branch\_path, platform\_type, base\_version)

ftp\_ab\_upload\_param = {'username': ftp\_ab\_usr,

'password': ftp\_ab\_pwd,

'hostpath': ftp\_ab\_host\_url,

'remotepath': ftp\_ab\_full\_path}

TeraUtility.ftp\_upload(autobuild\_art.export\_path, ftp\_ab\_upload\_param, '.manifest')

else:

# Update version

ftp\_ab\_full\_path = '%s/%s/%s/%s/Update\_%s' % (ftp\_ab\_remote\_path,

branch\_path,

platform\_type,

base\_version,

current\_version)

ftp\_ab\_upload\_param = {'username': ftp\_ab\_usr,

'password': ftp\_ab\_pwd,

'hostpath': ftp\_ab\_host\_url,

'remotepath': ftp\_ab\_full\_path}

TeraUtility.ftp\_upload('%s/Update' % autobuild\_art.export\_path, ftp\_ab\_upload\_param, '.manifest')

pass

del autobuild\_art

else:

# TODO 1.Download backup Assetbundles & 2.Backup to current Version

# Update version

print('Backup Update version')

ftp\_ab\_full\_path = '%s/%s/%s/%s' % (ftp\_ab\_remote\_path,branch\_path,platform\_type,base\_version)

# 创建上传，下载的工作路径

ab\_upload\_path = '%s/ab\_upload\_path' % \_G.param.tera\_workspace\_path

\_G.log('ab\_upload\_path = %s' % ab\_upload\_path)

if os.path.exists(ab\_upload\_path):

FileHelper.delete\_folder(ab\_upload\_path)

pass

if not os.path.exists(ab\_upload\_path):

os.makedirs(ab\_upload\_path)

pass

ab\_ftp\_upload\_param = None

# 基础包，不需要打资源的情况

if current\_version == base\_version and len(art\_backup\_version) > 1:

ftp\_ab\_full\_path = '%s/%s/%s/%s/Base' % (ftp\_ab\_remote\_path,branch\_path,platform\_type,art\_backup\_version)

ab\_ftp\_download\_param = {'username': ftp\_ab\_usr,

'password': ftp\_ab\_pwd,

'hostpath': ftp\_ab\_host\_url,

'remotepath': ftp\_ab\_full\_path}

# 存在传来的备份版本信息，下载。否则 重新打包

if TeraUtility.ftp\_file\_exist(ab\_ftp\_download\_param) is True:

TeraUtility.ftp\_download(ab\_upload\_path, ab\_ftp\_download\_param)

ftp\_ab\_full\_path = '%s/%s/%s/%s/Base' % (ftp\_ab\_remote\_path, branch\_path, platform\_type, base\_version)

else:

autobuild\_art = AutobuildArt()

AutobuildArt.start()

ftp\_ab\_full\_path = '%s/%s/%s/%s/Base' % (ftp\_ab\_remote\_path, branch\_path, platform\_type, base\_version)

ab\_upload\_path = autobuild\_art.export\_path

del autobuild\_art

pass

ab\_ftp\_upload\_param = {'username': ftp\_ab\_usr,

'password': ftp\_ab\_pwd,

'hostpath': ftp\_ab\_host\_url,

'remotepath': ftp\_ab\_full\_path}

else:

if last\_version == base\_version:

# 上一个版本是基础包，没有Update保存

#os.makedirs('%s/Update' % ab\_upload\_path)

pass

else:

# 上一次有备份 'Update\_%s' % Last\_version

ab\_ftp\_download\_param = {'username': ftp\_ab\_usr,

'password': ftp\_ab\_pwd,

'hostpath': ftp\_ab\_host\_url,

'remotepath': '%s/Update\_%s' % (ftp\_ab\_full\_path, last\_version)}

TeraUtility.ftp\_download(ab\_upload\_path, ab\_ftp\_download\_param)

pass

ab\_ftp\_upload\_param = {'username': ftp\_ab\_usr,

'password': ftp\_ab\_pwd,

'hostpath': ftp\_ab\_host\_url,

'remotepath': '%s/Update\_%s' % (ftp\_ab\_full\_path, current\_version)}

pass

TeraUtility.ftp\_upload(ab\_upload\_path, ab\_ftp\_upload\_param, '.manifest')

pass

# Just export app for baseVersion

if base\_version == current\_version:

# TODO Build Client

AutobuildClient.start()

