# **Unity资源加载入门**

作者：[枸杞忧天](http://gad.qq.com/user/index?id=1075523)

链接：<http://gad.qq.com/article/detail/42494>

**引言**

Unity的资源加载及管理，基础且重要。此篇文章作为近期梳理项目内资源管理器的一个小总结，尝试尽量用人话将Unity对资源管理的关键点梳理清楚，个人觉得比较适合像我这样刚入门且对AssetBundle还不甚了解的家伙。

**我理解的资源管理**

举一个不恰当的例子来描述我所理解的资源管理(因为我实在想不出更合适的例子了)，想象一个画面：一个表演者，站在一个台子后面，面向观众，按照规定的剧本，操作着台子后面不被观众看到的箱子，从里面不断的取出和放回各种新鲜的玩意儿，一会这么组合，一会那么拆散，博观众的眼球，最终完成表演。

我没有当表演者的经历，虽然我很想尝试，但想想也觉得这肯定不容易：

1、如果箱子里的东西都太大，拿起来会很费劲。

2、如果太小呢？恐怕会拿很多次。

3、不用的道具不收？放在台子上会影响接下来的表演。

4、用过的道具收了吧。万一收了后面需要的道具，一会儿用的时候还要再费劲拿一次，不拿的话吧还容易导致表演失败。

**带着问题看文章**

是选择合适的时间取出资源，并在合适的时候释放它们，尽可能保持较低的内存占用；还是选择让资源常驻内存，换取更快的读取和计算速度？如何在时间和空间上做出平衡，才能最大的提升游戏体验？我以为这些就是资源管理的目标和意义。可惜这并非易事。这不仅需要结合项目的实际情况，更需要丰富的实战经验。

**但是在这里，你将不会看到任何可以参考的经验或建议，因为我也不知道啊。**

**无论你是否单身，在Unity的世界里，你都不愁找不到对象，因为一切都是对象。**

无论是**纹理**、**音乐**还是**预制体**，在进入Unity的世界后，都变成了各种**对象**供我们使用，例如纹理转变为Texture2D或Sprite，音效文件转变为AudioClip，预制体变成了GameObject等等。这个由Asset(**资源文件**)转变为Object(**对象**)，从磁盘进入内存的过程，就是**实例化**。而对资源进行的管理，本质上是对Object的管理。



“小当家，这个黄金炒饭是怎么加载出来的？”

**简单介绍一下Unity加载资源的流程**

在介绍Unity的资源加载机制之前，先举一个生活中的例子，来辅助我们了解Unity是如何工作的。

因为我从小热爱欧洲文学，所以在这就拿我最喜欢的《三国演义》做例子，我们都知道书中多次提到“**集齐七颗龙珠，就可以召唤神龙，并帮你实现一个愿望**”这种说法。



咸鱼都有梦想，何况一个上了岁数的程序员呢？但是很可惜，我们一颗龙珠都没有，为了凑齐这七颗龙珠，我们首先要知道它们分别在哪。一摸左兜，哎？发现了一本《**召唤神龙的小诀窍**》，里面记录了召唤神龙所必须七颗龙珠的所在**位置**、**大小**、**颜色**以及**如何使用**等非常关键的信息。



根据《召唤神龙的小诀窍》指引，我们知道原来第一颗龙珠藏在了素有小巴黎之称的**北京通县**，可是通县在哪儿呢？一摸右兜，原来这还有一本1986年出版的《**中国地图**》。那就放心了，出发吧！

终于，历经了81难，我们来到了目的地并最终找到了这颗龙珠。费这么大劲找到的龙珠，当然应该认真记录下来，于是我们马上掏出一个黑皮小本本，认真的记下：“**第一颗龙珠放在背后小书包的左边缝有一个机器猫的侧兜里...**”。

...

