# Unity5.x资源

1. Unity资源类型

|  |  |  |
| --- | --- | --- |
| **名称** | **描述** | **支持格式** |
| Audio Clip | 音频 | 音频：.aif .wav .mp3 .ogg音轨：.xm .mod .it .s3m |
| Cubemap Texture | 立方体贴图纹理 |  |
| Flare | 耀斑 |  |
| Font | 字体 | ttf |
| Material | 材质 |  |
| Meshes | 网格 | .FBX .dae .3DS .dxf .obj |
| Movie Texture | 电影贴图 | .mov .mpg .mpeg .mp4 .avi .asf (导入需要QuickTime) |
| Procedural Material Assets | 程序材质资源 |  |
| Render Texture | 渲染纹理 |  |
| Text Asset | 文本资源 | .txt .html .htm .xml .bytes |
| Texture 2D | 二维纹理 | PSD TIFF JPG TGA PNG GIF BMP IFF PICT |

       特别介绍Prefab: 预置是一种资源类型——存储在项目视图中的一种可重复使用的游戏对象。预置可以多次放入到多个场景中。当你添加一个预置到场景中，就创建了它的一个实例。所有的预置实例链接到原始预置，基本上是它的克隆。不管你的项目存在多少实例，当你对预置进行任何更改，你将看到这些更改将应用于所有实例。

 除此之外，想使用Unity不支持导入，或者未经导入的资源，只能使用IO Stream或者WWW 方法。

        注意：AssetBundle不是资源组件，故无法用资源组件的方式载入，只能使用WWW或者AssetBundle相关接口载入与读取

AssetBundle是Unity pro提供的一种用来存储资源的文件格式，它可以存储任一

种Unity引擎能够识别的资源，如Scene、Mesh、Material、Texture、Audio等等，同时，AssetBundle也可以包含开发者自定义的二进制文件，只需要将自定义文件的扩展名改为.bytes，Unity就可以把它识别为TextAsset，进而就可以被打包到AssetBundle中。Unity引擎所能识别的资源我们称为Asset，AssetBundle就是Asset的一个集合。

AssetBundle的特点：压缩（缺省）、动态载入、本地缓存

二、Resource、AssetBundle、Editor模式资源分布和加载

Resource.Load()是Unity最常用的加载资源的方式，这种方式优点是简单易用，所有平台都能使用，只要知道资源所在路径就能加载资源，缺点是使用Resource.Load()方式只能加载在Resource文件夹下的资源，一旦打包以后该文件夹内资源就不能再改变，不能热跟新。

Editor下，还有一种加载方式，UnityEditor.AssetDatabase.LoadAssetAtPath，同样的简单易用，知道资源所在路径就能加载资源，且只要资源位于工程Assets目录下就能加载，不要求一定要在Resource目录下，缺点是只能在Editor模式下使用，只适合开发时使用。

AssetBundle模式，这种模式的资源可以位于任何目录都能够加载到，甚至可以放在远程服务器，且这种方式也是全平台支持，由于资源存放位置没有要求因此可热更新，缺点是这种方式加载的资源需要Unity提前将普通资源生成为AssetBundle然后才能加载，而前两种方式可以加载Unity认可的任何资源，且在加载出具体的asset之前还要先把asset所在AB包和其依赖包加载到内存后才能正确加载出asset，相较前两种方式更复杂。

三、Resource和AssetBundle内存

1、GUID和fileID：Unity会为每个导入到Assets目录中的资源创建一个meta文件，文件中记录了GUID，GUID用来记录资源之间的引用关系。还有fileID（本地ID），用于标识资源内部的资源。资源间的依赖关系通过GUID来确定；资源内部的依赖关系使用fileID来确定。

2、InstanceID：Unity为了在运行时，提升资源管理的效率，会在内部维护一个缓存表，负责将文件的GUID与fileID转换成为整数数值，这个数值在本次会话中是唯一的，称作实例ID(InstanceID)。

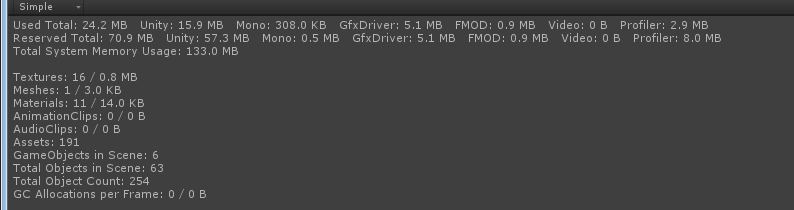
3、加载流程：两种动态加载机制Resources.Load和AssetBundle两者区别不大，Resources.Load就是从一个缺省打进程序包里的AssetBundle里加载资源，而一般AssetBundle文件需要自己创建，运行时动态加载，可以指定路径和来源的。

