AssetBundle框架设计

《AssetBundle框架设计》　讲师:刘国柱

AssetBundle框架设计

- AssetBundle技术在Unity引擎的网络开发中占据重要地位，不仅仅是网络大型游戏热更新的标配技术，同时也是中小网游甚至单机游戏(包含虚拟现实等项目)的重要优化技术体系。

- AssetBundle技术非常重要，但是却不是容易掌握的技术。本篇总结项目开发中应用AssetBundle技术可能遇到的各种问题，然后给出解决方案。最后按照方案设计了一套工程化实用全自动打包、加载管理框架。

目录

AssetBundle框架整体设计_问题分析

- 目前（基于Unity2017）AssetBundle技术虽然比之前版本有了很大改进，但是仍然无法进行工程化实战开发，分析有如下部分原因:

- 第1: 实战项目中成百上千的大量资源需要(批量)打包处理，不可能手工维护方式给每个资源添加assetbundle“包名称”。

- 第2: Unity维护AssetBundle包的依赖关系不是很完善，主要体现在Unity仅仅维护包与包之间依赖关系的记录上。（通过每个包创建的*.manifest文本文件实现）。如果要加载一个有多重依赖项的AssetBundle包，则要手工写代码，把底层所有依赖包关系需要预先进行加载后才可以。

AssetBundle框架整体设计_问题分析

- 第3： AssetBundle包的商业应用涉及很多步骤： AB包的加载、AB包依赖关系（要求： 不遗漏、不重复）、资源的提取与释放等。手工以及简单写代码实现功能，将是一项繁重海量工作，效率低下。

- 第4: 某些项目应用中，可能会出现反复加载同一AB包中的重复资源，导致性能降低。

AssetBundle框架整体设计_解决方案

- 分析以上问题，特制定如下解决方案与思路：

- 第1： (针对上述第1条)开发专门标记脚本，自动给指定目录下所有合法资源文件（预设、贴图、材质等）添加标记。

- 第2： (针对上述第2条)通过写专门的脚本读取Unity自动创建的*.manifest文件。自动分析与维护AssetBundle包之间的依赖关系，使得包的依赖关系可以实现循环依赖、自动化加载。

AssetBundle框架整体设计_解决方案

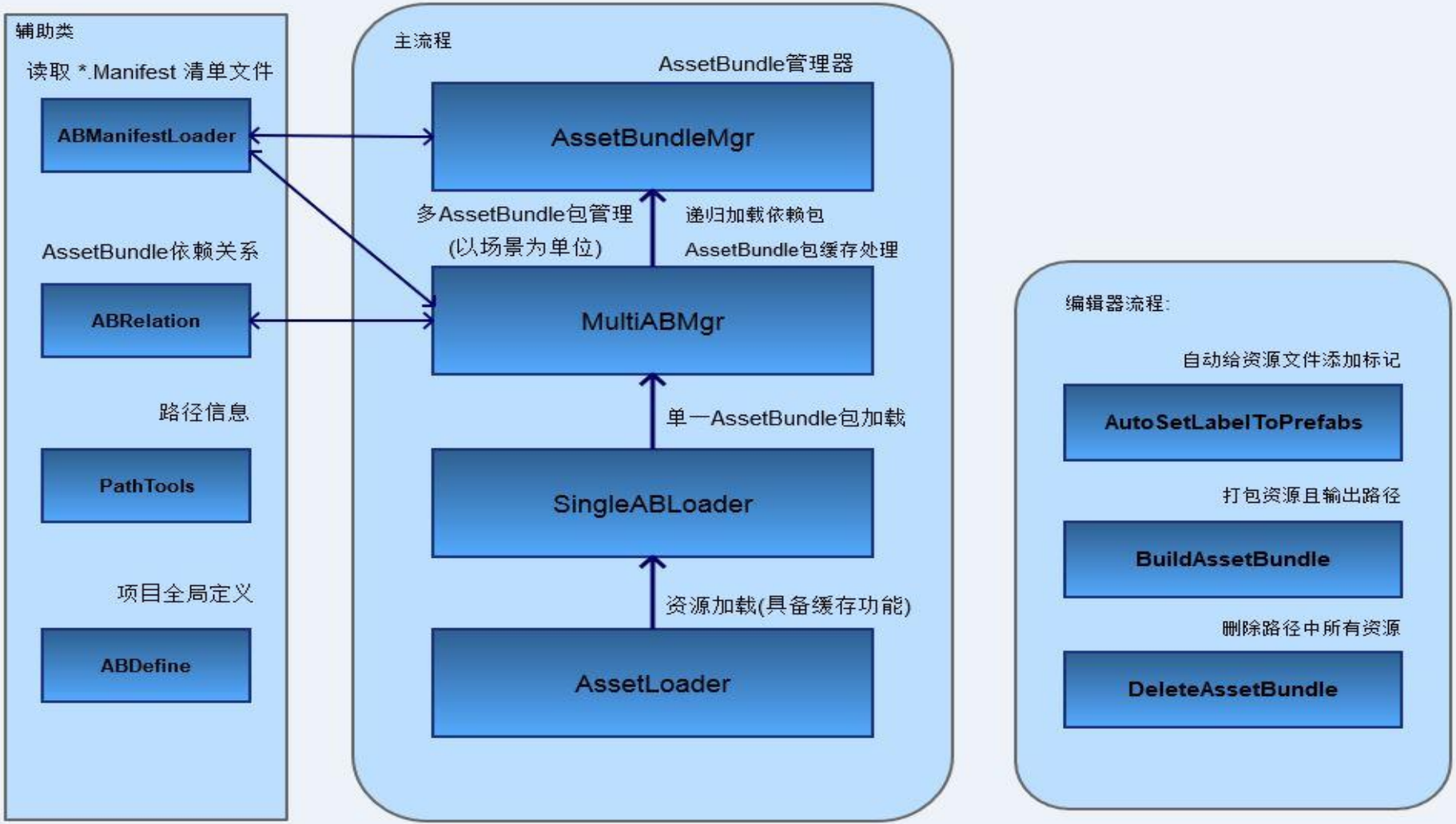
- 第3： (针对上述第3条)开发针对AssetBundle的专门框架。按照一定严格流程解决AB包加载、复杂依赖、资源提取释放等事宜，尽可能让最终使用框架人员，只关心输入与输出结果部分，屏蔽内部复杂性。