# Windows & Android Using PC, so except iOS

if 'iOS' != platform\_type:

package\_path = '%s\\%s\\Package' % (m1client\_path, branch\_path)

local\_package\_upload\_path = '%s\\%s\\ftp\_upload' % (m1client\_path, branch\_path)

if 'Android' == platform\_type:

local\_package\_upload\_path = package\_path

else:

# zip app

TeraUtility.zip\_file(project\_all\_name, package\_path, local\_package\_upload\_path)

pass

# upload app

ftp\_pkg\_usr = \_G.param.ftp\_backup\_package\_param.username

ftp\_pkg\_pwd = \_G.param.ftp\_backup\_package\_param.password

ftp\_pkg\_host\_url = \_G.param.ftp\_backup\_package\_param.host\_url

ftp\_pkg\_remote\_path = \_G.param.ftp\_backup\_package\_param.remote\_path

ftp\_pkg\_full\_path = '%s/%s' % (ftp\_pkg\_remote\_path,platform\_type)

pkg\_ftp\_upload\_param = {'username': ftp\_pkg\_usr,

'password': ftp\_pkg\_pwd,

'hostpath': ftp\_pkg\_host\_url,

'remotepath': ftp\_pkg\_full\_path}

TeraUtility.ftp\_upload(local\_package\_upload\_path, pkg\_ftp\_upload\_param, '.manifest')

pass

pass

# TODO Patch

autobuild\_patch = AutobuildPatch()

autobuild\_patch.start()

autobuild\_patch.clean()

del autobuild\_patch

pass

# Main Entrance

if \_\_name\_\_ == '\_\_main\_\_':

file\_handler = open(TeraUtility.get\_start\_log\_path(), 'a+')

start\_time = datetime.datetime.now()

file\_handler.write('===================================\n')

file\_handler.write('%s\n' % start\_time.strftime('%Y-%m-%d %H:%M:%S'))

for argv in sys.argv:

file\_handler.writelines(argv)

file\_handler.writelines('\n')

file\_handler.flush()

pass

file\_handler.close()

del file\_handler

# Init TeraParam

\_G.param = TeraParam()

#check is running, or start

log\_path = TeraUtility.get\_log\_path()

if os.path.exists(log\_path):

os.remove(log\_path)

pass

# stdout 写文件

\_G.param.file\_log\_handler = open(TeraUtility.get\_log\_path(), 'a+')

sys.stdout = \_G.param.file\_log\_handler

# 运行中不可同时操作, 直接跳出

TeraUtility.try\_start\_build()

# 初始化工作区, 项目名称

initialize()

# TODO 开始打包逻辑

start()

end\_time = datetime.datetime.now()

total\_seconds = (end\_time - start\_time).total\_seconds()

\_G.log('total time:< %d seconds > = < %0.3f hours >' % (total\_seconds / 60, total\_seconds / 3600))

# TODO post http succeed

TeraUtility.success\_exit()

del \_G.param

pass

//ElementJupGenerator.h

#pragma once

#include <vector>

#include "VersionMan.h"

#include <set>

#include <string>

#include <map>

#define PROJECT\_NAME "Tera-M1"

struct SJupFileEntry //jup文件

{

ELEMENT\_VER vOld;

ELEMENT\_VER vNew;

bool operator<(const SJupFileEntry& rhs) const{

if (vOld != rhs.vOld)

return vOld < rhs.vOld;

else

return vNew < rhs.vNew;

}

};

struct SUpdateFileEntry //一个jup内的文件

{

std::string strMd5; //compressed

std::string strFileName;

int64\_t nSize; //compressed

int64\_t nOriginSize;

bool operator<(const SUpdateFileEntry& rhs) const

{

if (nSize != rhs.nSize)

return nSize > rhs.nSize;

else if (strFileName != rhs.strFileName)

return strFileName < rhs.strFileName;

else if (nOriginSize != rhs.nOriginSize)

return nOriginSize > rhs.nOriginSize;

else

return strMd5 < rhs.strMd5;

}

};

struct SJupContent //一个jup的更新内容

{

ELEMENT\_VER verOld;

ELEMENT\_VER verNew;

std::vector<SUpdateFileEntry> UpdateList;

std::vector<std::string> IncString;

int64\_t GetTotalOriginSize() const

{

int64\_t total = 0;

for (const auto& entry : UpdateList)

{

total += entry.nOriginSize;

}

return total;

}

int64\_t GetTotalSize() const

{

int64\_t total = 0;

for (const auto& entry : UpdateList)

{

total += entry.nSize;

}

return total;

}

std::string ToJupFileName() const

{

std::string strOld = verOld.ToString();

std::string strNew = verNew.ToString();

return std\_string\_format("%s-%s.jup", strOld.c\_str(), strNew.c\_str());

