<http://blog.csdn.net/swj524152416/article/details/54022282>

<http://blog.csdn.net/a6627651/article/details/45369677>

<http://blog.csdn.net/leonwei/article/details/18406103>

**什么是AssetBundles？**

AssetBundles是可以把unity中你所创建的文件或任何资源导出的一种格式，这些文件导出后使用的是一种特定的文件格式（.unity3d），这些特定格式的文件能在需要的时候加载到场景中，二这些特定的文件格式可以是模型、贴图、声音甚至是场景文件，它们是先前就被设计好的文件，所有很容易就可以被下载道你所建立的游戏或场景中。

AssetBundles可以是任意类型的文件，只要unity能识别，识别主要是以文件扩展名为准，比如.prefab等，当然，如果你想包含自定义的二进制文件，需要命名这些文件为以“.bytes”为后缀，unity将会把这些文件导为TextAssets

Unity资源的类型：

1）unity内置的常用asset

Fbx/jpg…

2）textasset

Txt、binary等，对应了它的TextAsset类，可以直接读入文本或者二进制byte

3）scriptable object

它是一个序列化的object实例

4）asset bundle

它是一个资源压缩包，里面包含了一堆资源

通常我们自定义的文本类型可以通过textasset或scriptable object来存储，区别在于前者是一个字节或文本流，后者要对应于程序中一个定义了的类型，textasset还通常用于资源的加密

**unity资源加载方式总结：**

**1、Resource.load(string path)**  
优点：同步，使用方便  
缺点：只能加载Resource目录下的资源

**2、WWW**  
优点：灵活，可以加载Application.streamingAssetsPath、Application.persistentDataPath目录下的资源，以及从网络上下载资源  
缺点：异步，如果业务需要按需加载资源，容易打散逻辑

**3、AssetBundle.CreateFromFile(string path)**  
优点：同步，可以加载Application.persistentDataPath目录下的AssetBundle  
缺点：AssetBundle不能压缩，在Android下不能加载Application.streamingAssetsPath下的AssetBundle

**4、AssetBundle.CreateFromMemoryImmediate(byte[] binary)**  
优点：同步，可以加载Application.persistentDataPath目录下压缩过的AssetBundle  
缺点：在Android下不能加载Application.streamingAssetsPath下的AssetBundle

Unity必须通过导入将所支持的资源序列化，生成AssetComponents后才能被unity使用，下面是Unity对Assets的描述：

Assets are the models, texture, sounds and all other “content” files from which you make your game

资源是硬盘中的文件，存储在unity工程的Assets文件夹内。有些资源的数据格式是unity原生支持的，有些资源则需要转换为原生的数据格式后才能被使用。

对象（UnityEngine.Object）代表序列化数据的集合，表示某个资源的具体实例。它可以是unity使用的任何类型的资源，所有对象都是UnityEngine.Object基类的子类

资源与对象是一对多的关系：

**名称 描述 支持格式**

Audio Clip 音频剪辑 aif wav map3 ogg xm mod it s3m

Cubemap Texture 立方体贴图

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

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

Texture2D 二维纹理 psd tiff jpg tga png gif bmp iff pict

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，因此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

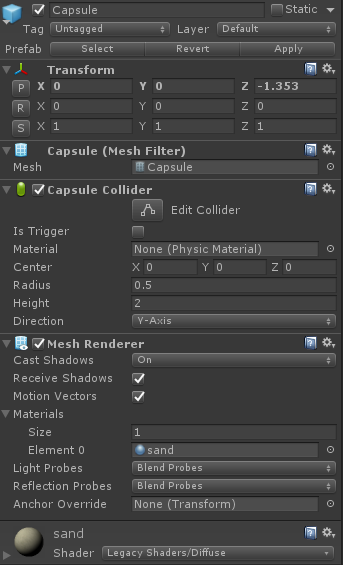
我们可以查看prefab的文本信息，需要设置：edit--project setting—-editor--

Asset Serialization（Force Text）

**Prefab**

预设，类似各种UI编辑器编辑后的输出文件，unity中几乎任何事物都可以打包成预设，然后通过外部文件的形式再加载进程序中，不过是png/jpg等图片图集资源；GameObject Chartater之类的对象资源