最终，经历了无数艰难险阻，我们凑齐了七颗龙珠（所以说人只要肯努力，老天就一定回馈你，**至少让你知道你浪费了时间啊**）。金光一闪，我们召唤出了神龙... 后面实现了什么愿望我们不谈，因为谁没有点小秘密呢。

**现在，让我们来回顾一下整个过程**

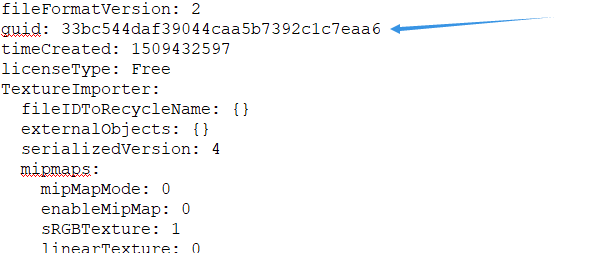
1、这条召唤出来的神龙，就好比我们想要**实例化的对象**，就比如**游戏对象**吧，因为它相对复杂些。而这七颗龙珠呢，就好似组成这个游戏对象所必须的各种**组件**(Component)、**纹理**(Texture)、**网格**(Mesh)等等。

2、《召唤神龙的小诀窍》就好比我们读取的这个**.prefab**文件，它记录了组成这个**GameObject**所必须的**其他对象**以及它们的位置。

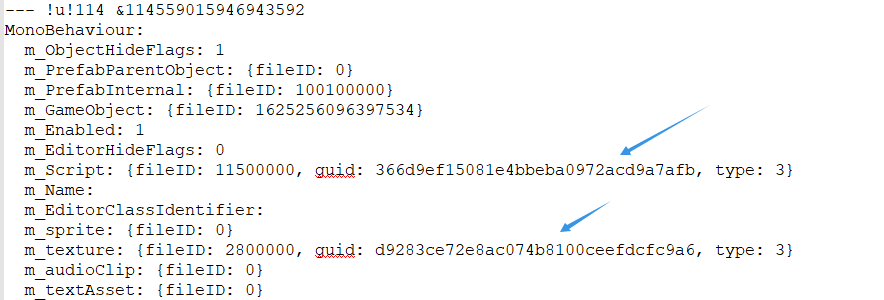
重点来了：**File GUID** 及 **Local ID**。

**File GUID**

Unity会为每一个加入到Assets文件夹中的文件，创建一个同级同名的.meta文件，虽然文件类型的不同会影响这个.meta的具体内容，但它们都包含一个用来标记文件身份的File GUID。



例如，如果一个资源引用了另一个外部资源，比如一个Prefab引用了其他脚本、纹理或Prefab等，则一定会标明引用资源文件的File GUID。



**Local ID**

如果说File GUID表示为文件和文件之间的关系，那么Local ID表示的就是文件内部各对象之间的关系，打开一个**\*.Prefab**文件可以很清晰的看到：



一个对象通常是由**一个**或**多个对象**构成，每个记录在&符号后面的数字都是一个Local ID，每一个Local ID也表示这它将来也会被实例化成一个对象。也就是说，当一个prefab文件要实例化成一个GameObject时，它会自动尝试获取其内部Local ID所指的那个对象。如果这个所指的对象当前还没有被实例化出来，那么Unity会自动实例化这个对象，如此递归，直到所有涉及的对象都被实例化。

3、我们可以发现手中没有龙珠，是因为我们手中的黑色小本本，并没有记录龙珠装在书包的那个位置里；同样，Unity通过**Instance ID**，来获取或判断一个对象是否已经被加载完毕。**Instance ID由File GUID和Local ID转换而成**，可以简单理解成**是记录了资源所在内存地址的写着数字的钥匙牌**。

每当Unity读入一个File GUID和LocalID时，就会自动将其转换成一个简单好记的数字牌，因为通过File GUID和Local ID定位资源的效率并没有直接解引用一个地址那么快。

如果发现这个牌上并没有挂着一把钥匙，表示当前这个这个资源还在磁盘中，尚不在内存里(没有加载)；相反，如果这个牌子上有一把钥匙，表示这个资源已经被加载完毕，你可以快速的找到并使用它。