}

std::string ToPckFileName() const

{

std::string strOld = verOld.ToString();

std::string strNew = verNew.ToString();

return std\_string\_format("%s-%s.pck", strOld.c\_str(), strNew.c\_str());

}

};

/\*

Class

\*/

class CElementJUPGenerator

{

public:

CElementJUPGenerator();

~CElementJUPGenerator(){}

public:

enum EPlatformType

{

Windows = 0,

iOS,

Android,

}m\_PlatformType;

struct SConfig

{

std::string JupGeneratePath;

std::string LastVersionPath;

std::string NextVersionPath;

bool bSmallPack;

} m\_SConfig;

struct SVersion

{

std::string BaseVersion;

std::string LastVersion;

std::string NextVersion;

} m\_SVersion;

std::string m\_strWorkDir;

std::string m\_strCompressDir;

public:

bool Init(const std::string& strLastPath,

const std::string& strNextPath,

const std::string& strJupGeneratePath,

bool bSmallPack);

void SetPlatform(const std::string& strPlatformType);

void SetVersion(const std::string& strBaseVersion,

const std::string& strLastVersion,

const std::string& strNextVersion);

const SVersion& GetSVersion() const { return m\_SVersion; }

bool GenerateUpdateList(const SVersion& sversion, SJupContent& jupContent) const;

void PrintUpdateList(const SJupContent& jupContent) const;

bool GenerateJup(const SJupContent& jupContent, bool bForceMx0);

bool GenerateVersionTxt(const SVersion& sversion, const char\* ext) const;

bool SplitJup(const SJupContent& jupContent, std::vector<SJupContent>& jupContentSplitList, int64\_t nLimitSize) const;

void ProcessUpdateList(const SJupContent& jupContent); //解析更新列表，将其中的guid美术资源文件解析成名字

bool GenerateJupUpdateText(const std::vector<SJupContent>& jupContentList, const char\* ext);

bool GeneratePck(const SJupContent& jupContent);

public:

static bool GenerateBaseVersionTxt(const std::string& strBaseVersion, const std::string& strJupGeneratePath);

static bool GenerateVersionTxt(const std::string& baseVersion, const std::string& nextVersion, const std::string& jupDir, const char\* ext);

static bool FindVersionPair(const std::vector<SJupFileEntry>& pairList, const ELEMENT\_VER& vBase, const ELEMENT\_VER& vLatest, const ELEMENT\_VER& curVer, SJupFileEntry& verPair);

private:

void GenerateIncFileString(const SJupContent& jupContent, std::vector<std::string>& strInc) const;

bool ReadVersionText(const char\* strFileName, std::vector<SUpdateFileEntry>& entries) const;

bool ReGenerateJupContentToDir(const SJupContent& jupContent, const char\* strDir) const;

bool CompareDir(const std::string& leftDir, const std::string& rightDir, const std::set<std::string>& fileList) const;

bool DoGenerateJup(const char\* szJupFile, bool useMx0);

bool GeneratePCKFile(const SJupContent& jupContent, const char\* destDir, const char\* packFileName) const;

private:

std::map<std::string, std::string> m\_assetPathMap;

};

//ElementJupGenerator.cpp

bool CElementJUPGenerator::GeneratePCKFile(const SJupContent& jupContent, const char\* destDir, const char\* packFileName) const

{

FileOperate::MakeDir(packFileName);

AFilePackage pckFile;

if (!pckFile.Open(packFileName, "", AFilePackage::CREATENEW))

{

printf("Create Pck Failed: %s\r\n", packFileName);

g\_pAFramework->Printf("Create Pck Failed: %s\r\n", packFileName);

return false;

}

std::string fullJupDir = destDir;

normalizeDirName(fullJupDir);

std::vector<std::string> fileList;

fileList.push\_back("inc");

for (const auto& entry : jupContent.UpdateList)

{

fileList.push\_back(entry.strFileName.c\_str());

}

for (const auto& shortFileName : fileList)

{

std::string fileName = fullJupDir + shortFileName;

FILE\* file = fopen(fileName.c\_str(), "rb");

if (file == nullptr)

{

printf("Open File Failed: %s\r\n", fileName.c\_str());

g\_pAFramework->Printf("Open File Failed: %s\r\n", fileName.c\_str());

return false;