- 第4： (针对上述第4条)开发的AssetBundle框架中，需要对AssetBundle包之间以及AssetBundle包内资源做缓存设计，且提供参数开关，让研发使用者自行决定是否应用缓存加载。

AssetBundle框架整体设计

- 按照以上解决思路，特开发一套AssetBundle(注：为方便起见后面简略称呼“AB”)框架项目。

- 此项目包含Unity编辑器中自动标记脚本、创建与销毁打包资源、单一AB包加载与测试脚本、专门读取manifest维护AB包依赖关系，实现递归依赖加载机制的脚本等。


辅助类
读取 *.Manifest 清单文件
ABManifestLoader
AssetBundle依赖关系
ABRelation
路径信息
PathTools
项目全局定义
ABDefine
主流程
AssetBundle管理器
AssetBundleMgr
多AssetBundle包管理
(以场景为单位)
递归加载依赖包
AssetBundle包缓存处理
MultiABMgr
单一AssetBundle包加载
SingleABLoader
资源加载(具备缓存功能)
AssetLoader
编辑器流程:
自动给资源文件添加标记
AutoSetLabelToPrefabs
打包资源且输出路径
BuildAssetBundle
删除路径中所有资源
DeleteAssetBundle

目录

自动化创建AssetBundle

- 自动创建AssetBundle，共分为以下三个脚本，分别来讨论：

- “自动给资源文件添加标记”(AutoSetLabelToPrefabs.cs)
- “打包资源且输出路径” (BuildAssetBundle.cs)
- “删除路径中所有资源”(DeleteAssetBundle.cs)。

自动化创建AssetBundle_自动标记

- 前面学习了基本的AssetBundle知识点。我们了解到给资源打包必须事先给资源作“标记”，标记的名称就是打包之后的包名。且经过测试我们也能发现多个资源打相同包名，就能得到一个含有多个资源的AB包。