**Unity会在项目启动后，创建并一直维护一张“映射表”，这张映射表记录的就是File GUID、Local ID以及由它们转换而成的Instance ID之间的关系，这样下次在请求资源时就可以快速的通过查看钥匙牌来获取资源了。**

4、刚才的例子里，因为没有龙珠(资源没有加载)，因此我们必须经历一场前往小巴黎的历险(LoadingAsset)，而能够帮助我们准确定位北京通县的86版《中国地图》，可以近似理解成是**Unity维护的一套将GUID和FileID解析为数据源地址的机制**，这套机制中的信息，来自于：

(1) 场景加载时，Unity收集了与该场景关联的资源信息。

(2) 项目启动时，Unity收集了所有Resources文件夹下的资源信息。

(3) 读取AssetBundle时，Unity获取了AssetBundle文件的头部信息(Header)。

**可以理解为：随着Unity知道更多的信息，这套机制将能够解析并定位更多的GUID和FileID。**

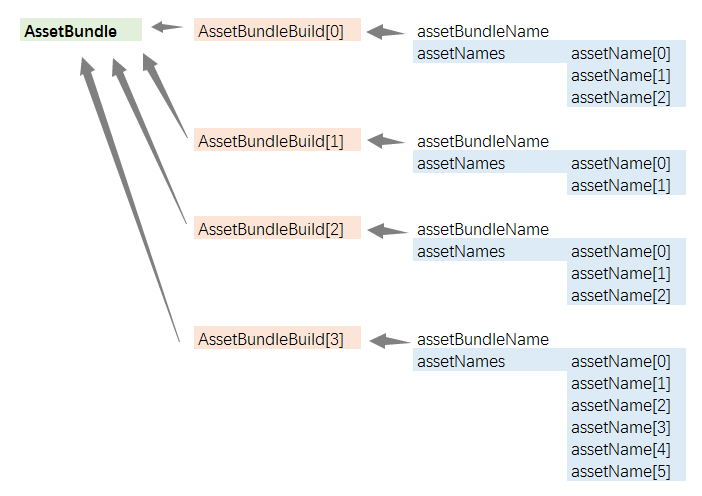
5、当我们费劲千辛万苦找到龙珠后，记录在小本本上的7条位置，就好比7个能帮助够准确定位内存位置的Instance ID。想象一下，当我们下次再看到诸如《三颗龙珠召唤小神龙》这样的小诀窍（**另外一个\*.prefab**），便可直接打开小本本(**查询映射表中的Instance ID**)，对着编号及位置从书包里掏出龙珠(**对InstanceID所指的内存地址进行解引用**)，啪啪啪一操作，小神龙这个游戏对象就能很快被召唤出来了，再也不用去什么通县了，可以节省大把时间，想想就觉的美滋滋呢。

**AssetBundle**

AssetBundle(**阿赛特邦豆**)是Unity官方推荐的资源加载方式，网上对AssetBundle的介绍有很多，且在了解了Unity对资源的加载机制后，其本身没有什么特别难以理解的地方了，因此在这不过多介绍，仅挑选几个关键点进行阐述。