}

fseek(file, 0, SEEK\_END);

auint32 dwFileSize = ftell(file);

auint32 dwCompressedSize = (auint32)(dwFileSize \* 1.1f) + 12;

unsigned char\* pFileContent = (unsigned char\*)malloc(dwFileSize);

unsigned char\* pFileCompressed = (unsigned char\*)malloc(dwCompressedSize);

fseek(file, 0, SEEK\_SET);

fread(pFileContent, dwFileSize, 1, file);

fclose(file);

int nRet = AFilePackage::Compress(pFileContent, dwFileSize, pFileCompressed, &dwCompressedSize);

if (-2 == nRet)

{

printf("Compress File Failed: %s\r\n", fileName.c\_str());

g\_pAFramework->Printf("Compress File Failed: %s\r\n", fileName.c\_str());

return false;

}

if (0 != nRet)

{

dwCompressedSize = dwFileSize;

}

if (dwCompressedSize < dwFileSize)

{

if (!pckFile.AppendFileCompressed(shortFileName.c\_str(), pFileCompressed, dwFileSize, dwCompressedSize))

{

printf("AppendFileCompressed Failed: %s\r\n", shortFileName.c\_str());

g\_pAFramework->Printf("AppendFileCompressed Failed: %s\r\n", shortFileName.c\_str());

free(pFileCompressed);

free(pFileContent);

return false;

}

}

else

{

if (!pckFile.AppendFileCompressed(shortFileName.c\_str(), pFileContent, dwFileSize, dwFileSize))

{

printf("AppendFileCompressed2 Failed: %s\r\n", shortFileName.c\_str());

g\_pAFramework->Printf("AppendFileCompressed2 Failed: %s\r\n", shortFileName.c\_str());

free(pFileCompressed);

free(pFileContent);

return false;

}

}

free(pFileContent);

free(pFileCompressed);

}

printf("Pck: %s Total %d files, %d bytes\n", packFileName, pckFile.GetFileNumber(), pckFile.GetFileHeader().dwEntryOffset);

g\_pAFramework->Printf("Pck: %s Total %d files, %d bytes\n", packFileName, pckFile.GetFileNumber(), pckFile.GetFileHeader().dwEntryOffset);

pckFile.Flush();

pckFile.Close();

return true;

}

bool CElementJUPGenerator::ReGenerateJupContentToDir(const SJupContent& jupContent, const char\* strDir) const

{

FileOperate::DeleteDir(strDir);

FileOperate::MakeDir(strDir);

//生成更新内容到 compress 目录

//重新生成inc文件

std::string strIncFile = "inc";

{

std::string path = std::string(strDir) + strIncFile;

FILE\* file = fopen(path.c\_str(), "wt");

if (!file)

{

printf("无法创建inc文件!\r\n");

g\_pAFramework->Printf("无法创建inc文件!\r\n");

return false;

}

for (const auto& str : jupContent.IncString)

{

fprintf(file, "%s\n", str.c\_str());

}

fclose(file);

}

//拷贝到本地compress目录

std::string strUpdateBase = m\_SConfig.NextVersionPath;

normalizeDirName(strUpdateBase);

std::string strSrc, strDest;

for (const auto& entry : jupContent.UpdateList)

{

std::string filename = entry.strFileName;

bool bNoCompress = true;

//只对Lua, Configs目录下的文件使用zlib压缩，因为在解压时大文件需要额外的大内存，且assetbundle文件压缩率本就不高

// if (strstr(filename, "Lua/") == (const char\*)filename || strstr(filename, "Configs/") == (const char\*) filename)

// {

// bNoCompress = false;

// }

strSrc = strUpdateBase + filename;

strDest = std::string(strDir) + filename;

FileOperate::MakeDir(strDest.c\_str());

if (!MakeCompressedFile(strSrc.c\_str(), strDest.c\_str(), bNoCompress))

{

printf("制作压缩文件失败! 从%s到%s\r\n", strSrc.c\_str(), strDest.c\_str());

g\_pAFramework->Printf("制作压缩文件失败! 从%s到%s\r\n", strSrc.c\_str(), strDest.c\_str());

return false;

}

}

return true;

}

bool CElementJUPGenerator::CompareDir(const std::string& leftDir, const std::string& rightDir, const std::set<std::string>& fileList) const

{

std::string strLeftDir = leftDir;

normalizeDirName(strLeftDir);

std::string strRightDir = rightDir;

normalizeDirName(strRightDir);

for (const std::string& strFile : fileList)

{

std::string strLeftFile = strLeftDir + strFile;

std::string strRightFile = strRightDir + strFile;

char md5Left[64];

char md5Right[64];

if (!FileOperate::FileExist(strLeftFile.c\_str()) || !FileOperate::FileExist(strRightFile.c\_str())) //文件必须存在

return false;

if (!FileOperate::CalcFileMd5(strLeftFile.c\_str(), md5Left) || !FileOperate::CalcFileMd5(strRightFile.c\_str(), md5Right)) //生成md5

return false;

if (FileOperate::Md5Cmp(md5Left, md5Right) != 0)

return false;