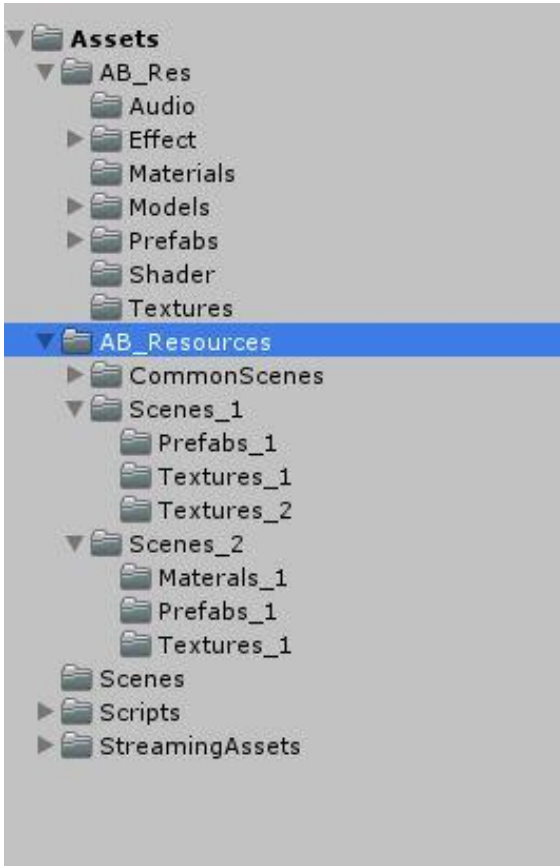
自动化创建AssetBundle_自动标记

- 打包分类依据：

对于一个大型游戏可能含有海量资源，而移动端设备一般内存与性能较低（相对于PC）。为了权衡起见，我们一般对大量的AB包按照“场景”进行分类，也就是说一个游戏在场景开始前只加载本场景用到的资源AB包即可。

- 资源包名的命名规则

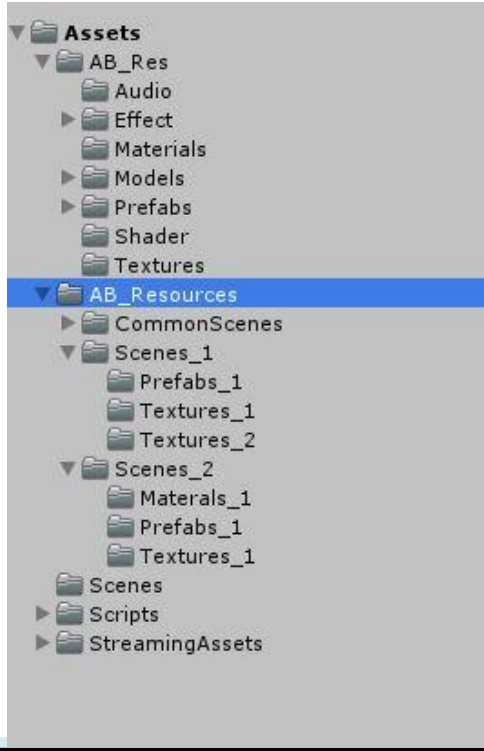
我们对于资源包名的命名规则就确立为：“场景名称”+“功能文件夹名”即可。结合Unity规定，我们进一步确立为：包名称=“场景名称/功能文件夹名”。



自动化创建AssetBundle_自动标记

- 开发（给资源文件）自动添加标记的（编辑器）脚本：
  开发思路如下：

- 定位需要打包资源的文件夹根目录。
- 遍历每个场景文件夹
  - 遍历本场景目录下的所有的目录或者文件。 如果是目录，则继续递归访问里面的文件，直到定位到文件。
  - 如果找到文件，修改AssetBundle 的标签（label）具体用AssetImporter 类实现，修改包名与后缀。



自动化创建AssetBundle_打包资源

- 打包资源核心API讲解：

```
BuildPipeline.BuildAssetBundles(strABOutPathDIR,BuildAssetBundleOptions.None,BuildTarget.StandaloneWindows64);
```

- 第1个参数“strABOutPathDIR”表示打包输出路径，也就是打包之后的AB包存储在项目的什么路径下。

- 第2个参数“BuildTarget.StandaloneWindows64 ”表示打包的不同平台，如果是笔者定义的StandaloneWindows64参数表示64位Windows环境下发布项目使用。这里读者最终打包使用的环境需要通过本参数进行正确设置，因为不同环境下打出的AB包是不能通用的。