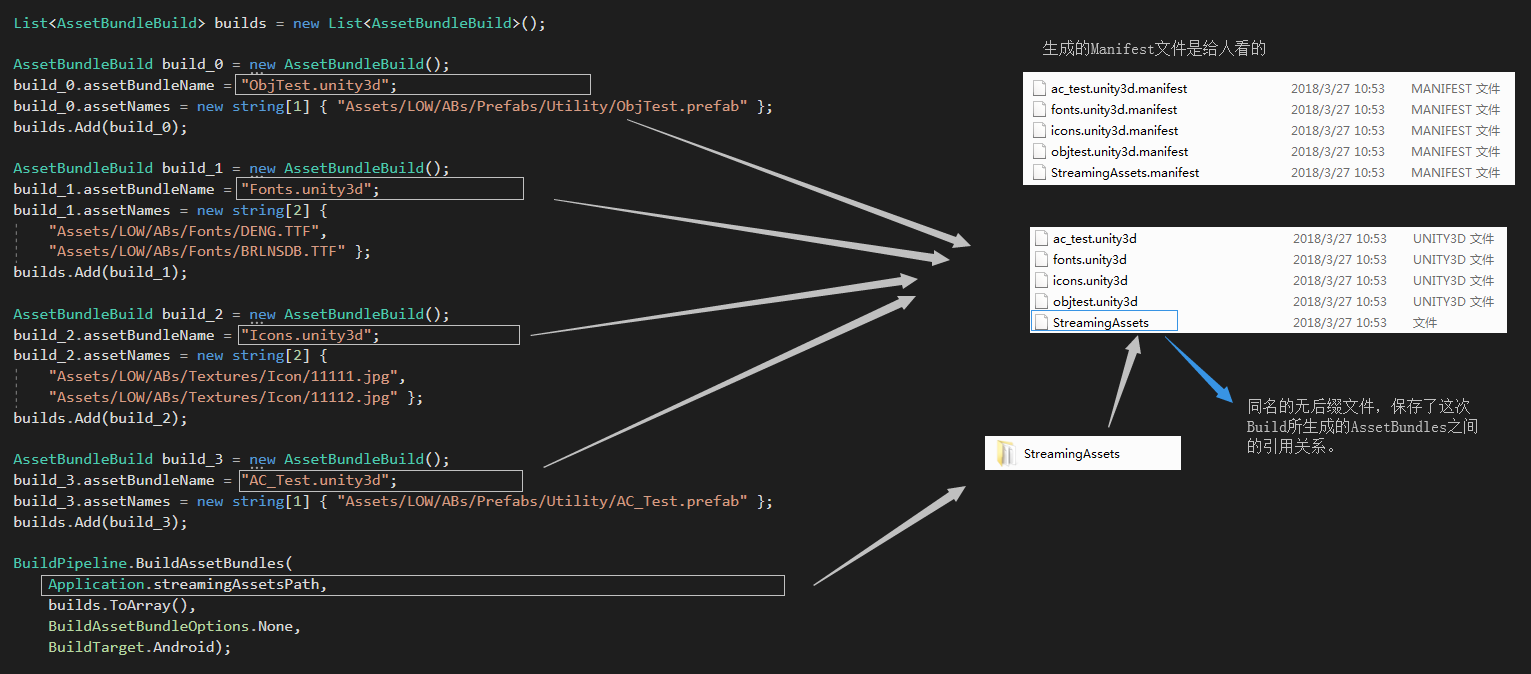
**AssetBundle的生成**

生成AssetBundle有很多种方式，在此仅简单说一下比较常用的方式，使用**BuildPipeline**生成AssetBundle文件。



每一次调用**BuildPipleLine.BuildAssetBundles**时，将会生成一批AssetBundle文件，具体数量根据传递AssetBundleBuild数组决定，**每一个AssetBundleBuild对象将对应一个AssetBundle及一个同名+.manifest后缀文件**。其中AssetBundle文件的后缀用户自行设置，比如".unity3d"，".ab"等等；而.manifest文件是给人看的，里面有这个AssetBundle的基本信息以及非常关键的资源列表。

除了AssetBundleBuild数组所定的AssetBundle外，还将额外在output路径下生成的一对与output文件夹同名的文件及一个同名.manifest后缀文件。这个同名文件**可厉害了**，它记录了这批次AssetBundle之间的相互依赖关系。当然.manifest文件还是给人看的，我们可以用它分析资源间的依赖关系，但是在项目实际运行时，Unity并不会关心它。