}

return true;

}

bool CElementJUPGenerator::GenerateVersionTxt(const SVersion& sversion, const char\* ext) const

{

return GenerateVersionTxt(sversion.BaseVersion, sversion.NextVersion, m\_SConfig.JupGeneratePath, ext);

}

bool CElementJUPGenerator::GenerateVersionTxt(const std::string& baseVersion, const std::string& nextVersion, const std::string& jupDir, const char\* ext)

{

std::string strJupDir = jupDir;

normalizeDirName(strJupDir);

ELEMENT\_VER vBase;

if (!vBase.Parse(baseVersion))

{

ASSERT(false);

return false;

}

ELEMENT\_VER vNext;

if (!vNext.Parse(nextVersion))

{

ASSERT(false);

return false;

}

printf("收集Jup文件: %s\r\n", strJupDir.c\_str());

g\_pAFramework->Printf("收集Jup文件: %s\r\n", strJupDir.c\_str());

std::set<ELEMENT\_VER> versionSet;

std::vector<SJupFileEntry> updateFileList;

//找所有的jup文件

Q\_iterateFiles(strJupDir.c\_str(),

[&versionSet, &updateFileList, vBase, ext](const char\* filename)

{

if (!hasFileExtensionA(filename, ext))

return;

// if (6 != sscanf(filename, "%d.%d.%d-%d.%d.%d.jup", &verOld[0], &verOld[1], &verOld[2], &verNew[0], &verNew[1], &verNew[2]))

// return;

SJupFileEntry entry;

//解析版本号

{

char shortFileName[QMAX\_PATH];

getFileNameNoExtensionA(filename, shortFileName, QMAX\_PATH);

std::string strFileName = shortFileName;

std::vector<std::string> arr;

std\_string\_split(strFileName, '-', arr);

if (arr.size() != 2 ||

!entry.vOld.Parse(arr[0]) ||

!entry.vNew.Parse(arr[1]))

{

ASSERT(false);

return;

}

}

if (entry.vOld < vBase)

return;

versionSet.insert(entry.vOld);

versionSet.insert(entry.vNew);

updateFileList.push\_back(entry);

},

strJupDir.c\_str());

std::sort(updateFileList.begin(), updateFileList.end());

if (updateFileList.empty() || versionSet.empty())

{

printf("要更新的jup文件数量为0, 生成基础version.txt!\r\n");

g\_pAFramework->Printf("要更新的jup文件数量为0, 生成基础version.txt!\r\n");

GenerateBaseVersionTxt(baseVersion, strJupDir);

return true;

}

for (auto ver : versionSet)

{

std::string str = ver.ToString();

printf("version: %s\r\n", str.c\_str());

g\_pAFramework->Printf("version: %s\r\n", str.c\_str());

}

//检查Version

{

if ((\*versionSet.begin()) != vBase)

{

std::string strBegin = (\*versionSet.begin()).ToString();

std::string strBase = vBase.ToString();

printf("jup不包括BaseVersion! versionSetBegin: %s , vBase: %s\r\n", strBegin.c\_str(), strBase.c\_str());

g\_pAFramework->Printf("jup不包括BaseVersion! versionSetBegin: %s , vBase: %s\r\n", strBegin.c\_str(), strBase.c\_str());

return false;

}

//中间版本必须包含

/\*

for (int i = vBase.iVer2 + 1; i < vNext.iVer2; ++i)

{

ELEMENT\_VER ver(vBase.iVer0, vBase.iVer1, vBase.iVer2, i, 0);

auto itr = std::find(versionSet.begin(), versionSet.end(), ver);

if (itr == versionSet.end())

{

std::string strVer = ver.ToString();

printf("jup不包括中间Version! ver: %s\r\n", strVer.c\_str());

g\_pAFramework->Printf("jup不包括中间Version! ver: %s\r\n", strVer.c\_str());

return false;

}

}

\*/

if ((\*versionSet.rbegin()) != vNext)

{

std::string strNext = vNext.ToString();

printf("jup不包括NextVersion! vNext: %s\r\n", strNext.c\_str());

g\_pAFramework->Printf("jup不包括NextVersion! vNext: %s\r\n", strNext.c\_str());

return false;

}

}

//检查VersionPair的完整性，是否能从base升级到latest

{

for (const SJupFileEntry& entry : updateFileList)