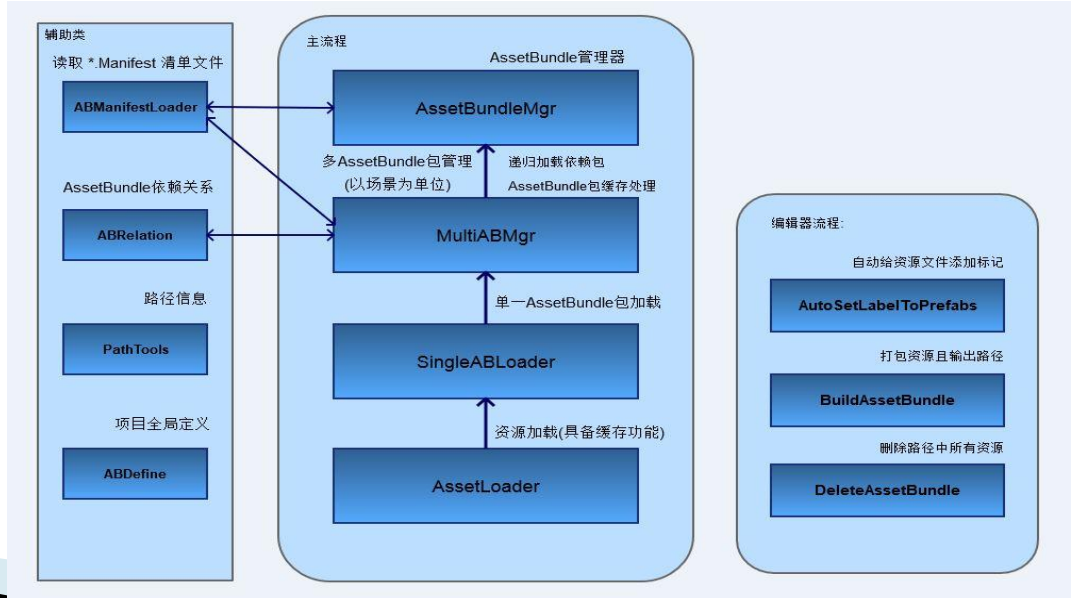
自动化创建AssetBundle_删除资源

- 对于已经打包成功的输出路径，如果需要二次重新打包，则需要删除已有包体

目录

单一AssetBundle包的加载与管理

- AssetLoader.cs （AB包内资源加载）： 完成AB包内资源加载、(包内)资源缓存处理、卸载与释放AB包、查看当前AB包内资源等。
- SingleABLoader.cs （WWW加载AB包）：完成WWW加载、定义（加载完毕）回调函数、以及通过引用AssetLoader.cs脚本调用卸载与释放AB包、查看当前AB包内资源等功能等。