可以通过这张图来看一下每次Build后资源的对应关系，当然这都不如你自己亲自Build一次看的清楚。

**AssetBundle的加载**

根据AssetBundle文件所在的位置(本地、远端)，AssetBundle有不同的加载方式，在此仅总结最常用的本地AssetBundle文件加载。

我个人将AssetBundle拆分理解为：**Bundle加载**和**Asset加载**两部分。因为AssetBundle文件可以从功能上分为两大块：

**1、记录文件标记、压缩信息、文件列表的Header部分；**

**2、记录资源实际内容的Data部分。**

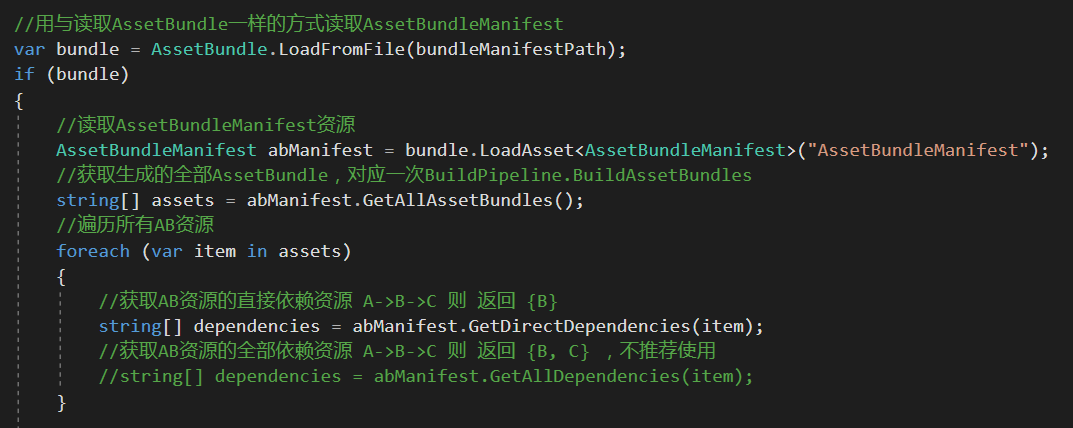
当使用**AssetBundle.LoadFromFile**或**LoadFromFileAsync**时，在pc平台及移动平台上，unity**仅会为我们读取AssetBundle的header部分，并不会将bundle的data部分整个读入内存。**

当调用上一步生成的AssetBundle对象读取具体资源时（**LoadAsset, LoadAssetAsync, LoadAllAssets**），Unity会参考已经缓存的文件列表，找到目标资源在data部分的位置并读入到内存中。

**如果一个资源引用到了其他资源，则必须要先读入被引用资源的AssetBundle文件，否则就会发生引用Miss**。这就好似召唤神龙时，通过《召唤神龙的小诀窍》得知第一颗龙珠在北京通县，但是当打开《中国地图》时，北京的地方被抠了一个窟窿，我去，这样我们就无法通过它准确定位龙珠位置了，只有六颗龙珠召唤出的神龙，当然有一部分是Miss喽。

为了避免上面Miss的情况，在加载资源时，**首先需要将该资源的依赖项全部加载完毕**，不过**仅需加载依赖资源的AssetBundle文件**。也就是说，我们只要将该依赖AssetBundle的Header部分加载（AssetBundle.LoadFromFile或LoadFromFileAsync）就可以，这样在真正读取Asset时，Unity会自动处理好真实依赖的Asset，我们不用操心。

AssetBundle的依赖关系如何读取呢？加载上面提到的那个**很厉害的文件**就可以了。



非常简单的获取依赖关系的方法，通常会在项目启动时将全部依赖关系保存下来。

**AssetBundle的使用**

当AssetBundle被成功加载后，调用该Assebbundle对象的**LoadAsset、LoadAllAssets**或对应的**异步**版本即可加载资源，也就是实例化对象。**如果这个对象已经被加载过，Unity并不会重复加载**，还记得之前所说的映射表么，被加载过的资源就好比挂上了数字牌的钥匙，直接对地址解引用即可。