{

ELEMENT\_VER curVer = vBase;

SJupFileEntry pair;

pair.vOld = vBase;

pair.vNew = vBase;

while (pair.vNew < vNext)

{

bool bFound = FindVersionPair(updateFileList, vBase, vNext, curVer, pair);

if (!bFound)

{

std::string strVer = curVer.ToString();

printf("无法找到版本对应的升级jup! curVer: %s\r\n", strVer.c\_str());

g\_pAFramework->Printf("无法找到版本对应的升级jup! curVer: %s\r\n", strVer.c\_str());

return false;

}

curVer = pair.vNew;

}

}

}

//

std::string strTxtFile = strJupDir + "version.txt";

FILE\* file = fopen(strTxtFile.c\_str(), "wt");

if (!file)

{

printf("无法创建version.txt文件!\r\n");

g\_pAFramework->Printf("无法创建version.txt文件!\r\n");

return false;

}

fprintf(file, "Version:\t%s/%s\n", nextVersion.c\_str(), baseVersion.c\_str());

fprintf(file, "Project:\t%s\n", PROJECT\_NAME);

for (const SJupFileEntry& entry : updateFileList)

{

std::string strOld = entry.vOld.ToString();

std::string strNew = entry.vNew.ToString();

std::string strFile = std\_string\_format("%s-%s.%s", strOld.c\_str(), strNew.c\_str(), ext);

std::string strJupFile = strJupDir + strFile;

char md5String[64];

if (!FileOperate::FileExist(strJupFile.c\_str()))

{

printf("jup文件不存在, %s!\r\n", strJupFile.c\_str());

g\_pAFramework->Printf("jup文件不存在, %s!\r\n", strJupFile.c\_str());

fclose(file);

return false;

}

if (!FileOperate::CalcFileMd5(strJupFile.c\_str(), md5String))

{

printf("md5计算错误, %s!\r\n", strJupFile.c\_str());

g\_pAFramework->Printf("md5计算错误, %s!\r\n", strJupFile.c\_str());

fclose(file);

return false;

}

char filename[MAX\_PATH];

getFileNameNoExtensionA(strJupFile.c\_str(), filename, MAX\_PATH);

int nSize = FileOperate::GetFileSize(strJupFile.c\_str());

fprintf(file, "%s\t%s\t%d\n", filename, md5String, nSize);

}

fclose(file);

return true;

}

bool CElementJUPGenerator::SplitJup(const SJupContent& jupContent, std::vector<SJupContent>& jupContentSplitList, int64\_t nLimitSize) const

{

ELEMENT\_VER vOrigOld = jupContent.verOld;

ELEMENT\_VER vOrigNew = jupContent.verNew;

jupContentSplitList.clear();

int64\_t nCurrentSize = 0;

int64\_t nLastOriginSize = INT\_MAX;

int64\_t nOriginSize = 0;

std::vector<SUpdateFileEntry> updateFileEntries;

for (const SUpdateFileEntry& entry : jupContent.UpdateList)

{

if (nCurrentSize + entry.nSize <= nLimitSize)

{

//添加此entry

updateFileEntries.push\_back(entry);

nCurrentSize += entry.nSize;

}

else

{

if (!updateFileEntries.empty()) //已有文件列表，结束本split

{

SJupContent content;

content.UpdateList = updateFileEntries;

jupContentSplitList.emplace\_back(content); //添加到SplitList

}

//添加此entry

{

nCurrentSize = 0;

updateFileEntries.clear();

//添加此entry

updateFileEntries.push\_back(entry);

nCurrentSize += entry.nSize;

}

}

}

//if (!updateFileEntries.empty()) //最后一个

{

SJupContent content;

content.UpdateList = updateFileEntries;

jupContentSplitList.emplace\_back(content); //添加到SplitList

}

//按解压后的size从大到小排序

std::sort(jupContentSplitList.begin(), jupContentSplitList.end(),

[](const SJupContent& v1, const SJupContent& v2)

{

return v1.GetTotalOriginSize() > v2.GetTotalOriginSize();

}

);

//分配verOld, verNew，并重新生成inc文件

for (size\_t i = 0; i < jupContentSplitList.size(); ++i)

{

ELEMENT\_VER vStart;

ELEMENT\_VER vEnd;

if (i == 0)

{

vStart = vOrigOld;

if (i + 1 == jupContentSplitList.size())

vEnd = vOrigNew;

else

vEnd.Set(vStart.iVer0, vStart.iVer1, vStart.iVer2, vStart.iVer3, vStart.iVer4 + 1);