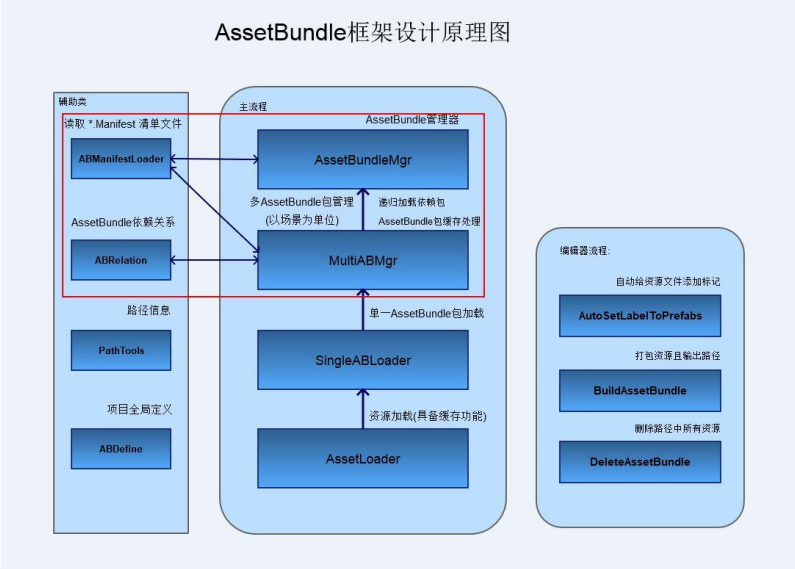


目录

AssetBundle整体管理

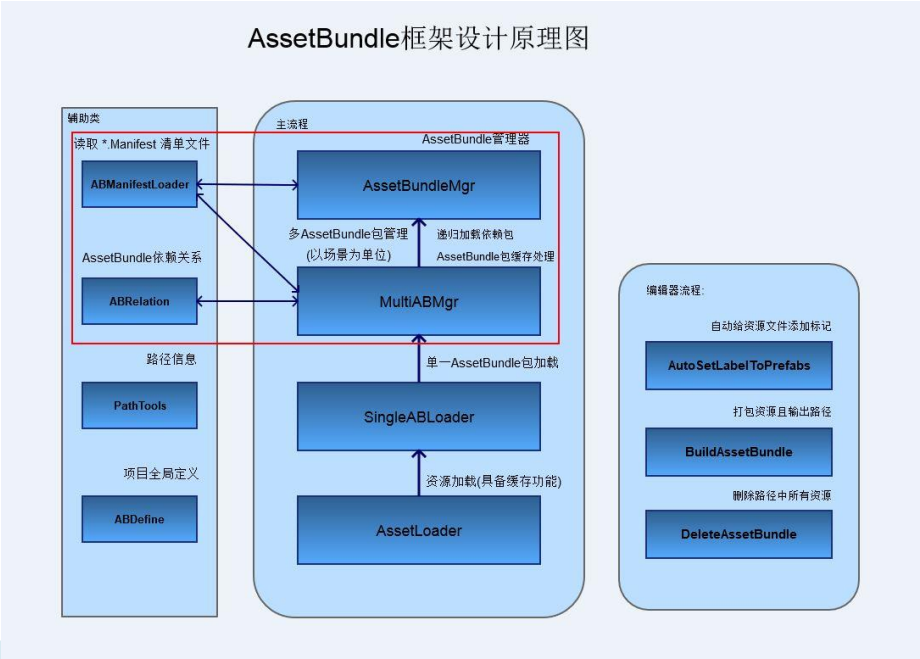
➢AB框架的整体管理主要包含两大部分：

- 第一部分： 主流程的AssetBundleMgr脚本。通过调用辅助类“ABManifestLoader”脚本，来读取Unity提供的Manifest清单文件。这个清单文件是编辑器打包脚本(即：BuildAssetBundle.cs)批量打包时，所产生的记录整个项目所有AB包依赖关系的文本文件。本框架为了管理海量AB包资源，把整个项目分为“场景”为单位进行管理，然后每个“场景”再处理AB包的加载与管理。



AssetBundle整体管理

- 第二部分：主流程的MultiABMgr脚本。这个脚本通过获取Manifest 清单文件，循环遍历需要加载AB包所有的底层依赖包。然后给每个AB包都记录相关依赖与引用关系，这些关系都记录在对应ABRelation 对象中。



AssetBundle整体管理_清单文件类

- ABManifestLoader类负责读取项目清单文件。主要功能： 读取清单文件（整个项目所有AB包依赖关系），且数据存储在自身“AssetBundleManifest”实例中。

- LoadManifestFile() ： 是加载Manifest清单文件协程类。
- GetABManifest(): 返回"AssetBundleManifest"系统类实例
- RetrivalDependences(): 查询清单文件中所有依赖项目。
- IsLoadFinish: 只读属性： 清单文件是否加载完成

AssetBundle整体管理_AB包关系类

- ABRelation.cs 是记录所有AB包之间相互依赖与引用的关系类，主要完成记录与存储指定AB包中所有的依赖包与引用包的关系集合。

- 依赖关系操作。
  AddDependence():
  RemoveDependence():
  GetAllDependences():
- 引用关系操作。
  AddReference():
  RemoveReference():
  GetAllReference():

AssetBundle整体管理_AB总管理脚本

- AssetBundleMgr是一个脚本，其核心功能：提取“Menifest清单文件”数据，以“场景”为单位，管理整个项目所有的AssetBundle 包。

- 核心字段：

  Dictionary<string, MultiABMgr> _DicAllScenes 保存项目所有场景资源。（一个“场景”包含若干AB包）

AssetBundle整体管理_AB总管理脚本

- 核心重要方法：
  - 通过Awake() 事件函数，调用ABManifestLoader.GetInstance().LoadManifestFile()，加载Manifest清单文件。
  - LoadAssetBundlePackage(); 加载AssetBundle指定包。
  - LoadAsset(); 加载AB包内资源。通过调用MultiABMgr类实例方法“LoadAsset()”来提取指定AB包内资源。

AssetBundle整体管理_多AB包管理类

- MultiABMgr是一个场景中负责多个AB包管理的核心类。其主要功能是获得AB包之间的依赖关系，使用递归方式遍历整个场景调用与加载所有的AB包。

- 核心字段：

  Dictionary<string, ABRelation> _DicABRelation; 缓存AB包依赖关系集合。

  Dictionary<string, SingleABLoader> _DicSingleABLoaderCache "单个AB加载实现类"缓存集合


谢谢大家！
刘老师讲Unity系列教程