#### 目录

[AssetBundle工具文档 2](#_Toc5830)

[前言摘要 2](#_Toc15175)

[目录规范 2](#_Toc15320)

[相关表格 2](#_Toc25692)

[项目中的资源加载策略 2](#_Toc20180)

[打包策略 3](#_Toc31565)

[加载策略 4](#_Toc10867)

[卸载策略 5](#_Toc22211)

[问题及解决方案 6](#_Toc24124)

[编辑器使用说明 7](#_Toc13705)

[参考文献 7](#_Toc28074)

# AssetBundle工具文档

## 前言摘要

本文档主要对项目中使用AssetBundle的流程规范和工具使用进行说明。

以下AssetBundle皆简称为AB。

## 资源相关的目录

* Res目录：项目主要的资源目录，asset表中的所有资源都存放在这个目录，需要打AB的目录。
* Resources目录：项目的固定资源，一定会打进包里的资源，比如资源更新界面的资源。
* StreamingAssets目录：随包发布的资源，一般存放setup.xml，AssetBundles，Data等。

## 相关表格设计

asset表

人工维护的excel表格，记录assetid到资源路径的对应关系，项目中主要通过assetid动态加载资源。

字段：

id：动态加载资源的唯一id，项目中使用此id来配置和加载指定资源。

dir：资源所在的目录，路径从Res目录之下开始，使用“\_”代替“/”分割子目录。

name：资源文件名。

suffix：资源文件名后缀。

### 固定AB配置表

使用AB编辑器人工维护的xml配置表，记录指定的目录下的资源需要打AB，有两个用途，一是实现某些目录支持按路径加载，二是设置同一目录打包到一个AB，人工控制打包粒度。

字段：

folder：目录路径。从Res目录之下开始。

### BundleTable

分析项目资源依赖关系后自动生成的json数据表，记录assetid或者资源路径所在AB的对应关系，运行时需要在这个表格中查询目标资源存放在哪个AB中。

字段：

id：AB的索引，支持通过assetId和资源路径两种方式索引到AB。

bundleName：该资源所在的ABName。

assetPath：该资源的资源路径。用于从AB中加载指定的资源。

## 项目开发中的资源加载方式

* 编辑器下支持通过AssetBundle.Load、Resources.Load、AssetDatabase.Load三种方式加载资源，其中AB模式需要在setup.xml中设置开启，真机强制开启，编辑器默认关闭。
* 按照目录规范，Resources目录只有少量固定资源，大部分资源在Res目录中，只能通过AssetDatabase.Load加载，AssetDatabase.Load没有异步加载接口。
* 真机无法使用AssetDatabase.Load，所以真机要加载Res目录的资源只能通过AssetBundle。打包真机必须包含AB。
* 综上所述，编辑器下根据setup.xml配置决定是否通过AB加载，AB加载失败则通过Resources.Load加载，失败再通过AssetDatabase.Load加载。真机则固定先通过AB加载，AB不存在则通过Resources.Load加载。

## 打包策略

* Unity5.x以后的打包策略是设置了ABName的资源会与该资源引用到的所有没有设置ABName的资源打成一个AB，如果引用的资源设置了ABName，则会生成与该AB的依赖关系，记录在AssetBundleManifest文件中，加载时需要先加载依赖AB再加载目标AB的资源。
* 针对上一点提到的规则，我们的打包策略的核心是以asset表中的资源加上固定包配置的资源作为所有动态加载资源的索引，生成BundleTable记录索引到AB的关系。然后统计索引资源依赖的资源的引用次数，将被重复引用的依赖资源设置ABName独立打包，没有重复引用的依赖资源则被收集到索引资源的AB中。这样项目中的被依赖资源就不会被重复打进不同的AB中。

打包步骤：

1. 读入asset表，读入固定包配置，这两个表的资源就是动态加载资源的直接索引资源。
2. 统计直接索引资源对其它依赖资源的引用记数。
3. 根据固定包配置，用固定包的路径名将每个目录下的所有资源设置成同一个ABName。每个资源在BundleTable中生成路径到AB的映射记录。额外生成一条目录到AB的映射记录，用于实现加载目录下所有资源的LoadAll接口。
4. 遍历asset表的直接索引资源，如果还没有设置过ABName，则使用assetID作为ABName，如果已经设置过ABName，则沿用上一个ABName，然后生成assetID到ABName的BundleTable记录。其中所有shader都设置同一个ABName打包到一起。
5. 遍历所有依赖资源，引用计数大于1的资源需要另外设置ABName独立打包，引用次数等于1的取消ABName以便打入引用它的AB包中。至此，项目中的所有ABName设置完毕。
6. 对比我们自己生成的所有ABName记录，和AssetDatebase.GetAllAssetBundleNames()获取的所有ABName，移除Unity中多余的ABName，这一步可以去除项目中不需要的ABName。
7. 调用Unity接口，根据设置好的ABName输出AB。
8. 使用AssetDatebase.GetAllAssetBundleNames()获取本次打包的所有ABName，删除输出目录中不属于本次打包需要生成的AB。