**AssetBundle的卸载**

如果说AssetBundle真的有什么容易出问题的地方，那恐怕就是卸载了。

在这里只说最常用的这个卸载方法吧：

**public void Unload(bool unloadAllLoadedObjects);**

一个被加载过的AssetBundle可以通过调用Unload来卸载这个Bundle下所有的Asset。但是调用这个函数时传入的参数对卸载结果**影响甚大**。

Unity官方对这个函数的讲解非常详细，配图也非常直观，因此我只是简单总结一下。

**相同点：**

无论传入参数为 **true** 或是 **false**，**调用Unload都可以Destroy当前AssetBundle对象**，**释放之前从AssetBundle文件中的Header部分所获取的信息**。当然，被释放的AssetBundle对象无法再使用诸如LoadAsset、LoadAllAssets等函数加载资源。

**不同点：**

unloadAllLoadedObjects == **true：**

不仅Destroy了AssetBundle这个对象，而且**这个AssetBundle下包含的所有对象，只要实例化了，有一个算一个，统统释放掉。**

感觉就像

foreach(Object asset in assets)

｛

if(asset != null)

｛

**delete asset;**

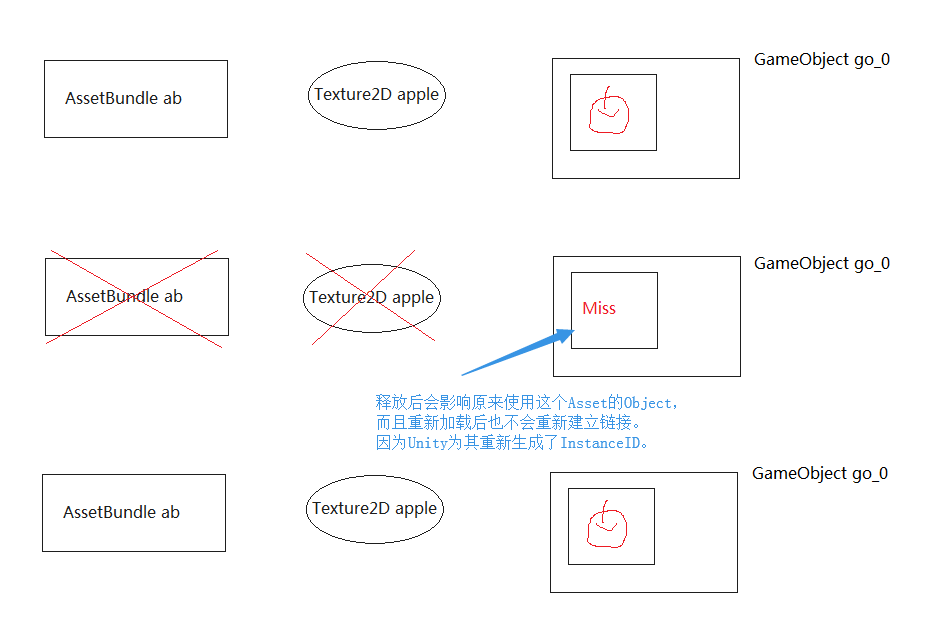
asset = null;

｝

｝

比如你通过ab.LoadAsset(apple)后，将apple设置给go\_0的一个Renderer，如果这时候ab.Unload(true)，那go\_0就傻了，咋回事儿啊，图咋没了呢？WTF啊。