在加载资源的时候，AssetBundle只会在内存中寻找其依赖资源所在的AssetBundle（寻找方式是通过AB包中记录的GUID和fileID），并自动从中加载出所需资源，AssetBundle在内存中的资源是Prefab的原始资源，进行实例化时，能够引用的资源都是指向AssetBundle中的原始资源。在实例化时一个Prefab时，Prefab所依赖的asset是引用+复制的，GameObject transform 是Clone是新生成的。其他mesh / texture / material / shader 等，这其中些是纯引用的关系的，包括：Texture和TerrainData，还有引用和复制同时存在的，包括：Mesh/material /PhysicMaterial。引用的Asset对象不会被复制，只是一个简单的指针指向已经Load的Asset对象。

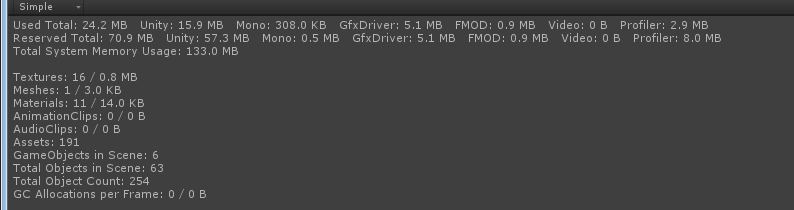
使用AB包时，内存中一般会有AB包所占内存以及从AB包中实例化的asset资源所占内存，在合适的时机释放AB包可以降低内存占用，对于.bytes文本文件可以在实例化以后立即释放AB，对于场景文件和shader文件也可以在加载完成以后立即释放AB包所占用的内存，对于其他的普通资源则需要场景中没有对该包资源的使用后才能释放。

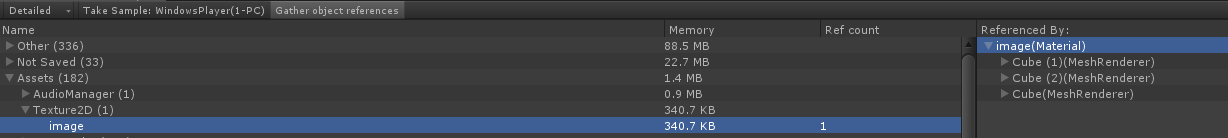
实际使用中对Texture,Material进行测试, 对比可知，纹理内存占用一个cube和三个cube一样大，texture没有被复制多份使用。可以看见cube的材质只有一个且被三个cube引用，texture只被材质引用。Texture在内存中是共享的

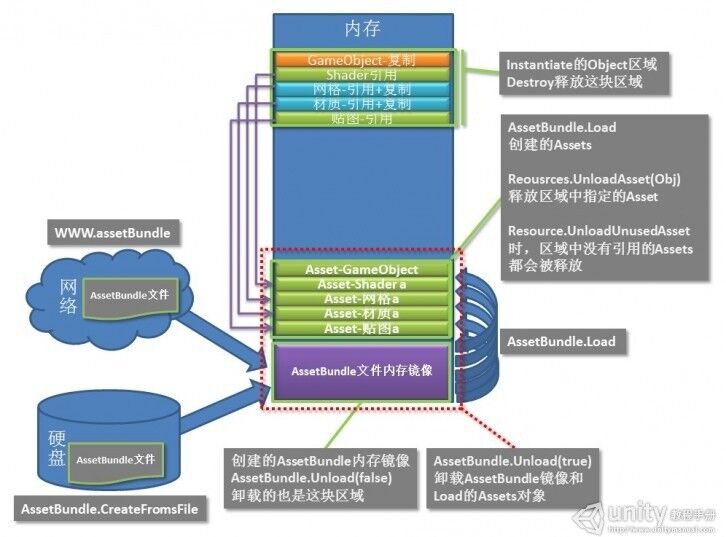
场景中加入一个cube，材质球使用了一张334KB texture



场景中将同一cube复制为三个







打包粒度

使用最小了粒度打包，即每个资源都打成一个单独的AB包。

四、打包

不同资源的加载方式有所不用，因此在打包时一些资源的打包方式有所不同

资源分类：普通、shader、场景

1. 设置包名：包名是AB包的路径，由于某些资源在unity中特殊的加载方式，因此在设置包名时对三种不同的资源有三种不同的包名设置方式。

普通资源：unity在加载普通资源是都是用Resources文件夹下相对路径的方式，普通资源的包名为其在Resources文件夹下的路径+资源名称，如data/config，这样的包名方便在可以在加载AB包时直接定位包的位置