一般的预设并不是直接把png、jpg合并成一个文件，而是添加他们的引用，预设与其说是UI界面等的带出文件生成文件，不如直接说是UI界面等的描述文件，他描述了某个界面引用了哪个图片，按钮默认状态是什么，按钮显示Title是什么，比如下面的就是看到的prefab文件：



%YAML 1.1

%TAG !u! tag:unity3d.com,2011:

--- !u!1001 &100100000

**Prefab:**

m\_ObjectHideFlags: 1

serializedVersion: 2

m\_Modification:

m\_TransformParent: {fileID: 0}

m\_Modifications: []

m\_RemovedComponents: []

m\_ParentPrefab: {fileID: 0}

m\_RootGameObject: {fileID: 1000011620259008}

m\_IsPrefabParent: 1

--- !u!1 &1000011620259008

**GameObject:**

m\_ObjectHideFlags: 0

m\_PrefabParentObject: {fileID: 0}

m\_PrefabInternal: {fileID: 100100000}

serializedVersion: 4

m\_Component:

- 4: {fileID: 4000012997215244}

- 33: {fileID: 33000010296479690}

- 136: {fileID: 136000011958227798}

- 23: {fileID: 23000013745129830}

m\_Layer: 0

m\_Name: Capsule

m\_TagString: Untagged

m\_Icon: {fileID: 0}

m\_NavMeshLayer: 0

m\_StaticEditorFlags: 0

m\_IsActive: 1

--- !u!4 &4000012997215244

**Transform:**

m\_ObjectHideFlags: 1

m\_PrefabParentObject: {fileID: 0}

m\_PrefabInternal: {fileID: 100100000}

m\_GameObject: {fileID: 1000011620259008}

m\_LocalRotation: {x: 0, y: 0, z: 0, w: 1}

m\_LocalPosition: {x: 0, y: 0, z: -1.353}

m\_LocalScale: {x: 1, y: 1, z: 1}

m\_LocalEulerAnglesHint: {x: 0, y: 0, z: 0}

m\_Children: []

m\_Father: {fileID: 0}

m\_RootOrder: 0

--- !u!23 &23000013745129830

**MeshRenderer:**

m\_ObjectHideFlags: 1

m\_PrefabParentObject: {fileID: 0}

m\_PrefabInternal: {fileID: 100100000}

m\_GameObject: {fileID: 1000011620259008}

m\_Enabled: 1

m\_CastShadows: 1

m\_ReceiveShadows: 1

m\_MotionVectors: 1

m\_LightProbeUsage: 1

m\_ReflectionProbeUsage: 1

m\_Materials:

- {fileID: 2100000, guid: a40a40a3e330dd64b88859d19e82b8ec, type: 2}

m\_SubsetIndices:

m\_StaticBatchRoot: {fileID: 0}

m\_ProbeAnchor: {fileID: 0}

m\_LightProbeVolumeOverride: {fileID: 0}

m\_ScaleInLightmap: 1

m\_PreserveUVs: 1

m\_IgnoreNormalsForChartDetection: 0

m\_ImportantGI: 0

m\_SelectedWireframeHidden: 0

m\_MinimumChartSize: 4

m\_AutoUVMaxDistance: 0.5

m\_AutoUVMaxAngle: 89

m\_LightmapParameters: {fileID: 0}

m\_SortingLayerID: 0

m\_SortingOrder: 0

--- !u!33 &33000010296479690

**MeshFilter:**

m\_ObjectHideFlags: 1

m\_PrefabParentObject: {fileID: 0}

m\_PrefabInternal: {fileID: 100100000}

m\_GameObject: {fileID: 1000011620259008}

m\_Mesh: {fileID: 10208, guid: 0000000000000000e000000000000000, type: 0}

--- !u!136 &136000011958227798

**CapsuleCollider:**

m\_ObjectHideFlags: 1

m\_PrefabParentObject: {fileID: 0}

m\_PrefabInternal: {fileID: 100100000}

m\_GameObject: {fileID: 1000011620259008}

m\_Material: {fileID: 0}

m\_IsTrigger: 0

m\_Enabled: 1

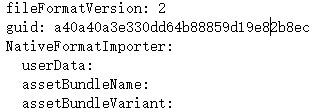
m\_Radius: 0.5

m\_Height: 2

m\_Direction: 1

m\_Center: {x: 0, y: 0, z: 0}

prefab中详细记录了每一项的信息，unity为每个资源都分配了一个guid，然后根据guid来引用各个资源，文件的guid存储于.meta文件中，比如上面的材质meta文件为：