## 加载策略

AssetBundleManager:

AB管理器。提供通过AssetBundle同步和异步加载的接口。AssetBundle作为资源加载的其中一种方式由ResourceManager调用，游戏代码不直接调用AssetBundleManager。

主要接口：

public T LoadAsset<T>(string assetID) where T : Object：根据资源索引同步加载资源。

public T[] LoadAllAsset<T>(string assetID) where T : Object：根据资源索引同步加载所有资源。

public void LoadAssetAsync(string assetID, bool isSceneBundle, Action<Object> callBack = null)：异步加载资源。

AssetBundleLoader

处理AssetBundle的加载，主要处理异步加载的任务管理。

主要接口：

public AssetBundleInfo LoadBundle(string bundleName,bool isMainBundle = true)：加载AB。

private void LoadDepBundle(string mainBundleName)：根据ABName加载依赖的AB。

public IEnumerator LoadAssetAsync(string bundleName, bool isSceneBundle, string assetName)：异步加载AB和AB中的资源。

AssetBundleInfo

封装AssetBundle对象，添加信息方便管理AssetBundle的加载卸载。

主要接口：

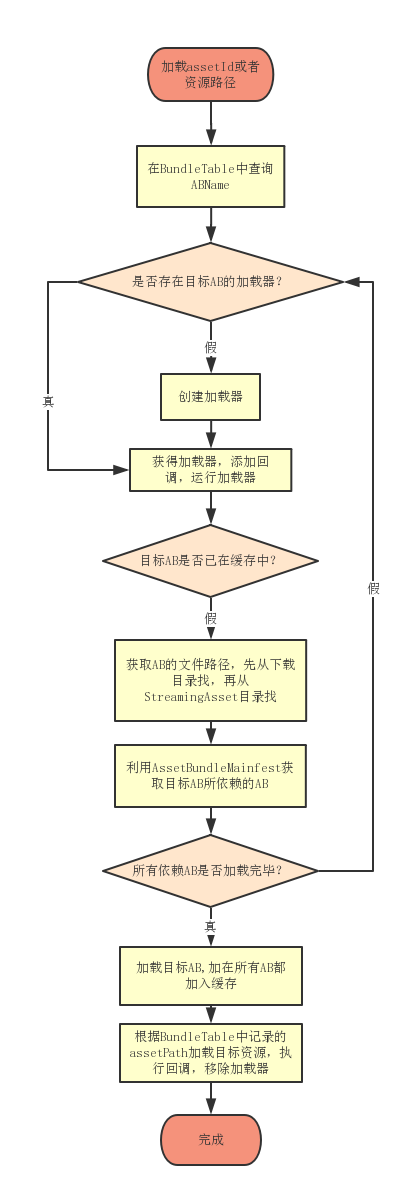
public T LoadAsset<T>(string assetName) where T : Object：同步加载资源。

public T[] LoadAllAsset<T>() where T : Object：同步加载所有资源。

public IEnumerator LoadAssetAsync(string assetName, Action<Object> OnFinish)：异步加载资源。

public bool UnLoadBundle()：卸载AB。

### 加载流程



## 

## 卸载策略

* AssetBundle的卸载有两种方式：UnLoad(true)和UnLoad(false)
* UnLoad(true)操作：会卸载AB对象，无法再从此AB中加载资源，并且会销毁所有从此AB加载出来的资源，若场景上或内存中还有GameObject使用到这些资源就会丢失引用看到各种显示错误，所以调用UnLoad(true)必须保证场景上及内存中都没有任何对象在引用此AB的资源。Unity没有提供这方面的接口，需要自己做引用记数，在实际开发中维护引用记数也很麻烦容易出错。
* UnLoad(false)操作：仅卸载AB对象，无法再从此AB中加载资源，已经加载出来的资源仍在内存中不受影响。需要注意的是，假如AB卸载时，场景上保持着AB中的一些资源的引用，这些资源不会被卸载，接着游戏流程又触发加载此AB，并从中加载曾加载过的资源，此时界面显示正常，但是这些相同的资源会在内存中存在两份。即同一份资源经过卸载过再加载的AB来加载会产生**冗余**。另外，同样的资源如果被重复打包到不同AB中，再通过不同AB加载出来，则此时这些资源在文件上和内存中都是**冗余**的。

综上所述，卸载策略出问题时，UnLoad(true)的后果是显示错误，且可能导致游戏无法继续运行。UnLoad(false)的后果是可能有部分资源存在一份内存冗余，且这些冗余在引用对象销毁后可以正常被销毁。显然UnLoad(false)的后果好接受些。