}

else

{

vStart = jupContentSplitList[i - 1].verNew;

if (i + 1 == jupContentSplitList.size())

vEnd = vOrigNew;

else

vEnd.Set(vStart.iVer0, vStart.iVer1, vStart.iVer2, vStart.iVer3, vStart.iVer4 + 1);

}

jupContentSplitList[i].verOld = vStart;

jupContentSplitList[i].verNew = vEnd;

//生成inc文件

GenerateIncFileString(jupContentSplitList[i], jupContentSplitList[i].IncString);

}

return true;

}

void CElementJUPGenerator::ProcessUpdateList(const SJupContent& jupContent)

{

std::string strPlatformAssetBundle = "AssetBundles/";

switch (m\_PlatformType)

{

case CElementJUPGenerator::Windows:

strPlatformAssetBundle += "Windows";

break;

case CElementJUPGenerator::iOS:

strPlatformAssetBundle += "iOS";

break;

case CElementJUPGenerator::Android:

strPlatformAssetBundle += "Android";

break;

default:

break;

}

std::string strPlatformUpdateAssetBundle = strPlatformAssetBundle + "/Update";

std::string strNewDir = this->m\_SConfig.NextVersionPath;

normalizeDirName(strNewDir);

strNewDir += strPlatformUpdateAssetBundle;

normalizeDirName(strNewDir);

std::string strPathIDFile = strNewDir + "PATHIDBACKUP.dat";

m\_assetPathMap.clear();

AFileImage File;

if (!File.Open("", strPathIDFile.c\_str(), AFILE\_OPENEXIST | AFILE\_TEXT))

{

//ASSERT(false);

return;

}

auint32 dwReadLen;

std::vector<std::string> stringList;

char szLine[AFILE\_LINEMAXLEN];

while (File.ReadLine(szLine, AFILE\_LINEMAXLEN, &dwReadLen))

{

std\_string\_split(szLine, ',', stringList);

if (stringList.size() >= 2 && stringList[0].length() == 32)

m\_assetPathMap[stringList[0]] = stringList[1];

}

}

bool CElementJUPGenerator::GenerateJupUpdateText(const std::vector<SJupContent>& jupContentList, const char\* ext)

{

std::string strJupDir = m\_SConfig.JupGeneratePath;

normalizeDirName(strJupDir);

std::set<ELEMENT\_VER> versionSet;

for (const auto& jupContent : jupContentList)

{

versionSet.insert(jupContent.verOld);

versionSet.insert(jupContent.verNew);

}

std::string minVer = versionSet.begin()->ToShortString();

std::string maxVer = versionSet.rbegin()->ToShortString();

ATIME time;

ASys::GetCurLocalTime(time, NULL);

std::string strDate;

strDate = std\_string\_format("%04d-%02d-%02d\_%02d\_%02d\_%02d-[%s-%s]",

time.year + 1900, time.month + 1, time.day, time.hour, time.minute, time.second, minVer.c\_str(), maxVer.c\_str());

std::string strTxtFile = strJupDir + "JupUpdateContent\_" + strDate + ".txt";

FILE\* file = fopen(strTxtFile.c\_str(), "wt");

if (!file)

{

printf("无法创建JupUpdateContent.txt文件!\r\n");

g\_pAFramework->Printf("无法创建JupUpdateContent.txt文件!\r\n");

return false;

}

std::map<std::string, SUpdateFileEntry> updateEntryList;

for (const auto& jupContent : jupContentList)

{

updateEntryList.clear();

std::string verOld = jupContent.verOld.ToString();

std::string verNew = jupContent.verNew.ToString();

std::string strJupFile = std\_string\_format("%s-%s.%s", verOld.c\_str(), verNew.c\_str(), ext);

strJupFile = strJupDir + strJupFile;

auint32 jupSize = FileOperate::GetFileSize(strJupFile.c\_str());

int64\_t totalSize = 1;

for (const auto& entry : jupContent.UpdateList)

{

updateEntryList[entry.strFileName] = entry;

totalSize += entry.nSize;

}

float fRatio = (float)jupSize / (float)totalSize;

fprintf(file, "[%s-%s.%s]\t%u / %u = %0.2f\n", verOld.c\_str(), verNew.c\_str(), ext, (auint32)jupSize, (auint32)totalSize, fRatio);

for (const auto& kv : updateEntryList)

{

const auto& entry = kv.second;

char tmp[QMAX\_PATH];

getFileNameNoExtensionA(entry.strFileName.c\_str(), tmp, QMAX\_PATH);

std::string strResName;

auto itr = m\_assetPathMap.find(tmp);

if (itr != m\_assetPathMap.end())