我们发现上面的prefab中的红色部分正好是当前材质sand的guid

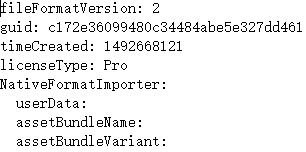
**GUIID与fileID（本地ID）**

**资源（Asset）与对象（Object）是一对多的关系，即一个资源文件可能会包括多个Object**

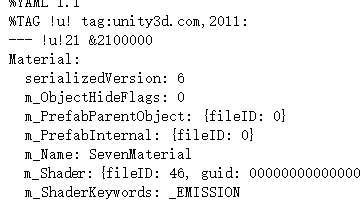
Unity会为每个导入到Assets目录中的资源创建一个meta文件，文件中记录了GUIID，GUID用来记录资源之间的引用关系。还有fileID（本地ID）用于标识资源内部的资源，资源间的依赖关系通过GUI来确定；资源内部的依赖关系使用fileID来确定。

所有UnitEngine.Object都可以应用其他的UnitEngine.Object。这里“其他的Object”可能存在于相同的资源文件中，或需要从其他资源文件中导入。例如，一个材质object通常有一个或多个纹理object的引用，这些纹理object一般是从一个或者多个纹理资源文件中导入的（例如png或者jpg），序列化后，这些引用由两部分数据组成：文件GUID和本地ID。文件GUID用于识别资源（Asset）文件中目标资源（Resource）的存储位置，而本地唯一的ID负责识别单个资源文件中的Object，因为一个资源文件可能会包含多个Object

材质的meta文件



材质文件



在上面的例子中，前面有&符号的数字就是材质的本地ID

如果这个材质的Object位于一个文件GUID为“abcdefg”的资源文件中，则该材质Object的唯一识别符就是文件GUID“abcdefg”和本地ID“2100000”的组合。

**为什么要用文件GUID和本地ID？**

在Unity中，为什么要使用文件GUID和本地ID这套系统呢？答案是为稳定服务，也是为了提供一套灵活的、无关具体平台的工作流程。文件GUID提供了文件存储位置的抽象，这样一个文件GUID就对应一个具体的文件，这个具体的文件存储在什么位置也就无关紧要了。因此我们才能随意移动这个文件而不破坏所有相关Object对这个文件的引用。

任何资源（Asset）文件中都可能含有（或通过导入产生）多个UnityEngine.Object资源（Resource），因此需要一个本地ID来对其中的Object做明确区分。

如果与资源文件相关联的文件GUID丢失，则所有对该资源文件中的Object的引用都会被破坏。这就是必须保证.meta文件具有和资源文件相同的文件名并存储在同一目录下的原因。注意Unity会重新生成丢失或被删除的.meta文件。

**Unity编辑器维护映射表**

Unity编辑器负责维护一张文件路径与文件GUID之间关系的映射表。只要资源文件被读取或导入，这个映射关系就会被建立，映射会将资源的具体位置和资源的文件GUID进行关联。Unity编辑器处于打开状态时，假设一个文件的.meta意外丢失，并且该资源文件的路径没有改变，编辑器可以保证这个资源会被分配到相同的文件GUID。

如果在Unity编辑器处于关闭状态时丢失.meta文件，或资源文件被移动但没有移动对应的.meta文件时，所有对资源文件中的Object的引用都会丢失。

**InstanceID（实例ID）**

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，就是实例ID

程序启动时，实例ID缓存与所有工程内建的对象(例如在场景中被引用)，以及Resource文件夹下的所有对象，都会被一起初始化。如果在运行时导入了新的资源，或从AssetBundle中载入了新的对象，缓存会被更新，并为这些对象添加相应条目。实例ID仅在失效时才会被从缓存中移除，当提供了指定文件GUID和fileID的AssetBundle被卸载时会产生移除操作。

    卸载AssetBundle会使实例ID失效，实例ID与其文件GUID和fileID之间的映射会被删除以便节省内存。重新载入AssetBundle后，载入的每个对象都会获得新的实例ID。

**AssetBundle**

AssetBundle是unity支持的一种文件存储格式，也是unity官方推荐的资源存储于更新方式，它可以对资源（Asset）进行压缩，分组打包，动态加载以及实现热更新，但是AssetBundle无法对Unity脚本进行热更新，因为其需要在打包时进行编译。