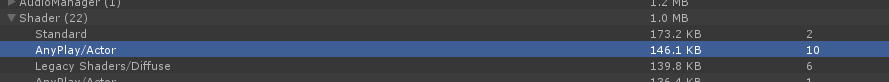
Shader资源: shader文件，特点是文件小io时间短，但是shader加载以后还需要解析，shader的解析是比较耗时的操作。

在实际使用中，将AB包加载进内存后，直接从AB包中使用LoadAsset的方式加载得到shader，若立即卸载AB包，则之后多次加载该shader不能共用，会出现多个shader的情况。若AB包常驻内存，则每次加载shader都会共用之前的shader，不会出现多份相同shader同时存在，因此shader最好是常驻内存，这样可以是shader的内存占用最小。

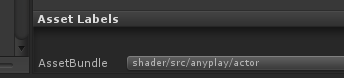
立即卸载AB



AB常驻

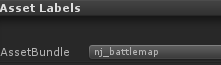


Shader的包名



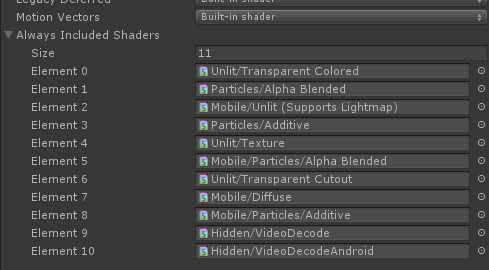
Scene资源：Scene资源和shader资源类似，加载方式只能使用SceneManager.LoadScene(sceneName)来加载，sceneName为场景文件名称。使用scene文件的名称作为scene文件的包名，可以在加载时直接定位包的位置。

场景文件的包名

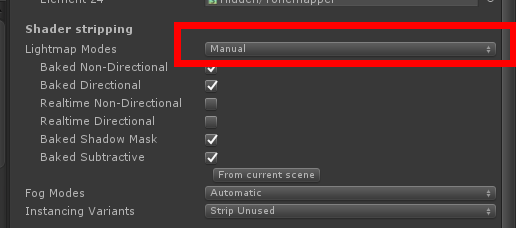


1. 工程设置： 使用AB作为资源时需要有一些特殊的工程设置

Shader，在unity中shader根据来源可分为unity自带shader和自定义shader，自带shader如果在工程Resources文件夹中没有引用到这些shader的资源，在打安装包的时候不会被打入安装包中，在非Editor环境中使用中会出现找不到shader的情况，为了能使用这些shader，需要在projectsetting ->graphicsetting ->alwaysIncludeShaders中进行添加。



Lightmap，Unity 5.x中加入了Shader Stripping功能，在打包时，默认情况下会根据当前场景的Lightmap及Fog设置对资源中的Shader进行代码剥离。这意味着，如果在一个空场景下进行打包，则Bundle中的Shader会失去对Lightmap和Fog的支持，从而出现运行时Lightmap和Fog丢失的情况。 而通过将Edit->Project Settings->Gaphics下Shader Stripping中的modes改为Manual，并勾选相应的Mode即可避免这一问题。



Shader和lightmap的设置都可以通过工具完成，正确配置路径后，使用IncludeAlways Shaders可以自动完成shader和lightmap的设置



1. 打包：设置完毕后，使用BuildPipeline.BuildAssetBundles根据对应平台打包

使用Rebuild，重头开始打包

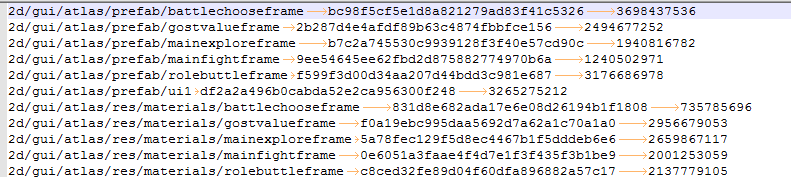
使用Build，会在上一次打包的基础上，只对有变化的资源重新打包，缩短打包时间。