{

strResName = itr->second;

}

fprintf(file, "%s\t\t%s\t\t%lld\t\t%s\n", entry.strFileName.c\_str(), entry.strMd5.c\_str(), entry.nSize, strResName.c\_str());

}

fprintf(file, "\n");

updateEntryList.clear();

}

fclose(file);

return true;

}

bool CElementJUPGenerator::GeneratePck(const SJupContent& jupContent)

{

std::string strPckFile = m\_SConfig.JupGeneratePath;

normalizeDirName(strPckFile);

std::string strFile = jupContent.ToPckFileName();

strPckFile += strFile;

printf("GeneratePck %s......\r\n", strPckFile.c\_str());

g\_pAFramework->Printf("GeneratePck %s......\r\n", strPckFile.c\_str());

//必须先删掉原来的jup文件

{

FileOperate::UDeleteFile(strPckFile.c\_str());

}

//重新生成更新内容到 compress 目录

if (!ReGenerateJupContentToDir(jupContent, m\_strCompressDir.c\_str()))

{

printf("无法生成更新内容到 %s!\r\n", m\_strCompressDir.c\_str());

g\_pAFramework->Printf("无法生成更新内容到 %s!\r\n", m\_strCompressDir.c\_str());

FileOperate::DeleteDir(m\_strCompressDir.c\_str());

return false;

}

if (!GeneratePCKFile(jupContent, m\_strCompressDir.c\_str(), strPckFile.c\_str()))

{

printf("无法生成PCK文件 %s!\r\n", strPckFile.c\_str());

g\_pAFramework->Printf("无法生成PCK文件 %s!\r\n", strPckFile.c\_str());

FileOperate::DeleteDir(m\_strCompressDir.c\_str());

return false;

}

FileOperate::DeleteDir(m\_strCompressDir.c\_str());

return true;

}

bool CElementJUPGenerator::FindVersionPair(const std::vector<SJupFileEntry>& pairList, const ELEMENT\_VER& vBase, const ELEMENT\_VER& vLatest, const ELEMENT\_VER& curVer, SJupFileEntry& verPair)

{

if (pairList.empty() || curVer == vLatest || curVer > vLatest || curVer < vBase)

return false;

ELEMENT\_VER vOld(-1, 0, 0, 0, 0);

for (const auto& pair : pairList)

{

if (curVer == pair.vOld)

{

vOld = pair.vOld;

break;

}

}

if (vOld.iVer0 < 0)

return false;

//找最高的目标版本

int iVer = -1;

ELEMENT\_VER verNew = vBase;

for (int i = 0; i < (int)pairList.size(); ++i)

{

if (pairList[i].vOld != vOld)

continue;

if (pairList[i].vNew > verNew)

{

iVer = i;

verNew = pairList[i].vNew;

}

}

if (iVer < 0) //没有找到

return false;

verPair = pairList[iVer];

return true;

}

bool CElementJUPGenerator::GenerateBaseVersionTxt(const std::string& strBaseVersion, const std::string& strJupGeneratePath)

{

//

std::string strJupDir = strJupGeneratePath;

normalizeDirName(strJupDir);

std::string strTxtFile = strJupDir + "version.txt";

FILE\* file = fopen(strTxtFile.c\_str(), "wt");

if (!file)

{

printf("无法创建version.txt文件!\r\n");

g\_pAFramework->Printf("无法创建version.txt文件!\r\n");

return false;

}

fprintf(file, "Version:\t%s/%s\n", strBaseVersion.c\_str(), strBaseVersion.c\_str());

fprintf(file, "Project:\t%s\n", PROJECT\_NAME);

fclose(file);

return true;

}

bool CElementJUPGenerator::ReadVersionText(const char\* strFileName, std::vector<SUpdateFileEntry>& entries) const

{

entries.clear();

AFileImage File;

if (!File.Open("", strFileName, AFILE\_OPENEXIST | AFILE\_TEXT))

return false;

auint32 dwReadLen;

char szLine[AFILE\_LINEMAXLEN];

char szMd5[256]; //compressed

char szFileName[256];

int64\_t nSize; //compressed

while (File.ReadLine(szLine, AFILE\_LINEMAXLEN, &dwReadLen))

{

if (3 == sscanf(szLine, "%s\t%s\t%lld", szFileName, szMd5, &nSize))

{

SUpdateFileEntry entry;

entry.strFileName = szFileName;

entry.strMd5 = szMd5;

entry.nSize = nSize;

entries.push\_back(entry);

}

}

return true;

}