想要做到最理想的卸载策略，则需要做好准确的引用记数，在引用记数为0时调用UnLoad(false)（若引用记数真的准确的话，此时UnLoad(false)与UnLoad(true)应该等价），卸载AB对象占用的内存。

目前采用的策略是：在场景中不自动卸载AB，仅在切场景时，先销毁场景上的所有GameObject，再对缓存中的所有AB执行UnLoad(false)。

没有采用引用记数方案的原因有两个：

1. 引用记数使用麻烦，需要所有开发人员特别注意资源的克隆和销毁，并且记数出错之后很难排查，维护困难，缺少能很好解决这些问题的完善方案。
2. Unity 5.x之前，AB对象会在内存中占用相当于文件大小的内存。Unity 5.x之后，AB可以采用LZ4压缩格式，采用LZ4压缩格式的AB对象在内存中仅占用极少的内存，实测创造者项目内存中有1000个AB时占用5.6m，可以不必频繁卸载AB。

## 问题及解决方案

* AssetBundle中资源重复打包的问题

问题详情：在上面打包策略一节中，我们通过设置ABName处理了项目中资源被重复打包到不同AB的情况，但是这个策略没办法处理引用到Unity内置资源的情况，因为内置资源无法单独设置ABName，所以在合理的打包策略下依然出现资源被重复打包的情况，基本上都是引用了内置资源的原因。

问题解决：修改资源，尽可能不要使用内置资源，比如UISprite、Default-Material、内置shader等等，可以新建贴图材质shader进行替换，内置shader可以在Unity官网下载后放入工程，则能被当作项目资源正确打包。

* Default-Material及Standard shader冗余打包的问题

问题详情：这两个都是Unity的内置资源，如果是prefab中引用到的话，可以按上一条提到的方式处理。但也有例外，有可能是fbx文件中引用到，此时fbx的renderer和material都是灰色不可编辑状态，则没办法直接进行替换。Unity在打包fbx时有这样的规则：如果fbx和引用它的prefab打包在一起，则只会收集prefab中引用的material，fbx中的material不会被打进包里，如果fbx单独设置ABName打包，则会将fbx引用的所有资源都打进包里。

问题解决：一种就是保持fbx和prefab一起打包，则没有此问题。但如果有多个prefab引用同一个fbx，为了不重复打包通常会设置fbx独立打包，则需要第二种解决方案：通过编辑器代码的AssetPostProcess接口在导入fbx的时候设置importMaterial=true，然后自动将生产的Material删除，这时项目中的fbx就会丢失material变成紫色，此时打包出来fbx就没有包含Default-Material。而游戏中通常是通过prefab加载模型的，使用的是prefab上关联的material，不会受影响。

* DontDestroyOnLoad被卸载的问题

问题详情：在创造者项目中遇到这种情况，有一些UI面板预加载后立即SetActive(false)并设置DontDestoryOnLoad，接着切场景时触发UnLoadAllAssetBundle，即对所有已加载AB执行UnLoad(false)，此时这些从没有卸载的UI就会出现资源引用错乱，贴图和字体都无法正常显示。经测试，如果预加载后没有SetActive(false)或没有设置DontDestoryOnLoad，都不会出现这个问题，并且根据文档，UnLoad(false)应该仅卸载AB对象，不应该影响到已经从AB中加载出来的资源对象，所以这个问题可以认为是Unity的一个Bug。

问题解决：针对需要DontDestroyOnLoad的资源，将它依赖的所有AB也设置成DontDestroyOnLoad，控制切场景时不卸载这些AB，则一切正常。

* Shader变体丢失问题

问题详情：在创造者项目中遇到这种情况，shader打成AB后，地图上的树失去了cutout效果，经查，这种情况是因为使用shader feature 定义的变体没有被编译到AB包中，Unity的打包机制是，如果Shader与Material一起打包，则会根据Material的情况将用到的变体编译出来，如果Shader单独打包，则不会包含任何变体。

问题解决：

1.使用dummy materials / ShaderVariantCollection 与shader打在同一ab内。

2.使用multi\_compile替换shader\_feature，则始终会包含所有变体。

创造者项目中最终采用了1的方式，创建一些dummyMaterials引用有问题的shader，在material中勾选所需变体，将这些material与所有shader打到同一个AB包中。

* Sprite同时作为Texture使用造成冗余

问题详情：在银河项目中遇到这种情况，一些贴图同时作为Sprite和Texture被其它资源引用，贴图本身设置了ABName，打包AB后发现，引用Sprite的资源能正确处理依赖关系没有冗余，而引用Texture的资源包都冗余了一张目标贴图。目前认为这是Unity打包的一个bug，ABName依赖关系仅应用在Sprite上而没有应用Texture对象上。目前回避这个问题的方法是一张图设置成Sprite就不要再用作Texture，实在需要，则复制一张图仅做Texture使用。

## 

## 编辑器使用说明

待补充。