五、AB包版本管理及依赖关系

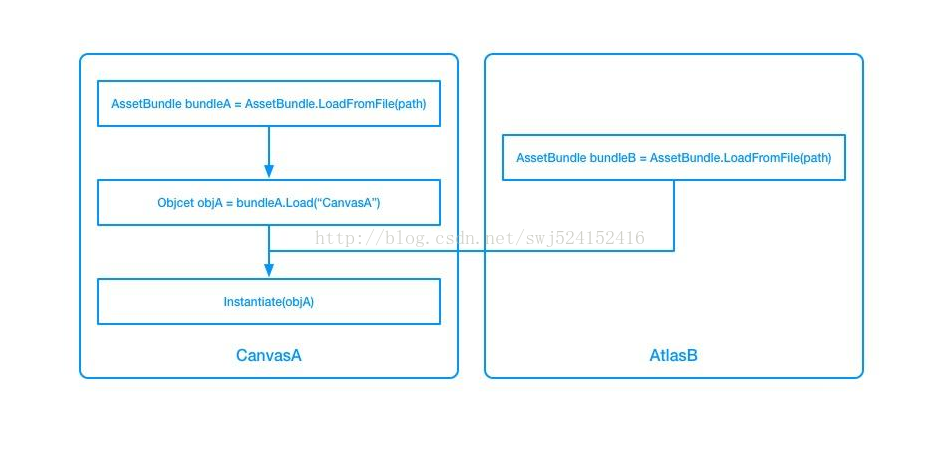
1、版本管理：打包完成后生成bundleinfo文件，记录包名、包hashcode、包crc值用于版本管理，在更新时只需要对比AB包的hash值，不同hash值的AB包需要更新。打包完成后会在资源所在路径下生成bundleinfo.bytes文件，同时在输出目录中会有bundleinfo AB文件，记录下本次打包后所有AB包的数据。

Bundleinfo文件



2、依赖关系：

假设有AssetBundleA与 AssetBundleB两个AssetBundle，AssetBundle中的资源引用了AssetBundleB中的资源，则称AssetBundleA依赖于AssetBundleB。在实例化具体资源之前，需要将其依赖的AB都加载到内存中。

在unity5.x中，所有包的依赖关系生成已经由unity自己完成，打包完成后会在输出目录生成manifest文件记录所有AB包的依赖关系。

Manifest文件



六、检查更新

1、将打包完成的所有资源完整的复制到资源服务器。

2、在客户端初始化下载器：

AssetBundleDownloader.Initialization (); // 初始化

AssetBundleDownloader.itfBaseDownloadingUrl = <http://...>; //设置资源服务器地址

AssetBundleDownloader.GetSingle().CheckUpdate(callback);

七、资源加载

1、加载器初始化：

DigiSky.AssetBundleKit.AssetBundleManager.Initialization();

if (DigiSky.AssetBundleKit.AssetBundleManager.IsExits())

{

DigiSky.AssetBundleKit.AssetBundleManager.GetSingel().itfDownloadPath = path; //设置AB包下载所在位置，一般在Application.persistentDataPath

DigiSky.AssetBundleKit.AssetBundleManager.GetSingel().itfStreamingPath = path; //设置安装包内的AB所在位置，一般在Application.streamingAssetsPath

DigiSky.AssetBundleKit.AssetBundleManager.GetSingel().InitManifest(); //加载manifest文件一般是平台名称

}

2、使用：

普通资源加载：普通资源的AB包在游戏中有物件使用到的时候需要一直保持在内存，

当不在不需要时可调用UnloadUnusedAssetBundles() 将不在有被物件引用到的AB包释放。

LoadAsset<T>(string strPath, string assetName)

同步加载

LoadAssetAsyc(string strPath, string strAssetName, System.Action<Object> finishLoadAssetCallback)

异步加载

进入游戏后只需要加载一次的资源，如配置档文件，可调用LoadSingleAsset 接口，AB包不会缓存，加载完成后自动释放

LoadSingleAsset<T>(string strPath, string assetName, string specialAssetName = null)

Shader资源： LoadShader(string strShaderName)

场景资源:场景资源加载完成后立即释放AB包

同步加载场景

LoadScene(string strSceneName)

异步加载场景，用这种方法加载场景需要手动在场景异步加载完毕后释放AB包

LoadSceneAsync(string strSceneName)