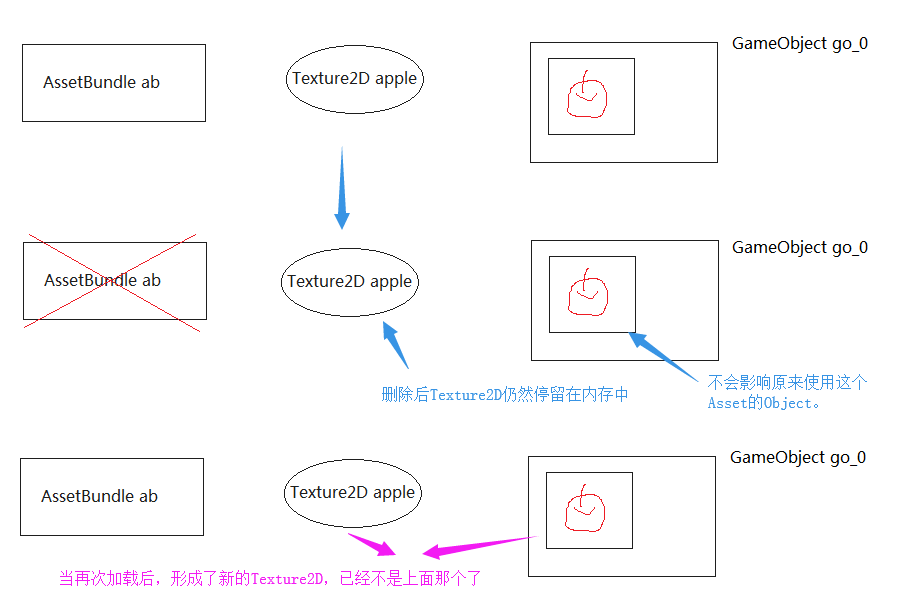
它的好处是：**不会有重复资源问题的情况发生，每次都处理的干干净净**。



unloadAllLoadedObjects == **false**：

仅仅Destroy了AssetBundle这个对象，但是**并没有释放这个AssetBundle下的任何Asset**，因此如果有对象引用了这些Asset，也不会有问题。

它的**风险(代价)**是：下次再Load这个AssetBundle，并且通过这个AssetBundle重新读取了这个Asset，**会在内存中重新创建一份**，这样如果之前的Asset没有被释放，那么现在内存中就有**两份**Asset了。



这种情况如果频繁发生，便意味着内存中有很多资源将“**不受控制**”，容易引发内存占用过高的问题，而释放这种不受控的资源，仅有**两种**方式：

1、当**没有对象引用**到这些不受控资源时，每次调用**Resources.UnloadUnusedAssets**，回收之。

2、加载场景时，如果加载模式没有设置为**LoadSceneMode.Additive**，则会自动调用**Resources.UnloadUnusedAssets**。

同样，再举一个生活中的小例子以阐述这两种释放的差异吧：

小A交女朋友时喜欢送心形的石头给对方，这天小A认识了一个女孩，并确定了关系，送了一个精心挑选的心形石头给她，海誓山盟又云雨一番后，第二天由于感情不和等原因两人分手了。小A是个**暖男**，他为了女孩能彻底忘记优秀的自己并开始一段新的感情，约见了女孩，将之前送给女孩的石头拿(搬)走了，从此注销了微信消失在茫茫人海中。



确实，小A喜欢强壮的女孩，因为这样比较有安全感

小B交女朋友时也喜欢送石头给对方，周一小B认识了一个女孩，并确定了关系，送了一个精心挑选的石头给她，海誓山盟又云雨一番后，第二天由于感情不和等原因两人分手了。但是小B家里是开石材加工场的，他并不关心这块石头，”送了就送了吧，至少我经历了浪漫的爱情“，小B这么想。并注销了微信消失在茫茫人海中...达1天之久。

周二的时候小B重出江湖，并认识了一个新的女孩，确定了关系，第三天...第四天..啪啪啪...第七天，第二周的时候，江湖上就出现了一个传说，**集齐小B凑齐的七颗石头，便可以召唤神龙**，于是就回到了文章开头我们提到的那个故事。

没错，小A对应的就是**Unload(true)**，而小B对应的则是**Unload(false)**。

**补充三点**

1、移动Unity资源时，要在Unity编辑器内拖动，不要在操作系统下剪切粘贴。因为这样Unity会为这个文件生成一个新的File GUID及.meta文件，它会打破之前建立好的关系，让所有引用过这个文件的prefab出现miss的情况。

2、实际上在项目build完成后，就已经不存在File GUID和Local ID的概念了，转而用相对简单方式建立映射，这也是为什么我们在项目运行的过程中无法获取到File GUID的原因，不过原理上它们是一样的。

3、尽管一个AssetBundle的Header部分非常小，通常只有**几十KB**，但是Unity并不能保证读入大量AssetBundle的Header部分后资源的加载效率。因此还是按需读取AssetBundle吧。